Barış Ayyıldız
1901042252

Part 1 :

I. Searching a product.

Time complexity is : **Θ(n*m*l)**

n is length of stocks array,

m is length of furnitures array,

l is length of String query

II. Add/remove product.

Add Products

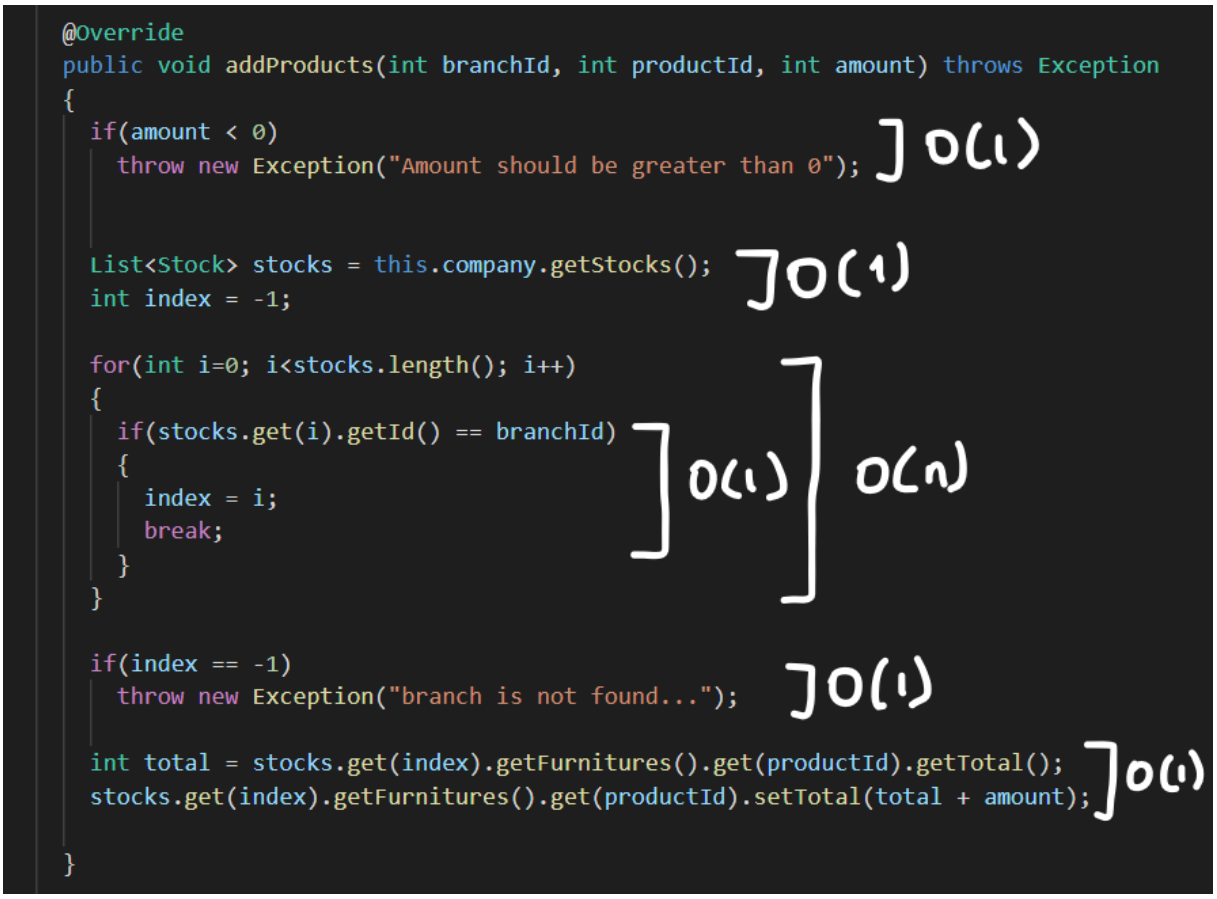
O(1)+ O(1)+ O(n)+ O(1) = **O(n)**

Since there is a breaking condition, the worst case is linear

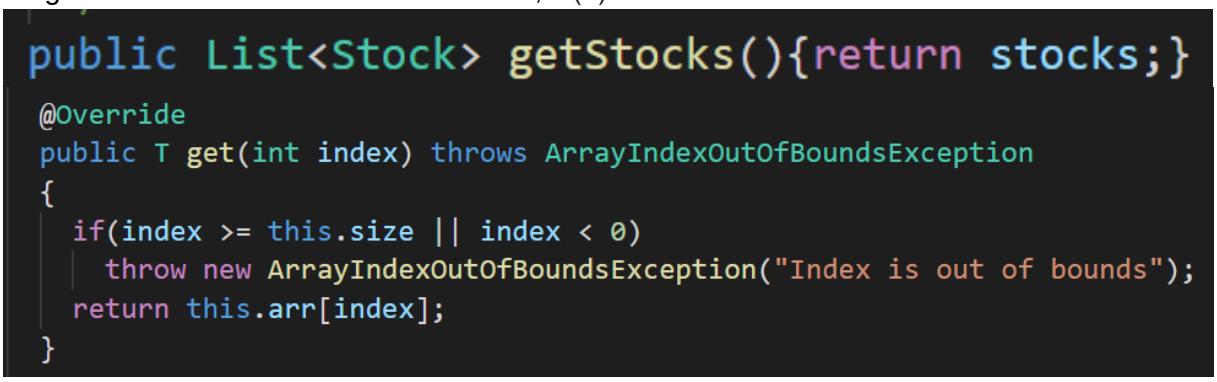
Tworst = **O(n)**

Tbest = **Ω(1)**, best case is constant time, when the algorithm iterates once inside for loop

n is equal to length of stock array in this case



All getters and setters take constant time, Θ(1)



Remove Products

Since there is a breaking condition, the worst case is linear

Tworst = **O(n)**

Tbest = **Ω(1)**, best case is constant time, when the algorithm iterates once inside the for loop

n is equal to length of stock array in this case

```java
@Override
public void removeProducts(int branchId, int productId, int amount) throws Exception
{
  if(amount < 0)
    throw new Exception("Amount should be greater than 0");

  List<Stock> stocks = this.company.getStocks();
  int index = -1;

  for(int i=0; i<stocks.length(); i++)
  {
    if(stocks.get(i).getId() == branchId)
    {
      index = i;
      break;
    }
  }

  if(index == -1)
    throw new Exception("branch is not found...");

