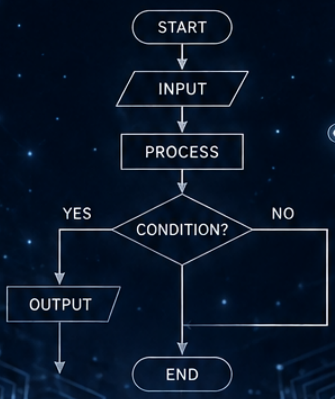
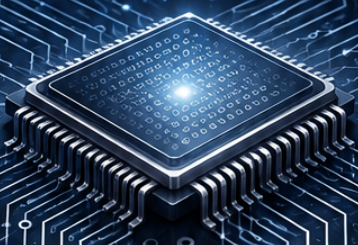
A-LEVEL COMPUTER SCIENCE

100 AI PROMPTS

for Smarter Revision *and* Exam Prep

Active recall, exam technique, and mark-scheme thinking — without cheating.





```
01 INPUT n
02 SET total ← 0
03 FOR i ← 1 TO n
04     SET total ← total + i
05 NEXT i
06 OUTPUT total
```

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

by James R. Martin

This book is an independent educational resource and is not affiliated with, endorsed by, or approved by any examination board or awarding organisation.

The author has made use of artificial intelligence tools to assist with drafting, structuring, and generating example material. All educational guidance, explanations, and exam-related advice have been reviewed, edited, and curated by the author. Any resemblance to specific published materials is unintentional.

This book is intended to support revision and exam preparation. It does not replace formal teaching, textbooks, or official specifications. Students are responsible for ensuring that all work submitted for assessment is their own.

ISBN: [TO BE ASSIGNED]

First published 2026

How to Use This Book

For a long time, high-quality tutoring has been a major contributor to elite academic achievement. Used well, AI can now act as a powerful tutor that most students and parents could not previously afford.

This book is a **starting point**, not a rulebook. Each prompt is designed to help you revise, test your understanding, and think more clearly — not to give perfect answers. You are encouraged to **adapt, improve, and remix** these prompts.

You are learning how to think carefully about the questions you ask — a skill that will matter far beyond these exams.

Note on Exam Boards and Syllabi

A-Level Computer Science is offered by AQA, Edexcel (Pearson), and OCR, and each board structures its specification with some differences in emphasis and terminology. AQA's 7517 specification covers fundamentals of programming, data structures, algorithms, theory of computation, data representation, computer systems, communications, databases, big data, and functional programming. Edexcel's 9CP0 specification covers computational thinking, programming, data and information, computer systems, and networks. OCR's H446 specification covers computing components, algorithms and programming, data and information, and computer systems. Despite these structural differences, the core content overlaps substantially across all three boards.

All three A-Level specifications require a strong grasp of both theoretical computer science and practical programming. You need to understand data representation and number systems at a deeper level than GCSE, including floating-point representation, normalisation, and two's complement arithmetic. You must be able to analyse and compare algorithms in terms of time complexity using Big O notation, implement and manipulate complex data structures such as trees, graphs, stacks, and queues, and understand the theoretical foundations of computation including finite state machines, regular expressions, and Turing machines.

Programming at A-Level is significantly more demanding than at GCSE. You must be comfortable with object-oriented programming concepts, recursion, file handling, and working with abstract data types. AQA specifically examines functional programming concepts, while all boards require you to write, trace, and debug code in at least one high-

level programming language. The prompts in this book use Python-style syntax and concepts that transfer across all boards, but you should adapt them to whichever language your school teaches.

Each board includes a substantial practical programming project (the Non-Exam Assessment or NEA) alongside written examination papers. While these prompts focus on the examined content, the programming and computational thinking skills they develop will also strengthen your project work. The prompts are designed to test understanding through active problem-solving rather than passive review, building the analytical skills that all three boards reward in their written papers.

Always check your own specification and past papers for the precise content boundaries, programming language requirements, and mark scheme expectations of your board. Use these prompts to build deep understanding and fluent problem-solving across the core topics, then tailor your final revision to the specific format and emphasis of your examination papers.

Contents

Section 1

Data Representation and Number Systems

Data representation at A-Level goes significantly beyond the GCSE foundations. While you still need fluency in binary, denary, and hexadecimal conversions, you must also master two's complement representation for negative numbers, fixed-point and floating-point binary representation, normalisation of floating-point numbers, and the trade-offs between range and precision. These topics underpin much of computer science and appear regularly in examination papers across all boards.

You must also understand how different types of data — text, images, sound, and video — are represented digitally, including the encoding schemes used and the factors that affect file size and quality. Compression algorithms, including both lossy and lossless techniques such as run-length encoding, dictionary-based compression, and Huffman coding, are examined at A-Level and require you to apply these techniques to specific data sets rather than simply describing how they work.

These prompts will test your ability to perform conversions, calculations, and analyses at A-Level depth. They require you to work with numbers precisely, understand the limitations of different representation schemes, and apply compression techniques to real examples. Practise them with pen and paper alongside the AI to build the manual calculation fluency that the written exam demands.

Prompt 1: Two's Complement Representation

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Teach me two's complement representation for signed integers using 8 bits. Explain how to convert positive and negative denary numbers to two's complement binary, and how to convert two's complement binary back to denary. Then give me 8 conversion tasks one at a time — a mix of denary to two's complement and two's complement to denary, including negative numbers. After each answer, show the correct working and explain any errors. Finish with a question about the range of values representable in 8-bit two's complement.

What this helps you practise:

Converting between denary and two's complement binary representation for signed integers.

How to use it well:

Write out each conversion step by step on paper before entering your answer. The most common error is forgetting to add 1 after flipping the bits — practise until this becomes automatic.

Prompt 2: Two's Complement Arithmetic

Copy this prompt into your AI tool:

Act as my A-Level tutor. Test me on binary arithmetic using two's complement. Give me 6 subtraction problems using 8-bit two's complement, presented one at a time. For each problem, ask me to perform the subtraction by adding the two's complement of the subtrahend. Check my working column by column and explain any errors, particularly around carry bits and overflow detection. After all 6 problems, ask me to explain what overflow means in two's complement arithmetic and how it can be detected.

What this helps you practise:

Performing binary subtraction using two's complement addition and detecting overflow.

How to use it well:

Line up your bits carefully in columns and track carry bits at every stage. Overflow occurs when the carry into the most significant bit differs from the carry out — learn to spot this.

Prompt 3: Fixed-Point Binary Representation

Copy this prompt into your AI tool:

Act as my tutor. Explain fixed-point binary representation, including how the binary point position determines the range and precision of values that can be represented. Show me how to convert fractional denary numbers to fixed-point binary and vice versa. Then give me 6 conversion tasks one at a time, using a variety of binary point positions. After each answer, show the correct working. Then ask me to explain the trade-off between range and precision when changing the position of the binary point. Give detailed feedback.

What this helps you practise:

Converting between denary and fixed-point binary and understanding the range-precision trade-off.

How to use it well:

Memorise the place values on both sides of the binary point (1, 0.5, 0.25, 0.125, etc. on the fractional side). This makes conversions much faster.

Prompt 4: Floating-Point Binary Representation

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Explain floating-point binary representation, including the concepts of mantissa and exponent, how they work together to represent a wide range of values, and how this relates to scientific notation in denary. Then give me 5 tasks one at a time: convert floating-point binary to denary, convert denary to floating-point binary, and identify the trade-offs between

allocating more bits to the mantissa versus the exponent. After each task, check my working and explain any errors. Finish by asking me why floating-point representation can lead to rounding errors.

What this helps you practise:

Converting floating-point binary numbers and understanding the mantissa-exponent trade-off.

How to use it well:

Think of the mantissa as controlling precision and the exponent as controlling range. Increasing mantissa bits gives more precision; increasing exponent bits gives a wider range of representable values.

Prompt 5: Normalisation of Floating-Point Numbers

Copy this prompt into your AI tool:

Test me on the normalisation of floating-point binary numbers. Explain why normalisation is necessary (to maximise precision), describe the rules for normalised form in both positive and negative numbers, and show me worked examples of normalising both positive and negative floating-point numbers. Then give me 6 floating-point numbers to normalise, one at a time, including both positive and negative values. After each answer, check my work and explain any errors. Then ask me to explain what happens to precision if a number is stored in unnormalised form.

What this helps you practise:

Normalising positive and negative floating-point binary numbers and understanding the precision implications.

How to use it well:

Remember the rules: a normalised positive number starts with 0.1 in the mantissa, and a normalised negative number starts with 1.0 in the mantissa

(using two's complement). Practise until these patterns become automatic.

Prompt 6: Huffman Coding

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Explain Huffman coding as a form of lossless compression, including how frequency analysis is used to build a Huffman tree, how variable-length codes are assigned, and why this achieves compression. Then give me a short string of text and ask me to: calculate the frequency of each character, build a Huffman tree, assign codes, encode the string, and calculate the compression ratio compared to a fixed-length encoding. Walk me through each step and check my work at every stage. Give detailed feedback.

What this helps you practise:

Constructing Huffman trees, assigning codes, encoding data, and calculating compression ratios.

How to use it well:

Practise building Huffman trees on paper — you may need to do this in the exam. Start with the two lowest-frequency characters and combine upwards. Double-check your tree by verifying that no code is a prefix of another.

Prompt 7: Dictionary-Based Compression (LZW)

Copy this prompt into your AI tool:

Test me on dictionary-based compression, specifically the LZW algorithm. Ask me to describe the compression process step by step for a given input string, explaining how the dictionary is built and entries are added. Then ask me to trace through a decompression example. Wait for my answer each time and check my steps are accurate.

What this helps you practise:
Understanding and applying dictionary-based compression algorithms and comparing them with statistical methods.
How to use it well:
Focus on understanding the principle of building a dictionary during compression. You may not need to implement the full algorithm in an exam, but you must be able to explain how and why it works.

Prompt 8: Error Detection and Correction
Copy this prompt into your AI tool:
Test me on error detection and correction techniques used in data transmission. Cover parity bits (even and odd parity), majority voting, checksums, and check digits. For each technique, ask me to explain how it works, demonstrate it with an example, and identify its limitations. Then give me 5 practical problems one at a time: adding parity bits to data, detecting errors in received data, and explaining scenarios where a technique would fail. Give detailed feedback on each answer.
What this helps you practise:
Applying error detection and correction techniques including parity, checksums, and majority voting.
How to use it well:
Understand both the strengths and limitations of each technique. A single parity bit can detect odd numbers of bit errors but not even numbers — knowing these limitations is frequently tested.

Prompt 9: Bitwise Operations and Masking
Copy this prompt into your AI tool:
Act as my tutor. Explain the bitwise operations AND, OR, NOT, and XOR, and describe how each can be used in practical programming and hardware contexts. Explain the concept of bit masking and

how it is used to isolate, set, clear, or toggle specific bits within a binary value. Then give me 6 practical tasks one at a time involving bitwise operations and masking, including extracting specific bits from a byte and setting particular bits. Check my answers and explain any errors. Finish by asking me to give a real-world use case for bit masking.

What this helps you practise:

Performing bitwise operations and applying bit masking techniques to manipulate binary data.

