CSE 222 HW2

1.
```
public boolean _searchProduct(Product product) throws NullPointerException{
    boolean result = false; ⟹ O(1)
    for(int i=0 ; i <getBranc_number(); i++){ ⟹ O(n+1)
        result= Company.getBranches()[i].getProducts()._check_product(product);
    }                       O(1)           O(1)               O(m)
    return result; ⟹
}
```

getBranches()[i] ⟹ O(1)

getProducts() ⟹ O(1)

_check_product ⟹ * first for loop ⟹ $f(oc\_mn, oc\_mc) = \underbrace{oc\_mn}_{\text{chair model number}} \times \underbrace{oc\_mc}_{\text{chair mode color}}$

* second for loop ⟹ $f(od\_mn, od\_mc) = \underbrace{od\_mn}_{\text{desk model number}} \times \underbrace{od\_mc}_{\text{desk model color}}$

* third for loop ⟹ $f(mt\_mn, mt\_mc) \Rightarrow \underbrace{mt\_mn}_{\text{table model number}} \times \underbrace{mt\_mc}_{\text{table model color}}$

* forth for loop ⟹ $f(bc\_mn) = \underbrace{bc\_mn}_{\text{book cases model number}}$

* fifth for loop ⟹ $f(oca\_mn) = \underbrace{ocamn}_{\text{cablnet model number}}$

Search Product

Line	Step/exec	Freq	Total
1	1	1	1
2	2	(n+1)	n+1
3	2	2m.n	2mn
4	1	1	1

Total = 2mn + n + 3

T(m,n) = O(m.n)

2

```
  public  void _addProduct (Branch selected, Product newProduct){
                                                 n
1       for(int i=0 ; i<Company.getBranch_number() ; i++){=> O(n+1)
2           if ( Company.getBranches () [i] == selected){=> O(1)
3               Company.getBrances()[i].getProducts()._update(newProduct);
                        O(1)            O(1)           O(1)
            }
        }
  }

public void _update (Product product){
    setOffice_desk(product.office_desk); -> O(1)
    set Book_cases (product.book_cases); -> O(1)
    set Meetmg_tables (product.meeting_tables); O(1)
    setOffice_cabnets( products.office_cabinets); O(1)
    setOffice_chair (productso ffice_chair); O(1)
}
```

Add Product

step/exec	Freq	Total
2	n+1	2n+2
2	n	2n
2	n	2n

Total = 6n + 2

T(n) = O(n)

```
(3) public   booleen -inquireProduct(Branch selected, Product product){
                                 n
1       for (Branch i : Company.getBranches()){ => n+1
2           if(i == selected){ => O(1)
3               return  i.getProducts().-checkproduct(product);
                          O(1)              O(m)
            }
        }
        return false;
    }
```

Inquire Product

Line	Step /exec	Freq	Total
1	2	n+1	2n+1
2	2	n	2n
3	2	2m.n	2mn

Total = 2mn + 4n+1

$T(n,m) = O(mn)$

Part 2

a) The running time of algoritm A is at least $O(n^2)$ does not prove any proper answer. $\Theta(n^2)$ gives us the worst case. So at least has to be used Ω notation. $O(n^2)$ has to be used with maximus

c) I. $2^{n+1} = \Theta(2^n)$ $2^n * 2 = \Theta(2n)$ we can ignore constants

$2^n = \Theta(2n)$ True

II. $2^{2n} = \Theta(2^n)$ 2n grows two times faster than 2. not correct False

III. $f(n) = O(n^2)$ is true $f(n) = 5n^2 + 2n + 1$ $f(n) * g(n) = 25n^4 + 10n^3 + 5n^2$

$g(n) = \Theta(n^2)$ is true $g(n) = 5n^2$ $\Theta(n^4)$

True

Part 3

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

$n^{1.01}, n\log^2 n, 2^n, \sqrt{n}, (\log n)^3, n2^n, 3^n, 2^{n+1}, 5^{\log_2 n}, \log n$

1. $O(1) < \log n < n < n\log n < n^2 < 2^n < n!$

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

1 2 3 4 5 6 7 8 9

1-) $\log n < (\log n)^3 \Rightarrow (\log n)^3$ grows faster than $\log n$ for all $n \geqslant 10$
3 times

2-) $(\log n)^3 < \sqrt{n} \Rightarrow n$ always grows faster than logarithmic functions $n \geqslant 1$

