Part 1-1.)

```
public int getIndexbyProduct(Product productObj,Branch brancjObj)
{
        int IndexbyProduct;
Θ(1)    if(productObj.getProductType() == "Chair" || productObj.getProductType() == "OfficeDesk"
                || productObj.getProductType() == "MeetingTable" )
        {
   n        for(int i=0;i<brancjObj.getBranchManagement().getProductTypeSize();i++) {      n
                if( brancjObj.getBranchManagement().products[i].
                        getProductType().equals(productObj.getProductType()) &&      all equals
Θ(1)                    brancjObj.getBranchManagement().products[i].                   method
                        getcolor().equals(productObj.getcolor()) &&
O(n) or Ω(1)            brancjObj.getBranchManagement().products[i].
                        getmodel().equals(productObj.getmodel()))
                {
Θ(1)                    IndexbyProduct = I;
                        return IndexbyProduct;
                }

            }
        }
        else
        {
   n        for(int i=0;i<brancjObj.getBranchManagement().getProductTypeSize();i++) {      n
                if( brancjObj.getBranchManagement().products[i].
                        getProductType().equals(productObj.getProductType()) &&      Θ(1)
                        brancjObj.getBranchManagement().products[i].
                        getmodel().equals(productObj.getmodel()))
                {
                        IndexbyProduct = I;
                        return IndexbyProduct;      Θ(1)
                }

            }
        }
        return 0;   Θ(1)
}
```

Best cases $\Omega(1)$

Worst cases $O(n)$

it can find it the first time or navigate the loop completely.

$T(n) = O(n)$ or $\Omega(1)$

Note: constant time are not important

Part-1-2.

```
public void addProduct(Product  productObj,Branch brancjObj,int stockSize)
{
        brancjObj.getBranchManagement().products[getIndexbyProduct(productObj, brancjObj)].
       setstock(brancjObj.getBranchManagement().products[getIndexbyProduct(productObj,brancjObj)].
              getstock()+stockSize) ;
       System.out.println("PRODUCT ADD SUCCESSFUL");
}
```

$O(n)$ or $\Omega(1)$

$O(n)$ or $\Omega(1)$

Best cases $\Omega(2) \Rightarrow \Omega(1)$

Worst cases $O(2n) \Rightarrow O(n)$

$T(n) = O(n)$ or $\Omega(1)$

Note

Constant is unimportant

```
public void removeProduct(Product  productObj,Branch branchObj,int stockSize)
{
        brancjObj.getBranchManagement().products[getIndexbyProduct(productObj, branchObj)].
       setstock(branchObj.getBranchManagement().products[getIndexbyProduct(productObj,branchObj)].
              getstock()-stockSize) ;
       System.out.println("PRODUCT ADD SUCCESSFUL");
}
```

$O(n)$ or $\Omega(1)$

$O(n)$ or $\Omega(1)$

Best cases $\Omega(2)$

Worst cases $O(2n)$

$T(n) = O(n)$ or $\Omega(1)$

CamScanner ile tarandı

Part-1-3.

```
public void productSupplied()
{
boolean control = true;
for(int i=0; i<getbranchSize();i++)
{
        if(dataBase.branch[i].getwarningStock())
        {
                for(int k=0;k<dataBase.branch[i].requiredProductCounter;k++)
                {
                        for(int j=0;j<dataBase.branch[i].getProductTypeSize();j++)
                        {
                                if(dataBase.branch[i].requiredProduct[k].getcolor() !=null) {
                                        if(dataBase.branch[i].products[j].getmodel().
                                        equals(dataBase.branch[i].requiredProduct[k].getmodel()) &&
                                        dataBase.branch[i].products[j].getcolor().
                                        equals(dataBase.branch[i].requiredProduct[k].getcolor()) &&
                                        dataBase.branch[i].products[j].getProductType().
                                        equals(dataBase.branch[i].requiredProduct[k].getProductType()))
                                        {
                                        dataBase.branch[i].products[j].setstock(5);
                                        dataBase.branch[i].setwarningStock(false);
                                        System.out.println("The product has been successfully supplied.");
                                        [redacted]
                                        }
                                }
                                else {
                                        if(dataBase.branch[i].products[j].getmodel().
                                        equals(dataBase.branch[i].requiredProduct[k].getmodel())&&
                                        dataBase.branch[i].products[j].getProductType().
                                        equals(dataBase.branch[i].requiredProduct[k].getProductType()))
                                        {
                                        dataBase.branch[i].products[j].setstock(5);
                                        dataBase.branch[i].setwarningStock(false);
                                        System.out.println("The product has been    successfullysupplied.");
                                        [redacted]
                                        }

                                }
                                control =false;
                        }
                }
        }
}
if(control)
        System.out.println("There is no product to be supplied.");
}
```

$\Theta(a)$ — a

b — b

c — c

$\Theta(1)$

constant time $\Theta(1)$

$\Theta(b.c) = \Theta(b.c)$

$\Theta(1)$

$T_B(a,b,c) = \Theta(a)$

$T_W(a,b,c) = \Theta(a.b.c)$

$T(a,b,c) = O(a.b.c)$

it necesorry to go around the branches one by one and check if there is any warning. if there is no warning, it is the best possibility. if there is a warning it is necesorry to examine details. This is the worst possibility.

