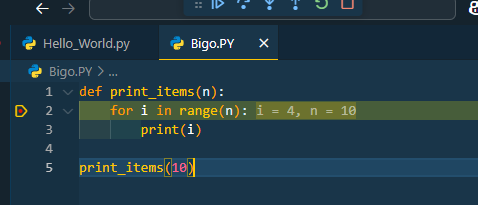
**Big o**  
Compare 2 code (which one is better?)  
Less lines = better code  
Time complexity = number of operations that it takes to complete   
Space complexity = number of space that it takes on memory

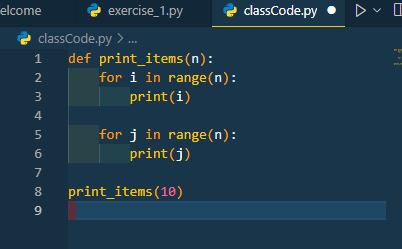
OMEGA = Best case scenario (1)  
TETA = Average case scenario (4)  
O = Worst case scenario (7) (Most cases)

[1,2,3,4,5,6,7]



0(n) (number of operations. In this case, n = 10)

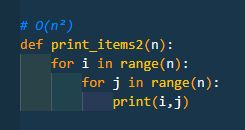
O(2N)



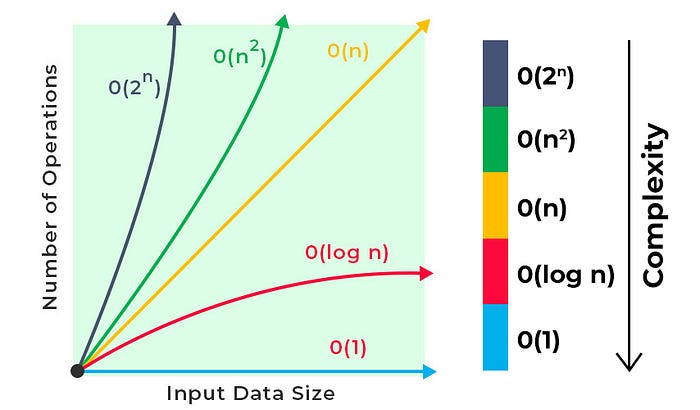
n + n = 2n = O(2n)

Simplifying = O(n)

O(n²)



