Barış Ayyıldız

1901042252

**Part 1 :**

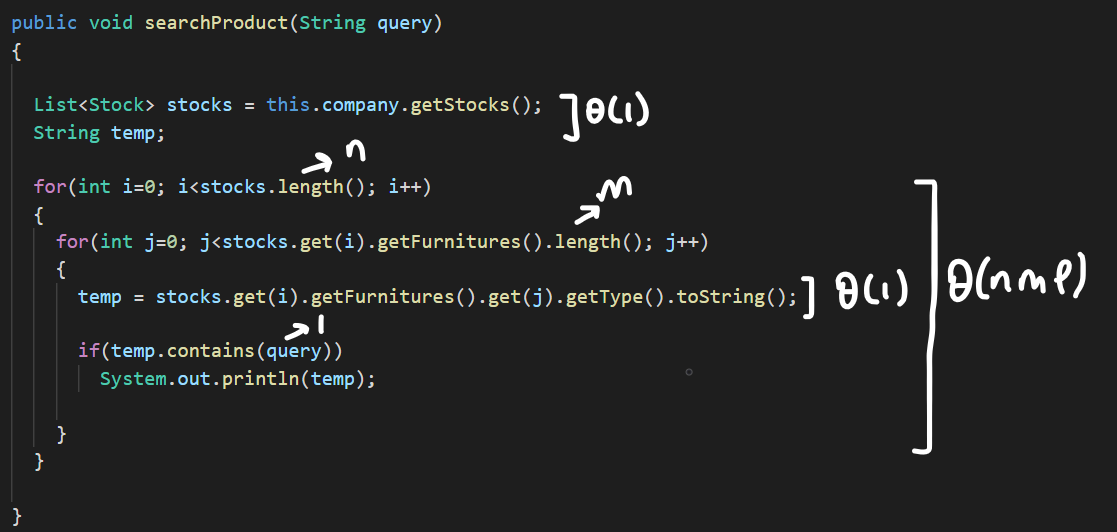
**I. Searching a product.**

Time complexity is : **Θ(n\*m\*l)**

n is length of stocks array,

m is length of furnitures array,

l is length of String query



**II. Add/remove product.**

**Add Products**

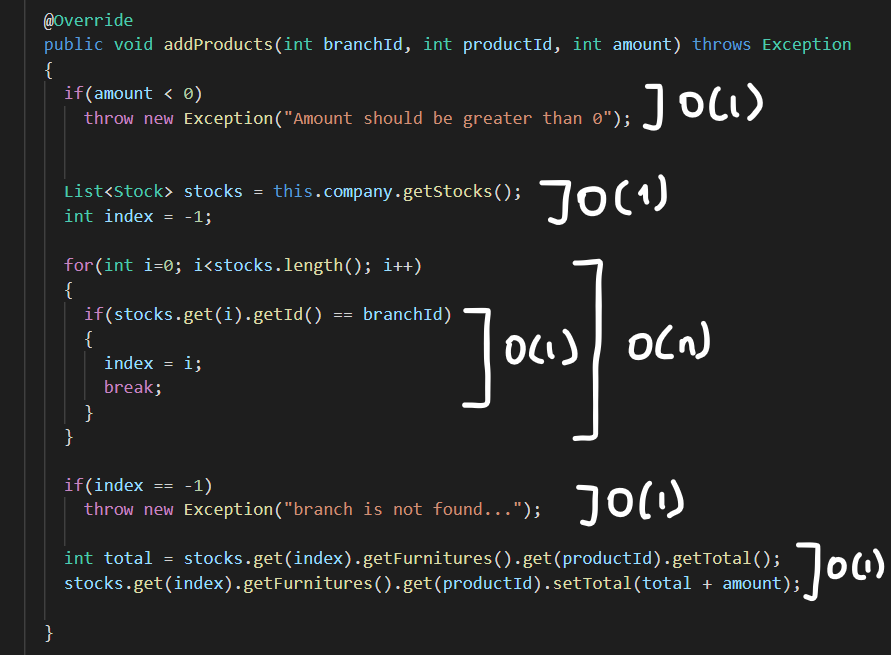
O(1)+ O(1)+ O(n)+ O(1) = **O(n)**

Since there is a breaking condition, the worst case is linear

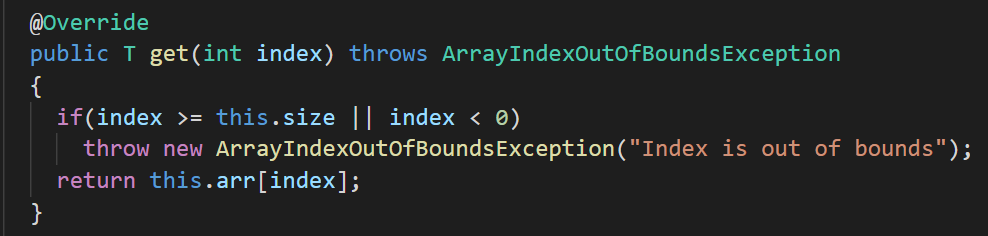
Tworst = **O(n)**

Tbest = **Ω(1)**, best case is constant time, when the algorithm iterates once inside for loop

n is equal to length of stock array in this case



All getters and setters take constant time, Θ(1)



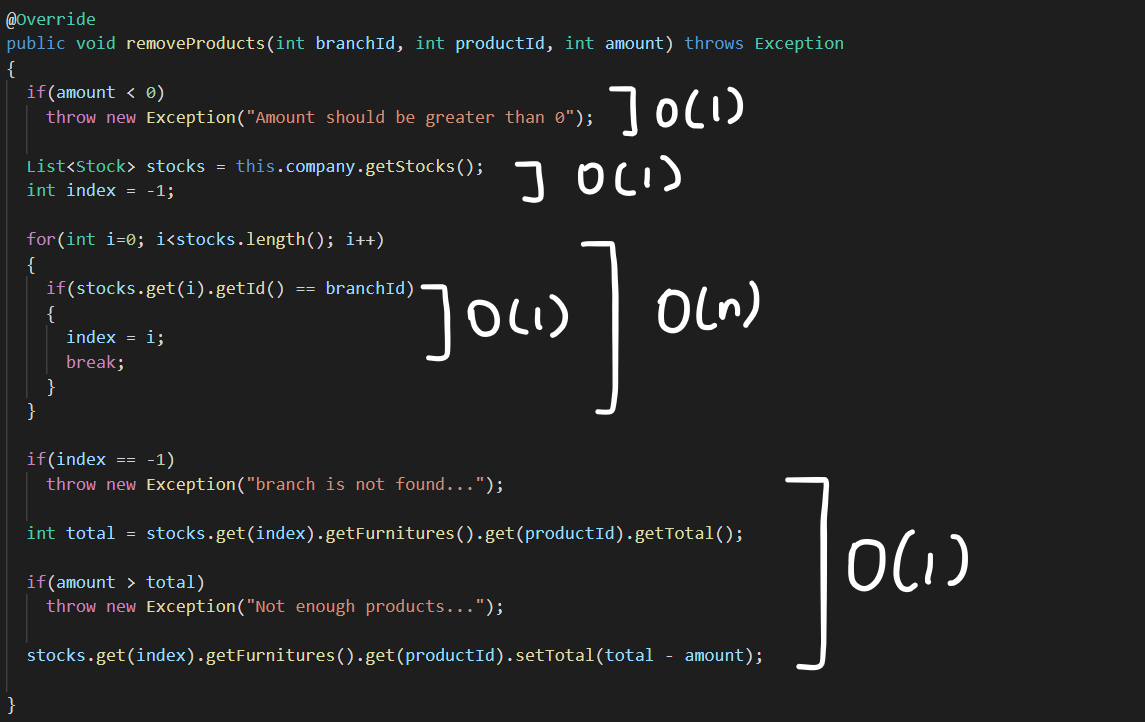
**Remove Products**

Since there is a breaking condition, the worst case is linear

Tworst = **O(n)**

Tbest = **Ω(1)**, best case is constant time, when the algorithm iterates once inside the for loop

