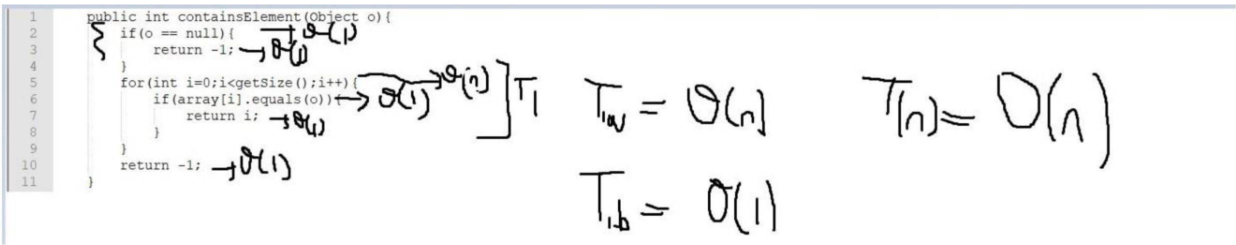
# GTU Department of Computer Engineering
# CSE 222/505 - Spring 2021
# Homework 2
## Due date: March 25 2021, 9:30 AM

**Part 1:**

Analyze the time complexity (in most appropriate asymptotic notation) of the following procedures by your solutions for the Homework 1:

I. Searching a product.

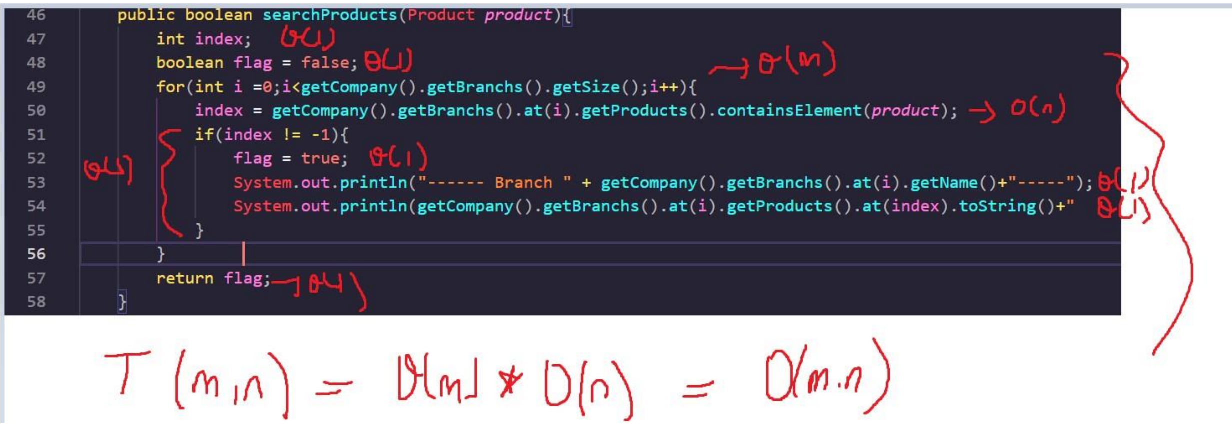


```
public int containsElement(Object o){
    if(o == null){
        return -1;
    }
    for(int i=0;i<getSize();i++){
        if(array[i].equals(o)){
            return i;
        }
    }
    return -1;
}
```

$\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(n)$ $\theta(1)$ $\theta(1)$ $T_1$

