MAH MCA CET Computer Concepts

Sample Paper – 2

Duration: 23 Minutes **Maximum Marks: 50**

Instructions

- This paper contains **25** Multiple Choice Questions (Single Correct Answer), modelled on the Computer Concepts section of **MAH MCA CET**.
- Each correct answer carries **+2 marks**. There is **no negative marking** for incorrect or unattempted answers.
- Only **one** option is correct per question.
- Use of mobile phones, calculators, or electronic gadgets is strictly prohibited. Rough work may be done in the space provided in the booklet.

Q1. In the **Von Neumann architecture**, which single component is responsible for storing **both** the program instructions and the data on which those instructions operate?

(A) Central Processing Unit (CPU)

(B) Control Unit (CU)

(C) Main Memory (RAM)

(D) Hard Disk Drive (HDD)

Q2. Which of the following statements **correctly** distinguishes a **plotter** from a conventional **printer**?

(A) A plotter is primarily used to produce large-format technical drawings such as engineering blueprints and architectural plans.

(B) A laser printer can never produce output at a higher resolution than a plotter.

(C) A plotter exclusively uses inkjet technology for all of its output.

(D) A plotter is classified as an *input* device, not an output device.

Q3. Which of the following **best** describes the concept of **paging** in virtual memory management?

(A) Paging divides memory into variable-size blocks whose size is determined dynamically by the process.

(B) The page table is stored inside CPU registers to allow the fastest possible address translation.

(C) Virtual memory through paging completely eliminates the need for physical RAM in a computer.

(D) Paging divides both physical and logical memory into fixed-size blocks (frames/pages), and a *page table* maintained by the OS maps each virtual page number to its physical frame address.

Q4. Which of the following **correctly** describes the access characteristic of **magnetic tape** as a secondary storage medium?

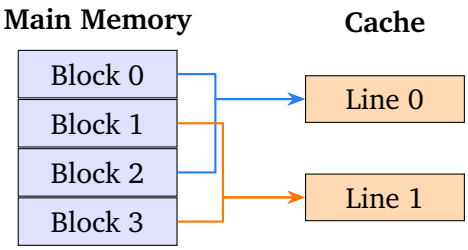
(A) Magnetic tape supports random (direct) access to any stored record, similar to a hard disk drive.

(B) Magnetic tape is a *sequential access* medium; to reach data stored earlier on the tape, the tape must be physically rewound, unlike an HDD which allows direct access to any sector.

(C) Magnetic tape provides faster data access speeds than solid-state drives (SSDs).

(D) Magnetic tape is a non-volatile *primary* storage device used directly by the CPU.

Q5. The diagram below illustrates the mapping of main memory blocks to cache lines. In a **direct-mapped cache**, which of the following **correctly** describes the mapping policy?



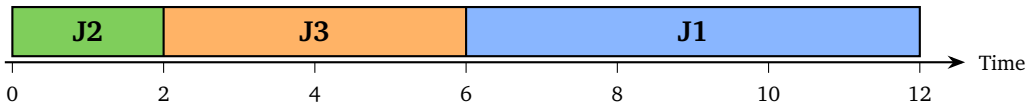
(A) Each main memory block maps to **exactly one** specific cache line, determined by block number $\bmod$ number of cache lines.

(B) Any main memory block can be placed in any available cache line (fully associative mapping).

(C) Multiple different blocks from main memory can simultaneously occupy the same cache line.

(D) A direct-mapped cache always achieves a higher hit ratio than a fully associative cache.

Q6. The Gantt chart below shows **Shortest Job First (SJF)** scheduling for three jobs arriving simultaneously at time 0: J1 (burst $= 6$), J2 (burst $= 2$), J3 (burst $= 4$). Which statement about SJF is **correct**?



(A) First Come First Served (FCFS) minimises average waiting time among all non-preemptive scheduling algorithms.

(B) Round Robin scheduling results in the fewest context switches for this set of jobs.

(C) Priority scheduling always prevents starvation of lower-priority processes.

(D) Shortest Job First (SJF) minimises average waiting time; FCFS can cause the *convoy effect* where short jobs are delayed behind a long-running job.

Q7. What is the **decimal equivalent** of the octal number $(347)_8$?

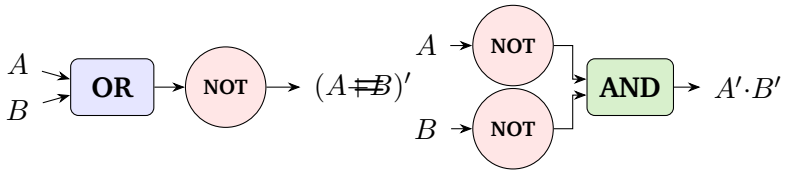
(A) 215

(B) 223

(C) 231

(D) 247

Q8. The circuit diagrams below illustrate **De Morgan's Theorem**. According to this theorem, which identity correctly represents the complement of a logical **OR** expression?



(A) $(A + B)' = A' + B'$

(B) $(A + B)' = A' \cdot B'$

(C) $(A \cdot B)' = A \cdot B'$

(D) $(A + B)' = A \cdot B$

Q9. Which of the following **correctly** describes the internal structure of a **singly linked list**?

(A) Each node contains a *data* field and a *next pointer* to the subsequent node; the last node's pointer is set to `NULL`.

(B) Each node contains two pointers: one pointing to the previous node and one pointing to the next node.

(C) A singly linked list supports $O(1)$ random access to any element by index.

(D) All nodes of a singly linked list are stored in contiguous (adjacent) memory locations.

Q10. Which of the following **correctly** describes the behaviour of a **circular queue** implemented using an array?

(A) A circular queue wastes more memory than a standard linear queue implementation.

(B) A circular queue uses an internal stack to manage overflow conditions.

(C) A circular queue does not support the dequeue (deletion) operation.

(D) In a circular queue, the rear pointer wraps around to index 0 when it reaches the end of the array, preventing a false "queue full" condition that occurs in a standard linear queue.

