24CB405/24CM405/24CS405/24DS405/24IT405

II/IV B.Tech (Regular)DEGREE EXAMINATION

Scheme & Key

May,2026 **Common to CB, CM, CS, DS &IT**

Fourth Semester **Design and Analysis of Algorithms**

1. a) **Define Inadmissible equations. [1M]**

Inadmissible equations or solutions are mathematical, statistical, or logical statements that fail to satisfy required constraints, boundary conditions, or that are uniformly outperformed by another superior formula

b) **Define Big O notation. [1M]**

Big O notation is a mathematical tool used to describe the upper bound of an algorithm's.

c) **What is the general method of divide and conquer? [1M]**

Divide and Conquer is an algorithmic paradigm that solves complex problems by breaking them into smaller subproblems, solving them, and combining the results.

d) **Define optimal solution. [1M]**

An optimal solution is the best possible, or most efficient, feasible solution among all potential solutions to an optimisation problem.

e) **List out the key characteristics of a greedy algorithm. [1M]**

The key characteristics of the greedy algorithm is that it selects the best available solution at each step based on a specific criterion, without reconsidering previous choices or evaluating future consequences. The main idea is to make a sequence of choices, each of which looks optimal at the moment, aiming to reach an overall optimal or near-optimal solution.

f) **What is the objective of the Job Sequencing problem? [1M]**

The primary objective of the job sequencing algorithm is to maximise total profit. It achieves this by selecting a subset of jobs that can be completed within their respective deadlines and assigning them to available time slots. Each job typically takes 1 unit of time to complete.

g) **Define a graph. [1M]**

It is a non-linear data structure used to represent complex, interconnected relationships between objects. It is formally defined as a pair (G = (V, E)), consisting of a finite set of **vertices** (nodes) (V) and a finite set of **edges** (links) (E) that connect pairs of vertices.

h) **What algorithm solves the single-source shortest path for non-negative weights? [1M]**

Dijkstra's algorithm.

i) **Define strongly connected components**.**[1M]**

A Strongly Connected Component in a directed graph is a maximal subgraph where every vertex is reachable from every other vertex within that subgraph

j) **Define LC branch and bound**.**[1M]**

Least Cost (LC) Branch and Bound is an optimisation algorithm that finds the best solution by traversing a state-space tree, prioritising nodes with the lowest estimated cost.

k) **Define Backtracking.[1M]**

Backtracking is a systematic, trial-and-error approach to solving problems by building candidates incrementally. It explores possibilities (like a Depth-First Search) and immediately abandons a path ("backtracks") as soon as it determines the path cannot lead to a valid solution.

l) **Name the class of problems that are the hardest among NP problems. [1M]**

The class of problems that are the hardest among NP problems is known as NP Complete**.** NP-Complete problems represent the intersection of two classes: NP and NP-Hard.

2. a) **Define an algorithm. Explain the characteristics of an algorithm.**

Definition – 2M

Characteristics of algorithm – 4M

An algorithm is a finite, step-by-step procedure or set of well-defined instructions designed to solve a specific problem or perform a particular task. It takes some input, processes it through a sequence of steps, and produces an output. An algorithm is a finite, step-by-step procedure or set of well-defined instructions designed to solve a specific problem or perform a particular task. It takes some input, processes it through a sequence of steps, and produces an output.

Characteristics of an Algorithm

1. **Input**
   An algorithm should accept zero or more inputs. These inputs are the data that the algorithm processes.
2. **Output**
   It must produce at least one output, which is the result of processing the input.
3. **Definiteness (Unambiguity)**
   Every step of the algorithm must be clearly and precisely defined. There should be no confusion about what each instruction means.
4. **Finiteness**
   The algorithm must terminate after a finite number of steps. It should not run indefinitely.
5. **Effectiveness (Feasibility)**
   Each step must be simple, basic, and executable within a finite amount of time using available resources.

b) **Explain about Asymptotic notations.**

Big O Notation – 2M

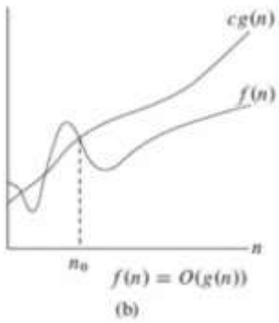
Big Omega Notation – 2M

Theta Notation – 2M

Big O Notation

The function f(n)= Og(n)) Iff they exist positive constant c and n0.

