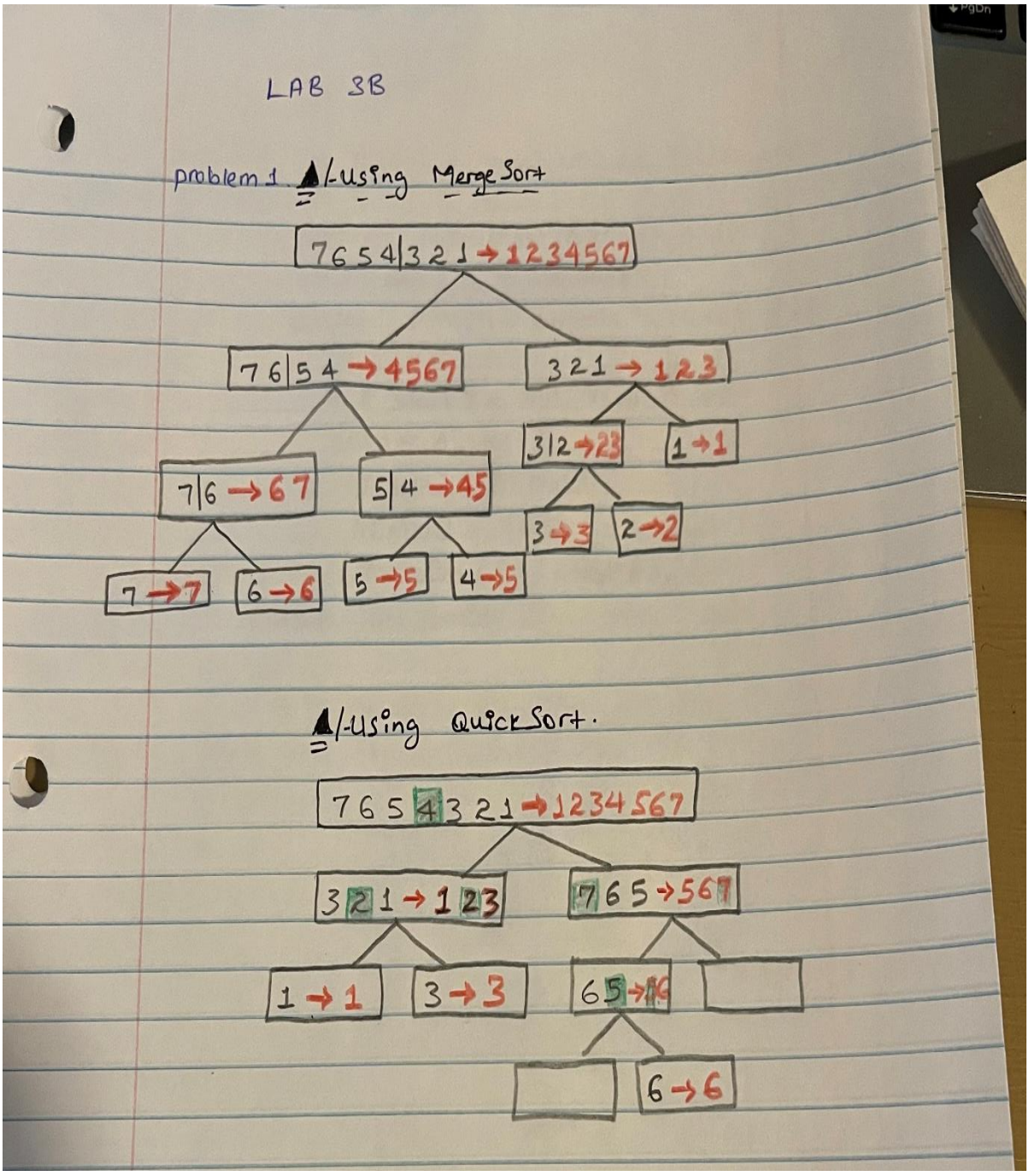
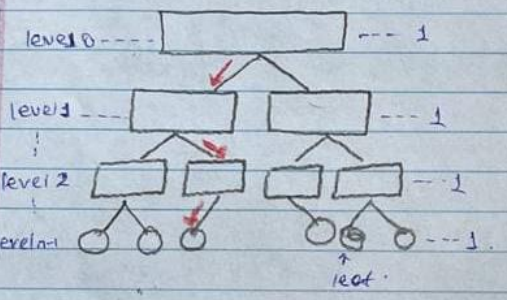
Lab 3B


LAB 3B
problem 1 A/- using Merge Sort
7654321 → 1234567
7654 → 4567
321 → 123
76 → 67
54 → 45
32 → 23
1 → 1
7 → 7
6 → 6
5 → 5
4 → 5
3 → 3
2 → 2
A/- using Quick Sort.
7654321 → 1234567
321 → 123
765 → 567
1 → 1
3 → 3
65 → 6
6 → 6

problem 2.

