GATE Computer Science and Information Technology

Sample Paper – 5

Duration: 128 Minutes **Maximum Marks: 72**

Instructions

- This paper contains **46 questions** covering the core **Computer Science and Information Technology** subject of the GATE paper conducted by the IITs and IISc (General Aptitude and Engineering Mathematics are provided as separate papers).
- **Q1–Q20 carry 1 mark each** and **Q21–Q46 carry 2 marks each**, for a total of **72 marks**.
- Each question carries **negative marking of one-third of its allotted marks**: a wrong 1-mark question costs **0.33** and a wrong 2-mark question costs **0.67**. Unattempted questions carry no penalty.
- Every question here is a single-correct **MCQ**. The real GATE paper also has Multiple Select (MSQ) and Numerical Answer Type (NAT) questions, and **those carry no negative marking**.
- Attempt this paper in one timed sitting of about **128 minutes**, the share this core subject earns at GATE's overall pace of 180 minutes for 65 questions. Unless stated otherwise, use $\log$ to base 2 for asymptotic work and standard byte $= 8$ bits.

Digital Logic & Computer Organization

Q1. An integer is stored in a 6-bit register using two's complement representation. The largest positive integer that this register can hold is: [1 mark]

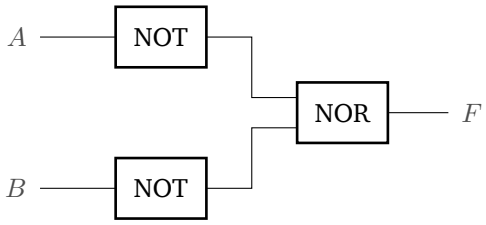
(A) 63

(B) 31

(C) 32

(D) 64

Q2. In the circuit below, inputs A and B are each passed through a NOT gate, and the two inverted signals feed a single NOR gate. The Boolean output F simplifies to: **[1 mark]**



(A) $A + B$

(B) $A \cdot B$

(C) $A \oplus B$

(D) $\overline{A + B}$

Q3. A four-variable Boolean function $F(A, B, C, D)$ is plotted on the Karnaugh map shown below (a blank cell is a 0). The minimal sum-of-products expression for F is: **[1 mark]**

AB \ CD	00	01	11	10
00			1	1
01			1	1
11			1	1
10			1	1

(A) $\overline{C}$

(B) $C\,D$

(C) A

(D) C

Q4. A 4-to-16 line decoder is to be built using only 2-to-4 line decoders that have an enable input. The minimum number of 2-to-4 decoders required is: **[1 mark]**

(A) 4

(B) 8

(C) 6

(D) 5

Q5. A 4-bit asynchronous (ripple) counter is built from 4 toggle flip-flops and is allowed to run freely through its full natural sequence. The number of distinct states (the modulus) of this counter is: **[1 mark]**

(A) 4

(B) 8

(C) 16

(D) 32

Q6. A ring counter is constructed from 4 D flip-flops connected in a loop and is initialised with the pattern 1000. As it is clocked, the number of distinct states it cycles through before the pattern repeats is: **[1 mark]**

(A) 2

(B) 4

(C) 8

(D) 16

Q7. In a certain addressing mode, the address field of the instruction names a register, and the *contents* of that register are taken as the memory address of the operand. This addressing mode is called: **[1 mark]**

(A) immediate addressing

(B) direct (absolute) addressing

(C) register-indirect addressing

(D) indexed addressing

Q8. A certain computation takes 8 ns on a non-pipelined processor. The same computation is split into 4 balanced pipeline stages of 2 ns each, and

register overhead is negligible. For a very large stream of independent computations, the maximum speedup achieved by pipelining is: [1 mark]

(A) 2

(B) 8

(C) 4

(D) 16

Q9. A direct-mapped cache has 64 lines. Main memory is divided into 4096 blocks, each exactly the size of one cache line. In a main-memory address, the number of bits in the tag field is: [1 mark]

(A) 12

(B) 6

(C) 18

(D) 8

Programming, Data Structures & Algorithms

Q10. Consider the following C code fragment:

```
int x = 12, y = 10;  printf("%d", x & y);
```

Here `&` is the bitwise-AND operator. The value printed is: [1 mark]

(A) 14

(B) 2

(C) 8

(D) 10

Q11. The postfix (reverse-Polish) expression

$$5\ \ 3\ +\ \ 2\ *$$

is evaluated using a stack. The final value left on the stack is: [1 mark]

(A) 13

(B) 16

(C) 25

(D) 11

Q12. A first-in-first-out queue is to be simulated using only the standard stack operations (push and pop). The minimum number of stacks required to implement one such queue is: [1 mark]

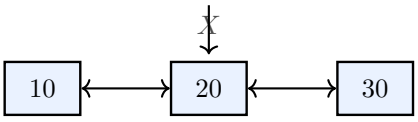
(A) 2

(B) 1

(C) 3

(D) 4

Q13. In the doubly linked list shown below, a pointer X is already positioned on a given node. Deleting that node from the list (adjusting the neighbours' pointers) takes worst-case time: [1 mark]



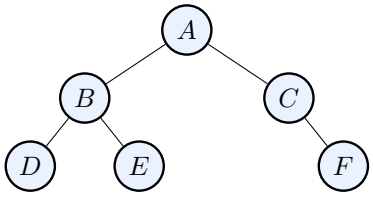
(A) $O(n)$

(B) $O(\log n)$

(C) $O(1)$

(D) $O(n^2)$

Q14. For the binary tree shown below, the sequence produced by a *postorder* traversal is: [1 mark]



(A) $D\ E\ B\ F\ C\ A$

(B) $A\ B\ D\ E\ C\ F$

(C) $D\ B\ E\ A\ C\ F$

(D) $A\ B\ C\ E\ D\ F$

Q15. A binary heap containing n elements is stored in an array using the usual parent-child index rules. The number of *leaf* nodes in this heap is: **[1 mark]**

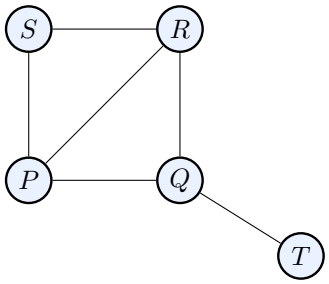
(A) $\lfloor n/2 \rfloor$

(B) $n/2$

(C) $\lceil n/2 \rceil$

(D) $\log_2 n$

Q16. For the simple undirected graph shown below, the total number of edges is: **[1 mark]**



