

Design and Analysis of Algorithms (DAA)

Unit-wise MCQ Bank — BEU Previous Year Questions + Extra Important Questions

Correct option shown in green | Simple explanation given under every question

Unit 1: Introduction to Algorithms & Asymptotic Analysis

Part A — MCQs from BEU Previous Year Papers (5 Questions)

Q1. An algorithm is said to be finite if [BEU 2025]

- a) It uses finite memory
- b) It produces correct output
- c) It has finite input
- d) It terminates after a finite number of steps (Correct)**

Explanation: Finiteness is one of the basic rules an algorithm must follow — it must stop after a limited number of steps, however large. If it never stops, it's not an algorithm, it's just an endless process.

Q2. In Master's theorem, if $f(n) = \Theta(n^{\log_b a})$, then the solution of the recurrence is [BEU 2025]

- a) $\Theta(f(n))$
- b) $\Theta(f(n) \cdot \log n)$ (Correct)**
- c) $\Theta(n)^{(\log_b a)}$
- d) $\Theta(\log n)$

Explanation: This is Case 2 of Master's theorem. When $f(n)$ matches $n^{\log_b a}$ exactly, you don't just get $\Theta(n^{\log_b a})$ — you get an EXTRA $\log n$ factor on top, so the real answer is $\Theta(f(n) \cdot \log n)$, i.e. $\Theta(n^{\log_b a} \cdot \log n)$.

Q3. Complexity of the recurrence relation $T(n) = 8T(n/2) + n^2$ is [BEU 2023]

- a) $O(n)$
- b) $O(n^2)$
- c) $O(\log_2 n)$
- d) $O(n^3)$ (Correct)**

Explanation: Here $a=8$, $b=2$, so $n^{\log_b a} = n^{\log_2 8} = n^3$. Compare with $f(n)=n^2$, which grows slower than n^3 (this is Case 1 of Master's theorem), so the recurrence is dominated by the recursive part: $\Theta(n^3)$.

Q4. An algorithm is made up of two independent time complexities $f(n)$ and $g(n)$, run one after another. Then the complexity of the algorithm is of the order of [BEU 2023]

- a) $f(n) \times g(n)$
- b) $\max(f(n), g(n))$
- c) $\min(f(n), g(n))$
- d) $f(n) + g(n)$ (Correct)**

Explanation: When two independent parts of a program run one after the other (not nested), their times simply add up: total time = $f(n) + g(n)$. Since only the faster-growing term matters for big n , this is the same as saying it's $\Theta(\max(f(n), g(n)))$.

Q5. The choice of the polynomial class has led to the development of an extensive theory called [BEU 2022]

- a) Computational complexity (Correct)**
- b) Time complexity
- c) Problem complexity
- d) Decision complexity

Explanation: Computational complexity theory is the whole field that studies how the resources (time, space) an algorithm needs grow with input size, and classifies problems (like P, NP) based on that growth.

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (4 Questions)

Q6. Which notation gives an upper bound on the growth rate of a function's running time? [Extra – GATE pattern]

- a) Ω (Omega)
- b) Θ (Theta)
- c) O (Big-O) (Correct)**
- d) ω (little-o)

Explanation: Big-O describes the worst case — it tells you the running time will never grow FASTER than this bound, no matter what the input is.

Q7. Using Master's theorem, the recurrence $T(n) = 4T(n/2) + n$ has complexity [Extra – GATE pattern]

- a) $\Theta(n)$
- b) $\Theta(n \log n)$
- c) $\Theta(n^2)$ (Correct)**
- d) $\Theta(n^2 \log n)$

Explanation: Here $a=4$, $b=2$, so $n^{(\log_b a)} = n^2$. Since $f(n)=n$ grows slower than n^2 (Case 1), the recursive calls dominate, giving $\Theta(n^2)$.

Q8. The recursion tree method is mainly used to [Extra – GATE pattern]

- a) Prove that an algorithm is correct
- b) Guess and verify the solution of a recurrence relation (Correct)**
- c) Sort an array of numbers
- d) Find the shortest path in a graph

Explanation: A recursion tree lets you draw out each level of recursive calls and add up the work done at every level — this helps you guess a formula for the total time, which you can then confirm using substitution.

Q9. The best-case time complexity of an algorithm describes [Extra – GATE pattern]

- a) The maximum time it can ever take
- b) The minimum time it takes, over all possible inputs of size n (Correct)**
- c) The average time over all inputs
- d) The time it takes on the largest possible input

Explanation: Best case means: out of every possible input of a given size, which one lets the algorithm finish the fastest? That fastest time is the best-case complexity.

