Gebze Technical University

Computer Engineering

CSE 222 - 2021

HOMEWORK 02 REPORT

Huseyin Omer GURAY

1901042696

Part 1:

I. Searching a product.

```java
/**
 * prints all branch items
 */
public void exploreAllBranches(){
    for(int i  = 0; i < company.getAllBranchs().length;i++){
        System.out.println(company.getAllBranchs()[i]);
    }
}
```

```java
/**
 *
 * @return String version of Branch
 */
@Override
public String toString() {
    String res = "Branchcode :"+branchCode+"\n";
    for(int i = 0;i< item.length;i++)
        res+= item[i];
    return res;
}
```

Searching al branches and in branches all item so ; branches length = m , items length = m

Time complexity : o(mn).

II. Add/remove product.

```java
public void openStock(int branchCode ,int itemNum, int model , int color,int amount){
    int index = -1;
    for(int i = 0;i<company.getAllBranchs().length;i++){
        if(branchCode == company.getAllBranchs()[i].getBranchCode()){
            index = i;
            break;
        }
    }
    if(index == -1){
        System.out.println("You enterend wrong information pls try again");
        return;
    }
    int[][] stock = company.getAllBranchs()[index].getItem()[itemNum].getStock();
    stock[model][color] += amount+3;
    company.getAllBranchs()[index].getItem()[itemNum].setStock(stock);
    System.out.println("Stock added to system!");
}
```

All information taking as a paremeter , and searching for a correct branch, so in worst case if the branches length = n

Time complexity : o(n)

III. Querying the products that need to be supplied

```
/**
 *
 * @param itemNum item number
 * @param modelNum model number
 * @param colorNum color number
 * @param amount amount of item
 */
public void callAdmin(int itemNum,int modelNum , int colorNum,int amount){
    System.out.println("oww sorry we are out of stock I have to call my admin pls wait a while");
    company.getAdmin().openStock(workBranch.getBranchCode(),itemNum, modelNum, colorNum, amount);
}
```

This function just calling the admin for the inform him/her so ,

Time complexity : o(1)

Part 2:

A) Explain why it is meaningless to say: “The running time of algorithm A is at least O(n2)”.

O(n^2) means that algorithm will be O(n^2) at worst case. ‘at least’ means that the algorithm will work O(n^2) at best case .this is not correct. O(n^2) is maximum time for the algorithm. It can be less than O(n^2) at some points. So the word is meanless.

B) Let f(n) and g(n) be non-decreasing and non-negative functions. Prove or disprove that: max(f (n), g(n)) = Θ(f(n) + g(n)).

$$\max(f(n), g(n)) = \Theta(f(n) + g(n))$$

time to prove

$$O(f(n)+g(n)) = \max(f(n), g(n))$$

$$\underbrace{\max(f(n), g(n))}_{h(n)} \leq 1 \cdot \underbrace{(f(n)+g(n))}_{l(n)}$$

$$h(n) = O(l(n))$$ proved

$$\Omega(f(n)+g(n)) = \max(f(n), g(n))$$

$$\begin{aligned} &\max(f(n), g(n)) \geq f(n) \\ + \; &\max(f(n), g(n)) \geq g(n) \\ \hline &2\max(f(n), g(n)) \geq f(n) + g(n) \end{aligned}$$

OR

$$\Rightarrow \underbrace{\max(f(n), g(n))}_{h(n)} \geq \underbrace{\frac{1}{2}(f(n)+g(n))}_{l(n)}$$

$$h(n) = \Omega(l(n))$$ ✓

C) Are the following true? Prove your answer.

1- $\lim\limits_{n \to \infty} \frac{2^{n-1}}{2^n} \Rightarrow \frac{2^n \cdot 2}{2^n} = \lim\limits_{n \to \infty} 2 = \underline{\Omega}$

this is Prove of they are $\underline{\text{equal}}$

2- c is constant

$2^{2n} \leq c\,2^{2n}$

$\ln 2 . 2n \leq \ln c + \ln 2 . n$

$2n \leq \ln c + n$

$n \leq \ln c$

This is wrong because we cant find that const number (because there is no!) in equality $\underline{R = \{\ \}}$

3-)

$\underline{f(n) = O(n^2)}$

Worstcase
O Notation

$\underline{g(n) = \Theta(n^2)}$

dont have certain information.

if we multiply with $O \times \Theta$ result should be O. T(n) should be multiplybynormally.