Q11. At which layer of the **OSI reference model** does the **HTTP (HyperText Transfer Protocol)** operate?

(A) Transport Layer (Layer 4)

(B) Network Layer (Layer 3)

(C) Application Layer (Layer 7)

(D) Session Layer (Layer 5)

Q12. What is the **primary function** of the **DNS (Domain Name System)** in computer networks?

(A) DNS dynamically assigns IP addresses to hosts on a network (like DHCP).

(B) DNS translates human-readable domain names (e.g., `www.example.com`) into their corresponding numerical IP addresses.

(C) DNS provides secure routing of email between mail servers.

(D) DNS encrypts all web traffic between client and server using SSL/TLS.

Q13. In SQL, what is the **key difference** between `COUNT(*)` and `COUNT(column_name)`?

(A) `COUNT(*)` returns the total number of rows (including rows with `NULL`s), while `COUNT(column_name)` counts only non-`NULL` values in that column.

(B) `COUNT(*)` excludes rows containing `NULL` values, whereas `COUNT(column_name)` includes them.

(C) Both `COUNT(*)` and `COUNT(column_name)` always include `NULL` values in their count.

(D) `COUNT(*)` and `COUNT(column_name)` always return the same result, regardless of `NULL` values.

Q14. Which of the following **correctly** defines **Second Normal Form (2NF)** in relational database normalisation?

(A) 2NF requires the relation to be in 1NF and all *transitive* dependencies to be eliminated.

(B) 2NF requires that every non-key attribute depends on *a* candidate key, with no restriction on partial dependencies.

(C) 2NF allows partial functional dependencies provided they involve the complete composite primary key.

(D) 2NF requires the relation to be in 1NF **and** that every non-key attribute is *fully functionally dependent* on the *entire* primary key, with no partial dependency on any subset of the key.

Q15. In an **Entity-Relationship (ER) model**, which cardinality type describes a relationship where **one** entity in set A is associated with **exactly one** entity in set B, and vice versa?

(A) One-to-Many: each entity in A is related to one or more entities in B.

(B) Many-to-Many: many entities in A are related to many entities in B.

(C) One-to-One: exactly one entity in A is related to exactly one entity in B.

(D) Many-to-One: many entities in A are all related to a single entity in B.

Q16. Which of the following statements about the `for` **loop** in the C programming language is **correct**?

(A) The `for` loop always executes its body at least once, regardless of the initial condition.

(B) The `for` loop is an *entry-controlled* loop; if the condition evaluates to false before the first iteration, the loop body does not execute at all.

(C) Both the `for` loop and the `do-while` loop are classified as exit-controlled loops.

(D) The `for` loop cannot be used to create an infinite loop in C.

Q17. What is the **worst-case time complexity** of the **Bubble Sort** algorithm?

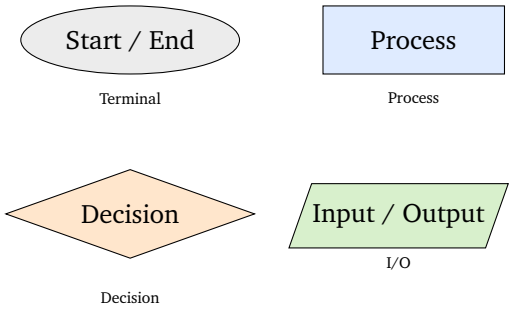
(A) $O(n^2)$ — due to the nested loop structure performing repeated adjacent comparisons

(B) $O(n \log n)$

(C) $O(n)$

(D) $O(\log n)$

Q18. The diagram below shows the four standard flowchart symbols. Which symbol is used to represent an **Input/Output** operation in a flowchart?



(A) Rectangle (Process box)

(B) Diamond (Decision box)

(C) Oval / Ellipse (Terminal)

(D) Parallelogram

Q19. Which of the following **correctly** states an advantage of the **hierarchical (tree-based) directory structure** used in modern operating systems?

(A) A hierarchical directory structure does not permit any two files to share the same filename, regardless of their directory location.

(B) A single-level directory structure is best suited for large multi-user operating systems.

(C) A hierarchical directory structure allows the same filename to exist in *different* directories, preventing naming conflicts across users and projects.

(D) A two-level directory structure supports unlimited levels of subdirectory nesting.

Q20. Why can a **Blu-ray disc** store significantly more data than a **DVD** of the same physical diameter?

(A) Blu-ray uses a longer-wavelength laser, which spaces data pits further apart, increasing capacity.

(B) Blu-ray uses a shorter-wavelength *blue-violet* laser ($\approx 405\,\text{nm}$) compared to the DVD's red laser ($\approx 650\,\text{nm}$), enabling smaller data pits and much higher storage density on the same disc area.

(C) Blu-ray and DVD use lasers of identical wavelength; the capacity difference is due solely to disc material composition.

(D) Blu-ray discs store less data than DVDs but read that data at a significantly higher speed.

Q21. Which of the following **correctly** compares the **TCP (Transmission Control Protocol)** and **UDP (User Datagram Protocol)**?

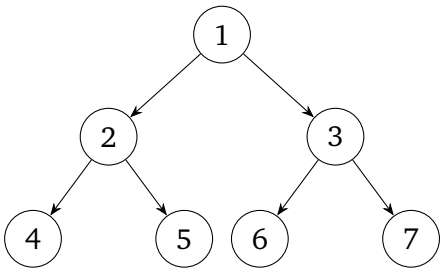
(A) TCP provides reliable, connection-oriented delivery with acknowledgements and flow control; UDP is connectionless and faster, but provides no guarantee of delivery or ordering.

(B) UDP provides reliable, ordered delivery of data while TCP is optimised for speed and low latency.

(C) TCP does not use acknowledgements; it relies on higher-level application protocols for reliability.

(D) Both TCP and UDP are connection-oriented protocols that establish a session before data transfer.

