GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 2 Report

Part 1:

I. Searching a product.

```
public <T> boolean  searchProductHelper(Branch currentBranch,MyArray<T> array,Furniture searchObject){
    for (int i = 0 ; i < array.getSize(); i ++){            Q(n)
        array.getArrayElement(i).toString().equalsIgnoreCase(searchObject.toString());       Q(1)
        return true;          Q(1)
    }
    return false;  Q(1)
}
```

```
public void searchProduct(Company company,Furniture searchObject){
    int counter = 0;
    System.out.println("Branches with stock related to" + searchObject.typeProduct + " " + search
    for (int i = 0 ; i < company.getBranches().getSize(); i++){          Q(m)
        Branch tempBranch = (Branch) company.getBranches().getArrayElement(i);
        if (searchObject.getTypeProduct().equals("Chair")){
            if(searchProductHelper(tempBranch,tempBranch.getChairs(),searchObject)) {      Q(n)
                counter++;
                System.out.println(counter + "." + tempBranch.getBranchName() + "Branch");
            }
        }
```

Part 1

1) Searching a Product

searchProductHelper → Time Complexity = $T(n) = \Theta(n)$

(n = size of furnitur array)

searchProduct →

$T(m,n) = \Theta(m) \times (\Theta(n) + \Theta(n) + \Theta(n) + \Theta(n))$

(m = size of branches)

$T(m,n) = \Theta(m) \times \Theta(4n) = \Theta(m.n)$

II.Add/remove product.

```java
public boolean add (T newObject) {
    if (array == null)
        array = new Object[initialCapacity];

    if (this.initialCapacity <= size) {
        initialCapacity+=10;
        Object[] newArray = new Object[initialCapacity];
        for(int i = 0; i < size; i++)
            newArray[i] = array[i];

        array= (Object[]) newArray;
    }
    array[size]=newObject;
    size++;

    return true;
}

/**
 * Remove element from array and decreased size of array
 * @param value object to be deleted
 */
public  void delete (T value) {
    int index = 0;
    for(int i=0; i<size; i++) {
        if (value == array[i])
            index = i;
    }
    for ( int i = index; i!= size; i++)
        array[i]=array[i+1];
    --size;
}
```

Q(1)

Q(n)

2) Add / Remove Product

Add product $= T(n) = \Theta(n) \rightarrow$ for loop

(n= size of array)

Remove Product $= T(n) = \Theta(n) + \Theta(n) = \Theta(2n)$

$= \Theta(n)$

n: size of array

Part 2:

a) Explain why it is meaningless to say: "The running time of algorithm A is at least $O(n^2)$".

Solution:

It is meaningless because;

$O(n^2)$ = It is a worst case scenerio of running time so running time of algorithm A will be n^2 or faster.But algoritm A could be anything that is smaller. Example:Constant 1 or n…

So there is no information about the lower bound of an algorithm and lower bound without upper bound isn't useful .

b) Let f(n) and g(n) be non-decreasing and non-negative functions. Prove or disprove that: $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

Solution:

$f(n) \le \max(f(n), g(n))$

$g(n) \le \max(f(n), g(n))$

$f(n) + g(n) \le 2\max(f(n), g(n))$

- $\frac{1}{2}(f(n) + g(n)) \le \max(f(n), g(n))$
- $\max(f(n), g(n)) \le 1(f(n) + g(n))$

So $c1 (f(n)+g(n)) \le \max(f(n),g(n)) \le c2(f(n)+g(n))$ for $n > n0$ => $c1 = \frac{1}{2}$ $c2 = 1$

it holds true for $c1 = \frac{1}{2}$ and $c2 = 1$

c) Are the following true? Prove your answer.

I. $2^{n+1} = \Theta(2^n)$

II. $2^{2n} = \Theta(2^n)$

III. Let $f(n)=O(n^2)$ and $g(n)= \Theta(n^2)$. Prove or disprove that: $f(n) * g(n) = \Theta(n^4)$.

Solution:

c)

I. $2^{n+1} = \Theta(2^n)$

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} \frac{\cancel{2^n}.2}{\cancel{2^n}} = \lim_{n\to\infty} 2 = 2$$

If $\lim_{n\to\infty} = c$ is constant $\in R$ $f(n) = \Theta(g(n))$

It is true ✓

II. $2^{2n} = \Theta(2^n)$

$2^{2n} = 4^n \rightarrow c_1 . 4^n \leq 2^n \leq c_2 . 4^n$

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty \Rightarrow \Theta(2^{2n}) > \Theta(2^n)$$

It is false X

III. $f(n) = O(n^2)$, $g(n) = \Theta(n^2) \rightarrow f(n) * g(n) = \Theta(n^4)$ X

It is false. Because $O(n^2)$ worst case scenerio of $f(n)$

Maybe;

$f(n) = \Theta(n) \longrightarrow f(n) * g(n) = \Theta(3)$

$f(n) = \Theta(1) \longrightarrow f(n) * g(n) = \Theta(2)$

If we dont have lower bound we couldn't say

$f(n) * g(n) = \Theta(n^4)$ X

It is false.

Part 3:

List the following functions according to their order of growth by explaining your assertions.

