Mehmet Acar 1801042095

1-

I. Searching a product.

```
System.out.println("\nWhich product do you want to see?");   -> Θ(1)
System.out.println("1-Office Chairs");    -> Θ(1)
System.out.println("2-Office Desks");     -> Θ(1)
System.out.println("3-Meeting Tables");   -> Θ(1)
System.out.println("4-Bookcases");        -> Θ(1)
System.out.println("5-Office Cabinets");  -> Θ(1)
System.out.printf("Enter your choice:");  -> Θ(1)
```

```
product_ch=s5.nextInt();  -> Θ(1)
```

```
System.out.println("\nWhich model do you want to see?");   -> Θ(1)
```

```
for(i=0;i<product[0][product_ch-1].getModel_num();i++){ ->Θ(product[0][product_ch-1].getModel_num())
    System.out.printf("%d- Model %d\n",i+1,i+1);  -> Θ(1)
}
```

```
System.out.printf("Enter your choice:");  -> Θ(1)
```

```
model_ch=s5.nextInt();  -> Θ(1)
```

```
System.out.println("\nWhich color do you want to see?");  -> Θ(1)
```

```
for(i=0;i<product[0][product_ch-1].getColor_num();i++){ ->Θ(product[0][product_ch-1].getColor_num())

    System.out.printf("%d- Color %d\n",i+1,i+1);  -> Θ(1)
}
```

```
System.out.printf("Enter your choice:");  -> Θ(1)
```

```
color_ch=s5.nextInt();  -> Θ(1)
```

If we assume that Θ(product[0][product_ch-1].getModel_num()) value is equal to k;

Θ(product[0][product_ch-1].getColor_num()) value is equal to l,

T(n)= Θ(k) + Θ(l)

= Θ(k+l)

Θ(1) has no effect in this addition operation. Therefore, I did not write it.

II. Add/remove product.

In Main.java

. Inside of this condition(if(ch==2)) is $T_4(n)$

```
if(ch==2) {    -> T6(n)

    System.out.printf("\nHow many product do you want to add:"); -> Θ(1)
    product_amount=s3.nextInt();  -> Θ(1)
    product[branch_choice-1][product_ch-1].add_product(model_ch-1, color_ch-1,
                                                       product_amount); -> Θ(1)

    System.out.println("Adding product to selected branch completed
                            successfully."); -> Θ(1)
 }
```

Inside of this condition(else if(ch==3)) is $T_5(n)$.

```
else if(ch==3) {

    System.out.printf("\nHow many product do you want to remove:");  -> Θ(1)
    product_amount=s3.nextInt();  -> Θ(1)

  if(product[branch_choice-1][product_ch-1].get_ProductNum(model_ch-1,color_ch-1)-
     product_amount<0) {    -> T3(n) = Θ(1)

     System.out.println("The amount of product you want to remove from this branch
                           is incorrect.");  -> T1(n) = Θ(1)

  }
```

Inside of this else condition is $T_2(n)$

```
  else {
     product[branch_choice-1][product_ch-1].remove_product(model_ch-1,color_ch-1,
                                                           product_amount); -> Θ(1)

     System.out.println("Removing product from selected branch completed
                           successfully.");  -> Θ(1)
  }

}
```

Helper functions in BranchProduct.java

```
public void add_product(int model_index,int color_index,int product_val) {
   product_num[model_index][color_index]+=product_val;
}

public void remove_product(int model_index,int color_index,int product_val) {
   product_num[model_index][color_index]-=product_val;
}
```

```
public int get_ProductNum(int model_index,int color_index) {
    return product_num[model_index][color_index];
}
```

