<u>Answer Script</u>

Question No. 01
Write down the time complexity of Bubble sort, Insertion sort and Merge sort.
Answer No. 01

Sorting Algorithm	**Time Complexity in Best Case**	**Time Complexity in Average Case**	**Time Complexity in Worst Case**
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)

Question No. 02
Write two differences between array and linked-list. Why do we need a head/root node in a linked list?

Answer No. 02

Difference between Array and Linked-list:

Array	**Linked-list**
array allocates memory for all its elements in a contiguous block of memory	linked list allocates memory for each element separately, and each element contains a reference to the next element.
Elements can be accessed by using the index of the element.	Elements can only be accessed sequentially, starting from the first element for finding an elements at a specific index.

Why do we need a head/root node in a linked list?

Ans: A head/root node in a linked list is needed to keep track of the first element of the list. It is a reference to the first node in the list and allows us to traverse the list starting from the first element. Without a head/root node, we would not have a starting point to traverse the list, making it difficult to access any elements in the list.

Question No. 03

What is the basic idea behind the binary search algorithm, and how does it differ from linear search? What is the requirement for an array to be suitable for binary search?

Answer No. 03

What is the basic idea behind the binary search algorithm, and how does it differ from linear search?

Ans: The basic idea behind the binary search algorithm is to repeatedly divide the search interval in half. The algorithm begins by comparing the middle element of the array with the target value. If the target value matches the middle element, its position in the array is returned. If the target value is less than the middle element, the algorithm continues its search in the lower half of the array. If the target value is greater than the middle element, the algorithm continues its search in the upper half of the array. This process is repeated until the target value is found or the search interval is empty.

Binary Search	**Linear Search**
binary search algorithm divides the search interval in half with each iteration.	Linear search algorithm scans the entire array sequentially, starting from the first element, looking for the target value.
Time Complexity O(nlogn)	Time Complexity O(n)

What is the requirement for an array to be suitable for binary search?

Ans: The requirement for an array to be suitable for binary search is that it must be sorted. The binary search algorithm relies on the fact that the array is sorted in order to divide the search interval in half with each iteration. If the array is not sorted, the algorithm will not be able to determine which half of the array to search next, resulting in an incorrect output.

Question No. 04

What is the time complexity of inserting an element at the beginning of a singly linked list? What is the time complexity of inserting an element at any index of a singly linked list? What is the time complexity of deleting an element at the beginning of a singly linked list? What is the time complexity of deleting an element at any index of a singly linked list?

Answer No. 04

Tasks	**Time Complexity**
Inserting an element at the beginning of a singly linked list	O(1)
Inserting an element at any index of a singly linked list	O(n)
Deleting an element at the beginning of a singly linked list	O(1)
Deleting an element at any index of a singly linked list	O(n)

Question No. 05

Suppose we have an array of 5 integer numbers and a singly linked list of 5 integer numbers. Here the singly linked list of 5 integer numbers takes twice as much memory compared to array. Why is that? Give a proper explanation.

Answer No. 05

A singly linked list takes up more memory than an array in this case because each node in the linked list not only stores a value, but also a reference pointer to the next node.

