ALGORITHMS SUGGESTION FOR SEMESTER

- **LIFO scheme is used in data structure.**

Answer: a) Stack
Explanation: LIFO (Last In, First Out) means the last element added is the first to be removed. This is the core principle of a stack, used in applications like function calls and undo operations.

- **Which of the following sorting algorithms has a worst-case time complexity of O(nlog⁡n)O(n \log n)O(nlogn)?**

Answer: b) Merge sort
Explanation: Merge sort is a divide-and-conquer algorithm with a worst-case and average-case time complexity of O(nlog⁡n)O(n \log n)O(nlogn). Other options like bubble, selection, and insertion sort have higher worst-case complexities.

- **O-notation provides an asymptotic:**

Answer: a) Upper bound
Explanation: Big-O notation describes the upper bound of an algorithm's time complexity, giving the worst-case scenario for performance.

- **Time complexity of worst case for Linear Search is:**

Answer: c) O(n)O(n)O(n)
Explanation: In the worst case, linear search examines each element of the array once, leading to O(n)O(n)O(n) complexity, where nnn is the number of elements.

- **Which of the following algorithm design techniques is used in the Merge sort algorithm?**

Answer: c) Divide and conquer
Explanation: Merge sort divides the array into smaller subarrays, sorts them recursively, and merges them. This is the essence of the divide-and-conquer approach.

- **Which of the following is false in the case of a spanning tree of a graph GGG?**

Answer: d) It can be either cyclic or acyclic
Explanation: A spanning tree is always acyclic and connects all vertices in a graph. Therefore, the claim that it can be cyclic is false.

- **Which of the following is true?**

Answer: a) Prim's algorithm initializes with a vertex
Explanation: Prim's algorithm starts with a single vertex and grows the minimum spanning tree by adding edges with the least weight.

- **Topological sort can be applied to which of the following graphs?**

Answer: d) Directed Acyclic Graphs
Explanation: Topological sorting is used to order the vertices of a Directed Acyclic Graph (DAG) in a linear sequence that respects the direction of edges.

- **A Circuit that does not repeat vertices is called:**

Answer: b) Path
Explanation: A path is a sequence of edges that connects vertices without repeating any vertex. If vertices are repeated, it forms a cycle instead.

- **What is a hash table?**

Answer: b) A structure that maps keys to values
Explanation: A hash table uses a hash function to map keys to values efficiently, allowing for fast data retrieval based on the key.

- **Dijkstra's algorithm is used for finding:**

Answer: b) Shortest path
Explanation: Dijkstra's algorithm calculates the shortest path between nodes in a weighted graph, making it widely used in network routing.

xii) A graph is called a _____ if it is a connected acyclic graph.
Answer: c) Tree
Explanation: A tree is a special type of graph that is connected (all vertices are reachable) and acyclic (no cycles). Each tree is a minimal connected graph with nnn vertices and n−1n-1n−1 edges.

xiii) A binary tree is balanced if the difference of height between left and right sub-tree is not more than:
Answer: b) 1
Explanation: A binary tree is balanced (e.g., AVL tree) if the height difference (balance factor) between the left and right subtrees of any node is at most 1. This ensures logarithmic time complexity for operations like search, insertion, and deletion.

xiv) The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the number of leaf nodes of the binary search tree?
Answer: a) 3

xv) What is a Rabin-Karp Algorithm?
Answer: a) String Matching Algorithm
Explanation:
The Rabin-Karp algorithm is used for pattern matching in strings. It uses hashing to find occurrences of a "pattern" string within a "text" string. Its average-case time complexity is O(n+m), but worst-case is O(n·m), where nnn is the length of the text and mmm is the pattern length.

i) ADT full form

Correct Answer: c) Abstract Data Type
Explanation:
An Abstract Data Type (ADT) is a mathematical model for a data structure, specifying the type of data it holds and the operations performed on it, without detailing the implementation.

ii) In a multiset, elements are arranged in an unordered manner, with elements having a multiplicity of?

Correct Answer: b) More than one
Explanation:
A multiset allows elements to occur multiple times (have multiplicity greater than one) and does not enforce any particular order of elements.

iii) Binary Search can be categorized into which of the following?