$T_{w} = \theta(n)$ $T(n) = O(n)$

$T_{b} = \theta(1)$



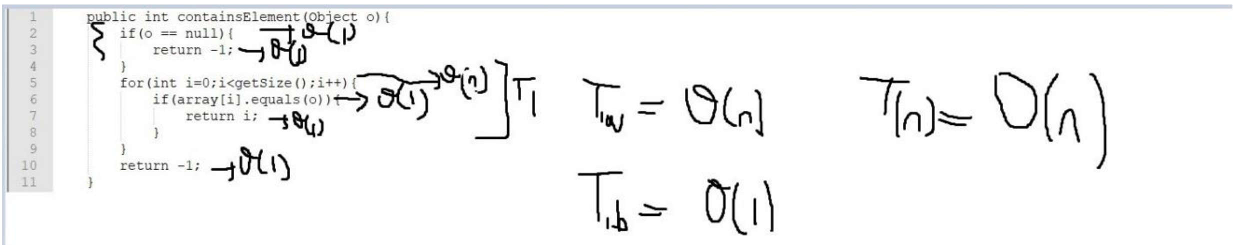
```
public boolean searchProducts(Product product){
    int index;
    boolean flag = false;
    for(int i =0;i<getCompany().getBranchs().getSize();i++){
        index = getCompany().getBranchs().at(i).getProducts().containsElement(product);
        if(index != -1){
            flag = true;
            System.out.println("------ Branch " + getCompany().getBranchs().at(i).getName()+"-----");
            System.out.println(getCompany().getBranchs().at(i).getProducts().at(index).toString()+"
        }
    }
    return flag;
}
```

$\theta(1)$ $\theta(1)$ $\theta(m)$ $O(n)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$

$T(m,n) = \theta(m) * O(n) = O(m.n)$

II. Add/remove product.

-Add Product-

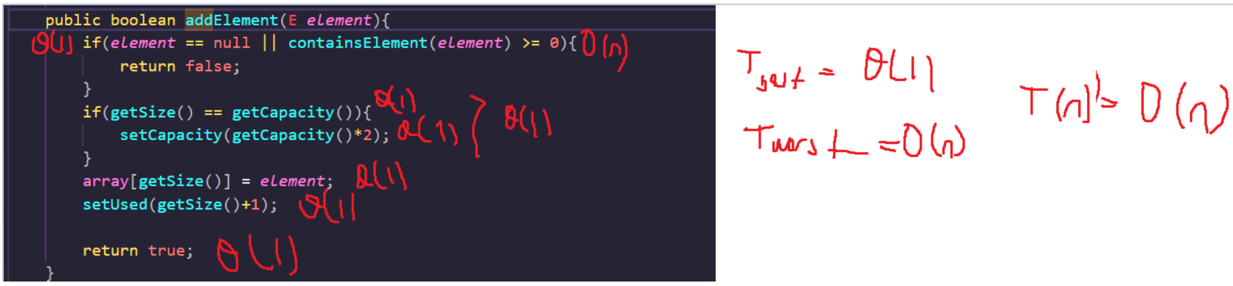


```
public int containsElement(Object o){
    if(o == null){
        return -1;
    }
    for(int i=0;i<getSize();i++){
        if(array[i].equals(o)){
            return i;
        }
    }
    return -1;
}
```

$\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(n)$ $\theta(1)$ $\theta(1)$ $T_1$

$T_{w} = \theta(n)$ $T(n) = O(n)$

$T_{b} = \theta(1)$



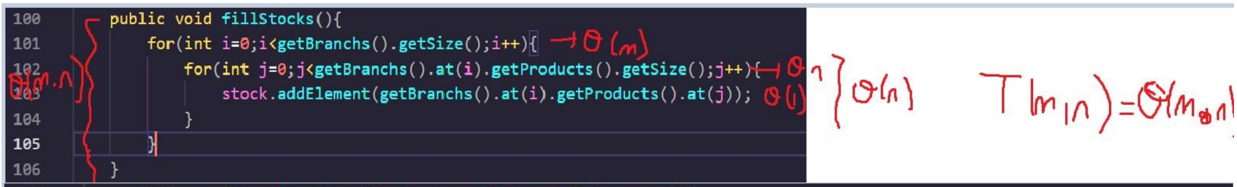
```
public boolean addElement(E element){
    if(element == null || containsElement(element) >= 0){
        return false;
    }
    if(getSize() == getCapacity()){
        setCapacity(getCapacity()*2);
    }
    array[getSize()] = element;
    setUsed(getSize()+1);

    return true;
}
```

