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

Emre SEZER

1901042640

PART 1 : (Starts From Next Page)

```java
protected int countFurniture(Furniture [] list , Furniture item)
{
    int total = 0;
    for(int i = 0; i < list.length; i++)
    {
        if(item.equals(list[i]))    total++;
    }
    return total;
}
```

```java
protected boolean searchFurniture (Furniture [] list, Furniture item, int num)
{

    if(countFurniture(list, item) < num)
    {
        return false;
    }
    else return true;
}
```

(n = list.length)

```java
protected int countFurniture(Furniture [] list , Furniture item)
{
    int total = 0;
    for(int i = 0; i < list.length; i++)
    {
        if(item.equals(list[i]))    total++;
    }
    return total;
}
```

$T_1(n) = \Theta(1)$

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

$T_3(n) = \Theta(n)$ [For Loop will be executed n times]

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

$$T(n) = T_1(n) + T_3(n) + T_4(n)$$

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

```java
protected boolean searchFurniture (Furniture [] list, Furniture item, int num)
{
    if(countFurniture(list, item) < num)
    {
        return false;
    }
    else return true;
}
```

$T_1(n) = \Theta(n)$ [I showed above countFurniture()' Time Complexity is Theta(n)]

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

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

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

```java
protected Furniture [] addFurniture(Furniture [] list, Furniture item, int num)
{
    int newSize = list.length + num;
    Furniture [] temp = new Furniture [newSize];
    System.arraycopy(list, 0, temp, 0, list.length);

    for(int i = 0; i < num; i++)
    {
        temp[list.length + i] = item;
    }

    System.out.println("You added " + num + " " + item);
    return temp;
}
```

```java
protected Furniture [] addFurniture(Furniture [] list, Furniture item, int num)
{
    int newSize = list.length + num;
    Furniture [] temp = new Furniture [newSize];
    System.arraycopy(list, 0, temp, 0, list.length);

    for(int i = 0; i < num; i++)
    {
        temp[list.length + i] = item;
    }

    System.out.println("You added " + num + " " + item);
    return temp;
}
```

(n = list.length)

$\Theta(1)$

$\Theta(1)$

$T_3(n) = O(n)$ (System.arraycopy's Time Complexity is O(n))

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

$T_5(m) = \Theta(m)$ (m = num)

$\Theta(1)$

$$T(n,m) = O(\max(m,n))$$

$$= O(m+n)$$

(O(m) when m>n and O(n) when m<n)

```java
protected Furniture [] removeFurniture(Furniture [] list, Furniture item, int num)  throws Error
{
    int counter = 0;
    int arrayCounter = 0;
    Furniture [] temp = new Furniture[list.length - num];
    int lim = list.length - num;

    if(countFurniture(list, item) < num)
    {
        throw new Error("There is not enough items");
    }
    else
    {
        for(int i = 0; i < list.length; i++)
        {
            if(counter < num && list[i].equals(item))
            {
                counter++;
            }
            else if(counter == num && list[i].equals(item))
            {
                temp[arrayCounter] = list[i];
                arrayCounter++;
            }
            else
            {
                temp[arrayCounter] = list[i];
                arrayCounter++;
            }
        }
    System.out.println("You removed " + num + " " + item);
    return temp;
}
```

```java
protected Furniture [] removeFurniture(Furniture [] list, Furniture item, int num)  throws Error
{
    int counter = 0;
    int arrayCounter = 0;
    Furniture [] temp = new Furniture[list.length - num];
    int lim = list.length - num;

    if(countFurniture(list, item) < num)
    {
        throw new Error("There is not enough items");
    }
    else
    {
        for(int i = 0; i < list.length; i++)
        {
            if(counter < num && list[i].equals(item))
            {
                counter++;
            }
            else if(counter == num && list[i].equals(item))
            {
                temp[arrayCounter] = list[i];
                arrayCounter++;
            }
            else
            {
                temp[arrayCounter] = list[i];
                arrayCounter++;
            }
        }
    System.out.println("You removed " + num + " " + item);
    return temp;
}

 }
```

$\Theta(1)$

$(n = list.length)(m = num)$

$\Theta(m)$

[I showed that Time Complexity of countFurniture()'s Time Complexity is Theta(m) before.]