Such that f(n)<=c * g(n) for all n>= n0

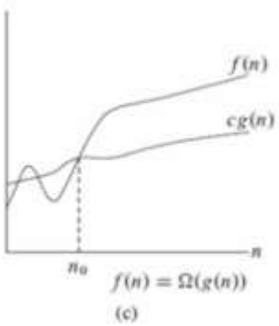


Example: If an algorithm runs in $3n^2 + 2n +$ 1 steps, it is $O(n^2)$ because n^2 dominates as n grows.

Big Omega notation

The function f(n)= **Ω** g(n)) Iff they exist positive constant c and n0.

Such that f(n)>=c * g(n) for all n>= n0

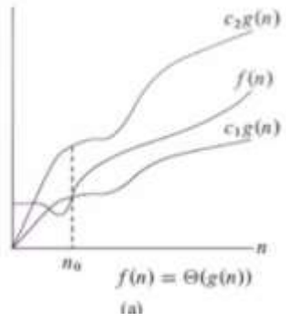


Example: If an algorithm takes at least $2n$ steps for large n, we can say it is $\Omega(n)$.

Theta Notation

The function f(n)= θ g(n)) Iff they exist positive constant c1 , c2 and n0.

Such that c1 * g(n)<=f(n)<= c2 * g(n) for all n>= n0



Example: If an algorithm always takes around $3n^2 + 7n$, then $f(n) = \Theta(n^2)$

3. a) **Define time complexity and space complexity. Develop an algorithm for adding n natural numbers.**

Definitions – 2M

Algorithm – 4 M

Time complexity is the measure of how the running time of an algorithm increases with the size of the input. It is usually expressed using Big O notation

Space complexity is the measure of the amount of memory an algorithm requires as the size of the input increases. It is usually expressed using Big O notation.

Algorithm to Add n Natural Numbers

1. Start
2. Read n
3. Set sum = 0
4. Repeat from i = 1 to n
    - sum = sum + i
5. Print sum
6. Stop

b) **Describe case 1, case 2 and case 3 of masters theorem and explain them with example.**

Case-1 – 2M

Case-2 – 2M

Case-3 – 2M

<u>Case-1:</u>

If $a > b^k$, then $T(n) = \theta (n^{\log_b a})$

<u>Case-2:</u>

If $a = b^k$ and

i. If $p < -1$, then $T(n) = \theta (n^{\log_b a})$
ii. If $p = -1$, then $T(n) = \theta (n^{\log_b a}.\log^2 n)$
iii. If $p > -1$, then $T(n) = \theta (n^{\log_b a}.\log^{p+1} n)$

<u>Case-3:</u>

If $a < b^k$ and

i. If $p < 0$, then $T(n) = O (n^k)$
ii. If $p >= 0$, then $T(n) = \theta (n^k \log^p n)$

4. a) **Explain the Quicksort algorithm with a step-by-step example. Analyze its best, average, and worst-case time complexities.**

Algorithm – 4M

Analysis – 2M

Algorithm:

```
QUICKSORT(A, low, high)
if low < high then
    p = PARTITION(A, low, high)
QUICKSORT(A, low, p-1)
QUICKSORT(A, p+1, high)
```

```
PARTITION(A, low, high)
pivot = A[high]
i = low - 1
for j = low to high-1 do
    if A[j] <= pivot then
i = i + 1
        swap A[i] and A[j]
swap A[i+1] and A[high]
return i + 1
```

Best Case — $O(n\log\ n)$
Average Case — $O(n\log\ n)$
Worst Case — $O(n^2)$

b) **Explain the Job Sequencing with Deadlines algorithm with a suitable example.**

Any Example with correct solution – 6M

5. a) **Derive the recurrence relation for Strassen's algorithm and solve it to find its time complexity.**

Recurrence Relation – 2M

Solution – 4M

The recurrence relation of Strassen's algorithm is
T(n)=7T(2n)+O(n^2)
Derivation of the recurrence relation
Assume $T(n) \le An^k$ for some constant $A > 0$. Then:

$$T(n) \le 7\,A\left(\frac{n}{2}\right)^k + cn^2 = 7A\frac{n^k}{2^k} + cn^2 = \frac{7A}{2^k}n^k + cn^2$$

Set $\frac{7}{2^k} = 1 \implies 2^k = 7 \implies k = log_{\,2}7$.