$T_{W1}(n) = T_3(n) + \max(T_1(n), T_2(n))$

$= \Theta(1) + \max(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

$T_{b1}(n) = T_3(n) + \min(T_1(n), T_2(n))$

$= \Theta(1) + \min(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

$T_5(n) = \Theta(1) + \Theta(1) + \Theta(1)$
$= \Theta(1)$

$T_{W2}(n) = T_6(n) + \max(T_4(n), T_5(n))$

$= \Theta(1) + \max(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

$T_{b2}(n) = T_6(n) + \min(T_4(n), T_5(n))$

$= \Theta(1) + \min(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

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

III. Querying the products that need to be supplied.

In Main.java

```
for(i=0;i<branch_num;i++) {   -> ϴ(branch_num)
    for(j=0;j<5;j++) {    -> ϴ(1)
        for(k=0;k<product[i][j].getModel_num();k++) {   -> ϴ( product[i][j].getModel_num() )
            for(l=0;l<product[i][j].getColor_num();l++) {   -> ϴ( product[i][j].getColor_num() )
                if(product[i][j].get_ProductNum(k, l)==0)   -> ϴ(1)
                  System.out.printf("In %d. branch %s Model %d Color %d product is need to be
                                          supplied\n",  i+1, product[i][j].getName() , k+1 , l+1); -> ϴ(1)
                }
            }
        }
}
```

Helper functions in Furniture.java

```
public int getModel_num() {
    return model_num;
}
public int getColor_num() {
    return color_num;
}
```

Helper function in BranchProduct.java

```
public int get_ProductNum(int model_index,int color_index) {
   return product_num[model_index][color_index];
}
```

If we assume that branch_num value is equal to m;

product[i][j].getModel_num() value is equal to n;

product[i][j].getColor_num() value is equal to o,

$T(n) = \Theta(m) \,.\, \Theta(n) \,.\, \Theta(o)$

$= \Theta(mno)$

ϴ(1) has no effect in this operation. Therefore, I did not write it.

2-

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

$O(n^2)$ is worst situation for running time of algorithm A. So, running time of algorithm A could be equal to n^2 or faster. In other words, running time of this algorithm could be at most n^2.

Therefore, “The running time of algorithm A is at least O(n2)” sentence is meaningless.

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)) operation selects high order function. If f(n) function’s order is bigger than g(n), selects f(n) order. If g(n) function’s order is bigger than f(n), selects g(n) order.

Θ(f(n) + g(n)) expression can be written in this way: Θ(f(n)) + Θ(g(n))

Θ(f(n)) + Θ(g(n)) operation holds only high order of result. This information means that if f(n) function’s order is bigger than g(n), addition result’s high order is f(n) order. If g(n) function’s order is bigger than f(n), addition result’s high order is g(n) order.

These two informations proves that max(f (n), g(n)) = Θ(f(n) + g(n)).

Also, we can prove max(f (n), g(n)) = Θ(f(n) + g(n)) expression by giving values to f(n) and g(n) function. Let’s assume that $f(n) = n^2$ and $g(n) = n$

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

$\max(n^2,n) = \Theta(n^2) + \Theta(n)$

$n^2 = n^2$ This equality proves that max(f (n), g(n)) = Θ(f(n) + g(n))

c)

Rules

$\lim_{N\to\infty} f(N)/g(N)=0$ -> $f(N) = o(g(N))$

$\lim_{N\to\infty} f(N)/g(N)= c!=0$ -> $f(N) = \Theta (g(N))$

$\lim_{N\to\infty} f(N)/g(N)= \infty$ -> $g(N) = o(f(N))$

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

$$\lim_{n\to\infty} 2^{n+1}/2^n = \lim_{n\to\infty} (2^n \cdot 2) / 2^n = \lim_{n\to\infty} 2 = 2$$

This result provides that $\lim_{N\to\infty} f(N)/g(N) = c != 0$ -> $f(N) = \Theta(g(N))$

So, $2^{n+1} = \Theta(2^n)$ is true.

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

$$\lim_{n\to\infty} 2^{2n}/2^n = \lim_{n\to\infty} 2^n = \infty$$

This result provides that $\lim_{N\to\infty} f(N)/g(N) = \infty$ -> $g(N) = o(f(N))$

So, $2^{2n} = \Theta(2^n)$ is false.

