Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 4 REPORT

NURULLAH GÜNDOĞDU
161044108

Ayşe Şerbetçi TURAN

1 PART1

1.1 a)

```
LinkedList sortedSublist(LinkedList<Integer> list) {

    Iterator<Integer> iterator = list.iterator();
    Iterator<Integer> iterator3 = list.iterator();
    Iterator<Integer> iterator2 = list.iterator();
    LinkedList<Integer> newList=new LinkedList<>();

    int sortedListLength = 1, tempLength = 1;
    int listLength = 1, startPoint = 0 ,i=0;

    iterator3.next();

    while (iterator3.hasNext()){
        if(iterator.next() < iterator3.next())
            sortedListLength++;
        else {                mkfifo(myfifo, 0666);
            if (tempLength < sortedListLength) {
                tempLength = sortedListLength;
                startPoint = listLength - sortedListLength;
            }
            sortedListLength = 1;
        }
        listLength++;
    }

    if (tempLength < sortedListLength) {
        tempLength = sortedListLength;
        startPoint = listLength - tempLength;
    }

    while (iterator2.hasNext()) {
        if (i == startPoint) {
            while (tempLength > 0) {
                newList.add(iterator2.next());
                tempLength--;
            }
            break;
        }
        iterator2.next();
        i++;
    }

    return newList;

}
```

There are 10 assignments, one arithmetic operations, one if condition and one return in function. In first loop turns n time. There are 3 assignments and 3 arithmetic operations and 2 if conditions in the loop. In second loop there are one loop and one arithmetic operation. In other loop there are 2 arithmetic operations. Second loop and inner loop turn n time in total.

In function;

10 + 1 + 1 + 1
+ First loop = n * (3 + 3 + 2)
+ Second loop = n * (1 + 2)

$T(n) = 13 + 8n + 3n$
$= 11n + 13$

$T(n) = O(11n + 13) \Rightarrow O(n)$

1.2 b)

```
LinkedList sortedSublistRecursively(LinkedList<Integer> list){
    LinkedList<Integer> newList = new LinkedList<>();

    int sortedListLength = 1, tempLength = 1;
    int listLength = 1, startPoint = 0;

    int arr[]=firstRecursive(list, sortedListLength, tempLength, listLength, startPoint, 0);

    if (arr[3] < arr[1]) {
        arr[3] = arr[1];
        arr[0] = arr[2] - arr[3];
    }
    newList=secondRecursive(list,newList,arr[3],arr[0],0,arr[0]);

    return newList;
}

public LinkedList secondRecursive(LinkedList<Integer> list,LinkedList<Integer> newList,int tempLength, int
startPoint,int start, int index){

    if (start == startPoint) {
        newList.add(list.get(index));
        index++;
        tempLength--;
    }
    if(start!=startPoint)
        start++;

    if(tempLength>0)
        newList=secondRecursive(list,newList,tempLength,startPoint,start,index);

    return newList;
}

public int[] firstRecursive(LinkedList<Integer> list,int sortedListLength, int tempLength,int listLength, int
startPoint, int index){

    int arr[]=new int[4];
    if(list.get(index)<list.get(index+1))
        sortedListLength++;
    else {
        if (tempLength < sortedListLength) {
            tempLength = sortedListLength;
            startPoint = listLength - sortedListLength;
        }
        sortedListLength = 1;
    }
    listLength++;
    index++;

    arr[0]=startPoint;
    arr[1]=sortedListLength;
    arr[2]=listLength;
    arr[3]=tempLength;

    if(listLength<list.size() && index<list.size())
        arr = firstRecursive(list, sortedListLength, tempLength, listLength, startPoint, index);
        return arr;
}
```

2 PART2

```
boolean findSum(int arr[], int sum){
    int i=0;
    int j=arr.length - 1;

    while (i <= j){
        if (arr[i] + arr[j] == sum)
            return true;
        if (arr[i] + arr[j] < sum)
            i += 1;
        else
            j-=1;
    }
    return false;
}
```

The list is an ordered list, which makes our job easier. First we make a cycle. In the loop, we get elements from the beginning and end of the list. We collect these elements and if the collection is equal we return true and say that we have found the numbers. If they are not equal, if the sum of the two numbers is greater than the sum, we make the last value of the last element of the list. If it is smaller than the total, we make the next element of the list from the beginning. This is how our cycle is progressing. If we cannot find the equation that is the collection, it returns false. And that sorting takes time Θ(nlogn).

3 PART3

```
for (i=2*n; i>=1; i=i-1)            first and second loop   (2n*(2n+1))/2
    for (j=1; j<=i; j=j+1)
            for (k=1; k<=j; k=k*3)      -> k == log(2n)
                    print("hello")      -> 1
```

Time complexity;

$1 * (2n*(2n+1))/2 * \log(2n) = (4n^2 + 2n) / 2 * \log(2n)$

$= (2n^2 + n) * \log(2n)$

$= \mathbf{n^2 * \log(2n)}$

Asymptotic Notations -> $O(n^2 * \log(n))$

4 PART4

```
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;
}
```

T(n) = a T(n/b) + f(n)

Since there are 4 recursive calls, the ‘a’ is equal to 4.

a = 4

The n parameter of the functions is given in 2 divided by each time. Therefore n / b is equal to n / 2.

n / b = n / 2

Since the loops in the function work in every function call, loop turns (n^2 * n^2) times. There are 4 assignments and 3 arithmetic operations in the loop and 4 arithmetic operations in return.

So; f (n) is equal to (n/2 * n/’2) * 7 + 4.

f(n) = (n/2 * n/’2) * 7 + 4
= n^2 / 4 * 7 + 4
= 7n^2 / 4 + 4
= n^2

d = 2

T(n) = 4T(n/2) + n^2

a = b^d -> 4 = 2^2 = 4 so; T(n) = Θ(n^2 * log(n))