  int total = stocks.get(index).getFurnitures().get(productId).getTotal();

  if(amount > total)
    throw new Exception("Not enough products...");

  stocks.get(index).getFurnitures().get(productId).setTotal(total - amount);

}
```


III. Querying the products that need to be supplied.

n is length of stocks, m is length of furnitures array

Since there is no breaking condition inside nested for loop

Θ(1) + Θ(n*m)+ Θ(1) = **Θ(n*m)**

```java
@Override
public void productsNeedToBeSupplied()
{
  List<Stock> stocks = this.company.getStocks();

  String str = "ID\tModelId\tType\tColor\t\tBranchId\t\tAmount\n";

  for(int i=0; i<stocks.length(); i++)
  {
    for(int j=0; j<stocks.get(i).getFurnitures().length(); j++)
    {
      int total = stocks.get(i).getFurnitures().get(j).getTotal();
      if(total == 0)
        str += stocks.get(i).getFurnitures().get(j).toString() + "\t\t" + stocks.get(i).getFurnitures().get(i).getBranch().getBranchId() + "\t\t" + total + "\n";
    }
  }

  System.out.println(str);

}
```


All getters and setters take constant time, Θ(1)

```java
public void setTotal(int val){this.amount = val;}
```

```java
public List<Furniture> getFurnitures(){return furnArr;}
```

```java
public Branch getBranch(){return this.branch;}
```

```java
public int getBranchId(){return branchId;}
```

