DAA Assignment No. 4

❖ 2 Marks Questions

Q.1.) Define the Divide-and-Conquer approach. (Remember)

Ans :- The Divide-and-Conquer approach is a problem-solving technique that breaks a problem into smaller sub-problems, solves each sub-problem independently, and then combines their solutions to solve the original problem.

Q.2.) List two advantages of using Divide-and-Conquer. (Understand)

Ans :- Two advantages of using Divide-and-Conquer are:

1. **Improved Efficiency**: It reduces the complexity of problems, often leading to faster algorithms (e.g., mergesort, quicksort).
2. **Parallelism**: Sub-problems can be solved independently, making it easier to implement parallel processing.

Q.3.) What is the time complexity of Merge Sort in the worst case? (Understand)

Ans :- The time complexity of Merge Sort in the worst case is (O(n \log n)).

Here's a brief explanation of why this is the case:

1. **Divide Step**: Merge Sort works by recursively dividing the array into two halves until each subarray contains a single element. This division takes (O(\log n)) time because the array is halved at each step.
2. **Conquer Step**: After dividing, the algorithm merges the subarrays back together. Merging two sorted arrays takes linear time, (O(n)), because each element from both arrays needs to be compared and placed in the correct order.

Combining these two steps, the overall time complexity can be expressed as:

[T(n) = 2T\left(\frac{n}{2}\right) + O(n)]

Q.4.) Differentiate between Quick Sort and Merge Sort in terms of recursion type. (Understand)

Ans :- Quick Sort and Merge Sort differ in their recursion types as follows:

Quick Sort:

- **Recursion Type**: **In-place** and **Divide-and-Conquer**.
- **Mechanism**: Quick Sort selects a "pivot" element and partitions the array into two subarrays: elements less than the pivot and elements greater than the pivot. It then recursively sorts the subarrays. The partitioning happens in place, meaning it does not require additional arrays for the subarrays.

Merge Sort:

- **Recursion Type**: **Divide-and-Conquer** with **additional space**.
- **Mechanism**: Merge Sort divides the array into two halves, recursively sorts each half, and then merges the sorted halves back together. This requires additional space for the temporary arrays used during the merging process.

Q.5.) What are the three main steps in a Divide-and-Conquer algorithm? (Remember)

Ans :- The three main steps in a Divide-and-Conquer algorithm are:

1. **Divide**: Split the problem into smaller subproblems that are similar to the original problem but smaller in size.
2. **Conquer**: Solve the subproblems recursively. If the subproblems are small enough, solve them directly.
3. **Combine**: Merge or combine the solutions of the subproblems to form a solution to the original problem.

❖ 10 Marks Questions

Q.1.) Illustrate the working of Merge Sort with an example and analyze its time complexity. (Apply)

Ans :- **Illustration of Merge Sort**

Let's illustrate the working of Merge Sort with an example array:

Example Array: ([38, 27, 43, 3, 9, 82, 10])

Step 1: Divide

1. Split the array into two halves:
    - Left: ([38, 27, 43])
    - Right: ([3, 9, 82, 10])
2. Continue dividing until each subarray contains a single element:
    - Left: ([38, 27, 43])
        - Split into ([38]) and ([27, 43])
            - Split ([27, 43]) into ([27]) and ([43])
    - Right: ([3, 9, 82, 10])
        - Split into ([3, 9]) and ([82, 10])
            - Split ([3, 9]) into ([3]) and ([9])
            - Split ([82, 10]) into ([82]) and ([10])

Now we have the following single-element arrays:

- ([38]), ([27]), ([43]), ([3]), ([9]), ([82]), ([10])

Step 2: Conquer (Merge)

Now we start merging the arrays back together in sorted order:

1. Merge ([27]) and ([43]):
    - Result: ([27, 43])
2. Merge ([38]) and ([27, 43]):
    - Result: ([27, 38, 43])

3. Merge ([3]) and ([9]):
    - Result: ([3, 9])
4. Merge ([82]) and ([10]):
    - Result: ([10, 82])
5. Merge ([3, 9]) and ([10, 82]):
    - Result: ([3, 9, 10, 82])
6. Finally, merge ([27, 38, 43]) and ([3, 9, 10, 82]):
    - Result: ([3, 9, 10, 27, 38, 43, 82])

Final Sorted Array

The sorted array is ([3, 9, 10, 27, 38, 43, 82]).

