**Types of Time Complexities**

1. **Omega (Ω) – Best Case**
   1. What it means: Omega (Ω) describes the best-case scenario for an algorithm.
   2. In simple terms: It tells you the fastest an algorithm can run in the best circumstances.
2. **Theta (Θ) – Average Case**
   1. In simple terms: It tells you what to generally expect in terms of time complexity.
3. **Big O (O) - Worst Case**
   1. What it means: Big O (O) describes the worst-case scenario for an algorithm.
   2. In simple terms: It tells you the slowest an algorithm can run in the worst circumstances.

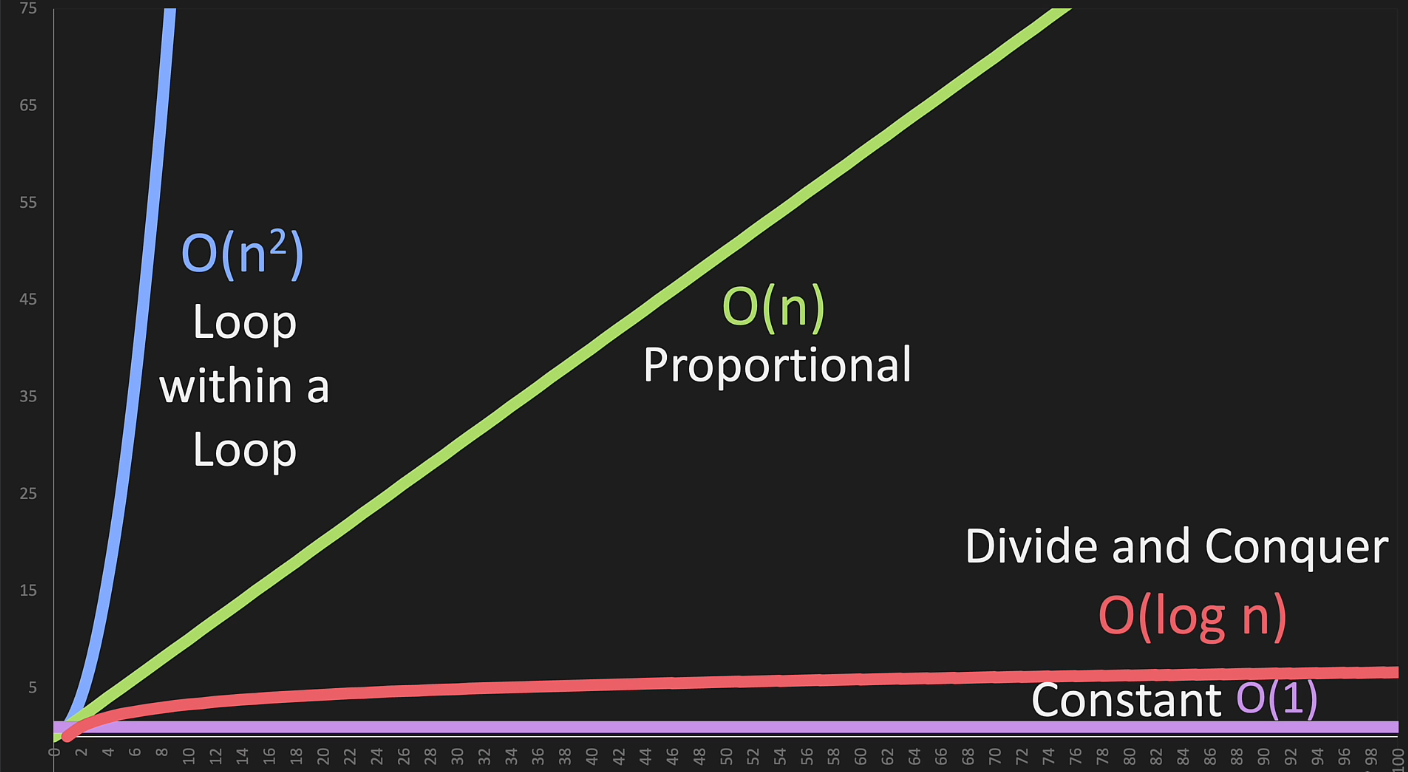