(A) 6

(B) 5

(C) 7

(D) 8

Q17. The running time of an algorithm is $f(n) = 3n^2 + 5n \log_2 n + 100$. The tightest asymptotic (Θ) bound for $f(n)$ is: **[1 mark]**

(A) $\Theta(n^2)$

(B) $\Theta(n \log n)$

(C) $\Theta(n^3)$

(D) $\Theta(n)$

Q18. A divide-and-conquer algorithm has the running-time recurrence

$$T(n) = T\left(\tfrac{n}{2}\right) + 1, \qquad T(1) = 1.$$

The asymptotic solution of this recurrence is: [1 mark]

(A) $\Theta(n)$

(B) $\Theta(n \log n)$

(C) $\Theta(\log n)$

(D) $\Theta(1)$

Q19. Merge sort is applied to an array of n elements. Its *worst-case* time complexity is: [1 mark]

(A) $\Theta(n^2)$

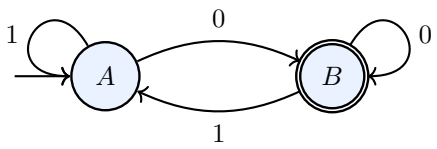
(B) $\Theta(n)$

(C) $\Theta(\log n)$

(D) $\Theta(n \log n)$

Theory of Computation & Compiler Design

Q20. The deterministic finite automaton (DFA) over the alphabet $\{0, 1\}$ is shown below; B (double circle) is the only accepting state and A is the start state. The language accepted by this DFA is the set of all strings that: [1 mark]



(A) end with the symbol 1

(B) end with the symbol 0

(C) contain the substring 00

(D) have even length

Q21. Over the alphabet $\{a, b\}$, the regular expression

$$a^*\, b^*$$

describes exactly the set of all strings that: **[2 marks]**

(A) are any string over $\{a, b\}$

(B) have an equal number of a's and b's

(C) end with the symbol b

(D) consist of zero or more a's followed by zero or more b's (no a occurs after any b)

Q22. Consider the language $L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$ over the alphabet $\{a, b, c\}$. In the Chomsky hierarchy, this language is: **[2 marks]**

(A) regular

(B) context-free but not regular

(C) not recursively enumerable

(D) context-sensitive but not context-free

Q23. Which one of the following decision problems is *decidable*? **[2 marks]**

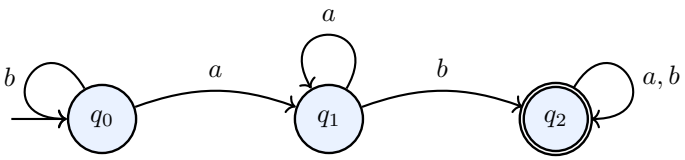
(A) Given a DFA M and a string w, does M accept w?

(B) Given an arbitrary Turing machine M and input w, does M halt on w?

(C) Given a Turing machine M, is $L(M) = \varnothing$?

(D) Given two context-free grammars G_1 and G_2, is $L(G_1) = L(G_2)$?

Q24. The DFA below over $\{a, b\}$ accepts exactly those strings that contain the substring ab (state q_2 is accepting). The minimum number of states in any DFA that recognises "all strings over $\{a, b\}$ containing ab as a substring" is: **[2 marks]**



(A) 2

(B) 3

(C) 4

(D) 5

Q25. In a classical compiler, the phase that checks whether every identifier is declared before use and whether operands have compatible types (type checking) is the: **[2 marks]**

(A) lexical analyzer (scanner)

(B) syntax analyzer (parser)

(C) semantic analyzer

(D) code generator

Q26. Consider the grammar with start symbol S:

$$S \to A\,B, \qquad A \to a \mid \varepsilon, \qquad B \to b.$$

The set $\mathrm{FOLLOW}(A)$ is: **[2 marks]**

(A) $\{a\}$

(B) $\{b\}$

(C) $\{a,\ b\}$

(D) $\{\$\}$

Q27. Eliminating left recursion and performing left factoring are grammar transformations carried out mainly so that the grammar becomes suitable for which parsing technique? **[2 marks]**

(A) LR (bottom-up) parsing

(B) LALR parsing

(C) top-down predictive LL(1) parsing

(D) operator-precedence parsing

Q28. During code optimization, replacing the multiplication $x \times 2$ by the cheaper operation $x + x$ (or a one-bit left shift) is an example of: **[2 marks]**

(A) strength reduction

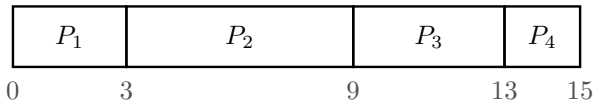
(B) constant folding

(C) common subexpression elimination

(D) dead-code elimination

Operating Systems & Databases

Q29. Four processes arrive together at time 0 with CPU burst times $P_1 = 3$, $P_2 = 6$, $P_3 = 4$ and $P_4 = 2$ (all in ms). They are scheduled by *First-Come-First-Served* (FCFS) in the order P_1, P_2, P_3, P_4, whose Gantt chart is shown. The average waiting time of the four processes is: **[2 marks]**



(A) 6.25 ms

(B) 5.5 ms

(C) 7.75 ms

(D) 4.75 ms

Q30. A counting semaphore S is initialised to 10 (it guards 10 identical resource units). Six processes each successfully perform one `wait`(S) (acquiring one unit), after which two `signal`(S) operations release two units. The current value of S is: **[2 marks]**

(A) 2

(B) 4

(C) 8

(D) 6

Q31. To *prevent* deadlock, an operating system wants to make the "circular wait" condition impossible. Which one of the following techniques directly eliminates the circular-wait condition? [2 marks]

(A) impose a total ordering on all resource types and require each process to request resources only in increasing order of that numbering

(B) run the banker's algorithm before granting every allocation request

(C) allow the system to forcibly preempt and roll back resources held by a process

(D) require every process to request all of its resources at once before it starts

Q32. A paging system uses 32-bit virtual addresses and a page size of 4 KB. For a single-level page table, the number of entries the page table must have is: [2 marks]

(A) 2^{12}

(B) 2^{32}

(C) 2^{10}

(D) 2^{20}

Q33. A demand-paging system has 3 page frames, all initially empty, and uses the *LRU* (least-recently-used) page-replacement policy. For the page-reference string shown below, the total number of page faults is:

ref:	1	2	3	4	1	2	5	1	2

[2 marks]

(A) 6

(B) 7

(C) 8

(D) 9

Q34. In contiguous file allocation, each file occupies a set of consecutive disk blocks. After many files have been created and deleted, the main disadvantage that this scheme suffers from is: [2 marks]

