GATE Computer Science and Information Technology

Sample Paper – 4

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. The hexadecimal number $(2\mathrm{AF})_{16}$ is converted to its decimal (base-10) equivalent. The decimal value is: **[1 mark]**

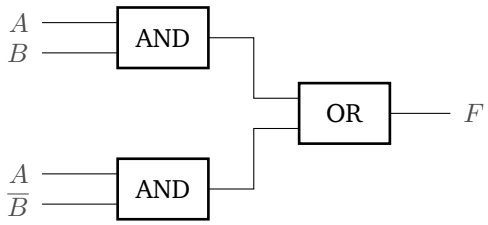
(A) 685

(B) 671

(C) 703

(D) 687

Q2. The combinational circuit shown below uses two AND gates and one OR gate. The Boolean output F produced by the circuit simplifies to: **[1 mark]**



(A) B

(B) $\overline{A}$

(C) A

(D) $A \cdot B$

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) D

(B) $\overline{C}$

(C) $B\,D$

(D) C

Q4. A 16-to-1 multiplexer selects one of its 16 data inputs and routes it to the single output. The number of select (control) lines the multiplexer requires is: **[1 mark]**

(A) 4

(B) 16

(C) 8

(D) 2

Q5. A 4-bit binary up-counter counts upward from 0 and rolls over after reaching its highest value. The maximum count value the counter reaches before rolling over is: **[1 mark]**

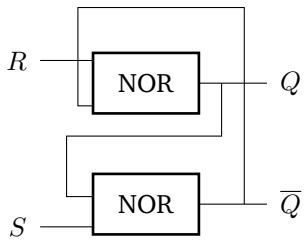
(A) 16

(B) 8

(C) 31

(D) 15

Q6. The NOR-based SR latch shown below is built from two cross-coupled NOR gates. When both inputs are driven simultaneously to $S = 1$ and $R = 1$, the latch enters a state that is: **[1 mark]**



(A) the set state with $Q = 1$

(B) a stable hold of the previous value

(C) forbidden, producing an indeterminate output

(D) the reset state with $Q = 0$

Q7. In one addressing mode, the operand value itself is encoded explicitly inside the instruction, so the CPU needs no additional memory access to fetch that operand. This addressing mode is called: **[1 mark]**

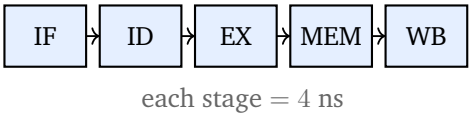
(A) indirect addressing

(B) immediate addressing

(C) indexed addressing

(D) register-indirect addressing

Q8. A non-pipelined processor takes 20 ns to execute one instruction. It is redesigned as the 5-stage pipeline shown below, with every stage taking 4 ns. For a very large number of instructions, the ideal (asymptotic) speedup of the pipelined design over the non-pipelined one is: **[1 mark]**



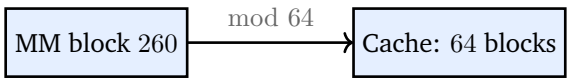
(A) 5

(B) 4

(C) 20

(D) 1

Q9. A direct-mapped cache has 64 blocks. Main memory is divided into blocks of the same size, and block number 260 of main memory is to be placed in the cache, as suggested by the mapping below. The cache block index to which memory block 260 maps is: **[1 mark]**



(A) 64

(B) 4

(C) 260

(D) 8

Programming, Data Structures & Algorithms

Q10. Consider the following C statements:

```
int a = 10, b = 4;    printf("%d", a%b + b%a);
```

The value printed is: **[1 mark]**

(A) 6

(B) 2

(C) 14

(D) 8

Q11. The infix arithmetic expression

$$A + B * C$$

(with the usual operator precedence, and no parentheses) is converted to its equivalent postfix (reverse-Polish) form. The correct postfix expression is: **[1 mark]**

(A) $A\,B + C\,*$

(B) $A\,B\,C + *$

(C) $A + B\,C\,*$

(D) $A\,B\,C\,* +$

Q12. Which one of the following data structures is the most natural choice for checking whether the parentheses in an arithmetic expression are correctly balanced? **[1 mark]**

(A) a simple (FIFO) queue

(B) a priority queue

(C) a circular linked list

(D) a stack

Q13. A singly linked list is given together with a pointer to its head node. Inserting a new node at the *front* (beginning) of this list takes time: **[1 mark]**

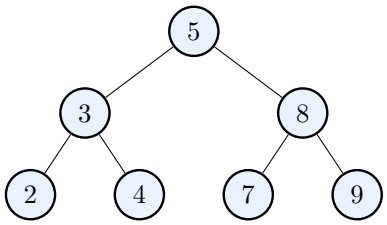
(A) $O(n)$

(B) $O(\log n)$

(C) $O(1)$

(D) $O(n^2)$

Q14. For the binary tree shown below, the sequence of nodes produced by a *preorder* traversal (visit root, then left subtree, then right subtree) is: [1 mark]



(A) 5 3 2 4 8 7 9

(B) 2 3 4 5 7 8 9

(C) 2 4 3 7 9 8 5

(D) 5 3 8 2 4 7 9

Q15. An unsorted array of n elements is turned into a binary max-heap using the standard bottom-up "build-heap" procedure (repeated heapify from the last internal node up to the root). The tight worst-case time complexity of this build-heap operation is: [1 mark]

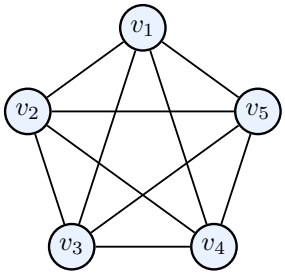
(A) $O(n \log n)$

(B) $O(n)$

(C) $O(\log n)$

(D) $O(n^2)$

Q16. The complete graph K_5 on five vertices is shown below, in which every pair of distinct vertices is joined by exactly one edge. The total number of edges in K_5 is: [1 mark]



(A) 20

(B) 5

(C) 25

(D) 10

Q17. Consider the function $f(n) = 3n^2 + 5n + 100$. Its tightest asymptotic (Θ) bound as $n \to \infty$ is: **[1 mark]**