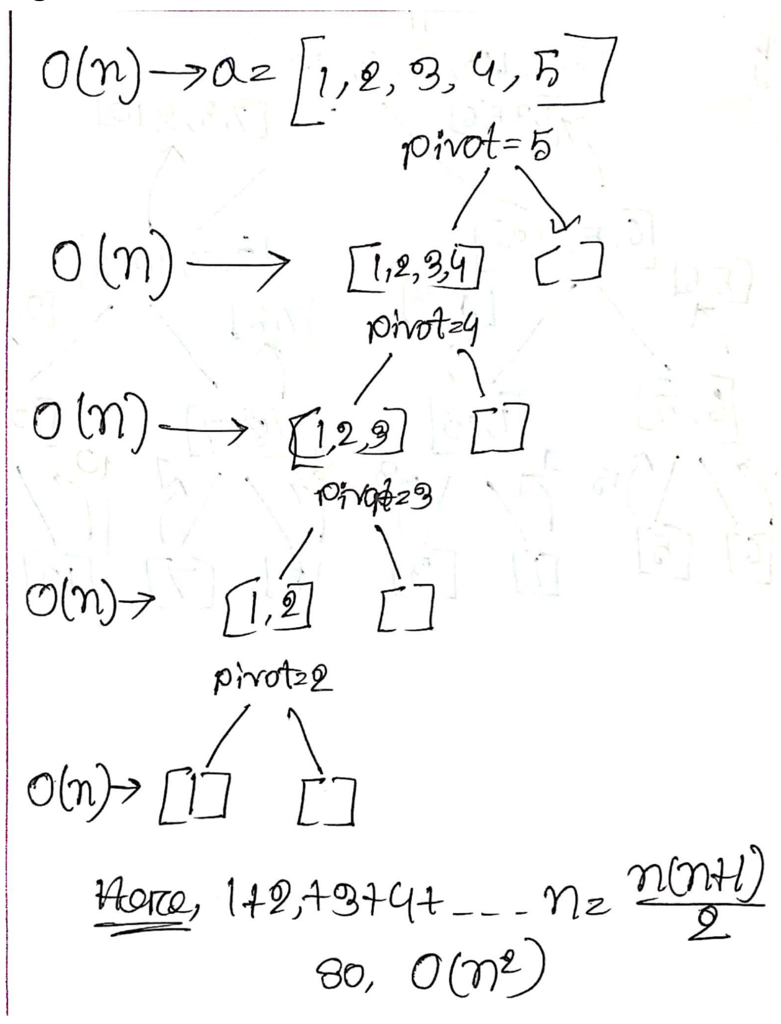
An **array** stores only the values and no references. So, for an array of 5 integers, it would take up to **(5 * the size of an integer)** in memory. While, in a **singly linked list** of 5 integers, each node would take up the size of an integer for the value, and the size of a pointer for the reference to the next node. So, it would take up to (1 + 1) = **2 * the size of an integer** in memory for each node, which is twice as much as an array.

Question No. 06
Derive the worst case and average case time complexity of Quick Sort with proper figures.

Answer No. 06

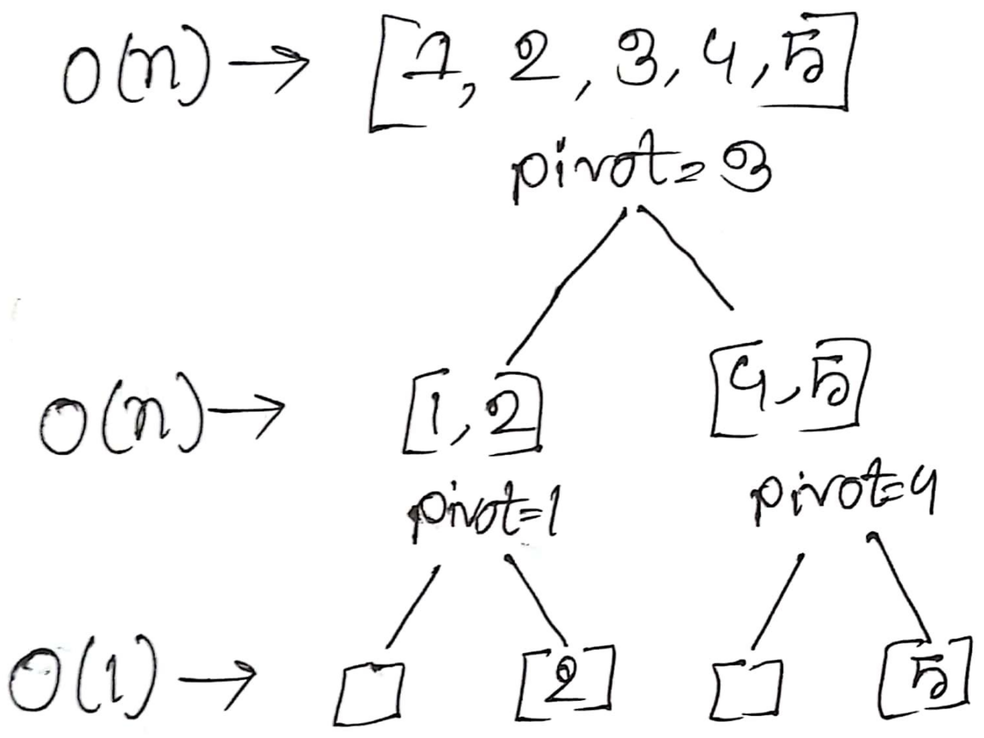
Worst Case:
The worst case time complexity occurs when the pivot element is chosen as the smallest or largest element in the array at each step of the algorithm. This results in one sub-array having n-1 elements and the other having 0 elements, leading to a maximum of n-1 levels of recursion. At each level of recursion, the algorithm performs a linear-time partition operation on the n elements of the sub-array, resulting in a total time complexity of O(n^2).