Unit 2: Divide & Conquer, Sorting Techniques & Heaps

Part A — MCQs from BEU Previous Year Papers (9 Questions)

Q1. What is the maximum number of swaps that can be performed in the Selection Sort algorithm? [BEU 2025]

- a) $n - 1$ (Correct)**
- b) n
- c) 1
- d) None of the above

Explanation: Selection sort makes one pass for each position except the last (which is automatically correct once everything else is placed), so at most $n-1$ swaps happen — one swap per pass at most.

Q2. The time complexity of Strassen's Matrix Multiplication is approximately [BEU 2025]

- a) $O(n^3)$
- b) $O(n^2)$**

c) $O(n^{2.81})$ (Correct)

d) $O(n \log n)$

Explanation: Normal matrix multiplication takes $O(n^3)$. Strassen's clever divide-and-conquer trick reduces the number of multiplications needed, bringing the complexity down to about $O(n^{2.81})$ (exactly $n^{\log_2 7}$).

Q3. Any decision tree that sorts n elements has height [BEU 2022]

a) $\Omega(\lg n)$

b) $\Omega(n)$

c) $\Omega(n \lg n)$ (Correct)

d) $\Omega(n^2)$

Explanation: This is the famous lower bound for comparison-based sorting: no matter how clever your sorting algorithm is (as long as it only compares elements), it can never do better than about $n \log n$ comparisons in the worst case.

Q4. The complexity of the binary search algorithm is [BEU 2022]

a) $O(n)$

b) $O(\log n)$ (Correct)

c) $O(n^2)$

d) $O(n \log n)$

Explanation: Binary search cuts the search space in half at every step, so it only needs about $\log_2 n$ steps to find an element — much faster than checking one by one.

Q5. In the linear search algorithm, the worst case occurs when [BEU 2022]

a) The item is somewhere in the middle of the array

b) The item is not in the array at all

c) The item is the last element in the array

d) The item is the last element in the array or is not there at all (Correct)

Explanation: Linear search checks elements one by one from the start. The most comparisons happen either when the item is the very last one checked, or when it isn't there at all (so every single element gets checked) — both take n comparisons.

Q6. Given an unsorted array where every element is at most k positions away from where it would be in the sorted array, which sorting algorithm can be easily modified for this, and what is the obtainable time complexity? [BEU 2022]

- a) Insertion sort with $O(kn)$
- b) Heap sort with $O(n \log k)$ (Correct)**
- c) Quick sort with $O(k \log k)$
- d) Merge sort with $O(k \log k)$

Explanation: This is the classic 'k-sorted array' problem. You keep a Min-Heap of just $k+1$ elements — since any element can only be at most k steps out of place, the correct next-smallest element is always inside that small heap. This gives $O(n \log k)$, much faster than fully sorting.

Q7. The approach of dynamic programming is similar to [BEU 2022]

- a) Parsing
- b) Hash table
- c) Divide and conquer algorithm (Correct)**
- d) Greedy algorithm

Explanation: Both DP and Divide-and-Conquer break a big problem into smaller subproblems and combine their answers. The key difference is DP saves (memoizes) answers to subproblems that repeat, while plain D&C usually doesn't need to (its subproblems don't overlap).

Q8. In the divide and conquer process, breaking the problem into smaller sub-problems is the responsibility of [BEU 2022]

- a) Divide / break (Correct)**
- b) Sorting / divide
- c) Conquer / solve
- d) Merge / combine

Explanation: Divide-and-conquer has 3 clear steps: Divide (split the problem into smaller pieces), Conquer (solve each piece, usually recursively), and Combine (merge the results). Splitting the problem is the 'Divide' step's job.

Q9. A priority queue is implemented as a Max-heap. It initially has 5 elements with level order 10, 8, 5, 3, 2. Two new elements '1' and '7' are inserted in that order. What is the level order traversal after insertion? [BEU 2022]

- a) 10, 8, 7, 5, 3, 2, 1
- b) 10, 8, 7, 2, 3, 1, 5
- c) 10, 8, 7, 1, 2, 3, 5
- d) 10, 8, 7, 3, 2, 1, 5 (Correct)**

Explanation: Inserting 1: it goes to the end and is smaller than its parent (5), so it stays put $\rightarrow [10, 8, 5, 3, 2, 1]$. Inserting 7: it goes to the end, and since 7 is bigger

than its parent (5), it 'bubbles up' and swaps with 5; 7 is still smaller than the root 10, so it stops there → [10,8,7,3,2,1,5].

