Emine Sultan Savran

1801042637

GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 2

Part 1:

- Searching a product

```java
public void searchProduct(Product product){
    Branch[] branches = company.getBranches();
    boolean check = false;
    for(int i=0; i<company.getNumOfBranches(); i++){
        if( containsProduct(branches[i], product) ){
            System.out.println( product );
            check = true;
        }
    }
    if( !check ){
        System.out.println("No product.");
    }
}
```

```java
protected boolean containsProduct(Branch branch, Product product){
    Product[] products = branch.getProducts();
    for(int i=0; i<branch.getNumOfProducts(); i++){
        if( products[i].isSame(product) )
            return true;
    }
    return false;
}
```

```java
public boolean isSame(Product newProduct){
    if( this.type.equals( newProduct.getType() ) &&
        this.model.equals( newProduct.getModel() ) &&
        this.color.equals( newProduct.getColor() ) )
        return true;
    return false;
}
```

Part 1.

- Searching Product

* isSame function:

if statement] $T_1(n)$

....] $T_2(n)$

....] $T_3(n)$

$T_{1\,worst}(n) = \Theta(n)$, because of equals method

$T_{1\,best}(n) = \Theta(1)$

$T_2(n) = T_3(n) = \Theta(1)$

$T(n) = T_1 + T_2 + T_3 = \Omega(1) = O(n)$ I

*containsProduct function:

...] $\Theta(1)$

for ...

if ...] $T_2(n)$

...] $\Theta(1)$] $T_1(n)$

...] $\Theta(1)$

$T_{1\,best} = \underbrace{\Theta(1)}_{\text{the loop executes one times}} * T_{2\,best}(n) + \Theta(1) = \Theta(1).$

$T_{2\,best}(n) \to \Theta(1)$ I

$\Theta(1)$ ↳ simple statement

$T_{1\,worst} = \underbrace{\Theta(n)}_{\text{execute } n \text{ times}} * T_{2\,worst}(n) + \Theta(1) = \Theta(n^2)$

$T_{2\,worst}(n) \to \Theta(n)$ I

$\Theta(1)$ ↳ simple statement

$T(n) = \Omega(1) = \Theta(n^2)$ II

* searchProduct function

```
...        ] Θ(1)
...        ] Θ(1)
for ...
    if ...        ] T2(n) ] T1(n)
        ... ] Θ(1)
        ... ] Θ(1)
if ... ] Θ(1)
    ... ] Θ(1)
```

$$T_{1best} = \Theta(1) * T_{2\,best}(n) = \Theta(1)$$

loop executes 1 time $\hookleftarrow$

$\hookrightarrow \Theta(1)$ (from II)

$$T_{2worst} = \Theta(n) * T_{2\,worst}(n) = n^3$$

loop executes n times $\hookleftarrow$

$\hookrightarrow \Theta(n^2)$

$$T(n) = \Omega(1) = O(n^3)$$

- Add/remove a product

```java
public void addProduct(Product newProduct){
    Product[] currentProducts = workBranch.getProducts();
    if( containsProduct(workBranch, newProduct) ){
        System.out.println("The product already exists");
    } else{
        //make sure there is a room
        if( workBranch.getCapacityOfProducts() <= workBranch.getNumOfProducts() ){
            reallocateProducts();
        }
        currentProducts = workBranch.getProducts();
        currentProducts[ workBranch.getNumOfProducts() ] = newProduct;
        workBranch.setNumOfProducts( workBranch.getNumOfProducts()+1 );
    }

}
```

```java
private void reallocateProducts(){
    Product[] products = workBranch.getProducts();
    workBranch.setCapacityOfProducts( workBranch.getCapacityOfProducts()*2 );
    Product[] temp = new Product[ workBranch.getCapacityOfProducts() ];
    for(int i=0; i<workBranch.getNumOfProducts(); i++){
        temp[i] = products[i];
    }
    workBranch.setProducts( temp );
}

 public void removeProduct(Product product){
     if( containsProduct(workBranch, product) ){
         int index = 0;
         Product[] products = workBranch.getProducts();
         Product[] temp = new Product[ workBranch.getCapacityOfProducts() ];
         for(int i=0; i<workBranch.getNumOfProducts(); i++){
             if( products[i].isSame(product) )
                 index++;
             temp[i] = products[index];
             index++;
         }
         workBranch.setProducts( temp );
         workBranch.setNumOfProducts( workBranch.getNumOfProducts()-1 );
     } else{
         System.out.println("No product.");

     }
 }
```