$f(n) \times g(n)\ \ != \Theta(n^4)$ Wrong

$f(n) \times g(n) = O(n^2 . n^2) = O(n^4)$ true

Part 3:

3^n > n.2^n> 2^n+1 = 2n> 5^log2 n> n^1.01 > nlog^2n >√n > (log n)3 > logn

$n^{1.01}$

1

- $\lim\limits_{n\to\infty} \frac{n^{1.01}}{n\log^2 n} = \frac{n^{0.01}}{\log^2 n} = \frac{n^{-0.99} \times 0.01}{2\log n \cdot \frac{1}{\ln(2)\, n}} = \frac{n^{-0.99}}{\frac{\log n}{n}} = \frac{(n^{0.01})'}{(\log n)'}$

$= \frac{n^{-0.99}}{\frac{1}{n}} = n^{0.01} = \boxed{\infty} \Rightarrow n^{1.01} > n\log^2 n$

- $\lim\limits_{n\to\infty} \frac{n^{1.01}}{2^n} = \frac{(n^{1.01})'}{(2^n)'} = \frac{1.01 \times n^{0.01}}{2^n \cdot \ln 2} = \frac{n^{\frac{0.01}{0.99}}}{2^n} = \frac{1}{2^n \cdot n^{0.99}} = \frac{1}{\infty} = 0$

$n^{1.01} < 2^n$

- $\lim\limits_{n\to\infty} \frac{n^{1.01}}{\sqrt{n}} = \frac{n^{1.01}}{n^{0.5}} = n^{0.51} = \infty$ $\quad n^{1.01} > \sqrt{n}$

- $\lim\limits_{n\to\infty} \frac{\log^3(n)}{n\log^2(n)} = \frac{\log(n)}{n} = \frac{\frac{1}{n}}{1} = \frac{1}{n} = 0$

$n\log^2(n) > \log^3(n) \Rightarrow n^{1.01} > \log^3(n)$

- $\lim\limits_{n\to\infty} \frac{n^{1.01}}{n \cdot 2^n} = \frac{n^{0.01}}{2^n} = \frac{n^{\frac{0.01}{0.99}}}{2^n \cdot \ln 2} = \frac{1}{2^n \cdot n^{0.99}} = \frac{1}{\infty} = 0$

$n^{1.01} < n \cdot 2^n$

- $\lim\limits_{n\to\infty} \frac{2^n}{3^n} = \left(\frac{2}{3}\right)^n = 0$ $\quad 2^n < 3^n$ (we dont need check but I did.)

$\Rightarrow n^{1.01} < 3^n$

$\lim\limits_{n\to\infty} \frac{2^n}{2^{n+1}} \quad \frac{1}{2}\left(\frac{2^n}{2^n}\right) \Rightarrow \frac{1}{2} \Rightarrow 2^n = 2^{n+1}$

and also $n^{1.01} < 2^{n+1}$

2

$$-\lim_{n\to\infty} \left(\frac{n^{1.01}}{\log n}\right) = \frac{1.01 \cdot n^{0.01}}{\frac{1}{n}} = 1.01 n^{1.01} = \infty$$

$$\underline{n^{1.01} > \log n}$$

mid res: $n \log^2 n$, $\sqrt{n}$, $\log^3 n$, $\log n$, $\underset{\text{best space}}{n^{1.01}}$, 2^n, 2^{n+1}, $n \cdot 2^n$, 3^n, $5^{\log_3 n}$

$$-\lim_{n\to\infty} \frac{n\log^2 n}{\sqrt{n}} = \sqrt{n}\log^2 n = \infty \quad \underline{n\log^2 n > \sqrt{n}}$$

$$\underline{\Rightarrow n\log^2 n > \log^3 n}$$

$$-\lim_{n\to\infty} \frac{n\log^2 n}{\log n} = n\log n = \infty \quad \underline{n\log^2 n > \log n}$$