Correct Answer: b) Divide and conquer
Explanation:
Binary Search divides the search space into halves recursively and conquers each subproblem by solving it, making it a classic example of the divide-and-conquer approach.

iv) What is the time complexity of binary search with iteration?

Correct Answer: b) O(logn)
Explanation:
Binary Search halves the search range at each step. Therefore, the time complexity is logarithmic, denoted as O(log⁡n).

v) Which of the following is/are property/properties of a dynamic programming problem?

Correct Answer: d) Both optimal substructure and overlapping sub-problems
Explanation:
Dynamic Programming solves problems by breaking them into smaller overlapping sub-problems and combining their solutions. It also relies on the property of optimal substructure, where the solution to a problem can be composed of solutions to its sub-problems.

vi) A circuit that does not repeat vertices is called:

Correct Answer: b) a path
Explanation:
A path is a sequence of vertices where no vertex is repeated. A cycle, on the other hand, starts and ends at the same vertex and may repeat vertices.

viii) The time complexity of insertion sort is

Correct Answer: b) Quadratic
Explanation:
Insertion Sort has a time complexity of O(n^2) in the worst and average cases due to the nested iterations when comparing and shifting elements.

ix) If A=(x,y,z), the number of elements of the power set is:

Correct Answer: b) 8
Explanation:
The power set of a set with n elements contains 2^n subsets. For n=3 8 subsets.

x) Which option contains two equal sets?

Correct Answer: b) X=(5,6,8,9)and Y=(6,8,5,9)
Explanation:
Two sets are equal if they have the same elements, irrespective of their order. Both X and Y contain {5,6,8,9}.

xi) Time complexity depends on

Correct Answer: b) Run time
Explanation:
Time complexity measures the amount of time an algorithm takes to run as a function of the input size, which is determined during run time.

xii) Which of the following is a linear data structure?

Correct Answer: a) Array
Explanation:
Arrays store elements sequentially in memory, making them a linear data structure. AVL Trees, Binary Trees, and Graphs are non-linear structures.

xiii) What is the maximum number of swaps that can be performed in the Selection Sort algorithm?

Correct Answer: a) n−1n-1n−1
Explanation:
In Selection Sort, at most n−1n-1n−1 swaps are performed, as each iteration selects the smallest element and places it in the correct position.

xiv) Which of the following has search efficiency of O(1)?

Correct Answer: c) Hash Table
Explanation:
Hash Tables provide O(1) average-case search efficiency due to direct indexing using hash functions. Trees, Heaps, and Linked Lists have higher time complexities for search operations.

LONG-ANSWER TYPESS:-

Selection Sort Algorithm

Algorithm:

```
SelectionSort(arr, n):
  for i from 0 to n-2:
    minIndex = i
    for j from i+1 to n-1:
      if arr[j] < arr[minIndex]:
        minIndex = j
    Swap arr[i] with arr[minIndex]
```

Steps Explanation with Example:

Sort the array: [64,25,12,22,11

1. **Pass 1**:
   Find the smallest element in the entire array: **11**.
   Swap it with the first element. Array: [11,25,12,22,64
2. **Pass 2**:
   Find the smallest element from index 1 to 4: **12**.
   Swap it with the element at index 1. Array: [11,12,25,22,64
3. **Pass 3**:
   Find the smallest element from index 2 to 4: **22**.
   Swap it with the element at index 2. Array: [11,12,22,25,64
4. **Pass 4**:
   Find the smallest element from index 3 to 4: **25**.
   Swap it with itself (no change). Array: [11,12,22,25,64
5. The array is sorted.

Worst-Case Time Complexity:

- Outer loop runs (n−1) times.
- Inner loop runs n−1,n−2,...,1 times.
- Total comparisons: (n−1)+(n−2)+...+1= n(n-1)/2.
- Time complexity:O(n^2).

Prim's Algorithm for Minimum Spanning Tree

Algorithm:

```
PrimMST(graph, V):
  MST[] = False
  Key[] = ∞
  Parent[] = -1
  Key[0] = 0

  for count from 1 to V:
    u = ExtractMin(Key, MST)
    MST[u] = True

    for v in graph[u]:
      if graph[u][v] < Key[v] and MST[v] == False:
        Parent[v] = u
        Key[v] = graph[u][v]
```

1. Start from vertex A (initial key values for A, B, C, D: 0, ∞, ∞, ∞).
2. Add the edge A-B (2). Update key values for B and C.
3. Add the edge C-D (3).
4. Continue adding the smallest weight edge until all vertices are included.

