**Gebze Technical University**

**Computer Engineering**

**CSE 222 - 2019 Spring**

**HOMEWORK 4 REPORT**

**CANER KARAKAŞ**

**131044061**

1. **A**

Given a single linked list of integers, we want to find the maximum length sorted sublist in this list. Our aim write an iterative function.

**public static** LinkedList<Integer> maxSortedList(LinkedList<Integer> list){  
 LinkedList<Integer> result = **new** LinkedList<Integer>(); //🡺 O(1)  
 LinkedList<Integer> tempList = **new** LinkedList<Integer>();//🡺 O(1)  
 **int** i=0; //🡺 O(1)  
 **int** maxSize = 0; //🡺 O(1)  
 **int** tempMaxSize = 0; //🡺 O(1)  
 **while** (list.get(i+1) != **null** && list.get(i)<=list.get(i+1)){ //🡺 O(n)  
 result.add(list.get(i)); //🡺 O(1)  
 maxSize++; //🡺 O(1)  
 i++; //🡺 O(1)  
 }  
 result.add(list.get(i)); //🡺 O(1)  
 **if** (list.get(i+1) != **null**){ //🡺 O(1)  
 i++; //🡺 O(1)  
 **while**(i+1 != list.size()){ //🡺 O(n)  
 **if** (list.get(i) <= list.get(i+1)){ //🡺 O(1)  
 tempList.add(list.get(i)); //🡺 O(1)  
 tempMaxSize++; //🡺 O(1)  
 i++; //🡺 O(1)  
 }  
 **else**{  
 tempList.add(list.get(i)); //🡺 O(1)  
 tempMaxSize++; //🡺 O(1)  
 **if**(maxSize <= tempMaxSize){ //🡺 O(1)  
 result = tempList; //🡺 O(1)  
 maxSize = tempMaxSize; //🡺 O(1)  
 tempMaxSize = 0; //🡺 O(1)  
 tempList = **null**; //🡺 O(1)  
 tempList = **new** LinkedList<Integer>(); //🡺 O(1)  
 i++; //🡺 O(1)  
 }  
 **else**{  
 tempList = **null**; //🡺 O(1)  
 tempList = **new** LinkedList<Integer>(); //🡺 O(1)  
 tempMaxSize = 0; //🡺 O(1)  
 i++; //🡺 O(1)  
 }  
 }  
 }  
 **if** (list.get(i-1)<= list.get(i)){ //🡺 O(1)  
 tempList.add(list.get(i)); //🡺 O(1)  
 tempMaxSize++; //🡺 O(1)  
 **if** (maxSize <= tempMaxSize){ //🡺 O(1)  
 result = tempList; //🡺 O(1)  
 maxSize = tempMaxSize; //🡺 O(1)  
 }  
 }  
 }  
 **else  
 return** result; //🡺 O(1)  
  
 **return** result; //🡺 O(1)  
}

Time Complexity 🡺 O(n)

1. **B**

Write a recursive function for the same purpose.

**public static** LinkedList<Integer> maxSortedListRecursion(LinkedList<Integer> list, LinkedList<Integer> maxList){  
 **if**(list.isEmpty()) //🡺 O(1)  
 **return** maxList; //🡺 O(1)  
 **if** (maxList.isEmpty()){ //🡺 O(1)  
 maxList.add(list.pop()); //🡺 O(1)  
 **while**(maxList.peekLast()<list.peek()) //🡺 O(n)  
 maxList.add(list.pop()); //🡺 O(1)  
 }  
 **else**{  
 LinkedList<Integer> temp = **new** LinkedList<Integer>(); //🡺 O(1)  
 temp.add(list.pop()); //🡺 O(1)  
 **while** (!list.isEmpty() && temp.peekLast() < list.peek()) //🡺 O(n)  
 temp.add(list.pop()); //🡺 O(1)  
 **if** (temp.size()>=maxList.size()) //🡺 O(1)  
 maxList = temp; //🡺 O(1)  
 }  
 **return** maxSortedListRecursion(list,maxList);  
}

Time Complexity 🡺 T(n) = O(1) + T(n-1)

=1 + T(n-1)

=2 + T(n-2)

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

**public static int**[] search(**int** array[], **int** first, **int** second, **int** number, **boolean** controlResult){  
 **if** (first + second == number && controlResult == **true**){ //🡺 O(1)  
 **int** [] numbers = **new int**[2]; //🡺 O(1)  
 numbers[0] = first; //🡺 O(1)  
 numbers[1] = second; //🡺 O(1)  
 **return** numbers; //🡺 O(1)  
 }  
  
 **if** (first==-1){ //🡺 O(1)  
 **int** [] numbers = **new int**[2]; //🡺 O(1)  
 numbers[0] = -1; //🡺 O(1)  
 numbers[1] = -1; //🡺 O(1)  
 **return** numbers; //🡺 O(1)  
 }  
  
 **if** (first==0 && second==0){ //🡺 O(1)  
 **if** (number%2==0){ //🡺 O(1)  
 first = number/2; //🡺 O(1)  
 second = number/2; //🡺 O(1)  
 }  
 **else**{  
 **int** temp = (number-1)/2; //🡺 O(1)  
 first = temp; //🡺 O(1)  
 second = temp+1; //🡺 O(1)  
 }  
 }  
 **else** {  
 **int** i=0; //🡺 O(1)  
 **boolean** control = **true**; //🡺 O(1)  
 **while** (control==**true** && i!=array.length){ //🡺 O(n)  
 **if** (array[i] == first){ //🡺 O(1)  
 control=**false**; //🡺 O(1)  
 }  
 **else** i++;//🡺 O(1)  
 }  
 **boolean** control2 = **true**; //🡺 O(1)  
 **if**(control==**false**){ //🡺 O(1)i++;  
 **while** (control2==**true** && i!=array.length){ //🡺 O(n)  
 **if** (array[i] == second) //🡺 O(1)  
 control2=**false**; //🡺 O(1)  
 **else** i++;//🡺 O(1)  
 }  
 }  
 **if** (control2==**false** && control==**false**){ //🡺 O(1)  
 **return** search(array,first,second,number, **true**);  
 }  
 }  
 **return** search(array,first-1,second+1,number, **false**);  
}

If the numbers are not found, time complexity is O(m/2). M = number.

Time Complexity 🡺 Θ (n)

**3-**

for (i=2\*n; i>=1; i=i-1) 🡺 O(2n)

for (j=1; j<=i; j=j+1) 🡺 O(2n)

for (k=1; k<=j; k=k\*3) 🡺 O(log3(2n+1))

print(“hello”)

Time Complexity 🡺 O(n2logn)

**4-**