(A) internal fragmentation within every block

(B) slow sequential (in-order) access to a file

(C) the need for very large file-allocation tables

(D) external fragmentation of free disk space

Q35. In relational algebra, the unary operation that returns a chosen subset of the *attributes* (columns) of a relation and removes duplicate rows from the result is: [2 marks]

(A) selection (σ)

(B) projection (π)

(C) rename (ρ)

(D) Cartesian product ($\times$)

Q36. A relation schema R is in Second Normal Form (2NF) if it is already in 1NF and, in addition: [2 marks]

(A) no non-prime attribute is partially dependent on any candidate key

(B) every determinant of a functional dependency is a superkey of R

(C) no non-prime attribute is transitively dependent on a candidate key

(D) every attribute value is atomic (indivisible)

Q37. Among the ACID properties of a database transaction, the property guaranteeing that the concurrent execution of several transactions leaves the database in a state equivalent to *some* serial execution of those transactions is: [2 marks]

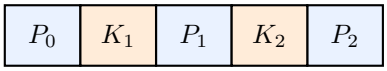
(A) atomicity

(B) durability

(C) isolation

(D) consistency

Q38. A B-tree of order m has internal nodes as illustrated below (child pointers separated by keys). Excluding the root, every internal node of a B-tree of order m must hold at least how many child pointers? **[2 marks]**



a node with 3 child pointers and 2 keys

(A) $\lceil m/2 \rceil$

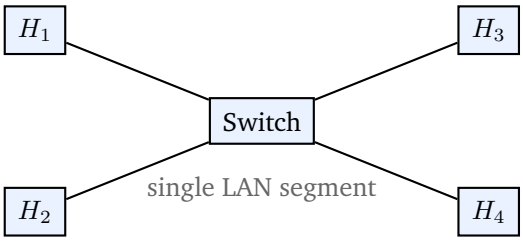
(B) $m - 1$

(C) $m/2$

(D) 2

Computer Networks

Q39. In the local network shown below, four hosts are joined through a single device that forwards frames by reading their destination MAC addresses. Considering the OSI reference model, such a switch operates primarily at the: **[2 marks]**



(A) physical layer (layer 1)

(B) network layer (layer 3)

(C) data-link layer (layer 2)

(D) transport layer (layer 4)

Q40. A single-error-correcting Hamming code is to be designed for a data word of 4 bits. The minimum number of parity (redundant) check bits that must be added is: **[2 marks]**

(A) 1

(B) 2

(C) 4

(D) 3

Q41. A Selective-Repeat sliding-window protocol uses an n-bit field for sequence numbers, giving 2^n distinct sequence numbers. To avoid ambiguity between new and retransmitted frames, the maximum allowable sender window size is: **[2 marks]**

(A) $2^n - 1$

(B) 2^{n-1}

(C) 2^n

(D) $2^{n-1} - 1$

Q42. In an Ethernet (IEEE 802.3) LAN, every network interface card is assigned a hardware (MAC) address. The length of a standard Ethernet MAC address is: **[2 marks]**

(A) 32 bits

(B) 16 bits

(C) 64 bits

(D) 48 bits

Q43. The network $172.16.0.0/16$ is divided into equal-sized subnets using the mask $/20$. The number of subnets created is: **[2 marks]**

(A) 4

(B) 8

(C) 32

(D) 16

Q44. An IPv4 network uses a prefix length of /29. The number of *usable* host addresses available in this subnet is: **[2 marks]**

(A) 6

(B) 8

(C) 14

(D) 2

Q45. At the transport layer of the TCP/IP suite, which protocol is connectionless, offers no reliability or ordering guarantees, and is favoured for its low overhead by applications such as DNS and live streaming? **[2 marks]**

(A) TCP

(B) IP

(C) UDP

(D) SCTP

Q46. When a host resolves a domain name to an IP address, it contacts a name server. The well-known port number that the Domain Name System (DNS) uses is: **[2 marks]**

(A) 25

(B) 53

(C) 80

(D) 110

Detailed Solutions

Q1.

Solution

Concept – range of an n-bit two's complement register: An n-bit two's complement register represents integers from -2^{n-1} to $+2^{n-1}-1$.

Step 1 – note the register width: $n = 6$ bits.

Step 2 – write the positive limit formula: largest positive value $= 2^{n-1}-1$.

Step 3 – substitute $n = 6$: $2^{6-1}-1 = 2^5-1$

Step 4 – evaluate: $2^5 = 32$

$32 - 1 = 31$

Why the other options are wrong:

- A, 63: is 2^6-1, the maximum for *unsigned* 6-bit, not two's complement.
- C, 32: is 2^5; the reachable maximum is one less because a code is spent on $+0$/sign split.
- D, 64: is 2^6, larger than any 6-bit code.

Final Answer: largest positive value $= 31 \Rightarrow$ B

Answer: (B) Go Back to Q1

Q2.

Solution

Concept – push the inversions through the NOR using De Morgan's law: $\overline{X+Y} = \overline{X} \cdot \overline{Y}$.

Step 1 – outputs of the two NOT gates: top gate gives $\overline{A}$, bottom gate gives $\overline{B}$.

Step 2 – feed these into the NOR gate: $F = \overline{\overline{A} + \overline{B}}$

Step 3 – apply De Morgan's law: $F = \overline{\overline{A}} \cdot \overline{\overline{B}}$

Step 4 – cancel the double complements: $\overline{\overline{A}} = A$ and $\overline{\overline{B}} = B$, so $F = A \cdot B$.

Why the other options are wrong:

- A, $A + B$: would need an OR of the true inputs, not a NOR of the complements.
- C, $A \oplus B$: requires cross terms $A\overline{B} + \overline{A}B$, which this circuit does not form.
- D, $\overline{A+B}$: is $\overline{A}\,\overline{B}$, the NOR of the *true* inputs, not of the inverted ones.

Final Answer: $F = A \cdot B \Rightarrow$ B

Answer: (B) Go Back to Q2

Q3.

Solution

Concept – group the 1s on the K-map into the largest power-of-two block: A group of 8 adjacent cells eliminates three variables, leaving a single literal.

Step 1 – locate the 1s: they fill the two columns labelled $CD = 11$ and $CD = 10$, across all four rows.

Step 2 – see what is common to these two columns: column $CD = 11$ has $C = 1, D = 1$; column $CD = 10$ has $C = 1, D = 0$. In both columns $C = 1$.

Step 3 – check the variable that changes: D is 1 in one column and 0 in the other, and A, B take every value down the rows, so A, B, D all change within the group.