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

Solution:

Part 3

$O(1)$
$O(\log n)$
$O(n.\log n)$
$O(n^2)$
$O(n^3)$
$O(2^n)$
$O(n!)$

$\downarrow$ order of growth

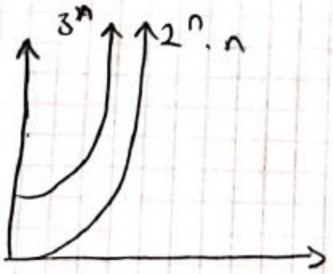
$2^n, 2^{n+1}, 5^{\log_2 n}, n.2^n, 3^n$ exponential numbers so they will be greatest

$\log n$, $(\log n)^3$ logaritmic. They will be smallest.

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

$$2^{n+1} = 2^n.2 \Rightarrow O(2^n.2) = O(2^n) \Rightarrow$$ constant isn't useful.

$$\sqrt{n} = n^{1/2} < n^{1.01}$$



Even in the diffirence in growth of 3^n and $2^n.n$ is small, it is even more 3^n

$$\lim_{n \to \infty} \frac{\log n}{(\log n)^3} = 0 \quad (\log n)^3 > \log n$$

Part 4:

Give the pseudo-code for each of the following operations for an array list that has n elements and analyze the time complexity:

- Find the minimum-valued item.
- Find the median item. Consider each element one by one and check whether it is the median.
- Find two elements whose sum is equal to a given value
- Assume there are two ordered array list of n elements. Merge these two lists to get a single list in increasing order.

Solution:

Part 4)

a) Function minValue(arrList, n):
  smallest = arrList.get(0)
  For index From 1 to n: $\longrightarrow \Theta(n-1)$
    if arrList.get(i) < smallest $\}\ \Theta(1)$
      smallest = arrList.get(i) $\}\ \Theta(1)$
    EndIf
  EndLoop
  Return Smallest $\}\ \Theta(1)$ = constant time
End Function

$T(n) = \Theta(n-1) \cdot \Theta(1) + \Theta(1) = \Theta(n)$

$T(n) = \Theta(n)$

b) Function findMedian(arrList, n):
  For i from 0 to n $\longrightarrow \Theta(n)$
    For j from i+1 to n
      if arrList.get(i) > arrList.get(j):
        temp = arrList.get(j)
        arr.set(i, arr.get(j))
        arr.set(j, temp)
    EndIf
  EndLoop
  Endloop

$\Theta(n-i) = \Theta(n)$

$T(n) = \Theta(n-i) \cdot \Theta(n)$

$T(n) = \Theta(n^2)$

$\Theta(1)$ if arr.size % 2 == 0
  return (arr.get(arr.size() % 2) + arr.size()/2 + 1) / 2
$\Theta(1)$ else
  return arr.get(arr.size() % 2)

c) Function findSum (arrList, n, sum):

```
counter = 0
For i from 0 to n                           ) θ(n)    Doesnt matter for growth (i)*(i+1)
  for j from i+1 to n                       )
    if arr.get(i) + arr.get(j) == sum       ) θ(n)
      counter ++                            )
  EndIF
EndLoop
EndLoop
if counter >= 1       ) θ(1)
  return 1            )
else
  return 0            ) θ(1)
```

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

d) Function merge (arrList1, arrList2, n)

```
For i from 0 to n, i=i+2                                          → θ(n/2) = θ(n)
  if arrList1.get(i) > arrList2.get(i)                            )
    arrList3.add (i, arrList1.get(i))                             ) θ(1)
    arrList3.add( i+1, arrList2.get(i))                           )
  EndIf
  else
    arrList3.add (i, arrList2.get(i))                             ) θ(1)
    arrList4.add (i+1, arrList1.get(i)                            )
  End Else
EndLoop
```

$T(n) = \theta(\frac{n}{2}) + \theta(1) + \theta(1)$

$T(n) = \theta(n)$

Part 5:

a) int p_1 (int array[]):

```
{
    return array[0] * array[2]    } Θ(1) = O(1)
}
```

$T(n) = \Theta(1) = O(1) =$ Constant time

Space Complexity $= n =$ Size of array

b) int p_2 (int array[], int n):

```
{
    int sum = 0    } Θ(1)
    for (int =0; i<n; i=i+5) ——> Θ(n/5) = Θ(n)
        sum += array[i] * array[i]    } Θ(1)
    return sum    } Θ(1)
```

$T(n) = \Theta(n) \cdot \Theta(1) + \Theta(1) + \Theta(1)$

$T(n) = \Theta(n)$

Space Complexity $= n + 1$

c) void p_3 (int array, int n):

```
{
    for (int i=0; i<n; i++) ——> Θ(n)
        for (int j=1; j<i; j=j*2) ——> Θ(logn)
            printf("%d", array[i] * array[j]) ——> Θ(1)
}
```

$T(n) = \Theta(n) \cdot \Theta(\log n) = \Theta(n \cdot \log n)$

Space Complexity $= n + 1$

d) void p_4 (int array[], int n):

{

if (p-2(array,n) > 1000) $\rightarrow \Theta(n)$

p-3(array, n) $\rightarrow \Theta(n \cdot \log n)$

else

printf("%d", $\underbrace{\text{p-1(array)}}_{\Theta(1)}$ * $\underbrace{\text{p-2(array,n)}}_{\Theta(n)}$)

if statement = $\Theta(n \cdot \log n) + \Theta(n)$

else statement = $\Theta(n)$

$T_{best} = \Theta(n)$

$T_{worst} = \Theta(n) + \Theta(n \cdot \log n) = \Theta(n \log n)$

Space Complexity = $n+1$