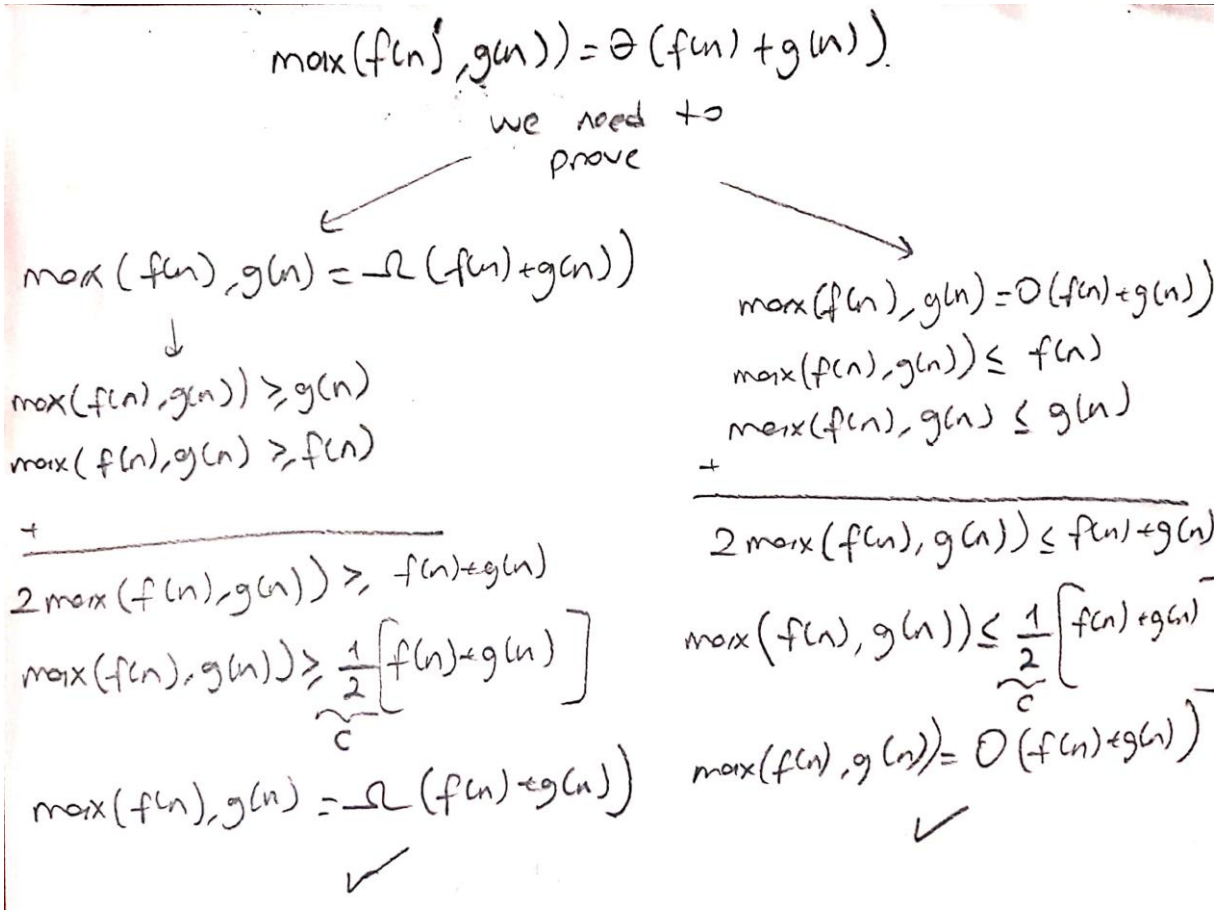
Part 2 :

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

Because by definition, Big O notation refers to the worst case of an algorithm. So it is meaningless to say ‘at least’.

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)).

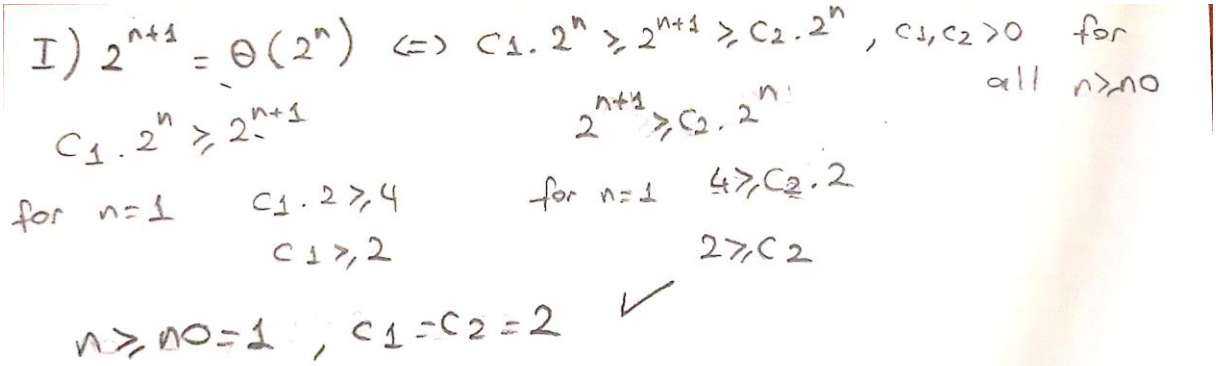
true



c) Are the following true? Prove your answer.