Result: MST includes edges **A-B (2)**, **A-C (4)**, and **C-D (3)** with a total weight of **9**.

5. Significance and Limitations of Linear Sort + Bucket Sort

Significance of Linear Sort:

- Sorting in O(n)O(n)O(n) time for specific cases (e.g., counting sort, bucket sort, radix sort).
- Useful when input size is large and elements are within a known range.

Limitations of Linear Sort:

- Cannot handle general sorting problems (e.g., when comparisons are required).
- Limited to integers or fixed ranges; performance degrades if the range is vast.

Bucket Sort with Example:

Sort [0.78,0.17,0.39,0.26,0.72,0.94,0.21,0.12,0.23,0.68]

1. Create 10 buckets for values in the range [0, 1).
2. Distribute elements:
   - Bucket 0: [0.12, 0.17, 0.21, 0.23]
   - Bucket 1: [0.26, 0.39, 0.68, 0.72]
   - Bucket 2: [0.78]
   - Bucket 3: [0.94]
3. Sort elements in each bucket using another algorithm (e.g., insertion sort).
4. Concatenate buckets for sorted array:
   [0.12,0.17,0.21,0.23,0.26,0.39,0.68,0.72,0.78,0.94].

5. Recursive Binary Search Algorithm

```
BinarySearch(arr, left, right, x):
  if left > right:
    return -1

  mid = (left + right) // 2
  if arr[mid] == x:
    return mid
  elif arr[mid] < x:
    return BinarySearch(arr, mid+1, right, x)
  else:
    return BinarySearch(arr, left, mid-1, x)
```

Explanation:

1. Start with the middle element of the sorted array.
2. If the middle element is the target, return its index.
3. If the target is smaller, search the left half; if larger, search the right half.

Is Binary Search Useful for Unsorted Array?

No, because binary search requires the array to be sorted to work effectively.

Time Complexity:

- Each step halves the search space.
- Time complexity: O(log⁡n)O(\log n)O(logn).

7. Hashing and Collision Resolution

What is Hashing?

Hashing is a technique to map keys to specific locations in a table using a hash function.

Uses of Hash Table and Hash Function:

- **Hash Table**: Used in databases, caching, and lookup tables for quick data access.
- **Hash Function**: Converts a key into an index to access a hash table.

Chaining Method for Collision Resolution:

- Each table index points to a linked list of elements with the same hash value.
- Insertion involves adding the new key to the linked list.
- Search involves traversing the list at the hashed index.

8. Bellman-Ford's Shortest-Path Algorithm

```
BellmanFord(graph, V, src):
    dist[] = ∞
    dist[src] = 0

    for i from 1 to V-1:
        for each edge (u, v, w):
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
```

1. Initialize distances: A=0, B=∞, C=∞, D=∞.
2. Relax edges: A → B (6), A → C (5). Update D through C (-2).

Final distances: A=0, B=6, C=5, D=3.

9. Naïve String-Matching Algorithm

```
NaiveStringMatch(text, pattern):
    n = len(text)
    m = len(pattern)
    for i from 0 to n-m:
        if text[i:i+m] == pattern:
            print("Pattern found at index", i)
```

Example:

Text: 1110101011101100
Pattern: 101110

1. Slide the pattern across the text and compare.
2. Matches found at indices: **6** and **10**.

i) Which is the best searching algorithm and why?

