**Question 1**

In this question I used theta theorem.

I used theta because I do not have in keyboard theta key.

**1.Searching a product.**

public static int linearSearch(int arr[], int elementToSearch) {

for (int index = 0; index < arr.length; index++) n+1

{ if (arr[index] == elementToSearch) return index; } return -1; 1

}

**Theta(n+1) time complexity has worst case. = 0(g(n)) Q(g(n))**

**Theta(1) time complexity has best case.= Ω(g(1))**

**2.Add/remove product**

// Function to add x in arr

    public static int[] addX(int n, int arr[], int x)

    {

        int i; 1

        // create a new array of size n+1

        int newarr[] = new int[n + 1]; 1

        // insert the elements from

        // the old array into the new array

        // insert all elements till n

        // then insert x at n+1

        for (i = 0; i < n; i++) n+1

            newarr[i] = arr[i]; 1

        newarr[n] = x; 1

        return newarr; 1

    }

**Theta(n+5) time complexity has worst case.= 0(g(n)) Q(g(n))**

**Theta(1) time complexity has best case. .= Ω(g(1))**

public static int[] remove(int[] numbers, int target) {

int count = 0; 1

// loop over array to count number of target values.

// this required to calculate length of new array

for (int number: numbers) {n+1

if (number == target) { 1

count++; 1

}

}

// if original array doesn't contain number to removed

// return same array

if (count == 0) {1

return numbers;1

}

int[] result = new int[numbers.length - count]; 1

int index = 0; 1

for (int value : numbers) { m+3

if (value != target) { 1

result[index] = value; 1

index++; 1

}

}

        numbers = null; // make original array       1

      return result; 1

**Theta(n+m+9) we can use n=m (2n+9 =n) time complexity has worst case. = 0(g(n)) Q(g(n))**

**Theta(1) time complexity has best case.= Ω(g(1))**

**3.Querying the products**

public static int linearSearch(int arr[], int elementToSearch) {

for (int index = 0; index < arr.length; index++) n+1

{ if (arr[index] == elementToSearch && numberofproduct(arr[index] == 0) return index; } return -1; 2

}

**Theta(n+2) time complexity has worst case. = 0(g(n)) Q(g(n))**

**Theta(1) time complexity has best case.= Ω(g(1))**

**Question 2**

**a)**

0(n) gives maximum complexivity so we cannot say at least. So it is wrong.

**b)**

max(f (n), g(n)) = Θ(f(n) + g(n)).

PROVE 1:

Theta: “f(n) is Θ(g(n))” iff f(n) is O(g(n)) and f(n) is Ω(g(n))

Let f(n) = n+3

g(n) = n^2

Max(f(n),g(n)) = Θ(n2+n+3)= n2 ?

Max(f(n),g(n)) =O(n) = n2= n2 proved no matter what did you choose if I would choose f(n) also n2 result will be 2n2=2n2

PROVE 2:

1-) Since we are requiring both f and g to be asymptotically non-negative, suppose that we are past some n1 where both are non-negative (take the max of the two bounds on the n corresponding to both f and g. Let c1 = 0.5 and c2=1.

2-)0 ≤ 0.5(f(n) + g(n)) ≤ 0.5(max(f(n), g(n)) + max(f(n), g(n)))

3-)max(f(n), g(n)) ≤ max(f(n), g(n)) + min(f(n), g(n)) = (f(n) + g(n))

**c)**

1-

2 n+1 = Θ(2n)?

2n\*2= Θ(2^n )

They are same we ignore 2 in theta notation

2-?

2^ 2n = Θ(2^n )?

2^2n= 4^n

4^n != 2^n false

3-

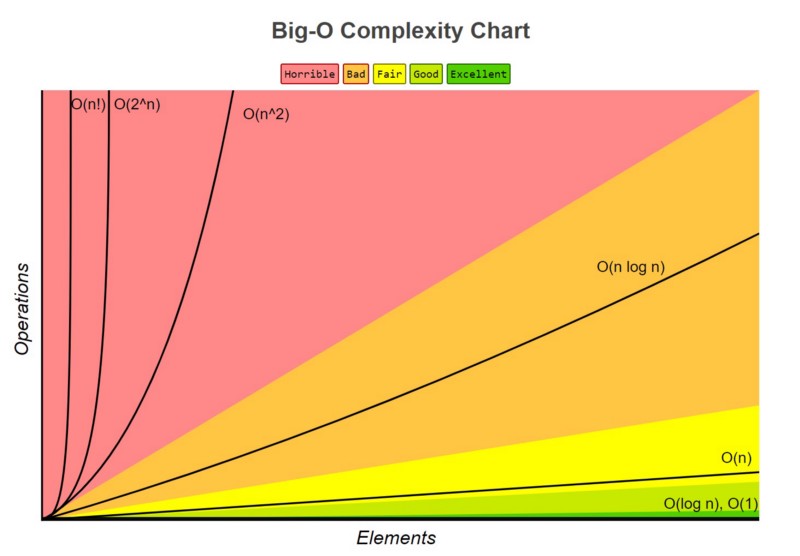
Let f(n)=O(n^2 ) and g(n)= Θ(n^2 ). Prove or disprove that: f(n) \* g(n) = Θ(n^4 )