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (5 Questions)

Q10. Merge sort's time complexity in the worst case is [Extra – GATE pattern]

- a) $O(n)$
- b) $O(n \log n)$ (Correct)**
- c) $O(n^2)$
- d) $O(\log n)$

Explanation: Merge sort always splits the array in half ($\log n$ levels) and merges everything at each level in $O(n)$ time, giving $O(n \log n)$ no matter how the input is arranged — this is why it's called a 'worst case efficient' sort.

Q11. Which of these sorting algorithms has the best average-case time complexity? [Extra – GATE pattern]

- a) Bubble sort
- b) Insertion sort
- c) Quick sort (Correct)**
- d) Selection sort

Explanation: Quick sort's average case is $O(n \log n)$, which is much faster than the $O(n^2)$ average case of bubble, insertion, and selection sort.

Q12. Karatsuba's algorithm is a divide-and-conquer trick used to speed up [Extra – GATE pattern]

- a) Sorting an array
- b) Matrix multiplication
- c) Multiplying two large integers/polynomials (Correct)**
- d) Searching a sorted list

Explanation: Normal long multiplication of two n -digit numbers takes $O(n^2)$. Karatsuba's algorithm smartly reduces the number of smaller multiplications needed, bringing the complexity down to about $O(n^{1.585})$.

Q13. In a Min-Heap, the smallest element is always found at [Extra – GATE pattern]

- a) The last leaf
- b) Any leaf node
- c) The root (Correct)**
- d) The middle of the array

Explanation: That's the whole point of a Min-Heap's structure — every parent is smaller than its children, so this rule guarantees the smallest value 'bubbles up' all the way to the top (the root).

Q14. The linear-time selection algorithm (Median of Medians) finds the k-th smallest element of an unsorted array in *[Extra – GATE pattern]*

- a) $O(n \log n)$
- b) $O(n^2)$
- c) $O(n)$ (Correct)**
- d) $O(\log n)$

Explanation: Even though it sounds like you'd need to sort first ($O(n \log n)$), the Median-of-Medians technique cleverly picks a good 'pivot' so that it can find the k-th smallest value in straight-line $O(n)$ time, without fully sorting.

Unit 3: Greedy Method: MST, Knapsack, Job Sequencing & Huffman Coding

Part A — MCQs from BEU Previous Year Papers (5 Questions)

Q1. Which of the following problems can NOT be solved optimally by a greedy approach? *[BEU 2025]*

- a) Minimum Cost Spanning Tree
- b) Fractional Knapsack
- c) Job Sequencing with Deadlines
- d) 0/1 Knapsack (Correct)**

Explanation: In 0/1 Knapsack you must take a WHOLE item or leave it (no breaking items into pieces), so simply grabbing the 'best value per weight' item first can lead you down a path that isn't actually optimal. It needs Dynamic Programming to guarantee the best answer.

Q2. In the Branch-and-Bound approach for the Travelling Salesman Problem (TSP), bounding is used to *[BEU 2025]*

- a) Find the exact optimal solution immediately
- b) Convert TSP into a dynamic programming problem
- c) Prune paths that cannot yield better solutions (Correct)**
- d) Sort cities by distance

Explanation: Bounding calculates a 'best possible' estimate for a partial path. If that estimate is already worse than the best complete solution found so far,

there's no point exploring further down that path — so it gets cut off (pruned), saving a lot of wasted work.

Q3. The fractional knapsack problem can be solved by using [BEU 2023]

- a) Greedy method (Correct)**
- b) Divide and conquer method
- c) Dynamic programming
- d) None of these

Explanation: Because you're allowed to take fractions of items, always picking the item with the best value-to-weight ratio first works perfectly and gives the optimal answer — this is a textbook example of the greedy method succeeding.

Q4. Kruskal's algorithm uses _____ and Prim's algorithm uses _____ in determining the MST. [BEU 2023]

- a) Edges, vertex (Correct)**
- b) Vertex, edges
- c) Edges, edges
- d) Vertex, vertex

Explanation: Kruskal's algorithm looks at the whole list of EDGES, sorted by weight, and keeps adding the smallest one that doesn't form a cycle. Prim's algorithm instead grows the tree one VERTEX at a time, always attaching the nearest new vertex.