n is equal to length of stock array in this case

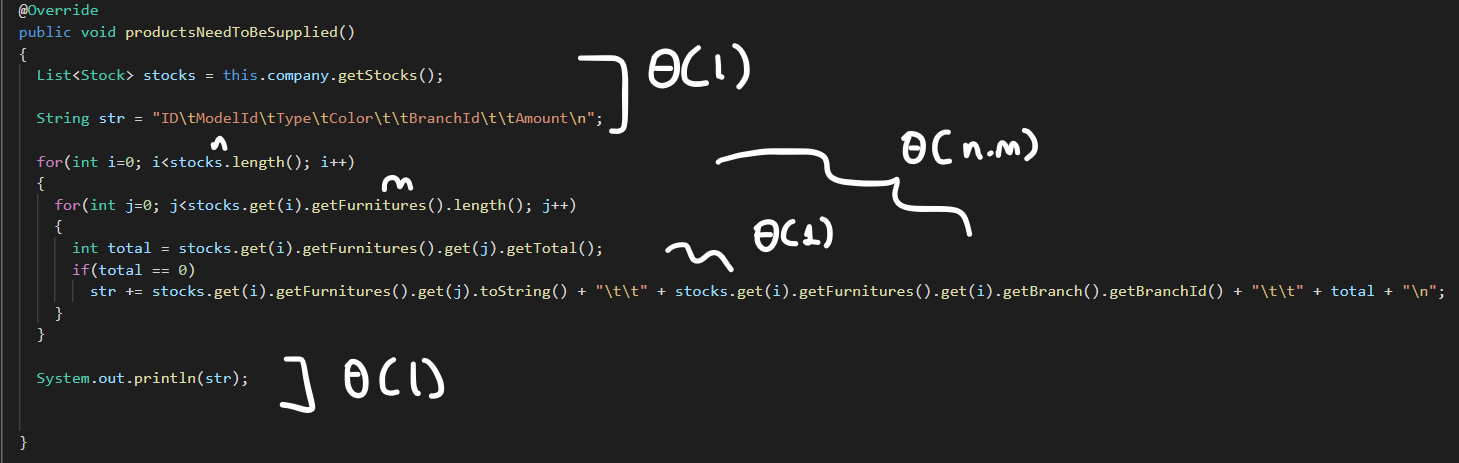


**III. Querying the products that need to be supplied.**

n is length of stocks, m is length of furnitures array

Since there is no breaking condition inside nested for loop

Θ(1) + Θ(n\*m)+ Θ(1) = **Θ(n\*m)**



All getters and setters take constant time, Θ(1)







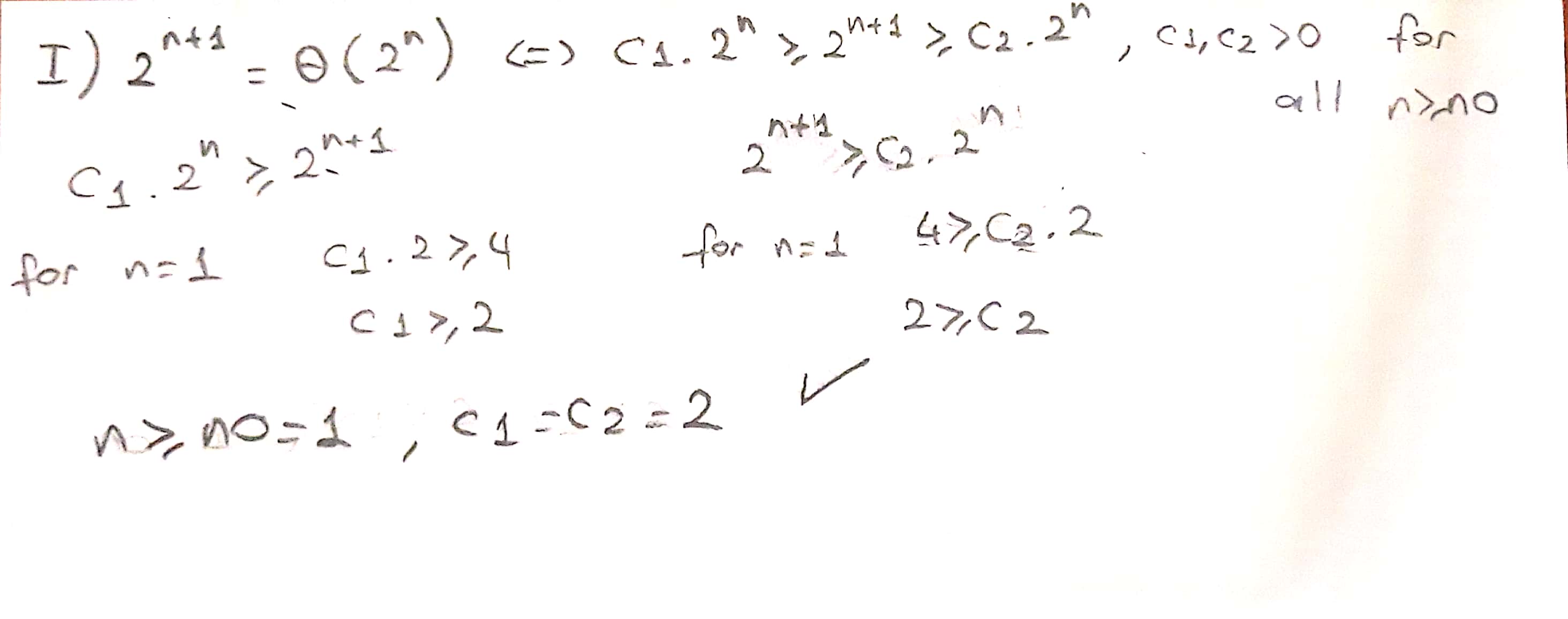
**Part 2 :**

1. Explain why it is meaningless to say: “The running time of algorithm A is at least O(n2 )”.

Because by definition, Big O notation refers to the worst case of an algorithm. So it is meaningless to say ‘at least’.