Step 4 – keep only the constant literal: the eight-cell group is described by the single variable that stays fixed, namely $C = 1$.

$F = C$

Final Answer: $F = C \Rightarrow$ D

Answer: (D) Go Back to Q3

Q4.

Solution

Concept – build a big decoder as a tree of small decoders: One decoder selects which lower-level decoder is enabled, and the lower-level decoders produce the 16 output lines.

Step 1 – outputs needed and outputs per small decoder: we need 16 output lines; each 2-to-4 decoder supplies 4 lines.

Step 2 – count the output-stage decoders: $16 \div 4 = 4$ decoders are needed to generate all 16 lines.

Step 3 – add the enable-selection decoder: the two most-significant address bits must choose which of the four output decoders is active; one more 2-to-4 decoder drives their enable inputs.

Step 4 – total the decoders: $4 + 1 = 5$.

Final Answer: 5 decoders $\Rightarrow$ D

Answer: (D) Go Back to Q4

Q5.

Solution

Concept – number of states of an m-bit counter: m flip-flops, each storing one bit, produce 2^m distinct binary patterns, and a free-running counter cycles through all of them.

Step 1 – note the number of flip-flops: $m = 4$.

Step 2 – compute the number of states: number of states $= 2^m = 2^4$.

Step 3 – evaluate: $2^4 = 16$.

Step 4 – interpret: the counter runs $0000, 0001, \ldots, 1111$ and then repeats, so its modulus is 16.

Final Answer: 16 states $\Rightarrow$ C

Answer: (C) Go Back to Q5

Q6.

Solution

Concept – a ring counter circulates a single 1: In an N-stage ring counter, one flip-flop is set and the 1 shifts one position each clock, returning to its start after N shifts.

Step 1 – note the number of stages: $N = 4$.

Step 2 – trace the pattern from 1000: $1000 \to 0100 \to 0010 \to 0001 \to 1000$.

Step 3 – count the distinct states before repetition: $1000, 0100, 0010, 0001$ are 4 distinct patterns, then it repeats.

Step 4 – conclude: an N-stage ring counter has exactly $N = 4$ states (unlike a 4-bit binary counter, which has 16).

Final Answer: 4 states $\Rightarrow$ B

Answer: (B) Go Back to Q6

Q7.

Solution

Concept – classify the addressing mode by where the operand's address comes from: If a register holds the *address* of the operand (not the operand itself), the mode is register-indirect.

Step 1 – read the rule given: the instruction names a register, and the contents of that register are used as the memory address of the operand.

Step 2 – match to the standard name: using a register's contents as a pointer to memory is register-indirect addressing.

Step 3 – rule out the others:

- Immediate: the operand value is in the instruction; no register, no memory address.
- Direct (absolute): the address is a constant in the instruction, not held in a register.
- Indexed: the effective address adds an index register to a base displacement, which is not what is described here.

Final Answer: register-indirect addressing $\Rightarrow$ $\boxed{\text{C}}$

Answer: (C) Go Back to Q7

Q8.

Solution

Concept – ideal pipeline speedup: For a very large number of tasks, speedup $\approx$ (non-pipelined time per task) $\div$ (pipelined time per task), where the pipelined time per task equals one stage time.

Step 1 – list the data: non-pipelined time $= 8$ ns; number of stages $= 4$; each stage $= 2$ ns.

Step 2 – find the pipelined throughput time: in steady state one task completes every clock period $= 2$ ns.

Step 3 – compute the speedup: speedup $= \dfrac{8 \text{ ns}}{2 \text{ ns}} = 4$.

Step 4 – cross-check with the number of stages: for balanced stages with no overhead, the ideal speedup equals the number of stages, 4. Consistent.

Final Answer: speedup $= 4 \Rightarrow$ $\boxed{\text{C}}$

Answer: (C) Go Back to Q8

Q9.

Solution

Concept – address fields in a direct-mapped cache: A memory (block) address splits into a tag and a line-index; tag bits = (block-number bits) − (line-index bits).

Step 1 – find the line-index bits: the cache has $64 = 2^6$ lines, so the line index needs 6 bits.

Step 2 – find the block-number bits: main memory has $4096 = 2^{12}$ blocks, so identifying a block needs 12 bits.

Step 3 – subtract to get the tag bits: tag bits $= 12 - 6 = 6$.

Step 4 – interpret: 6 tag bits plus 6 index bits fully identify any of the 2^{12} blocks.

Final Answer: 6 tag bits $\Rightarrow$ B

Answer: (B) Go Back to Q9

Q10.

Solution

Concept – bitwise AND compares the two operands bit by bit: The result bit is 1 only where both operand bits are 1.

Step 1 – write each value in binary: $12 = 1100_2$

$10 = 1010_2$

Step 2 – AND the aligned bits: $1 \,\&\, 1 = 1$ (bit 3)

$1 \,\&\, 0 = 0$ (bit 2)

$0 \,\&\, 1 = 0$ (bit 1)

$0 \,\&\, 0 = 0$ (bit 0)

Step 3 – assemble the result: 1000_2

Step 4 – convert back to decimal: $1000_2 = 8$.

Final Answer: the program prints $8 \Rightarrow$ C

Answer: (C) Go Back to Q10

Q11.

Solution

Concept – evaluate postfix by pushing operands and applying each operator to the top two: For a binary operator, pop two values, combine, and push the result.

Step 1 – push 5 and 3: stack (bottom→top): $5, 3$.

Step 2 – apply $+$ to the top two: $5 + 3 = 8$; stack: 8.

Step 3 – push 2: stack: $8, 2$.

Step 4 – apply $*$ to the top two: $8 \times 2 = 16$; stack: 16.

Step 5 – read the final value: one value remains: 16.

Final Answer: the expression evaluates to $16 \Rightarrow \boxed{B}$

Answer: (B) Go Back to Q11

Q12.

Solution

Concept – simulating a FIFO queue with stacks: A stack reverses order, so a single stack cannot preserve FIFO order; two stacks together restore it.

Step 1 – see why one stack is not enough: a stack is LIFO; popping returns the most recently pushed item, the opposite of queue (FIFO) order.

Step 2 – use an "inbox" and an "outbox" stack: push all enqueued items onto stack 1; when a dequeue is needed and stack 2 is empty, pop everything from stack 1 into stack 2, reversing the order so the oldest item is on top.

Step 3 – dequeue: pop from stack 2, which now yields items in arrival order.

Step 4 – conclude: exactly 2 stacks are required.

