Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4

MELİH YABAŞ
161044072

Course Assistant: AYŞE ŞERBETÇİ TURAN

QUESTION 1

A- Algorithm:

I determine that size of myList is n.

```
public static ArrayList<Integer> maxElementGroup(ArrayList<Integer> myList)
  {
      ArrayList<Integer> myList2 = new ArrayList<Integer>();

      int count = 1, j = 0;
      for (int i = 0; i < myList.size()-1; i++) { // n times
          if (myList.get(i) < myList.get(i + 1)) {
              count++;
              if (i + 1 == myList.size() - 1) {
                  myList2.add(count);
                  myList2.add(i + 1);
                  j += 2;
                  count = 1;

              }
          } else {

              myList2.add(count);
              myList2.add(i);
              j += 2;

              count = 1;
          }
      }
      myList2.add(0);

      int max = myList2.get(0), finalC = 0;
      finalC = myList2.get(1);

      for (int i = 0; i < myList2.size()-2; i += 2) {//This is less than n/2
          if (myList2.get(i) > max) {
              max = myList2.get(i);
              finalC = myList2.get(i+1);
          }
      }
      j = 0;

      ArrayList<Integer> maxGroup = new ArrayList<Integer>();
      for (int i = finalC - max + 1; i < finalC + 1; i++) {
          maxGroup.add(myList.get(i));//Less than n
          j++;
      }

      return maxGroup;
  }
```

n+c   (c is constant.)

Time complexity is O(n)

B- Algorithm:

This is a recursive method. So, first we must write the recurrence relation.

```
C- public  static  ArrayList<Integer> maxElRecursive(ArrayList<Integer> my-
   List, ArrayList<Integer> myList2, ArrayList<Integer> myList3, int i, int
   c1,int c2,int dir)
   { /*
       myList is the given list.
       myList2 and myList3 are empty lists.
       i is index and first, it is 0.
       c1 and c2 are counters.
       dir is a direction parameter.
       */
       if(i==myList.size()-1) //The statement will be apply n-1 times later
       {
           if (c1 > c2)
               return myList2; //1
           else
               return myList3; //1
       }

       if(dir==1)
       {
           if (myList.get(i) < myList.get(i + 1)) {
               myList2.add(myList.get(i));
               return maxElRecursive(myList, myList2, myList3, ++i, ++c1,
   c2,1);
           }
           else
           {
               myList2.add(myList.get(i));
               return maxElRecursive(myList, myList2, myList3, ++i, ++c1,
   c2,2);
           }
       }
       else if(dir==2)
       {
           if (myList.get(i) < myList.get(i + 1))
           {
               myList3.add(myList.get(i));
               return maxElRecursive(myList, myList2, myList3, ++i, c1, ++c2,
   2);
           }
           else
           {
               myList3.add(myList.get(i));
               return maxElRecursive(myList, myList2, myList3, ++i, c1, ++c2,
   3);
           }

       }
       else if(dir==3)
       {
           if(c1>c2)
           {
               myList3.clear();
               c2=0;
               return maxElRecursive(myList, myList2, myList3, ++i, c1, ++c2,
   2);

           }
```

```
            else {
                myList2.clear();
                c1=0;
                return maxElRecursive(myList, myList2, myList3, ++i, c1, c2,
    1);
            }
        }
        else
            return myList;
    }
```

This method goes on till i equals to n. After that, it returns the list. So we can say that the recurrence is;

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

Proof by Master Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + n^d$$

For this theorem,

$$a \geq 1 \; and \; b > 1 \; must \; be \; true$$

İf $\mathrm{a\ is\ equals, smaller\ or\ bigger\ than\ b^d}$ we can write the time complexity of the recurrence.

On our recurrence; a is 1, b is 1 and d is 0.

$1 > 1^0$ Therefor, complexity must be $\theta\left(n^{\log_1 1}\right)$ but $\log_1 1$ is not a valid value!

We cannot apply the master theorem to this recurrence.

Proof by Induction

We can say that T(0) = 1 because the method only returns a list.

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

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

T(n) = T(n-3) + 3

T(n) = T(n-k) + k

İf n=k;

T(n) = T(0) + n = 1 + n

So, we can say that T(n) = 1+n

<u>Base Case:</u>

C= 2 and n=1, $n_0 > n$

n≤c(n+1)

1≤2(1+1), This is true.

Inductive Step:

k≤2(k+1), Assume it is true.

k≤2k+2

k-2≤2k

We must Show this equation is true on k = k+1

k+1≤2(k+1+1)

k+1≤2k+4

Conclusion

k-3≤2k   This is true. (If k-2≤2k, this is absolutely true.)

QUESTION 2

Algorithm:

.If we say the length of array is n

```
public static int[] sumTwoNum(int[] arr, int sum)
{
    int i=0, j=arr.length-1;//1
    int[] nums = new int[2];//1

    while (arr[i]+arr[j]!=sum) // n times
    {
        if(arr[i]+arr[j]<sum)
        {
            i++;//1
        }
        else
        {
            j--;//1
        }
    }
    nums[0] = arr[i];//1
    nums[1] = arr[j];//1

    return nums;
}
```

n+4

Time complexity: $\theta(n)$

QUESTION 3

Algorithm:

```
for (i=2*n; i>=1; i=i-1)  //2*n times turns
        for (j=1; j<=i; j=j+1) //n times turns
                for (k=1; k<=j; k=k*3) //log_3 n times turns
                        print("hello")
```

2*n*n*$\log_3 n$ = $2 * n^2 * \log_3 n$

Time Complexity is $\theta(n^2 \log n)$

QUESTION 4

Algorithm:

```
float aFunc(myArray,n){
    if (n==1){
        return myArray[0];
    }
     //let myArray1,myArray2,myArray3,myArray4 be predefined arrays
    for (i=0; i <= (n/2)-1; i++){
        for (j=0; j <= (n/2)-1; j++){
            myArray1[i] = myArray[i];
            myArray2[i] = myArray[i+j];
            myArray3[i] = myArray[n/2+j];
            myArray4[i] = myArray[j];
        }
    }
    x1 = aFunc(myArray1,n/2);
    x2 = aFunc(myArray2,n/2);
    x3 = aFunc(myArray3,n/2);
    x4 = aFunc(myArray4,n/2);

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

This is a recursive method. First, we must write its recurrence relation.

There are 2 two loops and they are nested loops. Each of them turns n/2 times. So we have $\frac{n^2}{4}$ and the method calls it self 4 times for each time.

There 2 return statement. Therefor, recurrence relation of this method is

$T(n) = 4 * T\left(\frac{n}{2}\right) + \frac{n^2}{4} + 2$

Using Master Theorem; $4 = 2^2$ $(a = b^d)$

Time Complexity of the method is $\underline{n^2 \log n}$