Time Complexity Analysis

1. **Divide Step**: The array is divided in half at each level of recursion, leading to a height of (O(\log n)) for the recursion tree.
2. **Conquer Step**: Merging two sorted arrays takes linear time, (O(n)), where (n) is the total number of elements being merged.

Combining these two steps, the overall time complexity can be expressed as:

[T(n) = 2T\left(\frac{n}{2}\right) + O(n)]

Using the Master Theorem, we find that the time complexity of Merge Sort is:

[O(n \log n)]

This time complexity holds for the worst case, average case, and best case scenarios.

Q.2.) Analyze the worst-case and best-case time complexities of Quick Sort and compare them with Merge Sort. (Analyze)

Ans :- **Quick Sort Time Complexity**

1. Worst-Case Time Complexity:

- The worst-case occurs when the pivot selection is poor, leading to unbalanced partitions. This can happen, for example, when the smallest or largest element is always chosen as the pivot in a sorted or nearly sorted array.
- In this case, the time complexity is: [O(n^2)]
- This is because the array is divided into one subarray of size (n-1) and another of size (0), leading to a recursive depth of (n).

2. Best-Case Time Complexity:

- The best-case occurs when the pivot divides the array into two equal halves, leading to balanced partitions.
- In this case, the time complexity is: [O(n \log n)]
- This is because the array is divided into two equal parts at each level of recursion, resulting in a logarithmic depth of recursion.

Merge Sort Time Complexity

1. Worst-Case Time Complexity:

- Merge Sort consistently divides the array into two halves and merges them, regardless of the input order.
- Therefore, the worst-case time complexity is: [O(n \log n)]

2. Best-Case Time Complexity:

- Similarly, the best-case time complexity for Merge Sort is also: [O(n \log n)]
- This is because the algorithm always divides and merges the arrays in the same manner, regardless of the initial order of elements.

Comparison of Quick Sort and Merge Sort

Aspect	Quick Sort	Merge Sort
Worst-Case Time Complexity	(O(n^2))	(O(n \log n))
Best-Case Time Complexity	(O(n \log n))	(O(n \log n))
Average-Case Time Complexity	(O(n \log n))	(O(n \log n))
Space Complexity	(O(\log n)) (in-place)	(O(n)) (requires extra space)
Stability	Not stable	Stable

Q.3.) Evaluate the efficiency of Binary Search compared to Linear Search. (Evaluate)

Ans :- **Efficiency Evaluation of Binary Search vs. Linear Search**

1. Definition:

- **Linear Search**: A simple search algorithm that checks each element in a list sequentially until the desired element is found or the list ends.
- **Binary Search**: A more efficient search algorithm that works on sorted arrays. It repeatedly divides the search interval in half, comparing the target value to the middle element of the array.

Time Complexity

Search Method	Best Case	Average Case	Worst Case
Linear Search	(O(1))	(O(n))	(O(n))
Binary Search	(O(1))	(O(\log n))	(O(\log n))

- **Linear Search**:
  - **Best Case**: The target element is the first element in the list, resulting in (O(1)).
  - **Average/Worst Case**: The search may need to check every element, leading to (O(n)).

- **Binary Search**:
    - **Best Case**: The target element is the middle element, resulting in (O(1)).
    - **Average/Worst Case**: The search space is halved with each comparison, leading to (O(\log n)).

Space Complexity

Search Method	Space Complexity
Linear Search	(O(1))
Binary Search	(O(1)) (iterative) or (O(\log n)) (recursive)

- Both search methods have a space complexity of (O(1)) for iterative implementations. However, a recursive implementation of Binary Search may use (O(\log n)) space due to the call stack.

Requirements

- **Linear Search**:
    - Can be used on both sorted and unsorted lists.
    - Simple to implement and understand.
- **Binary Search**:
    - Requires the list to be sorted beforehand.
    - More complex to implement than Linear Search.