How to use it well:

Draw out the binary representations and apply the operations column by column. Understanding bitwise operations is essential for low-level programming, networking (subnet masks), and hardware interaction.

Prompt 10: Floating-Point Precision and Errors

Copy this prompt into your AI tool:

Act as my tutor. Quiz me on the limitations and potential errors of floating-point representation. Ask me to explain: why certain denary fractions cannot be represented exactly in binary, what rounding errors are and how they accumulate, what overflow and underflow mean in the context of floating-point arithmetic, and why comparing floating-point numbers for equality can be problematic in programming. For each concept, ask me to provide a specific example. Give detailed feedback on the accuracy and depth of my explanations.

What this helps you practise:

Understanding the limitations of floating-point representation including rounding, overflow, and underflow errors.

How to use it well:

These concepts often appear in exam questions that ask you to explain errors or limitations. Having

concrete examples ready — such as the inability to represent 0.1 exactly in binary — will strengthen your answers.

Prompt 11: Number Systems and Representation Exam Challenge

Copy this prompt into your AI tool:

Give me a comprehensive mini-exam on data representation covering all A-Level topics. Include: a two's complement conversion, a floating-point normalisation problem, a Huffman coding task, a question on the trade-off between mantissa and exponent sizes, a question about rounding errors in floating-point arithmetic, and a comparison of lossy and lossless compression with specific examples. Present each question one at a time, mark my answer with full working shown, and give a model answer. At the end, give me a score and identify my weakest topic with specific revision advice.

What this helps you practise:

Applying data representation knowledge across all A-Level sub-topics under exam conditions.

How to use it well:

Treat this as a genuine exam simulation. Use pen and paper for all calculations before entering your answers. Time yourself and aim to complete the full set within 30 minutes.

Section 2

Computer Architecture and Hardware

Computer architecture at A-Level builds significantly on GCSE foundations. You need to understand not just the Von Neumann architecture but also the Harvard architecture and be able to compare their advantages and disadvantages. You must have detailed knowledge of the CPU including the roles of all key registers, the fetch-decode-execute cycle at a granular level, pipelining, and the factors affecting processor performance including clock speed, cores, cache levels, and word length.

Beyond the CPU, you must understand the role of different types of memory (SRAM, DRAM, flash memory), input/output systems, bus architectures, and secondary storage technologies. You should also be familiar with the concepts of virtual memory, paging, and how the operating system manages hardware resources. The relationship between hardware and software — how high-level code is translated into machine instructions — connects this topic to programming and systems software.

These prompts will test your understanding of computer architecture at A-Level depth, requiring you to explain concepts precisely, compare different architectures and technologies, and apply your knowledge to analyse system performance. Use them to build the detailed technical understanding that examination papers demand.

Prompt 12: Von Neumann vs Harvard Architecture

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Explain the Von Neumann architecture and the Harvard

architecture, highlighting the key differences in how they handle instructions and data. Ask me to explain three advantages and two disadvantages of each architecture, then present me with four scenarios (such as an embedded system in a microwave, a general-purpose desktop PC, a digital signal processor, and a smartphone) and ask me to justify which architecture would be most suitable for each. Give feedback on my justifications.

What this helps you practise:

Comparing Von Neumann and Harvard architectures and selecting the appropriate architecture for given contexts.

How to use it well:

Focus on the key difference: Von Neumann uses a single memory and bus for both data and instructions, while Harvard separates them. This distinction drives all the advantages and disadvantages.

Prompt 13: CPU Registers Deep Dive

Copy this prompt into your AI tool:

Test me on the key CPU registers used in the fetch-decode-execute cycle. Present me with the following registers one at a time: Program Counter (PC), Memory Address Register (MAR), Memory Data Register (MDR), Current Instruction Register (CIR), Accumulator (ACC), and Status Register. For each register, ask me to explain its purpose, what data it holds, and how it interacts with other registers during instruction execution. Then describe three steps of the fetch-decode-execute cycle and ask me to trace the contents of each register. Give detailed feedback.

What this helps you practise:

Explaining the function of each CPU register and

tracing register contents through the fetch-decode-execute cycle.

How to use it well:

Create a table showing each register and its contents at each step of the cycle. Being able to trace instruction execution through registers is a common exam question format.

Prompt 14: Pipelining and Processor Performance

Copy this prompt into your AI tool:

Quiz me on instruction pipelining in a processor. Ask me to name and describe each stage of a typical five-stage pipeline, explain how pipelining increases throughput, and describe the problems that can occur — specifically data hazards, control hazards, and structural hazards. Ask me how each hazard can be mitigated. Wait for my answer before providing feedback.

What this helps you practise:

Understanding instruction pipelining, pipeline hazards, and their impact on processor performance.

How to use it well:

Draw a timeline diagram showing how three instructions overlap in a pipeline. This visual representation makes it much clearer how pipelining improves throughput.

Prompt 15: Cache Memory and the Memory Hierarchy

Copy this prompt into your AI tool:

Act as my A-Level tutor. Explain the memory hierarchy from registers through L1, L2, and L3 cache to RAM and secondary storage. For each level, ask me to state the relative speed, capacity, cost, and volatility. Then quiz me on: how cache memory works (locality of reference, cache hits and

misses), why multiple levels of cache exist, the impact of cache size on performance, and write-through versus write-back cache policies. Present 5 questions one at a time and give detailed feedback on each answer.

What this helps you practise:

Understanding the memory hierarchy, cache operation principles, and their impact on system performance.

How to use it well:

The key concept is the trade-off between speed, capacity, and cost at each level of the hierarchy. Make sure you can explain why we need multiple levels rather than simply making all memory as fast as L1 cache.

Prompt 16: Virtual Memory and Paging

Copy this prompt into your AI tool:

Test me on virtual memory and paging. Explain how virtual memory allows programs to use more memory than physically available, how pages are mapped between virtual and physical addresses using a page table, and what happens during a page fault. Then ask me to explain: the role of the memory management unit (MMU), the difference between paging and segmentation, why disk thrashing occurs, and how page replacement algorithms work. Present 5 questions one at a time and give detailed feedback.

What this helps you practise:

Understanding virtual memory systems including paging, page tables, page faults, and thrashing.

How to use it well:

Think of virtual memory as an abstraction layer that decouples the program's view of memory from physical RAM. Understanding this abstraction is key to answering questions about memory management.

Prompt 17: Assembly Language and Machine Code

Copy this prompt into your AI tool:

Act as my tutor. Explain the relationship between high-level languages, assembly language, and machine code. Describe the key assembly language concepts: mnemonics, opcodes, operands, addressing modes (immediate, direct, indirect, indexed). Then give me 5 short assembly language programs (using a simple instruction set) one at a time and ask me to trace through each one, stating the contents of registers and memory locations after execution. Check my traces and explain any errors.

What this helps you practise:

Tracing assembly language programs and understanding addressing modes and instruction execution.

How to use it well:

Create a table of register and memory contents and update it line by line as you trace each instruction. This systematic approach prevents errors and mirrors what examiners expect to see as working.

Prompt 18: Operating Systems — Process Scheduling and Memory Management

Copy this prompt into your AI tool:

Test me on operating systems at A-Level depth, with particular focus on process scheduling and memory management. Present the following areas one at a time: process management and scheduling algorithms (round-robin, priority, shortest job first, multi-level feedback queues), memory management techniques (paging, segmentation, virtual memory, page replacement algorithms), file system management (directories, access control, fragmentation), device management (drivers,

buffers, interrupts), and user interface provision. For each area, ask me to explain how the OS implements the function, compare different approaches and their trade-offs, and discuss real-world implications. Then give me 3 scenario-based questions. Give detailed feedback.

What this helps you practise:

Explaining operating system functions in detail including process scheduling, memory management, and device management.

How to use it well:

Think of the OS as a resource manager and an abstraction layer. Every function it performs is either managing a shared resource or providing a simpler interface to complex hardware.

Prompt 19: Interrupt Handling

Copy this prompt into your AI tool:

Test me on the interrupt mechanism in a computer system. Ask me to describe the steps the processor takes when an interrupt occurs, from detecting the interrupt signal through saving the current state to executing the interrupt service routine and returning. Then ask me to explain interrupt priorities and what happens when a higher-priority interrupt arrives during the handling of a lower-priority one. Check my answer for accuracy.

What this helps you practise:

Understanding the interrupt mechanism, interrupt service routines, and interrupt priority management.

How to use it well:

Trace through an interrupt scenario step by step: the interrupt signal, saving the current state to the stack, executing the ISR, and restoring the state. This step-by-step approach is what examiners expect.

Prompt 20: CISC vs RISC Architectures

Copy this prompt into your AI tool:

Act as my tutor. Explain the differences between CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) architectures. Cover instruction set design, clock cycles per instruction, code density, pipelining compatibility, and typical use cases. Then present me with 5 comparison questions one at a time, including scenarios asking which architecture would be more suitable for a given application and questions about how RISC and CISC have converged in modern processors. Give detailed feedback on each answer.

What this helps you practise:

Comparing CISC and RISC processor architectures and understanding their design trade-offs.

How to use it well:

In modern computing, the distinction between RISC and CISC has blurred — many processors use elements of both. Make sure your answers acknowledge this rather than presenting an oversimplified dichotomy.

Prompt 21: Compilers, Interpreters, and Assemblers

Copy this prompt into your AI tool:

Quiz me on language translators at A-Level depth. Ask me to explain the function and process of compilers, interpreters, and assemblers. For compilers, ask me to explain the stages of compilation: lexical analysis, syntax analysis, semantic analysis, code generation, and optimisation. Then present 5 comparison questions one at a time: scenarios where a compiler would be preferred over an interpreter and vice versa, the role of intermediate code, just-in-time (JIT)

compilation, and cross-compilation. Give detailed feedback on each answer.

What this helps you practise:

Understanding the compilation process, comparing language translators, and explaining compilation stages.

How to use it well:

Learn the stages of compilation in order and be able to explain what each stage does and what errors it can detect. This is a frequently examined topic that rewards precise technical knowledge.

Prompt 22: Hardware and Architecture Exam Challenge

Copy this prompt into your AI tool:

Give me a comprehensive mini-exam on computer architecture and hardware. Include questions on: tracing the fetch-decode-execute cycle through registers, comparing Von Neumann and Harvard architectures, explaining how pipelining improves performance, describing the memory hierarchy, explaining virtual memory and page faults, and comparing two scheduling algorithms. Present each question one at a time, mark my answer, show the model answer, and give feedback. At the end, give me a score and identify my weakest area.

What this helps you practise:

Applying computer architecture knowledge across all sub-topics under exam conditions.

How to use it well:

Treat this as a timed exam simulation. Write your answers on paper first, then type them in. This builds the habits you need for the written exam.

Section 3

Networks and Communication

Networking at A-Level extends well beyond the GCSE coverage of LANs, WANs, and basic protocols. You need to understand the TCP/IP protocol stack in detail, including how each layer operates and the role of specific protocols at each layer. You must be able to explain packet switching and circuit switching, understand how data is routed across networks, and analyse the security challenges of networked communication.

