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());
        return true;                    Q(1)                                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");
            }
        }
```

,II.Add/remove product.

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

    if (this.initialCapacity <= size) {
        initialCapacity+=10;                                          Q(1)
        Object[] newArray = new Object[initialCapacity];
        for(int i = 0; i < size; i++)
            newArray[i] = array[i];                                 Q(n)

        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;
}
```

Part 1

1) Searchig A Product

Search productHelper $\longrightarrow$ Time Complexity $\Rightarrow T(n) = \Theta(n)$

n = size of array furniture

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

m = size of branches

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

2) Add / Remove Product

Add Product

n = size of array container $\quad T(n) = \Theta(n)$

Remove Product

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

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:

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

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

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

- ½ (f(n) + g(n)) ≤ max(f (n), g(n))
- max(f (n), g(n)) ≤ 1(f(n) + g(n))

So c1 (f(n)+g(n)) ≤ max (f(n),g(n)) ≤c2(f(n)+g(n)) for n > *n*0 => c1 = ½ c2 = 1

it holds true for c1 = ½ 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. $2^{2n} = \Theta(2^n)$

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

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

It is false ✗

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

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

Maybe;

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

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

If we dont have lower bound we couldn't say

$f(n) * g(n) = \Theta(n^4)$ ✗

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}, $5^{\log_2 n}$, $\log n$

Solution:

Part 3,

$O(1)$

$O(\log n)$

$O(n.\log n)$

$O(n^2)$

$O(n^3)$

$O(2^n)$

$O(n!)$ ↓ Order Of Growth Rule

$2^n, 2^{n+1}, 5^{\log_2 n}, n.2^n, 3^n$ = Exponential

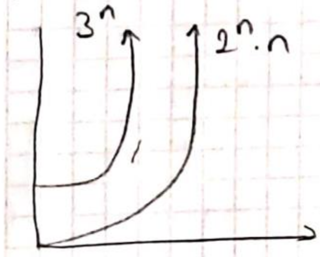
$\log n, (\log n)^3$ = logaritmic

$\sqrt{n}, n^{1.01}$ = Polynom

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

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

$\sqrt{n} = n^{1/2} \Rightarrow O(n^{1/2}) < O(n^{1.01})$



$O(3^n) > O(2^n\ n)$

So;

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

Part 4:

Give the pseudo-code for each of the following operations for an array list that has <u>n elements</u> 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

- Find the minimum valued item

```
Function minValue (arrList, n):
    smallest = arrList.get(0)                    ) θ(1)
    For i from 1 to n                            ) θ(n-1)
      if arrList.get(i) < smallest               ) θ(1)
         smallest = arrList.get(i)  ) θ(1)
      EndIf
    EndLoop
    Return smallest                              ) θ(1)
```

$T(n) = \theta(n-1) . \theta(1) + \theta($

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

n=size of array

- Find the median item

```
Function median (arrList, n):
  For i from 0 to n                                          ) θ(n²)
    For j from i+1 to n                                      ) θ(n)
        if arrList.get(i) > arrList.get(j)                   ) θ(1)
          temp = arrList.get(j)
          arrList.set(i, arr.get(j))
          arrList.set(j, temp)
      EndIf
    EndLoop
  EndLoop
  if arr.size % 2 == 0
      return (arrList.get (arr size()%2) + arr.size()%2 +1))/2   ) θ(1)
  else
    return arrList.get (arrList.size()/2)   ) θ(1)
```

$T(n) = \theta(n) . \theta(n) = \theta(n^2)$ n=size of array

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

```
Function findSum(arrList, sum):
counter = 0
For i from 0 to n
  For j from i+1 to n
    if arr.get(i) + arr.get(j) == sum      } Θ(1) } Θ(n) } O(n^2)
        counter ++
  EndIf
End Loop
EndLoop
  If   counter >= 1
        return true                        } Θ(1)
  else
      return false
```

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

n = size of array

- Merge two Lists.

```
Function merge (arrList 1, arrList 2, n)
 For  i from 0 to n , i=i+2
   if arrList1.get(i) > arrList2.get(i)      } Θ(1) } Θ(n)
    arrList3.add (i, arrList1.get(i))
    arrList3 . add( i+1, arrList 2 .get(i))
  EndIf
  else
       arrList 3.add (i, arrList2.get(i))    } Θ(1)
       arrList 3 .add (i+1 , arrList1 .get(i))
End Loop
```

$T(n) = \Theta(n) + \Theta(1) = \Theta(n) = O(n)$ n = size of array Lists

Part 5:

Part 5,

a) int p_1 (int array [ ]):

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

$T(n) = O(1)$ = constant time

Space Complexity = 1

b)

```
int p-2(int array[ ], int n)
{
    int sum = 0                              ) θ(1)
    for(int i=0; i<n; i=i+5)                 ) θ(n/5) = θ(n)
        sum += array[i] * array[i]           ) θ(1)
    return sum                               ) θ(1)
}
```

$T(n) = \theta(n)\,\theta(1) + \theta(1) = \theta(n) = O(n)$

Space Complexity = n

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) because 2^n ti
            printf ("%d", array[i] * array[j])  ) θ(1)
}
```

θ(n.logr

$T(n) = \theta(n).(\theta(\log n) + \theta(1)) = \theta(n \cdot \log n)$

Space Complexity = n

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

```
{
    if (p_2(array, n) > 1000)        } Θ(n)      } Θ(n) + Θ(n logn)
        p_3(array, n)                } Θ(n.logn)
    else
        printf("%d", p_1(array) * p_2(array, n))
}
```

$$\text{printf}("\%d", \underbrace{p\_1(array)}_{\Theta(1)} * \underbrace{p\_2(array,n)}_{\Theta(n)})$$

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

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

Space Complexity $= n$