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.

II. Add/remove product.

III. Querying the products that need to be supplied.

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

1)

```java
public E at(int index) throws ArrayIndexOutOfBoundsException
{
    if(index < 0 || index >= size())
        throw new ArrayIndexOutOfBoundsException("Invalid index!");

    return array[index];
}
```

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

$T_n = \Theta(1)$

```java
public OfficeChairs search_chair(String model, String color,OfficeChairs chair) {

    for(int i=0;i<getCompany().getBranches().size();i++) {
        for(int j=0;j<getCompany().getBranches().at(i).getChairs().size();j++) {
            if(getCompany().getBranches().at(i).getChairs().at(j).equals(new OfficeChairs(model,color,null))) {
                chair = getCompany().getBranches().at(i).getChairs().at(j);
                return chair;
            }
        }
    }
    return null;
}
```

Let's branch.size = n, chair.size = m

GET AND EQUAL METHODS ARE CONSTANT!

AT METHODS IS CONSTANT

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

This for loop divided into 2.Best case and worst case

$T_b = \Theta(m)$, $T_w = \Theta(1)$ $\Big\} O(n)$

nested loop

$T(n,m) = \Theta(m) . O(n) = O(n.m)$

2)

```java
public boolean contains(E e) {
    for(Object i:array) {
        if(e.equals(i)) {
            return true;
        }
    }
    return false;
}
```

$\Theta(1)$ $\Theta(1)$ $\Big\} \Theta(n)$ $\Theta(1)$

$T_n = \Theta(n)$

```java
public boolean add(E e) {

    if(array == null) {
        array = (E[]) new  Object[size+1];
        size++;
        array[size-1] = e;
        return true;
    }
    if(this.contains(e)) {
        return false;
    }
    E[] temp = (E[]) new Object[size+1];
    for(int i=0;i<size;i++) {
        temp[i] = array[i];
    }
    temp[size] = e;
    array = temp;
    size++;
    return true;
}
```

$\Theta(1)$ $\Theta(1)$ $\to \Theta(1)$ $\to \Theta(1)$ $\Big\} \Theta(n)$ $\Theta(1)$ $\Big\} \Theta(n)$ $\Theta(1)$ $\Theta(1)$ $\Theta(1)$

$T_b = \Theta(1)$

$T_w = \Theta(n)$

$T_n = O(n)$

```java
public boolean addChair(String model,String color) {
    int index=0;
    container<Branches> branches= getCompany().getBranches();
    for(int i=0;i<branches.size();i++) {
        if(branches.at(i).getName().equals(getInformation())) {
            index = i;
        }
    }
    if(!getCompany().getBranches().at(index).getChairs().add(new OfficeChairs(model,color,getInformation()))) {
        for(int i=0;i<branch.getChairs().size();i++) {
            if(branch.getChairs().at(i).equals(new OfficeChairs(model,color,getInformation()))) {
                branch.getChairs().at(i).setQuantity(branch.getChairs().at(i).getQuantity()+1);
            }
        }
    }
    return true;
}
```

branch.size = n

chair.size = m

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

string equal $= \Theta(1)$

at method is constant

T_1: $\Theta(1)$

$\boxed{T_1 = \Theta(n) . \Theta(1) = \Theta(n)}$

This add method works like arraylist add method.It depends on chair size.

$T_b = \Theta(1)$, $T_w = \Theta(m)$ $> O(m)$

T_2: $\to \Theta(m)$ $\Theta(1)$

This at method and set method is constant. $\Theta(1)$

This get method is constant,

$\Big\} \Theta(m) . \Theta(1) = \Theta(m)$

return true $\Theta(1)$

$$\boxed{T_2 = \begin{cases} T_b = \underset{\text{if statement}}{O(m)} + \underset{\text{return}}{\Theta(1)} = O(m) \\ T_w = \underset{\text{if statement}}{O(m)} + \underset{\text{for loop}}{\Theta(m)} + \underset{\text{return}}{\Theta(1)} = O(m) \end{cases} \simeq O(m)}$$

$$\boxed{T_{n,m} = T_1 + T_2 = \Theta(n) + O(m) = O(n+m)}$$

It is lineer

```java
public boolean addChair(String model,String color,int quantity) {
    for(int i=0;i<quantity;i++) {
        addChair(model,color);
    }
    return true;
}
```

let's quantity = k

$\Theta(k)$, $O(n)$, $\Theta(1)$

$T(n,k) = \Theta(k). O(n) = O(k.n)$

It is lineer

```java
public boolean removeChair(String model,String color,int quantity) {
    int index=0;
    boolean flag=false;
    container<Branches> branches= getCompany().getBranches();
    for(int i=0;i<branches.size();i++) {
        if(branches.at(i).getName().equals(getInformation())) {
            index = i;
        }
    }

    for(int i=0;i<branch.getChairs().size();i++) {
        if(branch.getChairs().at(i).equals(new OfficeChairs(model,color,getInformation()))) {
            if(!(branch.getChairs().at(i).getQuantity() < quantity)) {
                branch.getChairs().at(i).setQuantity(branch.getChairs().at(i).getQuantity()-quantity);
                flag = true;
            }
        }
    }
    return flag;
}
```

let's branch.size = n , chairs.size = m

SET,GET AND EQUALS METHODS ARE COSNTANT!

