

Bariř Ayyıldız

1901042252

## Part 1 :

### I. Searching a product.

Time complexity is :  $\Theta(n*m*l)$

n is length of stocks array,

m is length of furnitures array,

l is length of String query

```
public void searchProduct(String query)
{
    List<Stock> stocks = this.company.getStocks();
    String temp;

    for(int i=0; i<stocks.length(); i++)
    {
        for(int j=0; j<stocks.get(i).getFurnitures().length(); j++)
        {
            temp = stocks.get(i).getFurnitures().get(j).getType().toString();
            if(temp.contains(query))
            {
                System.out.println(temp);
            }
        }
    }
}
```

Handwritten annotations on the code:

- $\Theta(1)$  next to `getStocks()`
- $\rightarrow n$  above the first `for` loop
- $\rightarrow m$  above the second `for` loop
- $\rightarrow l$  above the `contains(query)` check
- $\Theta(1)$  next to the `contains(query)` check
- A large bracket on the right side of the loops is labeled  $\Theta(nml)$

### 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 =  $\Omega(1)$ , best case is constant time, when the algorithm iterates once inside for loop

n is equal to length of stock array in this case

```

@Override
public void addProducts(int branchId, int productId, int amount) throws Exception
{
    if(amount < 0)
        throw new Exception("Amount should be greater than 0"); } O(1)

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

    for(int i=0; i<stocks.length(); i++)
    {
        if(stocks.get(i).getId() == branchId)
        {
            index = i;
            break; } O(1) } O(n)

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

    int total = stocks.get(index).getFurnitures().get(productId).getTotal(); } O(1)
    stocks.get(index).getFurnitures().get(productId).setTotal(total + amount); } O(1)
}

```

All getters and setters take constant time,  $\Theta(1)$

```

public List<Stock> getStocks(){return stocks;}

@Override
public T get(int index) throws ArrayIndexOutOfBoundsException
{
    if(index >= this.size || index < 0)
        throw new ArrayIndexOutOfBoundsException("Index is out of bounds");
    return this.arr[index];
}

```

## Remove Products

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

Tworst =  $O(n)$

Tbest =  $\Omega(1)$ , 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

```

@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);
}

```

Handwritten annotations for time complexity analysis:

- $\Theta(1)$  for the first `if` statement.
- $\Theta(1)$  for the `getStocks()` call.
- $\Theta(1)$  for the `if` statement inside the loop.
- $\Theta(n)$  for the loop itself.
- $\Theta(1)$  for the `if` statement after the loop.
- $\Theta(1)$  for the `getTotal()` call.
- $\Theta(1)$  for the `if` statement before the `setTotal` call.
- $\Theta(1)$  for the `setTotal` call.

### 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

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

```

@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(j).getBranch().getBranchId() + "\t\t" + total + "\n";
        }
    }

    System.out.println(str);
}

```

Handwritten annotations for time complexity analysis:

- $\Theta(1)$  for the `getStocks()` call.
- $\Theta(1)$  for the `String` declaration.
- $\Theta(n)$  for the outer loop.
- $\Theta(m)$  for the inner loop.
- $\Theta(n*m)$  for the nested loops.
- $\Theta(1)$  for the `if` statement inside the inner loop.
- $\Theta(1)$  for the `println` call.

All getters and setters take constant time,  $\Theta(1)$

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

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

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

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

## Part 2 :

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

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:  $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ .

**true**

Handwritten proof of the theorem:  $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ .

we need to prove

Left side (Lower bound):

$$\max(f(n), g(n)) = \Omega(f(n) + g(n))$$

↓

$$\max(f(n), g(n)) \geq g(n)$$
$$\max(f(n), g(n)) \geq f(n)$$

+

$$2 \max(f(n), g(n)) \geq f(n) + g(n)$$
$$\max(f(n), g(n)) \geq \frac{1}{2} [f(n) + g(n)]$$

$\underbrace{\frac{1}{2}}_c$

$$\max(f(n), g(n)) = \Omega(f(n) + g(n))$$

✓

Right side (Upper bound):

$$\max(f(n), g(n)) = O(f(n) + g(n))$$
$$\max(f(n), g(n)) \leq f(n)$$
$$\max(f(n), g(n)) \leq g(n)$$

+

$$2 \max(f(n), g(n)) \leq f(n) + g(n)$$
$$\max(f(n), g(n)) \leq \frac{1}{2} [f(n) + g(n)]$$

$\underbrace{\frac{1}{2}}_c$

$$\max(f(n), g(n)) = O(f(n) + g(n))$$

✓

c) Are the following true? Prove your answer.

