GIT
DEPARTMENT OF
COMPUTER ENGINEERING
CSE 222/505 – Spring 2021

HOMEWORK 2 REPORT

GIT / CSE 222 - HOMEWORK 2

Name: Süleyman Gölbol No: 1801042656

PART1

When we using T_sum and multiply cost with number of times executed, cost doesn't matter because we know it and it is constant.

For example for i=0 to n , for that cost=2, and number of steps is 2*(n+1) which is → still Θ(2n+2) = O(n) so when I see a for loop like for i=0 to n, I will directly use Θ(n) (to not mention same thing in every part in part1.)

1) Searching a product

```
public boolean searchForProduct(Company c, Products.Product productType, Colors color, int model){
    for(int i=0; i<c.getAllBranches().size(); i++)
        if( c.getAllBranches().getIndex(i).findProductIndex(productType, color, model) != -1)
            return true;

    return false;
}
```

```
public int findProductIndex(Products.Product productType, Colors color, int model){
    int flag = -1;
    for(int i=0; i<products.size(); i++){
        if(productType == products.getIndex(i).getProduct() && color == products.getIndex(i).getColor() && model == products.getIndex(i).getModel() )
            flag = i;
    }
    return flag;
}
```

For searching a product, in a for loop, firstly we search for every branch. Then for every branch
we are trying to find the product index using with for loop.

If we say n (n and m are denoting steps and sizes of algorithm) to the first for loop (in searchForProduct() method) and m to the second for loop (in findProductIndex()) the worst case scenario would be Θ(n * m) . The other things in algorithm (for example flag=i; row) won't be counted because all of them are constant time.

2) Add/Remove Product

```
public void addProduct(Branch branchIn, Products product){
    branchIn.getProducts().add(product);
}
```

```
public boolean add(E e){
    int i;
    if( size() == 0 ){
        obj = (E[])new Object[size()+1];
        obj[0] = e;
        return true;
    }
    for(E i2: obj){  //If there is same element don't add it.
        if(e == i2)
            return false;
    }

    E[] temp = (E[])new Object[size() + 1]; //Copying to temp
    i=0;
    while( i++ < size() )
        temp[i-1] = obj[i-1];
    temp[size()] = e;

    int size = size();

    obj = (E[])new Object[size + 1];
    i=0;
    while( i++ < size() )
        obj[i-1] = temp[i-1];

    return true;
}
```

For the first for loop, we say its time is n (the size of obj), for the first while loop it is n too because while loop goes on until the size of obj, for third it is n+1, because I increased the size by 1 for the new number which will be added. So it will be Θ(n+n+n+1) which is same with Θ(n). The other things are in algorithm are not counted (for example i=0; row) because all of them are constant time.

```
public void removeProduct(Branch branchIn, Products product){
    branchIn.getProducts().remove(product);
}
```

```
public boolean remove(E e){
    int size = size();
    int i_temp = -1;

    for(int i=0; i<size; i++)
        if(obj[i] == e)
            i_temp = i;
    if(i_temp == -1)    //That means couldn't find e, so return false.
        return false;

    Object[] temp = new Object[size()];
    for(int i=0; i<size; i++)
        temp[i] = obj[i];
    clear();    //Clearing object
    for(int i=0; i<size; i++){  //Adding without the deleted one.
        if(i == i_temp)
            continue;
        add( (E) temp[i] );
    }

    return true;
}
```

In my remove method, there are 3 for loops. First for loop contains size steps which I call n.

Also second and third for loops algorithm sizes are n too. So for the case will be Θ(n+n+n) which is Θ(n) because constant numbers like 3 don't effect the algorithm time complexity of algrorithm because it is a constant number. The other things are in algorithm won't be counted (for example i_temp=i; row) because all of them are constant time.

