ANSWER - 1:

1. **Bubble Sort:** It is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order.

   The time complexity of Bubble Sort is $O(n^2)$, where n is the number of elements in the array. This is because in the worst-case scenario, the algorithm needs to make n-1 comparisons for each of the n elements, leading to a total of n(n-1)/2 comparisons.

2. **Insertion Sort:** Insertion Sort is another simple sorting algorithm that works by dividing the input array into two parts: a sorted subarray and an unsorted subarray. The algorithm repeatedly takes elements from the unsorted subarray and inserts them into the correct position in the sorted subarray.

   The time complexity of Insertion Sort is $O(n^2)$, where n is the number of elements in the array. This is because in the worst-case scenario, the algorithm needs to make n-1 comparisons for each of the n elements, leading to a total of n(n-1)/2 comparisons.

3. **Merge Sort:** Merge Sort is a divide-and-conquer sorting algorithm that works by dividing the input array into two halves, sorting each half, and then merging the two sorted halves back together.

   The time complexity of Merge Sort is O(n log n), where n is the number of elements in the array. This is because the algorithm needs to make log n divisions to reach single elements, and each division requires n comparisons to merge the two sorted halves back together.

ANSWER - 2:

Two differences between arrays and linked lists are:

1. **Storage:** Arrays are stored in contiguous memory locations, while linked lists store data in non-contiguous memory locations, linked by pointers.

2. **Size:** Arrays have a fixed size, determined at the time of creation. Linked lists, on the other hand, can grow or shrink dynamically, as elements are added or removed.

A head/root node in a linked list serves as the starting point of the list. It is a pointer to the first node in the list, and is essential to access the rest of the elements in the list. The head node is also used to keep track of the length of the list and to locate the end of the list for inserting new elements.

ANSWER - 3:

The basic idea behind the binary search algorithm is to repeatedly divide a sorted array into two halves and search for a target element in the half that could potentially contain the target.

Binary search differs from linear search in that it takes advantage of the sorted nature of the array by eliminating half of the remaining elements in each iteration, rather than searching through all elements sequentially.

For an array to be suitable for binary search, it must be sorted in ascending or descending order. If the array is not sorted, the algorithm cannot determine which half of the array to search in and will not work correctly.

ANSWER - 4:

The time complexity of inserting an element at the beginning of a singly linked list is O(1), as it only involves creating a new node and updating its pointer to the head node to become the new head.

The time complexity of inserting an element at any index of a singly linked list is O(n), where n is the number of elements in the list. This is because, in the

worst-case scenario, the algorithm needs to traverse the entire list to reach the desired index, which takes O(n) time.

The time complexity of deleting an element at the beginning of a singly linked list is O(1), as it only involves updating the head node pointer to the next node in the list.

The time complexity of deleting an element at any index of a singly linked list is O(n), where n is the number of elements in the list. This is because, in the worst-case scenario, the algorithm needs to traverse the entire list to reach the desired index, which takes O(n) time.

ANSWER - 5:

An array of 5 integer numbers requires a contiguous block of memory to store its elements, which means that the size of the block of memory is equal to the number of elements in the array times the size of an integer. For example, if an integer takes 4 bytes, an array of 5 integers would take 20 bytes (5 * 4 = 20).

A singly linked list of 5 integer numbers, on the other hand, requires additional memory to store the pointers to the next node in the list. Each node in the linked list contains an integer element and a pointer, and the size of each node is equal to the size of the integer plus the size of a pointer. For example, if an integer takes 4 bytes and a pointer takes 8 bytes, a node in the linked list would take 12 bytes (4 + 8 = 12).