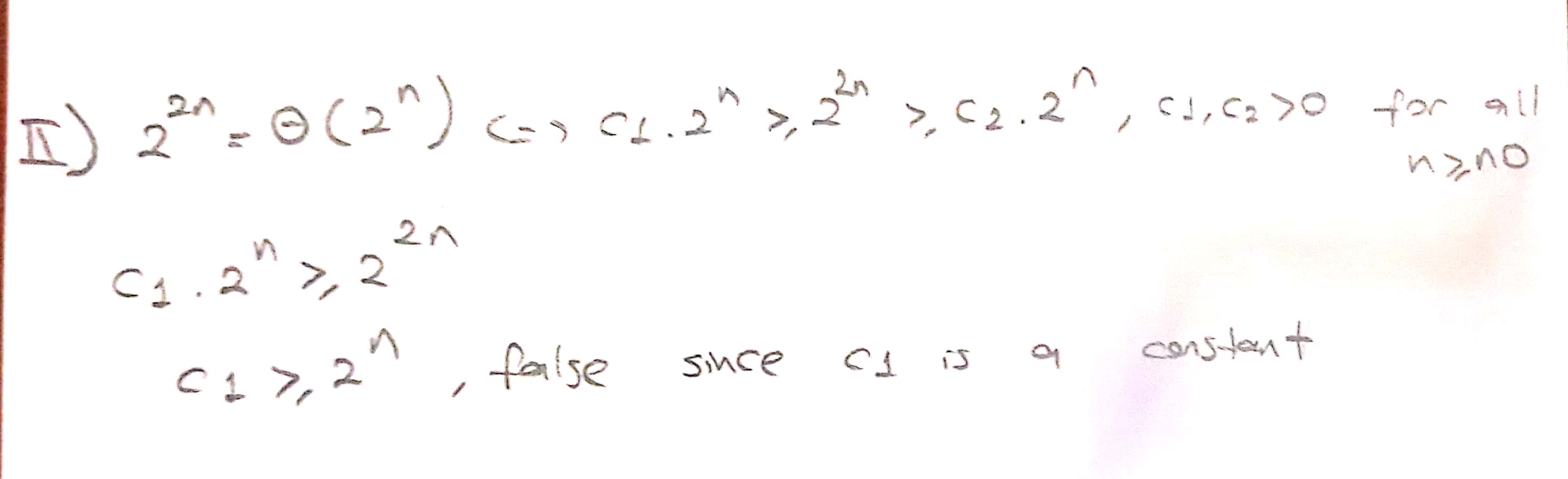
1. Let f(n) and g(n) be non-decreasing and non-negative functions. Prove or disprove that: max(f (n), g(n)) = Θ(f(n) + g(n)).
2. Are the following true? Prove your answer.
3. 2n+1= Θ(2n)

**true**



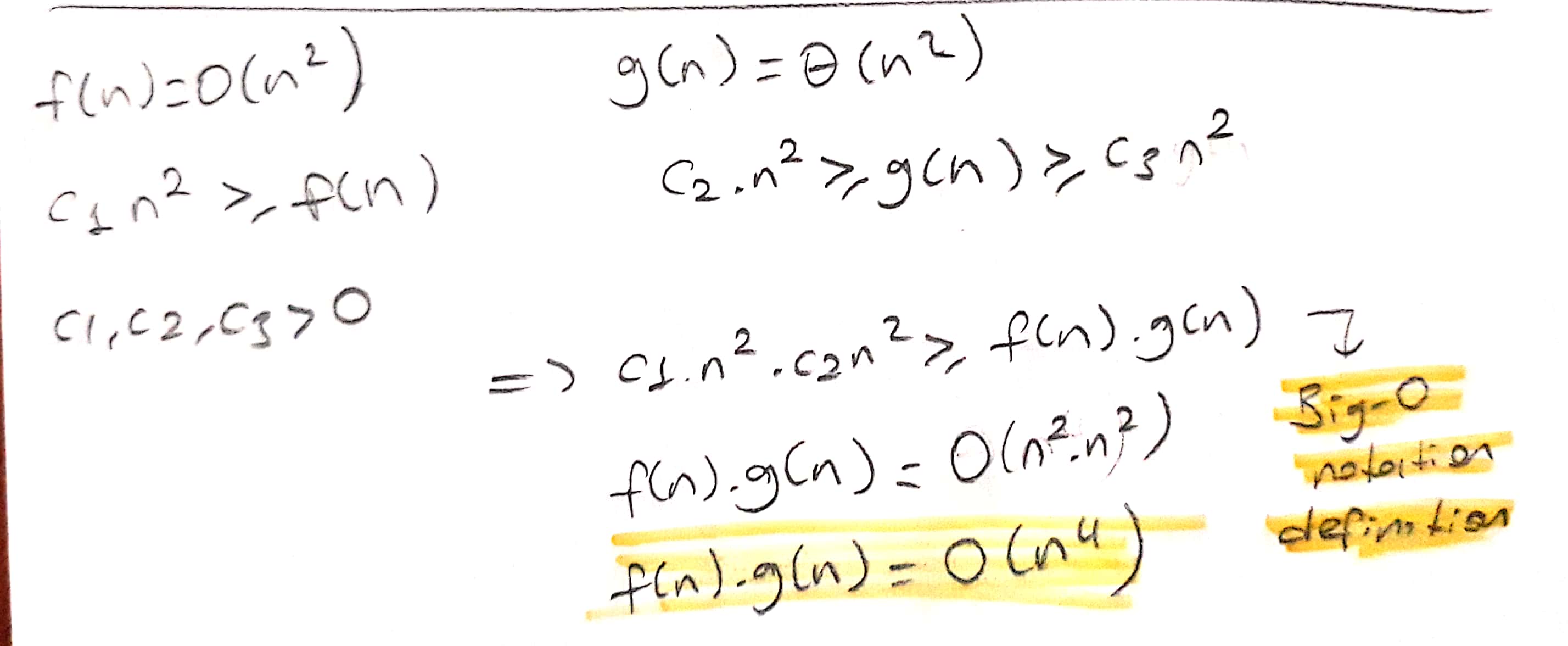
1. 22n= Θ(2n)

**false**



1. Let f(n)=O(n2 ) and g(n)= Θ(n2 ). Prove or disprove that: f(n) \* g(n) = Θ(n4 ).

**false**



Big-O notation doesn’t tell anything about the lower bound. So for time complexity**, it is not guarenteed to be Θ(n4)**. But we can say that:

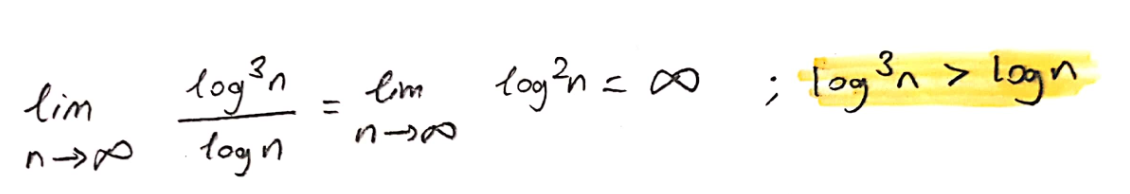
**f(n) \* g(n) = O(n4)**

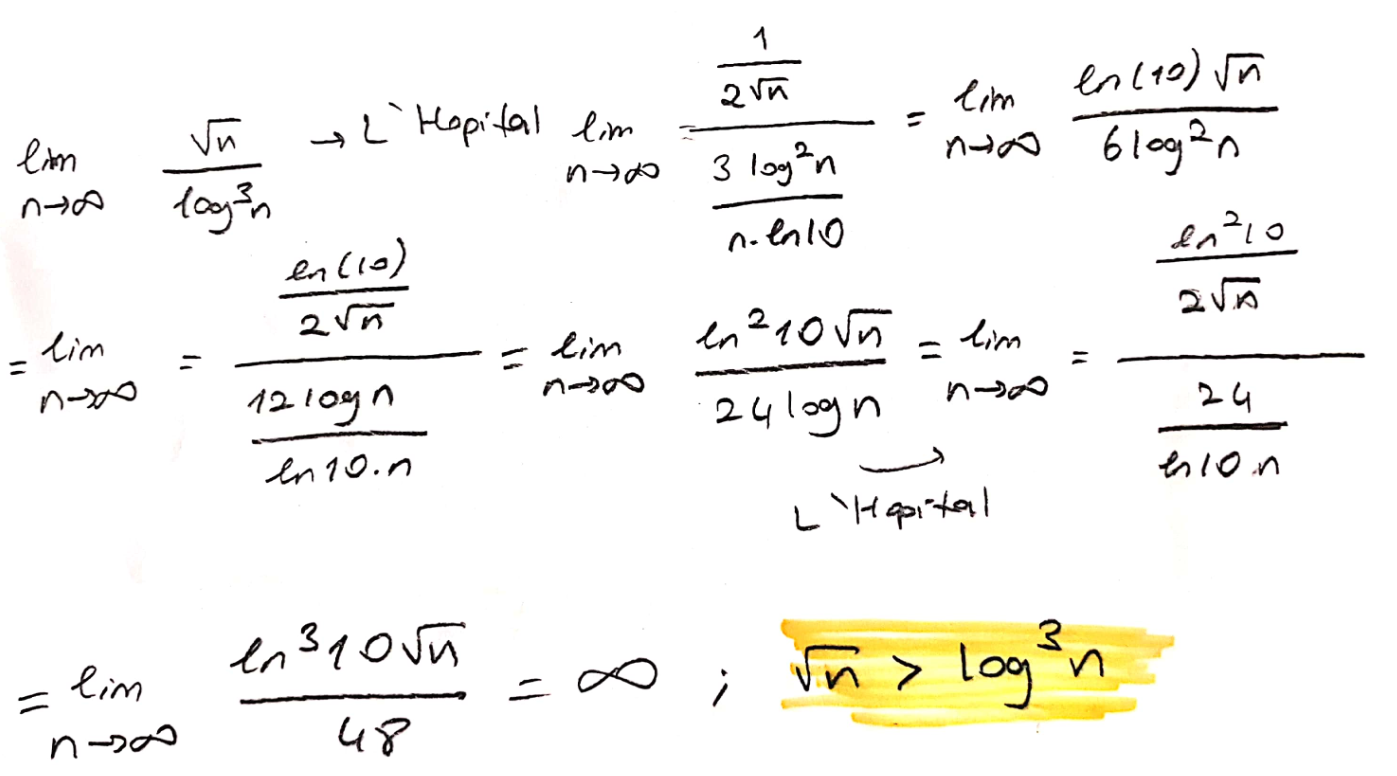
**Part 3 :**

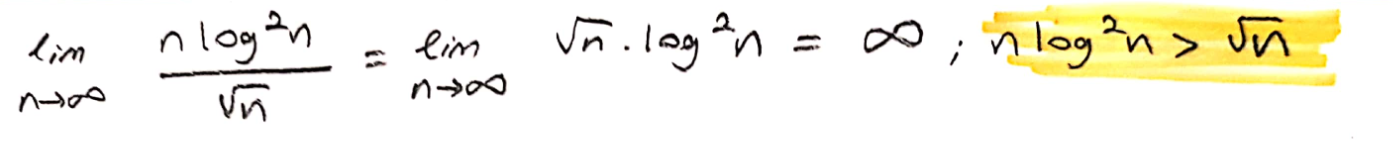
List the following functions according to their order of growth by explaining your assertions.

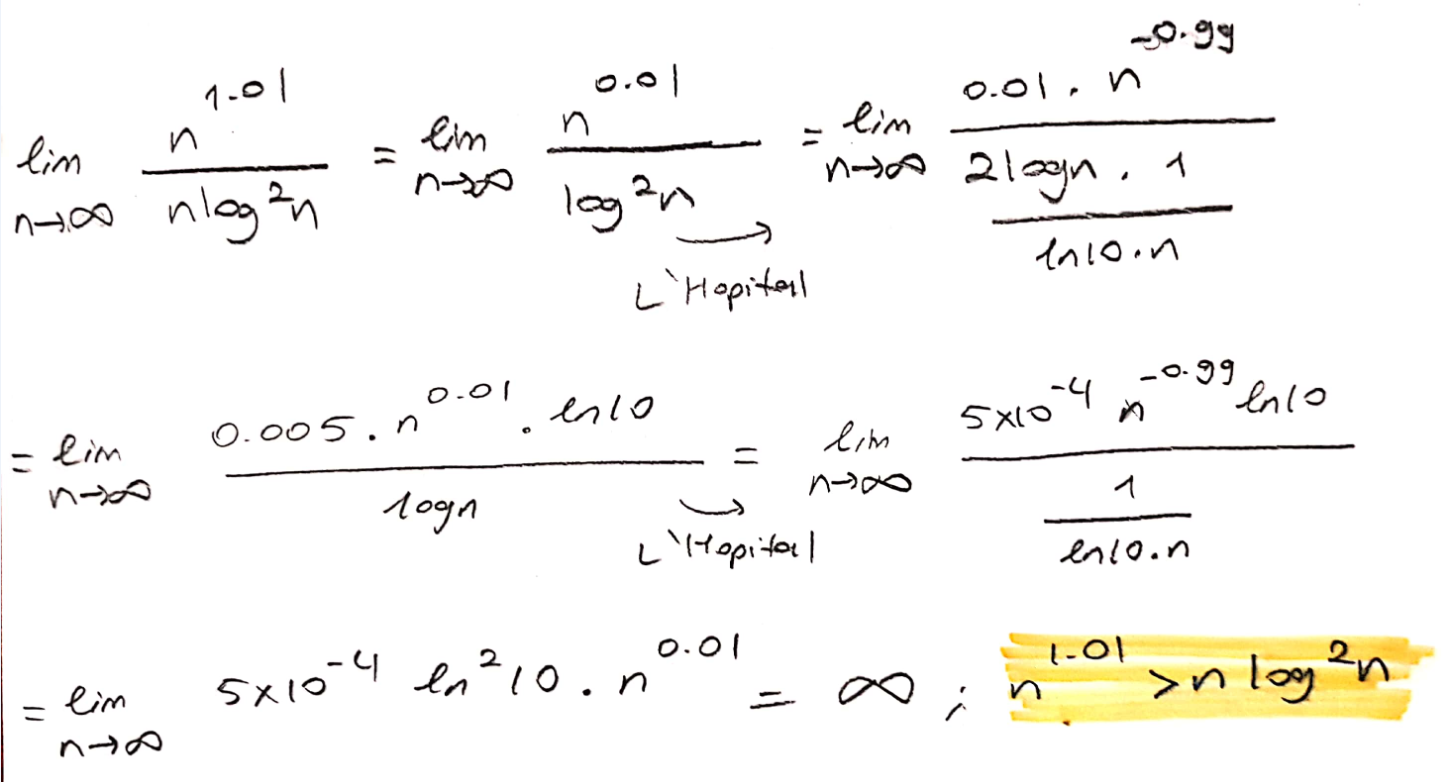
n 1.01, nlog2n, 2n , √n, (log n)3 , n2n , 3n , 2n+1, 5 log2 n , logn

