Answer Script

Question No. 01
Write down the time complexity of Bubble sort, Insertion sort and Merge sort. 1
Answer No. 01

Time complexity of Bubble sort:

```
void bubble sort (int arr[ ],int size)
{
  for(int i=0;i<size;i++)
{
  for(int j=0;j<size-i-1;j++)
  {
    if(arr[j+1]<arr[j])
    {
      swap(arr[j+1],arr[j])
    }
  }
}
}
```

Here first loop time complexity is O(n). and inner loop time complexity worse case also O(n).
so oberall time complexity will be worse case O(n*n) or, O(n^2).

Time complexity for InsertionSort:

```
void InsertionSort(int arr[ ],int size)
{
   for(int i=1;i<size;i++)
   {
      int index=i;
      while(index>=1)
      {
         if(arr[index]<arr[index-1])
         {
            swap(arr[index],arr[index-1]);
            index--;
         }
      }
   }
}
```

Here first loop time complexity is O(n). and 2nd while loop also works like outer loop and
how long swaping value it alse depend on
dataset and this loop time complexity worse case alse O(n).
new time complexity is O(n*n) or, O(n^2).

Time complexity of Merge Sort:

Merge sort working divide and conquer algorithm based on the idea of
breaking down a list into several sub-lists until each sublist
consists of a single element and merging those sublists in a manner that results into a sorted list.

so Merge sort time complexity worse case is 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

Arrays:
1. All index or node are list stored sequentilly.
2. An array all data are same tipe like if array integer type then all value is integer type.

LinkedList:
1. All index or node are not list stored sequentilly.
2. In LinkedList is collection of object known as node where node consists two or three part i.e., previus node address,
data , next node address.

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

Head node is most importent becouse when we access linked list where node will come first and every node is
connected like train so first node is head.
if we know the head then we can access full node or and node of linked list.
if we add value or delete value at taill then also need head. we go last node trough head.
if a link list don't have haed then it look like a human waithout head.
so head is the most import pointer of the linked 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

The basic ide of binary search is array must be sorted and we find the mid index value which is equal to target value.

If this value is less then target value then we ignor this value which is less than and equal mid index value.

If this value is gatter than target value than we ignor this value which is gatter than and equal mid value.

after complete the loop we will find the target value and this algorithm time complexity is O(logn).

In linear search is deffer because when find a value using this algorithm. this algorithm check all value is equal to

target value or not if this value is last of the index then it will take so much time to get the target value.

commont different binary swach and binary search is: binary search is check all value to get target value but linear search algorithm will check sequentilly to get target value.

The requirement for Array must be sorted otherwise it not working poperly.

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

1. What is the time complexity of inserting an element at the beginning of a singly linked list?

Answer: The time complexity of inserting an element at the beginning of a singly linked list O(1).

because we just create new node and connected to the head no need any loop so time complexity is O(1).

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

the time complexity of inserting an element at any index of a singly linked list worse case O(n).

because first we need to go before index node and connected to new node with this node so need a loop

to this loop thats-why the worse case time complexity is O(n).

3. What is the time complexity of deleting an element at the beginning of a singly linked list?

The time complexity of deleting an element at the beginning of a singly linked list O(1).

because we just delete head and head connecte to next node so time complexity is O(1).

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

the time complexity of deleting an element at any index of a singly linked list worse case O(n).

because firstly, we need to go before node index and delete next index and connected next node with previus node so need a loop

to this loop thats-why the worse case time complexity is O(n).