$\theta(1)$ $O(n)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$

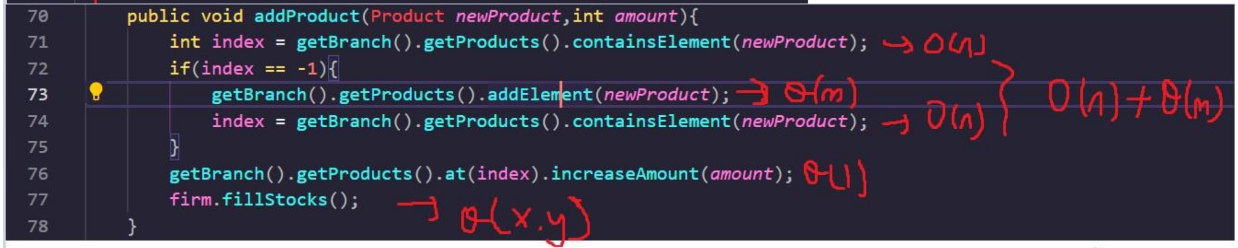
$T_{best} = \theta(1)$

$T_{worst} = O(n)$

$T(n) = O(n)$



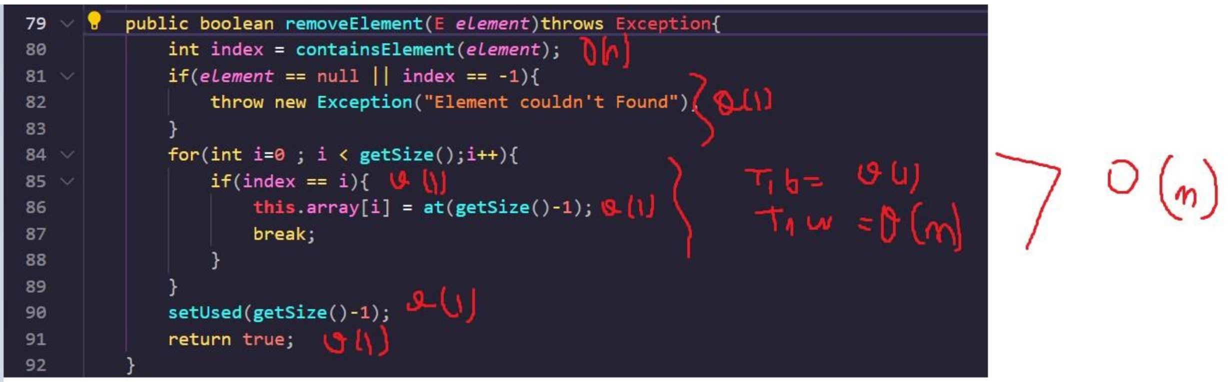
```java
public void fillStocks(){
    for(int i=0;i<getBranchs().getSize();i++){
        for(int j=0;j<getBranchs().at(i).getProducts().getSize();j++){
            stock.addElement(getBranchs().at(i).getProducts().at(j));
        }
    }
}
```



```java
public void addProduct(Product newProduct,int amount){
    int index = getBranch().getProducts().containsElement(newProduct);
    if(index == -1){
        getBranch().getProducts().addElement(newProduct);
        index = getBranch().getProducts().containsElement(newProduct);
    }
    getBranch().getProducts().at(index).increaseAmount(amount);
    firm.fillStocks();
}
```

