

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

**Ozan GEÇKİN**  
**1801042103**

## Part1:

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

### I. Searching a product

```
0.    public boolean searchProduct(OfficeProducts newProduct){
1.        for(int i=0; i < onlineBranch.getStockCapacity(); i++){
2.            if(onlineBranch.stocks[i] != null && test != null){
3.                if(onlineBranch.stocks[i].getName().equals(newProduct.getName())
                    &&onlineBranch.stocks[i].getModel().equals(newProduct.getModel())
                    &&onlineBranch.stocks[i].getColor().equals(newProduct.getColor())){
4.                    return true;
                    }
                }
            }
        }
5.    return false;
}
```

Not. I used the leading ordinal numbers to indicate rows in the table. onlineBranch is a pre-set branch.

### Analysis:

I called the get functions of the officeProduct object in my method. Their time complexity is  $O(1) = \Omega(1) = \theta(1)$ . I use for complexities of the equals functions are Worst case  $O(n)$ , Best case  $\Omega(1)$ , Average case  $\theta(n)$ .

In line 2, I use for loop, it returns  $n$  to scan all the array. The equals that I use in if conditions return  $n$

Let say `onlineBranch.getStockCapacity()`= $n$

index	Step/exec	Freq	Total
1	2	$n+1$	$2n+2$
2	1	$n$	$n$
3	1	$n$	$n$
4	1	1	1
5	1	1	1

**Time complexity:**

Total= $T(n) = 4n+4 = O(n) = \theta(n)$ .

**Space complexity:** $O(1)$

## II. Add/remove product.

### Add product method:

```
0. public void addProductBranch(OfficeProducts newProduct) {
1.     if(onlineBranch.getStockCapacity() == onlineBranch.getStockNumber()){
2.         onlineBranch.increaseStokcsCapacity();
3.     }
4.     onlineBranch.stocks[onlineBranch.getStockNumber()] = newProduct;
5.     int newStock = onlineBranch.getStockNumber()+1;
6.     System.out.println("Adding Product");
7.     onlineBranch.setStockNumber(newStock);
8. }
9.
```

### Helper method to the add method:

```
0. public void increaseStokcsCapacity(){
1.     OfficeProducts[] temp = new OfficeProducts[stockCapacity];
2.     for(int i=0 ; i < stockNumber ; i++){
3.         if(stocks[i] !=null){
4.             temp[i] = new OfficeProducts(stocks[i].getModel(),stocks[i].getColor(),stocks[i].getName());
5.         }
6.     }
7.     stocks = new OfficeProducts[stockCapacity * 2];
8.     for(int i=0; i < stockNumber ; i++){
9.         if(temp[i] !=null){
10.            stocks[i] = new OfficeProducts(temp[i].getModel(),temp[i].getColor,temp[i].getName());
11.        }
12.    }
13.    stockCapacity*=2;
14. }
15.
```

Not. I used the leading ordinal numbers to indicate rows in the table.

### Analysis:

The function that helps the add function is the function that increases the array capacity. Since there are separate for loops in it, its time complexity is (n).  
increaseStokcsCapacity(), Let say stockNumber =n.

index	Step/exec	Freq	Total
1	1	1	1
2	2	n+1	2n+2
3	1	n	n
4	1	n	n
5	1	1	1
6	2	n+1	2n+2
7	1	n	n
8	1	n	n
9	1	1	1

### Time Complexity:

Total =  $T(n) = 8n+7 = O(n) = \theta(n)$

Space Complexity =  $O(n)$ .

### Analysis of addProductBranch:

It allows the product to be added to the branch's stock array. There is one if condition in it, and it increases the capacity of the array. The time complexity of the function that increases the capacity is  $T(n) = \theta(n)$ . The time complexity of the get and set methods I use in the addProductBranch method is  $\theta(1)$ . If the capacity is not filled, the best case of the method is  $T(n) = \Omega(1)$ . Worst case is that the capacity increasing function is called  $T(n) = O(n)$ .