$\Theta(1)$

$\Theta(n)$

$$T_B(m,n) = \Theta(m) + \Theta(1) + \Theta(1) = \Theta(m)$$

$$T_W(m,n) = \Theta(m) + \Theta(n) + \Theta(1) = \Theta(\max(m,n))$$

$$T(m,n) = O(\max(m,n)) = O(m+n)$$

```
protected void query(wish w)
{
    if(wishNum > 0)
    {
        wishNum++;
        wish [] temp = new wish[wishNum];

        for(int i = 0; i < wishNum - 1; i++)
        {
            temp[i] = wishList[i];
        }
        temp[wishNum - 1] = w;
        wishList = temp;
    }
    else
    {
        wishList = new wish[1];
        wishList[0] = w;
        wishNum++;
    }
}
```

(n = wishNum)

```
protected void query(wish w)
{
    if(wishNum > 0)                              Θ(1)
    {
        wishNum++;
        wish [] temp = new wish[wishNum];        Θ(1)

        for(int i = 0; i < wishNum - 1; i++)
        {
            temp[i] = wishList[i];               Θ(n)
        }
        temp[wishNum - 1] = w;
        wishList = temp;                         Θ(1)
    }
    else
    {
        wishList = new wish[1];
        wishList[0] = w;                         Θ(1)
        wishNum++;
    }
}
```

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

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

$$T(n) = O(n)$$

Part 2:

a) "The running time of algorithm A is at least $O(n^2)$" statement is meaningless. $T(n) = O(f(n))$ means $T(n)$ grows no faster than $f(n)$. We use Big O notation for declaring upper bound. When we say: $T(n) \geqslant O(n^2)$, we declare lower bound with Big O and we don't know the upper bound. If we know lower bound, but don't know the upper bound, we can't know how much will it take in worst case. That's why, that sentence is meaningless.

b) $\max(f(n), g(n)) = \Theta(f(n) + g(n))$. In order to prove that, i need to prove that: $\max(f(n), g(n)) = O(f(n) + g(n))$ and $\max(f(n), g(n)) = \Omega(f(n) + g(n))$.

$$\left.\begin{array}{l}\max(f(n), g(n)) \geqslant g(n) \\ \max(f(n), g(n)) \geqslant f(n)\end{array}\right\} \begin{array}{l}\max(f(n), g(n)) + \max(f(n), g(n)) \geqslant g(n) + f(n) \\ 2\max(f(n), g(n)) \geqslant f(n) + g(n)\end{array}$$

$$= \max(f(n), g(n)) \geqslant \frac{[f(n) + g(n)]}{2} \quad \left(\text{I can say } \tfrac{1}{2} \text{ is any constant named } c\right)$$

$$= \max(f(n), g(n)) \geqslant c(f(n) + g(n)) \rightarrow \text{This is } \Omega \text{ definition}$$

So, i proved that $\max(f(n), g(n)) = \Omega(f(n) + g(n))$

Emre Sezer
1901042640

$f(n) \leq f(n) + g(n)$
$g(n) \leq f(n) + g(n)$
$\left(\begin{array}{l} \max(f(n), g(n)) = f(n) \quad \vee \\ \max(f(n), g(n)) = g(n) \end{array} \right)$

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

$\max(f(n), g(n)) \leq 2(f(n) + g(n))$ (I can say that 2 is any constant named c)

$\max(f(n), g(n)) \leq c(f(n) + g(n))$ → This is Big O definition

So, i proved that $\max(f(n), g(n)) = O(f(n) + g(n))$

Since, i proved $\max(f(n), g(n) = \Omega(f(n) + g(n))$ and $\max(f(n), g(n)) = O(f(n) + g(n))$, i proved that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

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

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

Result is not equal to 0 or ∞. So, $2^{n+1} = \Theta(2^n)$. Statement is true.

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

$$\lim_{n \to \infty} \frac{2^{2n}}{2^n} = \lim_{n \to \infty} 2^{2n-n} = \lim_{n \to \infty} 2^n = \infty$$

Result is ∞. So, $2^{2n} \neq \Theta(2^n)$

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

$f(n) \leq c.n^2$, $c_1 n^2 \leq g(n) \leq c_2 n^2$

$f(n) . g(n) \leq c.c_2.n^4$, $f(n) g(n) \leq c_3 n^4$ (1)

I cannot say anything about lower bound. Then,

$f(n) * g(n) \neq \Theta(n^4)$.

From (1) $f(n) * g(n) = O(n^4)$.