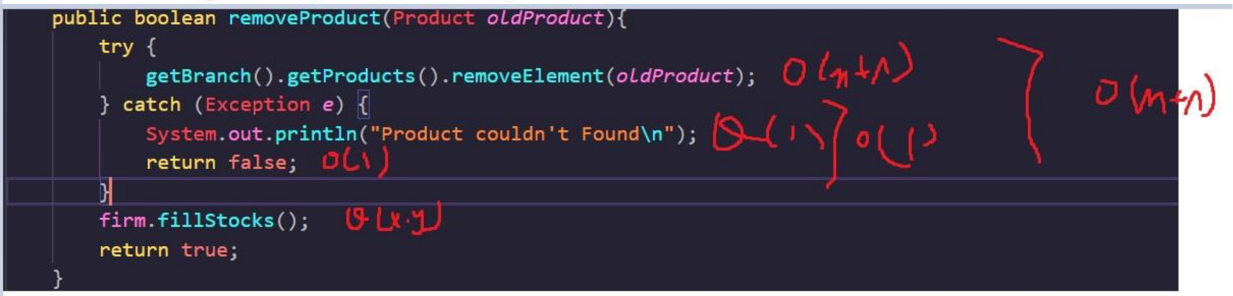
$$\Theta(x.y) + O(n) + \Theta(m) = O(x.y + m + n)$$

-Remove Product-



```java
public boolean removeElement(E element)throws Exception{
    int index = containsElement(element);
    if(element == null || index == -1){
        throw new Exception("Element couldn't Found");
    }
    for(int i=0 ; i < getSize();i++){
        if(index == i){
            this.array[i] = at(getSize()-1);
            break;
        }
    }
    setUsed(getSize()-1);
    return true;
}
```

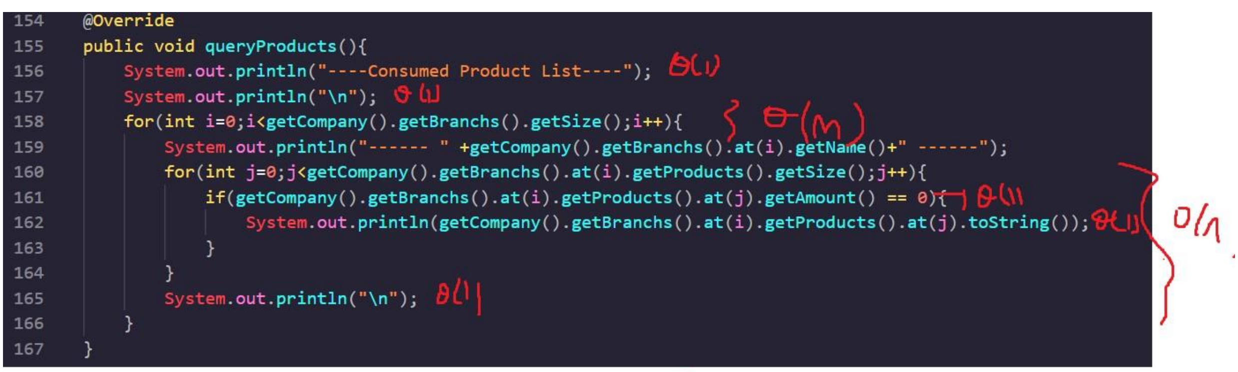
$$T(mn) = O(m+n)$$



```java
public boolean removeProduct(Product oldProduct){
    try {
        getBranch().getProducts().removeElement(oldProduct);
    } catch (Exception e) {
        System.out.println("Product couldn't Found\n");
        return false;
    }
    firm.fillStocks();
    return true;
}
```

$$\Theta(xy) + O(m+n) = O(m+n + xy)$$

III. Querying the products that need to be supplied.



```
@Override
public void queryProducts(){
    System.out.println("----Consumed Product List----");
    System.out.println("\n");
    for(int i=0;i<getCompany().getBranchs().getSize();i++){
        System.out.println("------ " +getCompany().getBranchs().at(i).getName()+" ------");
        for(int j=0;j<getCompany().getBranchs().at(i).getProducts().getSize();j++){
            if(getCompany().getBranchs().at(i).getProducts().at(j).getAmount() == 0){
                System.out.println(getCompany().getBranchs().at(i).getProducts().at(j).toString());
            }
        }
        System.out.println("\n");
    }
}
```

$$T(m,n) = O(m \cdot n)$$

Attach the code of your solution for each part just before its analysis.