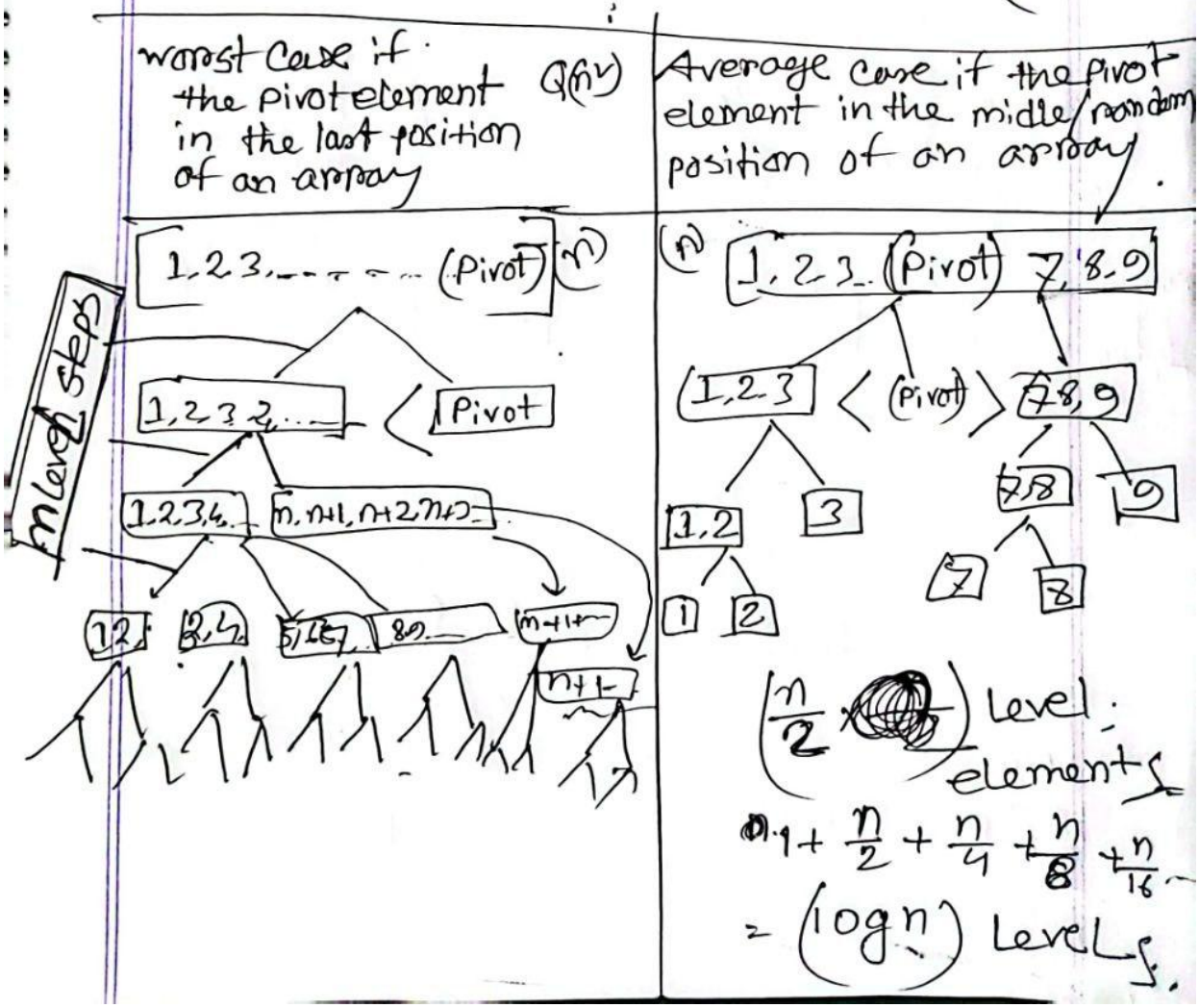
Therefore, a singly linked list of 5 integers takes 5 * 12 = 60 bytes of memory, which is 2 * 20 = 40 bytes more than an array of 5 integers. The additional memory overhead in a linked list is due to the need to store the pointers to the next node in the list, whereas an array does not have this requirement.

ANSWER - 6:

Answer – 6

(*) Worst case time complexity of QuickShort
$= O(n^2)$

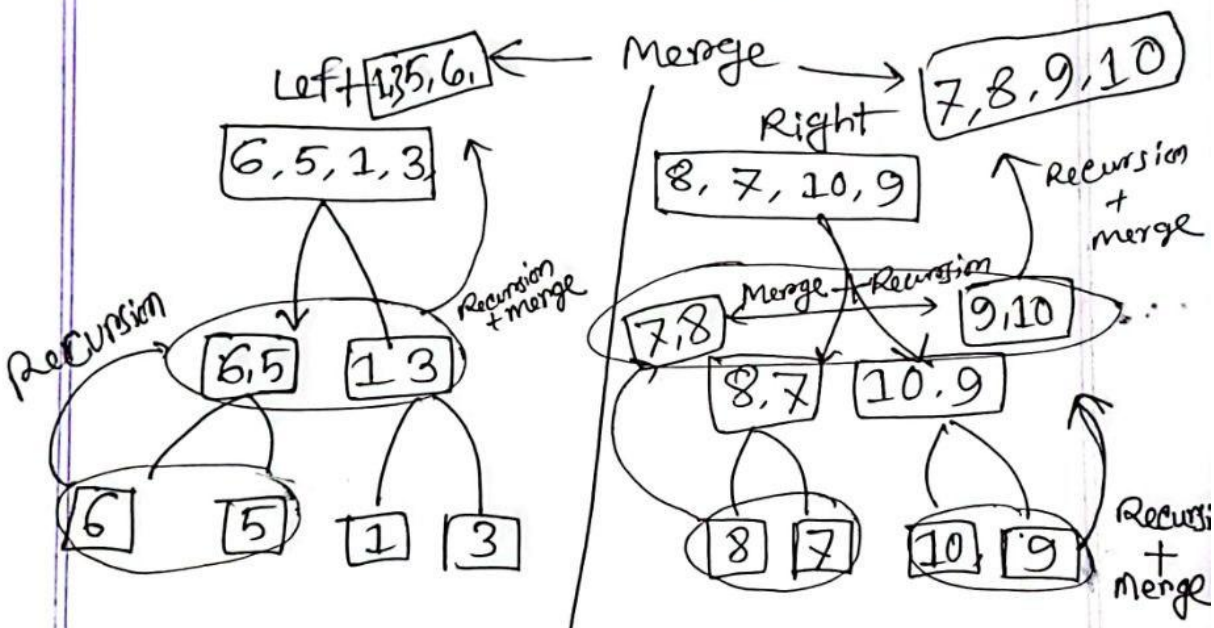
(*) Average case time complexity of QuickShort
$= O(n \log n)$



ANSWER - 7:

Answer - 7

[6, 5, 1, 3, 8, 7, 10, 9]



idx=0 ~~Output~~

1, 3, 5, 6 Left

7, 8, 9, 10 Right

idx2=0

1, 3, 5, 6, 7, 8, 9, 10 Final output

ANSWER - 8:

The time complexity of this code segment is O(√n).

The for loop runs from 1 to the square root of n. This means that it will run √n times. For each iteration, there is a check if n is divisible by i (n % i == 0). If it is, there are two cout statements. This means the overall time complexity of this code segment is O(√n).

ANSWER - 9:

In a project where many random memory accesses and binary search are needed, an array would be more suitable than a linked list. Random memory accesses in an array can be performed in constant time O(1), which means that the time to access an element in the array is independent of the size of the array. This is because the elements in an array are stored in a contiguous block of memory, and accessing an element in the array is as simple as calculating its memory address using the array's starting address and the index of the desired element.

In a linked list, on the other hand, each element is stored in a separate node, and accessing an element in the linked list requires following the pointers from node to node, which takes linear time O(n).

Binary search is a search algorithm that works by dividing a sorted array into two halves, and then checking if the target element is in the left or right half. This process is repeated until the target element is found or it is determined that it does not exist in the array. Binary search requires that the elements in the array be sorted, and it has a time complexity of O(log n), which is much faster than linear search.

In conclusion, for a project where many random memory accesses and binary search are needed, an array would be more suitable because it provides constant-time access to elements and supports efficient binary search.

ANSWER - 10:

A doubly linked list is more suitable for implementing undo-redo functionality in a text editor compared to a singly linked list.

Undo-redo functionality requires traversing the list of changes in both forward and backward directions, which can be easily achieved with a doubly linked list. In a doubly linked list, each node contains a pointer to both the previous and next nodes in the list, allowing for easy traversal in both directions.

In a singly linked list, on the other hand, each node only contains a pointer to the next node in the list, making it difficult to traverse the list in reverse order. To traverse the list in reverse order in a singly linked list, one must keep track of the entire list or store a separate pointer to the last node, which adds additional overhead and complexity.

Therefore, a doubly linked list is more suitable for implementing undo-redo functionality in a text editor, as it provides the ability to traverse the list of changes in both forward and backward directions.