Answer:

The best searching algorithm depends on the scenario. **Binary Search** is ideal for sorted arrays due to its O(log n) time complexity, making it faster than linear search. **Hashing** is efficient for direct access with O(1) time, but it requires more memory and careful handling of collisions. The choice depends on data structure and constraints.

ii) Describe best case time complexity.
Answer:
Best case time complexity refers to the minimal time an algorithm takes to complete under optimal conditions. For example, in **Linear Search**, the best case occurs when the target element is the first in the list, yielding a complexity of O(1). It measures the ideal performance of the algorithm.

iii) Define 'Big Theta' Notation.
Answer:
Big Theta (Θ) notation represents the tight bound of an algorithm, covering both the upper and lower bounds. It defines an algorithm's exact growth rate. For example, for an algorithm with complexity , Θ(n^2) implies it grows proportionally to n^2 regardless of best or worst case.

iv) What do you mean by sorting?
Answer:
Sorting is the process of arranging elements in a specific order, such as ascending or descending. It is crucial for efficient data organization and retrieval. Common sorting algorithms include Bubble Sort, which compares adjacent elements, and Merge Sort, which uses a divide-and-conquer approach.

v) Define Dynamic Programming algorithm.
Answer:

Dynamic Programming (DP) is a method used to solve problems by breaking them into smaller subproblems, solving each only once, and storing the results. This avoids redundant calculations. For example, the Fibonacci sequence uses DP to compute terms efficiently by reusing previously computed values.

vi) Why is a sorting technique called stable?
Answer:
A sorting technique is called stable if it maintains the relative order of elements with equal keys. For example, in Bubble Sort, if two elements have the same value, their original order remains unchanged. Stable sorting is essential in applications where the sequence of equal elements carries meaning.

vii) Which is more efficient: Insertion Sort or Merge Sort?
Answer:
Merge Sort is more efficient than Insertion Sort for large datasets due to its O(nlogn) time complexity compared to O(n^2). Merge Sort uses a divide-and-conquer approach, while Insertion Sort is better suited for small or nearly sorted datasets because of its simplicity.

viii) Write the advantages of height balancing.
Answer:
Height balancing minimizes the height of a binary tree, ensuring efficient operations like search, insertion, and deletion with O(logn) complexity. It prevents skewed trees, which degrade performance. Height-balanced trees like AVL and Red-Black Trees are widely used for dynamic datasets.

ix) What is Omega Notation & give example?
Answer:
Omega (Ω) notation represents the lower bound of an algorithm's performance, describing the best-case scenario. For instance, in Binary Search, the best-case time complexity is Ω(1), as the target might be found on the first comparison. It complements Big-O notation by setting a baseline.

x) What is inorder traversal in BST?
Answer:
Inorder traversal in a Binary Search Tree (BST) visits nodes in ascending order. The process is: traverse the left subtree, visit the root, then traverse the right subtree. For example, an inorder traversal of a BST with nodes {3, 1, 4} yields the sequence {1, 3, 4}.

a) What is an algorithm and what is the Complexity of the Algorithm?
Answer:
An algorithm is a step-by-step procedure or a set of instructions designed to solve a specific problem or perform a computation. It takes an input, processes it through a defined set of steps, and produces an output.

The **complexity of an algorithm** refers to the analysis of its efficiency, categorized into:

1. **Time Complexity:** The amount of time the algorithm takes to execute as a function of input size (n). Example: O(n^2) for Bubble Sort.
2. **Space Complexity:** The amount of memory required by the algorithm as a function of input size. Example: O(1)for Insertion Sort (in-place sorting).

b) Write an algorithm for the Bubble Sort method.
Answer:
Algorithm:

1. Input: Array A[] of size n.
2. Repeat for i=0 to n−1:
    - For j=0 to n−i−2:
        - If A[j]>A[j+1], swap A[j] and A[j+1].
3. Output: Sorted array A[].

Example:
Unsorted array: [5, 3, 8, 6]
Pass 1: [3, 5, 6, 8] (sorted).

Time Complexity: O(n^2)for the worst case.

6a) What is space complexity? Write down the complexity of merge sort.
Answer:
Space Complexity:
Space complexity refers to the amount of memory an algorithm uses, including the input data and auxiliary space for variables, recursion, or temporary storage.

For **Merge Sort**, the space complexity is O(n)due to the temporary arrays required during the merging process.

Time Complexity:

- Best case: O(nlogn)
- Worst case: O(nlogn).

6b) Write and explain recursive binary search algorithm.
Answer:
Algorithm:

1. Input: Sorted array A[], element x, low = 0, high = size-1.
2. Find mid: mid=(low+high)/2]
3. If A[mid]==x, return mid.

4. If x<A[mid], search in the left half: BinarySearch (A, low, mid-1, x).
5. Else, search in the right half: BinarySearch(A, mid+1, high, x).