The physical and logical aspects of networking are both examined. You should understand different network topologies and their trade-offs, how IP addressing works (including subnetting and the difference between IPv4 and IPv6), how DNS translates domain names to IP addresses, and how client-server and peer-to-peer architectures differ. You must also understand wireless networking, including how Wi-Fi operates and the security protocols used to protect wireless communications.

These prompts will test your networking knowledge at A-Level depth, requiring you to explain protocols precisely, analyse network designs, and solve practical networking problems. Use them to build the technical fluency that examiners expect when you describe how data travels from source to destination across complex networks.

Prompt 23: The TCP/IP Protocol Stack

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Quiz me on the TCP/IP protocol stack. Present each of the four layers one at a time (Application, Transport, Internet, Link) and ask me to: name the layer,

explain its function, identify at least two protocols that operate at that layer, and describe how data is encapsulated as it moves down the stack. After all four layers, ask me to trace the journey of an HTTP request from a browser to a web server, identifying what happens at each layer. Give detailed feedback on the precision of my explanations.

What this helps you practise:

Explaining the TCP/IP protocol stack layers, associated protocols, and data encapsulation.

How to use it well:

Learn the layers in order and associate specific protocols with each layer. Being able to trace data through all four layers from application to link is a key exam skill.

Prompt 24: DNS and Domain Name Resolution

Copy this prompt into your AI tool:

Quiz me on the Domain Name System (DNS). Ask me to explain: what DNS does, the hierarchical structure of the DNS system (root servers, TLD servers, authoritative servers), the process of recursive and iterative DNS resolution, DNS caching and TTL values, and the security risks associated with DNS (such as DNS spoofing and cache poisoning). Present 5 questions one at a time. After each answer, give detailed feedback on the accuracy and depth of my explanation.

What this helps you practise:

Understanding DNS architecture, the resolution process, caching, and DNS-related security risks.

How to use it well:

Trace through a complete DNS resolution from a browser typing a URL to receiving the IP address. Understanding each step of this process is frequently tested.

Prompt 25: Networking Exam Blitz

Copy this prompt into your AI tool:

Give me a rapid-fire networking exam with 15 questions covering all A-Level networking topics. Mix question types: protocol identification, subnetting calculations, comparison questions (TCP vs UDP, IPv4 vs IPv6), scenario-based topology recommendations, security protocol explanations, and tracing data through the TCP/IP stack. Present each question one at a time, mark immediately, and keep a running score. At the end, identify my three weakest areas and give specific revision targets.

What this helps you practise:

Rapid recall and application of networking knowledge across all sub-topics.

How to use it well:

Time yourself — aim for no more than 90 seconds per question. Speed of recall combined with technical precision is essential for scoring well in networking questions.

Prompt 26: TCP vs UDP

Copy this prompt into your AI tool:

Test me on the differences between TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Ask me to explain how TCP establishes a connection (three-way handshake), how it ensures reliable delivery (acknowledgements, retransmission, sequencing), and why this adds overhead. Then ask me to explain how UDP works without these guarantees and why it is faster. Present me with 5 application scenarios one at a time (such as file download, video streaming, online gaming, email, VoIP) and ask me to choose TCP or UDP and justify my choice. Give feedback.

What this helps you practise:

Comparing TCP and UDP and selecting the

appropriate protocol for different application scenarios.

How to use it well:

The key trade-off is reliability versus speed. TCP guarantees delivery at the cost of overhead; UDP sacrifices reliability for lower latency. Match each scenario to the most important requirement.

Prompt 27: IP Addressing and Subnetting

Copy this prompt into your AI tool:

Act as my tutor. Explain IPv4 addressing including dotted decimal notation, the structure of an IP address (network and host portions), subnet masks, and how subnetting divides a network. Then give me 5 subnetting problems one at a time: given an IP address and subnet mask, ask me to determine the network address, broadcast address, number of usable hosts, and the range of valid host addresses. Check my working at each step. Then ask me to explain three key differences between IPv4 and IPv6. Give detailed feedback.

What this helps you practise:

Performing IPv4 subnetting calculations and comparing IPv4 with IPv6 addressing.

How to use it well:

Convert IP addresses and subnet masks to binary to perform subnetting. The binary AND operation between the IP address and subnet mask gives you the network address — practise this until it is second nature.

Prompt 28: Packet Switching and Routing

Copy this prompt into your AI tool:

Test me on packet switching at A-Level depth. Ask me to explain: how data is divided into packets, the structure of a packet (headers, payload, trailer), how packets are routed across a network using routing

tables, how packets may take different paths and be reassembled at the destination, and the advantages and disadvantages of packet switching versus circuit switching. Then present me with a network diagram (described in text) and ask me to trace the route a packet might take. Give detailed feedback.

What this helps you practise:

Understanding packet switching, packet structure, routing, and comparison with circuit switching.

How to use it well:

Focus on the packet header fields and their purposes — source and destination IP, sequence number, TTL, checksum. Understanding why each field exists demonstrates deep comprehension.

Prompt 29: Network Security Protocols

Copy this prompt into your AI tool:

Act as my tutor. Quiz me on network security protocols and technologies. Cover the following one at a time: firewalls (packet filtering, stateful inspection, application-level gateways), encryption protocols (SSL/TLS, HTTPS), VPNs (tunnelling, encryption), WPA2 and WPA3 for wireless security, and digital certificates and certificate authorities. For each technology, ask me to explain how it works, what threats it protects against, and its limitations. Present 5 questions one at a time and give detailed feedback.

What this helps you practise:

Explaining network security protocols, their mechanisms, and their limitations.

How to use it well:

For each security technology, know what it protects against and what it does not protect against. No single security measure is sufficient on its own — defence in depth is the key principle.

Prompt 30: Client-Server vs Peer-to-Peer

Copy this prompt into your AI tool:

Compare client-server and peer-to-peer network architectures at A-Level depth. Ask me to explain the structure and operation of each, identify their advantages and disadvantages in terms of security, scalability, management, and performance, and present me with 4 scenarios where I must choose the more appropriate architecture and justify my choice. Include scenarios involving file sharing, web services, gaming, and distributed computing. Give detailed feedback on each justification.

What this helps you practise:

Comparing client-server and peer-to-peer architectures and selecting the appropriate model for given scenarios.

How to use it well:

Think about the practical implications of each architecture: who controls the data, how does the network scale, what happens if one node fails? These practical considerations drive the choice between models.

Prompt 31: The Internet and Web Technologies

Copy this prompt into your AI tool:

Quiz me on internet and web technologies at A-Level depth. Cover: the difference between the internet and the World Wide Web, HTTP and HTTPS request/response cycle, HTML and CSS as markup and styling languages, the role of web servers and web browsers, APIs and RESTful services, and the concept of thin versus thick clients. Present 6 questions one at a time including at least one that requires me to explain what happens when a user types a URL into a browser and presses Enter. Give detailed feedback.

What this helps you practise:

Understanding internet architecture, web protocols, and the HTTP request/response cycle.

How to use it well:

The question 'what happens when you type a URL into a browser' is a classic that tests your understanding of DNS, TCP, HTTP, and rendering all together. Practise answering it with maximum technical precision.

Prompt 32: Network Topologies and Their Trade-offs

Copy this prompt into your AI tool:

Test me on network topologies at A-Level depth. Present the following topologies one at a time: bus, star, ring, mesh, and hybrid. For each, ask me to draw it in text form, explain how data is transmitted, identify the single point of failure (if any), and assess the advantages and disadvantages in terms of performance, reliability, cost, and scalability. Then give me 3 scenario-based questions asking me to recommend a topology for a given situation. Give detailed feedback.

What this helps you practise:

Analysing network topologies and recommending appropriate topologies for specific requirements.

How to use it well:

When comparing topologies, use a consistent set of criteria: cost, reliability, performance, scalability, and ease of maintenance. This structured approach will help you write clear comparison answers in the exam.

Prompt 33: Network Design and Analysis

Copy this prompt into your AI tool:

Present me with a scenario describing an organisation's networking requirements (such as a

school with multiple buildings, a small business expanding to a second site, or a hospital requiring secure communication). Ask me to design a network solution specifying: topology, hardware required, protocols to be used, security measures, and IP addressing scheme. Then critique my design, identifying any weaknesses, missing components, or better alternatives. Challenge me to justify my choices with technical reasoning. Give detailed feedback.

What this helps you practise:

Designing and justifying network solutions for real-world scenarios using appropriate technologies and protocols.

How to use it well:

Network design questions test your ability to apply theoretical knowledge to practical situations. Always justify your choices with specific technical reasons rather than simply listing components.

Section 4

Algorithms and Computational Thinking

Algorithms are at the heart of A-Level Computer Science. You must be able to design, implement, trace, and analyse algorithms at a significantly higher level than GCSE. This means not only knowing how standard algorithms work (searching, sorting, graph traversal) but also being able to analyse their efficiency using Big O notation, compare algorithms based on time and space complexity, and understand the theoretical limits of algorithmic problem-solving.

Computational thinking — the ability to decompose problems, recognise patterns, abstract away unnecessary detail, and design algorithmic solutions — is tested both explicitly and implicitly across every paper. You need to be comfortable with recursion as a problem-solving technique, understand divide-and-conquer strategies, and be able to apply algorithmic thinking to unfamiliar problems. The ability to trace through algorithms step by step, including recursive calls, is a core exam skill.

These prompts will challenge you to go beyond memorising algorithms to genuinely understanding why they work, when they are appropriate, and how their performance scales. They cover sorting, searching, graph algorithms, recursion, complexity analysis, and algorithmic design strategies. Use them to build the deep understanding that allows you to tackle unfamiliar algorithm questions in the exam with confidence.

Prompt 34: Sorting Algorithm Comparison

Copy this prompt into your AI tool:

Test me on the A-Level sorting algorithms: bubble sort, insertion sort, merge sort, and quicksort. For each algorithm, ask me to: explain how it works step by step, state its best-case, worst-case, and average-case time complexity, identify whether it is stable, in-place, or recursive, and describe a scenario where it would be the best choice. Then give me a small unsorted list and ask me to trace through two different sorting algorithms on the same data, showing each step. Give detailed feedback on my traces and explanations.

What this helps you practise:

Explaining, tracing, and comparing sorting algorithms including their time complexity and characteristics.

How to use it well:

Create a comparison table for all four sorting algorithms covering: time complexity (best/worst/average), space complexity, stability, and best use case. This reference will help you answer comparison questions quickly.

Prompt 35: Binary Search and Search Algorithms

Copy this prompt into your AI tool:

Quiz me on searching algorithms at A-Level depth. Ask me to explain and compare linear search and binary search, including their time complexities. Then test me on binary search specifically: give me a sorted list of 15 items and a target value, and ask me to trace through the binary search process showing the low, mid, and high pointers at each step. Repeat with a target that is not in the list. Then ask me to explain why binary search requires sorted data and to calculate the maximum number of comparisons for a list of n items. Give detailed feedback.

What this helps you practise:

Tracing binary search step by step and analysing its time complexity compared to linear search.

How to use it well:

Practise tracing binary search on paper, being precise about how you calculate the midpoint at each step. Remember that binary search is O(log n) — explain what this means in practical terms for large datasets.

