Rumeysa KARAKAVAK
141044063

GTU Department of Computer Engineering

CSE 222/505 - Spring 2019

HW 04

1) Assuming Node class was written. ITERATIVE

```
List<int>    findSubList ( Node *head){
    List<int> returningList = new List<int>();
    int   temp =1;
    int   maximum=1;
    int   count=1;
    int   index = 0;
    for( Node  *walkerNode = head ; walkerNode -> next != NULL ;  walkerNode = walkerNode -> next){
        if ( walkerNode -> data < walkerNode -> next -> data)     } const
            ++temp;
        else{
            if( maximum < temp ){
                maximum = temp;
                index = count - temp;
            }                                       } const
            temp = 1;
        }
        ++count;
    }                                                          } list size = n
    if( temp > maximum){
        maximum = temp;
        index = count - maximum;          } const
    }
    // We finally found maximum list size and its begining index.
    int i=0; int j=0;
    for( Node * myNode = head ;  myNode != null ;  myNode = myNode -> next){
        if( i == index){
            while( j < maximum){
                returningList.push( myNode->data);
                myNode = myNode -> next;        } 0 to index = n
                ++j
            }
        }                                                } 0 to list size = n      } n^2
        ++i;
    }
    return returningList;
}
```

$$\underbrace{T(n) = n + n^2}_{max} \Rightarrow \boxed{T(n) = n^2}$$

1)b

```
int    findSublist (int head, List<Integer> tail, int maximum){

    if (tail == null)
       return 0;
    if ( tail.size()== 0)
       return maximum;

    if ( head <= tail.get(0) ){
          ++maximum;
    }

    else {
        if ( tail.sublist(0, tail.size()-1 ). size()== maximum ){
        System.Out.Println( tail.sublist(0, tail.size()-1));
        }
        maximum = 1;

    }

    return findSublist ( tail.get(0) , tail.subList( 1, tail.size() ), maximum );
```

returning yourself while tail is not null, and it returns amount of size of list.

Its time complexity is $O(n)$

its size (list)

2)

```
int[][]   find Numbers ( int array[], int size, int X ){

    int i=0;
    int[][] returningArray = new int[n][2];
    int j=0;
    while ( i < size-1){
       int k = 0;
       while ( k < size-1){
       { if (array[i] + array[k] == X){

              returningArray [j][0] = array[i];
              returningArray [j][1] = array [k];
              ++j;
         }
       } ++k;
     ++i;
    }

    return  returningArray;

}
```

const. time

while loop bounds go to 0 to arraysize $\underline{n}$

while loop bounds go to 0 to arraysize $= \underline{n}$

// if there are pairs that sum of them is equal more than 1, we must store them into a 2D array. In this case ⇒ returningArray [0][0] → first number + returningArray [0][1] → second number } X   returningArray [1][0] → first + returningArray [1][1] → second } X

There is 2 while loop in code and they go to the $\underline{n}$, another statements are take constant time. So time complexity of this code is $n * n = n^2$ $\Rightarrow \underline{\underline{T(n) = O(n^2)}}$

3)

```
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"); → 1
} } }
```

$\log n$, n , n $= \underline{n^2 \log n}$

3. inner loop bounds changing with multiplication. So its time complexity is $\underline{\log_3 n}$

2. inner loop bounds changing with single increment. and it goes to unstable value $(\underline{n})$

Outer loop bounds changing with single increment to. And it goes to nonconstant value (n)

$$\underline{T(n) = O(n * n * \log n) \Rightarrow (n^2 \log n)}$$

4)

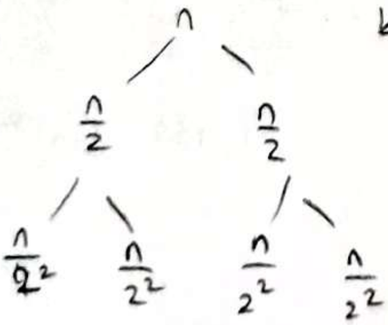
```
float aFunc (myArray, n){

    if (n == 1)
        return myArray[0];

    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);  → 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)
}
```

$\rightarrow \left(\frac{n}{2}\right) - 1$ $\quad \left(\frac{n}{2}\right) - 1$ $\quad \frac{n^2}{4} - n + 1$

$\Rightarrow T(n^2)$

$4T(n/2)$

$$T(n) = \begin{cases} c1 & n = 1 \\ T(n) & T(n^2) + 4T(n/2) \end{cases}$$



$\Rightarrow$ Assume $\frac{n}{2^k} = 1$

$n = 2^k$ $\quad k = \log n$

$\rightarrow$ So $\underbrace{T(n/2)}_{\log n} + \underbrace{T(n^2)}_{n^2}$

$\log n \cdot n^2$

$T(n) = O(n^2 \log n)$