- Add/remove product

* reallocate Products function:

...] $\Theta(1)$
...] $\Theta(1)$
...] $\Theta(1)$
for ...
...] $\Theta(1)$] $\Theta(n)$
...] $\Theta(1)$

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

* add Product function:

// ignore $\Theta(1)$'s.

...] $\Theta(1)$
if ...] $T_1(n)$
...] $\Theta(1)$
else
if ...] $\Theta(1)$
...] $\Theta(n)$ (from III)
...] $\Theta(1)$
...] $\Theta(1)$
...] $\Theta(1)$

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

from II

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

* remove Product function:

if ...] $T_1(n)$
...] $\Theta(1)$
...] $\Theta(1)$
...] $\Theta(1)$
for ...
if ...] $T_2(n)$
...] $\Theta(1)$
...] $\Theta(1)$
...] $\Theta(1)$] $T_3(n)$
...] $\Theta(1)$
...] $\Theta(1)$
else
...] $\Theta(1)$

$T_{1worst}(n) = \Theta(n^2)$
$T_{1best}(n) = \Theta(1)$] from II

$T_{2best}(n) = \Theta(1)$
$T_{2worst}(n) = \Theta(n)$] from I

$T_{3best}(n) = \Theta(n) * T_{2best}(n) = \Theta(n)$

$T_{3worst}(n) = \Theta(n) * T_{2worst}(n) = \Theta(n^2)$

$T_{best}(n) = T_{1best}(n) + \underbrace{\Theta(1)}_{(else)} = \Theta(1)$

$T_{worst}(n) = T_{1worst}(n) + T_{3worst} = \Theta(n^2) + \Theta(n^2) = \Theta(n^2)$

- Querying the products that need to be supplied

```java
@Override
public void printProductsDetailed(){
    Branch[] currentBranch = company.getBranches();
    String supply = "";
    System.out.printf("%25s", "Product's Type");
    System.out.printf("%25s", "Product's Model");
    System.out.printf("%25s", "Product's Color");
    System.out.printf("%25s", "Product's Stock");
    System.out.printf("%25s", "Product's Need Supply\n");
    for(int i=0; i<company.getNumOfBranches(); i++){
        Product[] currentProduct = currentBranch[i].getProducts();
        for(int j=0; j<currentBranch[i].getNumOfProducts(); j++){
            if( currentProduct[j].getNeedSupply().equals("Need") ){
                supply = "Need to be supplied.";
            } else{
                supply = "No need to be supplied.";
            }
            System.out.printf("%25s", currentProduct[j].getType());
            System.out.printf("%25s", currentProduct[j].getModel());
            System.out.printf("%25s", currentProduct[j].getColor());
            System.out.printf("%25s", currentProduct[j].getStock());
            System.out.printf("%25s", supply);
            System.out.println();
        }
    }
}
```

- Querying the products that need to be supplied.

* printProductsDetailed function:

// branch number = n

// product number = m

$$
\begin{array}{l}
\ldots \;]\ \Theta(1) \\
\ldots \;]\ \Theta(1) \\
\ldots \;]\ \Theta(1) \\
\ldots \;]\ \Theta(1) \\
\ldots \;]\ \Theta(1) \\
\ldots \;]\ \Theta(1) \\
\left.\begin{array}{l}
\text{for} \ldots \\
\quad \ldots \;]\ \Theta(1) \\
\quad \left.\begin{array}{l}
\text{for} \ldots \\
\quad \text{if} \ldots \;]\ T_1(m) \\
\quad\quad \ldots \;]\ \Theta(1) \\
\quad \text{else} \\
\quad\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1) \\
\quad \ldots \;]\ \Theta(1)
\end{array}\right] T_2(m)
\end{array}\right] T_3(n,m)
\end{array}
$$