Prompt 36: Recursion: Understanding and Tracing

Copy this prompt into your AI tool:

Act as my A-Level tutor. Explain recursion including the concepts of base case and recursive case, how the call stack works during recursive calls, and the relationship between recursion and iteration. Then give me 4 recursive functions one at a time (such as factorial, Fibonacci, sum of a list, and a recursive binary search). For each, ask me to: identify the base case and recursive case, trace the execution for a specific input showing all recursive calls and the call stack, and state the return value. Check my traces and correct any errors.

What this helps you practise:

Tracing recursive function execution including call stack management, base cases, and return values.

How to use it well:

Draw the call stack on paper as you trace each recursive call, showing the parameters and return values at each level. This visual approach prevents the confusion that recursive tracing often causes.

Prompt 37: Graph Traversal: BFS and DFS

Copy this prompt into your AI tool:

Test me on graph traversal algorithms. Explain breadth-first search (BFS) and depth-first search

(DFS) including the data structures used (queue for BFS, stack for DFS). Then present me with a graph (described as an adjacency list or adjacency matrix) and ask me to trace both BFS and DFS from a specified starting node, showing the order in which nodes are visited and the state of the queue/stack at each step. Compare my traversal orders with the correct answers and explain any errors. Then ask me to identify practical applications of each algorithm.

What this helps you practise:

Tracing BFS and DFS on graphs, managing the associated data structures, and comparing the two approaches.

How to use it well:

Use the adjacency list representation and maintain a clear record of visited nodes, the queue (BFS) or stack (DFS), and the order of exploration. Practise on multiple different graph structures.

Prompt 38: Big O Notation and Time Complexity

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Explain Big O notation and the most common time complexities: O(1), O(log n), O(n), O(n log n), O(n^2), and O(2^n). For each, give an example algorithm or operation that has that complexity. Then present me with 6 code snippets or algorithm descriptions one at a time and ask me to identify the Big O time complexity of each. After each answer, explain the reasoning and correct any errors. Finish by asking me to rank all six complexities from fastest to slowest growth.

What this helps you practise:

Identifying and comparing Big O time complexities of algorithms and code fragments.

How to use it well:

Focus on the dominant term and ignore constants and lower-order terms. Look for nested loops (often O(n^2)), single loops (often O(n)), and divide-and-conquer patterns (often O(n log n)).

Prompt 39: Dijkstra's Shortest Path Algorithm

Copy this prompt into your AI tool:

Act as my tutor. Explain Dijkstra's shortest path algorithm step by step, including how it maintains a set of visited nodes, a table of distances, and selects the next node to process. Then present me with a weighted graph (described in text) and ask me to apply Dijkstra's algorithm from a starting node, showing the distance table and visited set at each step. Check my work at every step and correct any errors. Then ask me about the algorithm's time complexity and a scenario where it would not work (negative edge weights).

What this helps you practise:

Applying Dijkstra's algorithm to weighted graphs and understanding its requirements and limitations.

How to use it well:

Create a table with columns for each node showing the current shortest distance and whether the node has been visited. Update this table systematically at each step of the algorithm.

Prompt 40: Divide and Conquer Strategy

Copy this prompt into your AI tool:

Quiz me on the divide-and-conquer algorithmic strategy. Ask me to define the three stages (divide, conquer, combine), then give me a specific problem and ask me to trace through how a divide-and-conquer algorithm would solve it step by step. Ask me to compare the time complexity of the divide-

and-conquer approach with a brute-force alternative. Wait for my answers before correcting errors.

What this helps you practise:

Understanding the divide-and-conquer strategy and applying it to design algorithmic solutions.

How to use it well:

The key insight is that dividing a problem in half at each step leads to O(log n) levels of recursion. Combined with O(n) work at each level, this gives the O(n log n) complexity characteristic of divide-and-conquer algorithms.

Prompt 41: Algorithm Design Challenges

Copy this prompt into your AI tool:

Present me with 4 algorithmic problems one at a time, each requiring me to design a solution from scratch. These should include: a problem solvable with a greedy approach, a problem requiring backtracking, a problem suited to dynamic programming thinking, and an optimisation problem. For each, ask me to describe my approach, explain why I chose that strategy, write pseudocode for my solution, and analyse its time complexity. Give detailed feedback on the correctness, efficiency, and clarity of my solutions.

What this helps you practise:

Designing algorithmic solutions to unfamiliar problems using appropriate strategies.

How to use it well:

When faced with an unfamiliar problem, start by identifying which algorithmic strategy might apply: brute force, greedy, divide and conquer, or dynamic programming. Then design your solution around that strategy.

Prompt 42: Algorithm Tracing Under Pressure

Copy this prompt into your AI tool:

Run a rapid algorithm tracing test. Give me 6 short algorithms or code snippets one at a time and ask me to trace through each with a given input, stating the final output. These should include: a nested loop, a recursive function, a sorting algorithm on a small list, a search algorithm, a string manipulation, and a function using a stack or queue. Time pressure is key — challenge me to trace each one quickly and accurately. Give immediate feedback and mark each answer correct or incorrect. Give a final score out of 6.

What this helps you practise:

Rapid and accurate tracing of algorithms and code under time pressure.

How to use it well:

Algorithm tracing is a core exam skill. Practise tracing systematically — update variables at each step, track loop counters, and maintain the call stack for recursive functions.

Prompt 43: Space Complexity and Trade-offs

Copy this prompt into your AI tool:

Act as my tutor. Explain the concept of space complexity and how it differs from time complexity. Give examples of algorithms that trade space for time (such as hash tables, memoisation) and algorithms that trade time for space (such as in-place sorting). Then present me with 4 pairs of algorithmic approaches to the same problem, where one uses more memory but runs faster, and ask me to analyse the space-time trade-off in each case and recommend which approach is more appropriate for a given set of constraints. Give detailed feedback.

What this helps you practise:

Analysing space complexity and evaluating space-time trade-offs in algorithm design.

How to use it well:

Space complexity is often overlooked in revision but appears in exam questions. For every algorithm you know, be prepared to state both its time and space complexity.

Prompt 44: Algorithms Knowledge Consolidation

Copy this prompt into your AI tool:

Give me a comprehensive assessment covering all algorithms topics. Include: identifying Big O complexity from code, tracing a recursive function, applying Dijkstra's algorithm to a graph, comparing two sorting algorithms for a given scenario, explaining a divide-and-conquer approach, and designing a solution to an unfamiliar problem. Present 8 questions one at a time, mark each immediately with full model answers, and give a final score. Identify my two strongest and two weakest areas with specific revision advice for the weak areas.

What this helps you practise:

Comprehensive assessment of algorithmic knowledge and problem-solving skills across all sub-topics.

How to use it well:

Use this as a diagnostic assessment before the exam. The feedback will tell you exactly which algorithmic topics need more attention and which you can be confident about.

Section 5

Programming and Data Structures

Programming at A-Level demands far more than the procedural coding skills developed at GCSE. You must understand and apply object-oriented programming (OOP) concepts including classes, objects, inheritance, polymorphism, and encapsulation. You need to be comfortable implementing abstract data types such as stacks, queues, linked lists, trees, and hash tables, understanding both their logical structure and their physical implementation using arrays or dynamic memory.

Data structures are closely linked to algorithms — choosing the right data structure for a problem can make the difference between an efficient and an inefficient solution. You must understand the operations supported by each data structure (insertion, deletion, traversal, searching), their time complexity, and the scenarios where each is most appropriate. Tree structures, including binary trees and binary search trees, are particularly important at A-Level.

These prompts will test your ability to implement, manipulate, and reason about data structures and write programs using OOP principles. They require active problem-solving rather than passive recall — you will be asked to write code, trace through operations, and choose appropriate data structures for given problems. Use them to build the programming confidence and data structure fluency that both the written papers and the NEA demand.

Prompt 45: Classes, Objects, and Constructors

Copy this prompt into your AI tool:

Test me on classes and objects in practice. Present me with a scenario and ask me to write a class definition including: private attributes, a constructor, getter and setter methods, and at least two other methods. Then ask me to write code that creates objects from this class and calls methods on them. Present three tasks one at a time, increasing in complexity. After each, check my code for correctness, proper encapsulation, and good coding practices. Give detailed feedback.

What this helps you practise:

Writing class definitions with proper encapsulation, constructors, and methods.

How to use it well:

Always make attributes private and provide getter/setter methods where needed. This encapsulation is a fundamental principle that examiners expect to see in your code.

Prompt 46: Linked Lists

Copy this prompt into your AI tool:

Act as my A-Level tutor. Explain how a linked list works using nodes and pointers, and compare it with an array-based list. Then ask me to implement: a linked list class with operations for inserting at the head, inserting at the tail, deleting a specified node, searching for a value, and traversing the list. Present each operation as a separate task. After each implementation, check my code and ask me to trace through it with specific data. Then ask me to compare the time complexity of linked list operations with array operations. Give detailed feedback.

What this helps you practise:

Implementing linked list operations and comparing linked list and array performance characteristics.

How to use it well:

When implementing linked list operations, always consider the edge cases: inserting into an empty list, deleting the head node, and deleting the only node. These edge cases are where bugs typically occur.

Prompt 47: Object-Oriented Programming Fundamentals

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Quiz me on the four pillars of object-oriented programming: encapsulation, inheritance, polymorphism, and abstraction. For each concept, ask me to: define it precisely, explain why it is useful in software design, and give a specific code example. Then present me with a scenario (such as designing a library management system) and ask me to design a class hierarchy identifying the classes, their attributes and methods, and the inheritance relationships. Give detailed feedback on my design and my understanding of OOP principles.

What this helps you practise:

Understanding and applying the four pillars of OOP and designing class hierarchies for given scenarios.

How to use it well:

When designing a class hierarchy, start by identifying the 'is-a' relationships (for inheritance) and the 'has-a' relationships (for composition). This helps you structure your classes logically.

Prompt 48: Inheritance and Polymorphism

Copy this prompt into your AI tool:

Act as my tutor. Explain inheritance (including superclass, subclass, and method overriding) and polymorphism (including method overloading and dynamic binding). Then give me a coding challenge: present a base class and ask me to create two

subclasses that inherit from it, override at least one method, and demonstrate polymorphism by writing code that treats different subclass objects through a common interface. Check my code and give feedback on whether I have correctly implemented inheritance and polymorphism.

What this helps you practise:

Implementing inheritance and polymorphism in code with method overriding and dynamic dispatch.

How to use it well:

Polymorphism means 'many forms' — the same method call produces different behaviour depending on the object's type. Show this explicitly in your code examples.

Prompt 49: Stack Implementation and Operations

Copy this prompt into your AI tool:

Test me on the stack data structure. Ask me to explain the LIFO (Last In, First Out) principle, the key operations (push, pop, peek, isEmpty, isFull), and when overflow and underflow occur. Then ask me to implement a stack using an array, including all operations and a stack pointer. Present 5 tasks one at a time: push values, pop values, trace the stack contents and pointer after a sequence of operations, and implement a function that uses a stack (such as reversing a string or checking balanced brackets). Check my implementations and traces.

What this helps you practise:

Implementing and manipulating stacks using arrays, and applying stacks to practical problems.

How to use it well:

Draw the array and the stack pointer position at each step when tracing stack operations. This visual

approach prevents errors and mirrors the systematic working examiners expect.