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

**true**

$$\begin{aligned} \text{I) } 2^{n+1} = \Theta(2^n) &\Leftrightarrow c_1 \cdot 2^n \geq 2^{n+1} \geq c_2 \cdot 2^n, c_1, c_2 > 0 \text{ for all } n \geq n_0 \\ c_1 \cdot 2^n &\geq 2^{n+1} & 2^{n+1} &\geq c_2 \cdot 2^n \\ \text{for } n=1 & c_1 \cdot 2 \geq 4 & \text{for } n=1 & 4 \geq c_2 \cdot 2 \\ & c_1 \geq 2 & & 2 \geq c_2 \\ n \geq n_0 = 1, & c_1 = c_2 = 2 & \checkmark \end{aligned}$$

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

**false**

$$\begin{aligned} \text{II) } 2^{2n} = \Theta(2^n) &\Leftrightarrow c_1 \cdot 2^n \geq 2^{2n} \geq c_2 \cdot 2^n, c_1, c_2 > 0 \text{ for all } n \geq n_0 \\ c_1 \cdot 2^n &\geq 2^{2n} \\ c_1 &\geq 2^n, \text{ false since } c_1 \text{ is a constant} \end{aligned}$$

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

$$\begin{aligned}
 f(n) &= O(n^2) & g(n) &= \Theta(n^2) \\
 c_1 n^2 &\geq f(n) & c_2 n^2 &\geq g(n) \geq c_3 n^2 \\
 c_1, c_2, c_3 &> 0 \\
 \Rightarrow c_1 n^2 \cdot c_2 n^2 &\geq f(n) \cdot g(n) \quad \downarrow \\
 f(n) \cdot g(n) &= O(n^2 \cdot n^2) & \text{Big-O notation definition} \\
 f(n) \cdot g(n) &= O(n^4)
 \end{aligned}$$

Big-O notation doesn't tell anything about the lower bound. So for time complexity, **it is not guaranteed to be  $\Theta(n^4)$** . 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.

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

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

$$\lim_{n \rightarrow \infty} \frac{\log^3 n}{\log n} = \lim_{n \rightarrow \infty} \log^2 n = \infty \quad ; \quad \log^3 n > \log n$$



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

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

L'Hopital

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

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

$$\lim_{n \rightarrow \infty} \frac{n^{1.01}}{n \log^2 n} = \lim_{n \rightarrow \infty} \frac{n^{0.01}}{\log^2 n} \xrightarrow{\text{L'Hopital}} \lim_{n \rightarrow \infty} \frac{0.01 \cdot n^{-0.99}}{2 \log n \cdot \frac{1}{\ln 10 \cdot n}}$$

$$= \lim_{n \rightarrow \infty} \frac{0.005 \cdot n^{0.01} \cdot \ln 10}{\log n} \xrightarrow{\text{L'Hopital}} \lim_{n \rightarrow \infty} \frac{5 \times 10^{-4} n^{-0.99} \ln 10}{\frac{1}{\ln 10 \cdot n}}$$

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

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

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

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

L'Hopital

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

L'Hopital

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

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

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

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

**Result :**

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

#### Part 4 :

##### - Find the minimum-valued item.

It takes linear time

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

```
DEFINE FUNCTION findMin(arr):
```

```
    SET minVal TO arr[0] ]  $\Theta(1)$ 
```

```
    FOR i IN range(1, len(arr)):
```

```
        IF arr[i] < minVal:
```

```
            SET minVal TO arr[i]
```

```
    RETURN minVal ]  $\Theta(1)$ 
```

$\left. \begin{array}{l} \Theta(1) \\ \Theta(n) \\ \Theta(1) \end{array} \right\} \Theta(n)$

- 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
```

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

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

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

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