now: $\sqrt{n}$, $\log^3 n$, $\log n$, $n\log^2 n$, $n^{1.01}$ - - - - - - -

$$-\lim_{n\to\infty} \frac{\log^3 n}{\log n} = \log^2 n = \infty \Rightarrow \underline{\log^3 n > \log n}$$

$$-\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \frac{2\sqrt{n}}{n} = \frac{2\sqrt{n}}{\sqrt{n}\sqrt{n}} = \frac{2}{\sqrt{n}} = \underline{0}$$

$$\underline{\log n < \sqrt{n}}$$

$$-\lim_{n\to\infty} \frac{\log^3 n}{\sqrt{n}} = \frac{\frac{3\ln^2 n}{n}}{1/2\sqrt{n}} \quad \frac{6\ln^2 n}{\sqrt{n}} = \frac{12\ln(n)/n}{1/2\sqrt{n}} = \frac{24\ln n}{\sqrt{n}}$$

$$= \frac{24/n}{1/2\sqrt{n}} = 48/\sqrt{n} = 0 \Rightarrow \log^3 n < \sqrt{n}$$

3

$$-\lim_{n\to\infty} \frac{n\cdot 2^n}{2^n} = n = \infty \Rightarrow \underline{n\cdot 2^n > 2^n}$$

$$-\lim_{n\to\infty} \frac{n\cdot 2^n}{3^n} = n\left(\frac{2}{3}\right)^n = \frac{n/1}{(2/3)^n} = \frac{1}{\left(\frac{3}{2}\right)^n \ln\left(\frac{3}{2}\right)} =$$

$$= \frac{2^n}{\ln\left(\frac{3}{2}\right)\cdot 3^n} = \frac{2^n \ln(2)}{\ln\left(\frac{3}{2}\right)\cdot 3^n \cdot \ln(3)} = \frac{\ln^2(2)\cdot 2^n}{\ln^2(3)\cdot \ln\left(\frac{3}{2}\right) 3^n}$$

$$= \frac{\ln^4(2)\cdot 2^n}{\ln^4(3)\ln\left(\frac{3}{2}\right)\cdot 3^n} = \frac{\ln^6(2)\cdot 2^n}{\ln^6(3)\ln\left(\frac{3}{2}\right)\cdot 3^n} = \frac{\ln^7(2)\, 2^n}{\ln^6(3)\ln\left(\frac{3}{2}\right)\cdot 3^n} = \frac{\ln^7(2)\cdot 2^n}{\ln^7(3)\cdot \ln\left(\frac{3}{2}\right) 3^n}$$

$$= \frac{2^n}{3^n} = \frac{2}{3}^n = 0 \Rightarrow 3^n > 2^n$$

Final result:

$$\log(n) < \log^3(n) < \sqrt{n} < n\log^2(n) < n^{1.01} < 5^{\log_2 n} < 2^n = 2^{n+1} < n2^n < 3^n$$

Part 4:

1- Find the minimum-valued item.

```
public static int minVal(ArrayList<Integer> arr){
    if(arr.size() == 0) return -1; // o(1)
    int res = arr.get(0);// o(1)

    for(int i = 1;i<arr.size();i++){// o(n)
        if(res > arr.get(i))res = arr.get(i) ;// o(1)
    }

    return res;// o(1)
}
```

```
/*
1-start
2-if list is empty return -1
3- create  res with list`s first value
4-check all members of list and if it is smaller than res rewrite res with that value.
5-return res
*/
```

Because of the worst case

Time complexity : o(n)

2- Find the median item. Consider each element one by one and check whether it is the median.

```java
public static int findMedian(ArrayList<Integer> list){
    //o(n^2)
    for(int i = 0;i<list.size();i++){//O(n)
        int big = 0;//O(1)
        int eq = 0;//O(1)
        for(int j = 0;j< list.size();j++){//O(n)
            if(i == j) continue;//O(1)
            if(list.get(i) == list.get(j))eq++;//O(1)
            if(list.get(i) > list.get(j))big++;//O(1)
        }
        if(big <= list.size()/2 && big+eq >= list.size()/2) return list.get(i);//O(1)
    }

    return -1;//O(1)
}
```

```
/*
1-start
2-from i to list size
    create big for the hold bigger values number
    create eq  for the hold equal  values number
    from j to list size
        if i equal j continue
        if list(i) equal list(j) increment eq
        if list(i) bigger list(j) increment big
    if big smaller or equal list size/2 and big+eq bigger or equal list size/2 return list(i)
3-return -1
*/
```