Q22. Consider the binary tree shown below. A binary tree in which every node has **either 0 or 2 children** (never exactly 1) is called a:



(A) Complete Binary Tree

(B) Perfect Binary Tree

(C) Skewed Binary Tree

(D) Full Binary Tree

Q23. What is the **decimal value** of the hexadecimal number $(FF)_{16}$?

(A) 240

(B) 250

(C) 255

(D) 256

Q24. Which of the following **correctly** identifies the standard protocols used for **sending** and **receiving** electronic mail?

(A) POP3 is used for sending email; SMTP is used for receiving email from a mail server.

(B) SMTP (Simple Mail Transfer Protocol) is used for *sending* email; POP3 and IMAP are used for *receiving* (accessing) email from a mail server.

(C) IMAP is used for sending email to a mail server; SMTP is used for retrieving messages.

(D) HTTP is the standard protocol used for both sending and receiving electronic mail.

Q25. Which of the following represents the **correct sequential order** of phases in the **Software Development Life Cycle (SDLC)**?

(A) Planning → Analysis → Design → Implementation → Testing → Maintenance

(B) Analysis → Planning → Design → Implementation → Testing → Maintenance

(C) Planning → Design → Analysis → Implementation → Testing → Maintenance

(D) Planning → Analysis → Implementation → Design → Testing → Maintenance

Detailed Solutions

Q1.

Solution

Concept — Von Neumann Architecture: Proposed by John von Neumann in 1945, this architecture is the blueprint for nearly all modern computers. Its defining characteristic is a **single shared memory** that holds both the program's instructions and the data those instructions manipulate, connected to the CPU via a common bus.

Step 1 — Identify each component's role:

- **CPU**: Executes instructions by fetching them from memory; does not store programs.
- **Control Unit**: Decodes instructions and coordinates CPU operations; stores nothing.
- **Main Memory (RAM)**: Holds the currently running program's code *and* its data simultaneously — the hallmark of Von Neumann design.
- **HDD**: Secondary storage; data must be loaded into RAM before the CPU can process it.

Step 2 — Von Neumann bottleneck: Because both instructions and data share the same memory bus, a performance limitation called the *Von Neumann bottleneck* can arise when CPU speed far outpaces memory bandwidth.

Why other options are wrong:

- Option A: CPU processes instructions; it does not persistently store programs or data.
- Option B: The Control Unit directs operations but holds no stored programs or data.
- Option C: Correct — Main Memory (RAM) stores both program instructions and data.
- Option D: HDD is secondary (auxiliary) storage; it cannot be accessed directly by the CPU.

Final Answer: Main Memory (RAM) stores both instructions and data in Von Neumann architecture $\Rightarrow$ C

Answer: (C) Go Back to Q 1

Q2.

Solution

Concept — Plotters vs. Printers: Both are output devices, but they serve distinct purposes. A **plotter** produces output by moving a pen (or cutting tool) across paper to draw continuous vector graphics. This makes it ideal for large-format technical output requiring precision over large areas.

Step 1 — Primary use of a plotter: Plotters are used in engineering, architecture, GIS mapping, and CAD applications to produce drawings like blueprints, circuit board layouts, and topographic maps — output that would be impractical to print accurately on a standard printer.

Step 2 — Why Option A is correct: Option A correctly identifies the plotter's primary application domain: large-format technical drawings such as engineering blueprints and architectural plans. This is the standard definition used in computer science curricula.

Why other options are wrong:

- Option A: Correct — plotters are used for large-format technical and engineering drawings.
- Option B: Not universally true; modern laser printers can match or exceed plotter resolution for standard page sizes.
- Option C: Plotters use various technologies (pen, inkjet electrostatic); they are not exclusively inkjet.
- Option D: A plotter is definitively an *output* device, producing hard-copy output from digital data.

Final Answer: Plotter is used for large-format technical drawings such as engineering blueprints $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q 2

Q3.

Solution

Concept — Virtual Memory and Paging: Paging is a memory management scheme that eliminates the need for contiguous physical memory allocation. The OS divides:

- **Logical (virtual) memory** into equal-sized units called **pages**.
- **Physical memory** into equal-sized units called **frames** (same size as pages).

A **page table** maps each virtual page number to the physical frame it currently occupies.

Step 1 — Fixed-size blocks: The key word is *fixed*. Unlike segmentation (which uses variable-size blocks), paging always uses uniform page sizes (e.g., 4 KB). This eliminates external fragmentation.

Step 2 — Page table location: The page table is stored in *main memory* (not CPU registers, which are far too few). The CPU has a special register (PTBR — Page Table Base Register) that points to the page table in memory.

Why other options are wrong:

- Option A: Paging uses *fixed*-size blocks; variable-size blocks characterise *segmentation*.
- Option B: The page table is in main memory; only the PTBR (a pointer to it) is in a CPU register.
- Option C: Virtual memory reduces pressure on RAM but absolutely requires some physical RAM to function.
- Option D: Correct — fixed-size pages; page table maps virtual page numbers to physical frame addresses.

Final Answer: Paging uses fixed-size pages; page table maps virtual pages to physical frames $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q 3

Q4.

Solution

Concept — Magnetic Tape and Sequential Access: Magnetic tape is one of the oldest and most cost-effective high-capacity storage media. Its fundamental limitation is that data is recorded linearly along the tape, making access inherently **sequential**: the tape must be physically wound forward or backward to reach any target data.

Step 1 — Sequential vs. direct access:

- **Sequential access** (tape): to read record at position 500, you must pass through records 1–499 (or rewind past them). Average access time depends on the physical location of data.
- **Direct (random) access** (HDD, SSD): the read/write head (or controller) can jump directly to any sector address in near-constant time.

Step 2 — Practical implication: This is why magnetic tape is used only for *backup and archival* (sequential batch access is acceptable) and not for databases or operating system files (which require random access).