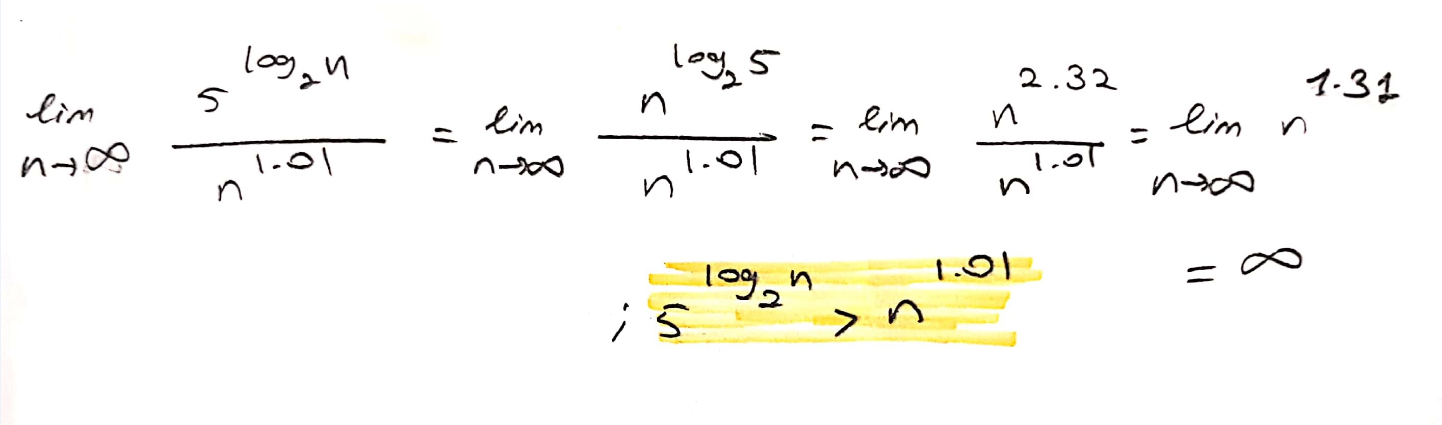
We can find growth rate by dividing two functions and taking it’s limit

