CSE 222 Hw-2

Mehmet Hüseyin YILDIZ
200104004095

Part 1:

I. Searching a product

```
public Iterator<E> iterator()                                → θ(1)
    {
        return new Iterator<E>() {

            int current = -1;                                → θ(1)

            @Override
            public boolean hasNext()                         → θ(1)
            {
                return current < (used-1);                   → θ(1)
            }

            @Override
            public E next()                                  → θ(1)
            {
                current++;                                   → θ(1)
                return (E) array[current];                   → θ(1)
            }

            @Override
            public void remove()                             → θ(1)
            {
                E element = (E)array[current];               → θ(1)
                MyList.this.remove(element);                 → θ(1)
            }
        };
    }
```

```
public E find(E e) throws Exception {                    → Tw = θ(n), Tb = θ(1)  → O(n)
        Iterator<E> it = iterator();                     → θ(1)
        while(it.hasNext())                              → Tw = θ(n), Tb = θ(1)
        {
                E element = it.next();           → θ(1)
                if(element.equals(e))            → Tw = θ(1), Tb = θ(1)  → Ta = θ(1)
                        return element; →  θ(1)
        }
        throw new Exception("The given element not included");   →  θ(1)
    }


public Product getProductById(String id) throws Exception{      →  O(n)
        Identifiable product = new Product(id) {};               →  θ(1)
        try {
                return products.find((Product) product);}        →  O(n)
        catch (Exception e) {
                throw new Exception("Couldnt find the product with that id !");   →  θ(1)
        }

    }
```

II. Add / remove product

```
public void addProduct(Product product) {                → Tw = θ(n), Tb = θ(1)  → O(n)
        products.add(product);
    }

public boolean add(E e)
    {
        if(capacity == used)                             →  p(T) = 1/5 and p(F) = 4/5
                                                         →  Tw = θ(n), Tb = θ(1)
                increaseCapacity(5);  →  θ(n)

        array[used] = e;                                 →  θ(1)
        used++;                                          →  θ(1)
        return true;                                     →  θ(1)
    }
```

```
private void increaseCapacity(int size)                    →  θ(n)
    {
            int oldCapacity = capacity;                     →  θ(1)
            Object[] temp = array;                          →  θ(1)
            capacity += size;                               →  θ(1)
            array = new Object[capacity];                   →  θ(1)

            for(int i=0; i<oldCapacity; i++)                →  θ(n) (oldCapacity cant be 1)
                    array[i] = temp[i];                 →  θ(1)
    }
```

III. Querying the products that need to be supplied

```
public IContainer<Message> getProductInforms(){     →  θ(1)
            return repository.getMessages();        →  θ(1)
    }

public IContainer<Message> getMessages() {                  →  θ(1)
            return messages;                        →  θ(1)
    }
```

Part 2:

a) It's meaningless because big O notation is used for the worst scenario so it gives the biggest time probable.

b) The statement is false because θ is used for average case so its not maximum of f(n)+ g(n). Its average of f(n)+g(n).

c) 1) It is true because asymptotic notations say the growing rate of the function so in first if we get n as 1 we get the result as 4, if we get n as 2 the result is 8
so the growth rate is 2^n in first statement.

2) It is false because for 2^{2n} if we get n as 1, the result is 4 and if we get n as 2, the result is 16; for 2^n if we get n as 1, the result is 2 and if we get n as 2, the result is 4 so the grow rates are not same.

3) $f(n) = O(n^2)$ so we can not know it in theta notation. It may be $\theta(n^2)$ or it may be $\theta(n)$ or smaller so according to the statement since $g(n) = \theta(n^2)$ $f(n)$ must be $\theta(n^2)$ but as I said we can not prove it is $\theta(n^2)$. It may be smaller.