Why other options are wrong:

- Option A: Magnetic tape does *not* support random access — that is HDD's advantage.
- Option B: Correct — sequential access; earlier data requires rewinding.
- Option C: Magnetic tape access is far slower than SSDs (milliseconds to minutes vs. microseconds).
- Option D: Magnetic tape is *secondary* (auxiliary) storage, never primary storage.

Final Answer: Magnetic tape is sequential access; reaching earlier data requires rewinding $\Rightarrow$ $\boxed{\text{B}}$

Answer: (B) Go Back to Q 4

Q5.

Solution

Concept — Direct-Mapped Cache: Cache memory has three mapping policies: direct-mapped, fully associative, and set-associative. In **direct-mapped** cache, each main memory block is assigned to **exactly one** cache line, determined by the formula:

$$\text{Cache Line Index} = \text{Block Number} \bmod \text{Number of Cache Lines}$$

Step 1 — Apply to the diagram (2 cache lines):

- Block 0: $0 \bmod 2 = 0 \Rightarrow$ maps to Line 0
- Block 1: $1 \bmod 2 = 1 \Rightarrow$ maps to Line 1
- Block 2: $2 \bmod 2 = 0 \Rightarrow$ maps to Line 0
- Block 3: $3 \bmod 2 = 1 \Rightarrow$ maps to Line 1

This matches the arrows shown in the diagram.

Step 2 — Trade-off: The advantage of direct-mapped cache is its *simple, fast lookup* (no search required). The disadvantage is *conflict misses* when two frequently used blocks compete for the same line.

Why other options are wrong:

- Option A: Correct — each block maps to exactly one cache line via the modulo formula.
- Option B: Describes *fully associative* cache, where any block can go in any line.
- Option C: Only one block can *reside* in a cache line at a time; multiple blocks may map to it, but not simultaneously occupy it.
- Option D: Direct-mapped actually has a *lower* hit ratio than fully associative due to conflict misses.

Final Answer: Direct-mapped cache: each block maps to exactly one line via block $\bmod$ lines $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q 5

Q6.

Solution

Concept — Shortest Job First (SJF) Scheduling: SJF is a non-preemptive CPU scheduling algorithm that always selects the process with the **smallest CPU burst time** next. It is mathematically proven to be **optimal** in minimising average waiting time among all non-preemptive scheduling algorithms.

Step 1 — Calculate average waiting time from the Gantt chart: SJF order: J2 (burst 2) $\rightarrow$ J3 (burst 4) $\rightarrow$ J1 (burst 6).

- Waiting time of J2 $= 0$ (runs first)
- Waiting time of J3 $= 2$ (starts at time 2)
- Waiting time of J1 $= 6$ (starts at time 6)
- Average $= (0 + 2 + 6)/3 = 8/3 \approx 2.67$ units

Step 2 — Convoy effect in FCFS: Under FCFS, if J1 (burst 6) runs first, both J2 and J3 wait unnecessarily — this is the *convoy effect*: short jobs are held up behind a long job.

Why other options are wrong:

- Option A: FCFS does *not* minimise waiting time; it may cause the convoy effect.
- Option B: Round Robin typically generates *more* context switches than SJF or FCFS.
- Option C: Priority scheduling can cause indefinite *starvation* of low-priority processes.

- Option D: Correct — SJF minimises average waiting time; FCFS causes convoy effect.

Final Answer: SJF minimises average waiting time; FCFS may cause the convoy effect $\Rightarrow$ D

Answer: (D) Go Back to Q 6

Q7.

Solution

Concept — Octal to Decimal Conversion: In the octal (base-8) number system, digits range from 0 to 7. To convert an octal number to decimal (base 10), multiply each digit by the corresponding power of 8 (positional value) and sum the products:

$$(d_n d_{n-1} \cdots d_1 d_0)_8 = d_n \times 8^n + \cdots + d_1 \times 8^1 + d_0 \times 8^0$$

Step 1 — Identify digit positions in $(347)_8$:

$$(347)_8 = 3 \times 8^2 + 4 \times 8^1 + 7 \times 8^0$$

Step 2 — Compute each term and sum:

$$= 3 \times 64 + 4 \times 8 + 7 \times 1 = 192 + 32 + 7 = \mathbf{231}$$

Why other options are wrong:

- Option A: 215 corresponds to $(327)_8$: $3 \times 64 + 2 \times 8 + 7 = 192 + 16 + 7 = 215$.
- Option B: 223 would require different digit values; no simple misreading of $(347)_8$ gives 223.
- Option C: Correct — $3 \times 64 + 4 \times 8 + 7 \times 1 = 192 + 32 + 7 = 231$.
- Option D: 247 = $(367)_8$: $3 \times 64 + 6 \times 8 + 7 = 192 + 48 + 7 = 247$; digit 4 was misread as 6.

Final Answer: $(347)_8 = 192 + 32 + 7 = 231 \Rightarrow$ C

Answer: (C) Go Back to Q 7

Q8.

Solution

Concept — De Morgan's Theorems: De Morgan's theorems are two fundamental identities in Boolean algebra that relate the complement of a compound expression to individual complements:

(a) $(A \cdot B)' = A' + B'$ — complement of AND equals OR of complements
(b) $(A + B)' = A' \cdot B'$ — complement of OR equals AND of complements

Step 1 — Apply to the question: The question asks for $(A + B)'$ (complement of an OR). By Theorem 2: $(A + B)' = A' \cdot B'$.

Step 2 — Circuit equivalence (from diagram): An **OR gate followed by a NOT gate** (i.e., a NOR gate) is logically equivalent to **two NOT gates feeding into an AND gate**. This is exactly what the circuit diagram illustrates.