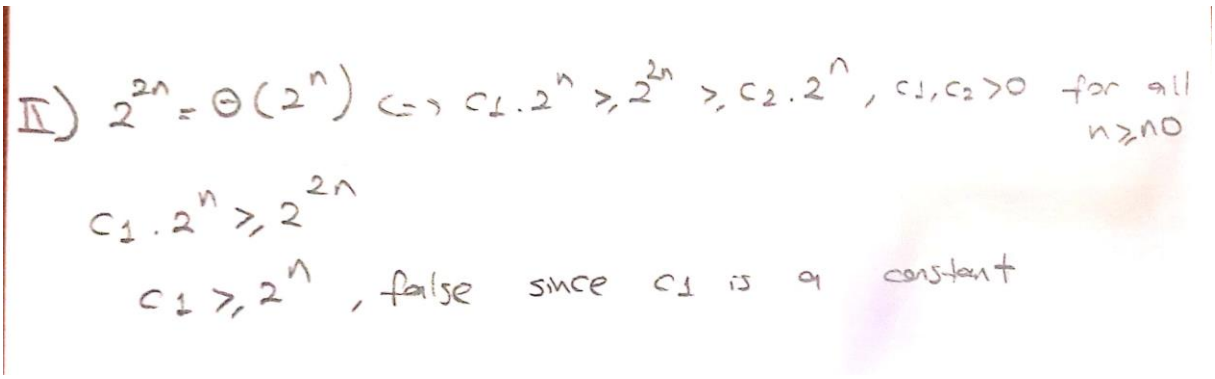
I. $2^{n+1} = \Theta(2^n)$

true

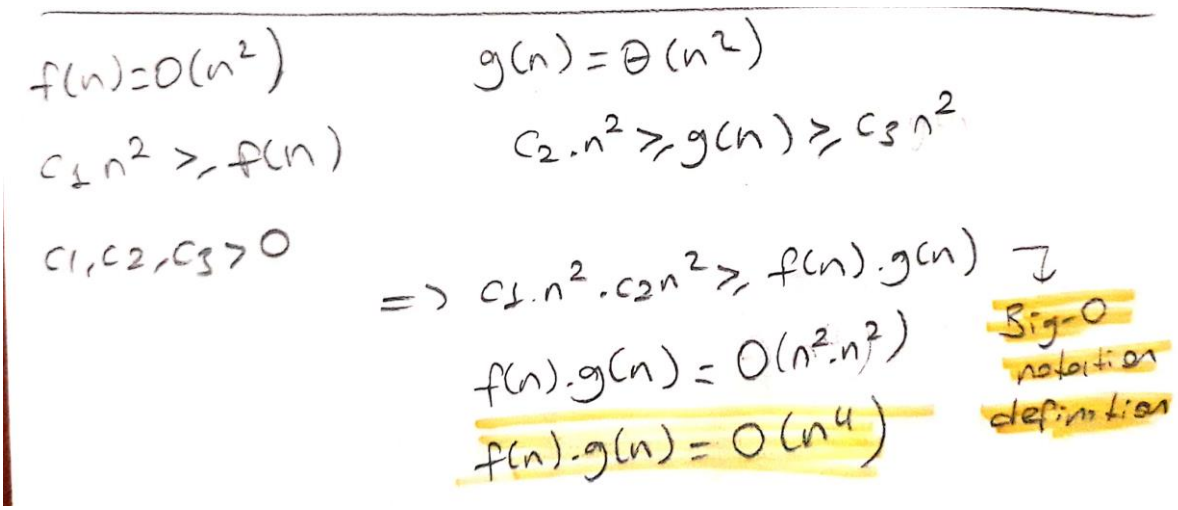


II. $2^{2n} = \Theta(2^n)$

false



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



Big-O notation doesn't tell anything about the lower bound. So for time complexity**, it is not guarenteed to be Θ(n^4)**. But we can say that:

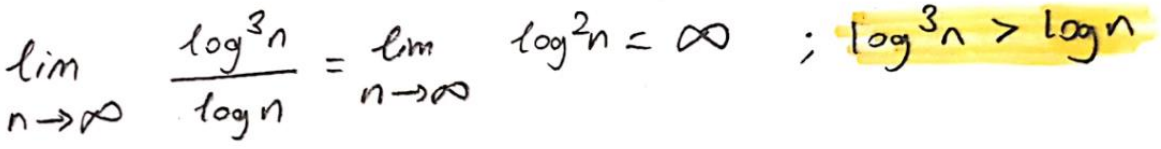
f(n) * g(n) = O(n^4)

Part 3 :

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

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

We can find growth rate by dividing two functions and taking it's limit



$$\lim_{n\to\infty} \frac{\sqrt{n}}{\log^3 n} \to \text{L'Hopital} \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{3\log^2 n}{n\cdot\ln 10}} = \lim_{n\to\infty} \frac{\ln(10)\sqrt{n}}{6\log^2 n}$$

$$= \lim_{n\to\infty} = \frac{\frac{\ln(10)}{2\sqrt{n}}}{\frac{12\log n}{\ln 10 \cdot n}} = \lim_{n\to\infty} \frac{\ln^2 10\sqrt{n}}{24\log n} = \lim_{n\to\infty} = \frac{\frac{\ln^2 10}{2\sqrt{n}}}{\frac{24}{\ln 10 \cdot n}}$$