$T_{1\,best}(m) = \Theta(1)$] because of equals method
$T_{1\,worst}(m) = \Theta(m)$

the loop will execute m times

$T_{2\,best}(m) = \Theta(m) * T_{1\,best}(1) = \Theta(m^2)$

$T_{2\,worst}(m) = \Theta(m) * T_{1\,worst}(m) = \Theta(m^2)$

the loop will execute n times

$T_{3\,best}(n,m) = \Theta(n) * T_{2\,best}(m) = \Theta(n) * \Theta(m^2) = \Theta(nm^2)$

$T_{3\,worst}(n,m) = \Theta(n) * T_{2\,worst}(m) = \Theta(n) * \Theta(m^2) = \Theta(nm^2)$

$T_{best}(n,m) = T_{3\,best}(n,m) = \Theta(nm)$

$T_{worst}(n,m) = T_{3\,worst}(n,m) = \Theta(nm^2)$

$T(n,m) = \Omega(nm) = O(nm^2)$

Part 2:

Part 2

a) Definition of big oh notation:

$T(n) = O(f(n))$ if there are positive constents c and n_0 such that

$T(n) \le c f(n)$ when $n \geq n_0$

It is meaningless to say: "The running time of algoritm A is at least $O(n^2)$".

the running time $=$ $T(n)$ and $f(n) = n^2$

The running time grows no faster than n^2. $f(n)$ is an upper bond on $T(n)$. The running time of algoritm A is at most $O(n^2)$.

b) $\max(f(n), g(n))$ can be $g(n)$ or $f(n)$.

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

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

$$\frac{f(n)+g(n)}{2} \geq \max(f(n)+g(n)) \geq \frac{f(n)+g(n)}{2}$$

$f(n)+g(n)$ can be $g(n)$ or $f(n)$

$g(n) \geq f(n)+g(n) \geq g(n)$

$f(n) \geq f(n)+g(n) \geq f(n)$.

$$\frac{f(n)+g(n)}{2} \geq f(n)+g(n) \geq \frac{f(n)+g(n)}{2}$$

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

c) 1. $f(n) = 2^{n+1}$

$g(n) = 2^n$

$2^{n+1} \leq c_1 . 2^n$, whenever $n > k$

$2^{n+1} \leq 2 . 2^n$, whenever $n > 5$

$\Rightarrow 1 \leq 1$, whenever $n > 5$ ✓

$2^{n+1} \geq c_2 2^n$, whenever $n > k$

$2^{n+1} \geq 2 . 2^n$, whenever $n > 5$

$1 \geq 1$, whenever $n > 5$ ✓

$\Rightarrow 2^{n+1}$ is $\Theta(2^n)$.

Note: Definition of theta notation:

A function $f(n)$ is $\Theta(g(n))$ if

$f(n) \leq c_1 . g(n)$ whenever $n > k$ (big oh)

and

$f(n) \geq c_2 g(n)$ whenever $n > k$ (omega) where c_1, c_2, k are positive.

Based on this definition, I solved questions.

II. $f(n) = 2^{2n}$

$g(n) = 2^n$

$2^{2n} \geq c_1 2^n$, wherever $n > k$

$2^{2n} \geq 2.2^n$, wherever $n > 5$

$2^n \geq 2$, wherever $n > 5$ ✓

But, $2^{2n} \leq c_2 . 2^n$, wherever $n > k$

$2^{2n} \leq 2.2^n$, wherever $n > 5$

$\boxed{2^n \leq 2, \text{ wherever } n > 5}$ is not true.

$\Rightarrow 2^{2n}$ is not $\Theta(2^n)$.

III. We can say that

$g(n) = \Theta(n^2) \Rightarrow g(n) \leq c_1 . n^2$ and $g(n) \geq c_2 . n^2$

and

$f(n) = O(n^2) \Rightarrow f(n) \leq c_3 . n^2$ and $f(n) \geq$? ? ?