(A) $\Theta(n)$

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

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

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

Q18. The running time of a divide-and-conquer algorithm satisfies the recurrence

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

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

(A) $\Theta(n \log n)$

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

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

(D) $\Theta(n)$

Q19. Merge sort divides the array into two halves, sorts each recursively, and merges them. The worst-case time complexity of merge sort on an array of n elements is: **[1 mark]**

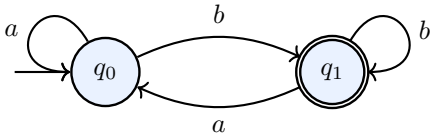
(A) $\Theta(n \log n)$

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

(C) $\Theta(n)$

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

Q20. The deterministic finite automaton (DFA) over the alphabet $\{a, b\}$ is shown below; q_1 (double circle) is the only accepting state and q_0 is the start state. The language accepted by this DFA is exactly the set of all strings that: **[1 mark]**



(A) contain an even number of b's

(B) begin with the symbol b

(C) end with the symbol b

(D) contain the substring ba

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

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

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

(A) contain exactly one a and one b

(B) have equal numbers of a's and b's

(C) contain the substring ab

(D) consist of zero or more a's followed by zero or more b's

Q22. Consider the language $L = \{\, a^n b^n c^n \mid n \geq 1 \,\}$ over the alphabet $\{a, b, c\}$. In the Chomsky hierarchy, this language is: **[2 marks]**

(A) regular

(B) context-free but not regular

(C) context-sensitive but not context-free

(D) not recursively enumerable

Q23. Let L_1 and L_2 be two regular languages over the same alphabet. The intersection $L_1 \cap L_2$ is always: **[2 marks]**

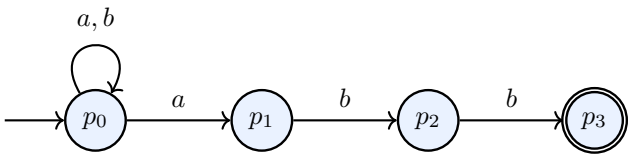
(A) regular

(B) context-free but not regular

(C) recursively enumerable but not recursive

(D) undecidable to recognise

Q24. The nondeterministic finite automaton (NFA) with 4 states is shown below. Using the subset (powerset) construction to convert it into an equivalent DFA, the maximum possible number of states in the resulting DFA is: **[2 marks]**



(A) 4

(B) 16

(C) 8

(D) 32

Q25. In a classical compiler, the phase that verifies type compatibility, checks that identifiers are declared before use, and enforces scope rules using the symbol table is the: **[2 marks]**

(A) lexical analyzer (scanner)

(B) semantic analyzer

(C) code generator

(D) syntax analyzer (parser)

Q26. Consider the grammar with start symbol S:

$$S \to A\,c, \qquad A \to a \mid \varepsilon.$$

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

(A) $\{a\}$

(B) $\{a, c\}$

(C) $\{c\}$

(D) $\{a, \varepsilon\}$

Q27. Which one of the following parsing techniques is a *bottom-up* parsing method? **[2 marks]**

(A) recursive-descent parsing

(B) LL(1) predictive parsing

(C) table-driven top-down parsing

(D) shift-reduce (LR) parsing

Q28. During code optimization, a compiler replaces the multiplication $x \times 2$ by the cheaper left-shift operation $x \ll 1$. This particular optimization is an example of: **[2 marks]**

(A) strength reduction

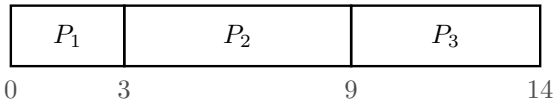
(B) dead-code elimination

(C) constant folding

(D) common subexpression elimination

Operating Systems & Databases

Q29. Three processes arrive together at time 0 in the order P_1, P_2, P_3 with CPU burst times $P_1 = 3$, $P_2 = 6$ and $P_3 = 5$ (all in ms). They are scheduled by non-preemptive First-Come-First-Served (FCFS), whose Gantt chart is shown. The average waiting time of the three processes is: **[2 marks]**



(A) 4 ms

(B) 6 ms

(C) 8 ms

(D) 5.33 ms

Q30. A counting semaphore S is initialised to 3. A sequence of operations is executed on it consisting of 5 successful `wait` (P) operations and 2 `signal` (V) operations. After all these operations complete, the value of S is: **[2 marks]**

(A) 3

(B) -2

(C) 0

(D) 5

Q31. To prevent deadlock, an operating system requires every process to request *all* the resources it will ever need at once, before it begins execution, and to hold none while waiting. This strategy directly attacks which one of the four Coffman (necessary) conditions? **[2 marks]**

(A) hold and wait

(B) mutual exclusion

(C) circular wait

(D) no preemption

Q32. A paging system has a logical (virtual) address space of 4 GB and a page size of 4 KB. The total number of pages in the virtual address space is: **[2 marks]**

(A) 2^{20}

(B) 2^{12}

(C) 2^{32}

(D) 2^{10}

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

$$1,\ 2,\ 3,\ 2,\ 4,\ 1,\ 2$$

the total number of page faults that occur is: [2 marks]

(A) 6

(B) 5

(C) 7

(D) 4

Q34. In a file system, a file is stored as a sequence of disk blocks in which each block, apart from the data, holds a pointer to the next block of the file. This file-allocation method is called: [2 marks]

(A) contiguous allocation

(B) indexed allocation

(C) linked allocation

(D) bit-vector allocation

Q35. In relational algebra, the unary operation that returns a relation containing only a specified subset of the *attributes* (columns) of the original relation, removing any duplicate rows, is: [2 marks]

(A) projection (π)

(B) selection (σ)

(C) rename (ρ)

(D) Cartesian product ($\times$)

Q36. A relation schema R is in second normal form (2NF) if it is in first normal form (1NF) and, in addition, every non-prime attribute of R is: [2 marks]

(A) transitively dependent on some candidate key

(B) itself a candidate key

(C) fully functionally dependent on every candidate key

(D) dependent only on a foreign key

Q37. Among the ACID properties of a database transaction, the property that guarantees a transaction is executed in an "all-or-nothing" manner — either every one of its operations takes effect or none of them does — is: **[2 marks]**