$\to$ L'Hopital

$$= \lim_{n\to\infty} \frac{\ln^3 10\sqrt{n}}{48} = \infty \; ; \; \sqrt{n} > \log^3 n$$

$$\lim_{n\to\infty} \frac{n\log^2 n}{\sqrt{n}} = \lim_{n\to\infty} \sqrt{n}\cdot\log^2 n = \infty \; ; \; n\log^2 n > \sqrt{n}$$

$$\lim_{n\to\infty} \frac{n^{1.01}}{n\log^2 n} = \lim_{n\to\infty} \frac{n^{0.01}}{\log^2 n} = \lim_{n\to\infty} \frac{0.01 \cdot n^{-0.99}}{\frac{2\log n \cdot 1}{\ln 10 \cdot n}}$$

$\to$ L'Hopital

$$= \lim_{n\to\infty} \frac{0.005 \cdot n^{0.01} \cdot \ln 10}{\log n} = \lim_{n\to\infty} \frac{5\times10^{-4} n^{-0.99} \ln 10}{\frac{1}{\ln 10 \cdot n}}$$

$\to$ L'Hopital

$$= \lim_{n\to\infty} 5\times10^{-4} \ln^2 10 \cdot n^{0.01} = \infty \; ; \; n^{1.01} > n\log^2 n$$

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

; $5^{\log_2 n} > n^{1.01}$

$$\lim_{n\to\infty} \frac{2^n}{5^{\log_2 n}} = \lim_{n\to\infty} \frac{2^n}{n^{\log_2 5}} = \lim_{n\to\infty} \frac{2^n}{n^{2.32}} = \frac{2^n \cdot \ln 2}{2.32 \cdot n^{1.32}}$$

L'Hopital

$$= \lim_{n\to\infty} \frac{\ln^2 2 \cdot 2^n}{3.06 \cdot n^{0.32}} = \infty \; ; \; 2^n > 5^{\log_2 n}$$

L'Hopital

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \lim_{n\to\infty} \frac{2^n \cdot 2}{2^n} = 2$$

$$\lim_{n\to\infty} \frac{n \cdot 2^n}{2^{n+1}} = \lim_{n\to\infty} \frac{n}{2} = \infty \; ; \; n \cdot 2^n > 2^{n+1}$$

$$\lim_{n\to\infty} \frac{3^n}{2^n \cdot n} = \lim_{n\to\infty} \left(\frac{3}{2}\right)^n n = \lim_{n\to\infty} \frac{(1.5)^n}{n}$$

$$= \text{L'Hopital} \lim_{n\to\infty} \frac{1 \cdot (1.5)^n \cdot \ln(1.5)}{1} = \infty \; ; \; 3^n > n \cdot 2^n$$

Result :