3) Qerying the products that need to be suplied

```java
public boolean queryToSupply(Company c, int customerNumber){
    for(int i=0; i<c.getAllCustomers().size(); i++)
        if( c.getAllCustomers().getIndex(i).getCustomerNumber() == customerNumber ){
            for(int j=0; j<c.getAllCustomers().getIndex(i).getPreviousOrders().size(); j++)
                if( c.getAllCustomers().getIndex(i).getPreviousOrders().getIndex(j).substring(0, 1).equals("N") ){
                    char[] temp = c.getAllCustomers().getIndex(i).getPreviousOrders().getIndex(j).toCharArray();
                    temp[0] = 'S'; //S means supplied.
                    String s = String.valueOf(temp);
                    c.getAllCustomers().getIndex(i).setPreviousOrders(s, j);
                    return true;
                }
        }

    return false;
}
```

There are 2 nested for loops and if we say n to the denoting size of first for loop, and m to the denoting size of second first loop the algorithm will be be Θ(n*m). The other things in algorithm won't be counted because all of them are constant time.

PART2

a) Big O notation shows us the upper bounds of function which is max time that will take. It is meaningless because there could no be something like at least max value. If it is, it has to be stick just one value which is maximum and minimum value at the same time. So actually there is no min or max value for that expression.

b) Because of both of them are non negative functions, we can say that f(n) has to be smaller(or equals) than f(n) + g(n) and also g(n) has to be smaller(or eqauls) than f(n) + g(n). So it means which ever bigger from these two functions has to be element of O(f(n) + g(n)) (Because big O notation shows the upper bound of function). $(f(n) + g(n)) \geq max(f(n), g(n))$

Also, to find lower bound we can multiply max by 2.

$2max(f(n), g(n)) \geq (f(n) + g(n))$. This is true because we take max one from f(n) and g(n) and multiply by 2 it has to be bigger than sum of f(n) and g(n).

So $\frac{1}{2}(f(n) + g(n)) \leq max(f(n), g(n))$ (Ω notation)

Theta notation for $f(x) = \Theta(g(x))$ is $c_1 g(x) \leq f(x) \leq c_2 g(x)$ and when we combine notations $\frac{1}{2}(f(n) + g(n)) \leq max(f(n), g(n)) \leq (f(n) + g(n))$

c.1) Theta Equation for f(x) = Θ(g(x)) is $f(x) = \Theta(g(x))$ & $c_1 g(x) \leq f(x) \leq c_2 g(x)$ and $2^{n+1} = 2.2^n$

So $c_1 2^n \leq 2.2^n \leq c_2 2^n$. If we simplify 2^ns then c1 ≤ 2 ≤ c2 and there exists contants for c1 and c2 like c1=1 and c2=3 because 2 is constant.

c.2) Theta Equation for f(x) = Θ(g(x)) is $f(x) = \Theta(g(x))$ & $c_1 g(x) \leq f(x) \leq c_2 g(x)$

$c_1 2^n \leq (2^n)^2 \leq c_2 2^n$ and we simplify 2^ns equation will be $c_1 \leq 2^n \leq c_2$. If we take logarithm in take base 2 of everything in equation.

$log_2 c_1 \leq log_2 2^n \leq log_2 c_2 = log_2 c_1 \leq n.log_2 2 \leq log_2 c_2$ and $log_2 2 = 1$

so = $log_2 c_1 \leq n \leq log_2 c_2$. But there is no any constant c_2 for all n(that is really big) such like $n \leq log_2 c_2$. Whatever we make c_2 we can find a bigger n. So there is no upper bound (O notation) and this means there is no Θ notation.

c.3) For f(n)=O(n²) there exists 0 ≤ f(n) ≤ cn² and for g(n)=Θ(n²) there exists $c_1 n^2 \leq g(n) \leq c_2 n^2$

$c_1 n^2$ is also bigger than 0 so If we multiply f(n) and g(n) than;

$0 \leq f(n).g(n) \leq c.c_2.n^4$ $c.c_2$ is also a constant.

for $\Theta(n^4)$ = f(n).g(n) equation is → $c_3.n^4 \leq f(n).g(n) \leq c_4.n^4$ → $c_3.n^4$ is not equal to 0 so it is disproved.

PART3

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

To order all of these we can take 2 of them and compare them with limits going to ∞.

$\lim_{n\to\infty} \frac{3^n}{n.2^n}$ = $\lim_{n\to\infty} \frac{(\frac{3}{2})^n}{n}$ because of exponential functions grow faster than polynoms and 3/2 ≥ 1 it should be = ∞. So $3^n > n.2^n$

$2^{n+1} = 2.2^n$ and because of 2 is constant and 2^n and 2^{n+1} should have equal growth.