**Part 2:**

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

O(n) represents the maximum time can take O(n). So, $O(n^2)$ actually represents the maximum time not minimum.

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)).

Let’s assume that $f(n) = \Theta(n^2)$ and $g(n) = \Theta(n)$, $\max(f(n),g(n)) = \Theta(n^2)$ and $\Theta(f(n) + g(n)) = \Theta(n^2+n) \rightarrow \Theta(n^2)$ so equation is proven.

c) Are the following true? Prove your answer.

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

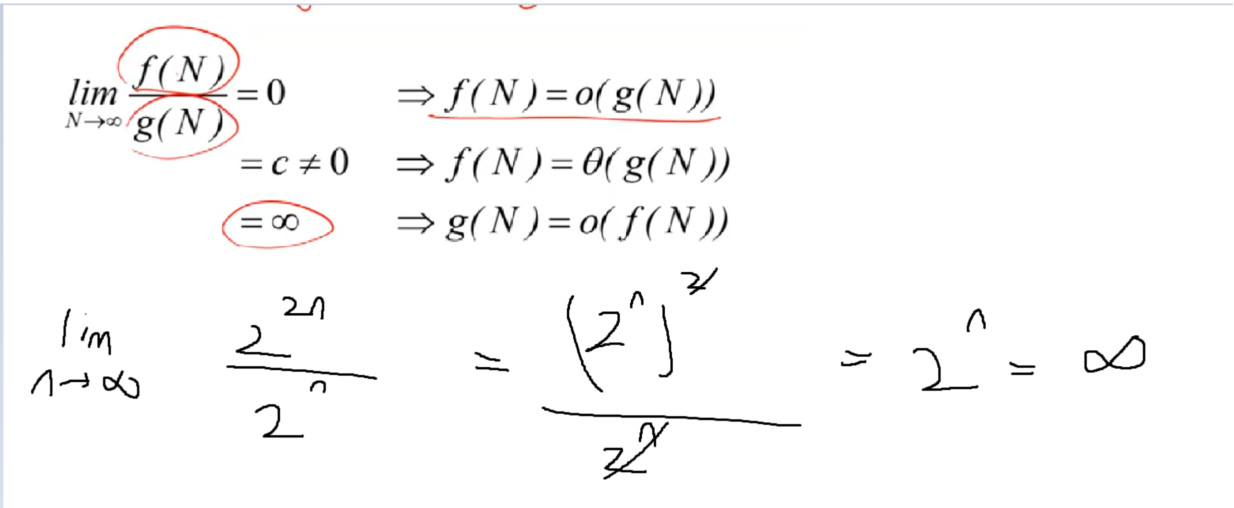
$$\begin{aligned} \lim_{N\to\infty}\frac{f(N)}{g(N)} &= 0 && \Rightarrow f(N)=o(g(N)) \\ &= c \neq 0 && \Rightarrow f(N)=\theta(g(N)) \\ &= \infty && \Rightarrow g(N)=o(f(N)) \end{aligned}$$

$$\lim_{n\to\infty}\frac{2^{n+1}}{2^n} = \frac{\cancel{2^n}\cdot 2^1}{\cancel{2^n}} \qquad c \neq 0$$

So this equation is true.

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

So it should be $O(2^n)$ not $\Theta(2^n)$. Equation is false.



$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \quad \Rightarrow f(N) = o(g(N))$$

$$= c \neq 0 \quad \Rightarrow f(N) = \theta(g(N))$$

$$= \infty \quad \Rightarrow g(N) = o(f(N))$$

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \frac{(2^n)^{\cancel{2}}}{\cancel{2^n}} = 2^n = \infty$$

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

$\Theta(n^2)$ is tight but $O(n^2)$ isn't tight. $O(n^2)$ it could be either constant or logarithmic. So result must not be tight. Equation is wrong.