Time Complexity: O(logn).

6c) Write Divide and Conquer algorithm with detailed explanation.
Answer:
Divide and Conquer:
This technique breaks a problem into smaller sub problems, solves them recursively, and combines their results.

Steps:

1. **Divide:** Split the problem into smaller independent subproblems.
2. **Conquer:** Solve each sub problem recursively.
3. **Combine:** Merge the results to solve the original problem.

Example: Merge Sort.

- **Divide:** Split the array into two halves.
- **Conquer:** Sort each half recursively.
- **Combine:** Merge the sorted halves into a single array.

Time Complexity: O(nlogn).

7a) Explain path, cycle, and directed acyclic graph (DAG) with examples.
Answer:

1. **Path:** A sequence of edges connecting vertices in a graph without repeating any vertex. Example: In a graph with vertices {A, B, C}, a path is A → B → C.
2. **Cycle:** A path where the first and last vertices are the same, and no other vertex repeats. Example: A → B → C → A.
3. **Directed Acyclic Graph (DAG):** A directed graph with no cycles. Example: A graph with edges A → B, B → C, and no path returns to A. DAGs are used in scheduling and dependency resolution.

7b) Explain Directed Acyclic Graphs with examples.
Answer:
A **Directed Acyclic Graph (DAG)** is a directed graph with no cycles, meaning there is no path from a vertex back to itself.

Features:

1. Represents dependencies (e.g., tasks in scheduling).
2. Nodes represent tasks; edges show dependencies.

Example:
In task scheduling, if Task A depends on Task B, there is a directed edge B → A. A valid sequence would ensure no circular dependencies exist.

Q. State the conditions when linear search can be considered.

Linear search is suitable when:

1. The dataset is **unsorted** or **unordered**, making more complex algorithms like binary search inapplicable.
2. The size of the dataset is **small**, as the simplicity of linear search avoids overheads associated with more complex algorithms.
3. Searching is required for **all occurrences** of a target value in the dataset, as linear search checks each element sequentially.

Q.Write an algorithm for binary search. Explain why the complexity of binary search is O(log2n).

Algorithm for Binary Search:

1. **Input:** A sorted array A, target value x, and indices low=0, high=n−1.
2. While low≤high:
   a. Calculate mid=⌊(low+high)/2⌋
3. b. If A[mid]=x, return mid (target found).
   c. If A[mid]<x, set low=mid+1 (search the right half).
   d. Else, set high=mid− (search the left half).
4. If low>high, return -1 (target not found).

Explanation of O(logn) :
Binary search divides the search space into two halves at each step, reducing the problem size by half. For an array of size n, the number of divisions until a single element is reached is log2n. Hence, the time complexity is O(logn).

Explain the significance and advantage of height balancing of a binary search tree (BST).

Significance of Height Balancing:

1. In a BST, height balancing ensures that the height difference between the left and right subtrees of any node is at most 1. This avoids degeneration of the tree into a skewed structure.

Advantages of Height Balancing:

1. **Efficient Operations:** A height-balanced BST ensures O(logn) time complexity for insertion, deletion, and search operations, as the tree remains approximately balanced.
2. **Prevents Skewness:** Preventing skewness avoids the worst-case O(n) time complexity seen in unbalanced BSTs.
3. **Optimized Space Usage:** A balanced tree minimizes wasted memory by avoiding unnecessary empty levels.

Write a pseudo-code for the insertion operation in a binary search tree (BST).

Pseudo-code for BST Insertion:

1. **Input:** A BST root node and value x to insert.
2. If the tree is empty, create a new node with value x and set it as the root.
3. Otherwise, compare x with the root value:
   a. If x<root.value
      - Recursively insert x into the left subtree.
      b. If x>root.value:
      - Recursively insert x into the right subtree.
4. Return the root node after insertion.

How does Prim's algorithm work?

Prim's Algorithm Explanation:
Prim's algorithm finds the Minimum Spanning Tree (MST) of a graph by growing the MST one edge at a time.

Steps of the Algorithm:

1. **Initialize:** Start with a single vertex (arbitrary) and mark it as part of the MST.
2. **Priority Edge Selection:** Among all edges connecting vertices in the MST to vertices outside the MST, select the edge with the smallest weight.
3. **Add Edge to MST:** Add the selected edge to the MST and include its connected vertex into the MST.
4. **Repeat:** Repeat the process until all vertices are included in the MST.