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

$\lim_{n\to\infty} \frac{2^n}{5log_2 n}$ = $\lim_{n\to\infty} \frac{1}{5}.\frac{2^n}{log_2 n}$ and if we use L'Hopital Rule $\frac{1}{5}.\lim_{n\to\infty} \frac{2^n ln2}{\frac{1}{nln2}}$ = $\frac{1}{5}.\lim_{n\to\infty} 2^n (ln2)^2.n$

= $\frac{(ln2)^2}{5}.\lim_{n\to\infty} 2^n.n$ = ∞ so $2^n > 5log_2 n$

$\lim_{n\to\infty} \frac{5^{log_2 n}}{n^{1.01}} = \lim_{n\to\infty} \frac{n^{log_2 5}}{n^{1.01}}$ because of $log_2 5 = 2.32$ so $\lim_{n\to\infty} n^{2.32-1.01} = \lim_{n\to\infty} n^{1.31} = \infty$

So $5^{log_2 n} > n^{1.01}$

$\lim_{n\to\infty} \frac{n^{1.01}}{nlog^2 n} = \lim_{n\to\infty} \frac{n^{0.01}}{log^2 n}$ with L'Hospital rule → $\lim_{n\to\infty} \left(\frac{0.005n^{0.01}}{\log(n)}\right) = \lim_{n\to\infty} \left(5.10^{-5} n^{0.01}\right) = 5.10^{-5} \left(\lim_{n\to\infty} (n)\right)^{0.01}$ for n=∞ → =∞

so $n^{1.01} > nlog^2 n$

$\lim_{n\to\infty} \frac{nlog^2 n}{\sqrt{n}} = \lim_{n\to\infty} \left(\sqrt{n}\log(n)^2\right) = \lim_{n\to\infty} (\sqrt{\infty}) = \infty$ so $nlog^2 n > \sqrt{n}$

$\lim_{n\to\infty} \frac{\sqrt{n}}{(logn)^3} = \lim_{n\to\infty} \left(\frac{\ln(10)\sqrt{n}}{6\log_{10}(n)^2}\right) = \lim_{n\to\infty} \left(\frac{\frac{\ln(10)}{2\sqrt{n}}}{\frac{12\log_{10}(n)}{\ln(10)n}}\right) = \lim_{n\to\infty} \left(\frac{\ln^2(10)\sqrt{n}}{24\log_{10}(n)}\right) = \lim_{n\to\infty} \left(\frac{\ln^3(10)}{48}\sqrt{n}\right) = \frac{\ln^3(10)}{48}\sqrt{\lim_{n\to\infty} (n)}$ and $\sqrt{\infty} = \infty$ so $\sqrt{n} > (logn)^3$

Lastly $log^3 n/logn = log^2 n$ and $\lim_{n\to\infty} log^2 n = \infty$ so $log^3 n > logn$

When we combine all of them order is;
$3^n > n.2^n > n.2^n > 2^{n+1} = 2^n > 5log_2 n > n^{1.01} > n.log^2 n > \sqrt{n} > (logn)^3 > logn$

PART4

Firstly when using Tsum and multiply cost with number of times, cost doesn't matter.

For example for i=0 to n , for that cost=2, and number of steps is 2*(n+1) which is still O(2n+2) = O(n) so when I see a for loop like for i=0 to n, I will directly use O(n)/Θ(n) (to not mention same thing in every part.)

Another thing I mention is, for i=0 to n, when I use to n, n is not included, it is like i<n, not i≤n.

```
-- finding minimum valued item
Variable min = -1
for variable i=0 to n
  if array's index of i'th element < min
    min = i
```

For loop's time complexity is O(n) [I mentioned why O(n) in the beginning of part4] and the things inside it is constant time. So complexity is Θ(n).

```
-- finding median
-- Firstly sorting all elements with Selection Sort method
Create variables valueTemp, minimum

for variable i=0 to n
    minimum = i
    for variable j=i+1 to n
      if array's j'th element < array's minimum'th element
        minimum = j

    Make valueTemp array's i'th element
    Make array's i'th element array's minimum'th element
    Make array'S minimum'th element valueTemp
-- Now array is sorted in increasing order from small to bigger.
Create variable median
-- Checking every element one by one
for variable i=0 to n
  if n is divisible by 2 and array's i'th element equals (array's (n+1)/2'th element
    median = array's i'th element
  else if n isn't divisible by 2 and array's i'th element equals array's n/2'th element
    median = array's i'th element
```

Selection sort's time complexity is $\Theta(n^2)$ because there are 2 nested loops in it. And because of the thing that I wrote in beginning of Part4, the other things in selection algorithm aren't counted.