Because of the worst case

Time Complexity : O(n^2)

3- Find two elements whose sum is equal to a given value

```
public static ArrayList<Integer> twoSum(ArrayList<Integer> arr,int goal){
    ArrayList<Integer> res = new ArrayList<>();//o(1)
    //O(n^2)
    for(int i = 0;i<arr.size();i++){//O(n)
        for(int j = 0;j< arr.size();j++){//o(n)
            if(i == j)continue;//O(1)
            if(arr.get(i)+arr.get(j) == goal){//O(1)
                res.add(arr.get(i));
                res.add(arr.get(j));
                i = arr.size();//O(1)
                j = arr.size();//O(1)
            }
        }
    }

    return res;//O(1)
}
```

```
/*
1-start
2-create new result list
3-check all 2 combinations of list if they are equal to goal add them to result list and break else continue
4-return res
*/
```

Because of the worst case

Time complexity : o(n)

4- Assume there are two ordered array list of n elements. Merge these two lists to get a single list in increasing order.

```
public static ArrayList<Integer> mergeTwoSortedList(ArrayList<Integer> arr1,ArrayList<Integer> arr2){
    ArrayList<Integer> res = new ArrayList<>();//O(1)
    int i1 = 0;//O(1)
    int i2 = 0;//O(1)

    while(i1!=arr1.size() || arr2.size() != i2){//O(mn) m = arr1.size() , n = arr2.size()
        if(i1==arr1.size()){//O(1)
            res.add(arr2.get(i2++));//O(1)
        }
        else if(i2==arr2.size()){//O(1)
            res.add(arr1.get(i1++));//O(1)
        }
        else if(arr1.get(i1) >= arr2.get(i2)){//O(1)
            res.add(arr2.get(i2++));//O(1)
        }
        else
            res.add(arr1.get(i1++));//O(1)
    }

    return res;//O(1)
}
```

```
/*
1-start
2-create result list
3-create i1 for index of first list
4-create i2 for index of second list
5-while i1 not equal arr1 size or arr2 size not equal i2
    if i1 equal arr1 size
        add second list`s current member to result and increment i2
    else if i2 equal arr2 size
        add first list`s current member to result amd increment i1
    else if arr1(i1) bigger or equal arr2(i2)
        add second list`s current member to result and increment i2
    else
        add first list`s current member to result amd increment i1
6-return result
*/
```

Because of the worst case

Time complexity : o(mn)

Part 5:

a)

int p_1 (int array[]):

{

```
        return array[0] * array[2])    -> O(1)
    }
```

Time complexity : O(1)

Space Complexity : O(1) (because we did not allocate any memory in function)

b)

```
int p_2 (int array[], int n):
 {
    Int sum = 0    -> O(1)
    for (int i = 0; i < n; i=i+5)    -> O(n)
        sum += array[i] * array[i])    ->O(1)
    return sum    -> O(1)
}
```

Time complexity : O(n)

Space Complexity : O(1) (because we did not allocate any memory in function)

c)

```
void p_3 (int array[], int n):
{
        for (int i = 0; i < n; i++)    -> O(n)
                for (int j = 0; j < i; j=j*2)    -> O(logn) (worst case) because j multiplying by two
                        printf(“%d”, array[i] * array[j])    -> O(1)
}
```

two for loop (for loop inside for loop) so we have to multiply them

Time complexity : O(nlogn)

Space Complexity : O(1) (because we did not allocate any memory in function)

d)

```
void p_4 (int array[], int n):
    {
        If (p_2(array, n)) > 1000)    -> O(1)
```

```
                p_3(array, n)    -> O(nlogn)
        else
                printf("%d", p_1(array) * p_2(array, n))    -> O(n)
    }
```

Time complexity : O(nlogn)

Space Complexity : O(1)(because we did not allocate any memory in function)