Properties:

- It ensures the MST property by always adding the minimum-weight edge that doesn't form a cycle.
- The time complexity is O(ElogV) using a priority queue and adjacency list representation.

Example Visualization:
For a graph with vertices A,B,C,D:

- Start with A.
- Add edge (A,B) if it has the smallest weight.
- Then, add edge (B,C) and so on, maintaining the MST properties.

Explain Merge Sort algorithm with a suitable example.

(8 Marks)

Merge Sort Algorithm:
Merge Sort is a divide-and-conquer algorithm that splits the array into halves, recursively sorts each half, and then merges the sorted halves.

Steps:

1. **Divide:** Recursively split the array into two halves until each subarray contains one element.
2. **Conquer:** Sort each half while merging them.
3. **Combine:** Merge the two sorted halves into a single sorted array.

Algorithm:

1. If the array has one or zero elements, it is already sorted.
2. Split the array into two halves.
3. Recursively apply Merge Sort to each half.
4. Merge the two sorted halves.

Example:
Sort the array [38,27,43,3,9,82,10]:

1. Split: [38,27,43,3] [9,82,10].
2. Split further: [38,27,43,3],[9, 82,10].
3. Continue splitting: [38],[27] etc.
4. Merge: [38],[27]→[27,38];[38], [43],[3]→[3,43][43].
5. Merge further: [27,38],[3,43]→[3,27,38,43].
6. Merge the halves: [3,27,38,43],[9,10,82]→[3,9,10,27,38,43,82]

7. **Time Complexity:**

- Best, Worst, and Average: O(nlogn).

What is hashing?

(2 Marks)
Hashing is a technique for mapping data to a fixed-size value (hash) using a hash function. It allows efficient data retrieval in constant time O(1) for well-designed hash tables. Hashing is widely used in databases, caching, and searching.

Briefly discuss different collision resolution techniques.

(6 Marks)

When two keys map to the same hash value, a collision occurs. Common resolution techniques include:

1. **Chaining:**
   - Maintain a linked list at each index of the hash table to store all elements hashing to the same index.
   - Example: Keys 101010 and 202020 both hash to index 0. Store both in a linked list at index 0.
2. **Open Addressing:**
   - Use probing to find the next available index.
   - **Types of Probing:**
     a. **Linear Probing:** Check the next slot sequentially.
     b. **Quadratic Probing:** Use a quadratic function to find the next slot.
     c. **Double Hashing:** Use a secondary hash function to find the next slot.
3. **Rehashing:**
   - Increase the hash table size and rehash all elements when the table becomes full.
4. **Separate Addressing:**
   - Use a different data structure (like BST) for storing collisions.

1. Compute the hash value of the pattern and the first substring of the text.
2. Slide the pattern over the text one position at a time:
   - Update the hash value using a rolling hash formula.
   - Compare the hash values. If they match, verify character by character.
3. If the pattern matches, return the starting index.

Advantages: Efficient for multiple pattern searches.

Write down Dijkstra's Shortest-Path Algorithm.

(5 Marks)

Dijkstra's Algorithm:

1. **Initialize:**
   - Set the source vertex distance to 0 and all other vertices to infinity.
   - Mark all vertices as unvisited.
2. **Process:**
   - Select the unvisited vertex with the smallest distance.
   - Update the distances of its neighbors if a shorter path is found.
3. **Repeat:**
   - Mark the processed vertex as visited.
   - Repeat until all vertices are visited.
4. **Output:**
   - Shortest distances from the source to all other vertices.

Explain Dijkstra's algorithm with an example.

(3 Marks)

Example Graph:
Vertices: A,B,C,D,E.
Edges: A→B(2),A→C(4),B→C(1),B→D(7),C→E(3).

Steps:

1. Start at A: A=0, B=2,C=4,D=∞,E=∞.
2. Process B: Update C=3, D=9.
3. Process C: Update E=6.
4. Final shortest paths: A→B→C→E.

Write down Rabin-Karp algorithm for string matching.

(6 Marks)

Rabin-Karp Algorithm: