Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 4 REPORT

STUDENT NAME
Zeynep Nazire YÜKSEL

STUDENT NUMBER
161044068

Course Assistant
Ayşe Şerbetçi TURAN

Q1)

a)

```
public static LinkedList<Integer> findMaxList(LinkedList<Integer> list) {
    LinkedList<LinkedList<Integer>> myList = new
LinkedList<LinkedList<Integer>>();
    LinkedList<Integer> temp = new LinkedList<>();
    for (int i = 0; i < list.size(); ++i) {
        if (i + 1 != list.size() && list.get(i) < list.get(i + 1)) {
            temp.add(list.get(i));
        } else if (i + 1 == list.size()) {
            temp.add(list.get(i));
            myList.add(temp);
        } else {
            temp.add(list.get(i));
            myList.add(temp);
            temp = new LinkedList<>();
        }
    }
    LinkedList<Integer> maxList = new LinkedList<>();
    int maxSize = myList.get(0).size();
    for (int j = 1; j < myList.size(); ++j) {
        if (maxSize < myList.get(j).size()) {
            maxSize = myList.get(j).size();
            maxList = myList.get(j);
        }
    }
    return maxList;
}
```

Firstly I split the list into sorted sublists.Then I return sublist that has maximum length.

First for turns list size(n) times so its complexity is O(n).Second for turns less than first for because sublist number is smaller than list size.Therefore functions's complexity is O(n + m(m<n)) = O(n).

b)

```
public static LinkedList<Integer> findListRec(LinkedList<Integer> list,
LinkedList<Integer> temp, LinkedList<LinkedList<Integer>> myList, int i) {
        if (i == list.size()) {
            LinkedList<Integer> returnlist = new LinkedList<>();
            int maxSize = myList.get(0).size();
            int n=0;
            for (int j = 1; j < myList.size(); ++j) {
                if (maxSize < myList.get(j).size()) {
                    maxSize = myList.get(j).size();
                    n=j;
                }
            }
            returnlist = myList.get(n);
            return returnlist;
        }
        if (i + 1 != list.size() && list.get(i) < list.get(i + 1)) {
            temp.add(list.get(i));
        }
        else if (i + 1 == list.size()) {
            temp.add(list.get(i));
            myList.add(temp);
        }
        else{
            temp.add(list.get(i));
            myList.add(temp);
            temp = new LinkedList<Integer>();
        }
        return findListRec(list, temp, myList, i+1);
    }
}
```

Firstly to split the list into sorted sublists , function calls itselfs and all time i increases until it reach list size. Therefore I obtain sorted sublists. Then when i equals list size, returned sublist that has maximum length.

M = number of sublists.

Proof of Induction

$T(n) = n*(n+1)/2$

1.Base Case : $n = 1 : 1*2/2 = 1$ (True)

2. Induction Hypothesis : Assume $T(n) = n*(n+1)/2$ true.

3. Prove $T(n+1)$ is true

4. $T(n) = (n+1)*(n+1+1)/2$

So $T(n) = O(n^2)$

Proof of Recurrence

$T(n) = T(n-1) + n \quad T(1) = O(m)$ (m is number of sublists)

$= T(n-2) + n-1 + n$

$= T(n-3) + n-2 + n-1 + n$

$= T(1) + n-3 + n-2 + n-1 + n = n*(n+1)/2 = (n^2 + n)/2 = O(n^2) \qquad (m < n)$

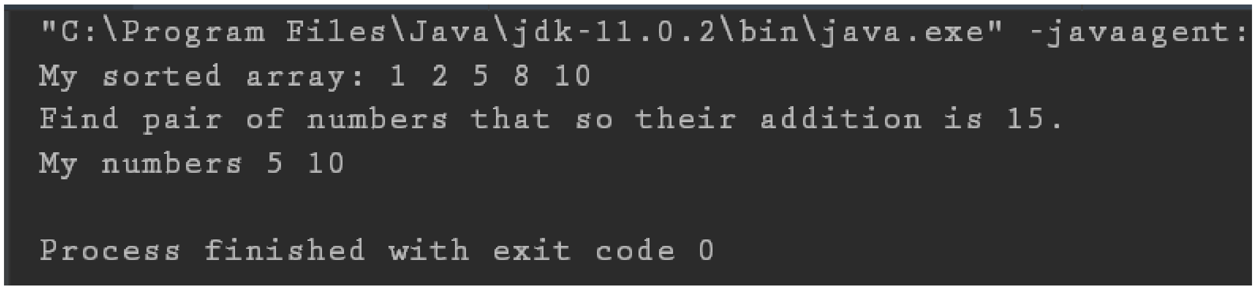
2)

```
public static void findPairNumber(int[] arr, int find) {
    int begin = 0;
    int end = arr.length - 1;

    while (begin < end && arr[begin] + arr[end] != find) {
        if (arr[begin] + arr[end] < find)
            begin++;
        else
            end--;
    }
    if (begin != end)
        System.out.println("My numbers " + arr[begin] + " " + arr[end]);
}
```

I create two variables so that their names are begin and end. Begin initializes with head of array and end initializes with end of array. If addition of begin and end is smaller than given number, begin increases one. If addition of begin and end is bigger than given number, end decreases one. This process continue until addition of begin and end is not equals given number and begin is smaller than end. Finally if numbers be present, they print.

My Output :



```
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" -javaagent:
My sorted array: 1 2 5 8 10
Find pair of numbers that so their addition is 15.
My numbers 5 10

Process finished with exit code 0
```

This function's worst complexity is Θ(n) because if addition of begin and end is continuous smaller than given number or is continuous bigger than given number, begin or end increases or decreases n times either both them increases and decreases and total turns n/2 times so worst complexity is Θ(n) = Θ(n/2).This function's best complexity is Θ(1) because if addition of first and end elements of array is equals given number, while turns once.

3)

for (i=2*n; i>=1; i=i-1) → this for turns 2*n times.
for (j=1; j<=i; j=j+1) → this for turns i times(so 2*n).
for (k=1; k<=j; k=k*3) → this for turns log(2*n) base 3 times.
print("hello") → this turns once.

They are inner so I multiply their turn numbers.

T(n) = O(2*n*2*n*log(n)) = O(4n^2*log(2*n)) = O(n^2 * log(2*n)) Note : log(n) base 3

4)

$T(n) = T(n/2) + T(n/2) + T(n/2) + T(n/2) + n^2/4$

$T(n) = 4T(n/2) + 1 = 4^1 T(n/2) + n^2/4$

$= 4 (4T(n/4) + n^2/4) + n^2/4 = 4^2 T(n/4) + 4*n^2/4 + n^2/4$

$= 4^2 (4T(n/8) + n^2/4) + 4*n^2/4 + n^2/4$

$= 4^3 T(n/8) + 16* n^2/4 + 4*n^2/4 + n^2/4$

$= 4^3 T (4T(n/16) + n^2/4) + 16* n^2/4 + 4*n^2/4 + n^2/4$

$= 4^k T(n/2^k) + \sum 2^2k*n^2/4$

$n/2^k = 1 \rightarrow k = \log(n)$ base 2.

$4^{\log(n)} T(n) + 2^2k = n^2 + 2^2\log(n)*n^2$

$2^2\log(n)*n^2$ is bigger than n^2 so complexity is $2^2\log(n)*n^2$