Prompt 50: Queue Implementations

Copy this prompt into your AI tool:

Quiz me on queues at A-Level depth. Ask me to explain FIFO (First In, First Out), the key operations (enqueue, dequeue, front, rear, isEmpty, isFull), and the difference between a linear queue and a circular queue. Then ask me to implement a circular queue using an array, handling wrap-around correctly. Present 5 tasks one at a time including tracing enqueue and dequeue operations on a circular queue, identifying when the queue is full versus empty, and implementing a priority queue. Check my implementations and traces.

What this helps you practise:

Implementing linear and circular queues, tracing operations, and understanding priority queue concepts.

How to use it well:

The circular queue is trickier than the linear queue because of wrap-around. Use the modulo operator to handle this: rear = (rear + 1) % maxSize. Practise this until it is automatic.

Prompt 51: Binary Trees and Binary Search Trees

Copy this prompt into your AI tool:

Test me on binary trees and binary search trees (BSTs). Ask me to explain the BST property, then present me with the following tasks one at a time: insert a series of values into a BST (showing the resulting tree structure), search for a value, delete a node (including all three cases: leaf, one child, two children), and perform in-order, pre-order, and post-order traversals stating the output for each. Check

my answers at each step and correct any errors. Then ask me about the worst-case performance of a BST and when it occurs.

What this helps you practise:

Building, searching, deleting from, and traversing binary search trees.

How to use it well:

Draw the tree at each step — BST questions are much easier to answer with a visual representation. For deletions, the two-children case (replacing with in-order successor) is the trickiest — practise it specifically.

Prompt 52: Hash Tables and Hashing

Copy this prompt into your AI tool:

Act as my tutor. Explain hash tables including: hash functions, how keys are mapped to indices, collision handling (open addressing with linear probing, chaining with linked lists), load factor, and rehashing. Then give me a hash table with a specific size and hash function, and present 8 insertion operations one at a time, including some that cause collisions. Ask me to show where each value is stored and how collisions are resolved. Then ask me to explain the average and worst-case time complexity for hash table operations. Give detailed feedback.

What this helps you practise:

Understanding hash table operations including hashing, collision resolution, and performance analysis.

How to use it well:

Trace through insertions step by step, showing the hash calculation and the resolution of each collision. Understanding what happens when collisions accumulate is key to understanding hash table performance.

Prompt 53: Graph Representations

Copy this prompt into your AI tool:

Test me on graph representations. Ask me to explain the two main ways of representing a graph in code: adjacency matrices and adjacency lists. For each, ask me to explain how they work, their space complexity, and the time complexity of common operations (checking if an edge exists, finding neighbours, adding an edge). Then give me a graph and ask me to represent it as both an adjacency matrix and an adjacency list. Challenge me to explain when each representation is more appropriate. Give detailed feedback.

What this helps you practise:

Representing graphs using adjacency matrices and adjacency lists and comparing their performance characteristics.

How to use it well:

A dense graph (many edges) suits an adjacency matrix; a sparse graph (few edges) suits an adjacency list. Know the space complexity of each: matrix is O(n^2), list is O(n + e) where e is the number of edges.

Prompt 54: Choosing the Right Data Structure

Copy this prompt into your AI tool:

Present me with 6 programming scenarios one at a time and ask me to recommend the most appropriate data structure for each, justifying my choice based on the operations required, time complexity, and memory usage. Scenarios should cover situations suited to: an array, a linked list, a stack, a queue, a hash table, and a binary search tree. After each recommendation, evaluate my reasoning and discuss any alternative data structures that could also work. Give detailed feedback.

What this helps you practise:

Selecting appropriate data structures for given problems based on performance requirements.

How to use it well:

Always start by identifying the key operations the problem requires (insertion, deletion, search, ordering) and then match these to the data structure that performs those operations most efficiently.

Prompt 55: Programming and Data Structures Challenge

Copy this prompt into your AI tool:

Give me a comprehensive programming and data structures challenge. Present 6 tasks one at a time covering: writing a class with proper OOP principles, implementing a stack-based algorithm, tracing operations on a BST, resolving hash table collisions, choosing between data structures for a scenario, and implementing a recursive function using a tree. For each task, check my code or trace, provide the model answer, and give feedback. At the end, give me a score and identify my strongest and weakest areas.

What this helps you practise:

Comprehensive assessment of programming and data structures skills across all sub-topics.

How to use it well:

Write your code on paper before typing it in — this simulates exam conditions where you cannot run your code. Building this skill is essential for the written programming questions.

Section 6

Databases and SQL

Databases at A-Level go well beyond the basic database concepts covered at GCSE. You need to understand the relational database model in depth, including entity-relationship modelling, the process of normalisation through First, Second, and Third Normal Forms, and the purpose and application of each normalisation stage. You must be able to design database schemas, identify primary and foreign keys, and understand the role of referential integrity.

SQL (Structured Query Language) is a core practical skill at A-Level. You must be able to write queries using SELECT, INSERT, UPDATE, and DELETE statements, apply WHERE clauses with multiple conditions, use aggregate functions (COUNT, SUM, AVG, MAX, MIN) with GROUP BY and HAVING, perform JOIN operations to combine data from multiple tables, and use subqueries. These skills are tested both in isolation and within broader problem-solving questions.

These prompts will test your ability to design databases from requirements, normalise data to remove redundancy, and write SQL queries of increasing complexity. They require active problem-solving — you will design schemas, normalise tables, and write queries rather than simply describing concepts. Use them to build the database design and SQL fluency that examination papers demand.

Prompt 56: Normalisation to Third Normal Form

Copy this prompt into your AI tool:

Test me on database normalisation. Present me with an unnormalised table containing redundant data

and ask me to normalise it step by step. First, ask me to identify the problems with the unnormalised table (redundancy, insertion anomalies, deletion anomalies, update anomalies). Then ask me to normalise to 1NF, then 2NF, then 3NF, explaining what I am doing at each stage and why. Show me the resulting tables with their primary and foreign keys. Check my work at each stage and correct any errors. Give detailed feedback.

What this helps you practise:

Normalising database tables through 1NF, 2NF, and 3NF while identifying and resolving data anomalies.

How to use it well:

Remember the rules: 1NF removes repeating groups, 2NF removes partial dependencies, 3NF removes transitive dependencies. Apply these systematically and check your results by verifying that no anomalies remain.

Prompt 57: SQL SELECT Queries

Copy this prompt into your AI tool:

Act as my tutor. Present me with a database schema consisting of at least three related tables. Then give me 8 SQL query challenges one at a time, progressing in difficulty: start with simple SELECT statements with WHERE clauses, then add ORDER BY, then add aggregate functions with GROUP BY, then add HAVING clauses. After each query I write, check whether it would return the correct results, identify any syntax errors, and suggest improvements. Give detailed feedback on each query.

What this helps you practise:

Writing SQL SELECT queries of increasing complexity including filtering, sorting, aggregation, and grouping.

How to use it well:

Build queries step by step: start with the basic SELECT...FROM, add the WHERE clause, then add GROUP BY and ORDER BY. Testing each part incrementally prevents complex errors.

Prompt 58: Primary Keys, Foreign Keys, and Referential Integrity

Copy this prompt into your AI tool:

Test me on the concepts of primary keys, foreign keys, composite keys, and referential integrity. Present me with a database scenario and ask me to: identify appropriate primary keys for each table (including when a composite key is needed), identify foreign keys that link tables together, explain what referential integrity means, and describe what should happen when a record referenced by a foreign key is deleted (CASCADE, SET NULL, RESTRICT). Present 5 questions one at a time. Give detailed feedback.

What this helps you practise:

Identifying primary and foreign keys, understanding composite keys, and enforcing referential integrity.

How to use it well:

A primary key must be unique and not null. A foreign key references a primary key in another table. Referential integrity ensures that every foreign key value has a matching primary key — understand why this matters for data consistency.

Prompt 59: Entity-Relationship Modelling

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Present me with a real-world scenario (such as a school timetabling system, an online bookshop, or a hospital patient management system) and ask me to identify the entities, their attributes, and the

relationships between them (one-to-one, one-to-many, many-to-many). Ask me to draw an entity-relationship diagram in text form showing entities, attributes, primary keys, and relationship types. Then evaluate my design and suggest improvements or point out missing entities and relationships. Give detailed feedback.

What this helps you practise:

Identifying entities, attributes, and relationships from requirements and creating entity-relationship diagrams.

How to use it well:

Start by listing all the nouns in the scenario — these are likely entities or attributes. Then identify the verbs that link entities — these indicate relationships. Check every relationship cardinality carefully.

Prompt 60: SQL JOIN Operations

Copy this prompt into your AI tool:

Test me on SQL JOIN operations at A-Level depth. Present a database with multiple related tables and ask me to write queries using: INNER JOIN (matching records in both tables), LEFT JOIN (all records from the left table), RIGHT JOIN (all records from the right table), and multi-table joins. Present 6 query challenges one at a time, each requiring a different type of join. After each query, check my syntax and logic, and explain any errors. Then ask me to explain the difference between INNER and OUTER joins with a specific example.

What this helps you practise:

Writing SQL JOIN queries to combine data from multiple related tables.

How to use it well:

Always identify which tables you need data from and what field links them before writing the JOIN.

Visualising the tables and their relationships helps you construct the correct JOIN condition.

Prompt 61: SQL INSERT, UPDATE, and DELETE

Copy this prompt into your AI tool:

Quiz me on SQL data manipulation statements. Present a database schema and ask me to write: INSERT statements to add new records (including into tables with foreign key constraints), UPDATE statements to modify existing records (using WHERE clauses to target specific rows), and DELETE statements to remove records. Present 6 tasks one at a time. After each, check my syntax and logic, particularly whether my WHERE clauses are correct and whether I have considered referential integrity constraints. Give detailed feedback.

What this helps you practise:

Writing SQL INSERT, UPDATE, and DELETE statements with proper use of WHERE clauses and referential integrity.

How to use it well:

Never write an UPDATE or DELETE without a WHERE clause unless you genuinely want to affect all rows. This is both a practical programming rule and a common exam pitfall.

Prompt 62: Subqueries and Nested SELECT

Copy this prompt into your AI tool:

Act as my tutor. Explain how subqueries (nested SELECT statements) work in SQL and when they are useful. Present me with 5 query challenges one at a time that require subqueries: using a subquery in a WHERE clause, using a subquery with IN, using a subquery with EXISTS, using a correlated subquery, and rewriting a subquery as a JOIN. After each query, check my syntax and logic, and explain any errors. Then ask me to explain when a subquery is

preferable to a JOIN and vice versa. Give detailed feedback.

What this helps you practise:

Writing SQL subqueries and understanding when to use subqueries versus JOINs.

How to use it well:

A subquery runs first and its result is used by the outer query. Think of it as breaking a complex question into two simpler questions: the inner query answers the first, and the outer query uses that answer.

Prompt 63: Transaction Processing and ACID

Copy this prompt into your AI tool:

Quiz me on transaction processing in databases. Ask me to explain: what a transaction is, the ACID properties (Atomicity, Consistency, Isolation, Durability), why transactions are necessary, what can go wrong without them (lost updates, dirty reads, phantom reads), and how locking mechanisms prevent concurrency problems. Present 4 scenarios one at a time and ask me to explain what could go wrong without proper transaction management and which ACID property would be violated. Give detailed feedback.