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

T_1: $\boxed{T_1 = \Theta(n). \Theta(1) = \Theta(n)}$

$\Theta(m)$, $\Theta(1)$, $\Theta(1)$, $\Theta(1)$

T_2: $\boxed{T_2 = \Theta(m). \Theta(1) = \Theta(m)}$

$\boxed{T(n,m) = T_1 + T_2 = \Theta(n) + \Theta(m) = \Theta(n+m)}$

It is lineer

3)

Θ very

```java
public void addMessages(String message) {
    if(this.getCompany().getAdmin().getMessages().at(0).equals("")) {
        this.getCompany().getAdmin().getMessages().add(message);
    }
    else {
        this.getCompany().getAdmin().getMessages().add(message);
    }
}
```

$\rightarrow \Theta(1)$

$\rightarrow O(n)$

$O(n)$

$\boxed{T_n = O(n)}$

Part 2:

a) Explain why it is meaningless to say: “The running time of algorithm A is at least $O(n^2)$”. answer:the meaning of O(n^2) is the function's running time can be max n^2.Therefore,it is meaningless.

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)).
answer: Let's f(n) = Θ(n) and g(n) = Θ(n) :
max(f (n), g(n)) = Θ(n) and also Θ(f(n) + g(n)) = Θ(n) .So max(f (n), g(n)) = Θ(f(n) + g(n))

c) Are the following true? Prove your answer.

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

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

$$\lim_{n\to\infty} \frac{2^{n+1}}{2^n} = \frac{\infty}{\infty} = ? \Rightarrow \lim_{n\to\infty} \frac{\cancel{2^n}.2}{\cancel{2^n}} = 2$$

Yes,it is proved!

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

answer:

No,it isn't proved.

2^2n = O(2^n) --> it is true!

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

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \frac{\infty}{\infty} = ? \Rightarrow \lim_{n\to\infty} \frac{\cancel{2^n}.2^n}{\cancel{2^n}} = \infty$$

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

answer: f(n) = O(n^2) --> it is divided into 2.Best case and worst case.

Worst case: f(n) = Θ(n^2) and best case: f(n) = Θ(1).So ,f(n) * g(n) is also divided into 2.

Worst case : f(n) * g(n) = Θ(n^4) and best case: f(n) * g(n) = Θ(n^2).Therefore,

f(n) * g(n) = O(n^4), not f(n) * g(n) = Θ(n4).

Part 3:

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

n1.01, n(logn)^2, 2^n, √n, (log n)^3, n2^n, 3^n, 2n+1, 5 ^(log2 n), logn

Big-O	Name
O(1)	Constant
O(log n)	Logarithmic
O(n)	Linear
O(n log n)	Log-linear
O(n^2)	Quadratic
O(n^3)	Cubic
O(2^n)	Exponential
O($n!$)	Factorial

According to grow grates: Factoriel>exponential>cubic>quadratic>log-linear>linear>logaritmic>constant

Compare log:

$$\lim_{n\to\infty} \frac{\log^3 n}{\log n} = \frac{\infty}{\infty} = ? \lim_{n\to\infty} \frac{\cancel{\log n}.\log^2 n}{\cancel{\log n}} = \infty$$

$\log^3 n > \log n$

Compare polynomal:

$$\lim_{n\to\infty} \frac{n^{1.01}}{\sqrt{n}} = \frac{\infty}{\infty} \Rightarrow \lim_{n\to\infty} n^{1,01-0,5} = \lim_{n\to\infty} n^{0,51} = \infty$$

$n^{1.01} > \sqrt{n}$

Compare Exponantial:

$3^n, n.2^n, 2^n, 2^{n+1}, 5^{\log_2 n}$

$$\lim_{n\to\infty} \frac{3^n}{n.2^n} = \frac{\infty}{\infty} = ? \Rightarrow \lim_{n\to\infty} \frac{(\frac{3}{2})^n}{n} = \infty$$

$3^n > n.2^n$

$$\lim_{n\to\infty} \frac{n.2^n}{2^n} = \frac{\infty}{\infty} = ? \lim_{n\to\infty} \frac{n.\cancel{2^n}}{\cancel{2^n}} = \infty$$