(A) durability

(B) atomicity

(C) isolation

(D) consistency

Q38. In SQL, after rows have been grouped with a `GROUP BY` clause, the clause used to filter the resulting groups according to an aggregate condition (for example, keeping only groups whose `COUNT(*)` exceeds a value) is the: **[2 marks]**

(A) `WHERE` clause

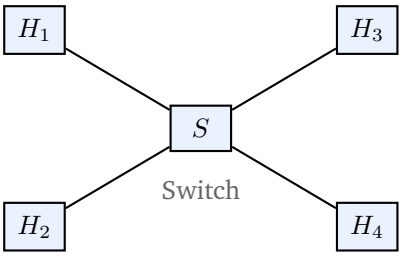
(B) `GROUP BY` clause

(C) `ORDER BY` clause

(D) `HAVING` clause

Computer Networks

Q39. In the local-area network shown below, four hosts are connected to a central switch S, which forwards frames by examining their MAC (physical) addresses and maintaining a MAC-address table. In the OSI reference model, such a switch operates primarily at the: **[2 marks]**



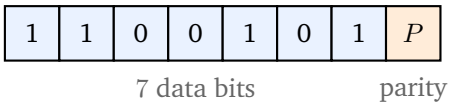
(A) network layer (layer 3)

(B) data-link layer (layer 2)

(C) physical layer (layer 1)

(D) transport layer (layer 4)

Q40. A 7-bit data word 1100101 is to be transmitted using a single *even-parity* bit appended at the end, as shown in the frame below. The value of the appended even-parity bit P is: **[2 marks]**



(A) 1

(B) it depends on the frame length

(C) 2

(D) 0

Q41. A Selective-Repeat ARQ (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 (and receiver) window size is: **[2 marks]**

(A) 2^{n-1}

(B) $2^n - 1$

(C) 2^n

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

Q42. A communication link has a bandwidth (transmission rate) of 1 Mbps (10^6 bits per second). The time taken purely to *transmit* (push out) a file of size 10^6 bits onto this link is: **[2 marks]**

(A) 0.1 s

(B) 1 s

(C) 10 s

(D) 0.5 s

Q43. A network administrator subnets a network by borrowing 3 bits from the host portion of the address for the subnet field. The number of distinct subnets that this borrowing creates is: [2 marks]

(A) 6

(B) 8

(C) 16

(D) 3

Q44. An IPv4 host is assigned the address $172.16.5.4$ (with the default classful interpretation). The address class to which this IP address belongs is: [2 marks]

(A) Class A

(B) Class C

(C) Class B

(D) Class D

Q45. At the transport layer of the TCP/IP suite, which protocol is connectionless, performs no handshake, and is therefore preferred for DNS lookups and real-time streaming where low overhead matters more than guaranteed delivery? [2 marks]

(A) TCP

(B) UDP

(C) ICMP

(D) ARP

Q46. An application-layer protocol resolves human-readable domain names (such as `example.com`) into their corresponding IP addresses, and uses well-known port 53. This protocol is: [2 marks]

(A) SMTP

(B) FTP

(C) HTTP

(D) DNS

Detailed Solutions

Q1.

Solution

Concept – hexadecimal to decimal by positional weights: Each hex digit is multiplied by a power of 16 according to its position, then the products are summed.

Step 1 – write the digit values: $2 = 2$, $\mathrm{A} = 10$, $\mathrm{F} = 15$.

Step 2 – attach positional weights $16^2, 16^1, 16^0$**:** $(2\mathrm{AF})_{16} = 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$

Step 3 – evaluate the first term: $2 \times 256 = 512$

Step 4 – evaluate the second term: $10 \times 16 = 160$

Step 5 – evaluate the third term: $15 \times 1 = 15$

Step 6 – add the three terms: $512 + 160 + 15 = 687$

Final Answer: $(2\mathrm{AF})_{16} = 687 \Rightarrow \boxed{\mathrm{D}}$

Answer: (D) Go Back to Q1

Q2.

Solution

Concept – read the gate network into a Boolean expression, then simplify: Each AND gate forms a product term and the OR gate sums the two products.

Step 1 – write the output of the top AND gate: Inputs A and B give $A \cdot B$.

Step 2 – write the output of the bottom AND gate: Inputs A and $\overline{B}$ give $A \cdot \overline{B}$.

Step 3 – combine them at the OR gate: $F = AB + A\overline{B}$

Step 4 – factor out the common literal A**:** $F = A(B + \overline{B})$

Step 5 – apply $B + \overline{B} = 1$**:** $F = A \cdot 1 = A$

Why the other options are wrong:

- A, B: the output does not depend on B at all.
- B, $\overline{A}$: this is the complement of the true output.
- D, AB: only one of the two product terms, ignoring $A\overline{B}$.

Final Answer: $F = A \Rightarrow \boxed{\mathrm{C}}$

Answer: (C) Go Back to Q2

Q3.

Solution

Concept – group the 1s on the K-map into the largest power-of-two block: A group covering a whole set of columns eliminates the variables that change inside it.

Step 1 – locate the 1s: Every cell in the two columns labelled $CD = 11$ and $CD = 10$ holds a 1 (eight cells in all).

Step 2 – identify what stays constant in that block: The columns $CD = 11$ and $CD = 10$ are exactly the columns in which the first bit $C = 1$.

Step 3 – identify what varies in that block: Across these eight cells the variables A, B and D each take both values, so they cannot appear in the product term.

Step 4 – write the product term: Only $C = 1$ is common to the whole group, so $F = C$.

Final Answer: $F = C \Rightarrow \boxed{\text{D}}$

Answer: (D) Go Back to Q3

Q4.

Solution

Concept – select lines of a multiplexer: A multiplexer with 2^k data inputs needs k select lines, because k bits address 2^k inputs.

Step 1 – state the requirement: Number of data inputs $= 16 = 2^k$.

Step 2 – solve for k: $2^k = 16 = 2^4$

Step 3 – read off the exponent: $k = 4$.

