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

Refik Orkun ARSLAN

151044063

Part 1:

I. Searching a product. (best case : first find branch and officechair O(1))
(worst case :as shown below)

```
public officeChair searchProduct(cont<branch> branch,officeChairModel model,
officeChairColor color)    ==> O(branch.size() * getOfficeChair().size())
{                                number of branch    number of officeChair which branch

    for(int i=0 ; i < branch.size() ; ++i) ==>  O(branch.size())
    {


      for(int j=0 ; j < branch.at(i).getOfficeChair().size() ; ++j)
      {   ==>  O(getOfficeChair().size())
         if(branch.at(i).getOfficeChair().at(j).getOfficeChairModel()== model &&
        branch.at(i).getOfficeChair().at(j).getOfficeChairColor()== color )
        ==>O(1)
         {
            System.out.println("Found it"); ==> O(1)
            System.out.println("Branch :" + branch.at(i).getBranchCode()); ==> O(1)
            return  branch.at(i).getOfficeChair().at(j);  ==> O(1)
         }
         else
         {
            System.out.println(" NOT Found "); ==> O(1)



         }

      }
    }

    return null;  ==> O(1)
}
```

Methods in Function:

```
public T at(int index) throws ArrayIndexOutOfBoundsException ==> O(1)
{
    if(index < 0 || index >= size())   ==> O(1)
       throw new ArrayIndexOutOfBoundsException("Invalid index!");  ==> O(1)

    return content[index]; ==> O(1)
}
public int size() { return used; }    ==> O(1)
public int getBranchCode() { return branchCode; } ==> O(1)
```

II. Add product.

(best case :no fix capacity O(1))
*(worst case :*as shown below)

```
public boolean enterShipmentInformation(officeChairModel a,officeChairColor c)
{    O(temp.length && i < size())
   return getBranch().getOfficeChair().insert(new officeChair(a,c));
}
```

Methods in enterShipmentInformation:

```
public branch getBranch() { return branch; }
public cont<officeChair> getOfficeChair() {
   return officeChair;
}
public boolean insert(T newContent)  O(temp.length && i < size())
{                                        fix capacity array copy delay

   if(newContent == null || contains(newContent) >= 0)==> O(1)
      return false;==> O(1)

   fixCapacity(); ==> O(temp.length && i < size())
   content[size()] = newContent;  ==> O(1)

   setUsed(size()+1);==> O(1)

   return true;
}
```

Methods in insert:

```
protected void fixCapacity() ==> O(temp.length && i < size())
{
   if(size() == getCapacity())   ==> O(1)
      setCapacity(getCapacity()*2);  ==> O(temp.length && i < size())
}
```

Methods in fixCapacity:

```
public int getCapacity() { return capacity; }  ==> O(1)

private void setCapacity(int capacity)  ==> O(temp.length && i < size())
{
   if(capacity < 0)==> O(1)
      capacity = 1;==> O(1)

   T[] temp = (T[])new Object[capacity];   ==> O(1)
```

```
   for(int i=0 ; i < temp.length && i < size() ; ++i) ==> O(temp.length && i < size())
      temp[i] = at(i);

   setUsed(temp.length > size() ? size() : temp.length-1);  ==> O(1)
   content = temp; ==> O(1)
   this.capacity = capacity;   ==> O(1)
}
```

II. Remove product.(best case :find first element and remove O(1)) (worst case :as shown below)

```
public void removeShipmentInformation(officeChairModel a,officeChairColor c)
{
   int j;

   for( j=0 ; j < getBranch().getOfficeChair().size() ; ++j)
   {

      if(getBranch().getOfficeChair().at(j).getOfficeChairModel()== a &&
getBranch().getOfficeChair().at(j).getOfficeChairColor()== c )
      {
         erase(getBranch().getOfficeChair().at(j));==> O(size()^2)

      }


   }
}
```