Why other options are wrong:

- Option A: $(A + B)' \neq A' + B'$; if it were, De Morgan would imply $(A + B)' = A + B$ (apply twice), which is false.
- Option B: Correct — $(A + B)' = A' \cdot B'$ is De Morgan's second theorem.
- Option C: $(A \cdot B)' = A' + B'$ by De Morgan's first theorem; $A \cdot B'$ is not a De Morgan result.
- Option D: $A \cdot B \neq (A + B)'$ in general (e.g., for $A = B = 1$: $A \cdot B = 1$, $(A + B)' = 0$).

Final Answer: $(A + B)' = A' \cdot B'$ (De Morgan's second theorem) $\Rightarrow$ B

Answer: (B) Go Back to Q 8

Q9.

Solution

Concept — Singly Linked List: A singly linked list is a dynamic, linear data structure in which elements (called *nodes*) are connected through pointers. Each node consists of exactly two fields: a **data** field (holding the actual value) and a **next pointer** (holding the memory address of the next node). The last node's next pointer is set to `NULL`, signalling the end of the list.

Step 1 — Node structure in C:

```
struct Node { int data; struct Node* next; };
```

The `next` pointer of the last node is `NULL`.

Step 2 — Access pattern: Traversal is inherently *sequential* starting from the head. Accessing the k-th element requires following k pointers: $O(k)$ time — *not* $O(1)$ random access. Nodes are scattered in heap memory, not contiguous.

Why other options are wrong:

- Option A: Correct — data field + next pointer; last node's next is `NULL`.
- Option B: Describes a *doubly* linked list, which has both `prev` and `next` pointers.
- Option C: Linked lists require $O(n)$ traversal to access any element; only arrays offer $O(1)$ random access.
- Option D: Linked list nodes are stored at *non-contiguous* (arbitrary) memory addresses; contiguous storage describes arrays.

Final Answer: Each node = data + next pointer; last node's next = `NULL` $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q 9

Q10.

Solution

Concept — Circular Queue: A circular queue is an improved array-based implementation of a queue. In a standard (linear) queue, once the rear pointer reaches the last array index, the queue reports "full" even if dequeue operations have freed space at the front. A circular queue eliminates this by treating the array as a **ring**: the rear pointer wraps around to index 0 after reaching the last index.

Step 1 — Wraparound formula: On each enqueue, the new rear is computed as:

$$\text{rear} = (\text{rear} + 1) \bmod \text{capacity}$$

Step 2 — Full vs. empty condition: The queue is full when $(\text{rear} + 1) \bmod \text{capacity} = \text{front}$, and empty when $\text{rear} = \text{front}$. This makes full use of all allocated array slots.

Why other options are wrong:

- Option A: Circular queue uses memory *more* efficiently by reusing vacated front slots.
- Option B: No internal stack is used; the wraparound modulo arithmetic handles all cases.

- Option C: Circular queue fully supports dequeue; the front pointer similarly wraps around.
- Option D: Correct — rear pointer wraps via modulo; prevents false "queue full" condition.

Final Answer: Rear pointer wraps around to index 0, preventing false "queue full" $\Rightarrow$ D

Answer: (D) Go Back to Q 10

Q11.

Solution

Concept — OSI Model and Application Layer: The OSI (Open Systems Interconnection) model defines 7 layers of network communication. The topmost layer is the **Application Layer (Layer 7)**, which provides network services directly to end-user applications. Protocols that operate here include HTTP, HTTPS, FTP, SMTP, DNS, Telnet, and SSH.

Step 1 — OSI layer summary (relevant layers):

- Layer 7 — Application: HTTP, HTTPS, FTP, SMTP, DNS
- Layer 5 — Session: NetBIOS, RPC, PPTP
- Layer 4 — Transport: TCP, UDP
- Layer 3 — Network: IP, ICMP, ARP

Step 2 — HTTP's role: HTTP defines how web clients (browsers) request resources from web servers and how those servers respond. This is purely an application-level concern; the lower layers handle routing, reliability, and framing transparently.

Why other options are wrong:

- Option A: Transport Layer (Layer 4) handles TCP/UDP end-to-end communication, not HTTP.
- Option B: Network Layer (Layer 3) handles logical addressing and routing via IP.
- Option C: Correct — HTTP operates at the Application Layer (Layer 7).
- Option D: Session Layer (Layer 5) manages session establishment, maintenance, and termination.

Final Answer: HTTP operates at the Application Layer (Layer 7) of the OSI model $\Rightarrow$ C

Answer: (C) Go Back to Q 11

Q12.

Solution

Concept — Domain Name System (DNS): DNS is often called the "phone book of the Internet." Its core function is **name resolution**: converting human-readable domain names (e.g., `www.collegedunia.com`) into machine-readable **IP addresses** (e.g., `203.0.113.45`) that routers use to forward packets to the correct destination.

Step 1 — DNS resolution chain:

(a) Browser queries the local DNS resolver (usually ISP-provided or a public DNS like 8.8.8.8).
(b) Resolver queries a *root nameserver* → *TLD nameserver* (e.g., `.com`) → *authoritative nameserver* for the domain.
(c) The authoritative server returns the IP address; the resolver caches it for the TTL period.

Step 2 — Distinguish DNS from DHCP: DHCP (Dynamic Host Configuration Protocol) dynamically *assigns* IP addresses to hosts. DNS *translates* domain names; it does not assign addresses.

Why other options are wrong:

- Option A: Dynamic IP assignment is DHCP's function, not DNS.
- Option B: Correct — DNS translates domain names to IP addresses.
- Option C: Email routing is handled by SMTP using MX records, not DNS directly.
- Option D: DNS does not encrypt traffic; that is the role of TLS/SSL (HTTPS).

Final Answer: DNS translates human-readable domain names to IP addresses ⇒ B

Answer: (B) Go Back to Q 12

Q13.

Solution

Concept — SQL COUNT Function: `COUNT` is an aggregate function in SQL. Its behaviour depends on what argument is passed:

- `COUNT(*)` — counts **every row** in the result set, including rows that have `NULL` in any column.
- `COUNT(column_name)` — counts only rows where the specified column value is **not** `NULL`.