Step 4 – interpret: 4 select lines can address exactly 16 inputs, which matches the requirement.

Final Answer: 4 select lines $\Rightarrow \boxed{\text{A}}$

Answer: (A) Go Back to Q4

Q5.

Solution

Concept – range of an n-bit binary counter: An n-bit counter represents values from 0 up to $2^n - 1$, then rolls back over to 0.

Step 1 – find the number of distinct states: $n = 4$ bits give $2^4 = 16$ distinct states.

Step 2 – list the counted values: The counter runs through $0, 1, 2, \ldots$ up to the largest value.

Step 3 – compute the maximum value: maximum $= 2^4 - 1 = 16 - 1 = 15$.

Step 4 – interpret: After reaching 15 (1111_2) the next clock pulse rolls the count over to 0.

Final Answer: maximum count $= 15 \Rightarrow$ D

Answer: (D) Go Back to Q5

Q6.

Solution

Concept – input combinations of a NOR-based SR latch: For a NOR SR latch: S=0, R=0 holds; S=1, R=0 sets; S=0, R=1 resets; S=1, R=1 is invalid.

Step 1 – apply $S = 1, R = 1$ to the two NOR gates: A NOR gate outputs 0 whenever any input is 1, so both gate outputs are forced to 0.

Step 2 – observe the outputs: This makes $Q = 0$ and $\overline{Q} = 0$ at the same time.

Step 3 – spot the contradiction: Q and $\overline{Q}$ are supposed to be complements, but here they are equal, which is not a valid latch state.

Step 4 – describe what happens next: If both inputs then return to 0 together, the latch settles into an unpredictable (race-dependent) state, so the S=R=1 input is forbidden.

Final Answer: the state is forbidden / indeterminate $\Rightarrow$ C

Answer: (C) Go Back to Q6

Q7.

Solution

Concept – classify the addressing mode by where the operand lives: The key clue is that the operand *value* is carried inside the instruction itself.

Step 1 – read the rule given: The operand is a constant encoded directly in the instruction word.

Step 2 – match it to the standard names: When the instruction supplies the operand value itself (not an address), the mode is called immediate addressing.

Step 3 – rule out the others:

- Indirect: the instruction holds the address of a pointer to the operand, need-

ing extra memory accesses.
- Indexed: the effective address is a base plus an index register.
- Register-indirect: the operand's address is held in a register.

Step 4 – conclude: Only immediate addressing needs no memory access to obtain the operand.

Final Answer: immediate addressing $\Rightarrow$ $\boxed{\text{B}}$

Answer: (B) Go Back to Q7

Q8.

Solution

Concept – ideal pipeline speedup: For a large number of instructions, speedup = (time per instruction without pipelining) $\div$ (time per stage, i.e. the pipelined cycle time).

Step 1 – list the data: non-pipelined time per instruction $= 20$ ns; pipelined stage time $= 4$ ns; 5 stages.

Step 2 – find the pipelined throughput cycle: With a balanced pipeline one instruction completes every 4 ns.

Step 3 – compute the ideal speedup: speedup $= \dfrac{20 \text{ ns}}{4 \text{ ns}}$

Step 4 – evaluate: $\dfrac{20}{4} = 5$

Step 5 – cross-check with the number of stages: An ideal k-stage pipeline gives at most a speedup of $k = 5$, which matches.

Final Answer: ideal speedup $= 5 \Rightarrow \boxed{\text{A}}$

Answer: (A) Go Back to Q8

Q9.

Solution

Concept – direct-mapped cache placement: In a direct-mapped cache with C blocks, memory block B maps to cache index $B \bmod C$.

Step 1 – list the data: $C = 64$ cache blocks; memory block $B = 260$.

Step 2 – set up the modulo: cache index $= 260 \bmod 64$.

Step 3 – divide to find the quotient: $260 \div 64 = 4$ remainder r, since $4 \times 64 = 256$.

Step 4 – compute the remainder: $r = 260 - 256 = 4$.

Step 5 – state the mapping: memory block 260 maps to cache index 4.

Final Answer: cache block index $= 4 \Rightarrow$ B

Answer: (B) Go Back to Q9

Q10.

Solution

Concept – the C modulo operator: $x \,\%\, y$ yields the remainder when x is divided by y.

Step 1 – read the values: $a = 10$, $b = 4$.

Step 2 – evaluate $a \,\%\, b = 10 \,\%\, 4$: $10 = 2 \times 4 + 2$, so the remainder is 2.

Step 3 – evaluate $b \,\%\, a = 4 \,\%\, 10$: $4 = 0 \times 10 + 4$, so the remainder is 4.

Step 4 – add the two remainders: $a\%b + b\%a = 2 + 4 = 6$

Step 5 – state the printed value: The program prints 6.

Final Answer: the program prints $6 \Rightarrow$ A

Answer: (A) Go Back to Q10

Q11.

Solution

Concept – infix to postfix conversion honouring precedence: $*$ has higher precedence than $+$, so the multiplication is performed first and its operands are grouped before the addition.

Step 1 – identify the sub-expression with higher precedence: $B*C$ is evaluated before the addition.

Step 2 – write $B * C$ in postfix: operands first, then operator: $B\,C\,*$.

Step 3 – combine with the addition of A: $A + (B\,C\,*)$ becomes, in postfix, the two operands followed by $+$.

Step 4 – assemble the full postfix string: $A\ (B\,C\,*) + \ = \ A\,B\,C\,*\,+$

Step 5 – sanity check by re-parsing: Reading $A\,B\,C\ *\ +$ left to right: push A, B, C; apply $*$ to B, C giving $B*C$; apply $+$ to A and $B*C$. This reproduces $A + B * C$.

Final Answer: $A\,B\,C\ *\ + \Rightarrow$ D

Answer: (D) Go Back to Q11

Q12.

Solution

Concept – matching parentheses needs last-in-first-out access: The most recently opened bracket must be the first one closed, which is exactly the LIFO discipline of a stack.

Step 1 – state the checking rule: Scan left to right; push every opening bracket, and on each closing bracket pop and verify it matches.

