Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 4 REPORT

Can BEYAZNAR
161044038

Question 1:

To solve this problem, we first create a variable that holds our longest list. Then we send this list to the list of elements in the list given to us. After doing this, we keep the variable in a new list type and continue the same process as the list given to us. If the length of the second list variable is longer than the variable that holds the longest list, the second list is assigned to the longest list. We implement this solution in iterative and recursive code.

A) Iterative Function

The time complexity is O(n).

```java
public static LinkedList<Integer> findMaxList(LinkedList<Integer> myList){
    LinkedList<Integer> maxList = new LinkedList<Integer>();
    maxList.add(myList.get(0));
    int i=1;
    boolean control=true;
    while(i<myList.size() && control)
    {
        if(myList.get(i) >= maxList.getLast() )
        {
            maxList.add(myList.get(i));
            i++;
        }
        else
            control=false;
    }
    LinkedList<Integer> tempList = new LinkedList<Integer>();
    tempList.add(myList.get(i));
    i++;
    while(i<myList.size())
    {
        if( myList.get(i) >= tempList.getLast() )
            tempList.add(myList.get(i));
        else
        {
            if(maxList.size() < tempList.size() )
                maxList = new LinkedList<Integer>(tempList);
            tempList = new LinkedList<Integer>();
            tempList.add(myList.get(i));
        }
        i++;
    }
    if(tempList.size() > maxList.size())
        maxList = new LinkedList<Integer>(tempList);

    return maxList;
}
```

Time Complextiy : O(n)

Time Complextiy : O(n)

```
Our List:
[1, 9, 2, 7, 20, 13]
Result:
[2, 7, 20]
```

B) Recursive Function

Lets say our list is {8,7,6,5,4,3,2,1}, if we call our recursive function, it call itself 8 times without loops. Thus our worst case is O(1)+T(n-1).

On the other hand, lets say our list is {1,2,3,4,5,6,7,8} and if we call our function, this list continues 8 times in the while loop. And it calls itself one time because our “currentIndex” variable equals to list’s size. Thus our best case is O(n)+T(1).

These two results are the same, so the result is:

T(n) = O(1) + T(n-1) we can not solve this equation with master theorem. Because our equation is not suitable for master theorem.

We can solve it with using induction:

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

= 2 + T(n-2)
= 3 + T(n-3)
= …

= n + T(0) = O(n)

```
public static LinkedList<Integer> findMaxList_Recursive(LinkedList<Integer> myList, LinkedList<Integer> maxList, int currentIndex)
{
    if(currentIndex == myList.size())
        return maxList;
    if(maxList.isEmpty())
    {
        maxList.add(myList.get(currentIndex));
        currentIndex++;
        while(currentIndex < myList.size() && maxList.getLast() <= myList.get(currentIndex) )
        {
            maxList.add(myList.get(currentIndex));
            currentIndex++;
        }
    }
    else if( maxList.getLast() >= myList.get(currentIndex))
    {
        LinkedList<Integer> tempList = new LinkedList<Integer>();

        tempList.add(myList.get(currentIndex));
        currentIndex++;
        while(currentIndex < myList.size() && tempList.getLast() <= myList.get(currentIndex)
        {
            tempList.add(myList.get(currentIndex));
            currentIndex++;
        }
        if(tempList.size() > maxList.size())
            maxList = tempList;

    }
    return findMaxList_Recursive(myList,maxList,currentIndex);
}
```

Time complexity: O(n)

Time complexity: O(n)

```java
public static void main(String args[])
{
    LinkedList<Integer> testList = new LinkedList<~>();
    testList.add(1);
    testList.add(9);
    testList.add(2);
    testList.add(7);
    testList.add(20);
    testList.add(13);

    System.out.println("Our List: ");
    System.out.println(testList);

    LinkedList<Integer> result = new LinkedList<Integer>();
    LinkedList<Integer> maxList = new LinkedList<~>();
    result=findMaxList_Recursive(testList,maxList,  currentIndex: 0 );
    System.out.println("Result: ");
    System.out.println(result);
```

```
Our List:
[1, 9, 2, 7, 20, 13]
Result:
[2, 7, 20]
```