Final Answer: 2 stacks $\Rightarrow \boxed{A}$

Answer: (A) Go Back to Q12

Q13.

Solution

Concept – deletion in a doubly linked list with a node pointer: Because each node stores both a previous and a next pointer, a given node's neighbours are reachable in constant time.

Step 1 – locate the neighbours from X: let $P = X$.prev and $N = X$.next; both

are available directly from X's own pointers.

Step 2 – splice X out: set P.next $= N$ and N.prev $= P$ (guarding the ends if P or N is null).

Step 3 – free X: delete the node X; all steps used a fixed number of pointer updates.

Step 4 – state the cost: no traversal is needed, so the deletion runs in $O(1)$ time.

Final Answer: $O(1) \Rightarrow$ C

Answer: (C) Go Back to Q13

Q14.

Solution

Concept – postorder traversal: Postorder visits (left subtree, right subtree, root) recursively.

Step 1 – traverse the left subtree of root A (rooted at B): B has children D (left) and E (right); postorder of this subtree visits D, then E, then B.

Left part $= D,\ E,\ B$.

Step 2 – traverse the right subtree of root A (rooted at C): C has only the child F; postorder visits F, then C.

Right part $= F,\ C$.

Step 3 – visit the root last: A.

Step 4 – concatenate the three parts: $D,\ E,\ B,\ F,\ C,\ A$.

Final Answer: $D\ E\ B\ F\ C\ A \Rightarrow$ A

Answer: (A) Go Back to Q14

Q15.

Solution

Concept – leaf nodes of an array-based heap: With 1-based indexing, indices $\lfloor n/2 \rfloor + 1$ through n have no children, so they are the leaves.

Step 1 – identify the internal (non-leaf) nodes: a node at index i has children only if $2i \le n$, i.e. $i \le \lfloor n/2 \rfloor$; these are the internal nodes.

Step 2 – count the internal nodes: there are $\lfloor n/2 \rfloor$ internal nodes.

Step 3 – subtract from the total: leaves $= n - \lfloor n/2 \rfloor$.

Step 4 – simplify: $n - \lfloor n/2 \rfloor = \lceil n/2 \rceil$.

Step 5 – spot check ($n = 7$): $\lceil 7/2 \rceil = 4$ leaves; indeed indices $4, 5, 6, 7$ are leaves. Correct.

Final Answer: $\lceil n/2 \rceil$ leaves $\Rightarrow$ C

Answer: (C) Go Back to Q15

Q16.

Solution

Concept – count the edges directly from the drawing: Each line segment joining two vertices is one edge.

Step 1 – list the edges shown: $PQ,\ QR,\ RS,\ SP,\ PR,\ QT$.

Step 2 – tally them: that is 6 distinct edges.

Step 3 – cross-check with the handshaking lemma: $\deg(P) = 3,\ \deg(Q) = 3,\ \deg(R) = 3,\ \deg(S) = 2,\ \deg(T) = 1$; sum $= 3 + 3 + 3 + 2 + 1 = 12$, and $12 = 2 \times 6$, confirming 6 edges.

Final Answer: 6 edges $\Rightarrow$ A

Answer: (A) Go Back to Q16

Q17.

Solution

Concept – the dominant term sets the Θ bound: Drop lower-order terms and constant factors; the fastest-growing term remains.

Step 1 – list the terms of $f(n)$: $3n^2$, $5n \log_2 n$, and the constant 100.

Step 2 – compare growth rates: n^2 grows faster than $n \log n$, which grows faster than a constant, so n^2 dominates.

Step 3 – drop the constant factor and lower terms: $f(n) = \Theta(n^2)$.

Why the other options are wrong:

- B, $\Theta(n \log n)$: ignores the larger $3n^2$ term.
- C, $\Theta(n^3)$: overstates the growth; there is no cubic term.
- D, $\Theta(n)$: far too small.

Final Answer: $\Theta(n^2) \Rightarrow$ A

Answer: (A) Go Back to Q17

Q18.

Solution

Concept – Master theorem for $T(n) = a\,T(n/b)+f(n)$: Compare $f(n)$ with $n^{\log_b a}$.

Step 1 – identify the parameters: $a = 1$, $b = 2$, $f(n) = 1$.

Step 2 – compute the critical exponent: $\log_b a = \log_2 1 = 0$, so $n^{\log_b a} = n^0 = 1$.

Step 3 – compare $f(n)$ with $n^{\log_b a}$: $f(n) = 1 = \Theta(n^0)$, which is Master-theorem Case 2.

Step 4 – apply Case 2: $T(n) = \Theta\big(n^{\log_b a}\log n\big) = \Theta(\log n)$.

Step 5 – sanity note: this is the binary-search recurrence, whose known cost is $\Theta(\log n)$.

Final Answer: $\Theta(\log n) \Rightarrow \boxed{\text{C}}$

Answer: (C) Go Back to Q18

Q19.

Solution

Concept – merge sort always splits evenly: Merge sort divides the array into two halves regardless of the input, so its recurrence never degrades.

Step 1 – write the recurrence: $T(n) = 2\,T(n/2) + \Theta(n)$, where $\Theta(n)$ is the merge step.

Step 2 – apply the Master theorem: $a = 2$, $b = 2$, $f(n) = n$; $n^{\log_2 2} = n$, so $f(n) = \Theta(n)$ is Case 2.

Step 3 – solve: $T(n) = \Theta(n \log n)$.

Step 4 – note it is input-independent: because the split is always balanced, best, average and worst cases are all $\Theta(n \log n)$.

Final Answer: $\Theta(n \log n) \Rightarrow \boxed{\text{D}}$

Answer: (D) Go Back to Q19

Q20.

Solution

Concept – track what each state remembers: The accepting state is reached exactly after the symbol that the language cares about.

Step 1 – effect of reading 0: from A, a 0 goes to B; from B, a 0 stays in B. So after any 0 the machine is in B.

Step 2 – effect of reading 1**:** from A, a 1 stays in A; from B, a 1 goes back to A. So after any 1 the machine is in A.

Step 3 – determine when we accept: B is the only accepting state, and the machine is in B precisely when the last symbol read was a 0.

Step 4 – state the language: the DFA accepts exactly the strings that end with 0.

Final Answer: strings ending with $0 \Rightarrow$ B

Answer: (B) Go Back to Q20

Q21.

Solution

Concept – read the concatenation a^*b^* **left to right:** a^* matches any block of a's, then b^* matches any block of b's; the order is fixed.

Step 1 – interpret a^***:** zero or more a's at the front.