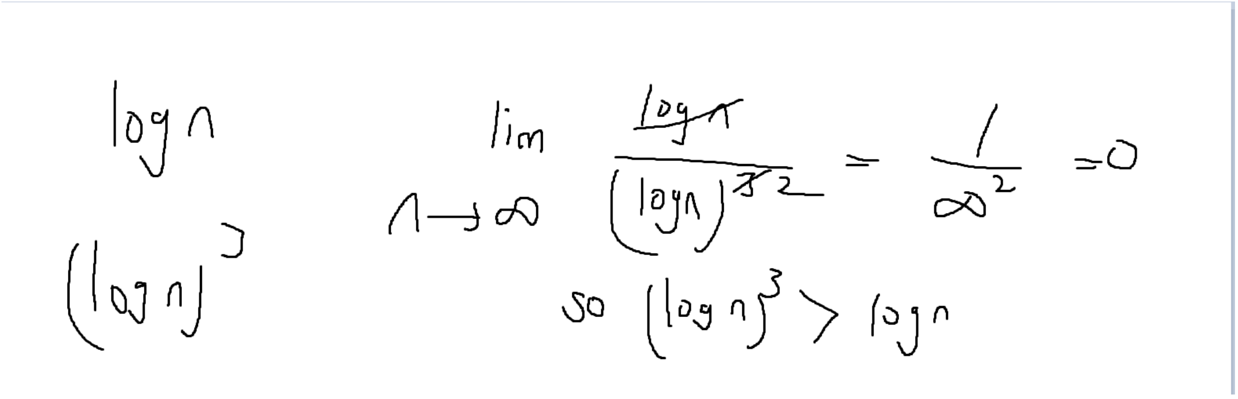
**Part 3:**

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

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

According to common grown rates:

Constant < Logarithmic < Linear < Log-Linear < Quadratic < Cubic < Exponential < Factorial



$\log n$

$(\log n)^3$

$$\lim_{n\to\infty} \frac{\cancel{\log n}}{(\log n)^{\cancel{3}\,2}} = \frac{1}{\infty^2} = 0$$

so $(\log n)^3 > \log n$

According to common grown rates ,$\sqrt{n} > (\log n)^3$

According to common grown rates ,$n\log^2 n > \sqrt{n}$

According to common grown rates, $n^{1.01}$ >$n\log^2 n$

According to common grown rates ,$5^{\log n}$ > $n\log^2 n$

According to common grown rates ,$2^n$ > $5^{\log n}$

$$\lim_{n \to \infty} \frac{2^{n+1}}{2^n} = \frac{\cancel{2^n} \cdot 2}{\cancel{2^n}} = \text{so they are equal}$$

$2^n = 2^{n+1}$

$$\lim_{n \to \infty} \frac{n \cdot \cancel{2^n}}{\cancel{2^n}} = \infty \quad \text{so } n \cdot 2^n > 2^n$$

$n2^n$ > $2^n$

$$\lim_{n \to \infty} \frac{3^n}{n \cdot 2^n} = \frac{\left(\frac{3}{2}\right)^m}{n} \quad \text{exponential} > \text{linear} \quad \text{so } 3^n > n \cdot 2^n$$

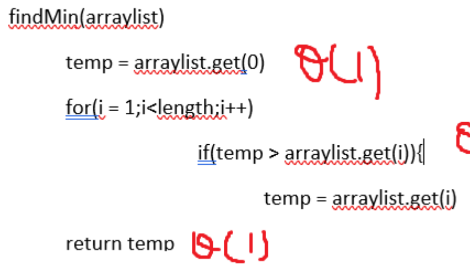
$3^n > n2^n$

So, $\log n < (\log n)^3 < \sqrt{n} < n\log^2 n < n^{1.01} < 5^{\log n} < 2^n = 2^{n+1} < n2^n < 3^n$