What this helps you practise:

Understanding transaction processing, ACID properties, and concurrency control in databases.

How to use it well:

Think of ACID as a set of guarantees that databases provide. Atomicity means all or nothing, Consistency means rules are maintained, Isolation means transactions do not interfere, and Durability means completed transactions persist.

Prompt 64: Client-Server Databases and SQL Injection

Copy this prompt into your AI tool:

Test me on client-server database architectures and security. Ask me to explain how a client-server database works, the role of the database management system (DBMS), how queries are sent from the client to the server, and what SQL injection is. Then present me with 3 examples of vulnerable code and ask me to: identify the SQL injection vulnerability, demonstrate how an attacker could exploit it, and explain how to prevent it (parameterised queries, input validation). Give detailed feedback on each answer.

What this helps you practise:

Understanding client-server databases, SQL injection attacks, and prevention techniques.

How to use it well:

SQL injection is a critical security topic that appears in both the database and security sections of the exam. Always recommend parameterised queries as the primary defence — string concatenation of user input into SQL is never safe.

Prompt 65: Database Design from Requirements

Copy this prompt into your AI tool:

Present me with a detailed real-world scenario with business requirements (such as an e-commerce platform tracking customers, orders, products, and reviews) and ask me to: design a complete database schema with appropriate tables, attributes, data types, primary keys, and foreign keys; normalise the design to 3NF; and write 5 SQL queries that the business would need. Evaluate my entire design for correctness, normalisation quality, and query accuracy. Suggest improvements. Give detailed feedback.

What this helps you practise:

Designing a complete normalised database from business requirements and writing practical queries.

How to use it well:

Work through the design process systematically: requirements analysis, entity identification, relationship mapping, attribute assignment, normalisation, then query writing. Each step builds on the previous one.

Prompt 66: Database and SQL Mastery Test

Copy this prompt into your AI tool:

Give me a comprehensive database exam. Present a multi-table database schema and then give me 10 tasks one at a time covering: writing SELECT queries with conditions, writing JOIN queries, using aggregate functions with GROUP BY, writing a subquery, inserting data with foreign key constraints, normalising an unnormalised table to 3NF, identifying primary and foreign keys, explaining referential integrity, writing an UPDATE with a WHERE clause, and explaining an ACID property with an example. Mark each answer and give a final score. Identify my two weakest areas.

What this helps you practise:

Comprehensive assessment of database design and SQL skills across all A-Level sub-topics.

How to use it well:

Write your SQL queries carefully, checking syntax before submitting. Common errors include missing commas, incorrect table aliases, and wrong JOIN conditions — proofread your queries.

Section 7

Software Development Methodologies

Software development at A-Level goes beyond simply writing code. You need to understand the full software development lifecycle and the different methodologies that teams use to plan, design, implement, test, and maintain software systems. This includes traditional approaches such as the waterfall model and iterative development, as well as modern agile methodologies including Scrum and Extreme Programming.

You must also understand the principles of good software engineering: writing maintainable and reusable code, using appropriate design techniques (top-down design, modular programming), applying testing strategies (unit testing, integration testing, system testing, acceptance testing), and understanding the role of documentation in the development process. These principles underpin the Non-Exam Assessment as well as the written examination papers.

These prompts will test your understanding of development methodologies, testing strategies, and software engineering principles. They require you to analyse scenarios, recommend appropriate approaches, and evaluate the strengths and limitations of different methodologies. Use them to build the analytical understanding that allows you to discuss software development with the precision and depth that A-Level examiners expect.

Prompt 67: The Software Development Lifecycle

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Quiz me on the stages of the software development lifecycle

(SDLC): analysis, design, implementation, testing, and maintenance. For each stage, ask me to explain its purpose, the key activities involved, and the outputs produced. Then present me with 3 project scenarios and ask me to explain how each SDLC stage would be applied in that context. Give detailed feedback on the depth and accuracy of my explanations.

What this helps you practise:

Explaining each stage of the software development lifecycle with practical applications.

How to use it well:

Learn the stages in order and understand what happens at each stage and why it matters. Exam questions often present scenarios and ask you to describe how a specific stage would be carried out.

Prompt 68: Testing Strategies

Copy this prompt into your AI tool:

Quiz me on software testing at A-Level depth. Present the following testing types one at a time: unit testing, integration testing, system testing, acceptance testing, alpha and beta testing, white-box testing, and black-box testing. For each, ask me to explain what it tests, when it occurs in the development process, who performs it, and give an example. Then present me with test scenarios and ask me to design appropriate test cases including normal, boundary, and erroneous data. Give detailed feedback on my test case designs.

What this helps you practise:

Understanding different testing types and designing effective test cases with normal, boundary, and erroneous data.

How to use it well:

Test case design is a practical skill tested in exams. For every function, identify: normal inputs that

should work correctly, boundary inputs at the edge of valid ranges, and erroneous inputs that should be rejected.

Prompt 69: Modular Design and Decomposition

Copy this prompt into your AI tool:

Act as my tutor. Explain top-down design, modular programming, and the benefits of decomposing a large problem into smaller, manageable modules. Then present me with a complex programming task (such as designing a student grade management system) and ask me to decompose it into modules, identify the interfaces between modules (parameters and return values), and explain how modular design improves maintainability, reusability, and testing. Evaluate my decomposition and suggest improvements. Give detailed feedback.

What this helps you practise:

Applying top-down design and modular decomposition to complex programming problems.

How to use it well:

Each module should have a single, clear responsibility and well-defined inputs and outputs. If a module is doing too many things, decompose it further. This principle of 'separation of concerns' is fundamental to good software design.

Prompt 70: Version Control and Collaborative Development

Copy this prompt into your AI tool:

Quiz me on version control systems and collaborative development practices. Ask me to explain: what version control is and why it is essential, how branching and merging work, the difference between centralised and distributed version control systems (such as Git), the concept of commits, repositories, and pull requests, and how

version control supports team collaboration. Present 4 scenarios one at a time and ask me to explain how version control would be used in each. Give detailed feedback.

What this helps you practise:

Understanding version control systems, branching strategies, and collaborative development workflows.

How to use it well:

Even if your NEA is an individual project, you should use version control. Understanding Git workflows — commit, branch, merge, pull request — is valuable both for the exam and for practical programming.

Prompt 71: Documentation and Code Quality

Copy this prompt into your AI tool:

Act as my examiner. Test me on code quality and documentation practices. Ask me to explain: the importance of meaningful variable and function names, the role of comments and docstrings, the purpose of technical documentation and user documentation, the concept of code refactoring, and coding standards and conventions. Then present me with a poorly written code snippet and ask me to identify the quality issues and rewrite it following best practices. Give detailed feedback on my improvements.

What this helps you practise:

Understanding code quality principles, documentation practices, and applying them to improve code.

How to use it well:

Good code should be self-documenting through clear naming conventions, but comments are still necessary for complex logic. Balance self-documenting code with appropriate comments.

Prompt 72: Agile Practices: Scrum and Kanban

Copy this prompt into your AI tool:

Quiz me on specific agile practices at A-Level depth. Ask me to explain the Scrum framework including: roles (Product Owner, Scrum Master, Development Team), artefacts (product backlog, sprint backlog, increment), and ceremonies (sprint planning, daily stand-up, sprint review, retrospective). Then explain Kanban and its key principles (visualise workflow, limit WIP, manage flow). Present 3 scenarios and ask me to recommend Scrum, Kanban, or neither, with justifications. Give detailed feedback.

What this helps you practise:

Understanding Scrum and Kanban agile frameworks and selecting the appropriate approach for different projects.

How to use it well:

Scrum works well for new product development with fixed-length iterations. Kanban works well for continuous flow work like support and maintenance. Understanding this distinction helps you recommend the right approach.

Prompt 73: Debugging and Error Handling Strategies

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Explain the different types of programming errors: syntax errors, runtime errors, and logic errors. Then explain debugging strategies including: trace tables, breakpoints and stepping through code, watch variables, and the use of print statements. Present me with 4 code snippets one at a time, each containing a different type of error. For each, ask me to identify the type of error, locate the bug, explain why it occurs, and fix it. Also ask me to explain the difference between error handling (try/except) and

error prevention (validation). Give detailed feedback.

What this helps you practise:

Identifying, classifying, and fixing different types of programming errors using systematic debugging strategies.

How to use it well:

When debugging, start by classifying the error type — this narrows down where to look. Syntax errors are caught by the translator, runtime errors crash the program at a specific point, and logic errors produce wrong results without crashing.

Prompt 74: File Handling and Data Persistence

Copy this prompt into your AI tool:

Test me on file handling at A-Level depth. Ask me to explain: the difference between text files and binary files, file operations (open, read, write, append, close), reading and writing CSV files, serialisation and deserialisation of objects, and the importance of exception handling when working with files. Then present me with 4 practical coding tasks one at a time: reading data from a CSV file into a data structure, writing program output to a text file, appending to an existing file, and handling file-not-found errors gracefully. Check my code and give detailed feedback.

What this helps you practise:

Implementing file handling operations including reading, writing, and error handling for persistent data storage.

How to use it well:

Always close files after use or use context managers (with statements). Forgetting to close files can lead to data loss and resource leaks — this is both a practical programming principle and an exam point.

Prompt 75: Waterfall vs Agile Methodologies

Copy this prompt into your AI tool:

Test me on the comparison between the waterfall model and agile methodologies. Ask me to explain how each approach works, including the sequential nature of waterfall and the iterative sprint-based approach of agile. Then present me with 5 project scenarios one at a time (such as developing safety-critical software for an aircraft, building a social media app, creating a government tax system, developing a startup's MVP, and maintaining a legacy banking system) and ask me to recommend and justify the most appropriate methodology. Give detailed feedback on my justifications.

What this helps you practise:

Comparing waterfall and agile methodologies and selecting the appropriate approach for different project types.

How to use it well:

Consider the project characteristics that favour each methodology: stable requirements favour waterfall; evolving requirements favour agile; safety-critical systems may need waterfall's rigour; user-facing products benefit from agile's feedback loops.

Prompt 76: Rapid Application Development and Prototyping

Copy this prompt into your AI tool:

Test me on Rapid Application Development (RAD) and the role of prototyping in software development. Ask me to explain: how RAD works, the different types of prototyping (throwaway, evolutionary, incremental), the advantages and disadvantages of prototyping, and when prototyping is most useful. Then present me with 3 scenarios and ask me whether prototyping would be appropriate for each,

and if so, which type. Give detailed feedback on my answers and reasoning.

What this helps you practise:

Understanding RAD and prototyping approaches, and evaluating their suitability for different project contexts.

How to use it well:

Prototyping is most useful when requirements are unclear and user feedback is essential for refining them. Think about whether the project would benefit from early visual models or whether requirements are already well-defined.