Step 2 – interpret b^***:** zero or more b's immediately after, and nothing else follows.

Step 3 – combine: every matched string is a run of a's followed by a run of b's, so once a b appears no further a may occur.

Step 4 – reject the wrong readings: it is *not* every string (e.g. ba is excluded), it does not force equal counts (e.g. aab is allowed), and it need not end in b (e.g. aa and ε are allowed).

Final Answer: zero or more a's followed by zero or more b's $\Rightarrow$ D

Answer: (D) Go Back to Q21

Q22.

Solution

Concept – place $\{a^n b^n c^n\}$ **in the Chomsky hierarchy:** This is the classic example separating context-free from context-sensitive languages.

Step 1 – show it is not context-free: by the pumping lemma for context-free languages, pumping $s = a^p b^p c^p$ cannot keep all three counts equal, so L is not context-free (hence not regular).

Step 2 – show it is context-sensitive: a linear-bounded automaton can mark and match one a, one b and one c at a time, verifying equal counts within linear space, so L is context-sensitive.

Step 3 – combine the results: L is context-sensitive but not context-free.

Why the other options are wrong:

- A, regular: fails, since even the context-free property fails.
- B, context-free: disproved in Step 1.
- C, not recursively enumerable: wrong; L is decidable, hence certainly recursively enumerable.

Final Answer: context-sensitive but not context-free $\Rightarrow$ D

Answer: (D) Go Back to Q22

Q23.

Solution

Concept – decidable vs undecidable problems: Problems about DFAs are decidable; many general problems about Turing machines and CFG equivalence are not.

Step 1 – test option A (DFA membership): simulate the DFA on w; it halts after exactly $|w|$ steps and reports accept/reject. Decidable.

Step 2 – test option B (TM halting): the general halting problem is undecidable (Turing).

Step 3 – test option C (TM emptiness): deciding whether a Turing machine accepts no strings is undecidable (Rice's theorem).

Step 4 – test option D (CFG equivalence): deciding $L(G_1) = L(G_2)$ for context-free grammars is undecidable.

Step 5 – conclude: only option A is decidable.

Final Answer: DFA membership is decidable $\Rightarrow$ A

Answer: (A) Go Back to Q23

Q24.

Solution

Concept – states track progress toward the pattern ab: We only need to remember how much of the target substring has been matched so far.

Step 1 – define the states: q_0 = "no useful progress", q_1 = "last symbol was a", q_2 = "ab already seen" (accepting, absorbing).

Step 2 – justify the transitions (as drawn): from q_0: $a \to q_1$, $b \to q_0$; from q_1: $a \to q_1$, $b \to q_2$; from q_2: any symbol stays in q_2.

Step 3 – argue minimality: the three states are pairwise distinguishable (a suffix accepted from one is rejected from another), so no DFA with fewer than 3 states can recognise this language.

Step 4 – conclude: the minimum number of states is 3.

Final Answer: 3 states $\Rightarrow$ B

Answer: (B) Go Back to Q24

Q25.

Solution

Concept – match the task to the compiler phase: Checking declarations, scope and type compatibility is done after parsing, using the symbol table.

Step 1 – describe the task: verifying that identifiers are declared before use and that operand types agree is semantic (meaning) checking.

Step 2 – name the phase: this is performed by the semantic analyzer, which annotates the parse/syntax tree and consults the symbol table.

Step 3 – separate it from neighbouring phases: the lexical analyzer only tokenizes, the parser only checks grammatical structure, and the code generator emits target code; none performs type checking.

Final Answer: semantic analyzer $\Rightarrow$ C

Answer: (C) Go Back to Q25

Q26.

Solution

Concept – FOLLOW of a non-terminal: $\text{FOLLOW}(A)$ is the set of terminals that can appear immediately to the right of A in some derivation.

Step 1 – find where A appears: A occurs only in $S \to A\,B$, immediately followed by B.

Step 2 – add $\text{FIRST}(B)$: $B \to b$, so $\text{FIRST}(B) = \{b\}$; add b to $\text{FOLLOW}(A)$.

Step 3 – check whether B is nullable: B derives only b, so B is not nullable; therefore $\text{FOLLOW}(S)$ is *not* added to $\text{FOLLOW}(A)$.

Step 4 – collect the set: $\text{FOLLOW}(A) = \{b\}$.

Final Answer: $\{b\} \Rightarrow$ B

Answer: (B) Go Back to Q26

Q27.

Solution

Concept – preparing a grammar for predictive parsing: Top-down predictive parsers need a grammar with no left recursion and no common prefixes among alternatives.

Step 1 – why remove left recursion: immediate left recursion sends a top-down parser into infinite recursion, so it must be eliminated first.

Step 2 – why left factoring: common prefixes make the parser unable to choose a production by looking at one token; left factoring defers the choice.

Step 3 – name the beneficiary technique: both transformations make the grammar suitable for LL(1) top-down predictive parsing.

Step 4 – rule out the others: LR, LALR and operator-precedence parsers are bottom-up and handle left recursion directly, so they do not require these transformations.

Final Answer: top-down predictive LL(1) parsing $\Rightarrow$ C

Answer: (C) Go Back to Q27

Q28.

Solution

Concept – name the optimization by what it does: Replacing an expensive operation with an equivalent cheaper one is strength reduction.

Step 1 – read the transformation: $x \times 2$ is rewritten as $x + x$ (or a left shift), which computes the same value more cheaply.

Step 2 – match to the standard term: turning a costly multiply into a cheap add/shift is exactly strength reduction.

Step 3 – separate the distractors:

- Constant folding evaluates constant sub-expressions at compile time (e.g. $3 \times 4 \rightarrow 12$).
- Common subexpression elimination reuses a value already computed.
- Dead-code elimination removes computations whose results are never used.

Final Answer: strength reduction $\Rightarrow$ A

Answer: (A) Go Back to Q28

Q29.

Solution

Concept – FCFS runs jobs in arrival order; waiting time = start time here: With all arrivals at 0, a process's waiting time equals the sum of the burst times of the jobs run before it.

Step 1 – read the run order and start times from the Gantt chart: P_1 starts at 0, P_2 at 3, P_3 at 9, P_4 at 13.

Step 2 – write each waiting time (start − arrival, arrival = 0): $W_{P_1} = 0, \quad W_{P_2} = 3, \quad W_{P_3} = 9, \quad W_{P_4} = 13.$

Step 3 – sum the waiting times: $0 + 3 + 9 + 13 = 25.$

Step 4 – divide by the number of processes: avg $= \dfrac{25}{4} = 6.25$ ms.