Let say `onlineBranch.getStockCapacity()`=n.

index	Step/exec	Freq	Total
1	1	1	1
1	1	n	n
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1

**Time complexity :**  $T(n) = n+5$

Worst case=  $O(n)$

Best case=  $\Omega(1)$

**Average=**  $\theta(n)$

**Space complexity :**  $O(n)$

## Remove product method:

```
0. public void removeProduct(OfficeProducts product) throws MyException {
1.     boolean check = false;
2.     try{
3.         for(int i=0; i< onlineBranch.getStockCapacity();i++){
4.             if(onlineBranch.stocks[i].getName().equals(newProduct.getName())
5.               &&onlineBranch.stocks[i].getModel().equals(newProduct.getModel())
6.               &&onlineBranch.stocks[i].getColor().equals(newProduct.getColor())){
7.                 onlineBranch.stocks[i] = null;
8.                 onlineBranch.setStockNumber(online.getStockNumber()-1);
9.                 System.out.println("Remove Product");
10.                check=true;
11.            }
12.        }
13.        if(check==false){
14.            throw new MyException("Product is not Branch Stock");
15.        }
16.    }catch(MyException e){
17.        System.err.println(e.getMessage());
18.    }
19. }
```

### Analysis of removeProduct:

I used the table method to explain my method that takes a Product object as a parameter.

Let say `onlineBranch.getStockCapacity()`=n.

index	Step/exec	Freq	Total
1	1	1	1
2	2	n+1	2n+2
3	1	n	n
4	1	n	n
5	1	n	n
6	1	n	n
7	1	n	n
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1

### Time complexity:

$$T(n) = 2n+10$$

Best case= average case

$$O(n) = \theta(n).$$

$\Omega(1)$  because of throw exception

### III. Querying the products that need to be supplied.

```
0. public boolean checkStock(Branch branch){
1.     counter =0;
2.     counter2=1;
3. for(int i=0;i<branch.stock.length;i++){
4.     for(int j=0;j<product.length;j++){
5.         if(branch.stock[i].getProduct().equals(product[j])){
6.             counter++;
7.         }
8.         else{
9.             counter2++;
10.        }
11.    }
12. }
13. If(counter<5 || counter2<5){
14.     System.out.println("Product need");
15. }
16. }
```

#### Analysis of checkStock:

I used the table method to explain my method that takes a Branch object as a parameter.

Let say  $n = \text{branch.stock.length}$

Let say  $m = \text{product.length}$

#### Time complexity:

$$T(m,n) = 3mn + 8 + 2n + 2m$$

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

**Space complexity :**  $O(1)$

index	Step/exec	Freq	Total
1	1	1	1
2	1	1	1
3	2	$n+1$	$2n+2$
4	2	$m+1$	$2m+2$
5	1	$mn$	$mn$
6	1	$mn$	$mn$
7	1	$mn$	$mn$
8	1	1	1
9	1	1	1

## Part 2:

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

\* Let  $T(n)$  be the running time for algorithm A and Let a function  $f(n)=O(n^2)$ .

\*  $T(n)$  is at least  $O(n^2)$ . That is,  $T(n)$  is an upper bound of  $f(n)$ .

\*  $f(n)$  can run in a running time less than  $n^2$ .

\* For example A be a find algorithm.

\* If it finds what is looked for at first, or if the array is checked to see if it is null and the array is null, this statement as The running time of algorithm A is at least constant.

\* So this expression "The running time of algorithm A is at least  $O(n^2)$ " is meaningless the running time for every algorithm is at least constant.

b)

b)  $\max(f(n), g(n)) = \Theta(f(n) + g(n))$  for prove

Primarily prove upper bound

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

Let's show that this statement is true.

Let  $\max(f(n), g(n)) = f(n)$  then  $g(n) \geq 0$

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

So,  $\max(f(n), g(n)) \leq c(f(n) + g(n))$  for all  $n \geq n_0$  and  $c=1, n_0=1$

$$\text{prove} \Rightarrow \max(f(n), g(n)) = O(f(n) + g(n))$$

Secondly prove lower bound

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

Let  $\max(f(n), g(n)) = f(n)$  -  $f(n) \geq g(n)$  so  $2f(n) \geq g(n) + f(n)$

$$\text{So } 2 \max(f(n), g(n)) = f(n) \geq f(n) + g(n)$$

Simplification,  $\max(f(n), g(n)) \geq c(f(n) + g(n))$  for all  $n \geq n_0$  and  $c=\frac{1}{2}, n_0=1$

$$\text{prove} \Rightarrow \max(f(n), g(n)) = \Omega(f(n) + g(n)).$$

finally, When you combine proofs of two statements.

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

c) Are the following true? Prove your answer.

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

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

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

The same theta notation because coefficients are ignored.

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

$$2^{2n} = 4^n$$

$$4^n \neq 2^n$$

This condition is false.

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

Functions have 3 states: average, best time, and worst time. However, if we know the worst case of another function, there cannot be anything worse. Since one of the functions is worst time, we cannot say it is true for such a multiplication operation. So this state is wrong since we don't have the mean for the function  $f$ .



### Part 3:

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

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

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

for order of growth Using Limit calculating :

The order I made when I examined the given functions.