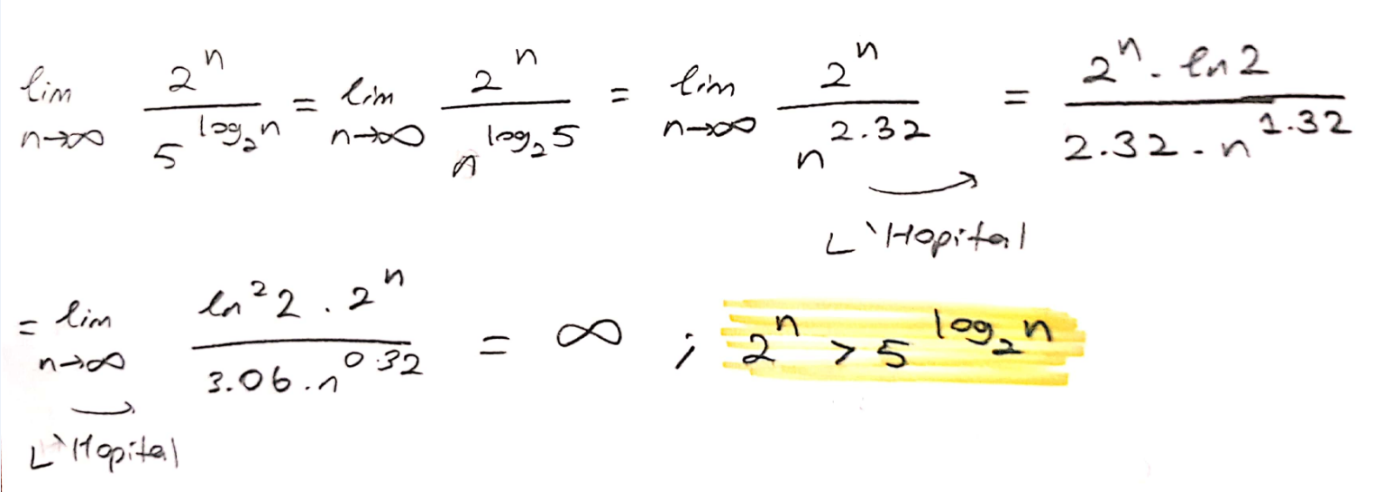


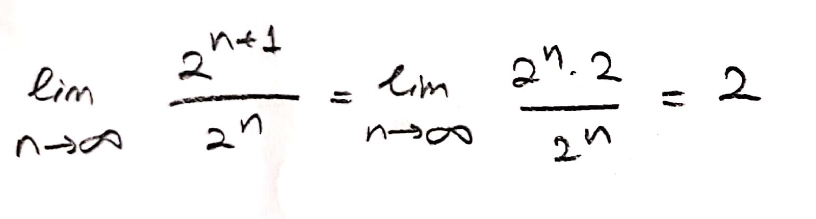


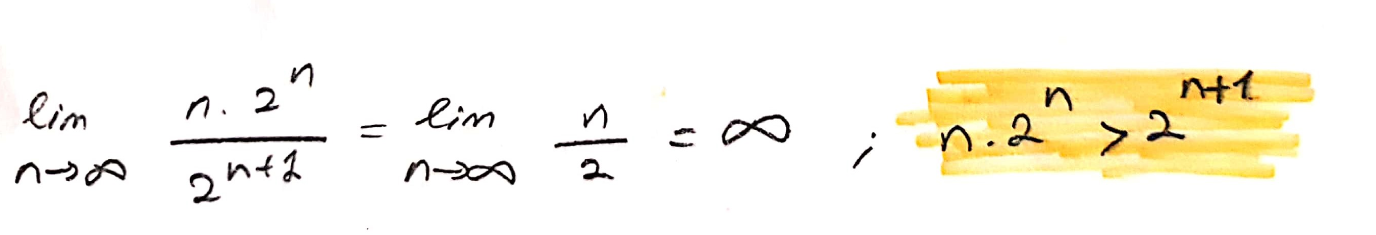


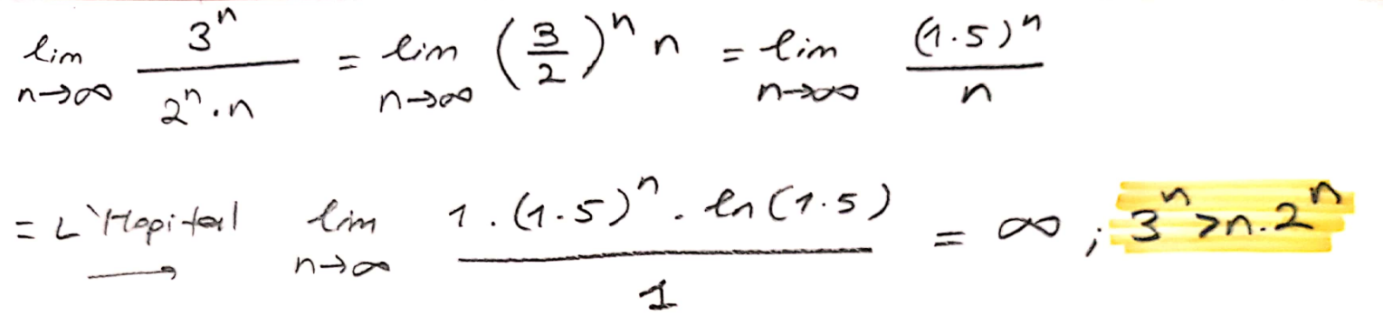












**Result :**

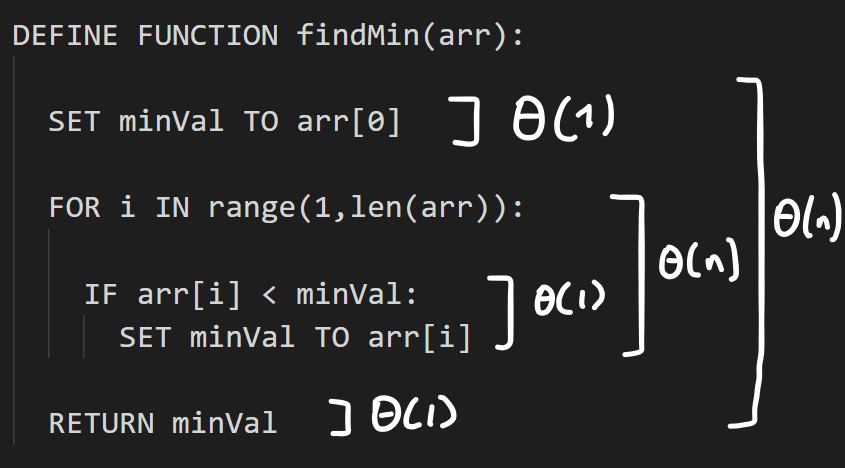
3n > n.2n > 2n+1 = 2n > 5log2n > n1.01 > n.log2n > **√**n > log3n > logn