Methods in removeShipmentInformation:

```
public branch getBranch() { return branch; }
public cont<officeChair> getOfficeChair() {
   return officeChair;
}
public T at(int index) throws ArrayIndexOutOfBoundsException ==> O(1)
{
   if(index < 0 || index >= size())   ==> O(1)
      throw new ArrayIndexOutOfBoundsException("Invalid index!");  ==> O(1)

   return content[index]; ==> O(1)
}
public officeChairColor getOfficeChairColor() { ==> O(1)
   return officeChairColor; ==> O(1)
}
public officeChairModel getOfficeChairModel() { ==> O(1)
   return officeChairModel; ==> O(1)
}
public boolean erase(T content)==> O(size()^2)
{
   if(content == null || contains(content) == -1)==> O(1)
      return false;==> O(1)

   boolean flag = true;==> O(1)

   for(int i=0 ; i < size() && flag; ++i)==> O(size())
      if(at(i).equals(content))  ==> O(size())
```

```
      {
         this.content[i] = at(size()-1);==> O(1)
         flag = false;==> O(1)
      }

   setUsed(size()-1);==> O(1)
   return true;
}
```

Methods in erase:

```
public boolean equals(Object o)  O(size())
{
   if(o == null)                          ==> O(1)
      return false;                  ==> O(1)
   if(!(o instanceof cont))     ==> O(1)
      return false;                  ==> O(1)

   cont<T> temp = ((con<T>)o); ==> O(1)

   if(size() != temp.size()) ==> O(1)
      return false;   ==> O(1)

   for(int i=0 ; i < size() ; ++i)  ==> O(size())
      if(content[i] != temp.content[i]) ==> O(1)
         return false;==> O(1)

   return true;==> O(1)
}
```

III. Querying the products that need to be supplied.

(best case :2 function best case O(1))

*(worst case :*as shown below)

```
public officeChair querying(cont<branch> branch,officeChairModel model, officeChairColor
color)    ==> O(branch.size() * getOfficeChair().size())
{                                                 number of branch    number of officeChair which branch

    for(int i=0 ; i < branch.size() ; ++i) ==>  O(branch.size())
   {


      for(int j=0 ; j < branch.at(i).getOfficeChair().size() ; ++j)
      {    ==>  O(getOfficeChair().size())
         if(branch.at(i).getOfficeChair().at(j).getOfficeChairModel()== model &&
         branch.at(i).getOfficeChair().at(j).getOfficeChairColor()== color )
         ==>O(1)
         {
            System.out.println("Found it"); ==> O(1)
            System.out.println("Branch :" + branch.at(i).getBranchCode()); ==> O(1)
            return  branch.at(i).getOfficeChair().at(j);  ==> O(1)
         }
         else
         {
            System.out.println(" NOT Found "); ==> O(1)
```

```
      }
                                   Search product
    }
  }
      if don't find enter
  enterShipmentInformation(model,color); O(temp.length && i < size())
}
```

Part 2:

a) Let T(n) be the running time for algorithm A and let a function f(n) = O(n^2). The
statement says that T(n) is at least O(n^2). That is, T(n) is an upper bound of f(n).
Since f(n) could be any function \smaller" than n^2 (including constant function), we
can rephase the statement as \The running time of algorithm A is at least constant."
This is meaningless because the running time for every algorithm is at least

constant

b) $f(n) \leq f(n) + g(n)$ and $g(n) \leq f(n) + g(n)$

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

$$f(n) + g(n) \leq 2\max(f(n),g(n))$$

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

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

$$\max(f(n),g(n)) = f(n) \text{ if } f(n) \geq g(n)$$

$$= g(n) \text{ if } g(n) \geq f(n)$$

c)

I. We can choose c = 2 and n_0 = 0, such that $0 \leq 2^{n+1} \leq c \times 2^n$ for all $n \geq n_0$. By

denition, $2^{n+1} = O(2^n)$. ***(True)***

İf $\lim_{n\to+\infty} f(n)/g(n)=c \in R$ then $f(n)=\Theta(f(n)+g(n))$

II. We can not find any c and n_0, such that $0 \leq 2^{2n} = 4^n \leq c \times 2^n$ for all $n \geq n_0$.

Therefore, $2^{2n} \,!= O(2^{2n})$. ***(False)***

III. Choose k = 1. ***(False)***

Assuming n > 1, then
f(n)/g(n)=n^2/ n^2 ≤ n^2/ n^2 = 1

Choose C = 1. Note that i n because n is the upper limit.