n \* n = n²

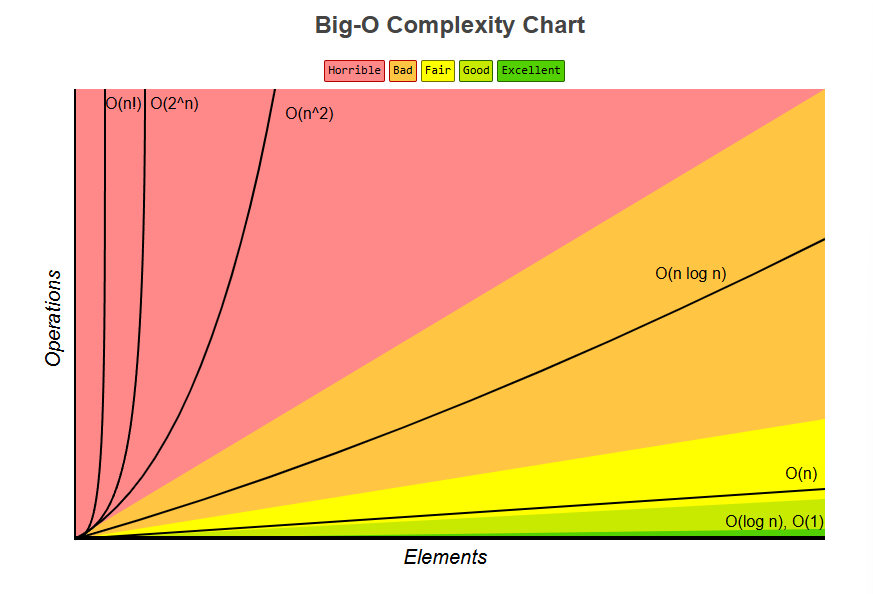


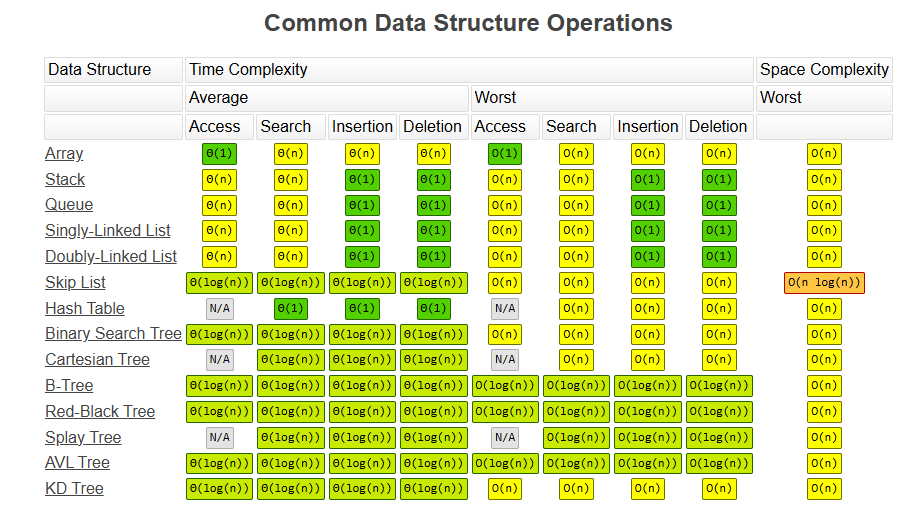
O(n²) = 1000000 (Loop within a loop)

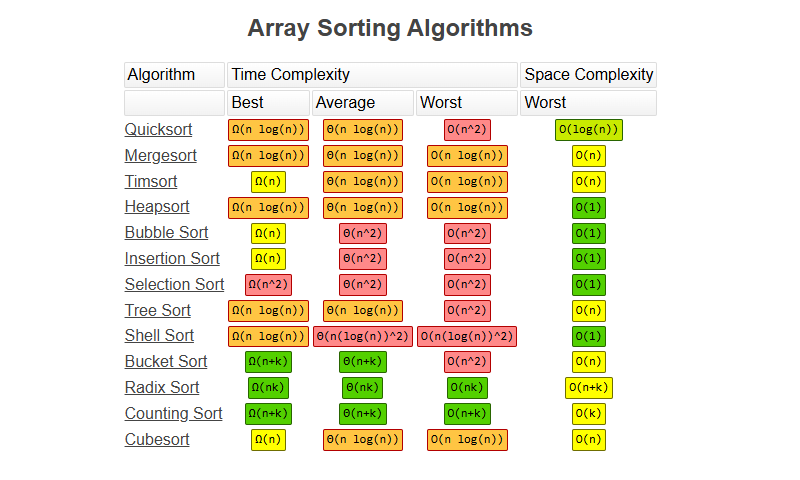
O(n) = 1000 (Proportional)

O(log n) = 10 (Divide and conquer)

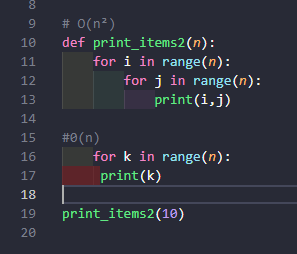
O(1) = 1 (Constant)







Dropping non-dominants



O(n² + n)

Here, n² is the dominant part. (Therefore, O(n² + n) = O(n²)

O(log(n))  
  
Very very performative

Big O on lists

My\_list.append(17) = O(1)

My\_list.pop() = O(1)

My\_list.pop(0) = O(n) (remove the item + rename the indexes)

My\_list.insert(0,11) = O(n) (add the item + rename the indexes)  
  
Therefore, in the last item is O(1). In the first one is O(n).  
  
If I want to add in the middle, will be O(n) as well

LOOKING FOR VALUE:  
Iterating through = O(n)  
By index of the value = O(1) (goes direct to the place of the value on the list)