Q5. Which of the following standard algorithms is NOT a greedy algorithm? [BEU 2023]

- a) Dijkstra's shortest path algorithm
- b) Kruskal's algorithm
- c) Bellman-Ford shortest path algorithm (Correct)**
- d) Prim's algorithm

Explanation: Dijkstra's, Kruskal's, and Prim's algorithms are all classic greedy algorithms — they always make the locally best choice. Bellman-Ford, on the other hand, repeatedly relaxes ALL edges over several rounds, which is a dynamic-programming style approach, not greedy.

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (4 Questions)

Q6. Huffman coding is an example of which algorithmic strategy? [Extra – GATE pattern]

- a) Dynamic programming
- b) Greedy method (Correct)**

- c) Divide and conquer
- d) Backtracking

Explanation: Huffman coding always merges the two characters/nodes with the lowest frequency first, at every single step — a locally greedy choice that is proven to build a globally optimal (shortest total length) prefix code.

Q7. Which algorithm builds an MST by always picking the smallest edge that doesn't create a cycle, examining the whole graph's edge list? *[Extra – GATE pattern]*

- a) Prim's algorithm
- b) Kruskal's algorithm (Correct)**
- c) Dijkstra's algorithm
- d) Bellman-Ford algorithm

Explanation: This is exactly how Kruskal's algorithm works — sort all edges by weight, then keep adding the cheapest ones as long as they don't form a cycle, using a Union-Find structure to check for cycles quickly.

Q8. The Job Sequencing with Deadlines problem (maximize profit, one job per slot) is optimally solved using *[Extra – GATE pattern]*

- a) Dynamic programming only
- b) A greedy approach (sort by profit, place each job as late as possible before its deadline) (Correct)**
- c) Backtracking only
- d) Brute force only

Explanation: You sort jobs by profit (highest first), then for each job try to place it in the latest free slot before its deadline. This simple greedy rule is proven to give the maximum total profit.

Q9. In Huffman coding, at each step the two nodes with the _____ frequencies are merged first. *[Extra – GATE pattern]*

- a) Highest
- b) Lowest (Correct)**
- c) Equal
- d) Random

Explanation: Merging the two least-frequent nodes first keeps the rarest characters deepest in the tree (longest codes) and the most common characters closest to the root (shortest codes) — which is exactly what makes the total encoding length minimal.

Unit 4: Dynamic Programming: Applications & Heuristics

Part A — MCQs from BEU Previous Year Papers (3 Questions)

Q1. In the Matrix Chain Multiplication problem, the DP table stores [BEU 2025]

- a) The sum of matrix elements
- b) Minimum number of scalar multiplications needed for each subchain (Correct)**
- c) Maximum determinant of the matrix chain
- d) Sequence of matrices in ascending order

Explanation: The DP table $m[i][j]$ holds the cheapest (minimum) number of multiplications needed to multiply the chain of matrices from position i to j — so you can find the best way to group/parenthesize the multiplications.

Q2. In the 0/1 Knapsack DP solution, the state $K[i][w]$ represents [BEU 2025]

- a) Maximum profit using the first i items for weight limit w (Correct)**
- b) Total number of items used
- c) Minimum weight required to achieve profit i
- d) Fraction of items included

Explanation: $K[i][w]$ is built up item by item: it stores the best possible total value you can get by choosing among only the first i items, without going over a total weight of w .

Q3. The approach of dynamic programming is similar to divide and conquer, but with the key difference that DP is used when a problem can be solved by combining optimal solutions to OVERLAPPING sub-problems, whereas if the sub-problems are non-overlapping, the strategy is called [BEU 2022]

- a) Dynamic programming
- b) Greedy
- c) Divide and conquer (Correct)**
- d) Recursion

Explanation: The real difference between the two: Divide-and-Conquer works best when the smaller pieces of the problem don't repeat/overlap (like in merge sort). When the same smaller subproblem shows up again and again, you need Dynamic Programming to avoid solving it repeatedly.

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (4 Questions)

Q4. Which of these problems CANNOT be solved optimally with a greedy approach but CAN be solved with dynamic programming? [Extra – GATE pattern]

- a) Fractional Knapsack
- b) Minimum Spanning Tree
- c) 0/1 Knapsack (Correct)**
- d) Huffman Coding

Explanation: 0/1 Knapsack requires you to consider whole combinations of items together (because you can't split an item), so a simple greedy 'best first' choice can miss the true optimal — but a DP table that checks every possibility systematically always gets it right.