CamScanner ile tarandı

Part 2

a) The Big O notation is a notation that defines the upper bound, so it would be meaningless to say the least because the function we are comparing can be at most equal to this.

b) The sum of the functions $f(n)$ and $g(n)$ is approximately equal to any one of them being the maximum, because we don't care about the lower terms and constant values in the result.

ex; $f(n) = n+2$

$g(n) = n^2$

$max(f(n), g(n)) \Rightarrow n^2$

$\Theta(f(n)+g(n)) \Rightarrow \Theta(n^2+n+2) = \Theta(n^2)$

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

c) 1.) $\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = 2 =$ constant $\Rightarrow$ so $2^{n+1} = \Theta(2^n)$

2.) $\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \frac{4^n}{2^n} = \infty$ The growth rate of 2^{2n} is faster than 2^n

so $2^{2n} = \Theta(2^n)$ is wrong.

$2^n = O(2^{2n})$ is true.

3.) We can not say that this statement is absolutely correct.

$g(n) = \Theta(n^2)$ according to these statement $f(n) = \Theta(n^2)$ must be.

$f(n) * g(n) = \Theta(n^4)$ $f(n)$ is a max of n^2 in $f(n) = O(n^2)$.

but $f(n)$ can be lower term.

in this case this statement would be false.

Part-3

Okan Torun
1801042662

$2^n, 2^{n+1}, n.2^n, 3^n, 5^{\log_2 n} >$ others

$\underbrace{\hphantom{2^n, 2^{n+1}, n.2^n, 3^n, 5^{\log_2 n}}}_{}$

exponantial

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = o(g(n))$$
$$= c \neq 0 \Rightarrow f(n) = \Theta(g(n))$$
$$= \infty \Rightarrow g(n) = o(f(n))$$

$$\lim_{n\to\infty} \frac{2^n}{5^{\log_2 n}} = \infty \qquad 5^{\log_2 n} = o(2^n)$$

$$\lim_{n\to\infty} \frac{2^n}{2^{n+1}} = \frac{1}{2} \qquad 2^n = \Theta(2^{n+1})$$

$$\lim_{n\to\infty} \frac{2^n}{n.2^n} = 0 \qquad 2^n = o(2^n.n)$$

$$\lim_{n\to\infty} \frac{3^n}{n.2^n} = \infty \qquad n.2^n = o(3^n)$$

$$\lim_{n\to\infty} \frac{n^{1.01}}{n.\log^2 n} = \infty \qquad n.\log^2 n = o(n^{1.01})$$

$$\lim_{n\to\infty} \frac{n.\log^2 n}{\sqrt{n}} = \infty \qquad \sqrt{n} = o(n.\log^2 n)$$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log^3 n} = \infty \qquad \log^3 n = o(\sqrt{n})$$

$$\lim_{n\to\infty} \frac{\log^3 n}{\log n} = \infty \qquad \log n = o(\log^3 n)$$

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

Part - 4 - a

```
find_min_Value(ArrayList<Integer>arr, int n)
{
    int min;
    min= arr.get(0);              -> θ(1)
    for(int i=0; i<n ; i++)       -> n
    {
        if(arr.get(i)<min)        -> θ(1)
            min= arr.get(i);      -> θ(1)     θ(1)
    }
    return min;                   -> θ(1)
}
```

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

best case

&

Worst case

Part 4-b

```
int find_median(ArrayList<Integer> arr, int n)
{
    int temp;
    int median;
    ArrayList<Integer> copyArr = new ArrayList<>(n)

    for(int i=0; i<n; i++)
    {
        copyArr.add(arr.get(i));   } average O(1)
    }                              } Θ(n)
    for(int i=0; i<n-1; i++)
    {
        for(int j=0; j<n-i-1; j++)
        {
            if(copyArr.get(j) > copyArr.get(j+1))
            {
                temp = copyArr.get(j);
                copyArr.set(j, copyArr.get(j+1));
                copyArr.set(j+1, temp);        } Θ(1)
            }
        }
    }

    if(n%2==1)
        median = copyArr.get((n+1)/2-1);   } Θ(1)
    else
        median = copyArr.get(n/2);         } Θ(1)

    for(int i=0; i<n; i++)
    {
        if(arr.get(i) == median)   } Θ(1)
            return arr.get(i);
    }                              Θ(n/2)
    return 0;
}
```

$$\sum_{i=0}^{n-1} n-i-1 = (n-1)+(n-2)+ \dots +2+1 = \frac{n(n+1)}{2} = \Theta(n^2)$$

