Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

FATİH DURAL
151044041

Course Assistant: Ayşe Şerbetçi Turan

PART1

Given a single linked list of integers, we want to find the maximum length sorted sublist in this list. For example for the list L= {1, 9, 2, 7, 20, 13} the returned list should be S = {2, 7, 20}.

a)

Write an iterative function which performs this task. Analyze its complexity.

```
public LinkedList maxSubList(LinkedList<Integer> list){
    int count = 1, max = 1, index_first = 0, index_last = 0;
    LinkedList sublist = new LinkedList<Integer>();
    for( int i = 1; i < list.size(); i++){
        if( list.get(i) >= list.get(i-1) ) {
            count++;
        }
        else{
            count = 1;
        }
        if( count > max){
            index_last = i+1;
            max = count;
        }
    }
    Iterator iter =  list.listIterator(index_last - max);
    count = 0;
    while(iter.hasNext()){
        if( count == max ){
            break;
        }
        count++;
        sublist.add(iter.next());
    }
    return sublist;
}
```

Function takes a LinkedList and return the maximum lenght sorted sublist. If the element is larger than previous, count rises. If the count larger than maximum element, it update and largest element found. Return the sublist using iterator. Time complexity become O(n) because of wanders of all elements.

b)

Write a recursive function for the same purpose. Analyze its complexity by using both the Master theorem and induction.

```
public maxSubList(Node temp, return, current, max, size)
```

```
{
    if(temp.next == null)
    {
        if(currentSize > sizeMax)
        {
            max = size;
            return temp = current;
        }
        return copyList(temp, return, current, max, size);
    }
    else
    {
        if(temp.data <= temp.next.data)
        {
            size++;
            temp = temp.next;
            return maxSubList(temp, return, current, max, size);
        }
        else
        {
            if(size > max)
            {
                max = size;
                temp = current;
            }
            current = temp.next;
            temp = temp.next;
            return MaxSubList(temp, return, current, max, size);
        }
    }
}
```

In order to apply the master method, an equation is required first.

$$T(n) = aT(n / b) + f(n)$$

An important warning here is that the master method cannot be applied to every equation. It can only be applied to the equations above. So we can not analyze its complexity by using both the Master theorem. Addionality,

Complexity by using induction:

$$T(n) = \begin{cases} \text{Theta(m), if } (n == 1) \\ \text{A + 2*T(n - 1), if } (n != 1) \end{cases}$$

So

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

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

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

...

T(n) = k*2 + T(n-k) k= n-1 =>

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

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

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

T(n) = θ (n)

PART2

Describe and analyze a Θ (n) time algorithm that given a sorted array searches two numbers in the array whose sum is exactly x.

```
// Prints the pair with sum cloest to x
static void printClosest(int arr[], int n, int x)
{
    int res_l = 0, res_r = 0;  // To store indexes of result pair

    // Initialize left and right indexes and difference between
    // pair sum and x
    int l = 0, r = n-1, diff = Integer.MAX_VALUE;

    // While there are elements between l and r
    while (r > l)
    {
        // Check if this pair is closer than the closest pair so far
        if (Math.abs(arr[l] + arr[r] - x) < diff)
        {
            res_l = l;
            res_r = r;
            diff = Math.abs(arr[l] + arr[r] - x);
        }

        // If this pair has more sum, move to smaller values.
        if (arr[l] + arr[r] > x)
            r--;
        else // Move to larger values
            l++;
    }
```

PART3

Calculate the running time of the code snippet below

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

```
for (j=1; j<=i; j=j+1)
    for (k=1; k<=j; k=k*3)
        print("hello")
```

At the top of loop turn 2n times. Underneath, turn n times and at the bottom of loop turn log3n times. So, 2n * n * log3n = n^2(log3n)

PART4

Write a recurrence relation for the following function and analyze its time complexity T(n).
T(n) = 4*T(n/2) + n^2
4 means that there are 4 recursive call
n^2 means that the for loops work n^2 times
a = 4, b = 2, d = 2, a = b^d.

- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break