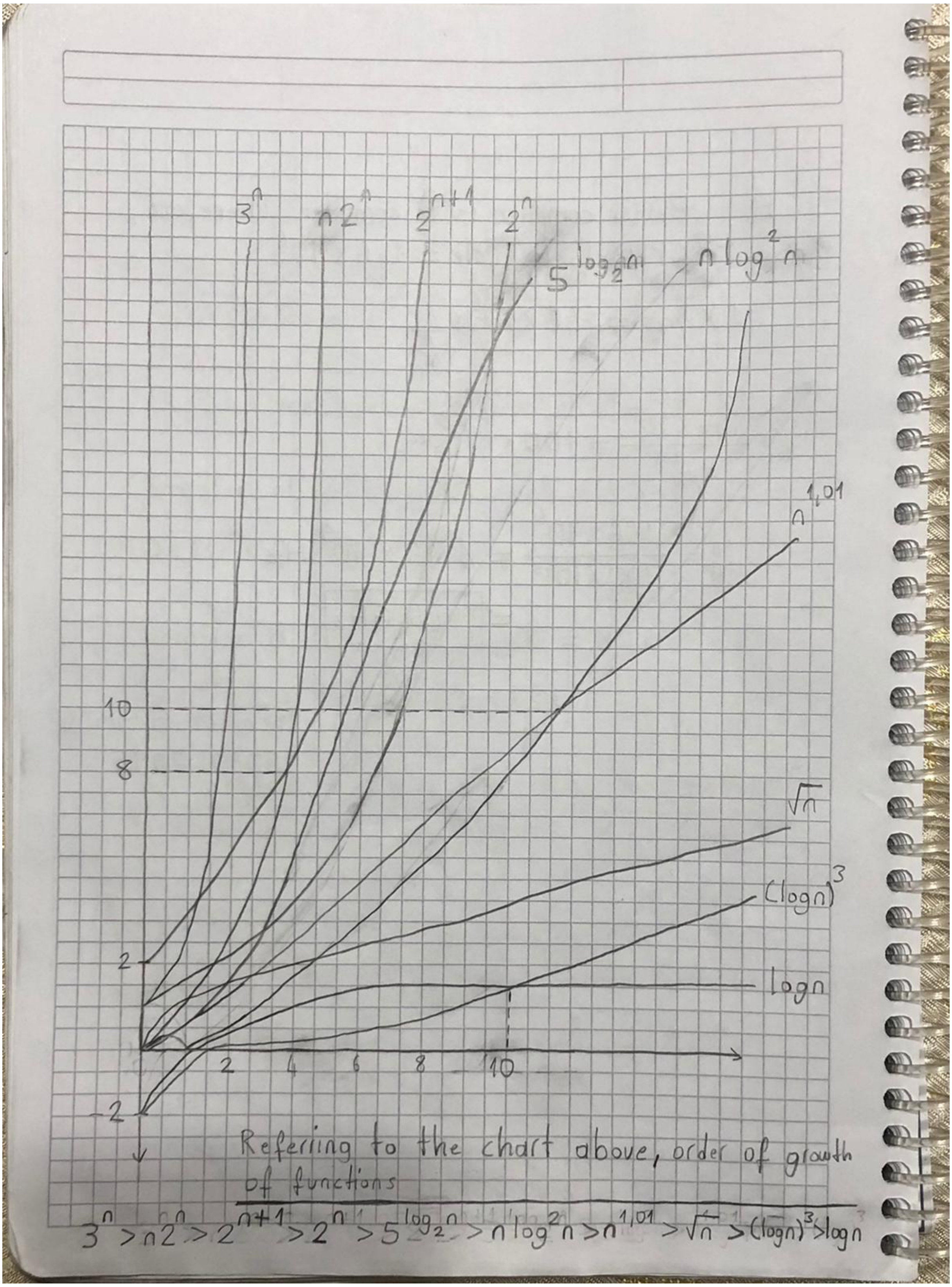
III. Let $f(n)=O(n^2)$ and $g(n)= \Theta(n^2)$. Prove or disprove that: $f(n) * g(n) = \Theta(n^4)$.