Step 1 — Concrete example: Suppose a table has 5 rows and the column `phone` has `NULL` in 2 rows:

- `SELECT COUNT(*) FROM students;` → returns **5**
- `SELECT COUNT(phone) FROM students;` → returns **3**

Step 2 — Practical use: Use `COUNT(*)` to measure table size; use `COUNT(col)` to count how many records actually have a value for that column.

Why other options are wrong:

- Option A: Correct — `COUNT(*)` counts all rows; `COUNT(col)` excludes `NULL`s.
- Option B: This reverses the actual behaviour of the two forms.
- Option C: `COUNT(*)` includes `NULL` rows; `COUNT(col)` does not.
- Option D: They return different results whenever the column contains `NULL` values.

Final Answer: `COUNT(*)` counts all rows; `COUNT(column)` excludes `NULL` values ⇒ A

Answer: (A) Go Back to Q 13

Q14.

Solution

Concept — Second Normal Form (2NF): Database normalisation reduces data redundancy and update anomalies. A relation is in **2NF** if and only if it satisfies two conditions:

(a) It is in **1NF** (all attributes are atomic; no repeating groups).
(b) Every non-prime (non-key) attribute is **fully functionally dependent** on the *entire* primary key — no attribute may depend on only a *part* of a composite

key (no partial dependency).

Step 1 — What is a partial dependency? A partial dependency exists only when the primary key is *composite* (multi-attribute). Example: if PK = {StudentID, CourseID} and `StudentName` depends only on `StudentID` (not on `CourseID`), that is a partial dependency violating 2NF.

Step 2 — 2NF vs. 3NF: Eliminating *transitive* dependencies (non-key attribute depending on another non-key attribute) defines **3NF**, not 2NF. These are often confused.

Why other options are wrong:

- Option A: Eliminating transitive dependencies defines 3NF, not 2NF.
- Option B: Vague and incomplete; 2NF specifically bans *partial* dependencies on the whole key.
- Option C: 2NF *forbids* partial dependencies; it does not allow them under any condition.
- Option D: Correct — 1NF plus no partial dependency on the primary key.

Final Answer: 2NF = 1NF + no partial functional dependency on the entire primary key $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q 14

Q15.

Solution

Concept — ER Model Cardinality: In Entity-Relationship modelling, **cardinality** specifies how many instances of one entity set can be associated with instances of another entity set through a given relationship. The four standard cardinality types are:

Step 1 — Cardinality types:

- **One-to-One (1:1)**: Each entity in A is associated with at most one entity in B, and vice versa. Example: each employee has exactly one company-issued laptop.
- **One-to-Many (1:N)**: One entity in A can be associated with many in B (e.g., one department has many employees).
- **Many-to-One (N:1)**: Many in A relate to one in B.
- **Many-to-Many (M:N)**: Many in A relate to many in B (e.g., students and courses).

Step 2 — Match to the question: The question specifies "one entity in A is related to exactly one entity in B" — this is the definition of a **One-to-One (1:1)** relationship.

Why other options are wrong:

- Option A: Describes One-to-Many (1:N).
- Option B: Describes Many-to-Many (M:N).
- Option C: Correct — One-to-One (1:1): each entity in A relates to exactly one in B.
- Option D: Describes Many-to-One (N:1).

Final Answer: One entity in A relates to exactly one entity in B = One-to-One relationship ⇒ C

Answer: (C) Go Back to Q 15

Q16.

Solution

Concept — Loop Control in C: C provides three loop constructs, classified by when the loop condition is tested:

- **Entry-controlled** (pre-test): `for` and `while` — condition checked *before* each iteration.
- **Exit-controlled** (post-test): `do-while` — condition checked *after* each iteration.

Step 1 — `for` loop execution flow:

(a) Initialisation (runs once).
(b) **Condition check**: if `false`, skip body entirely and exit.
(c) Body executes.
(d) Update expression.
(e) Repeat from step 2.

If the condition is false at step 2 on the *first* evaluation, the body **never executes**.

Step 2 — Contrast with `do-while`: In `do-while`, the body always executes at least once because the condition is checked *after* the body.

Why other options are wrong:

- Option A: "Always executes at least once" describes `do-while`, not `for`.

- Option B: Correct — `for` is entry-controlled; body skipped if initial condition is false.
- Option C: Only `do-while` is exit-controlled; `for` is entry-controlled.
- Option D: `for(;;) {}` or `for(int i=0; 1; i++) {}` creates an infinite `for` loop.

Final Answer: `for` is entry-controlled; body does not execute if condition is initially false $\Rightarrow$ B

Answer: (B) Go Back to Q 16

Q17.

Solution

Concept — Bubble Sort Time Complexity: Bubble Sort is a simple comparison-based sorting algorithm. In each pass, it compares adjacent elements and swaps them if they are out of order. After k passes, the k largest elements are correctly positioned at the end.

Step 1 — Count comparisons in the worst case: The worst case is a reverse-sorted array. The outer loop runs $n-1$ times; in the i-th pass, the inner loop makes $n-1-i$ comparisons. Total comparisons:

$$\sum_{i=0}^{n-2}(n-1-i) = (n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2} = O(n^2)$$

Step 2 — Best case (optimised Bubble Sort): With an early-exit flag that stops if no swap occurred in a pass, the best case (already sorted array) is $O(n)$.

Why other options are wrong:

- Option A: Correct — worst-case $O(n^2)$ due to $\frac{n(n-1)}{2}$ comparisons.
- Option B: $O(n \log n)$ is the complexity of efficient sorts like Merge Sort, Quick Sort (average), and Heap Sort.
- Option C: $O(n)$ is the *best-case* complexity of Bubble Sort (optimised), not the worst case.
- Option D: $O(\log n)$ is the complexity of binary search; no general-purpose sort achieves this.

Final Answer: Bubble Sort worst-case time complexity $= O(n^2) \Rightarrow$ A

Answer: (A) Go Back to Q 17

Q18.