Final Answer: average waiting time $= 6.25$ ms $\Rightarrow$ A

Answer: (A) Go Back to Q29

Q30.

Solution

Concept – `wait` decrements and `signal` increments a counting semaphore: Net change = (number of signals) − (number of waits).

Step 1 – record the initial value: $S = 10.$

Step 2 – apply the six `wait` operations: each successful `wait` subtracts 1: $10-6 = 4.$

Step 3 – apply the two `signal` operations: each `signal` adds 1: $4 + 2 = 6.$

Step 4 – state the result: $S = 6$, meaning 6 resource units remain free.

Final Answer: $S = 6 \Rightarrow$ D

Answer: (D) Go Back to Q30

Q31.

Solution

Concept – attack one Coffman condition to prevent deadlock: Each deadlock-prevention strategy negates exactly one of the four necessary conditions.

Step 1 – target the circular-wait condition: number all resource types $1, 2, 3, \ldots$ and require every process to request resources only in increasing order of that number.

Step 2 – see why this breaks a cycle: in any set of processes each holding one resource and waiting for another, following the numbering forbids a closed loop, so no circular wait can form.

Step 3 – rule out the other options:

- Banker's algorithm is a deadlock *avoidance* method, not a prevention of circular wait.
- Preempting/rolling back resources attacks the "no preemption" condition.
- Requesting all resources at once attacks the "hold and wait" condition.

Final Answer: resource ordering with increasing-order requests $\Rightarrow$ A

Answer: (A) Go Back to Q31

Q32.

Solution

Concept – number of page-table entries: A single-level page table has one entry per page, and (number of pages) = (virtual address space) $\div$ (page size).

Step 1 – express the sizes as powers of two: virtual address space $= 2^{32}$ bytes; page size $= 4$ KB $= 2^{12}$ bytes.

Step 2 – find the number of pages: $\dfrac{2^{32}}{2^{12}} = 2^{32-12} = 2^{20}$ pages.

Step 3 – relate entries to pages: one page-table entry per page $\Rightarrow 2^{20}$ entries.

Step 4 – interpret: that is about $1{,}048{,}576$ entries in the single-level table.

Final Answer: 2^{20} entries $\Rightarrow$ D

Answer: (D) Go Back to Q32

Q33.

Solution

Concept – LRU replaces the page unused for the longest time: Simulate the three frames; on a miss when full, evict the least recently used page.

Step 1 – 1**:** miss (fault 1); frames $\{1\}$.

Step 2 – 2**:** miss (fault 2); frames $\{1, 2\}$.

Step 3 – 3**:** miss (fault 3); frames $\{1, 2, 3\}$.

Step 4 – 4**:** miss (fault 4), least recently used is 1; evict 1; frames $\{2, 3, 4\}$.

Step 5 – 1**:** miss (fault 5), least recently used is 2; evict 2; frames $\{3, 4, 1\}$.

Step 6 – 2: miss (fault 6), least recently used is 3; evict 3; frames $\{4, 1, 2\}$.

Step 7 – 5: miss (fault 7), least recently used is 4; evict 4; frames $\{1, 2, 5\}$.

Step 8 – 1: hit (1 present); frames $\{1, 2, 5\}$.

Step 9 – 2: hit (2 present); frames $\{1, 2, 5\}$.

Step 10 – total the faults: 7 page faults in all.

Final Answer: 7 page faults $\Rightarrow$ B

Answer: (B) Go Back to Q33

Q34.

Solution

Concept – drawback of contiguous allocation: Requiring each file in one consecutive run of blocks scatters free space into small gaps over time.

Step 1 – describe what happens with use: as files of different sizes are created and deleted, the freed regions become interleaved with allocated ones.

Step 2 – name the effect: enough total free blocks may exist, but no single contiguous run is large enough for a new file; this is external fragmentation.

Step 3 – rule out the distractors:

- Sequential access is actually *fast* under contiguous allocation (adjacent blocks).
- Large index/allocation tables are a concern of indexed/FAT schemes, not contiguous allocation.
- Internal fragmentation (partly used last block) is minor here and not the main disadvantage.

Final Answer: external fragmentation $\Rightarrow$ D

Answer: (D) Go Back to Q34

Q35.

Solution

Concept – match the relational-algebra operation to its role: Choosing a subset of columns (and removing resulting duplicates) is projection π.

Step 1 – describe projection: $\pi_{A_1,\ldots,A_k}(R)$ keeps only the listed attributes and eliminates duplicate rows from the result.

Step 2 – contrast with the distractors:

- Selection σ chooses rows by a predicate, keeping all columns.
- Rename ρ only changes relation/attribute names.
- Cartesian product $\times$ pairs every tuple of one relation with every tuple of another.

Step 3 – conclude: column selection with duplicate removal is projection π.

Final Answer: projection $(\pi) \Rightarrow$ B

Answer: (B) Go Back to Q35

Q36.

Solution

Concept – the 2NF condition: R is in 2NF if it is in 1NF and no non-prime attribute is partially dependent on a candidate key (i.e. it depends on the whole key, not just part of it).

Step 1 – state the rule precisely: a partial dependency is a non-prime attribute determined by a proper subset of some candidate key; 2NF forbids all such dependencies.

Step 2 – distinguish from the other forms:

- "every determinant is a superkey" is the BCNF condition.
- "no transitive dependency of a non-prime attribute" is the 3NF condition.
- "all attribute values atomic" is the 1NF condition.

Step 3 – conclude: the added requirement for 2NF is that no non-prime attribute is partially dependent on any candidate key.

Final Answer: no partial dependency of a non-prime attribute $\Rightarrow$ A

Answer: (A) Go Back to Q36

Q37.

Solution

Concept – the ACID properties: Atomicity (all or nothing), Consistency (valid to valid state), Isolation (concurrent runs act as if serial), Durability (committed effects survive failures).

Step 1 – match the description: "concurrent execution equivalent to some serial order" is the definition of serializability, which the isolation property provides.

Step 2 – how it is achieved: concurrency-control mechanisms such as two-phase

locking or timestamp ordering enforce serializable schedules.

Step 3 – eliminate the others: atomicity concerns all-or-nothing execution, durability concerns surviving crashes, and consistency concerns integrity constraints, none of which is about equivalence to a serial schedule.

Final Answer: isolation $\Rightarrow$ C

Answer: (C) Go Back to Q37

Q38.

Solution

Concept – minimum occupancy of a B-tree node: In a B-tree of order m, every node holds at most m children, and every internal node except the root must be at least half full.