Part 3:

0) 3^n	$\lim n \to \infty$ $3^n / n2^n = \infty$ so 3^n is bigger
1) $n2^n$	$\lim n \to \infty$ $n2^{2n} / 2^{n+1} = \infty$ so $n2^n$ is bigger
2) 2^{n+1}	since $\log_2 n$ is very lower than n
3) $5^{\log 2n}$	
4) 2^n	2^n is bigger than $n\log^2 n$ because exponential bigger than linear
5) $n\log^2 n$	$\lim n \to \infty$ $n\log^2 n / (\log n)^3 = \infty$ so $n\log^2 n$ is bigger
6) $(\log n)^3$	let say n=8 then $(\log n)^3 = 9$ but $n^{1.01}$ = about 8.
7) $n^{1.01}$	$\sqrt{n} = n^{1/2}$ so $n^{1.01}$ is bigger
8) $\sqrt{n}$	$\sqrt{n} = n^{1/2}$ logarithmic is slower than exponential
9) logn	

Part 4:

```
min(elements)                                  → θ(n)
    min = elements.get(0)                      → θ(1)
    for i to n                                 → θ(n)
        if( min >= elements.get(i) )           → p(T) = 1/2 and p(F) = 1/2
                                               → Tw = θ(1), Tb = θ(1)
                                               → θ(1)
            min = elements.get(i)              → θ(1)

    return min                                 → θ(1)
```

```
median(elements)                               → θ(n²)
    medianIndex = n / 2                        → θ(1)
    for i to medianIndex                       → θ(n²)
        min = min(elements)                    → θ(n)
        elements.remove(min)                   → θ(n)
    median = min(elements)                     → θ(n)
return median                                  → θ(1)
```

twoElements(number) → $\theta(1)$

first = Math.random() * number → $\theta(1)$

second = number – first → $\theta(1)$

ArrayList mergeList(list1, list2) → $O((m+n)*max(m,n))$

index1 = 0 → $\theta(1)$

index2 = 0 → $\theta(1)$

ArrayList myList → $\theta(1)$

while (index1< list1.size && index2<list2.size)

if(list1.get(index1) < list2.get(index2)) → $O(m)$

myList.add(list1.get(index1++)) → $T_w = \theta(m)$, $T_b = \theta(1)$ → $O(m)$

else → $O(n)$

myList.add(list2.get(index2++)) → $T_w = \theta(n)$, $T_b = \theta(1)$ → $O(n)$

T_{all}

while (index1< list1.size) → $O(m)$

myList.add(list1.get(index1++)) → $T_w = \theta(m)$, $T_b = \theta(1)$ → $O(m)$

while(index2 < list2.size) → $O(n)$

myList.add(list2.get(index2++)) → $T_w = \theta(n)$, $T_b = \theta(1)$ → $O(n)$

return myList

There are two case of T_{all}. If list1(m) is bigger than list2(n) and list2(n) is bigger than list1(m)

$T_{all-1} = O((m+n)* m)$ $T_{all-2} = O((m+n)* n)$

$T_{all} = O((m+n)*max(m,n))$

Part 5:

a) int p_1 (int array[]): → $T=\theta(1)$, $S=\theta(1)$

```
{
    return array[0] * array[2])    → T=θ(1) , S= θ(1)
}
```

b) int p_2 (int array[], int n): → $T=\theta(n)$, $S=\theta(1)$

```
{
    Int sum = 0                          → T=θ(1)
    for (int i = 0; i < n; i=i+5)        → T=θ(n)
        sum +=array[i] * array[i])   → T=θ(1)
    return sum                           → T=θ(1)
}
```

c) void p_3(int array[], int n): → undefined , $S=\theta(1)$

```
{
    for (int i = 0; i < n; i++)                      → T=θ(n)
        for (int j = 0; j < i; j=j*2)            → undefined(infinity loop)
            printf("%d", array[i] * array[j])    → T=θ(1)
}
```

d) void p_4(int array[], int n): → $T=O(n) + \max(O(n), a)$, $S=\theta(1)$

```
{                                   → T=θ(n)
    If (p_2(array, n))> 1000)       → Tw=θ(n) + max( θ(n), a)
                                      ,Tb=θ(n) + min( θ(n), a)
        p_3(array, n)      → undefined → let say x
    else
        printf("%d", p_1(array)*p_2(array, n)) →  T=θ(n)
}
```