If $f(n)=O(n^2)$, $f(n)$ can be $\Theta(n^2)$, $\Theta(n)$ or $\Theta(1)$.

Therefore, $f(n) * g(n)$ can be $\Theta(n^4)$, $\Theta(n^3)$ or $\Theta(n^2)$.

As a result, , $f(n) * g(n)$ can be $\Theta(n^4)$ but it is not certain.

3.



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

4-

a)Find the minimum-valued item.

```
input ArrayList<Integer> arrlist (parameter)
n= arrlist.size()  -> Θ(1)
min= arrlist.get(0)  -> Θ(1)
for i=0 to n   -> Θ(n)
  if  arrlist.get(i) < min   ->Θ(1)
     min= arrlist.get(i)     ->Θ(1)
   endif
endloop1
output min (return)     -> Θ(1)
```

If statement running time $T_1(n) = \Theta(1). \Theta(1) = \Theta(1)$

loop1 running time $T_2(n) = \Theta(n). T_1(n) = \Theta(n). \Theta(1) = \Theta(n)$

Running time of code $T(n) = T_2(n) + \Theta(1) + \Theta(1) + \Theta(1)$

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

$= \Theta(n)$

b)Find the median item. Consider each element one by one and check whether it is the median.

```
input ArrayList<Integer> arrlist (parameter)
Collections.sort(arrlist)  -> Θ (n*log(n))

if arrlist.size() mod 2 == 1    -> T3(n)
   output arrlist.get( arrlist.size()/2 )  (return)   -> T1(n)
endif
else
   output ( arrlist.get( (arrlist.size()/2)-1 ) + arrlist.get( arrlist.size()/2 ) ) /2    -> T2(n)
endelse
```

$T_{W1}(n) = T_3(n) + \max(T_1(n) , T_2(n))$

$= \Theta(1) + \max(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

$T_{b1}(n) = T_3(n) + \min(T_1(n), T_2(n))$

$= \Theta(1) + \min(\Theta(1), \Theta(1))$

$= \Theta(1) + \Theta(1)$

$= \Theta(1)$

$T(n) = \Theta(n*\log(n)) + \Theta(1)$

$= \Theta(n*\log(n))$

c)Find two elements whose sum is equal to a given value

```
input ArrayList<Integer> arrlist (parameter)
input value (parameter)
n= arrlist.size()  -> Θ(1)
for i=0 to n-1     -> Θ(n)
  for j=i+1 to n   -> Θ(n)
      if  arrlist.get(i) + arrlist.get(j) == value   -> Θ(1)
        print   arrlist.get(i) "and"  arrlist.get(j)  "are two elemets whose sum is equal to given
                                                      value"  -> Θ(1)
        output 1 (return)  -> Θ(1)
      endif
    endloop1
endloop2
    output 0 (return)  -> Θ(1)
```

If arrlist's first two elements' sum is equal to given value, this is the best case.

If statement running time $T_{1b}(n) = \Theta(1).\ \Theta(1) = \Theta(1)$

loop1 running time $T_{2b}(n) = \Theta(1).\ T_{1b}(n) = \Theta(1).\ \Theta(1) = \Theta(1)$

loop2 running time $T_{3b}(n) = \Theta(1).\ T_{2b}(1) = \Theta(1).\ \Theta(1) = \Theta(1)$

Running time of best case is $T_b(n) = T_{3b}(n) + \Theta(1) + \Theta(1)$

$= \Theta(1) + \Theta(1) + \Theta(1)$

$= \Theta(1)$

If we can not find any two elements whose sum is equal to given value, this is the worst case.