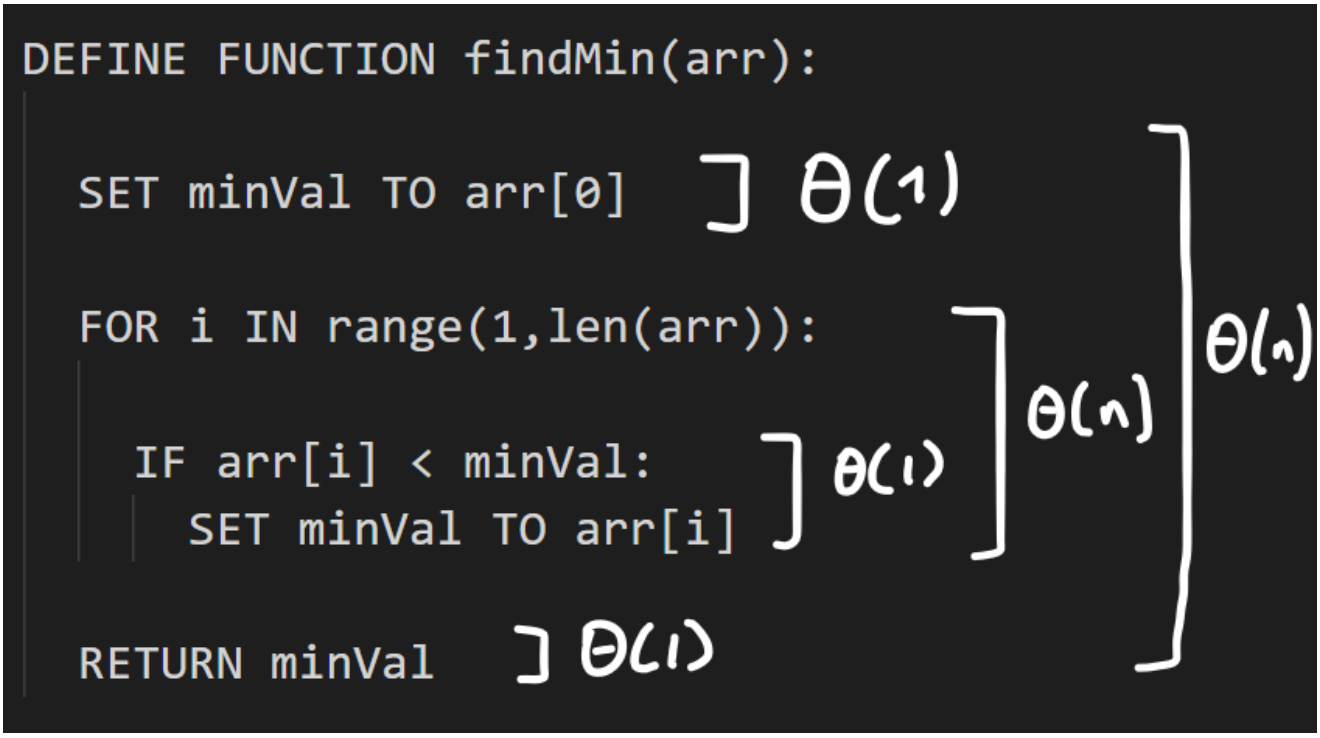
$3^n > n.2^n > 2^{n+1} = 2^n > 5^{\log_2 n} > n^{1.01} > n.\log^2 n > \sqrt{n} > \log^3 n > \log n$

Part 4 :

- Find the minimum-valued item.

It takes linear time

T(n) = **Θ(n)**



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

Bubble sort

```
DEFINE FUNCTION bubbleSort(arr):

  SET n TO len(arr) → 2

  FOR i IN range(n):  n
    FOR j IN range(0, n-i-1):  n-1-i

      IF arr[j] > arr[j+1] :→3
        SET temp TO arr[j] → 2
        SET arr[j] TO arr[j+1] → 3
        SET arr[j+1] TO temp → 2

  RETURN arr → 1
```

$$T(n) = \left(\sum_{i=0}^{n} \sum_{j=0}^{n-1-i} 10 \right) + 3$$

$$T(n) = \frac{10\,(n-1)(n)}{2} + 3$$

$$T(n) = 5n^2 - 5n + 3 = O(n^2)$$

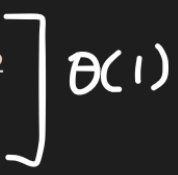
Bubble sort's time complexity is **$O(n^2)$**

```
DEFINE FUNCTION median(arr):

  SET arr TO bubbleSort(arr)

  SET length TO len(arr)

  IF length % 2 EQUALS 0:
    RETURN ( arr[int(length/2)] + arr[int(length/2) - 1] ) / 2
  ELSE:
    RETURN ( arr[int(length/2)])
  ENDIF
```