Part 3: Functions: $n^{1,01}$, $n\log^2 n$, 2^n, $\sqrt{n}$, $(\log n)^3$, $n2^n$, 3^n, 2^{n+1}, $5^{\log_2 n}$, $\log n$

I will compare functions above with the formula: $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)}$

$\lim\limits_{n\to\infty} \frac{\log n}{(\log n)^3} = \lim\limits_{n\to\infty} \frac{1}{(\log n)^2} = 0$ $\log n = o((\log n)^3)$. $(\log n)^3$ grows strictly faster than $\log n$.

$\lim\limits_{n\to\infty} \frac{(\log n)^3}{\sqrt{n}} = 0$ $(\log n)^3 = o(\sqrt{n})$. $\sqrt{n}$ grows strictly faster than $(\log n)^3$.

$\lim\limits_{n\to\infty} \frac{\sqrt{n}}{n\log^2 n} = 0$ $\sqrt{n} = o(n\log^2 n)$. $n\log^2 n$ grows strictly faster than $\sqrt{n}$.

$\lim\limits_{n\to\infty} \frac{n\log^2 n}{n^{1,01}} = 0$ $n\log^2 n = o(n^{1,01})$. $n^{1,01}$ grows strictly faster than $n\log^2 n$.

$\lim\limits_{n\to\infty} \frac{n^{1,01}}{5^{\log_2 n}} = 0$ $n^{1,01} = o(5^{\log_2 n})$. $5^{\log_2 n}$ grows strictly faster than $n^{1,01}$.

$\lim\limits_{n\to\infty} \frac{5^{\log_2 n}}{2^n} = 0$ $5^{\log_2 n} = o(2^n)$. 2^n grows strictly faster than $5^{\log_2 n}$.

$\lim\limits_{n\to\infty} \frac{2^n}{2^{n+1}} = \frac{1}{2}$ $2^n = \Theta(2^{n+1})$. 2^{n+1} grows as fast as 2^n.

$\lim\limits_{n\to\infty} \frac{2^{n+1}}{n2^n} = 0$ $2^{n+1} = o(n2^n)$. $n2^n$ grows strictly faster than 2^{n+1}.

$$\lim_{n \to \infty} \frac{n 2^n}{3^n} = 0$$

$n2^n = o(3^n)$. 3^n grows strictly faster than $n2^n$.

With help of the calculations, I will list the functions according to their order of growth.

$$3^n > n2^n > 2^{n+1} = 2^n > 5^{\log_2 n} > n^{1,01} > n\log^2 n > \sqrt{n}$$

$$> (\log n)^3 > \log n$$

Emre Sezer
190104264O

PART 4:

a) Integer findMinimumElement (ArrayList<Integer> list)

Initialize "min" as an integer equals to list's 0'th element; $\} T_1(n)$

```
LOOP FOR i=0 IN 1 TO list's size
    IF list's i'th element < min   } T2(n)
        min = list's i'th element; } T3(n)
END IF;
END LOOP;
RETURN min; } T6(n)
END.
```

($T_2(n)$ and $T_3(n)$ together: $T_4(n)$; the whole loop: $T_5(n)$)

(n = list's size. For loop iterates n times. $T(n)$ is time complexity of the whole program.)

$T_1(n) = \Theta(1)$, $T_2(n) = \Theta(1)$, $T_3(n) = \Theta(1)$, $T_6(n) = \Theta(1)$

$T_{4W} = T_2(n) + T_3(n)$, $T_{4B} = T_2(n) = \Theta(1)$

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

$T_5(n) = \Theta(n) . \Theta(1) = \Theta(n)$ [Loop will not break at any point.]

[IN in the for loop means "increases by"]

$T(n) = T_1(n) + T_5(n) + T_6(n)$

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

Emre Sezer
1901042640

b) double findMedian (ArrayList<Integer> list)

Initialize "temp" as an empty ArrayList of integers.
Initialize "length" as an integer equal to list's size.) $T_1(n)$
Initialize "med" as a double equal to 0.