Q.4.) Explain different Binary Tree traversal techniques (Inorder, Preorder, Postorder) with examples. (Apply)

Ans :- Binary tree traversal techniques are methods for visiting all the nodes in a binary tree. The three primary traversal techniques are Inorder, Preorder, and Postorder. Each technique visits nodes in a specific order.

1. Inorder Traversal

Definition: Inorder traversal visits the nodes in the following order: left subtree, root node, right subtree.

Algorithm:

1. Traverse the left subtree.
2. Visit the root node.
3. Traverse the right subtree.

Example: Consider the following binary tree:

Inorder Traversal Steps:

- Start at A, go left to B.
- Go left to D (no left child), visit D.
- Go back to B, visit B.
- Go right to E (no left child), visit E.
- Go back to A, visit A.
- Go right to C (no left child), visit C.

Inorder Result: D, B, E, A, C

2. Preorder Traversal

Definition: Preorder traversal visits the nodes in the following order: root node, left subtree, right subtree.

Algorithm:

1. Visit the root node.
2. Traverse the left subtree.
3. Traverse the right subtree.

Example: Using the same binary tree:

Preorder Traversal Steps:

- Start at A, visit A.
- Go left to B, visit B.
- Go left to D (no left child), visit D.
- Go back to B, go right to E (no left child), visit E.
- Go back to A, go right to C (no left child), visit C.

Preorder Result: A, B, D, E, C

3. Postorder Traversal

Definition: Postorder traversal visits the nodes in the following order: left subtree, right subtree, root node.

Algorithm:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root node.

Example: Using the same binary tree:

Postorder Traversal Steps:

- Start at A, go left to B.
- Go left to D (no left child), visit D.
- Go back to B, go right to E (no left child), visit E.
- Go back to B, visit B.
- Go back to A, go right to C (no left child), visit C.
- Finally, visit A.

Postorder Result: D, E, B, C, A

Q.5.) Explain the Divide-and-Conquer approach with an example. (Understand)

Ans :- **Divide-and-Conquer Approach**

The Divide-and-Conquer approach is a powerful algorithmic paradigm used to solve complex problems by breaking them down into smaller, more manageable subproblems. The main idea is to divide the problem into smaller subproblems, conquer each subproblem recursively, and then combine the solutions of the subproblems to form a solution to the original problem.

Steps in Divide-and-Conquer

1. **Divide**: Split the problem into smaller subproblems that are similar to the original problem but smaller in size.
2. **Conquer**: Solve the subproblems recursively. If the subproblems are small enough, solve them directly.

3. **Combine**: Merge or combine the solutions of the subproblems to form a solution to the original problem.

Example: Merge Sort

Let's illustrate the Divide-and-Conquer approach using the Merge Sort algorithm, which is a classic example of this paradigm.

Problem: Sort an array of integers.

Example Array: ([38, 27, 43, 3, 9, 82, 10])

Step 1: Divide

1. Split the array into two halves:
    - Left: ([38, 27, 43])
    - Right: ([3, 9, 82, 10])
2. Continue dividing until each subarray contains a single element:
    - Left: ([38, 27, 43])
        - Split into ([38]) and ([27, 43])
            - Split ([27, 43]) into ([27]) and ([43])
    - Right: ([3, 9, 82, 10])
        - Split into ([3, 9]) and ([82, 10])
            - Split ([3, 9]) into ([3]) and ([9])
            - Split ([82, 10]) into ([82]) and ([10])

Now we have the following single-element arrays:

- ([38]), ([27]), ([43]), ([3]), ([9]), ([82]), ([10])

Step 2: Conquer (Merge)

Now we start merging the arrays back together in sorted order:

1. Merge ([27]) and ([43]):
    - Result: ([27, 43])
2. Merge ([38]) and ([27, 43]):
    - Result: ([27, 38, 43])

3. Merge ([3]) and ([9]):
    - Result: ([3, 9])
4. Merge ([82]) and ([10]):
    - Result: ([10, 82])
5. Merge ([3, 9]) and ([10, 82]):
    - Result: ([3, 9, 10, 82])
6. Finally, merge ([27, 38, 43]) and ([3, 9, 10, 82]):
    - Result: ([3, 9, 10, 27, 38, 43, 82])

Final Sorted Array

The sorted array is ([3, 9, 10, 27, 38, 43, 82]).