**Part 4 :**

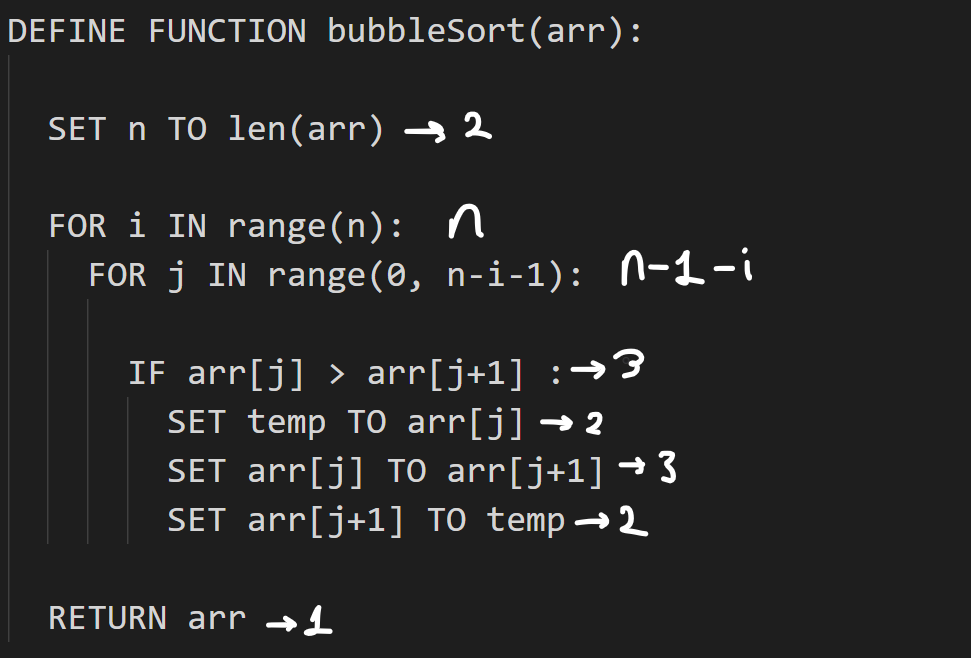
**- Find the minimum-valued item.**

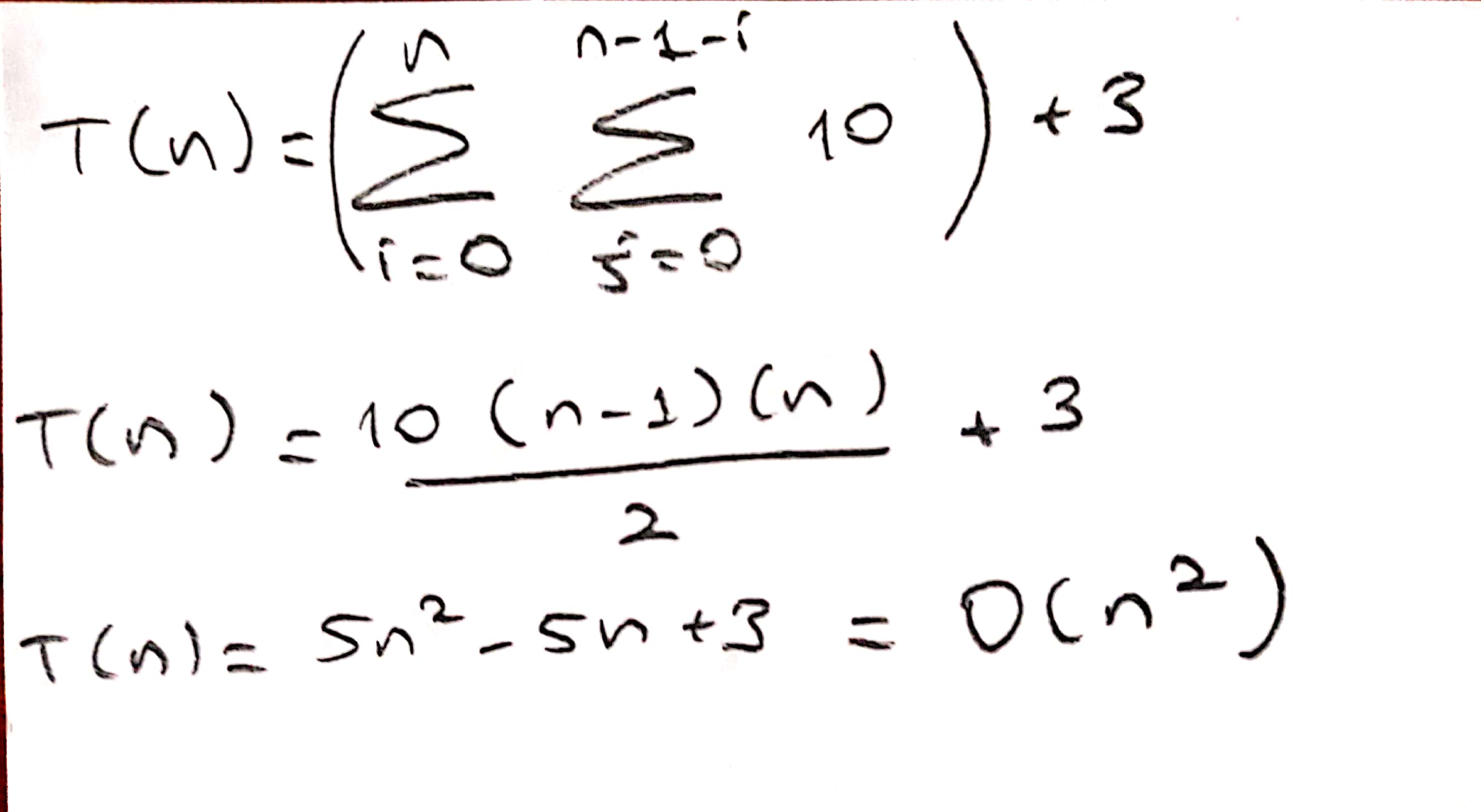
It takes linear time

T(n) = **Θ(n)**

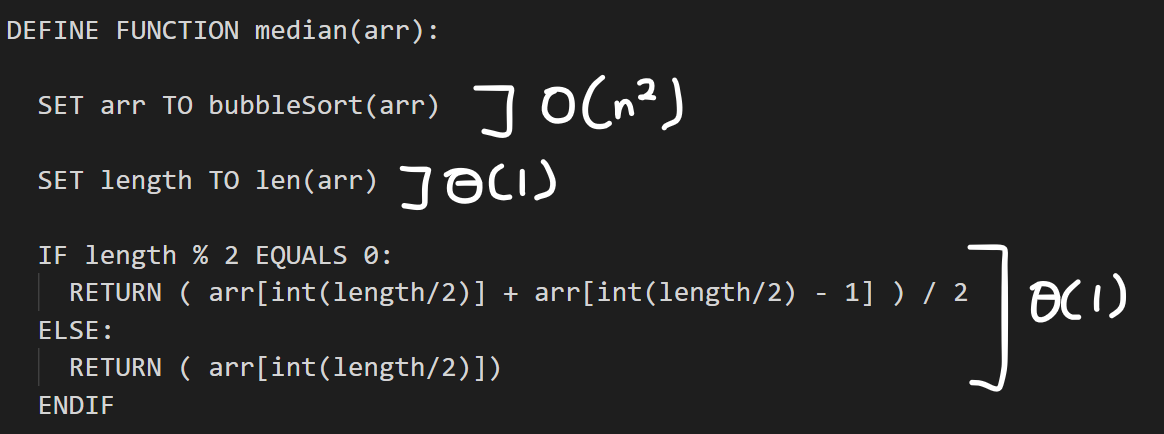


**- Find the median item. Consider each element one by one and check whether it is the median.**

Bubble sort



Bubble sort’s time complexity is **O(n2)**



O(n2)+ Θ(1)+ Θ(1)

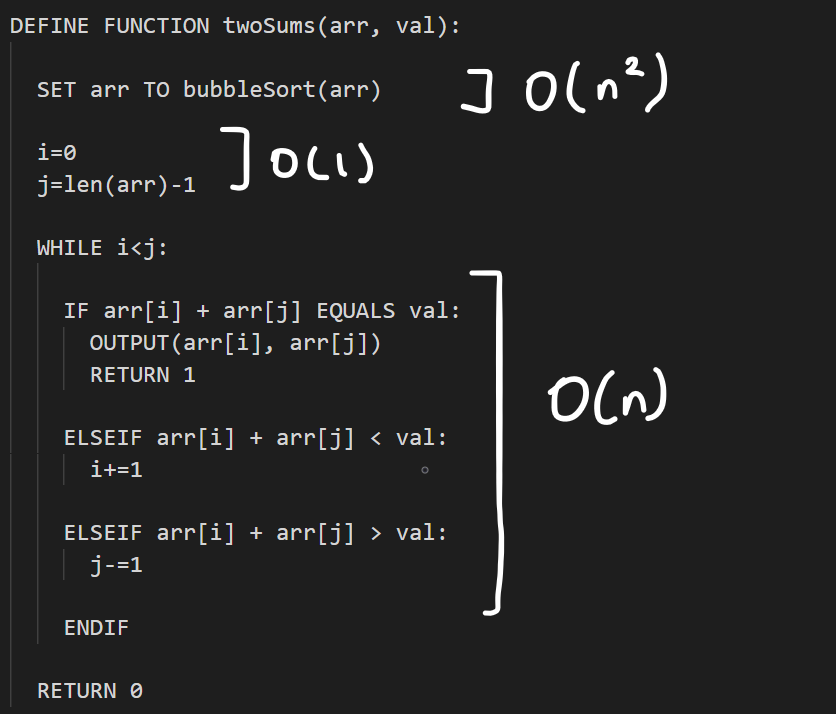
In worst case : Tworst = **O(n2)**

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

At first we sort the array with bubblesort algoritm ( O(n2) )

Inside the while loop, time complexity is O(n). When the two elements are in the middle of the array. Or two sums does not exists.

Tworst = **O(n2)**

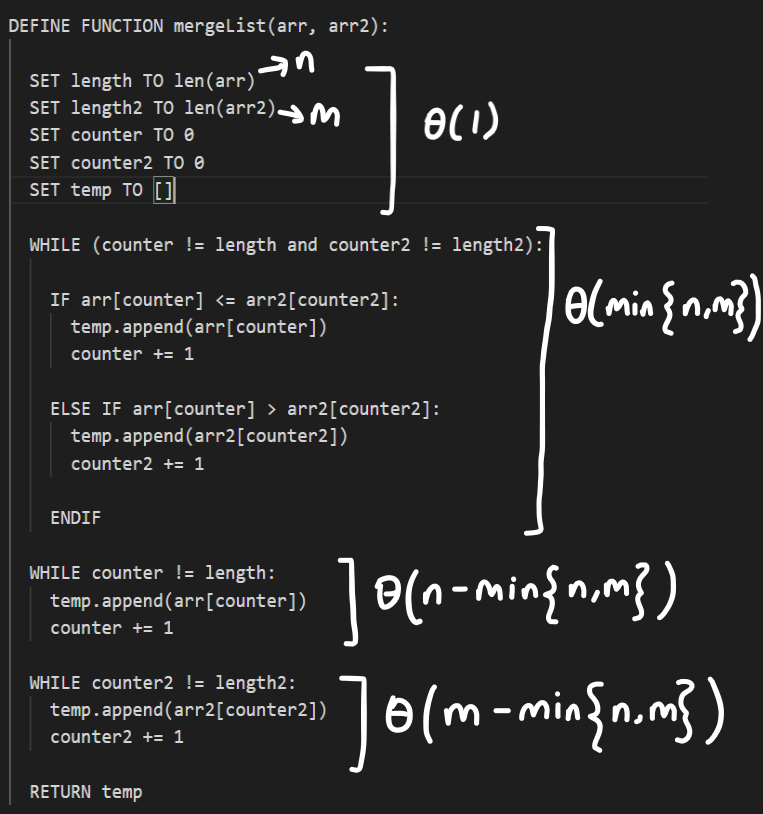
****

**- Assume there are two ordered array list of n elements. Merge these two lists to get a single list in increasing order.**

In this example, n is equal to the length of the first and m is equal to the length of the second array

Time complexity depends on the size of the bigger array and it takes linear time

Θ(1) + Θ(min{n,m}) + Θ(n - min{n,m}) + Θ(m - min{n,m}) = Θ(n+m) = **Θ(max{n,m})**