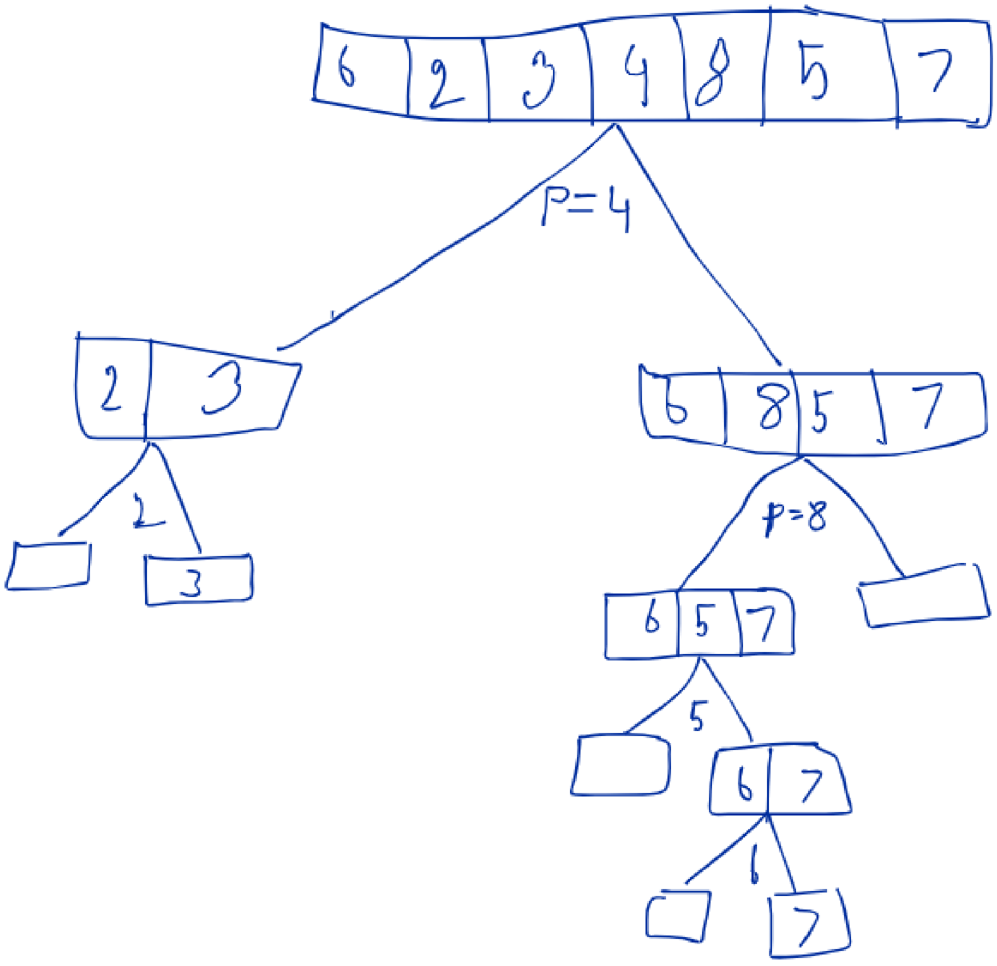
Question No. 05
Suppose we have an array of 5 integer numbers and a singly linked list of 5 integer numbers. Here the singly linked list of 5 integer numbers takes twice as much memory compared to array. Why is that? Give a proper explanation.
Answer No. 05
single LinkedList one node has two part first one is data and another one is next node address. Since there are 5 nodes in the list, this means that we need to store 5 integer values and 5 pointers or references. Assuming that each integer value and pointer takes the same amount of memory (e.g. 4 bytes on a 32-bit system), the total memory required to store the singly linked list would be: 5 integers x 4 bytes/integer + 5 pointers x 4 bytes/pointer = 40 bytes but the 5 integer of array required memory is 5 integers x 4 bytes = 20 bytes. Therefore, the singly linked list of 5 integer numbers takes twice as much memory as the array because it

requires additional memory to store pointers or references to the next element in the list.