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

→  $3^n$  and  $n 2^n$

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

Use L'Hospital

$$= \lim_{x \rightarrow \infty} \left( \frac{x}{\left( \frac{3}{2} \right)^x} \right) = \lim_{x \rightarrow \infty} \left( \frac{1}{\left( \frac{3}{2} \right)^x \ln \left( \frac{3}{2} \right)} \right) = \lim_{x \rightarrow \infty} \left( \frac{2^x}{\ln \left( \frac{3}{2} \right) \cdot 3^x} \right)$$

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

Use condition,

$$\lim_{x \rightarrow \infty} (a^x) = 0, \quad 0 < a < 1$$

$$= \frac{\ln^2(2)}{\ln^2(3) \ln \left( \frac{3}{2} \right)} \cdot 0 = 0$$

Result

$$3^n > n 2^n$$

→  $n 2^n$  and  $2^{(n+1)}$

$$\lim_{n \rightarrow \infty} \frac{2^{(n+1)}}{n 2^n} \Rightarrow \lim_{n \rightarrow \infty} \frac{2^{n+1}}{n 2^n} \Rightarrow \lim_{n \rightarrow \infty} \left( \frac{2}{n} \right) = 0$$

Result

$$n 2^n > 2^{(n+1)}$$

→  $2^{(n+1)}$  and  $2^n$

$$\lim_{n \rightarrow \infty} \left( \frac{2^n}{2^{n+1}} \right) = \frac{1}{2}$$

Result

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

$\rightarrow 2^n$  and  $5^{\log_2 n}$   
 $\lim_{n \rightarrow \infty} \frac{5^{\log_2 n}}{2^n} = 0$       Result  
 $2^n > 5^{\log_2 n}$

$\rightarrow 5^{\log_2 n}$  and  $n \log^2 n$   
 $\lim_{n \rightarrow \infty} \frac{n \log^2 n}{5^{\log_2 n}} = 0$       Result  
 $5^{\log_2 n} > n \log^2 n$

$\rightarrow n^{1.01}$  and  $\sqrt{n}$   
 $\lim_{n \rightarrow \infty} \left( \frac{\sqrt{n}}{n^{1.01}} \right) = 0$       Apply infinity property  
 $\lim_{x \rightarrow \infty} \left( \frac{\sqrt{x}}{x^{1.01}} \right) = 0$       Result  
 $n^{1.01} > \sqrt{n}$

$\rightarrow \sqrt{n}$  and  $(\log n)^3$   
 $\lim_{n \rightarrow \infty} \left( \frac{(\ln n)^3}{\sqrt{n}} \right) = 0$       Apply L'Hospital  
 $\lim_{n \rightarrow \infty} \left( \frac{\frac{3 \ln^2(n)}{2n}}{\frac{1}{2\sqrt{n}}} \right) = \lim_{n \rightarrow \infty} \left( \frac{6 \ln^2(n)}{\sqrt{n}} \right)$   
 $= \lim_{n \rightarrow \infty} \left( \frac{24 \ln(n)}{\sqrt{n}} \right)$   
 $= \lim_{n \rightarrow \infty} \left( \frac{48}{\sqrt{n}} \right)$   
 $= 0$       Result  
 $\sqrt{n} > (\log n)^3$

$\rightarrow (\log n)^3$  and  $\log n$   
 $\lim_{n \rightarrow \infty} \left( \frac{(\ln n)}{(\ln n)^3} \right) = 0$   
 $\lim_{n \rightarrow \infty} \left( \frac{1}{\ln^2(n)} \right) = \frac{1}{\infty} = 0$       Result  
 $(\log n)^3 > \log n$