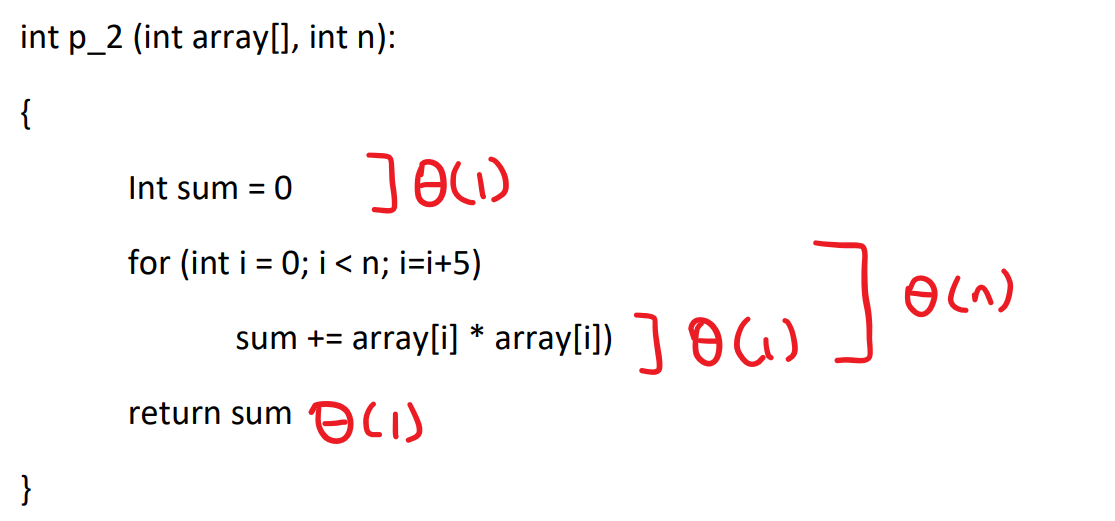


**Part 5 :**

**a)**

Time complextiy : Θ(1)

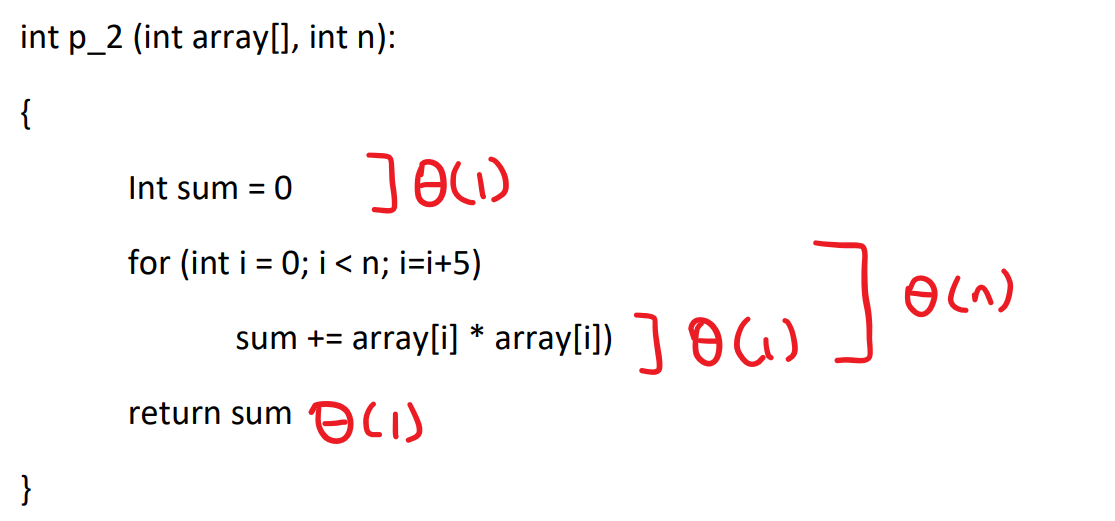
Space complexity : Θ(1)



**b)**

Time Complexity : Θ(1) + Θ(n) + Θ(1) = Θ(n)

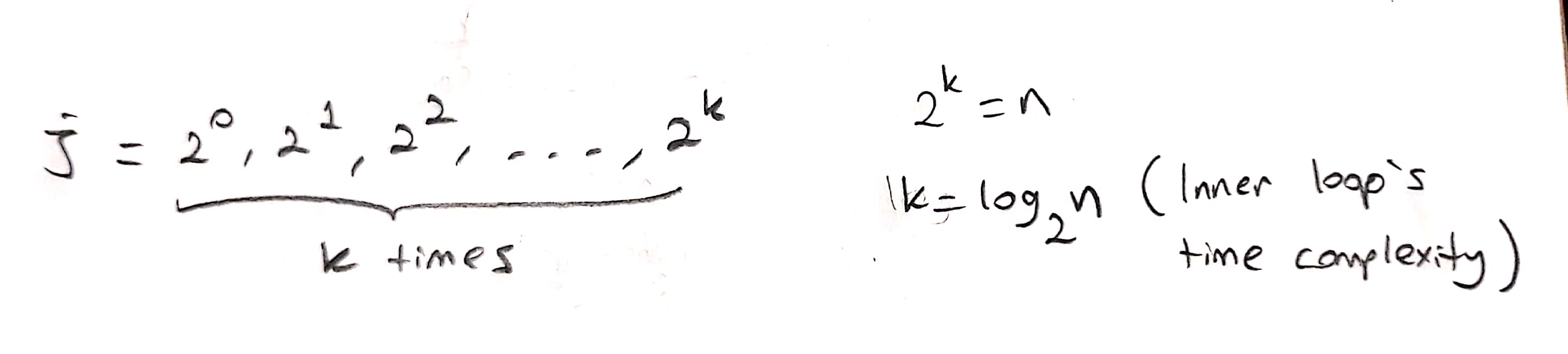
Space Complexity : Θ(1)

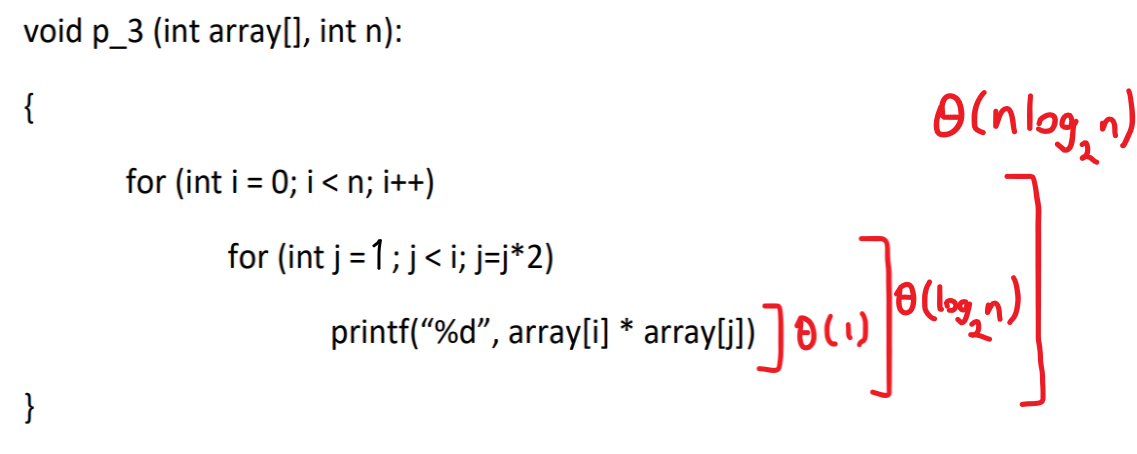


**c)**

Time Complexity : Θ(n log2n)

Space Complexity : Θ(1)





**d)**

Time Complexity :

Tworst(n) = Θ(n) + Θ(n log2n) = Θ(n log2n)

Tbest(n) = Θ(n) + Θ(n) = Θ(n)

Space Complexity : Θ(1)