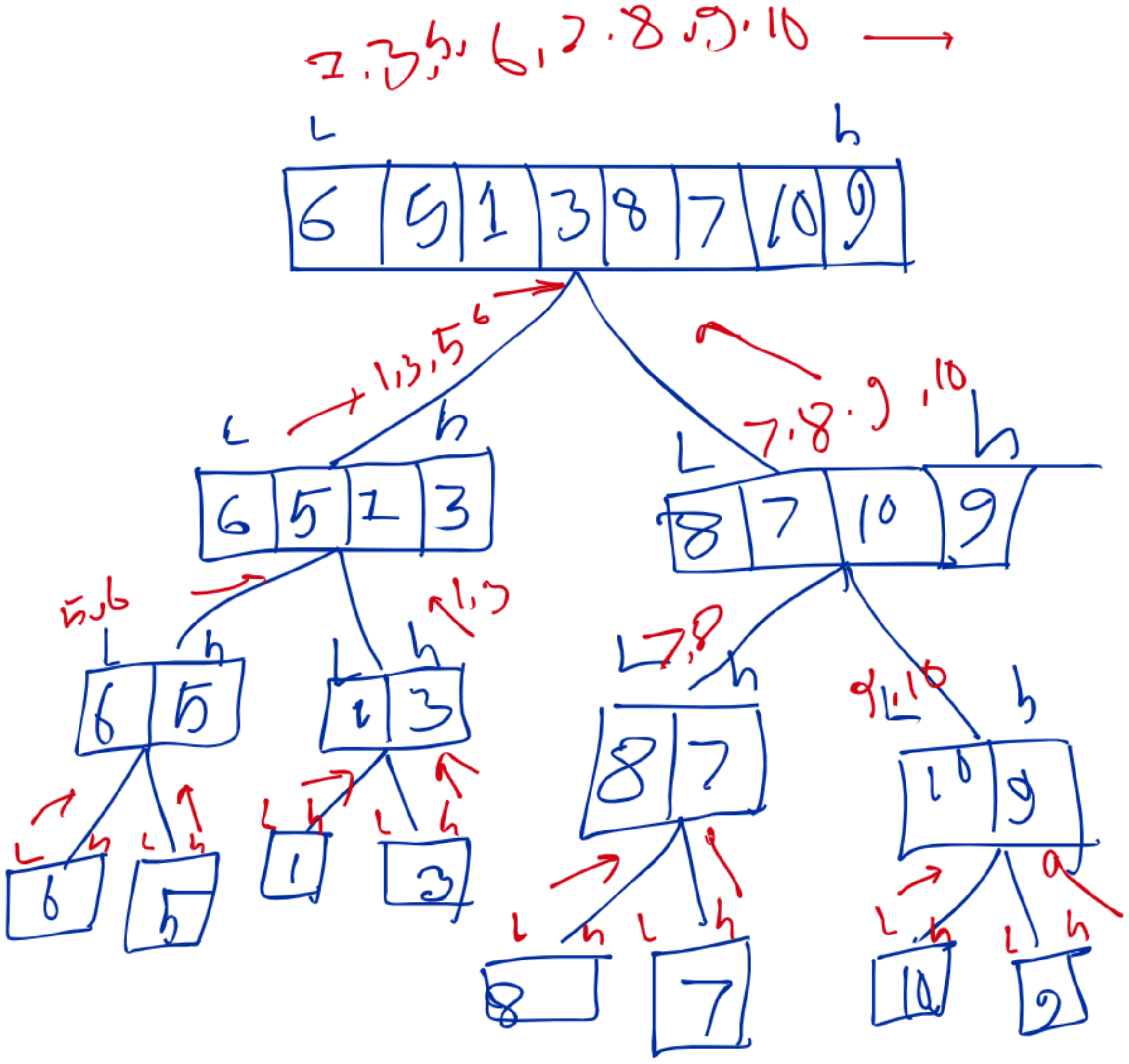
Question No. 06
Derive the worst case and average case time complexity of Quick Sort with proper figures.
Answer No. 06
In worst case time complexity: In the worst scenario, each partition results in only one element being placed in the left or right subarray, depending on whether the pivot element is the smallest or biggest in the array. In this scenario, the Quick Sort recursion tree is converted into a skewed tree with n levels, where n is the array's length. In this situation, Quick Sort's worst-case time complexity is O(n2). In average case time complexity: Quick Sort's average case time complexity, which is O(nlog(n)), is substantially better than its worst case. This is due to the fact that, on average, the pivot element will be picked at random and divide the array

into two portions that are about equal. Next, using Quick Sort, each of these components will be sorted
recursively, with a final time complexity of O(nlog(n)).

Here i chose pivot in mid index value of the array.



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

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

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

for example if n=10 then loop will be exquete 3 times where sqrt(10)=3.
so the time complexity is O(sqrt(n)),

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

9. 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?

If i need to do many random memory accesses and binary search, an array is generally

more suitable than a linked list.

because of this is that arrays provide constant-time random access to elements, while linked lists require

linear time to access elements at arbitrary positions.

Binary search requires random access to elements, which means that it is much more efficient to perform it on

an array than on a linked list. With an array, i can access any element in constant time by using its index.

However, with a linked list, i need to traverse the list from the beginning each time i want to access an element at a specific position, which takes linear time.

Therefore, if i need to perform many random memory accesses and binary search, an array would be a better

choice than a linked list. However, if i frequently need to insert or delete elements from the middle of

the list, a linked list may be more appropriate.

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
single LinkedList allow to go only one direction and doubly LinkedList allow go forword and previus in doubly LinkedList node has three part first part has previus node address and mid has node data nad last part node contain next node address. and the The reason for this is that a doubly linked list allows to traverse

the list in both directions. This means that if i need to undo an operation, i can simply go back one

step by following the previous pointer of the current node. Conversely, if i need to redo an operation,

i can go forward one step by following the next pointer of the current node.

on the othar hand, with a singly linked list, i need to keep track of both the

current node and the previous node in order to undo an operation. This is because a singly linked list

only allows you to traverse the list in one direction, so i cannot easily go back one step.

Therefore, Bob's advice to use a doubly linked list in this scenario is appropriate, as it provides more

efficient and flexible functionality for implementing undo-redo operations in a text editor.