Prompt 77: Feasibility Study and Requirements Analysis

Copy this prompt into your AI tool:

Test me on the analysis stage of software development. Ask me to explain: what a feasibility study covers (technical, economic, legal, operational, schedule feasibility), how requirements are gathered (interviews, questionnaires, observation, document analysis), the difference between functional and non-functional requirements, and how requirements are documented (use cases, user stories). Then present a scenario and ask me to: conduct a brief feasibility assessment, identify functional and non-functional requirements, and write user stories. Give detailed feedback.

What this helps you practise:

Conducting feasibility analysis, gathering requirements, and distinguishing functional from non-functional requirements.

How to use it well:

Requirements analysis is often tested through scenario-based questions. Practice identifying requirements from a description — functional requirements describe what the system must do;

non-functional requirements describe how well it must do it.

Prompt 78: Development Methodologies Exam Preparation

Copy this prompt into your AI tool:

Give me a mini-exam on software development methodologies. Include questions on: comparing waterfall and agile for a given scenario, explaining three types of testing with examples, designing test cases with normal, boundary, and erroneous data, explaining the benefits of modular design, describing the stages of the SDLC, and evaluating whether prototyping would be appropriate for a given project. Present 8 questions one at a time, mark each answer, and give a final score. Identify my weakest area with specific revision advice.

What this helps you practise:

Comprehensive assessment of software development methodology knowledge across all sub-topics.

How to use it well:

Development methodology questions often require you to justify your answer with reasoning rather than simply stating facts. Always explain why a particular approach is suitable or unsuitable for the given context.

Section 8

Theory of Computation

The theory of computation is one of the most challenging and distinctive topics at A-Level Computer Science. It takes you into the mathematical and theoretical foundations of computing, covering Boolean algebra and logic gates, finite state machines, regular expressions, the Backus-Naur Form (BNF) for defining languages, and Turing machines. These concepts underpin everything else in computer science and help you understand what computers can and cannot do.

Boolean algebra at A-Level goes beyond simple truth tables to include algebraic simplification, De Morgan's laws, and the use of Karnaugh maps for minimising Boolean expressions. Finite state machines (FSMs) are used to model computation, and you must be able to design, interpret, and trace the execution of both deterministic and non-deterministic FSMs. Regular expressions provide a compact notation for describing patterns, and you must be able to write and interpret them.

These prompts will test your understanding of theoretical concepts through active problem-solving. You will simplify Boolean expressions, design finite state machines, write regular expressions, define languages using BNF, and explore the concept of computability through Turing machines. These topics require careful, precise thinking — use pen and paper alongside the AI to work through problems methodically.

Prompt 79: Finite State Machines

Copy this prompt into your AI tool:

Act as my A-Level tutor. Explain finite state machines (FSMs) including states, transitions, inputs, outputs, start states, and accepting states. Explain the difference between Mealy and Moore machines. Then present me with 4 tasks one at a time: design an FSM for a given problem (such as accepting strings that end in '01'), trace the execution of a given FSM on a specific input string, convert between a state transition diagram and a state transition table, and identify the language accepted by a given FSM. Check each answer and give detailed feedback.

What this helps you practise:

Designing, tracing, and interpreting finite state machines for pattern recognition and language acceptance.

How to use it well:

Draw FSM diagrams clearly with labelled states and transitions. When tracing, show the current state and remaining input at each step. This systematic approach prevents errors.

Prompt 80: Regular Expressions and FSMs

Copy this prompt into your AI tool:

Act as my tutor. Explain the relationship between regular expressions and finite state machines — that they describe the same class of languages (regular languages). Then present me with 3 regular expressions one at a time and ask me to design an FSM that accepts the same language. For each, check my FSM design by tracing several test strings through it. Then present an FSM and ask me to write the corresponding regular expression. Give detailed feedback on each conversion.

What this helps you practise:

Converting between regular expressions and finite state machines and understanding their equivalence.

How to use it well:

This conversion tests your understanding of both formalisms. Start by identifying the simple patterns in the regular expression and mapping each to states and transitions in the FSM.

Prompt 81: Backus-Naur Form (BNF)

Copy this prompt into your AI tool:

Test me on Backus-Naur Form (BNF) notation. Give me a simple language rule described in English and ask me to write the BNF definition for it. Then give me a BNF definition and ask me to determine whether specific strings are valid according to that grammar. Wait for my answers each time and check whether my syntax and reasoning are correct. Finally, ask me to explain the difference between BNF and Extended BNF.

What this helps you practise:

Reading, writing, and applying BNF grammars to define and validate formal languages.

How to use it well:

Trace through BNF derivations step by step, replacing non-terminals with their definitions until you reach a string of terminals. This systematic approach is how you verify whether a string belongs to the language.

Prompt 82: Boolean Algebra Simplification

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Teach me Boolean algebra simplification rules including: AND, OR, NOT, commutative, associative, distributive, identity, complement, double negation, absorption, and De Morgan's laws. Present me with 6 Boolean expressions one at a time and ask me to simplify each as far as possible, showing my working and stating which laws I apply at each step. After

each simplification, check my working and correct any errors. Give detailed feedback.

What this helps you practise:

Simplifying Boolean expressions using algebraic laws including De Morgan's laws and absorption.

How to use it well:

Write out each step of your simplification and label the law you used. This systematic approach prevents errors and is what examiners expect to see as your working.

Prompt 83: Karnaugh Maps

Copy this prompt into your AI tool:

Test me on using Karnaugh maps for simplifying Boolean expressions. Give me a Boolean expression with four variables and ask me to draw the K-map, group the cells correctly, and derive the simplified expression. Then ask me to explain the rules for grouping (groups must be powers of 2, wrapping is allowed). Check my working and identify any errors in my grouping or simplification.

What this helps you practise:

Using Karnaugh maps to simplify Boolean expressions with up to four variables.

How to use it well:

Draw your K-maps carefully with the correct Grey code ordering of column and row headings. Remember that groups must be powers of 2 and can wrap around edges. Missing valid groups is the most common error.

Prompt 84: Logic Gate Circuits

Copy this prompt into your AI tool:

Test me on logic gate circuits. Present me with 4 tasks one at a time: given a Boolean expression, ask me to draw the corresponding logic gate circuit; given a circuit diagram (described in text), ask me to

derive the Boolean expression; given a circuit, ask me to produce the truth table; and given a truth table, ask me to derive the simplified Boolean expression using algebraic laws or a K-map and then draw the minimised circuit. Check each answer and explain any errors. Give detailed feedback.

What this helps you practise:

Converting between Boolean expressions, logic gate circuits, and truth tables in all directions.

How to use it well:

Practise moving between all three representations: expression, circuit, and truth table. Being fluent in these conversions is essential for the exam, where questions may start from any representation.

Prompt 85: Regular Expressions

Copy this prompt into your AI tool:

Test me on regular expressions at A-Level depth. Explain the core operators: concatenation, alternation (|), Kleene star (*), Kleene plus (+), and optional (?). Then present 8 tasks one at a time mixing two types: given a description of a set of strings, ask me to write a regular expression that matches them; and given a regular expression, ask me to describe the set of strings it matches and provide examples of matching and non-matching strings. Check each answer and correct any errors. Give detailed feedback.

What this helps you practise:

Writing and interpreting regular expressions to describe sets of strings.

How to use it well:

Build regular expressions incrementally: start with the simplest pattern that matches some of the required strings, then refine it to match all required strings and reject non-matching ones. Test your expression against edge cases.

Prompt 86: Turing Machines

Copy this prompt into your AI tool:

Test me on Turing machines. Explain the components of a Turing machine (infinite tape, read/write head, finite set of states, transition function), describe how it computes, and explain why it is important as a theoretical model of computation. Then present me with 3 tasks one at a time: trace the execution of a simple Turing machine on a given input, design a Turing machine to perform a specific task (such as adding 1 to a binary number or copying a string), and explain what it means for a problem to be Turing-decidable versus Turing-recognisable. Give detailed feedback.

What this helps you practise:

Understanding, tracing, and designing Turing machines as a model of universal computation.

How to use it well:

Show the tape contents, head position, and current state at each step when tracing. This step-by-step trace is what examiners expect and prevents errors.

Prompt 87: Computability and the Halting Problem

Copy this prompt into your AI tool:

Act as my tutor. Explain the concept of computability: what it means for a problem to be computable, the Church-Turing thesis, and why some problems are not computable. Then explain the halting problem: what it is, why it matters, and the proof (by contradiction) that no algorithm can solve it. Ask me to explain the halting problem proof in my own words, then present 3 questions testing my understanding of computability, including identifying examples of computable and non-computable problems. Give detailed feedback.

What this helps you practise:

Understanding computability, the Church-Turing thesis, and the halting problem including its proof of undecidability.

How to use it well:

The halting problem proof by contradiction is challenging but important. Walk through the proof slowly: assume a halting detector exists, construct a program that creates a paradox, and show that the assumption must be wrong.

Prompt 88: Classification of Languages

Copy this prompt into your AI tool:

Test me on the Chomsky hierarchy and the classification of formal languages. Ask me to explain: regular languages and their recognisers (finite state machines), context-free languages and their recognisers (pushdown automata), context-sensitive languages, and recursively enumerable languages (recognised by Turing machines). For each class, ask me to give an example language and explain why it belongs to that class. Then present me with 4 specific languages and ask me to classify each, justifying my answer. Give detailed feedback.

What this helps you practise:

Understanding the Chomsky hierarchy and classifying formal languages by their computational complexity.

How to use it well:

The key relationships are: every regular language is context-free, every context-free language is context-sensitive, and so on. Each level adds computational power. Understanding what each level can and cannot express is the key insight.

Prompt 89: Theory of Computation Exam Drill

Copy this prompt into your AI tool:

Give me a comprehensive exam-style test on the theory of computation. Include: a Boolean algebra simplification problem, a K-map minimisation, a finite state machine design task, a regular expression writing task, a BNF grammar interpretation task, a Turing machine trace, and a question about the halting problem. Present each question one at a time, mark my answer with full working shown, and give a model answer. At the end, give me a score and identify my two weakest areas with revision advice.

What this helps you practise:

Comprehensive assessment of theory of computation knowledge across all sub-topics under exam conditions.

How to use it well:

Theory of computation questions require precise formal reasoning. Show all your working — in Boolean algebra, label each law you apply; in FSM traces, show each state transition; in K-maps, circle your groups clearly.

Section 9

Cyber Security and Legal/Ethical Issues

Cyber security at A-Level encompasses both the technical mechanisms used to protect computer systems and data, and the legal and ethical frameworks that govern the use of technology. You must understand common attack vectors and the technical defences against them, as well as the principles of cryptography, authentication, and access control. This topic connects directly to networking, databases, and programming through the security vulnerabilities that arise in each area.

The legal and ethical dimensions of computing are also examined at A-Level. You need to understand the key legislation governing data protection (including the Data Protection Act 2018 and GDPR), computer misuse (the Computer Misuse Act 1990), intellectual property (copyright, patents, Creative Commons), and freedom of information. Beyond the law, you must be able to discuss the ethical implications of emerging technologies including artificial intelligence, surveillance, automation, and algorithmic decision-making.

These prompts will test your knowledge of both technical security measures and the legal and ethical context of computing. They require you to analyse scenarios, evaluate the effectiveness of security measures, and discuss ethical dilemmas with nuance and balance. Use them to develop the ability to answer questions that require both technical precision and thoughtful ethical reasoning.