Question 2:

public boolean findSum(int[] array, searchingNum)

set startArr to 0
set endArr to array's length-1

while the startArr is not bigger than endArr

if the sum of array[startArr] and array[endArr] is equal to searchingNum

print startArr and endArr
return true

else if the sum of array[startArr] and array[endArr] is bigger than searchingNum

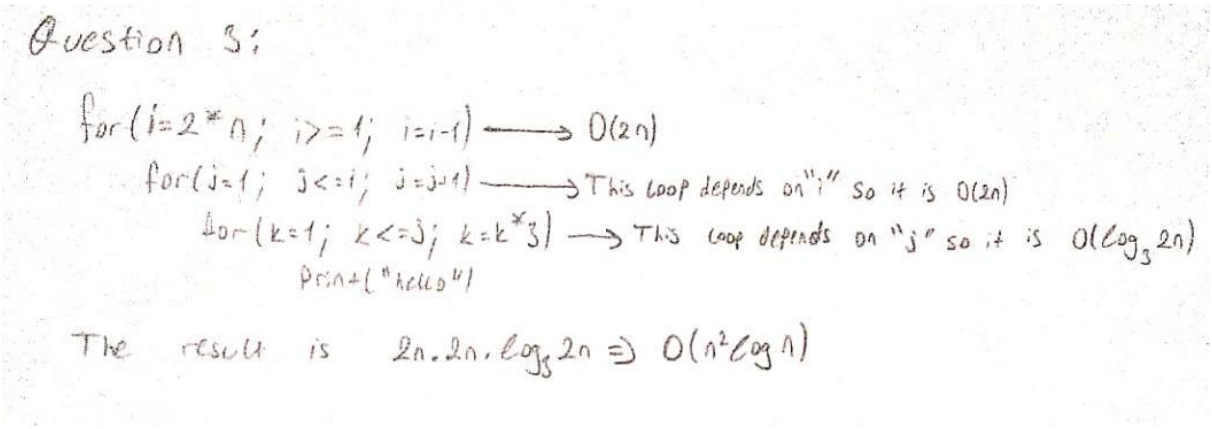
assign the endArr to endArr-1 (endArr = endArr-1)

else
assign the startArr to startArr +1 (startArr = startArr +1)

end of while loop

return false

Question 3:



Question 4:

```
float aFunc(myArray,n){
    if (n==1){                  //1
        return myArray[0];      //2
    }
     //let myArray1,myArray2,myArray3,myArray4 be predefined arrays
    for (i=0; i <= (n/2)-1; i++){   //1 before the loop; 2 or 3 per pass; 1 to exit loop
        for (j=0; j <= (n/2)-1; j++){   //1 before the loop; 2 or 3 per pass; 1 to exit loop
            myArray1[i] = myArray[i];   //3
            myArray2[i] = myArray[i+j];   //3 or 4
            myArray3[i] = myArray[n/2+j];   //5
            myArray4[i] = myArray[j];   //3
        }
    }
    x1 = aFunc(myArray1,n/2);   //T(n/2)
    x2 = aFunc(myArray2,n/2);   //T(n/2)
    x3 = aFunc(myArray3,n/2);   //T(n/2)
    x4 = aFunc(myArray4,n/2);   //T(n/2)

    return x1*x2*x3*x4;   //1
}
```

$$T(N) = 1 + 2 + 1 + \underbrace{\sum_{i=1}^{(N/2)-1} \sum_{j=1}^{(N/2)-1} (3+4+5+3)}_{= ((N^2)/4) * 15} + 4T(N/2)$$

$T(N) = O(N^2) + 4T(N/2)$

with using master theorem :

$a=4$, $b=2$ and $c=2$ ⟹ $\log_b a = \log_2 4 = 2 = c$

Thus the result is $T(N) = \Theta(N^2 \log N)$