Solution

Concept — Standard Flowchart Symbols: Flowcharts use standardised geometric symbols to represent different types of operations. Each symbol has a specific meaning:

Step 1 — Standard symbol definitions:

- **Oval/Ellipse** — Terminal: marks the *Start* or *End* of the flowchart.
- **Rectangle** — Process: represents a computation, calculation, or action step.
- **Diamond** — Decision: represents a Yes/No or True/False branching point (conditional).
- **Parallelogram** — Input/Output: represents reading input from a user/device or writing output to screen/file.

Step 2 — Identify the I/O symbol: The **parallelogram** (a four-sided shape with slanted sides) is the universal flowchart symbol for any input or output operation — such as reading a value from the keyboard or printing a result to the screen.

Why other options are wrong:

- Option A: Rectangle represents a *Process* (computation or assignment), not I/O.
- Option B: Diamond represents a *Decision* (conditional branch with two or more paths).
- Option C: Oval/Ellipse represents a *Terminal* (Start or End of program).
- Option D: Correct — Parallelogram represents Input/Output operations.

Final Answer: Parallelogram is the standard flowchart symbol for Input/Output operations $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q 18

Q19.

Solution

Concept — Directory Structures in Operating Systems: OS file management systems use different directory structures to organise files. The three common types are:

- **Single-level**: All files in one directory — simple but impractical for multiple users; no duplicate filenames allowed.

- **Two-level**: One directory per user; no subdirectory nesting; no cross-user name conflicts.
- **Hierarchical (tree)**: Directories contain files *and* subdirectories, forming a tree. Used by all modern OSes (Windows, Linux, macOS).

Step 1 — Key advantage of hierarchical structure: The tree structure allows the same filename to exist in *different* branches of the directory tree. For example, `/home/alice/report.txt` and `/home/bob/report.txt` are two entirely different files. Users and projects are isolated in their own subtrees.

Step 2 — Why this matters: Without hierarchical directories, every file in the system would need a globally unique name — impractical for large multi-user systems.

Why other options are wrong:

- Option A: Hierarchical structure *does* allow the same filename in different directories; that is its advantage.
- Option B: Single-level is the *worst* choice for multi-user systems (no isolation, no duplicate names).
- Option C: Correct — same filename permitted in different directories; prevents naming conflicts.
- Option D: Two-level supports only two levels; hierarchical (tree) supports unlimited nesting.

Final Answer: Hierarchical directory structure allows identical filenames in different directories $\Rightarrow$ $\boxed{\text{C}}$

Answer: (C) Go Back to Q 19

Q20.

Solution

Concept — Optical Storage and Laser Wavelength: The storage capacity of an optical disc is fundamentally determined by how small the laser can make the data pits (bumps and lands) on the disc surface. A **shorter laser wavelength** allows focusing the beam to a smaller spot, producing smaller pits and thus packing more data per unit area.

Step 1 — Laser wavelengths and capacities:

- **CD**: Infrared laser ≈ 780 nm; capacity ≈ 700 MB
- **DVD**: Red laser ≈ 650 nm; capacity ≈ 4.7 GB

- **Blu-ray**: Blue-violet laser ≈ 405 nm; capacity ≈ 25–50 GB

Step 2 — Why shorter wavelength = more capacity: By the diffraction limit, the minimum focused spot size is proportional to the wavelength. Shorter wavelength $\Rightarrow$ smaller spot $\Rightarrow$ smaller pits $\Rightarrow$ higher data density on the same disc area.

Why other options are wrong:

- Option A: Blu-ray uses a *shorter* wavelength (405 nm), not longer. Longer would mean fewer data pits.
- Option B: Correct — Blu-ray's 405 nm blue-violet laser enables smaller pits and higher density than DVD's 650 nm.
- Option C: The wavelengths are significantly different; this is the primary engineering reason for the capacity difference.
- Option D: Blu-ray stores *more* data than DVD (25+ GB vs. 4.7 GB), not less.

Final Answer: Blu-ray uses shorter blue-violet laser ($\approx$405 nm vs. DVD's 650 nm) $\Rightarrow$ B

Answer: (B) Go Back to Q 20

Q21.

Solution

Concept — TCP vs. UDP (Transport Layer Protocols): Both TCP and UDP are Transport Layer (Layer 4) protocols. They represent two fundamentally different design philosophies:

Step 1 — TCP characteristics:

- **Connection-oriented**: establishes a connection via a 3-way handshake (SYN $\rightarrow$ SYN-ACK $\rightarrow$ ACK).
- **Reliable**: uses acknowledgements (ACKs), sequence numbers, and retransmission of lost segments.
- **Flow & congestion control**: prevents overwhelming the receiver or network.
- **Use cases**: HTTP/HTTPS, FTP, SMTP, SSH (where data integrity matters).

Step 2 — UDP characteristics:

- **Connectionless**: no handshake; datagrams sent immediately.
- **Unreliable**: no ACKs, no retransmissions, no ordering guarantee.

- **Low overhead**: faster, lower latency.
- **Use cases**: DNS, video streaming, VoIP, online gaming (where speed matters more than perfect delivery).

Why other options are wrong:

- Option A: Correct — TCP is reliable and connection-oriented; UDP is fast but connectionless and unreliable.
- Option B: This reverses the roles; UDP is unreliable, TCP is reliable.
- Option C: TCP's reliability *is* its acknowledgement mechanism — this statement is factually wrong.
- Option D: Only TCP is connection-oriented; UDP is explicitly connectionless.

Final Answer: TCP = reliable, connection-oriented; UDP = connectionless, faster, unreliable $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q 21

Q22.

Solution

Concept — Binary Tree Classification: Binary trees are classified based on the structural properties of their nodes:

- **Full Binary Tree**: every node has either **0 or 2 children** (no node has exactly 1 child).
- **Complete Binary Tree**: all levels are fully filled except possibly the last, which is filled from *left to right*.
- **Perfect Binary Tree**: all internal nodes have exactly 2 children, and all leaf nodes are at the same level.
- **Skewed Binary Tree**: every node has at most 1 child (all left or all right).