Step 2 – note the access pattern needed: The bracket that must be matched next is always the most recently pushed one (last in, first out).

Step 3 – pick the structure that gives LIFO: Only a stack provides last-in-first-out access.

Step 4 – rule out the others: A queue is FIFO, a priority queue orders by key, and a circular linked list gives no inherent LIFO behaviour.

Final Answer: a stack $\Rightarrow$ D

Answer: (D) Go Back to Q12

Q13.

Solution

Concept – front insertion in a singly linked list with a head pointer: Because we already hold the head pointer, only a fixed number of pointer updates are needed.

Step 1 – allocate the new node: Create a node and store the value; this is constant work.

Step 2 – link the new node to the current head: Set newNode.next $=$ head.

Step 3 – update the head pointer: Set head $=$ newNode.

Step 4 – count the operations: The work is a constant number of pointer assignments, independent of the list length n, so the time is $O(1)$.

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

Answer: (C) Go Back to Q13

Q14.

Solution

Concept – preorder traversal order: Preorder visits (root, left subtree, right subtree) recursively.

Step 1 – visit the root: Output 5.

Step 2 – traverse the left subtree rooted at 3: Visit 3, then its left child 2, then its right child 4.

Left part $= 3,\ 2,\ 4$.

Step 3 – traverse the right subtree rooted at 8: Visit 8, then its left child 7, then its right child 9.

Right part $= 8,\ 7,\ 9$.

Step 4 – concatenate root, left, right: $5,\ 3,\ 2,\ 4,\ 8,\ 7,\ 9$.

Final Answer: $5\ 3\ 2\ 4\ 8\ 7\ 9 \Rightarrow \boxed{\text{A}}$

Answer: (A) Go Back to Q14

Q15.

Solution

Concept – cost of bottom-up build-heap: Although a single heapify can cost $O(\log n)$, most nodes are near the bottom and sift down only a little, giving a tighter total bound.

Step 1 – count nodes by height: At height h there are at most $\lceil n/2^{h+1} \rceil$ nodes, and each costs $O(h)$ to sift down.

Step 2 – write the total work as a sum: $T(n) = \sum_{h=0}^{\log n} \frac{n}{2^{h+1}} \cdot O(h)$

Step 3 – bound the series: $\sum_{h=0}^{\infty} \frac{h}{2^h}$ converges to a constant $(= 2)$.

Step 4 – multiply by n: $T(n) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n)$

Step 5 – conclude: Bottom-up build-heap runs in linear time $O(n)$ (unlike inserting n items one by one, which is $O(n \log n)$).

Final Answer: $O(n) \Rightarrow \boxed{\text{B}}$

Answer: (B) Go Back to Q15

Q16.

Solution

Concept – number of edges in a complete graph: K_n joins every pair of distinct vertices exactly once, so its edge count is the number of 2-element subsets, $\binom{n}{2}$.

Step 1 – write the formula: edges $= \binom{n}{2} = \dfrac{n(n-1)}{2}$.

Step 2 – substitute $n = 5$: edges $= \dfrac{5 \times 4}{2}$.

Step 3 – evaluate the numerator: $5 \times 4 = 20$.

Step 4 – divide by 2: $\dfrac{20}{2} = 10$.

Step 5 – interpret: K_5 has 10 edges, matching the figure.

Final Answer: 10 edges $\Rightarrow$ D

Answer: (D) Go Back to Q16

Q17.

Solution

Concept – the tight Θ bound keeps the dominant term: For a polynomial, the term of highest degree dominates as $n \to \infty$, and constant coefficients do not affect the Θ class.

Step 1 – identify the highest-degree term: In $3n^2 + 5n + 100$ the highest-degree term is $3n^2$.

Step 2 – drop the lower-order terms: For large n, $5n$ and 100 are negligible compared with $3n^2$.

Step 3 – drop the constant coefficient: $3n^2$ is $\Theta(n^2)$.

Step 4 – state the bound: $f(n) = \Theta(n^2)$.

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

Answer: (C) 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 – read off the parameters: $a = 2$, $b = 2$, $f(n) = n^2$.

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

Step 3 – compare $f(n)$ with n^1: $f(n) = n^2$ grows polynomially faster than n (it is $\Omega(n^{1+\varepsilon})$ with $\varepsilon = 1$).

Step 4 – check the regularity condition (Case 3): $a\, f(n/b) = 2\,(n/2)^2 = 2{\cdot}n^2/4 = n^2/2 \le \frac{1}{2} f(n)$, so the condition holds.

Step 5 – apply Case 3: $T(n) = \Theta(f(n)) = \Theta(n^2)$.

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

Answer: (B) Go Back to Q18

Q19.

Solution

Concept – merge sort's recurrence: Merge sort always splits into two equal halves and merges in linear time, regardless of the input arrangement.

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

Step 2 – apply the Master theorem: $a = 2$, $b = 2$, so $n^{\log_b a} = n^1 = n$, and $f(n) = \Theta(n) = \Theta(n^{\log_b a})$ (Case 2).

Step 3 – read the Case 2 result: $T(n) = \Theta(n \log n)$.

Step 4 – note the worst case: Because the split is always even and the merge is always linear, the best, average and worst cases are all $\Theta(n \log n)$.

Final Answer: $\Theta(n \log n) \Rightarrow$ A

Answer: (A) Go Back to Q19

Q20.

Solution

Concept – trace the DFA to characterise the accepted language: Follow the transitions from the start state and note when the machine is in the accepting state q_1.

Step 1 – list the transitions: From q_0: on a stay at q_0, on b go to q_1. From q_1: on b stay at q_1, on a go to q_0.

Step 2 – see what puts the machine in q_1: The machine is in q_1 exactly after reading a b; reading an a always drops it back to q_0.

Step 3 – decide acceptance: A string is accepted iff it finishes in q_1, i.e. iff its last symbol was a b.

Step 4 – state the language: $L =$ all strings over $\{a, b\}$ that end with the symbol

b.

Final Answer: strings ending with $b \Rightarrow$ C

Answer: (C) Go Back to Q20

Q21.

Solution

Concept – read a^*b^* as concatenation of two stars: a^* matches any block of a's (including none) and b^* matches any block of b's (including none); the concatenation puts the a-block strictly before the b-block.