Q5. Dynamic programming works best when a problem has *[Extra – GATE pattern]*

- a) No repeating subproblems
- b) Overlapping subproblems and optimal substructure (Correct)**
- c) Only one possible solution
- d) No recursive structure at all

Explanation: 'Overlapping subproblems' means the same smaller calculation is needed again and again — DP saves (memoizes) the answer the first time so it never has to be recomputed. 'Optimal substructure' means the best overall answer can be built from the best answers of its smaller pieces.

Q6. The Longest Common Subsequence (LCS) of two strings of length m and n can be computed by dynamic programming in *[Extra – GATE pattern]*

- a) $O(m + n)$
- b) $O(mn)$ (Correct)**
- c) $O(m^n)$
- d) $O(\log(mn))$

Explanation: LCS is solved by filling a DP table of size $m \times n$, where each cell takes constant time to compute using the two cells before it — so the total time is proportional to the size of the table, $O(mn)$.

Q7. The Rod Cutting problem (cut a rod into pieces to maximize profit) is typically solved using *[Extra – GATE pattern]*

- a) Greedy method
- b) Dynamic programming (Correct)**
- c) Randomized algorithm
- d) Branch and bound

Explanation: The best way to cut a rod of length n depends on the best way to cut smaller lengths, and those smaller subproblems repeat often — a classic sign that Dynamic Programming (storing answers for each length) is the right tool.

Unit 5: Graph & Tree Algorithms: DFS, BFS, Shortest Paths, Topological Sort

Part A — MCQs from BEU Previous Year Papers (6 Questions)

Q1. Topological sorting is applicable to [BEU 2025]

- a) Any graph
- b) Directed Acyclic Graph (DAG) only (Correct)**
- c) Undirected graphs only
- d) Cyclic graphs only

Explanation: Topological sorting arranges vertices so every edge points from an earlier vertex to a later one. This only makes sense if there are no cycles and the edges have direction — that's exactly a DAG (Directed Acyclic Graph).

Q2. BFS on a graph $G = (V, E)$ has running time [BEU 2023]

- a) $O(|V| + |E|)$ (Correct)**
- b) $O(|V|)$
- c) $O(|E|)$
- d) None of the above

Explanation: Breadth First Search visits every vertex once ($O(|V|)$) and looks at every edge once ($O(|E|)$) while exploring from each vertex, so the total time adds up to $O(|V| + |E|)$.

Q3. The minimum number of colors needed to properly color a graph having $n > 3$ vertices and only 2 edges is [BEU 2023]

- a) 2 (Correct)**
- b) 3
- c) 4
- d) 1

Explanation: With just 2 edges spread among many vertices, the graph can't contain any odd-length cycle (like a triangle needs at least 3 edges), so it stays 'bipartite-like' and can always be colored using just 2 colors.

Q4. Level order traversal of a rooted tree can be done by starting from the root and performing [BEU 2023]

- a) Depth first search
- b) Breadth first search (Correct)**
- c) Pre-order traversal
- d) In-order traversal

Explanation: Level order means visiting all nodes level by level (all children of the root, then all grandchildren, and so on) — that is exactly what Breadth First Search does, usually using a queue.

Q5. The node whose removal makes a graph disconnected is called [BEU 2023]

- a) Pendant vertex
- b) Bridge
- c) Articulation point (Correct)**
- d) Coloured vertex

Explanation: This special kind of vertex is called an Articulation Point (or cut vertex) — removing it breaks the graph into two or more separate pieces. (A 'Bridge' is the same idea but for an EDGE, not a vertex.)

Q6. Which one of the following is an application of the Queue data structure? [BEU 2022]

- a) When a resource is shared among multiple consumers
- b) When data is transferred asynchronously between two processes
- c) Load balancing
- d) All of the above (Correct)**

Explanation: A queue naturally serves things in the order they arrive (First-In-First-Out), which fits all three situations — sharing a resource fairly, buffering data between processes, and spreading out work evenly (load balancing). It's also exactly what BFS uses to explore a graph level by level.

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (4 Questions)

Q7. Dijkstra's algorithm gives incorrect results when the graph has [Extra – GATE pattern]

- a) Cycles
- b) Negative edge weights (Correct)**
- c) Disconnected components
- d) Self-loops

Explanation: Dijkstra's algorithm assumes that once a vertex's shortest distance is finalized, it can never get smaller later. A negative edge can break that assumption, so the algorithm can give a wrong (too-large) answer.