For $g(n)$, both sides are written. But for $f(n)$, only one side is written. In this case, we canot multiply. And also Θ notation wants both side. but we do not have both sides. We can not multiply and we canot write the aswer with Θ notation.

Part 3:

Part 3

$$* \lim_{n\to\infty} \frac{n2^n}{2^n} = \infty \Rightarrow n2^n > 2^n$$

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

$$\left.\right\} n2^n > 2^n = 2^{n+1} \quad \text{I}$$

$$* \; 5^{\log_2 n} = n^{\log_2 5} \quad \text{and} \quad \log_2 5 \approx 2.32$$

$$\sqrt{n} = n^{1/2} = n^{0.5}$$

$$\Rightarrow 5^{\log_2 n} > n^{1.01} > \sqrt{n} \quad \text{II}$$

$$* \lim_{n\to\infty} \frac{n.2^n}{3^n} = \lim_{n\to\infty} n.\left(\frac{2}{3}\right)^n = \lim_{n\to\infty} \frac{n}{(3/2)^n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{1}{(3/2)^n \ln 3/2}$$

$$= \lim_{n\to\infty} \frac{2^n}{3^n \ln 3/2} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{2^n.\ln 2}{3^n.\ln 3.\ln 3/2} = \Big[\underbrace{\lim_{n\to\infty} \left(2/3\right)^n}_{0}\Big] \cdot \frac{\ln 2}{\ln 3 \cdot \ln 3/2} = 0.$$

$$\Rightarrow 3^n > n.2^n$$

$$\text{I} \rightarrow 3^n > n.2^n > 2^n = 2^{n+1} \quad \text{III}$$

$$* \lim_{n\to\infty} \frac{(\log n)^3}{\log n} = \lim_{n\to\infty} (\log n)^2 = \infty \Rightarrow (\log n)^3 > \log n \quad \text{III a}$$

$$* \lim_{n\to\infty} \frac{(\log n)^3}{n(\log n)^2} = \lim_{n\to\infty} \frac{\log n}{n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{n.\ln 10}}{1} = \lim_{n\to\infty} \frac{1}{n.\ln 10} = \frac{1}{\infty} = 0$$

$$\text{IIIa} \Rightarrow n(\log n)^2 > (\log n)^3 > \log n \quad \text{IV}$$

$$* \text{ compare } 5^{\log_2 n} \text{ and } 2^{n+1}$$

$$\lim_{n\to\infty} \frac{2^{n+1}}{5^{\log_2 n}} = \lim_{n\to\infty} \frac{2^{n+1}}{n^{2.32}} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{2.\ln 2.2^n}{2.32.n^{0.32}} = \left[\lim_{n\to\infty} \frac{2^n}{n^{0.32}}\right] \frac{2\ln 2}{2.32}$$

$$\overset{\text{L'Hospital}}{=} \left[\lim_{n\to\infty} \frac{\ln 2.2^n}{0.32\, n^{-0.68}}\right] \cdot \frac{2\ln 2}{2.32} = \Big[\underbrace{\lim_{n\to\infty} 2^n.n^{0.68}}_{\infty}\Big] \frac{2\ln^2 2}{2.32 \times 0.32} = \infty$$

$$\Rightarrow 2^{n+1} > n^{2.32} \underset{\text{II}}{\overset{\text{III}}{\Rightarrow}} \Rightarrow 3^n > n.2^n > 2^n = 2^{n+1} > 5^{\log_2 n} > n^{1.01} > \sqrt{n} \quad \text{V}$$

* compare $\sqrt{n}$ and $n\log^2 n$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{n\log^2 n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}.(\log^2 n)} = \frac{1}{\infty} = 0 \Rightarrow n\log^2 n > \sqrt{n} \quad A$$