Step 1 – describe the left factor a^*: zero or more a's.

Step 2 – describe the right factor b^*: zero or more b's.

Step 3 – combine by concatenation: every accepted string is some a's followed by some b's, with no b appearing before any a.

Step 4 – rule out the distractors: The counts of a's and b's need not be equal, the string need not contain ab (e.g. aaa or the empty string is allowed), and it is not restricted to one a and one b.

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 – placing $a^nb^nc^n$ in the Chomsky hierarchy: Matching three counts at once exceeds the power of a pushdown stack but is within a linear-bounded automaton.

Step 1 – test regularity: A finite automaton has no unbounded memory to count n, so L is not regular.

Step 2 – test context-freeness with the pumping lemma: A pushdown automaton's single stack can match two counts (a^nb^n) but cannot simultaneously enforce a third equal count of c's; the CFL pumping lemma fails for $a^nb^nc^n$, so L is not context-free.

Step 3 – show it is context-sensitive: L is generated by a context-sensitive grammar (equivalently accepted by a linear-bounded automaton), so it is context-sensitive.

Step 4 – conclude: L is context-sensitive but not context-free.

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

Answer: (C) Go Back to Q22

Q23.

Solution

Concept – closure of regular languages under intersection: The regular languages are closed under intersection.

Step 1 – take DFAs for both languages: Let M_1 accept L_1 and M_2 accept L_2.

Step 2 – form the product automaton: Build a DFA whose states are pairs (s_1, s_2), running M_1 and M_2 in parallel on the same input.

Step 3 – set the accepting condition: Accept a pair iff *both* components are accepting; this DFA recognises exactly $L_1 \cap L_2$.

Step 4 – conclude: Since a DFA recognises $L_1 \cap L_2$, the intersection is regular.

Final Answer: the intersection is regular $\Rightarrow$ A

Answer: (A) Go Back to Q23

Q24.

Solution

Concept – subset construction blow-up: Each DFA state corresponds to a *subset* of the NFA's states, so an n-state NFA yields a DFA with at most 2^n states.

Step 1 – count the NFA states: The NFA has 4 states: p_0, p_1, p_2, p_3.

Step 2 – count the possible subsets: The number of subsets of a 4-element set is 2^4.

Step 3 – evaluate: $2^4 = 16$.

Step 4 – interpret: The equivalent DFA has at most 16 states (some may be unreachable, but 16 is the upper bound).

Final Answer: at most 16 states $\Rightarrow$ B

Answer: (B) Go Back to Q24

Q25.

Solution

Concept – responsibilities of the compiler phases: Type checking, declaration-before-use and scope enforcement are meaning-level checks, done after parsing.

Step 1 – eliminate the scanner: The lexical analyzer only groups characters into tokens; it does not know about types.

Step 2 – eliminate the parser: The syntax analyzer checks grammatical structure (the parse tree) but not type compatibility.

Step 3 – match the described tasks: Type checking, use-before-declaration detection and scope rules are exactly the job of the semantic analyzer, which consults the symbol table.

Step 4 – eliminate the code generator: Code generation happens later and assumes the program is already semantically valid.

Final Answer: semantic analyzer $\Rightarrow$ B

Answer: (B) Go Back to Q25

Q26.

Solution

Concept – FOLLOW set: $\mathrm{FOLLOW}(A)$ is the set of terminals that can appear immediately to the right of A in some sentential form.

Step 1 – find where A appears on a right-hand side: A appears only in $S \to A\,c$.

Step 2 – read what follows A there: The symbol immediately after A is the terminal c.

Step 3 – add FIRST of the following string: $\mathrm{FIRST}(c) = \{c\}$, so $c \in \mathrm{FOLLOW}(A)$.

Step 4 – check whether \$ (end marker) belongs: A is never at the very end of a production (it is always followed by c), so nothing else is added.

Step 5 – state the set: $\mathrm{FOLLOW}(A) = \{c\}$. (Note $A \to \varepsilon$ affects A's own FIRST, not its FOLLOW.)

Final Answer: $\{c\} \Rightarrow$ C

Answer: (C) Go Back to Q26

Q27.

Solution

Concept – top-down versus bottom-up parsing: Bottom-up parsers build the parse tree from the leaves (tokens) up to the start symbol by repeatedly reducing handles.

Step 1 – classify each option: Recursive-descent, LL(1) predictive and table-driven predictive parsing are all top-down methods.

Step 2 – identify the odd one out: Shift-reduce (LR) parsing works by shifting tokens onto a stack and reducing them, building the tree from the bottom up.

Step 3 – conclude: The only bottom-up technique listed is shift-reduce (LR) parsing.

Final Answer: shift-reduce (LR) parsing $\Rightarrow$ $\boxed{\text{D}}$

Answer: (D) Go Back to Q27

Q28.

Solution

Concept – strength reduction: Strength reduction replaces an expensive operation by an equivalent cheaper one.

Step 1 – examine the transformation: $x \times 2$ is rewritten as $x \ll 1$ (a left shift by one bit).

Step 2 – verify the equivalence: Shifting a binary number left by one position doubles it, so $x \ll 1 = x \times 2$.

Step 3 – classify it: Swapping a costly multiply for a cheap shift is the definition of strength reduction.

Step 4 – rule out the others: It is not dead-code elimination (the value is used), not constant folding (no compile-time constants are combined), and not common subexpression elimination (no repeated expression is removed).

Final Answer: strength reduction $\Rightarrow$ $\boxed{\text{A}}$

Answer: (A) Go Back to Q28

Q29.

Solution

Concept – waiting time under FCFS: With all arrivals at time 0, a process's waiting time equals the sum of the burst times of the processes scheduled before it.

Step 1 – fix the service order: FCFS runs them in arrival order P_1, P_2, P_3.

Step 2 – waiting time of P_1: P_1 starts at time 0, so its waiting time is 0.

Step 3 – waiting time of P_2: P_2 starts after P_1's burst of 3, so its waiting time is 3 ms.