Q8. The Floyd-Warshall algorithm computes shortest paths between [Extra – GATE pattern]

- a) A single source and all other vertices
- b) All pairs of vertices in the graph (Correct)**

- c) Only two chosen vertices
- d) Vertices on a spanning tree only

Explanation: Unlike Dijkstra's or Bellman-Ford's (which start from one source), Floyd-Warshall finds the shortest distance between EVERY pair of vertices at once, using dynamic programming over all possible 'via' vertices.

Q9. Depth First Search (DFS), whether implemented recursively or iteratively, essentially uses which data structure to keep track of vertices to visit next?

[Extra – GATE pattern]

- a) Queue
- b) Stack (Correct)**
- c) Priority queue
- d) Hash table

Explanation: DFS always goes as deep as possible before backtracking — that 'go deep, then come back' behaviour is exactly what a Stack gives you (Last-In-First-Out), whether it's an explicit stack or the function call stack during recursion.

Q10. Bellman-Ford's algorithm, unlike Dijkstra's, is able to correctly detect the presence of *[Extra – GATE pattern]*

- a) Only positive weight cycles
- b) Negative weight cycles (Correct)**
- c) Only disconnected graphs
- d) Only bipartite graphs

Explanation: Bellman-Ford relaxes every edge repeatedly ($V-1$ times). If, on one extra round, some distance can STILL be reduced, that means there's a negative weight cycle — something Dijkstra's algorithm can't even detect properly.

Unit 6: Tractable & Intractable Problems: P, NP, NP-Complete & Approximation

Part A — MCQs from BEU Previous Year Papers (2 Questions)

Q1. An approximation algorithm guarantees *[BEU 2025]*

- a) An exact optimal solution
- b) An incorrect solution always
- c) A random solution without bounds
- d) A solution within a known factor of the optimal (Correct)**

Explanation: For very hard (NP-hard) problems, finding the perfect answer can take too long. An approximation algorithm instead promises a solution that is

provably 'close enough' — within some known ratio of the true best answer — but found much faster.

Q2. The Travelling Salesman Problem (decision version) belongs to [BEU 2023]

- a) P class
- b) NP class
- c) NP-hard
- d) NP-complete class (Correct)**

Explanation: The decision version of TSP ('is there a tour shorter than K?') is one of the classic NP-complete problems — it's in NP (a proposed answer can be checked quickly) and every other NP problem can be reduced to it. (Note: the optimization version, 'find the shortest tour', is called NP-hard.)

Part B — Extra Important MCQs (GATE Pattern, for Exam Prep) (4 Questions)

Q3. A problem is called NP-complete if it is [Extra – GATE pattern]

- a) In P and also NP-hard
- b) In NP, and every other problem in NP can be reduced to it in polynomial time (Correct)**
- c) Impossible to solve by any computer
- d) Solvable only by brute force, always

Explanation: NP-complete problems are the 'hardest' problems within NP — you can quickly check a proposed solution (that's what being in NP means), and every other NP problem can be transformed into it in polynomial time.

Q4. Cook's theorem was the first to prove that a specific problem is [Extra – GATE pattern]

- a) In P
- b) NP-complete (the Boolean Satisfiability / SAT problem) (Correct)**
- c) Undecidable
- d) Solvable in constant time

Explanation: Cook's theorem (1971) showed that the SAT (Boolean Satisfiability) problem is NP-complete — this was the very first NP-completeness proof, and it opened the door to proving many other problems NP-complete by reduction from SAT.

Q5. If it were proven that $P = NP$, it would mean [Extra – GATE pattern]

- a) No problem could ever be solved by a computer
- b) Every problem whose solution can be verified quickly could also be SOLVED quickly (Correct)**

- c) All algorithms would take exponential time
- d) NP-hard problems would stop existing

Explanation: $P = NP$ would mean there's no real gap between 'checking an answer fast' and 'finding an answer fast' — anything you can verify quickly, you could also solve quickly. This is one of the biggest open questions in computer science.

Q6. A randomized algorithm's running time or output may depend on *[Extra – GATE pattern]*

- a) The input size only
- b) Random choices made during its execution (Correct)**
- c) The programming language used
- d) The compiler settings

Explanation: A randomized algorithm deliberately makes some random choices while running (like picking a random pivot in quicksort), so its exact speed or answer can vary a little between runs, even on the exact same input.

