Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4

Yusuf Can Kan
161044007

Course Assistant: Ayşe Şerbetçi Turan

1 Question 1

a)

Pseudocode:

```
public LinkedList Iterative(LL HeadOfTheList){
        while( Node is not null and Next node is not null){
            if ( current data smaller or equal to next data ) {
                Go next node.
                Increase counter.
            }
            else if( current data is bigger than the next data ){
                if the last list has the maximum length{
                    Store the head of last list.
                    Set tail of the last list is null.
                    Store the length of the element which has last list.(Counter).
                }
                Reset the counter.
                Go to next element.
            }
        }
        return answer;
    }
```

Java Code:

```
public LL Iterative(LL head){
    LL answer=null;
    LL temp=head;
    LL node = head;

    int answer_count=0; /*Holds the longest sub list count.*/
    int temp_count = 0; /*Holds the current sub list counnt.*/

    while(node !=null && node.next != null){
        if(node.data<=node.next.data){
            node = node.next;
            temp_count++;
        }
        else if(node.data > node.next.data){
            if(temp_count > answer_count){
                answer=temp;
                answer_count=temp_count;
            }
            temp_count=0;
            LL tempp = node;
            temp = node.next;
            node=node.next;
            tempp.next=null; /*Sets last list tail is null.*/
        }
    }
    return answer;
}
```

Analyze:

Inside the method it has one while loop which has O(n) complexity which n is the number of element which linked list has. Besides that inside the while loop it has if,else if comparisons and they all has O(1) complexity. So the final complexity is O(n).

b)

Pseudocode:

```
LinkedList Recursive(head of longest sub linked list ,
                     head of current sub linked list ,
                     current element, current count, longest count){
    if(current element or next element is null){
      return head of longest sub linked list;
    }
    if(current data is smaller than next data){
      if(current count is bigger than the longest count){
        set head of the longest sub list to current list head;
      }
      return Recursive(head of longest sub linked list ,
                     head of current sub linked list ,
                     current element, current count+1, longest count);
    }
    else if(current data is bigger than the next data){
      if(current count is bigger than or equal to the longest count){
        set head of the longest sub list to current list head;
        set tail of the list null;
        return recursive (head of the longest sub list, next node, next node,
                     0,current count);
      }
      return recursive (head of the longest sub list, next node, next node, 0,
     longest count);
    }
    return head;
  }
```

Java Code:

```
  public LL foo(LL head,LL head_temp,LL node,int currentCount,int longestCount){
    if(node == null || node.next == null){
      return head;
    }
    if(node.data <= node.next.data){
      if(currentCount>longestCount){
          head = head_temp;
      }
      return foo(head,head_temp,node.next,currentCount+1,longestCount);
    }
    else if(node.data> node.next.data){
      if(currentCount>=longestCount){
          head=head_temp;
          LL temp = node.next;
          node.next=null;
          return foo(head,temp,temp,0,currentCount);
      }
      return foo(head,node.next,node.next,0,longestCount);
    }
    return head;
  }
```

Analyze:

When we examine the code, first if statement is ours base case, its time complexity is 1. The second if statement has one recursive call and that recursive call goes 1 forward inside the list. So it is T(n) = T(n-1) which n is the number of element inside the linked list.

The else if statement has another if statement which has another recursive call. Since the both recursive calls goes one forward inside the linked list and they both returns the result, we have T(n-1) for both calls. So the sum of the situation we have T(n) = T(n-1).
So when we combine all of them since the recursive calls doesn't affect each other and they all does the same operations (going one forward inside the linked list) we can say all statement is T(n) = T(n-1) + c.(n is the number of element which linked list has.)

T(n)= {

T(n) = T(n-1) if n>1,

T(n) = 1 if n = 1.

}

Master Theorem

We can’t proof the statement by master theorem because if we examine the statements (a=1,b=1,d=0), since d is 0, we cannot reach any solutions.

Induction

For n=2;

T(2) = T(1)+1

T(2) = 1+1 = 2. Statement is correct.

For n=x;

T(x) = T(x-1) + 1

T(x-1) = T(x-2) + 1

T(x-2) = T(x-1) + 1

.

.

.

T(2) = T(1) + 1 = 2

T(1) = 1.

When we examine the statement we can see that with evet step it adds result to 1.

((((((1+1)+1)+1)+1)+1)……….+1). It goes x times. So the result is x. So it is true.
Since result is x and true, statement T(n) = T(n-1) has O(n) time complexity.

2 Question 2

Since array is sorted we can start looking numbers from beginning and at the end of the array. We can point first and last element. If these two elements sum is smaller than what we search, we iterate the left point one forward. If the sum is smaller than what we search we can iterate end point one backward. So when we find the the right numbers we can stop searching.

Since it depends on one loop and loop depends on number of elements the time complexity is ϴ(n) which n is the number of element which the array has.

The function pseudocode is;

```
void foo(array,sum){
    int left=0,right=array.length-1;
    while(array[left] + array[sum] != sum){
        if(array[left] + array[right] < sum) left++;
        else (array[left] + array[right] > sum) right--;
    }
    print( Index1: left, Index2 : right);
}
```

3 Question 3

First when we examine the outer loop;

```
    for (i=2*n; i>=1; i=i-1)
```

Its best case and worst case complexity is n. When we combine it with second loop;

```
    for (i=2*n; i>=1; i=i-1)
        for (j=1; j<=i; j=j+1)
```

The inner loop turns count depends the outer loop. So we can think that loop like that;

```
1
1 2
1 2 3
1 2 3 4
.
.
1 2 3 4 5 ….. n
```

So we can obtain 1+2+3+4+….+n. It can express as (n+(n+1))/2 = (n^2)/2 + n/2 and when we take theta of that expression we can obtain ϴ(n^2). But we have one more inner loop so we have to start with that.

When we examine the inner loop;

```
    for (k=1; k<=j; k=k*3)
```

It increases 3 times with every iteration and it depends on j and for that reason it does the same calcutaions 2 times. So it does the n/2 times same thing and with that we can obtain 2*(n/2)* log(3n) = ϴ(n * log(n)). And with the outer loop we have ϴ(n).

So finally we can calculate the total complexity with combining 3 loops.
ϴ(n) * ϴ(n * log(n)) = ϴ(n^2 * log(n)).

4 Question 4

Inside the function we have nested two loop and they both depends n/2. Inside the loop we have O(1) complexity operations. So in here we can obtain (n^2) / 4 in here.

After the loop we have 4 recursive call and they both calls dividin n to 2. So in here we have 4*T(n/2).
The others all have 1 complexity.

So as a result the answer is, T(n) = 4*T(n/2) + (n^2)/4
If we examine the statement with master theorem (a=4,b=2,d=2), since a and b^d are equals to each other, we can obtain ϴ(n^d * log(n)) so the complexity is ϴ((n^2)* log(n)).