Step 4 – waiting time of P_3: P_3 starts after P_1 and P_2, i.e. after $3 + 6 = 9$, so its waiting time is 9 ms.

Step 5 – average the waiting times: $\text{avg} = \dfrac{0+3+9}{3} = \dfrac{12}{3} = 4$ ms.

Final Answer: average waiting time $= 4$ ms $\Rightarrow$ A

Answer: (A) Go Back to Q29

Q30.

Solution

Concept – counting semaphore arithmetic: Each `wait` (P) decrements the semaphore by 1; each `signal` (V) increments it by 1.

Step 1 – start value: $S = 3$.

Step 2 – apply the 5 wait operations: Total decrement $= 5$, so $3 - 5 = -2$.

Step 3 – apply the 2 signal operations: Total increment $= 2$, so $-2 + 2 = 0$.

Step 4 – combine in one expression: $S = 3 - 5 + 2 = 0$.

Step 5 – interpret: The final semaphore value is 0.

Final Answer: $S = 0 \Rightarrow$ C

Answer: (C) Go Back to Q30

Q31.

Solution

Concept – the four Coffman conditions and how to break them: Deadlock needs mutual exclusion, hold and wait, no preemption and circular wait to all hold; removing any one prevents deadlock.

Step 1 – read the policy: Each process must acquire every resource it needs at once, before starting, and holds nothing while waiting.

Step 2 – match it to a condition: "Holding some resources while waiting for others" is precisely the hold-and-wait condition.

Step 3 – see why it is broken: By demanding all resources up front, a process never holds one resource while waiting for another, so hold and wait cannot occur.

Step 4 – conclude: The strategy attacks the hold-and-wait condition.

Final Answer: hold and wait $\Rightarrow$ A

Answer: (A) Go Back to Q31

Q32.

Solution

Concept – number of pages = address-space size ÷ page size: Both quantities are powers of two, so the division is a subtraction of exponents.

Step 1 – express the virtual address space in bytes: $4\text{ GB} = 2^2 \times 2^{30} = 2^{32}$ bytes.

Step 2 – express the page size in bytes: $4\text{ KB} = 2^2 \times 2^{10} = 2^{12}$ bytes.

Step 3 – divide to get the number of pages: number of pages $= \dfrac{2^{32}}{2^{12}} = 2^{32-12}$.

Step 4 – simplify the exponent: $2^{32-12} = 2^{20}$.

Step 5 – interpret: The virtual address space contains 2^{20} pages.

Final Answer: 2^{20} pages $\Rightarrow$ A

Answer: (A) Go Back to Q32

Q33.

Solution

Concept – LRU replacement: On a fault with all frames full, evict the page that was used least recently.

Step 1 – reference 1**:** frames empty $\rightarrow$ fault; frames $\{1\}$.

Step 2 – reference 2**:** not present $\rightarrow$ fault; frames $\{1, 2\}$.

Step 3 – reference 3**:** not present $\rightarrow$ fault; frames $\{1, 2, 3\}$ (recency order oldest$\rightarrow$newest: $1, 2, 3$).

Step 4 – reference 2**:** present $\rightarrow$ hit; recency becomes $1, 3, 2$.

Step 5 – reference 4**:** not present, frames full $\rightarrow$ fault; evict LRU $= 1$; frames $\{3, 2, 4\}$, recency $3, 2, 4$.

Step 6 – reference 1**:** not present $\rightarrow$ fault; evict LRU $= 3$; frames $\{2, 4, 1\}$, recency $2, 4, 1$.

Step 7 – reference 2**:** present $\rightarrow$ hit; recency becomes $4, 1, 2$.

Step 8 – count the faults: faults occur at references $1, 2, 3, 4, 1 = 5$ faults (two hits at the repeated 2's).

Final Answer: 5 page faults $\Rightarrow$ $\boxed{\text{B}}$

Answer: (B) Go Back to Q33

Q34.

Solution

Concept – file-allocation methods: The three classical methods are contiguous, linked and indexed allocation.

Step 1 – read the description: Each block stores data plus a pointer to the next block of the file.

Step 2 – match it to a method: Chaining blocks together through next-pointers is exactly linked allocation.

Step 3 – rule out the others:

- Contiguous: the file occupies a run of consecutive blocks, with no per-block pointers.
- Indexed: all block addresses are gathered into one index block.
- Bit-vector: a free-space management structure, not a file layout.

Final Answer: linked allocation $\Rightarrow$ $\boxed{\text{C}}$

Answer: (C) Go Back to Q34

Q35.

Solution

Concept – relational-algebra projection: Projection π selects a subset of columns; since relations are sets, duplicate rows in the result are removed.

Step 1 – read the description: The operation keeps only chosen attributes (columns) and eliminates duplicate rows.

Step 2 – match the symbol: This is the projection operator π.

Step 3 – contrast with selection: Selection σ chooses *rows* matching a predicate while keeping all columns, which is the opposite of what is described.

Step 4 – rule out the rest: Rename ρ only changes names, and Cartesian product $\times$ combines two relations.

Final Answer: projection (π) $\Rightarrow$ A

Answer: (A) Go Back to Q35

Q36.

Solution

Concept – second normal form (2NF): 2NF removes *partial* dependencies: a non-prime attribute must depend on the whole of a candidate key, not just part of it.

Step 1 – state the 2NF requirement: The relation is in 1NF and no non-prime attribute is partially dependent on any candidate key.

Step 2 – restate "no partial dependency" positively: Every non-prime attribute must be fully functionally dependent on every candidate key.

Step 3 – match the option: This matches "fully functionally dependent on every candidate key".

Step 4 – rule out the others: Removing transitive dependencies is 3NF; being a candidate key or depending on a foreign key is unrelated to the 2NF definition.

Final Answer: fully functionally dependent on every candidate key $\Rightarrow$ C

Answer: (C) Go Back to Q36

Q37.

Solution

Concept – the ACID properties: Atomicity is the "all-or-nothing" guarantee for a transaction.

Step 1 – read the described guarantee: Either all operations of the transaction take effect, or none do.

Step 2 – match it to an ACID property: This all-or-nothing behaviour is the definition of atomicity.