$n.2^n > 2^n$

$$\lim_{n\to\infty} \frac{2^n}{2^{n+1}} = \frac{\infty}{\infty} = ? \Rightarrow \lim_{n\to\infty} \frac{\cancel{2^n}}{\cancel{2^n}.2} = 0,5$$

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

$$\lim_{n\to\infty} \frac{2^n}{5^{\log_2 n}} = \frac{\infty}{\infty} = ? \Rightarrow \lim_{n\to\infty} \frac{2^n}{5^{\log_2 n}} = \infty$$

$2^n > 5^{\log_2 n}$

Answer:

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

Part 4:

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.

  Initialize minimum to first element of arraylist $\Theta(1)$

  Initialize counter to zero $\Theta(1)$

  While counter is less than to arraylist of size $\Theta(n)$

  if the counter is less than to minimum $\Theta(1)$

  set the minimum to the element of arraylist which is counterth $\Theta(1)$

  $T(n) = \Theta(1).2 + \Theta(n)$

  $\boxed{T(n) = \Theta(n)}$

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

  sort(arraylist) $\rightarrow T(n) = \Theta(n^2)$

  Initialize i to zero,Initialize j to zero $\Theta(1)$

  While i is less than to arraylist of size $\Theta(n)$

  While j is less than to arraylist of size-1 $\Theta(n)$

  if the j. element of arraylist is less than to (j+1). element of arraylist $\Theta(1)$

  swap j. element of arraylist , (j+1). element of arraylist $\Theta(1)$

  median(arraylist)

  sort arraylist $\Theta(n^2)$

  if size of arraylist mod 2 equal 1 $\Theta(1)$

  return element of the size of arraylist divide 2 $\Theta(1)$

  else

  return (arraylist.get(arraylist.size()) add arraylist.get(arraylist.size() divide 2)) divide 2 $\Theta(1)$

  $\boxed{T(n) = \Theta(n^2)}$

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

  Let K be given value.

  Initialize i to zero,Initialize j to zero $\Theta(1)$

  While i is less than to arraylist of size $\Theta(n)$

  While j is less than to arraylist of size $\Theta(n)$

  if i not equal j and add arraylist.get(i) arraylist.get(j) equal K $\Theta(1)$

  print "index" i arraylist.get(i) "index" j "arraylist.get(j) $\Theta(1)$

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

  $\boxed{T(n) = \Theta(n^2)}$

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

  Let arr and arr1 given values

  While i is less than to arr of size $\Theta(n)$

  add i. element of arr to arr1 $\Theta(n)$

  sort arr1 $\Theta(n^2)$

  $T(n) = \Theta(n).\Theta(n) + \Theta(n^2)$

  $\boxed{T(n) = \Theta(n^2)}$

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]) $\to \Theta(1)$

}

$\boxed{T(n) = \Theta(1)}$

$\boxed{S(n) = \Theta(1)}$

b)

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

{

Int sum = 0 $\to \Theta(1)$

for (int i = 0; i < n; i=i+5) $\to \Theta(\frac{n}{5})$

sum += array[i] * array[i]) $\to \Theta(3)$

return sum $\to \Theta(1)$

}

$T(n) = \Theta(1) + \Theta(\frac{n}{5}) \cdot \Theta(3) + \Theta(1)$

$\boxed{T(n) = \Theta(n)}$

$\boxed{S(n) = \Theta(1)}$

c)

void p_3 (int array[], int n):

{

for (int i = 0; i < n; i++) $\to \Theta(n)$

for (int j = 0; j < i; j=j*2) $\to \Theta(\log(n))$

printf("%d", array[i] * array[j]) $\to \Theta(1)$

}

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

$\boxed{T(n) = \Theta(n \cdot \log n)}$

$\boxed{S(n) = \Theta(1)}$

d)

$\boxed{S(n) = \Theta(1)}$

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

{

$S_1 \leftarrow$ If (p_2(array, n)) > 1000) $\to \Theta(n)$

$S_2 \leftarrow$ p_3(array, n) $\to \Theta(n \cdot \log n)$

else

$S_3 \leftarrow$ printf("%d", p_1(array) * p_2(array, n)) $\to \Theta(1)$

}

T_n :

$T_b = S_1 + \min(S_2 + S_3)$

$T_w = S_1 + \max(S_2 + S_3)$

$T_w = \Theta(n) + \Theta(n.\log n) = \Theta(n.\log n)$

$T_b = \Theta(n) + \Theta(1) = \Theta(n)$

$\boxed{T(n) = \Theta(n.\log n)}$