3-) $\sqrt{n} = n^{1/2} < n^{1.01} \Rightarrow$ exponential values shows the growth rate

4-) $n^{1.01} < n\log_2 n =$

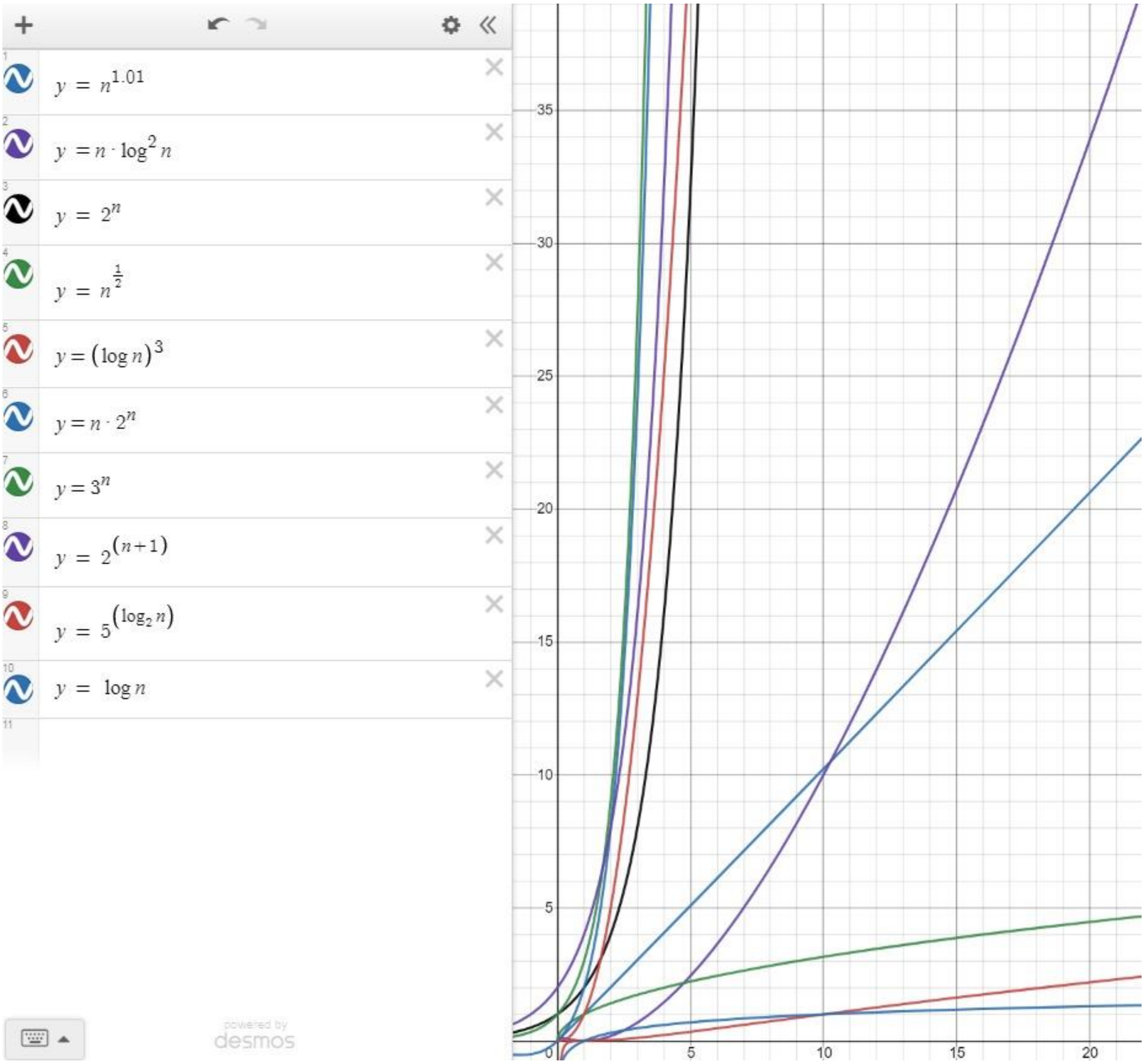
5-) $n\log_2 n < 2^n \Rightarrow 2^n$ always grows faster than $n\log_2 n$