Step 1 – state the maximum: an internal node has at most m child pointers.

Step 2 – apply the "at least half full" rule: each non-root internal node must have at least $\lceil m/2 \rceil$ child pointers.

Step 3 – note why the root is exempt: the root is allowed as few as 2 children so the tree can start small; the bound $\lceil m/2 \rceil$ applies to all other internal nodes.

Step 4 – conclude: minimum children of a non-root internal node $= \lceil m/2 \rceil$.

Final Answer: $\lceil m/2 \rceil$ children $\Rightarrow$ A

Answer: (A) Go Back to Q38

Q39.

Solution

Concept – map network devices to OSI layers: A device is placed at the highest layer whose addressing it processes; a switch forwards using MAC (physical) addresses.

Step 1 – identify what a switch examines: it reads the destination MAC address of each frame and consults its MAC address table to choose the outgoing port.

Step 2 – match MAC addressing to a layer: MAC addressing and frame forwarding belong to the data-link layer (layer 2).

Step 3 – contrast with other devices: a router works at the network layer (layer 3, IP addresses), and a hub/repeater at the physical layer (layer 1).

Final Answer: data-link layer (layer 2) $\Rightarrow$ C

Answer: (C) Go Back to Q39

Q40.

Solution

Concept – number of check bits in a single-error-correcting Hamming code: For m data bits and r check bits, the code must satisfy $2^r \geq m + r + 1$.

Step 1 – state the data size: $m = 4$ data bits.

Step 2 – test $r = 2$: $2^2 = 4$ and $m + r + 1 = 4 + 2 + 1 = 7$; since $4 < 7$, two check bits are not enough.

Step 3 – test $r = 3$: $2^3 = 8$ and $m + r + 1 = 4 + 3 + 1 = 8$; since $8 \geq 8$, three check bits suffice.

Step 4 – conclude: the minimum number of parity bits is 3 (giving a $(7, 4)$ Hamming code).

Final Answer: 3 parity bits $\Rightarrow$ D

Answer: (D) Go Back to Q40

Q41.

Solution

Concept – window-size limit for Selective Repeat: With n-bit sequence numbers there are 2^n numbers; Selective Repeat needs the sender window $W_S \leq 2^{n-1}$.

Step 1 – count the sequence numbers: an n-bit field gives 2^n distinct sequence numbers.

Step 2 – why only half may be used: the sender and receiver windows must not overlap; if the window exceeded 2^{n-1}, a retransmitted old frame could fall inside the receiver's window and be mistaken for a new one.

Step 3 – state the maximum: maximum sender window $= 2^{n-1}$.

Step 4 – contrast with Go-Back-N: Go-Back-N instead allows up to $2^n - 1$; the question specifically asks about Selective Repeat.

Final Answer: $2^{n-1} \Rightarrow$ B

Answer: (B) Go Back to Q41

Q42.

Solution

Concept – length of an Ethernet MAC address: A standard MAC address is written as six hexadecimal octets.

Step 1 – count the octets: a MAC address has 6 octets, e.g. `00:1A:2B:3C:4D:5E`.

Step 2 – convert octets to bits: $6 \times 8 = 48$ bits.

Step 3 – interpret the structure: the first 24 bits are the OUI (manufacturer) and the last 24 bits are the device-specific portion, totalling 48 bits.

Why the other options are wrong:

- A, 32 bits: is the length of an IPv4 address, not a MAC address.
- C, 64 bits: is the length of an EUI-64 identifier, not a standard Ethernet MAC.

Final Answer: 48 bits $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q42

Q43.

Solution

Concept – number of subnets from extra borrowed bits: Number of subnets $= 2^{(\text{new prefix} - \text{old prefix})}$.

Step 1 – find the borrowed bits: new prefix $/20$ minus old prefix $/16$ gives $20 - 16 = 4$ borrowed bits.

Step 2 – compute the number of subnets: $2^4 = 16$.

Step 3 – interpret: the $/16$ block splits into 16 equal $/20$ subnets.

Why the other options are wrong:

- A, 4: would be 2^2, i.e. only 2 borrowed bits.
- C, 32: would be 2^5, i.e. 5 borrowed bits (a $/21$ split).

Final Answer: 16 subnets $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q43

Q44.

Solution

Concept – usable hosts in a subnet: With h host bits, usable hosts $= 2^h - 2$ (subtracting the network and broadcast addresses).

Step 1 – find the number of host bits for $/29$**:** $h = 32 - 29 = 3$ host bits.

Step 2 – compute the total addresses in the subnet: $2^3 = 8$.

Step 3 – subtract the two reserved addresses: $8 - 2 = 6$ usable host addresses.

Step 4 – interpret: a $/29$ subnet provides 6 assignable host addresses.

Final Answer: 6 usable hosts $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q44

Q45.

Solution

Concept – transport-layer protocols in TCP/IP: UDP is connectionless and best-effort; TCP is connection-oriented and reliable.

Step 1 – list UDP's characteristics: no connection setup, no acknowledgements, no ordering or retransmission, and a small 8-byte header giving low overhead.

Step 2 – match to the described service: "connectionless, no reliability or ordering, low overhead, used by DNS and streaming" is exactly UDP.

Step 3 – eliminate the others: TCP is connection-oriented and reliable; IP is a network-layer protocol; SCTP is connection-oriented with reliability, unlike the description.

Final Answer: UDP $\Rightarrow$ $\boxed{\text{C}}$

Answer: (C) Go Back to Q45

Q46.

Solution

Concept – well-known port numbers: Common application protocols use fixed well-known ports below 1024.

Step 1 – recall DNS's port: the Domain Name System uses port 53 (over UDP for ordinary queries, and TCP for zone transfers/large responses).

Step 2 – distinguish the distractors: port 25 is SMTP (mail), port 80 is HTTP (web), and port 110 is POP3 (mail retrieval).

Step 3 – conclude: the well-known port for DNS is 53.

Final Answer: port $53 \Rightarrow \boxed{\text{B}}$

Answer: (B) Go Back to Q46

Answer Key

Q	Ans	Q	Ans	Q	Ans	Q	Ans	Q	Ans
1	B	2	B	3	D	4	D	5	C
6	B	7	C	8	C	9	B	10	C
11	B	12	A	13	C	14	A	15	C
16	A	17	A	18	C	19	D	20	B
21	D	22	D	23	A	24	B	25	C
26	B	27	C	28	A	29	A	30	D
31	A	32	D	33	B	34	D	35	B
36	A	37	C	38	A	39	C	40	D
41	B	42	D	43	D	44	A	45	C
46	B								