If statement running time $T_{1w}(n) = \Theta(1). \Theta(1) = \Theta(1)$

loop1 running time $T_{2w}(n) = \Theta(n). T_{1w}(n) = \Theta(n). \Theta(1) = \Theta(n)$

loop2 running time $T_{3w}(n) = \Theta(n). T_{2w}(1) = \Theta(n). \Theta(n) = \Theta(n^2)$

Running time of worst case is $T_w(n) = T_{3w}(n) + \Theta(1) + \Theta(1)$

$= \Theta(n^2) + \Theta(1) + \Theta(1)$

$= \Theta(n^2)$

Running time of this code $T(n) = O(n^2)$

$= \Omega(1)$

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

```
input ArrayList<Integer> arrlist1 (parameter)

input ArrayList<Integer> arrlist2 (parameter)

ArrayList<Integer> arrlist3       ->Θ(1)

for i=0 to arrlist1.size()        ->Θ(n)

   arrlist3.add( i, arrlist1.get(i) )    ->Θ(n)

endloop1


for j=0 to arrlist2.size()        ->Θ(n)

   arrlist3.add( i, arrlist2.get(j) )   ->Θ(n)

   i=i+1   ->Θ(1)

endloop2
```

```
for i=0 to arrlist3.size()-1      ->Θ(n)
  for j=i+1 to arrlist3.size()   ->Θ(n)
        if arrlist3.get(i) > arrlist3.get(j)   ->Θ(1)
            temp= arrlist3.get(i)   ->Θ(1)
            arrlist3.set( i , arrlist3.get(j) )   ->Θ(1)
            arrlist3.set( j, temp )   ->Θ(1)
          endif
   endloop3
endloop4
output arrlist3 (return)   ->Θ(1)
```

We can assume that arrlist1.size() is equal to n. This information is given in the question.

loop1 running time $T_1(n) = \Theta(n) . \Theta(n)$

$= \Theta(n^2)$

We can assume that arrlist2.size() is equal to n. This information is given in the question.

loop2 running time $T_2(n) = \Theta(n) . \Theta(n)$

$= \Theta(n^2)$

If statement running time $T_3(n) = \Theta(1). \Theta(1) = \Theta(1)$

loop3 running time $T_4(n) = \Theta(n). T_3(n) = \Theta(n). \Theta(1) = \Theta(n)$

loop4 running time $T_5(n) = \Theta(n). T_4(n) = \Theta(n). \Theta(n) = \Theta(n^2)$

Running time of code $T(n) = T_5(n) + T_1(n) + T_2(n) + \Theta(1) + \Theta(1)$

$= \Theta(n^2) + \Theta(n^2) + \Theta(n^2) + \Theta(1) + \Theta(1)$

$= \Theta(n^2)$

5-

a)

```
int p_1 (int array[]):
{
return array[0] * array[2])   -> Θ(1)
}
```

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

$S(n) = O(1)$

b)

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

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

$S(n) = O(1)$

c)

```
void p_3 (int array[], int n):
{
for (int i = 0; i < n; i++)   -> Θ(n)
      for (int j = 0; j < i; j=j*2)   -> Θ( log₂n )
            printf("%d", array[i] * array[j])   -> Θ(1)
}
```

$T(n) = \Theta(n) \cdot \Theta(\log_2 n) = \Theta(n.\log_2 n)$

$S(n) = O(1)$

d)

```
void p_4 (int array[], int n):
{
If (p_2(array, n)) > 1000)   -> T₃ (n)
            p_3(array, n)   -> T₁ (n)
```

else

    printf(“%d”, p_1(array) * p_2(array, n))   -> $T_2(n)$

}

$T_W(n) = T_3(n) + \max(T_1(n), T_2(n))$

$= \Theta(n) + \max(\Theta(n.\log_2 n), \Theta(1))$

$= \Theta(n) + \Theta(n.\log_2 n)$

$= \Theta(n.\log_2 n)$

$T_b(n) = T_3(n) + \min(T_1(n), T_2(n))$

$= \Theta(n) + \min(\Theta(n.\log_2 n), \Theta(1))$

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

$= \Theta(n)$

$T(n) = O(n.\log_2 n)$

$= \Omega(n)$

$S(n) = O(1)$

All space complexities are O(1).

Reasons

1- These functions contains only primitives and primitives’ size are constant.

2- These functions do not have array declaration in the function body part. Array declaration in the function body means that size allocation.

3- These functions have integer array parameter but space complexity does not include parameter list of function.