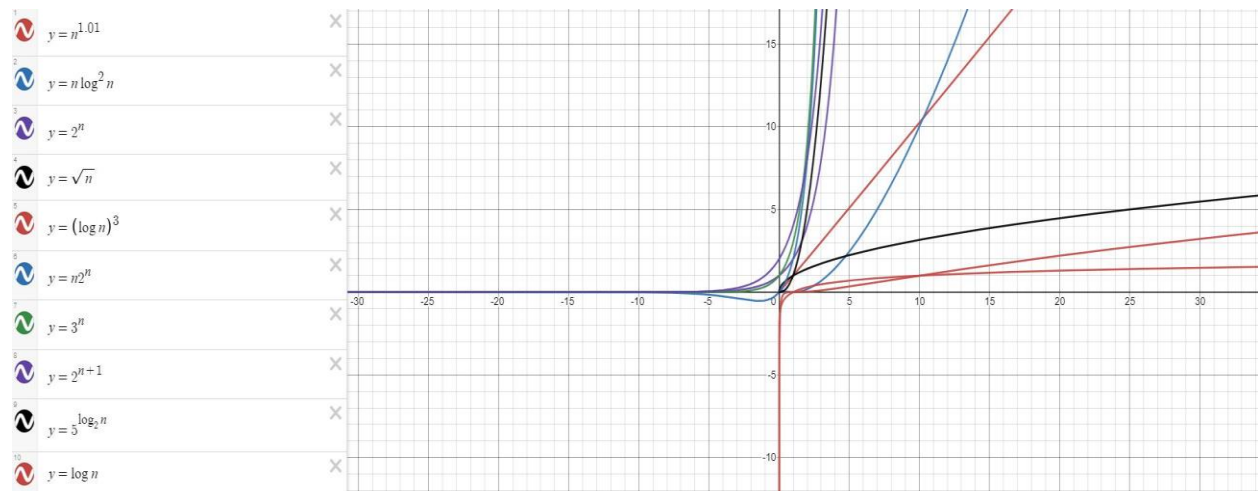
CS CamScanner ile tarandı

### Analyze every function asymptotically

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

1.  $3^n \rightarrow O(2^n)$  [logarithm of  $3^n$ , convert  $2^n$ ]
2.  $n 2^n \rightarrow O(n 2^n)$
3.  $2^{n+1} \rightarrow O(2^n)$  [constant coefficient does not matter]
4.  $2^n \rightarrow O(2^n)$
5.  $5^{\log_2 n} \rightarrow O(n^2)$
6.  $n \log^2 n \rightarrow O(n \log n)$  [square of the log becomes a constant coefficient  $2n \log n$  is asymptotically  $n \log n$ ]
7.  $n^{1.01} \rightarrow O(n)$  [1.01 is about 1]
8.  $\sqrt{n} \rightarrow O(\sqrt{n})$
9.  $(\log n)^3 \rightarrow O(\log n)^3$
10.  $(\log n) \rightarrow O(\log n)$

for order of growth Using drawing graph:



## Part4:

Give the pseudo-code for each of the following operations for an array list that has n elements and analyze the time complexity:

**-Find the minimum-valued item.**

### Pseudo code

```
findMinValue(arr[],int size)
```

```
min = arr[0]
```

```
for i = 1 to n
```

```
    if arr[i] < min
```

```
        min = arr[i]
```

```
    End-if
```

```
End-for
```

Step/exec	Freq	Total
1	1	1
2	n+1	2n+1
1	n	n
1	n	n

### Analysis:

I compare 1 time in each for loop. If the comparison is correct, it is +1 step. This is in the worst case. If the first element is the smallest and does not enter the if in the loop, this is a best case.

Time complexity  $T(n) = 4n+1 = O(n) = \Omega(n) = \theta(n)$

Space complexity =  $O(1)$

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

### Pseudo code

```
findMedian(arr[],int size)
```

```
for i=0 to arr.size()-1
```

```
    for j=0 to arr.size()-i-1
```

```
        if arr[j] < arr[j+1]
```

```
            temp=arr[j]
```

```
            arr[j]=arr[j+1]
```

```
            arr[j+1]=temp
```

```
        End-if
```

```
    End-for
```

```
End-for
```

```
If(size %2 ==0)
```

```
    mid=(size-1)/2
```

```
else
```

```
    mid=(size+1)/2
```

```
return arr[mid];
```

Step/exec	Freq	Total
2	n	2n
2	(n-1)*(n-2)	2n <sup>2</sup> -6n+4
1	2n <sup>2</sup> -6n+3	2n <sup>2</sup> -6n+3
1	2n <sup>2</sup> -6n+3	2n <sup>2</sup> -6n+3
1	2n <sup>2</sup> -6n+3	2n <sup>2</sup> -6n+3
1	2n <sup>2</sup> -6n+3	2n <sup>2</sup> -6n+3
1	1	1
1	1	1
1	1	1
1	1	1

### Analysis:

#### Time Complexity

I used bubble sort algorithm. I showed step / exec and freq on the table. Since the forms are nested, their total values will be multiplied.

$$\text{Total}=T(n)=10n^2-28n+20$$

#### General running time

Worst case  $O(n^2)$

Best case  $\Omega(n)$

Average case  $\theta(n^2)$

#### Space Complexity= $O(1)$

The space complexity for Bubble Sort is  $O(1)$ , because only a single additional memory space is required for temp variable.

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

### **Pseudo Code**

```
subArrayTwoElements(arr[],int size,sum)
    Arrays.sort(arr)
    i=0
    while i < size-1
        if(arr[i] +arr[size-1] == sum)
            return 1;
        else if(arr[i] + arr[size-1]< sum)
            i++;
        else
            size--;
    End-while
```

### **Analysis:**

#### **Time Complexity**

I sort the array. There are 3 sorting methods.