Figure:



Average Case:
The average case time complexity occurs when the pivot element is chosen in such a way that the resulting partition is roughly the same size. In this case, the algorithm divides the input array into two roughly equal-sized sub-arrays at each step, resulting in O(log n) steps (or "levels") of recursion. At each level of recursion, the algorithm performs a linear-time partition operation on the n elements of the sub-array, resulting in a total time complexity of O(n log n).

Figure:

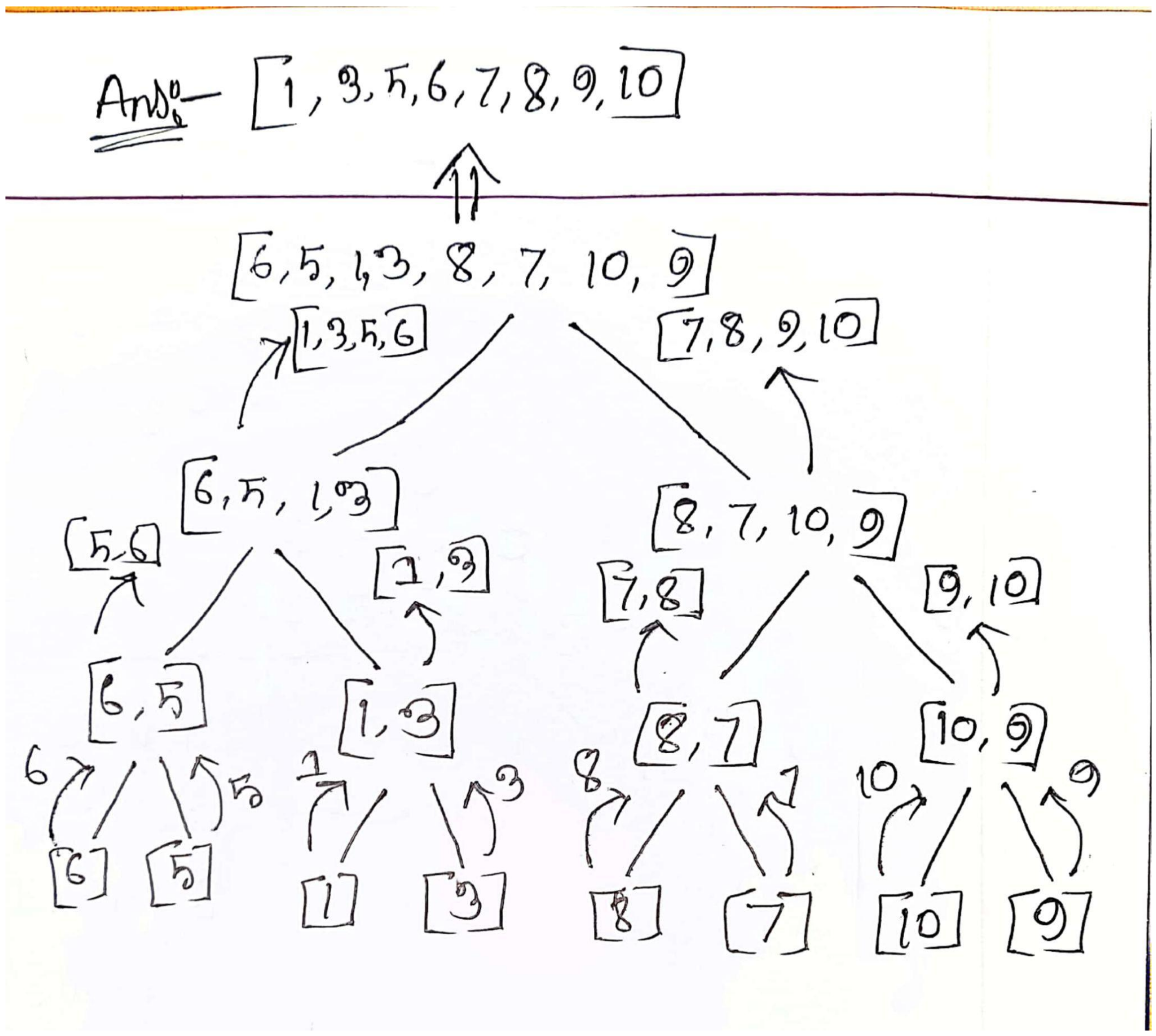


Level= log(n)
Here, n levels
∴ time complexity = O(n log n).

Question No. 07

Draw the recursion call tree with return values for Merge Sort in the array [6, 5, 1, 3, 8, 7, 10, 9]

Answer No. 07



Question No. 08

Write down the time complexity with proper explanation of the following code segment.

```
for (int i = 1; i * i <= n; i++) {
    if (n % i == 0) {
        cout<< i <<"\n";
        cout<< (n/i) <<"\n";
    }
}
```

Answer No. 08

This code segment has a time complexity of O(sqrt(n)) because it is iterating from i = 1 up to the square root of n (i * i <= n).

In each iteration of the loop, it checks if n is divisible by i (n % i == 0) and if so, it prints i and it divides n by i (n/i). As the maximum number of iterations the loop can go through is sqrt(n) times and for that the time complexity is O(sqrt(n)). This is because the number of times the loop will run is determined by the value of i*i and the maximum value of i*i will be n when i = sqrt(n) and in that case, the loop will run sqrt(n) times.

Question No. 09

Suppose you are working in a project where you need to do many random memory accesses and binary search. Array vs Linked list which is more suitable in this case? Why?

Answer No. 09

An array would be more suitable in this case because it allows for faster random memory accesses and binary search.