Binary search algorithm only searchs at one side of the list. element is compared with the middle element if larger search right if smaller search left if equal to mid element return true and if not found return false

Note: work done at each step is 1 which is comparing the mid element with the given number to be searched.



the worst case in this algorithm is if we are looking for element that doesn't exist on the tree. at each level there is $O(1)$ work done and hight of the tree is $O(\log n)$

$$T(n) = O(1 * \log n)$$
$$= O(\log n)$$

Problem 3 : recursive algorithm to reverse order.

Algorithm reverseOrder (A, start, end)
Input : Array of n elements
output : Array of n elements in reversed order.

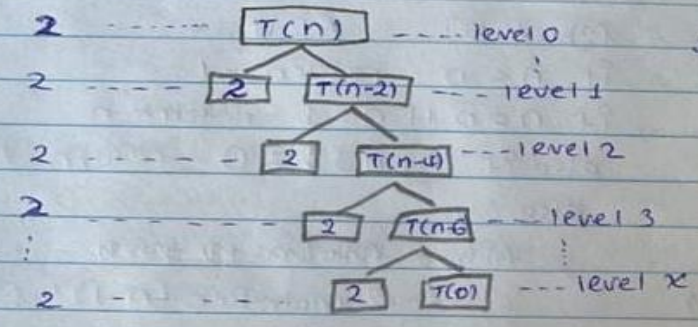
if start >= end then return + c
temp ← A[start] ..... + c
A[start] ← A[end] ..... + c
A[end] ← temp ..... + c
reverseOrder (A, start + 1, end - 1) .... T(n-2)
Compute the running time using count-sely cols.

- in the given algorithm we can observe that the next method (recursive) call is smaller by 2 than the first one, because start +1 and end -1

$$T(n) = \begin{cases} 2 & n = 0 \\ T(n-2) + c & \text{otherwise} \end{cases}$$

Because we don't have formula to resolve this I will use the tree method.

Work done at each step



So we will have
$2 + 2 + 2 + 2 \ldots + 2$ (x times)
$\therefore 2x$

Assume
$n - x = 0$
$2x = 2n$
$T(n) = 2n$
$O(n)$

Problem 4: Divise an iterative Algorithm for fibonacci that runs at O(n) time.

Algorithm fibonacciIterative (n)
Input: a non negative integer n
output: the fibonacci value at the n^{th} sequence

```
fib ← 1 ---------- +1
pfib ← 1 --------- +1
for i ← 2 to n-1 do ... n
    temp ← fib ---- n-1
    fib ← fib + pfib --- n-1
    pfib ← temp --- n-1
return fib ------ +1
```

4n

$T(n) = 4n \Rightarrow O(n)$

?

1- recursive Algorithm O(n) time.

Algorithm: fibonacciRecWithMemoization(n)
Input: a non negative integer n
Output: the fibonacci value at the n^{th} sequence (pos)

```
C ---- fib ← 0;      A[]
C ---- if n < 0  return -1
C ---- if n = 0 || n = 1  return n
C ---- else if A[n] != 0  return A[n]
       else
T(n-2) ---- fib ← fibonacciRec(n-1) + fibonacciRec(n-2)
       A[n] ← fib
       return fib
```

$$T(n) = \begin{cases} C & n = 1 \\ T(n-2) & \text{otherwise} \end{cases}$$

problem 3 like the privous tree count this is O(n) time

Problem 5 ▲1- We showed that Second Smallest can be solved O(n) time, can that be used to find third smallest. in O(n) time?

Yes, it can be used to solve the third problem in O(n) time. Because looping from 0 to 3 on the outer loop is going to take constant time. it is not related to the input size n. no matter how large n is the outter loop will iterate 3 times to find the 3rd Smallest.

▲1- Can this be used k^{th} element to run O(n)?

NO, Because n can range from some c to n-1. the running time is going to be closer and closer to $O(n^2)$ time.

▲1- fast algorithm to find the $n/2$ smallest element ?

I believe it can be done using $O(n \log n)$ time By improving using Quicksort or mergesort, I don't think we can do any better than this for now!

▲1- is the sorting approach the fastest way in this case?

Yes, I think to find the k^{th} smallest element in a given sequence n, it need to be sorted and the $k-1^{th}$ element is the k^{th} smallest