Step 1 — Analyse the given tree (7 nodes):

- Node 1 (root): 2 children (nodes 2 and 3) — satisfies "0 or 2".
- Node 2: 2 children (nodes 4 and 5) — satisfies "0 or 2".
- Node 3: 2 children (nodes 6 and 7) — satisfies "0 or 2".
- Nodes 4, 5, 6, 7 (leaves): 0 children — satisfies "0 or 2".

Every node has 0 or 2 children $\Rightarrow$ this is a **Full Binary Tree**.

Step 2 — Note: This specific 7-node tree happens to also be perfect and complete,

but the property *asked about* (0 or 2 children) corresponds specifically to "Full Binary Tree."

Why other options are wrong:

- Option A: Complete Binary Tree does not require every node to have 0 or 2 children.
- Option B: Perfect Binary Tree is a specific sub-case; the general definition requested is Full.
- Option C: Skewed trees have nodes with only one child; opposite of what is shown.
- Option D: Correct — Full Binary Tree: every node has 0 or 2 children.

Final Answer: A tree where every node has 0 or 2 children is a **Full Binary Tree** $\Rightarrow$ D

Answer: (D) Go Back to Q 22

Q23.

Solution

Concept — Hexadecimal to Decimal Conversion: Hexadecimal (base 16) uses digits 0–9 and letters A–F, where A = 10, B = 11, C = 12, D = 13, E = 14, F = 15. To convert to decimal, multiply each digit by the corresponding power of 16 and sum:

$$(d_n \cdots d_1 d_0)_{16} = d_n \times 16^n + \cdots + d_1 \times 16^1 + d_0 \times 16^0$$

Step 1 — Apply conversion to $(FF)_{16}$:

$$(FF)_{16} = F \times 16^1 + F \times 16^0 = 15 \times 16 + 15 \times 1$$

Step 2 — Compute:

$$= 240 + 15 = \mathbf{255}$$

Why other options are wrong:

- Option A: $240 = 15 \times 16 + 0 \times 1 = (F0)_{16}$; the second digit was incorrectly taken as 0.
- Option B: $250 = 15 \times 16 + 10 \times 1 = (FA)_{16}$; the second digit was incorrectly taken as A=10.
- Option C: Correct — $(FF)_{16} = 240 + 15 = 255$.
- Option D: $256 = 16^2 = (100)_{16}$; $(FF)_{16}$ is exactly one less than 256, i.e., 255.

Final Answer: $(FF)_{16} = 15 \times 16 + 15 \times 1 = 240 + 15 = 255 \Rightarrow$ C

Answer: (C) Go Back to Q 23

Q24.

Solution

Concept — Email Protocols: Electronic mail uses different protocols for sending and receiving, each optimised for its specific task:

Step 1 — Protocol roles:

- **SMTP** (Simple Mail Transfer Protocol): Used for *sending* email from an email client to a mail server ("mail submission") and for routing email between mail servers. Default port: 25 (server-to-server) or 587 (client submission).
- **POP3** (Post Office Protocol version 3): Downloads email from the server to the local client, typically deleting it from the server. Port: 110. Simple, offline-oriented.
- **IMAP** (Internet Message Access Protocol): Accesses email stored on the server without permanently downloading it; keeps messages synchronised across multiple devices. Port: 143. Modern standard.

Step 2 — Memory aid: "**S**end **M**ail **T**o **P**eers" = SMTP for sending; POP3/IMAP for retrieving.

Why other options are wrong:

- Option A: POP3 is for *receiving*, not sending; SMTP is for sending, not receiving.
- Option B: Correct — SMTP sends; POP3 and IMAP receive.
- Option C: IMAP is for receiving, not sending.
- Option D: HTTP is for web browsing (HyperText), not email transmission.

Final Answer: SMTP sends email; POP3 and IMAP receive email $\Rightarrow$ B

Answer: (B) Go Back to Q 24

Q25.

Solution

Concept — Software Development Life Cycle (SDLC): The SDLC is a structured framework that defines the phases required to plan, build, test, and deliver software systems. Each phase has specific inputs, outputs, and deliverables, and they must follow a logical order.

Step 1 — Correct phase order and rationale:

(a) **Planning**: Define project scope, feasibility, resources, schedule, and cost estimate.
(b) **Analysis**: Gather and document stakeholder requirements (what the system must do).
(c) **Design**: Translate requirements into system architecture, database schema, and UI design (how the system will do it).
(d) **Implementation**: Write, integrate, and document the actual source code.
(e) **Testing**: Verify correctness (unit, integration, system, acceptance tests); identify and fix defects.
(f) **Maintenance**: Deploy to production; monitor, update, patch, and enhance post-release.

Step 2 — Key dependencies: Analysis *must* precede Design (you cannot design without knowing requirements). Design *must* precede Implementation (you cannot code without a blueprint). Testing *must* follow Implementation.

Why other options are wrong:

- Option A: Correct — the standard SDLC order.
- Option B: Putting Analysis before Planning is illogical; feasibility must be assessed first.
- Option C: Putting Design before Analysis is illogical; requirements must be understood before designing.
- Option D: Putting Implementation before Design violates the blueprint-before-construction principle.

Final Answer: Planning $\rightarrow$ Analysis $\rightarrow$ Design $\rightarrow$ Implementation $\rightarrow$ Testing $\rightarrow$ Maintenance $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q 25

Answer Key

Q	Ans	Q	Ans	Q	Ans	Q	Ans	Q	Ans
1	C	2	A	3	D	4	B	5	A
6	D	7	C	8	B	9	A	10	D
11	C	12	B	13	A	14	D	15	C
16	B	17	A	18	D	19	C	20	B
21	A	22	D	23	C	24	B	25	A