```
DEFINE FUNCTION median(arr):  
    SET arr TO bubbleSort(arr) ]  $O(n^2)$   
    SET length TO len(arr) ]  $\Theta(1)$   
    IF length % 2 EQUALS 0:  
        RETURN ( arr[int(length/2)] + arr[int(length/2) - 1] ) / 2  
    ELSE:  
        RETURN ( arr[int(length/2)])  
    ENDIF ]  $\Theta(1)$ 
```

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

In worst case :  $T_{\text{worst}} = O(n^2)$

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

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

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 =  $O(n^2)$

```
DEFINE FUNCTION twoSums(arr, val):
```

```
    SET arr TO bubbleSort(arr)
```

$O(n^2)$

```
    i=0
```

```
    j=len(arr)-1
```

$O(1)$

```
    WHILE i<j:
```

```
        IF arr[i] + arr[j] EQUALS val:
```

```
            OUTPUT(arr[i], arr[j])
```

```
            RETURN 1
```

```
        ELSEIF arr[i] + arr[j] < val:
```

```
            i+=1
```

```
        ELSEIF arr[i] + arr[j] > val:
```

```
            j-=1
```

```
        ENDIF
```

```
    RETURN 0
```

$O(n)$

- 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

$$\Theta(1) + \Theta(\min\{n, m\}) + \Theta(n - \min\{n, m\}) + \Theta(m - \min\{n, m\}) = \Theta(n+m) = \Theta(\max\{n, m\})$$

```
DEFINE FUNCTION mergeList(arr, arr2):
```

```
  SET length TO len(arr)  $\rightarrow n$ 
  SET length2 TO len(arr2)  $\rightarrow m$ 
  SET counter TO 0
  SET counter2 TO 0
  SET temp TO []
```

$\Theta(1)$

```
  WHILE (counter != length and counter2 != length2):
```

$\Theta(\min\{n, m\})$

```
    IF arr[counter] <= arr2[counter2]:
      temp.append(arr[counter])
      counter += 1

    ELSE IF arr[counter] > arr2[counter2]:
      temp.append(arr2[counter2])
      counter2 += 1

  ENDIF
```

```
  WHILE counter != length:
```

$\Theta(n - \min\{n, m\})$

```
    temp.append(arr[counter])
    counter += 1
```

```
  WHILE counter2 != length2:
```

$\Theta(m - \min\{n, m\})$

```
    temp.append(arr2[counter2])
    counter2 += 1
```

```
  RETURN temp
```

**Part 5 :**

**a)**

Time complexity :  $\Theta(1)$

Space complexity :  $\Theta(1)$

```
int p_2 (int array[], int n):
```

```
{
```

```
    Int sum = 0
```

$\Theta(1)$

```
    for (int i = 0; i < n; i=i+5)
```

```
        sum += array[i] * array[i]
```

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

```
    return sum
```

$\Theta(1)$

```
}
```

**b)**

Time Complexity :  $\Theta(1) + \Theta(n) + \Theta(1) = \Theta(n)$

Space Complexity :  $\Theta(1)$

```
int p_2 (int array[], int n):
```

```
{
```

```
    Int sum = 0
```

$\Theta(1)$

```
    for (int i = 0; i < n; i=i+5)
```

```
        sum += array[i] * array[i]
```

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

```
    return sum
```

$\Theta(1)$

```
}
```



c)

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

Space Complexity :  $\Theta(1)$

$$j = 2^0, 2^1, 2^2, \dots, 2^k$$

$\underbrace{\hspace{10em}}_{k \text{ times}}$

$$2^k = n$$
$$k = \log_2 n \quad (\text{Inner loop's time complexity})$$

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])

}

$\Theta(n \log_2 n)$

$\left. \begin{array}{l} \left. \left. \right] \Theta(1) \right] \right] \Theta(\log_2 n) \end{array} \right] \Theta(n \log_2 n)$

d)

Time Complexity :

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

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

Space Complexity :  $\Theta(1)$

void p\_4 (int array[], int n):

{

    If (p\_2(array, n)) > 1000  $\rightarrow \Theta(n)$

        p\_3(array, n)  $\rightarrow \Theta(n \log_2 n)$

    else

        printf("%d", p\_1(array) \* p\_2(array, n))  $\rightarrow \Theta(1 \cdot n) = \Theta(n)$

}