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

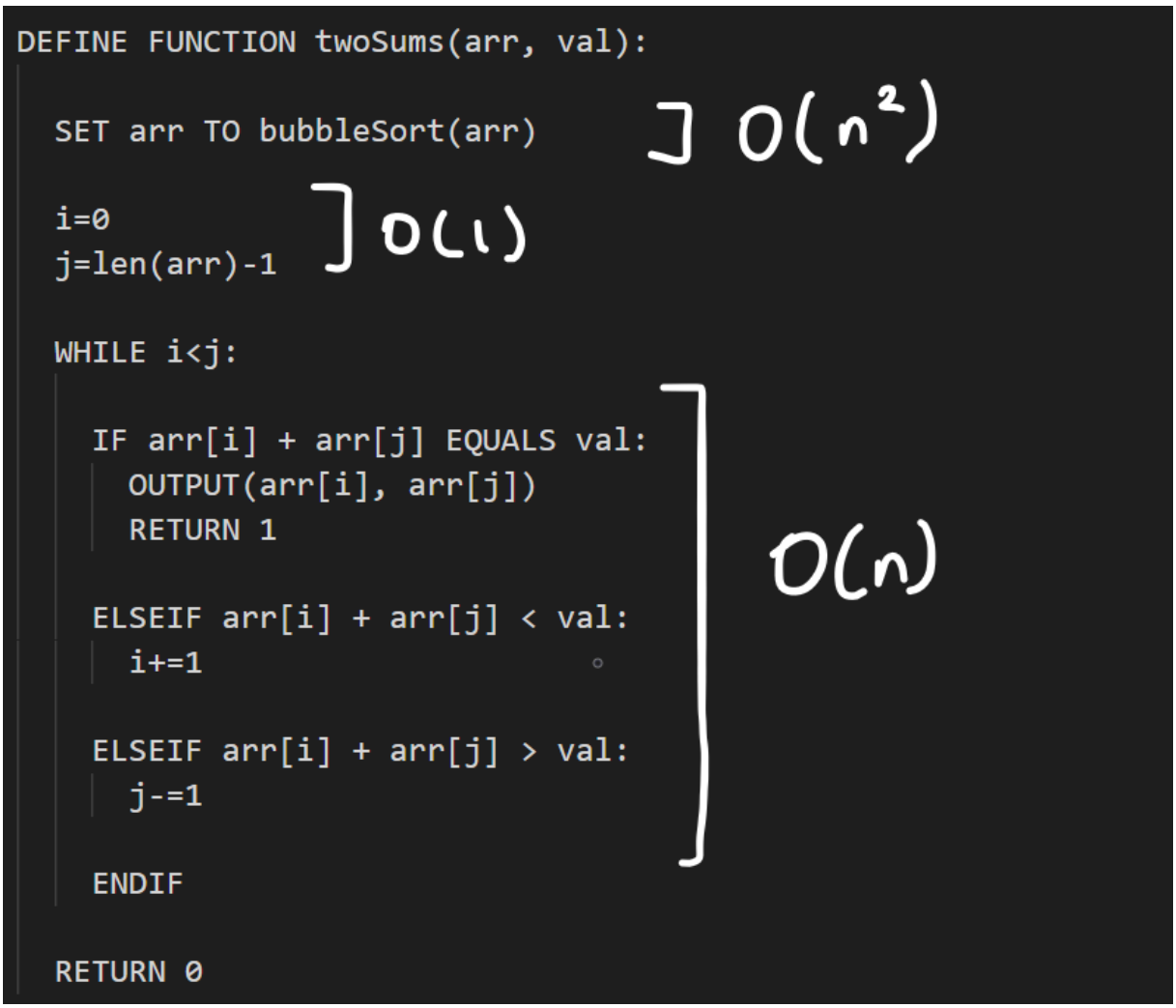
In worst case : Tworst = **$O(n^2)$**

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

At first we sort the array with bubblesort algoritm ($O(n^2)$)

Inside the while loop, time complexity is O(n). When the two elements are in the middle of the array. Or two sums does not exists.