When determining the median, for loop's time complexity is Θ(n) because there is one for loop which is continues from 0 to n. Because for loops are not nested we should sum them. $\Theta(n^2+n)$ equals $\Theta(n^2)$.

```
-- finding 2 elements whose sum equal to a given value.
Variable element1 , element2
for variable i=0 to n
  for variable j=0 to n
    if array's index of i'th element + array's index of j'th element equals givenValue
        element1 = array's index i'th element
        element2 = array's index j'th element
```

Because of 2 nested for loop, and inside loops all things are constant time, it should be $\Theta(n*n) = \Theta(n^2)$.

```
-- merging 2 ordered lists with order.
Create new List called list in size 2n
Create variable cursor
for variable i=0, j=0 to n and every step increase i,j by 1.
    if array2's index of j'th element < array1's index of i'th element
      make list's cursor'th element array2's j'th element
      increase cursor by 1 and decrease i by 1
    else if array1's index of i'th element <= array2's index of j'th element
      make list's cursor'th element array1's i'th element
      increase cursor by 1 and decrease j by 1
-- for to put last elements left in array to new array.
while i+j < 2n
  if i<n
    make list's cursor'th element array1's i'th element
    increase cursor and i by 1
  if j<n
    make list's cursor'th element array2's j'th element
    increase cursor and j by 1
```

In the first for loop, doesn't matter which 'if' statement is true because time complexities of if's are same and constant time. Also for the while loop, 'if' statements also has the same time complexities. So there worst case and best case scenarios are same. For loop → $\Theta(n)$ and while loop → $\Theta(n)$.

So time complexity is $\Theta(n+n) = \Theta(2n)$ which is same with $\Theta(n)$ because 2 is constant and ignorable.

PART5

```
a)
int p_1 (int array[]){
```

```
  return array[0] * array[2])
}
```

Time complexity: Θ(1) because just returning a constant.

Space complexity: O(1) because we use new memory just for a constant.

```
b)
int p_2 (int array[], int n){
  int sum = 0
  for (int i = 0; i < n; i=i+5)
    sum += array[i] * array[i])
  return sum
}
```

Time complexity : Θ(n) because for loop runs n time and other constant time doesn't effect.

Space complexity: O(1) because we use new memory just for a constant.

```
c)
void p_3 (int array[], int n){
  for (int i = 0; i < n; i++)
    for (int j = 0; j < i; j=j*2)
      printf("%d", array[i] * array[j])
}
```

Time complexity: Θ(nlogn) because for first loop Θ(n) and for second for loop j increases every time multiplying by 2 so this is a logarithmic increase ($log_2 n$) , it will go logn time so it should be Θ(logn).

When we multiply Θ(n) and Θ(logn) → time complexity becomes Θ(nlogn).

Space complexity: O(1) because we use new memory just for a constant.

```
d)
void p_4 (int array[], int n){
if (p_2(array, n)) > 1000)
  p_3(array, n)
else
  printf("%d",  p_1 (array) * p_2 (array, n))
}
```

Time complexity: For worst case → $\Theta(n^2 \log n)$ because firslt we call p_2(array,n) method which is $\Theta(n)$, than if statement is true we also call p_3(array,n) method which is $\Theta(n \log n)$. When we combine them both worst case scenario becomes $\Theta(n^2 \log n)$.

For best case → $\Theta(n^2)$ because firstly we call p_2(array,n) method which is $\Theta(n)$, than if statement is false we also call p_1(array) * p_2 (array, n) which is $\Theta(n*1)$. When we combine them it becomes $\Theta(n*n)$ which is $\Theta(n^2)$.

Space complexity: $O(1)$ because we use new memory just for a constant.