6-) $2^n < 5^{(\log_2 n)} \Rightarrow$

7-) $5^{(\log_2 n)} < 2^{(n+1)}$

8-) $2^{(n+1)} < n.2^n \Rightarrow 2.2^n < n.2^n \Rightarrow 2 < n$ for

9-) $n.2^n < 3^n \Rightarrow$



1 $y = n^{1.01}$

2 $y = n \cdot \log^2 n$

3 $y = 2^n$

4 $y = n^{\frac{1}{2}}$

5 $y = (\log n)^3$

6 $y = n \cdot 2^n$

7 $y = 3^n$

8 $y = 2^{(n+1)}$

9 $y = 5^{(\log_2 n)}$

10 $y = \log n$

11

Part 4

(1) Find minunum value⇒ (arr[ ], size)

* checks if the size is zero ⇒ O(1)

* assign the firstindex of iten to variable min ⇒ O(1)

* search all the indexes ⇒ O(n)

* checks if the min is greater than indexes of array ⇒ O(n)

* return minumum ⇒ O(1)

T(n) = O(n)

(2) Find the median⇒ (arr[ ], size) 5, 6, 7, 4, 3, 1, 8

* order all the elements

* checks if the size is odd or even ⇒ O(1)

* return the median value ⇒ [size/2] or ([size/2]+[size/2+1])/2 O(1)

ordering

```
for (int i=0; i<size-1; i++){ ⇒ (n-1)
    for(int j=i+1; j<size; j++){ (n-1).logn      > (n-1)+(n-1).logn
        if ( arr[i] > arr[j]{ ⇒ (n-1).logn          O(n.logn)

        }

} }
```

T(n) = O(n.logn)

③ Find two elements whose sum is equal to a given value.

* declare a sum variable $\Rightarrow O(1)$
* search all the indexes $i=0$ array[i] $\Rightarrow n$
* search all the indexes inside of first loop $\underset{\text{array}[j]}{j=0}$ $\Rightarrow n^2$
* add two array[i] and array[j] and assign to sum $\Rightarrow O(1)$
* checks if sum is equal to given value. $\Rightarrow O(1)$

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

④ Assume there are two ordered list of n elements. Merge these two list to set a single list in increasing order.

* Assign three variable $i=1, j=1, k=1$
* in a while loop checks i smaller than first list size and j is smaller than second list size
* checks if first list's i is bigger than second list's j

$\downarrow$
$O(mn)$
$\downarrow$
$O(n\log m)$ or $O(m\log n)$

    * assign first list's i to new list's k $\Rightarrow O(1)$
    * increase i and k

* checks the other situation $\Rightarrow O(1)$

    * assign second lists j to new list's k
    * increase i and

* assigns the remaining first or second lists values to new list if the sizes are differen in a for loop $\Rightarrow O(m)$ and $O(n)$

$$T(n) = O(\underbrace{n}_{\text{first arry size}} \log \underbrace{m}_{\text{second arry size}}) \text{ or } O(n\log n)$$

Part 5

a)

```
int p-1(int array[])
{
    return array[0]*array[2])
}
```

line

return array[0]* array[2]	1+1	2

2 ⇒ O(1)

space conplexity ⇒ O(1)

b)

```
int p-2(int array[], int n)
{
    int sum=0  -> O(1) => space
    for (int i=0 ; i<n ; i=i+5)   0 to n   n×5
        sum += array[i]*array[i]
    return sum
}
```

cost	Freq	Tot.cost
c_1	1	c_1
c_2	$n/5+1$	$c_2(n/5+1)$.
c_3	$n/5$	$c_3(n/5)$
c_4	1	c_4

space conplexity = O(1)

$$T_{p\text{-}2}(n) = c_1 + c_2(n/5+1) + c_3(n/5) + c_4$$

$$= c_1 + c_2 \cdot n/5 + c_2 + c_3 n/5 + c_4$$

$$= n/5\,(c_2+c_3) + c_1 + c_2 + c_4$$

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

c)

```
void p-3(int array[], int n)
{
    for (int i=0 ; i<n ; i++)
        for (int j=0; j<i; j=j*2)   0 i (log n)
            printf("%d",array[j]*array[j])
```

cost	Freq	Tot-cost
c_1	$n+1$	$c_1(n+1)$
c_2	$(\log n/2).n$	$c_2(n.(\log n/2))$
c_3	log n	log n

i=0   j=0

i1=   j=∅

i=2   j=1

i=3   j=2

Space conplexity ⇒ O(1)