LOOP FOR i=0 IN 1 TO length
add list's i'th element to end of the temp.) $T_2(n)$) $T_3(n)$
END LOOP;

LOOP FOR i=0 IN 1 TO length
LOOP FOR J=0 IN TO length
$T_4(n)$(IF (J+1)'th element of temp < J'th element of temp
(Initialize X as an integer equal to temp's (J+1)'th element.
$T_5(n)$(Set temp's (J+1)'th element to temp's J'th element.) $T_7(n)$
Set temp's J'th element to X.
$T_6(n)$ END IF;
END LOOP;
END LOOP;

LOOP FOR i=0 IN 1 TO length
$\theta(1)$(IF i == length/2 AND length % 2 == 1
$\theta(1)$(med = temp's i'th element; BREAK;
END IF;
$\theta(1)$(ELSE IF i == length/2 AND length % 2 == 0 $T_8(n)$
$\theta(1)$(med = temp's i'th element + temp's (i−1)'th element;
med /= 2; BREAK;
END ELSE IF;
END FOR;
RETURN med;) $\theta(1)$ END.

Scanned with CamScanner

$T_1(n) = \Theta(1)$

Time complexity of adding an element to the end of the ArrayList is $\Theta(1)$. So, $T_2(n) = \Theta(1)$.

First for loop will work n times, $T_3(n) = \Theta(n)$

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

Time complexity of changing an element's value of an ArrayList is $\Theta(1)$. So, $T_5(n) = \Theta(1)$.

$T_{6B}(n) = T_4(n) = \Theta(1)$, $T_{6W}(n) = T_4(n) + T_5(n) = \Theta(1)$

$T_6(n) = \Theta(1)$

Inner for loop will work n times. $T_7(n) = \Theta(n)$.

Time complexity for outer for loop is $T_8(n)$. Outer for loop will work n times. $T_8(n) = \Theta(n^2)$.

Inside the last loop: For both if and else if cases Time complexity is $\Theta(1)$. Loop can break at any point. So, Time Complexity of the loop which is $T_9(n) = O(n)$.

$$T(n) = T_1(n) + T_3(n) + T_7(n) + T_9(n) + \Theta(1)$$
$$= \Theta(1) + \Theta(n) + \Theta(n^2) + O(n) + \Theta(1) = \Theta(n^2)$$

[n = list's size. IN in the for loops means "increases by".
T(n) is time complexity of the whole program.]

c) void findEqualSum(ArrayList<Integer list, int sum)

Initialize "check" as a boolean equals to false; $\} T_1(n) = \Theta(1)$

```
LOOP FOR i=0 IN 1 TO list's size
  LOOP FOR J=0 IN 1 TO list's size
    IF list's i'th element + list's J'th element == sum      } T2(n)
    AND i != J
      PRINT list's i'th element;
      PRINT list's J'th element;                             } T3(n) = Θ(1)
      check = true; BREAK;
    END IF;
  END LOOP;                                                  } T5(n)
    IF check == true } T6(n)
  BREAK; } T7(n)                                             } T8(n) = Θ(1)
END LOOP;                                                    } T9(n) = O(n^2)
END.
```

(n = list's size. For loops iterate n times. $T(n)$ is the time complexity of the whole program. IN in the for loops means "increases by".)

$T_1(n) = \Theta(1)$, $T_4(n)$ is time complexity for the if statement's condition is true.

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

$T_{4W}(n) = T_2(n) + T_3(n) = \Theta(1)$, $T_{4B}(n) = T_2(n) = \Theta(1)$

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

$T_{5B}(n) = \Theta(1)$, $T_{5W} = \Theta(n)$, $T_5(n) = O(n)$

$T_6(n) = \Theta(1)$, $T_7(n) = \Theta(1)$

$T_{8B}(n) = T_6(n) + T_7(n) = \Theta(1) + \Theta(1) = \Theta(1)$

$T_{8W}(n) = T_6(n) = \Theta(1)$, $T_8(n) = \Theta(1)$

For the worst case both loops iterate n times.
For the best case both loops iterate 1 time.

$T_{9W}(n) = \Theta(n^2)$, $T_{9B}(n) = \Theta(1)$

$T_9(n) = O(n^2)$

$T(n) = T_9(n) + T_1(n) = O(n^2) + \Theta(1) = O(n^2)$

d)

```
ArrayList<Integer> mergeTwoArraylist(ArrayList<Integer> list1, ArrayList<Integer> list2)

Initialize "temp" as an empty ArrayList of integers.   } T1(n) = Θ(1)
Declare "counter" as an integer.

        LOOP FOR i = 0 IN 1 TO list1's size
T3(n)   add list1's i'th element to the end of the temp;   ) T2(n)
        END LOOP;

        LOOP FOR i = 0 IN 1 TO list2's size

        set "counter" to size of temp - 1;   ) Θ(1)
                LOOP WHILE counter > 0
                IF list2's i'th element >= temp's counter'th element

                        IF counter != size of temp - 1   ) Θ(1)
                        add list2's i'th element to temp's (counter + 1)'th position.   ) T4(n)
T7(n)   T6(n)           END IF;                                          T8(n)     T9(n)

                        ELSE
                        add list2's i'th element to the end of the temp;   ) T5(n)
                        END ELSE;
                END IF;
                Decrease counter by 1;   ) Θ(1)

                END LOOP;

        END LOOP;

RETURN temp;   ) Θ(1)
END.
```