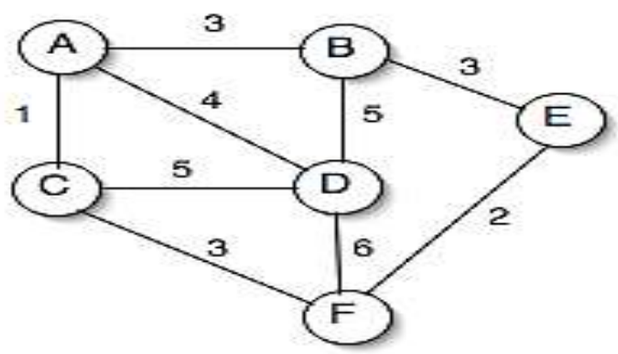
Since $O(n^2)$ is the non-recursive term, compare powers:

- If $k > 2$, the $O(n^2)$ The term is dominated by recursive growth.
- If $k < 2$, the recursive term grows more slowly than n^2, and $O(n^2)$ dominates.
- Here $k = log_{\,2}7 \approx 2.807 > 2$, so the recursive term dominates.
- By the substitution method (or induction), we can verify that:
- $T(n) = O(n^{log_{\,2}7})$
- and similarly, a lower bound $T(n) = \Omega(n^{log_{\,2}7})$ can be established, so:

$$T(n) = \Theta(n^{log_{\,2}7})$$

- Assume $T(m) \le c\,m^{log_{\,2}7}$ for all $m < n$. Then:
- $T(n) \le 7T(n/2) + c_0n^2 \le 7 \cdot c\left(\frac{n}{2}\right)^{log_{\,2}7} + c_0n^2 = 7 \cdot c\frac{n^{log_{\,2}7}}{7} + c_0n^2 = cn^{log_{\,2}7} + c_0n^2$
- Since $n^{log_{\,2}7} \gg n^2$ for large n, The inductive step holds.
  Hence, the solution is
- $\boxed{T(n) = \Theta(n^{log_{\,2}7}) \approx \Theta(n^{2.807})}$

b) **Construct minimum cost spanning tree using Kruskal's algorithm.**



Algorithm – 3M

Solution – 3M

Step 1: List all edges with weights

From the graph:

Edge	Weight
A–C	1
E–F	2
A–B	3
B–E	3
C–F	3
A–D	4
B–D	5
C–D	5
D–F	6

Step 2: Sort edges in ascending order of weight

AC(1), EF(2), AB(3), BE(3), CF(3), AD(4), BD(5), CD(5), DF(6)

Step 3: Apply Kruskal's Algorithm

Select edges one by one without forming cycles:

1. A–C (1) → Add
2. E–F (2) → Add
3. A–B (3) → Add
4. B–E (3) → Add (connects two components)
5. C–F (3) → (forms cycle, skip)
6. A–D (4) → Add

Now we have (n−1 = 5 edges) → MST complete.

Step 4: Minimum Spanning Tree

Edges in MST:

- A–C (1)
- E–F (2)
- A–B (3)
- B–E (3)
- A–D (4)

Step 5: Total Minimum Cost

Cost=1+2+3+3+4=13

Minimum Cost Spanning Tree = 13

Selected Edges: {A–C, E–F, A–B, B–E, A–D}

6. a) **Solve the solution for 0/1 knapsack problem using dynamic programming (p1,p2,p3, p4) = (11, 21, 31, 33), (w1, w2, w3, w4) = (2, 11,22, 15), M=40, n=4.**

Solution – 6M

Given Data

- Profits: (p1,p2,p3,p4)=(11,21,31,33)
- Weights: (w1,w2,w3,w4)=(2,11,22,15)
- Capacity: M=40
- Number of items: n=4

Step 1: DP Table Definition

Let DP[i][w]DP[i][w]DP[i][w] = maximum profit using first iii items and capacity www.

We compute values row by row.

Step 2: Filling Key Values (Summary)

Instead of writing the full 2D table (very large), we compute optimal combinations:

Check feasible combinations

Items Selected	Total Weight	Total Profit
(1)	2	11
(2)	11	21

(3)	22	31
(4)	15	33
(1,2)	13	32
(1,3)	24	42
(1,4)	17	44
(2,3)	33	52
(2,4)	26	54
(3,4)	37	64
(1,2,3)	35	63
(1,2,4)	28	65
(1,3,4)	39	75
(2,3,4)	48	(exceeds capacity)
(1,2,3,4)	50	(exceeds capacity)

Step 3: Optimal Solution

- Best valid combination: Items (1, 3, 4)
- Total weight = 2+22+15=39 ≤ 40
- Total profit = 11+31+33=75

Step 4: Final Answer

Maximum Profit = 75
Selected Items = {1,3,4}

b) **Explain the Travelling sales man problem with suitable example.**

Any example with solved solution – 6M

7. a) **Write the algorithm to compute 0/1 Knapsack problem using dynamic programming and explain it.**

Algorithm – 3M

Explanation – 3M

Algorithm Knapsack(w, v, n, W)
Input: weights w[1..n], values v[1..n], capacity W
Output: Maximum profit

Create a table DP[0..n][0..W]

```
for i = 0 to n:
    DP[i][0] = 0

for w = 0 to W:
    DP[0][w] = 0

for i = 1 to n:
    for w = 1 to W:
      if w[i] <= w then
        DP[i][w] = max(DP[i-1][w], v[i] + DP[i-1][w - w[i]])
      else
        DP[i][w] = DP[i-1][w]

return DP[n][W]
```

We fill the table row by row.
For each item, we decide:

- Exclude it → take value from previous row

- Include it → add its value and reduce capacity

We choose the maximum of both choices.