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

$T(n) = \Theta(n^2 + \frac{3n}{2} + 2)$

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

Worst case = Base case

CamScanner ile tarandı

Part 4-c

```
sumTwoElements(ArrayList<Integer>arr, int arr_size, int sum)
{
    for(int i=0; i<arr_size; i++)              -> T_2
    {
        for(int j=0; j<arr_size; j++)          -> T_1
        {
            if(i!=j)
            {
                if(arr.get(i)+arr.get(j)==sum)     Θ(1)
                {
                    return sum;                    -> Θ(1)
                }
            }
        }
    }
    return false;
}
```

n (over arr_size)

$T_{1B} = \Theta(1)$

$T_{1W} = \Theta(n)$

$T_{2B} = \Theta(1)$

$T_{2W} = \Theta(n)$

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

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

$T_B(n) = \Theta(1).\Theta(1)$

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

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

Part 4-d

```
boolean merge(ArrayList<Integer>arr1,ArrayList<Integer>arr2,ArrayList<Integer>arr3, int arr_size)
  {
        int arr1_count=0;
        int arr2_count=0;
        int arr3_count=0;

        arr3= new ArrayList<>(2*arr_size);   ) 1

        while(arr1_count<arr_size && arr2_count <arr_size)
        {
              if( arr1.get(arr1_count) < arr2.get(arr2_count))   ) Θ(1)
              {
                    arr3.add(arr1.get(arr1_count));   ) Θ(1)
                    arr3_count++;
                    arr1_count++;   ) Θ(1)
              }
              else if(((arr2.get(arr2_count) < arr1.get(arr1_count)) ||
              (arr2.get(arr2_count) == arr1.get(arr1_count))))   ) Θ(1)
              {
                    arr3.add(arr2.get(arr2_count));   ) Θ(1)
                    arr3_count++;
                    arr2_count++;   ) Θ(1)
              }

        }

        while(arr1_count<arr_size)   O(n) or Ω(1)
        {
              arr3.add(arr1.get(arr1_count));   ) Θ(1)
              arr3_count++;
              arr1_count++;   ) Θ(1)
        }
        while(arr2_count<arr_size)   ) O(n) or Ω(1)
        {
              arr3.add(arr2.get(arr2_count));
              arr3_count++;
              arr2_count++;
        }

        return true;   Θ(1)
  }
```

Average

$T_W = \Theta(n) + \Theta(n-1)$
$= \Theta(2n-1)$

$T_B = \Theta(n)$

if the above loop happens to $\Theta(2n-1)$, the following loops become $\Theta(1)$. if the above loop happens to $\Theta(n)$, the following loops become $O(n)$.

$T_W = T_B = \Theta(2n)$

$T(n) = O(n)$

CamScanner ile tarandı

Part 5:

Analyze the time complexity and space complexity of the following code segments:

a)

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

Space complexity $O(1)$ because we don't create a new place in memory

```
int p_1 (int array[]):
{
    return array[0] * array[2])
}
```

$\Theta(1)$

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
}
```

$\text{i=i+5} \rightarrow \frac{n}{5}$

$T_b = T_w = \Theta\left(\frac{n}{5}\right) = \Theta(n)$

sum += array[i] * array[i]) $\Theta(1)$

return sum $\Theta(1)$

Space comp. $O(1)$

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])
}
```

$\text{j=j*2} \rightarrow$ multiply 2

for (int j ...) $\Theta(\log i)$

printf(...) $\Theta(1)$

$\underbrace{\log 1 + \log 2 + \dots \log n}_{\Theta(n.\log n) = T(n)}$

$T_1 = T_2$

Space comp. $O(1)$

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))
}
```

Space comp. $O(1)$

If (p_2(array, n)) > 1000) $\rightarrow T_1 = \Theta(n)$

p_3(array, n) $\rightarrow T_2 = \Theta(n.\log n)$

p_1(array) $\rightarrow \Theta(1)$, p_2(array, n) $\rightarrow \Theta(n)$

$T_3 = \Theta(n)$

$T_B = T_1 + T_3$

$T_B(n) = \Theta(2n)$

$T_W = T_1 + T_2$

$T_W(n) = \Theta(n + n.\log n)$

$T_W(n) = \Theta(n.\log n)$

$\boxed{T(n) = O(n.\log n)}$

CamScanner ile tarandı