**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.



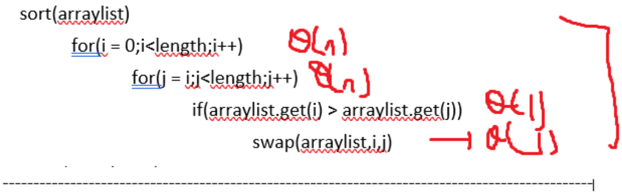
```
findMin(arraylist)
    temp = arraylist.get(0)                 θ(1)
    for(i = 1;i<length;i++)
            if(temp > arraylist.get(i)){    θ(1)
                temp = arraylist.get(i)     θ(1)
    return temp                             θ(1)
```

$\theta(n)$

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

-

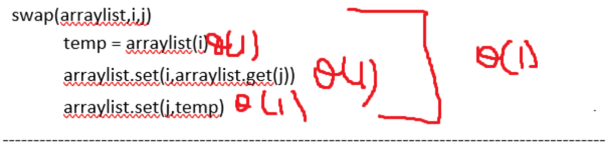
- Find the median item. Consider each element one by one and check whether it is the median.



```
sort(arraylist)
    for(i = 0;i<length;i++)                          θ(n)
        for(j = i;j<length;j++)                      θ(n)
            if(arraylist.get(i) > arraylist.get(j))  θ(1)
                swap(arraylist,i,j)              →   θ(1)
```

$\theta(n^2)$

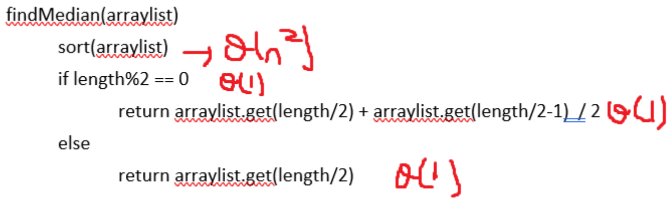
---



```
swap(arraylist,i,j)
    temp = arraylist(i)                   θ(1)
    arraylist.set(i,arraylist.get(j))     θ(1)
    arraylist.set(j,temp)                 θ(1)
```

$\theta(1)$

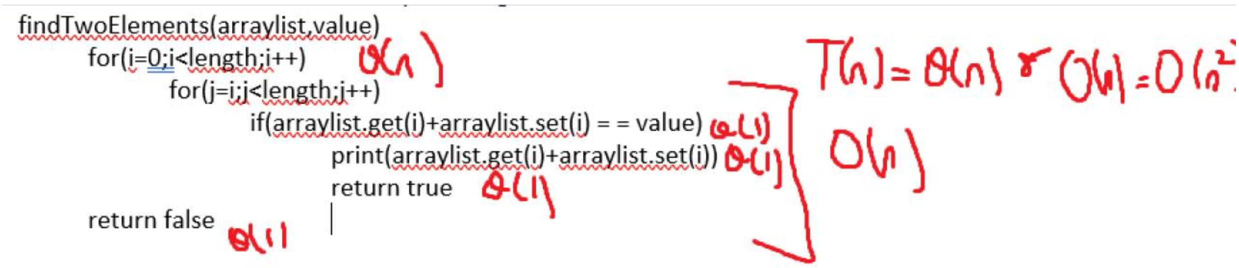
---



```
findMedian(arraylist)
    sort(arraylist)     →  θ(n²)
    if length%2 == 0       θ(1)
        return arraylist.get(length/2) + arraylist.get(length/2-1) / 2   θ(1)
    else
        return arraylist.get(length/2)     θ(1)
```

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

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

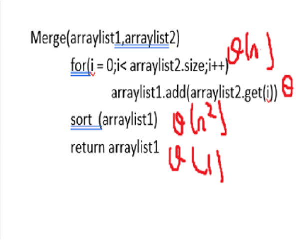


```
findTwoElements(arraylist,value)
    for(i=0;i<length;i++)                                       θ(n)
        for(j=i;j<length;j++)
            if(arraylist.get(i)+arraylist.set(i) = = value)     θ(1)
                print(arraylist.get(i)+arraylist.set(i))        θ(1)
                return true                                     θ(1)
    return false                                                θ(1)
```