Step 3 – distinguish from the others:

- Durability: committed effects survive crashes.
- Isolation: concurrent transactions do not interfere.
- Consistency: the database moves from one valid state to another.

Final Answer: atomicity $\Rightarrow$ B

Answer: (B) Go Back to Q37

Q38.

Solution

Concept – filtering groups in SQL: `WHERE` filters individual rows before grouping; `HAVING` filters the groups formed by `GROUP BY`.

Step 1 – note when aggregates are known: Aggregate values such as `COUNT(*)` exist only after the rows have been grouped.

Step 2 – eliminate `WHERE`: `WHERE` is applied before grouping and cannot reference group aggregates.

Step 3 – identify the correct clause: `HAVING` is evaluated after `GROUP BY` and can test conditions on aggregates, e.g. `HAVING COUNT(*) > k`.

Step 4 – rule out the rest: `ORDER BY` only sorts the output, and `GROUP BY` forms the groups but does not filter them.

Final Answer: the `HAVING` clause $\Rightarrow \boxed{\text{D}}$

Answer: (D) Go Back to Q38

Q39.

Solution

Concept – OSI layer of a switch: A device is classified by the addresses it uses to forward data.

Step 1 – read what the switch uses: It forwards frames using MAC (physical) addresses and a MAC-address table.

Step 2 – recall which layer owns MAC addresses: MAC addressing and frames belong to the data-link layer (layer 2).

Step 3 – conclude: A switch therefore operates primarily at the data-link layer.

Step 4 – rule out the others: A router uses layer-3 (IP) addresses, a hub/repeater works at layer 1, and layer 4 is the transport layer.

Final Answer: data-link layer (layer 2) $\Rightarrow \boxed{\text{B}}$

Answer: (B) Go Back to Q39

Q40.

Solution

Concept – even parity: The parity bit is chosen so that the total number of 1s in (data + parity) is even.

Step 1 – count the 1s in the data word 1100101: bits $= 1, 1, 0, 0, 1, 0, 1$.

Step 2 – add them up: $1 + 1 + 0 + 0 + 1 + 0 + 1 = 4$.

Step 3 – check the parity of the count: 4 is already even.

Step 4 – choose the parity bit: To keep the total even, the parity bit must add nothing, so $P = 0$.

Step 5 – verify: data + parity has $4 + 0 = 4$ ones, which is even. Correct.

Final Answer: $P = 0 \Rightarrow \boxed{\text{D}}$

Answer: (D) Go Back to Q40

Q41.

Solution

Concept – window size limit for Selective Repeat: In Selective Repeat the sender and receiver windows must not overlap in the sequence-number space, which caps the window at half the space.

Step 1 – find the size of the sequence-number space: An n-bit field gives 2^n distinct sequence numbers.

Step 2 – apply the non-overlap requirement: To keep the sender and receiver windows disjoint, each window can be at most half of the space.

Step 3 – compute the maximum window: maximum window $= \dfrac{2^n}{2} = 2^{n-1}$.

Step 4 – contrast with Go-Back-N: Go-Back-N allows $2^n - 1$; Selective Repeat is stricter at 2^{n-1}.

Final Answer: $2^{n-1} \Rightarrow \boxed{\text{A}}$

Answer: (A) Go Back to Q41

Q42.

Solution

Concept – transmission (transfer) time: transmission time $= \dfrac{\text{message size (bits)}}{\text{bandwidth (bits/s)}}$.

Step 1 – list the data: message size $= 10^6$ bits; bandwidth $= 1$ Mbps $= 10^6$ bits/s.

Step 2 – substitute into the formula: time $= \dfrac{10^6 \text{ bits}}{10^6 \text{ bits/s}}$.

Step 3 – simplify: $\dfrac{10^6}{10^6} = 1$.

Step 4 – attach units: time $= 1$ second.

Final Answer: 1 s $\Rightarrow$ B

Answer: (B) Go Back to Q42

Q43.

Solution

Concept – number of subnets from borrowed bits: Borrowing s bits for the subnet field yields 2^s subnets.

Step 1 – read the number of borrowed bits: $s = 3$ bits are taken from the host portion.

Step 2 – apply the formula: number of subnets $= 2^s = 2^3$.

Step 3 – evaluate: $2^3 = 8$.

Step 4 – note on conventions: With classical subnetting all 8 subnet patterns (including all-zeros and all-ones) are usable on modern equipment, giving 8 subnets.

Final Answer: 8 subnets $\Rightarrow$ B

Answer: (B) Go Back to Q43

Q44.

Solution

Concept – classful IP address ranges (first octet): Class A: 1–126; Class B: 128–191; Class C: 192–223; Class D: 224–239.

Step 1 – read the first octet: The address is $172.16.5.4$, so the first octet is 172.

Step 2 – locate 172 in the ranges: $128 \leq 172 \leq 191$, which is the Class B range.

Step 3 – conclude: $172.16.5.4$ is a Class B address.

Final Answer: Class B $\Rightarrow$ C

Answer: (C) Go Back to Q44

Q45.

Solution

Concept – connectionless transport protocol: UDP is connectionless and lightweight; TCP is connection-oriented and reliable.

Step 1 – match the described properties: No handshake, connectionless and low overhead point to UDP.

Step 2 – confirm the use cases: DNS queries and real-time streaming favour UDP because speed matters more than guaranteed, in-order delivery.

Step 3 – rule out the others: TCP is connection-oriented; ICMP is a network-layer control protocol; ARP resolves IP-to-MAC addresses and is not a transport protocol.

Final Answer: UDP $\Rightarrow$ B

Answer: (B) Go Back to Q45

Q46.

Solution

Concept – well-known application-layer protocols and ports: DNS (Domain Name System) maps domain names to IP addresses on port 53.

Step 1 – read the described function: Resolving names like `example.com` into IP addresses.

Step 2 – match the port: The service on well-known port 53 is DNS.

Step 3 – rule out the others: SMTP (mail) uses port 25, FTP control uses port 21, and HTTP uses port 80; none of these performs name resolution.

Final Answer: DNS $\Rightarrow$ D

Answer: (D) Go Back to Q46

Answer Key

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