Final answer is stored in DP[n][W].

i. Time Complexity: O(nW)
ii. Space Complexity: O(nW)

b) **Explain bi-connected components and articulation points with an example.**

Explanation – 3M

Articulation Point – 3M

1. Articulation Point (Cut Vertex)

An articulation point in a graph is a vertex whose removal increases the number of connected components.

Explanation

- If removing a vertex (and its edges) disconnects the graph, that vertex is an articulation point.
- It represents a critical node whose failure breaks connectivity.

Example

Consider a graph:

```
A —— B —— C
     |
     D
```

- Vertex B is an articulation point.
- If B is removed, the graph splits into:
  - Component 1: A
  - Component 2: C
  - Component 3: D

2. Bi-Connected Component (Biconnected Graph)

A bi-connected component is a maximal subgraph that has no articulation point.

Explanation

- In such a component, removing any one vertex does NOT disconnect the graph.
- There are at least two disjoint paths between every pair of vertices.

3. Example of Bi-Connected Components

Consider the graph:

```
A —— B —— C
     |
     D —— E
```

Analysis

- B is an articulation point.
- The graph can be divided into bi-connected components:

1. Component 1: A — B
2. Component 2: B — C
3. Component 3: B — D — E

Each of these is a maximal subgraph with no articulation point inside it.

8. a) **Explain the Backtracking techniquewith an example.**

Example – 3M

State Space Tree – 3M

b) **State and explain Cook's theorem.**

Theorem – 2M

Explanation – 4M

Statement of Cook's Theorem

Cook's theorem states that:

The Boolean Satisfiability Problem (SAT) is NP-complete.

This means:

1. SAT belongs to the class **NP**, and
2. Every problem in NP can be reduced to SAT in **polynomial time**.

Key Concepts Involved

- **NP (Nondeterministic Polynomial time):**

Class of decision problems where a given solution can be verified in polynomial time.

- **Polynomial-Time Reduction:**
  A way of converting one problem into another efficiently (in polynomial time).
- **SAT (Satisfiability Problem):**
  Given a Boolean formula, determine whether there exists an assignment of truth values that makes the formula true.

Explanation of the Theorem

Cook's theorem proves that **SAT is the first NP-complete problem**, meaning it is among the "hardest" problems in NP.

Main Idea of the Proof

- Consider any problem in NP.
- Such a problem can be solved by a **nondeterministic Turing machine** in polynomial time.
- The computation of this machine can be encoded as a **Boolean formula**.
- This formula is constructed such that:
  - It is **satisfiable** if and only if
    the machine accepts the input.
- Hence, solving the original problem reduces to solving a SAT instance.

Importance of Cook's Theorem

- It introduced the concept of **NP-completeness**.
- It provides a **standard method** to prove other problems NP-complete (via reduction from SAT).
- It is central to understanding the famous **P vs NP problem**.
- Many real-world problems (like scheduling, circuit design, optimization) are linked to SAT.

9. a) **Explain non deterministic algorithm for sorting and searching.**

ND_Sorting Algorithm – 3M

ND_Searching Algorithm – 3M

Non deterministic algorithm for searching

```
Algorithm ND_Search(A, n, x)
{
   j = choice(1, n)
   if (A[j] == x) then
      write(j)
success()
   else
failure()
}
```

Non deterministic algorithm for sorting.

```
Algorithm ND_Sort(A, n)
{
   for i = 1 to n do
      j = choice(1, n)
      if (B[j] is not occupied) then
         B[j] = A[i]
      else
failure()

   for i = 1 to n-1 do
      if (B[i] > B[i+1]) then
failure()

   write(B)
success()
}
```

b) **Distinguish between NP Complete and np-hard problems.**

NP Complete – 3M

NP Hard – 3M

NP-Complete:
1. Problems belong to NP.
2. Solutions verifiable in polynomial time.
3. Every NP problem reducible to it.

Examples:
SAT, Hamiltonian Cycle.

NP-Hard:
1. At least as hard as NP problems.
2. Need not belong to NP.
3. May not be verifiable in polynomial time.

Examples:
Halting Problem, TSP optimization version.