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

$O(n)$

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



```
Merge(arraylist1,arraylist2)
    for(i = 0;i< arraylist2.size;i++)
        arraylist1.add(arraylist2.get(i))
    sort (arraylist1)
    return arraylist1
```

$\theta(n)$ $\theta(1)$ $\theta(n)$

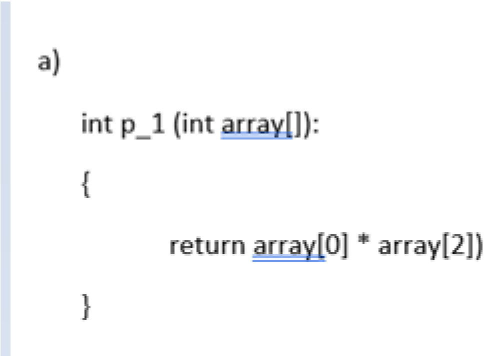
$\theta(n^2)$

$\theta(1)$

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

**Part 5:**

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



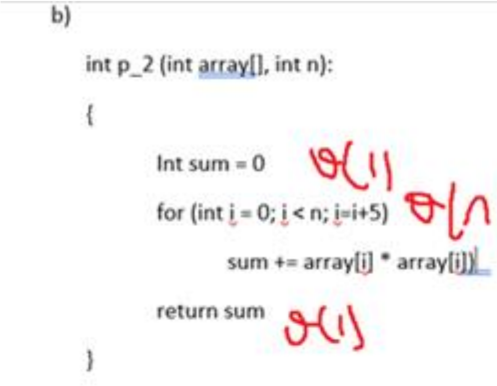
a)

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

$\rightarrow \theta(1)$

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

$S(n) = O(1)$



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
}
```

$\theta(1)$

$\theta(n/5)$

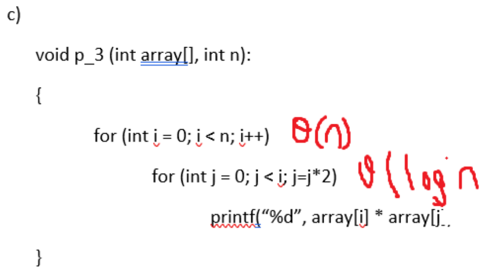
$\theta(1)$

$\theta(1)$

$\theta(n/5)$

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

$S(n) = O(1)$



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..
}
```

$\theta(n)$

$\theta(\log n)$

$\theta(1)$

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

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))
}
```

$\theta(n)$

$\theta(n.\log n)$

$\theta(1)$

$$\left.\begin{array}{l} T_{best} = \theta(1) \\ T_{worst} = \theta(n.\log n) \end{array}\right\} T_n = O(n.\log n)$$

**RESTRICTIONS:**

- Answer in detail the questions by using asymptotic notations.
- Yes / no answers and plagiarisation from the web will not be accepted.

**GENERAL RULES:**

- For any question firstly use course news forum in Moodle, and then the contact TA.
- You can submit assignment one day late and will be evaluated over sixty percent (%60).

**REPORT RULES:**

- All the analysis must be stated in the report/answer sheet in details.
- The report may be handwritten (only for this homework) if you want but, it must be scanned well and submitted to Moodle.

**GRADING :**

- **Part 1: 20 pts**
- **Part 2: 10 pts**
- **Part 3: 25 pts**
- **Part 4: 30 pts**
- **Part 5: 15 pts**
- Disobey restrictions: -100
- **Cheating: -200**
- Your assignment is evaluated over 100 as your performance.

**CONTACT :**

- Teaching Assistant : Mehmet Burak Koca
- b.koca@gtu.edu.tr