Part 3:

$$\lim_{x\to 999999} \frac{x^{1.01}}{x\log^2(x)} = 0.00601542 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{2^x} = 0 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{\sqrt{x}} = 1148.15 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{\log^3(x)} = 435.41$$

$$\lim_{x\to 999999} \frac{x^{1.01}}{x\,2^x} = 0 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{3^x} = 0 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{2^{x+1}} = 0 \qquad \lim_{x\to 999999} \frac{x^{1.01}}{x^{2.32192809488736}} = 1.3441\times 10^{-8} \qquad \lim_{x\to 999999} \frac{x^{1.01}}{\log(x)} = 83\,106.1$$

2^n = 3^n = 2^n+1 = n2^n> x^1.01> 5^ log2 n > nlog2n> √n > (log n)^3 > logn

----

3^n >n2^n >2^n+1=2^n>5^ log2 n >x^1.01 > nlog2n> √n >(log n)^3 > logn

Part 4:

Have to tour the array to find the minimum

- Find the minimum-valued item. Best case =O(n) Worst case=O(n)

```
SET Min to array[0]  ==> set array list O(1)
FOR i = 1 to array length – 1  ==> O(n)
  IF array[i] < Min THEN  ==> O(1)
    SET Min to array[i] ==> O(1)
  ENDIF
ENDFOR
PRINT Max   ==> O(1)
```

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

Best case =O(n) *the first checked element is the median, but it still takes time to check n time*

Worst case=O(n ^2)

```
SET array[n]   ==> O(1)
FOR i = 0 to array length – 1  ==> ==> O(n)
      FOR j = 0 to array length – 1==> O(n)
          IF i=j  ==> O(1)
            j++;
          ENDIF
          IF i>j ==> O(1)
            count++;
          ENDIF
      ENDFOR
      IF count==n/2 ==> O(1)
      return i ==> O(1)
 ENDFOR
```

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

Best case = O(1) *select first two number add given number*

Worst case=O(n ^2)

```
FOR i = 1 to n– 1  ==> O(n)
  For i+1 to n-1  ==> O(n)
    IF array[i]+array[j]==given value==> O(1)
            return 1 ==> O(1)
    ENDIF
  ENDFOR
ENDFOR
```

-Merge these two lists to get a single list in increasing order. O(n)

```
CREATE array c having size = array length a + aray length b

SET aa to 0  ==> O(1)
SET bb to 0  ==> O(1)
SET cc to 0  ==> O(1)
WHILE aa < length of a - 1 and bb < length of b – 1 ==> O(n)
  IF a [aa] < b [bb] THEN==> O(1)
    SET c[cc] to a[aa]==> O(1)
    aa++ ==> O(1)
    cc++ ==> O(1)
  ELSE
    SET c[cc] to b[bb]==> O(1)
     bb++ ==> O(1)
     cc++ ==> O(1)
  ENDIF
ENDWHILE
```

```
WHILE aa < length of a - 1==> O(n)
  SET c[cc] to a[aa]==> O(1)
   aa++ ==> O(1)
   cc++ ==> O(1)
ENDWHILE

WHILE bb < length of b - 1==> O(n)
  SET c[cc] to b[bb]==> O(1)
   bb++ ==> O(1)
   cc++ ==> O(1)
ENDWHILE

RETURN c
```

Part 5:

```
a) int p_1 (int array[]): ==> best case : O(1) / worst case : O(1)
{
      return array[0] * array[2]) ==> O(1)
              4 bytes        4 bytes   = 8 bytes    Space complexity =O(1)
}
b) int p_2 (int array[], int n): ==> best case :O(n)/ worst case : O(n)
{                              space Compexity = o(n) use array
      Int sum = 0 ==> O(1)
      for (int i = 0; i < n; i=i+5)  ==> O(n)
            sum += array[i] * array[i]) ==> O(1)
      return sum ==> O(1)
}
c) void p_3 (int array[], int n):==> best case /worst case :O(nlogn)
 {                             space Compexity = o(n) because use 1 array
      for (int i = 0; i < n; i++)==> O(n)
             for (int j = 0; j < i; j=j*2) ==> O(logn)
                   printf("%d", array[i] * array[j]) ==> O(1)
 }
d) void p_4 (int array[], int n): ==worst case : O(nlogn) / best case: o(n)else
                                                                      part
{                       space Compexity = o(n) because use 1 array
      If (p_2(array, n)) > 1000) ==> O(n)
             p_3(array, n) ==>O(nlogn)
      else
```

```
        printf("%d", p_1(array) * p_2(array, n)) ==>O(n)
}
```