add's time complexity when adding element between is $O(n)$.

$T_2(n) = O(n)$, $T_3(n) = O(n^2)$ [add function will work n times.]

add's time complexity when adding end of the list is $\Theta(1)$.

$T_{6W}(n) = \Theta(1) + T_4(n) = \Theta(1) + O(n) = O(n)$

$T_{6B}(n) = \Theta(1) + T_5(n) = \Theta(1) + \Theta(1) = \Theta(1)$

$T_4(n) = O(n)$, $T_5(n) = \Theta(1)$, $T_6(n) = O(n)$

$T_{7B}(n) = \Theta(1)$, $T_{7W}(n) = \Theta(1) + T_6(n) = O(n)$

$T_7(n) = O(n)$

$T_8(n) = O(n^2)$

[while loop will iterate n to 2n times.]

$T_9(n) = O(n^3)$ [for loop will iterate n times]

$$T(n) = T_1(n) + T_9(n) + \Theta(1)$$
$$= \Theta(1) + O(n^3) + \Theta(1) = O(n^3)$$

[n = size of list1 = size of list2 . IN in the for loops means "increases by". T(n) is the Time Complexity of the whole program.]

Part 5:

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

Space Complexity:

Space = 4 bytes

Since, function doesn't copy array to another array.

Space Complexity = $O(1)$

[I didn't include input size to the Space Complexity. Because, in lesson our teacher said so. This note is valid for each 4 questions.]

Time Complexity:

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

Function just returns a multiplication.

Emre Sezer
1901042640

b) int p-2 (int array[], int n) {

```
int sum = 0;                                   } T_1(n)
for (int i=0; i<n; i=i+5)                       } T_3(n)
sum += array[i] * array[i];   } T_2(n)
return sum;  } T_4(n)
}
```

Space Complexity:

Space : sum - 4 bytes, i - 4 bytes, 4 bytes

Since, function doesn't copy array to another array.
Space Complexity = $O(1)$

Time Complexity:

$T_3(n) = \theta(\frac{n}{5}) = \theta(n)$, $T_2(n) = \theta(1)$

$T(n) = T_1(n) + T_3(n) + T_4(n)$
$T(n) = \theta(1) + \theta(n) + \theta(1) = \theta(n)$

c) 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]);
}
```

$\Big\} T_2(n)$

Space Complexity:

Space : i – 4 bytes , J – 4 bytes , 4 bytes

Since, function doesn't copy array to another array.
Space Complexity $= O(1)$

Time Complexity:

$T_2(n) = O(\log_2 n)$

$T(n) = \Theta(n) T_2(n) = \Theta(n) O(\log_2 n)$

$T(n) = O(n\log_2 n)$

(I changed J=0 to J=1 in the second for loop.)

d) void p-4 (int array[], int n) {

```
If (P_2 (array, n) > 1000)   } T_1(n)
    P_3 (array, n)           } T_2(n)
else
    printf("%d", P_1(array) * P_2 (array, n))   } T_3(n)
}
```

Space Complexity :

Since, function doesn't copy array to another array.

Space Complexity $= O(1)$

Time Complexity :

$$T_B(n) = T_1(n) + \min(T_2(n), T_3(n))$$

$$= T_1(n) + T_3(n) \quad , \quad T_3(n) = \theta(1) + \theta(n) = \theta(n)$$

$$= \theta(n) + \theta(n) = \theta(n)$$

$$T_W(n) = T_1(n) + \max(T_2(n), T_3(n)) = T_1(n) + T_2(n)$$

$$T_2(n) = O(n\log_2 n) \quad , \quad T_W(n) = \theta(n) + O(n\log_2 n) = O(n\log_2 n)$$

$$T(n) = O(n\log_2 n) = \Omega(n)$$

Scanned with CamScanner