Tworst = **$O(n^2)$**

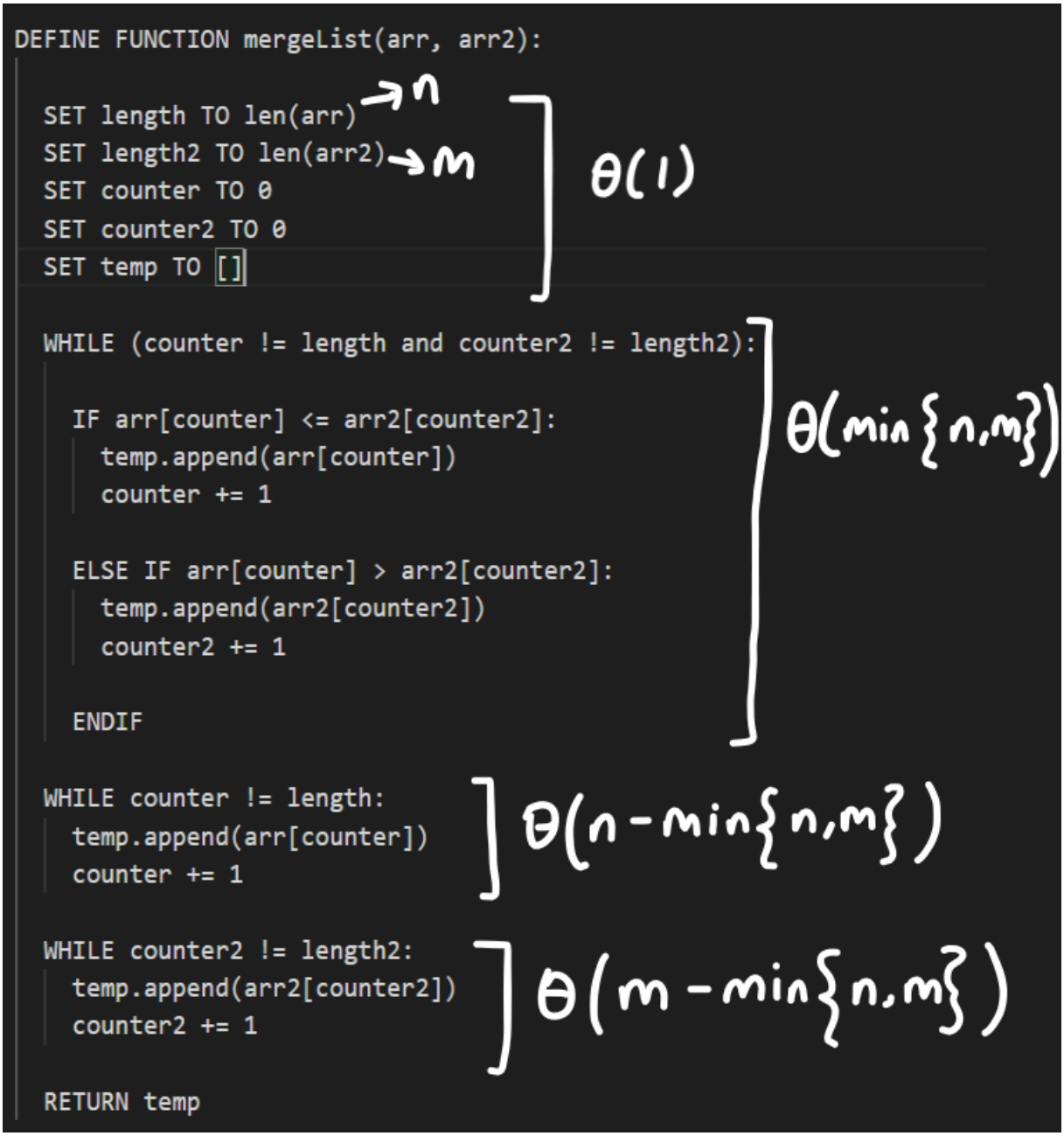


- Assume there are two ordered array list of n elements. Merge these two lists to get a single list in increasing order.

In this example, n is equal to the length of the first and m is equal to the length of the second array

Time complexity depends on the size of the bigger array and it takes linear time

Θ(1) + Θ(min{n,m}) + Θ(n - min{n,m}) + Θ(m - min{n,m}) = Θ(n+m) = **Θ(max{n,m})**



Part 5 :

a)

Time complextiy : Θ(1)

Space complexity : Θ(1)

```
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
}
```

Θ(1) Θ(1) Θ(n) Θ(1)

b)

Time Complexity : Θ(1) + Θ(n) + Θ(1) = Θ(n)

Space Complexity : Θ(1)

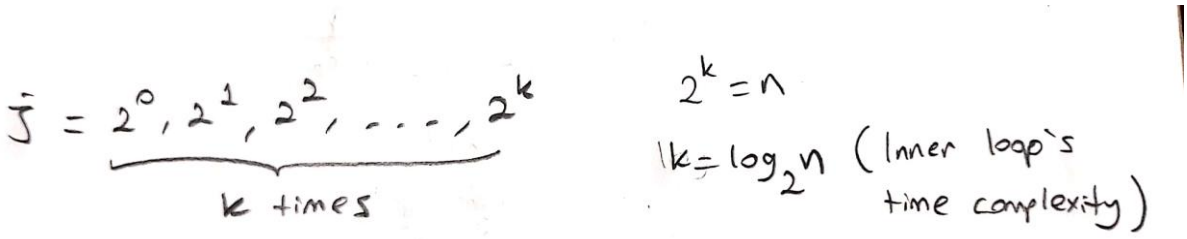
```
int p_2 (int array[], int n):
{
        Int sum = 0
        for (int i = 0; i < n; i=i+5)
                sum += array[i] * array[i])
        return sum
}
```

Θ(1) Θ(1) Θ(n) Θ(1)

c)

Time Complexity : $\Theta(n \log_2 n)$

Space Complexity : $\Theta(1)$




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

$\Theta(1)$ $\Theta(\log_2 n)$ $\Theta(n \log_2 n)$

d)

Time Complexity :

$T_{worst}(n) = \Theta(n) + \Theta(n \log_2 n) = \Theta(n \log_2 n)$

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

Space Complexity : $\Theta(1)$

```
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))
}
```

→ $\Theta(n)$

→ $\Theta(n \log_2 n)$

→ $\Theta(1 \cdot n) = \Theta(n)$