Merge sort and Heap sort => worst case  $O(n \log n)$

Quic sort =>  $O(n^2)$  =>  $\theta(n^2)$

#### **Space complexity**

$O(n)$  since I'm using the sort algorithm.

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

### Pseudo Code

```
mergeAndSort(ArrayList<> arr,ArrayList<> arr2)
for i=0 to arr2.size()
    arr.add(arr2[i])
for i=0 to arr.size()-1
    for j=0 to arr.size()-i-1
        if arr[j] < arr[j+1]
            temp=arr[j]
            arr[j]=arr[j+1]
            arr[j+1]=temp
        End-if
    End-for
End-for
return arr;
```

Step/exec	Freq	Total
2	m	2m
1	1	1
2	n	2n
2	$(n-1)*(n-2)$	$2n^2-6n+4$
1	$2n^2-6n+3$	$2n^2-6n+3$
1	$2n^2-6n+3$	$2n^2-6n+3$
1	$2n^2-6n+3$	$2n^2-6n+3$
1	$2n^2-6n+3$	$2n^2-6n+3$
1	1	1

### Analysis:

First, I put all the elements in arr using a single for loop. Then I sort by using nested for loop. I analyzed it line by line using the table model.

$$T(n,m)=2m+10n^2-28n+18$$

**General Time Complexity:**

$$T(n,m)=O(n^2+m)=\theta(n^2+m)$$

**Space Complexity =  $O(n)$**

## Part 5:

a)

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

Step/exec	Freq	Total
2	1	2

### Analysis:

Total=  $f(n) = 2$

Since there is no condition and there is no loop we don't have best worst and average case Since we don't have any condition.

Time complexity=  $T(n) = O(1) = \theta(1)$

Space complexity=  $O(1)$

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  
}
```

Step/exec	Freq	Total
1	1	1
2	$(n/5)+1$	$(2n/5)+2$
3	$n/5$	$3n/5$
1	1	1

### Analysis:

Since there is no condition and there is no loop we don't have best worst and average case Since we don't have any condition. The for loop for the index 5 and the remainder of 5 evil.  $(N / 5)$  returns +1 times.

Total =  $n+4$

Time compleixy= $T(n)=n+4=O(n) = \theta(n)$ , Space complexty=  $O(1)$

**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])  
}
```

Step/exec	Freq	Total
2	n+1	2n+2
2	nlogn	2nlogn
1	logn	logn

### Analysis:

Since we don't have any condition we don't have best worst and average case.

Total=2n+2+2nlogn+logn

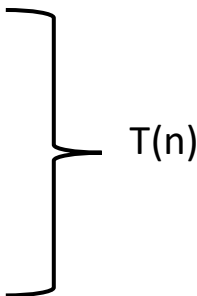
Time complexiy= $T(n) = 2n+2+2n\log n + \log n = O(n\log n) = \theta(n\log n)$

Space complexty=  $O(1)$

**d)**

```
void p_4 (int array[], int n):
```

```
{  
    If (p_2(array, n)) > 1000 } T3(n) =  $\theta(n)$   
    p_3(array, n) } T1(n) =  $\theta(n\log n)$   
    else  
    printf("%d", p_1(array) * p_2(array, n)) } T2(n) =  $\theta(n)$   
}
```



### Analysis:

Since there is an if else case in this function, I will show the time complexity by calculating Tbest, Tworst, Taverage. Since I analyzed the functions called in the function above, I showed their time complexity on the code.



- $T_w(n) = T_3(n) + \max(T_1(n), T_2(n))$
- $T_b(n) = T_3(n) + \min(T_1(n), T_2(n))$
- $p(T) \rightarrow p$  (condition true)
- $p(F) \rightarrow p$  (condition false)
- $T_{ave}(n) = p(T)T_1(n) + p(F)T_2(n) + T_3(n)$

This code:

$$T_w(n) = \theta(n) + \max(\theta(n \log n), \theta(n)) = \theta(n \log n)$$

$$T_b(n) = \theta(n) + \min(\theta(n \log n), \theta(n)) = \theta(n)$$

$$\text{If } p(T) = p(F) = 1/2$$

$$T_{ave}(n) = 1/2 \theta(n \log n) + 1/2 \theta(n) + 1/2 \theta(n) = \theta(n \log n)$$

General Running Time:

$$T(n) = O(n \log n)$$

$$= \Omega(n)$$

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

$$\text{Space complexity} = O(1)$$