* compare $n^{1.01}$ and $n\log^2 n$

$$\lim_{n\to\infty} \frac{n^{1.01}}{n\log^2 n} = \lim_{n\to\infty} \frac{n^{0.01}}{\log^2 n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{0.01 . n^{-0.99}}{2\log n \frac{1}{n\ln 10}} = \lim_{n\to\infty} \frac{0.01\, n^{0.01} . \ln 10}{2\log n}$$

$$= \lim_{n\to\infty} \frac{(0.01)^2 . n^{-0.99} . \ln 10}{2 . \frac{1}{n\ln 10}} = \lim_{n\to\infty} \frac{(0.01)^2 . n^{-0.99} . \ln^2 10 . n}{2} = \infty$$

$$\Rightarrow n^{1.01} > n(\log n)^2 \quad B$$

* compare $\sqrt{n}$ and $(\log n)^3$

$$\lim_{n\to\infty} \frac{\sqrt{n}}{(\log n)^3} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{2\sqrt{n}}}{\frac{3\log^2 n}{\ln 10\, n}} = \lim_{n\to\infty} \frac{\ln 10 . \sqrt{n}}{6\log^2 n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{\ln^2 10}{2\sqrt{n}}}{\frac{24}{\ln 10\, n}}$$

$$= \lim_{n\to\infty} \frac{\ln^3 10}{48} . \sqrt{n} = \infty \Rightarrow \sqrt{n} > (\log n)^3 \quad C$$

A B C I II

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

Part 4:

Part 4.

- void minValue (ArrayList<Integer> list) {

    Set min to first element of list] $T_1(n)$

    for i:=0 to n

        if min is greater than i.th element of list then] $T_2(n)$

            min = ith element of list] $T_3(n)$

    print min. $T_4(n)$

}

$T_1(n) = \Theta(1)$ Because of simple statement.

$T_2(n) = \Theta(1)$

$T_3(n) = \Theta(1)$

$T_4(n) = \Theta(1)$

Time complexity of for loop = $\Theta(n)$ because loop will execute n times.

$T_B = T_1(n) + \Theta(n) * T_2(n) + T_4(n) = \Theta(n)$

$T_W = T_1(n) + \Theta(n) * (T_2(n) + T_3(n)) + T_4(n) = \Theta(n)$

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

- void median (ArrayList <Integer> List) {

```
for i=0 to n
    Set min to zero ] Θ(1)
    Set max to zero ] Θ(1)
    for j=0 to n
Θ(1) [ if list's ith element is equal or greater than jth element of list then
Θ(1) [      add one to min
Θ(1) [ if list's ith element is equal or lower than jth element of list then
Θ(1) [      add one to max
    end for
Θ(1) [ if min equals to max
Θ(1) [      Set median to ith element of List
Θ(1) [ if i equals to one short of half of ith element of list then
Θ(1) [ Set median to ith element of list.
end for
}
```

* Time complexity will not affect whether to all if. Because of simple statements $(\Theta(1))$

* Outer loop's time complexity $= \Theta(n)$

Inner loop's time complexity $= \Theta(n)$

total $= \Theta(n) * \Theta(n) = \Theta(n^2)$

* $T(n) = \Theta(n^2)$. //ignore all $\Theta(1)$.

- void Sum (int value, ArrayList <Integer> list) {

for i=0 to n

for j=i+1 to n.

$\Theta(1)$ [if (i.th element of list + j.th element of list) equals to value then

$\Theta(1)$ [print i.th and j.th elements

}

Outer loop's time complexity = $\Theta(n)$

Inner loop's time complexity = $\Theta(n)$

total = $\Theta(n) * \Theta(n) = \Theta(n^2)$

time complexity will not affect whether or not to enter if.
Because of simple statement ($\Theta(1)$).

$T(n) = \Theta(n^2)$ //ignore all $\Theta(1)$'s

- void merge (Arraylist<Integer> List1, ArrayList<Integer> List2) {

Create singleList as an arraylist.] $\Theta(1)$

Add list1 to single List] $\Theta(n)$ //size of List 1 = n

Add list2 to single List.] $\Theta(n)$ //size of List2 = n.