Prompt 90: Common Cyber Security Threats

Copy this prompt into your AI tool:

Act as my A-Level Computer Science tutor. Quiz me on common cyber security threats. Present the following one at a time: malware (viruses, worms, trojans, ransomware, spyware), social engineering (phishing, pretexting, baiting), denial-of-service (DoS and DDoS) attacks, man-in-the-middle attacks, SQL injection, and cross-site scripting (XSS). For each threat, ask me to: explain how it works, identify what it exploits (technical vulnerability or human vulnerability), and describe how to defend against it. Give detailed feedback on the accuracy and depth of my explanations.

What this helps you practise:

Understanding common cyber security threats, their mechanisms, and appropriate countermeasures.

How to use it well:

Categorise threats into those that exploit technical vulnerabilities and those that exploit human vulnerabilities. This distinction helps you recommend appropriate defences — technical controls for technical attacks, training and awareness for social engineering.

Prompt 91: Encryption: Symmetric and Asymmetric

Copy this prompt into your AI tool:

Test me on encryption at A-Level depth. Ask me to explain: the difference between symmetric and asymmetric encryption, the key distribution problem and how asymmetric encryption solves it, how digital signatures work using public and private keys, the role of hashing in password storage and data integrity, and the concept of key exchange (Diffie-Hellman). Present 5 questions one at a time including practical scenarios requiring me to choose the appropriate encryption method. Give detailed feedback.

What this helps you practise:

Understanding symmetric and asymmetric encryption, digital signatures, hashing, and key exchange mechanisms.

How to use it well:

Remember that symmetric encryption uses one key (fast but key distribution is a problem), while asymmetric uses a key pair (slower but solves key distribution). Know when to use each and how they can be combined.

Prompt 92: Authentication and Access Control

Copy this prompt into your AI tool:

Quiz me on authentication and access control mechanisms. Ask me to explain: the three factors of authentication (something you know, have, are), multi-factor authentication, biometric authentication (advantages and disadvantages), role-based access control (RBAC), the principle of least privilege, and CAPTCHA systems. Then present me with 4 scenarios and ask me to design an appropriate authentication and access control scheme for each, justifying my choices. Give detailed feedback on my designs.

What this helps you practise:

Designing appropriate authentication and access control schemes for different security requirements.

How to use it well:

Match the level of security to the sensitivity of the data and the risk. A social media login needs different security measures than a banking system. Always consider usability alongside security.

Prompt 93: Network Security in Practice

Copy this prompt into your AI tool:

Test me on practical network security at A-Level depth. Present a scenario involving a business

network and ask me to identify potential security vulnerabilities. Then quiz me on: firewall configuration and types, intrusion detection systems (IDS) and intrusion prevention systems (IPS), virtual private networks (VPNs), penetration testing, and security policies and procedures. For each topic, ask me to explain the technology and how it reduces risk. Present 4 practical questions one at a time. Give detailed feedback.

What this helps you practise:

Applying network security technologies and practices to protect systems in realistic scenarios.

How to use it well:

Think of security as layers of defence (defence in depth). No single measure is sufficient — the strongest security combines multiple technical controls with proper policies and user training.

Prompt 94: The Data Protection Act and GDPR

Copy this prompt into your AI tool:

Quiz me on data protection legislation at A-Level depth. Ask me to explain: the key principles of the Data Protection Act 2018 and GDPR (lawfulness, purpose limitation, data minimisation, accuracy, storage limitation, security, accountability), the rights of data subjects (access, rectification, erasure, portability), the role of the Information Commissioner's Office (ICO), and the concept of privacy by design. Then present 4 scenarios involving data handling and ask me to identify any legal issues. Give detailed feedback.

What this helps you practise:

Understanding data protection principles, data subject rights, and applying legislation to practical scenarios.

How to use it well:

Learn the key principles by heart and be able to

apply them to unfamiliar scenarios. Exam questions often present a case study and ask you to identify which principles are being violated and what should be done differently.

Prompt 95: The Computer Misuse Act

Copy this prompt into your AI tool:

Test me on the Computer Misuse Act 1990 and its amendments. Ask me to explain the three main offences: unauthorised access to computer material, unauthorised access with intent to commit further offences, and unauthorised modification of computer material. Then present me with 5 scenarios one at a time (such as an employee accessing a colleague's email, a student modifying school records, a researcher testing a website's security without permission) and ask me to identify which offence, if any, has been committed. Give detailed feedback on my analysis.

What this helps you practise:

Understanding the Computer Misuse Act offences and applying them to specific scenarios.

How to use it well:

The key word in all three offences is 'unauthorised'. Consider carefully in each scenario whether access or modification was authorised, and by whom. Authorisation matters more than intent.

Prompt 96: Intellectual Property and Copyright

Copy this prompt into your AI tool:

Quiz me on intellectual property in the context of computing. Ask me to explain: copyright and how it applies to software, the difference between proprietary and open-source software licences, Creative Commons licences and their variants, patents and their role in protecting inventions, and the ethical debates around software piracy and

DRM. Then present 4 scenarios involving intellectual property issues and ask me to identify the legal and ethical considerations. Give detailed feedback.

What this helps you practise:

Understanding intellectual property law as it applies to software, including copyright, licensing, and patents.

How to use it well:

Know the difference between copyright (protects expression), patents (protect inventions), and trademarks (protect brand identity). Exam questions often test whether you can identify which form of IP protection applies to a given situation.

Prompt 97: Ethical Issues in Computing

Copy this prompt into your AI tool:

Act as my tutor. Present me with 5 ethical dilemmas in computing one at a time: artificial intelligence and algorithmic bias, surveillance and privacy (CCTV, internet monitoring), autonomous vehicles and moral decisions, the digital divide and access inequality, and the impact of automation on employment. For each dilemma, ask me to: identify the key ethical issues, present arguments on both sides, and reach a balanced conclusion. Evaluate whether my responses demonstrate genuine ethical reasoning rather than one-sided argument. Give detailed feedback.

What this helps you practise:

Analysing ethical dilemmas in computing with balanced reasoning considering multiple perspectives.

How to use it well:

Ethical questions in the exam require balanced analysis. Present arguments from multiple perspectives — individuals, businesses, society,

government — and reach a nuanced conclusion rather than a simplistic one.

Prompt 98: Environmental Impact of Technology

Copy this prompt into your AI tool:

Test me on the environmental impact of computing technology. Ask me to explain: the energy consumption of data centres and cryptocurrency mining, the environmental cost of manufacturing electronic devices, the problem of e-waste and its disposal, the role of technology in monitoring and addressing climate change, and strategies for making computing more sustainable (green computing). Present 4 questions one at a time and ask me to evaluate both the negative environmental impact and the positive potential of technology. Give detailed feedback.

What this helps you practise:

Understanding the environmental impact of computing and evaluating strategies for sustainable technology use.

How to use it well:

Show awareness of both sides: technology contributes to environmental problems through energy use and waste, but also provides tools for monitoring, modelling, and addressing environmental challenges.

Prompt 99: Cyber Security and Ethics Exam Challenge

Copy this prompt into your AI tool:

Give me a comprehensive exam on cyber security and legal/ethical issues. Include questions on: identifying a security threat and recommending countermeasures, explaining how encryption protects data, applying the Data Protection Act to a

scenario, identifying Computer Misuse Act offences, discussing an ethical dilemma with balanced arguments, and evaluating the effectiveness of a security measure. Present 8 questions one at a time, mark each answer, and give a final score. Identify my two weakest areas with specific revision advice.

What this helps you practise:

Comprehensive assessment of cyber security knowledge and legal/ethical understanding across all sub-topics.

How to use it well:

Security and ethics questions often require extended written answers. Practise structuring your responses with clear paragraphs, specific examples, and balanced evaluation.

Prompt 100: Security Audit Scenario

Copy this prompt into your AI tool:

Present me with a detailed description of a small company's IT infrastructure (including their network setup, data storage practices, staff training, and current security measures). Ask me to conduct a security audit by: identifying at least 5 vulnerabilities in their current setup, ranking them by severity, recommending specific countermeasures for each vulnerability, and suggesting a priority order for implementing the improvements. Evaluate my audit for thoroughness, accuracy, and practicality. Give detailed feedback on any vulnerabilities I missed or any impractical recommendations.

What this helps you practise:

Conducting a security audit on a realistic scenario, identifying vulnerabilities, and recommending practical countermeasures.

How to use it well:

Think systematically about all attack surfaces:

physical security, network security, application security, human factors, and data management. A thorough audit considers all of these dimensions.

Final Closing Note

You have now worked through 100 prompts designed to help you think more clearly, revise more effectively, and prepare more confidently for your GCSE.

Remember: the goal was never to rely on AI for answers. The goal was to use it as a tool to test, challenge, and strengthen your own understanding.

The strongest students are not those who avoid difficulty, but those who engage with it deliberately. Each mistake you identified, each explanation you improved, and each gap you filled has strengthened your thinking.

As you continue your studies, aim to depend less on prompts and more on your own judgement. AI can support you — but your reasoning, clarity, and persistence are what earn marks.
Approach your exams calmly. Think carefully. Write clearly.
You are more prepared than you think.

Using AI Beyond This Book

The prompts in this book are starting points, not final forms.

As you grow more confident, begin modifying them:

- Add constraints (for example, “limit to three key points”).
- Increase difficulty gradually.
- Ask the AI to challenge your reasoning.
- Request alternative explanations.
- Ask it to critique your thinking rather than provide answers.

The most powerful use of AI is not asking it to tell you things — it is asking it to test and refine your thinking.

In the future, those who understand how to use tools intelligently will have an advantage. Treat AI as a tutor, not a shortcut. The skill of asking better questions will continue to matter long after your exams are over.

About the Author

James R. Martin holds an MSci in Physics from the University of Bristol and a PGCE with a Physics focus from the University of Oxford. He has over a decade of experience teaching and tutoring students aged 11–18 across a range of subjects, including Physics, Biology, Chemistry, Mathematics, Economics, and Electronics.

He has worked with multiple syllabi, including GCSE, A-Level, KS3, and the International Baccalaureate Diploma Programme (IBDP), supporting students of varying abilities to develop clarity, confidence, and exam success.

His work focuses on effective revision strategies, independent thinking, and the responsible use of artificial intelligence as a tool to strengthen — not replace — understanding.

Other Titles in This Series

The *100 AI Prompts for Smarter Revision* series supports students across GCSE, A-Level, and IBDP subjects.

GCSE
• English Language
• English Literature
• Mathematics
• Physics
• Biology
• Chemistry
• Geography
• History
• Computer Science
• Economics
• Business Studies
• Religious Studies
• Psychology
• French
• Spanish
• German

A-Level
• Mathematics
• Further Mathematics
• Physics
• Chemistry
• Biology
• Economics
• History
• Geography
• English Literature
• Psychology
• Computer Science

• Politics
• Business

IBDP

• Mathematics: Analysis & Approaches
• Mathematics: Applications & Interpretation
• Physics
• Chemistry
• Biology
• Economics
• Geography
• History
• English A: Literature
• English A: Language & Literature
• Psychology
• Business Management
• Computer Science