If O(n^2) I can say f(n) =Θ(n^2 ). It is not wrong

So f(n)\*g(n) can be Θ(n^4 ). Proved

**Question 3**

We know normally this table



Prove for this we can do limit comparesion

Lim infinive n2n /3n from l’opital ln(2)n⋅2n+2n /3n = 0 so 3n is bigger

Lim infinive n1.01 / nlog2nfrom l’opital 1/ ln(n)(ln(n)+2) 1/ln(n) =0 so nlog2n is bigger

Etc.

So that 3n >n2n >2n+1 >2n > 5log2n>nlog2n > n1.01 > √n > (log n)3> logn

.

**Question 4**

Find the minimum-valued item

**1)**

SET minValue numbers[index 0] //T(1)

FOR i smaller than numbers.length //T(n)

IF numbers[i] < minValue

SET minValue numbers[i]

RETURN minValue //T(1)

Time complexity is O(n) worst case

Time complexity is Omega(1) best case

**2)**

SET m=0; // All of them T(1)

IF n%2==1

SET m a[(n+1)/2-1

ELSE

SET m=(a[n/2-1]+a[n/2])/2;

WHILE for each element in the array is it equal for m value // T(n)

       PRINT("Median :"+m);

Time complexity is O(n) worst case

Time complexity is omega(n) best case //no matter we are inside loop

**3)**

        SET l, r

        ARRAYS.SORT(A) //T(n)

        SET l to 0; //T(1)

        SET r to arr\_size - 1; //T(1)

        WHILE l < r // T(n)

            IF A[l] + A[r] == sum //T(1)

                RETURN true;

            ELSE IF(A[l] + A[r] < sum) //T(1)

                l++;

            ELSE

                r--; //T(1)

        RETURN false; //T(1)

Time complexity O(n) worst case

Time complexity Omega(n) best case // array sort

**4)**

**inputs** A, B : list

**returns** list

C := new empty list

WHILE A is not empty and B is not empty  **//T (n)**

IF head(A) ≤ head(B) //T(1)

append head(A) to C

drop the head of A

ELSE

append head(B) to C //T(1)

drop the head of B

RETURN C //T(1)

    Time complexity is O(n) worst case

Time complexity is Omega(1) best case // it can be not go to while loop

**Question 5**

In this question if we not have if else statement so we do not need best worse case.

**a)**

int p\_1 (int array[]): step/exec freq total

{ 2 1 2

return array[0] \* array[2]) Time complexity= T(n)=O(1)

} Space complexity=O(1)

**b)** step/exec freq total

int p\_2 (int array[], int n): theta(1)

{

Int sum = 0 theta(1) 1 1 1

for (int i = 0; i < n; i=i+5) theta(n) 2 n/5 +1 2n/5 +2

sum += array[i] \* array[i]) theta(1) 3 n/5 3n/5

return sum theta(1) 1 1 1

Time complexity =n+4=0(n)

Space complexity is 0(1)

}

**c)** step/exec freq total

void p\_3 (int array[], int n):

{

for (int i = 0; i < n; i++) 2 n+1 2n+2

for (int j = 1; j < i; j=j\*2) 2 n(logn) 2n(logn)

printf(“%d”, array[i] \* array[j]) 1 logn logn

T(n)=2n+2 +2nlogn+logn=0(nlogn)

Space complexity is 0(1)

}

**d)**

void p\_4 (int array[], int n):

{

If (p\_2(array, n)) > 1000) T3(n) = theta(n)

p\_3(array, n) T1(n)=theta(nlogn) all of them T(n)

else

printf(“%d”, p\_1(array) \* p\_2(array, n)) T2(n)=theta(n)

}

Tw(n) =T3(n)+max(T1(n),T2(n))=nlogn

Tb(n) =T3(n)+min(T1(n),T2(n))=n

Tav(n) = p(T) T1(n)+ p(F)T2(n)+T3(n)=nlogn all of them if p(T)=p(F)=1/2

P(T) P(condition true) T(n)=O(nlogn)

P(F) P(condition false) Omega(n)

Space complexity is 0(1)