Set temp to zero] $\Theta(1)$

for i=0 to 2n

for j = i+1 to 2n

$\Theta(1)$ [if ith element of list greater than j th element of list //singleList

$\Theta(1)$ [temp is ith element of singleList

$\Theta(1)$ [singleList's ith element is singleList's jth element //set

$\Theta(1)$ [singleList's jth element is temp //set

}

* Time complexicty will not affect whether to enter if. Because of simple statement.

* Outer loop's time complexity = $\Theta(2n) = \Theta(n)$

Inner loop's time complexity = $\Theta(2n) = \Theta(n)$

total = $\Theta(n) * \Theta(n) = \Theta(n^2)$

* $T(n) = \Theta(n) + \Theta(n) + \Theta(n^2) \pm \Theta(n^2)$. //ignore all $\Theta(1)$

Note: I assume that time complexity of "adding a list to a single list" is $\Theta(n)$.

Part 5:

Part 5:

a)

```
int p_1 (int array []):
{
    return array[0] * array [2]     ] Θ(1) because this is a simple statement
}
```

to find time complexity

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

$S(n) = O(1)$ because there is no extra space is required.

b)

```
int p_2 (int array [], int n):
{
    int sum = 0                              ] Θ(1)
    for (int i = 0; i < n ; i = i+5)         ]
        sum += array [i] * array [i]  ] Θ(1) ] Θ(n) (I)
    return sum                               ] Θ(1)
}
```

to find time complexity

Time complexity of for loop is $\Theta(n)$. Because: (I)

The loop body will execute k-1 times, with i having the following values: 0, 5, 10, ... 5k until 5k is greater than n value.

$$5k > n$$

$$k > n/5 \Rightarrow \Theta(n)$$

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

$S(n) = O(1)$ because there is no extra space is required (did not create new array, did not do new memory allocation, etc.)

c) void p-3 (int array[], int n):   to find time complexity

```
{
    for(int i=0; i<n; i++){
        for (int j=1; j<i; j=j*2)
            printf ("%d", array[i] * array[j])   ] Θ(1)   ] Θ(logn) (I)   ] Θ(nlogn) (II)
    }
}
```

Time complexity of for loop (I) is $\Theta(\log n)$, because:

The loop body will execute k-1 times, with j having the following values: 1, 2, 4, 8, ... 2^k until k is greater than i

$2^k > n$

$k > \log_2(n)$ $\qquad \log_2(n) = \dfrac{\log n}{\log 2} \Rightarrow \Theta(\log n)$

(Because constants are not important)

Time complexity of for loop (II) is $\Theta(n\log n)$ because

The loop body will execute n times. The value of each turn is $\Theta(\log n)$. So, $\Theta(\log n) * \Theta(n) = \Theta(n\log n)$

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

$S(n) = O(1)$ because there is no extra space is required.
(did not create new array, new memory allocation, etc.)

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

```
{
        If (p_2(array,n) > 1000)] T1(n)
             p_3(array, n) ] T2(n)
        else
             printf("%d", p_1(array) * p_2(array,n)) ] T3(n)
}
```

I assume that answers of a,b,c are correct.

$T_1(n) = \Theta(n)$, because of calling p_2() func.

$T_2(n) = \Theta(n\log n)$, because of calling p_3() func.

$$T_3(n) = \underbrace{\Theta(1)}_{\text{because of calling p\_1()}} + \underbrace{\Theta(n)}_{\text{because of calling p\_2()}} = \Theta(n)$$

- $T_W(n) = T_1(n) + \max(T_2(n) + T_3(n)) = \Theta(n) + \Theta(n\log n) = \Theta(n\log n)$
- $T_B(n) = T_1(n) + \min(T_2(n) + T_3(n)) = \Theta(n) + \Theta(n) = \Theta(n)$
- $T_{av}(n) = p(T).T_2(n) + p(F).T_3(n) + T_1(n)$

  if $p(T) = p(F) = 1/2$

$$T_{av} = \frac{1}{2}\Theta(n\log n) + \frac{1}{2}.\Theta(n) + \Theta(n) = \Theta(n\log n)$$

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

$S(n) = O(1)$, because there is no extra space is required. (did not create new array, not new memory allocation, etc.)