In terms of random memory access, an array stores its elements in a contiguous block of memory, so accessing an element at a specific index can be done in constant time, O(1). On the other hand, in a linked list, elements are not stored in contiguous memory, so finding an element at a specific index would require traversing the list from the beginning, resulting in a time complexity of O(n).

In case of binary search, in a sorted array, binary search can be applied on it which has a time complexity of O(log n) as it cuts the search space in half with each iteration. While in linked list, it's not possible to apply binary search, as the elements are not stored in contiguous memory and the time complexity of finding an element in a linked list is O(n)

Therefore, for a project that requires many random memory accesses and binary search, an **array** would be more suitable as it is more efficient in terms of time complexity.

Question No. 10
Alice is using a singly linked list to implement undo-redo functionality in a text editor. Bob advised Alice to use a doubly linked list in this scenario. Which approach seems more suitable to you? Give a proper explanation.
Answer No. 10
A doubly linked list would be more suitable in this problem for implementing undo-redo functionality in a text editor. A singly linked list is a linear data structure where each node contains a reference to the next node in the list. In the case of undo-redo functionality, if Alice is using a singly linked list, she would need to traverse the entire list to undo or redo. This would result in a time complexity of O(n) for undo and redo operations. On the other hand, a doubly linked list is also a linear data structure, but each node contains references to both the next and previous nodes in the list. This means that it is possible to traverse the list in both directions which makes the undo and redo operations more efficient. The time complexity for undo and redo operation in doubly linked list is O(1) as the previous node's reference is already available in each node.