**Big O Basic Concepts:**

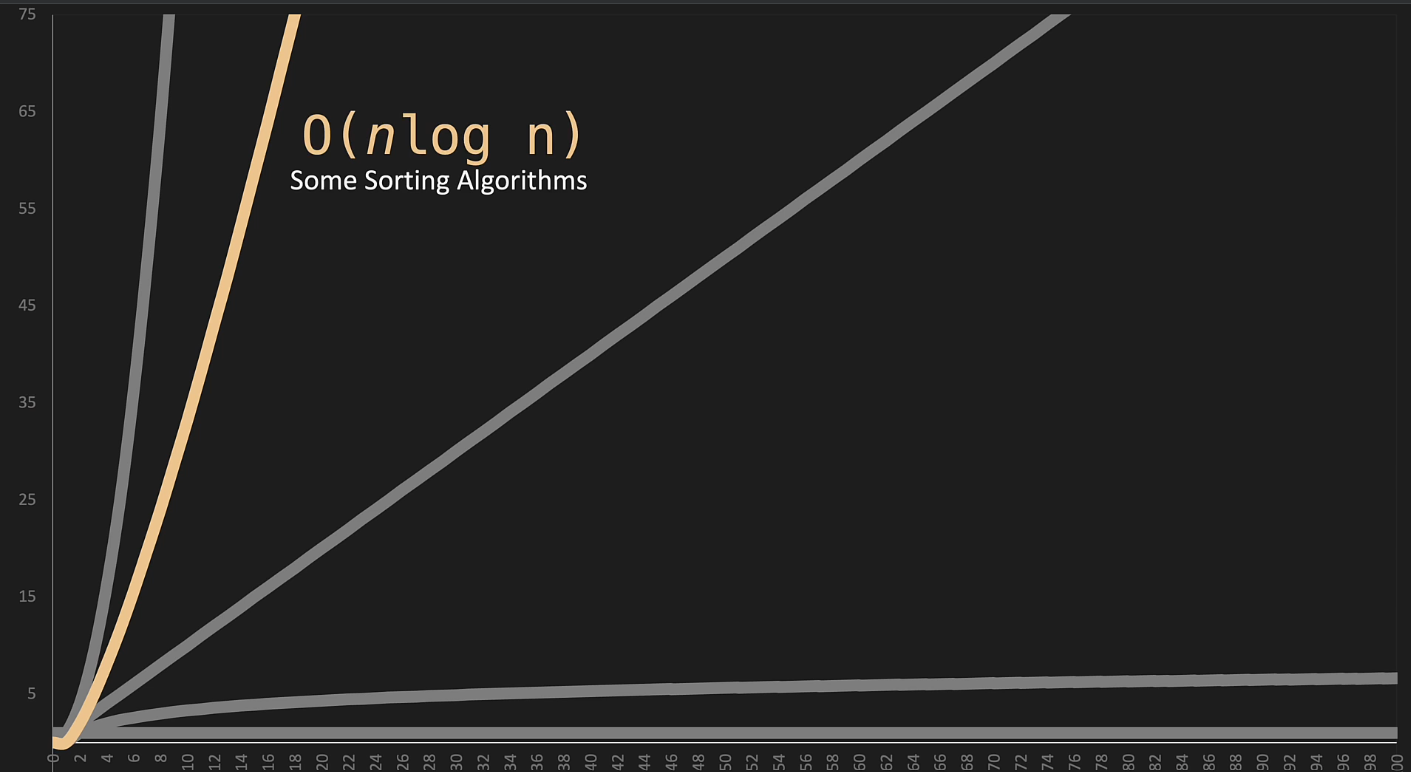
1. **O(1): Constant Time**
   1. Doesn't depend on the size of the data set.
   2. Example: Accessing an array element by its index.
2. **O(log n): Logarithmic Time** 
   1. Splits the data in each step (divide and conquer).
   2. Example: Binary search.
3. **O(n): Linear Time** 
   1. Directly proportional to the data set size.
   2. Example: Looping through an array.
4. **O(n log n): Linearithmic Time** 
   1. Splits and sorts or searches data.
   2. Example: Merge sort, quick sort.
5. **O(n2 ): Polynomial Time** 
   1. Nested loops for each power of n.
   2. Example: Bubble sort (O(n^2 )).

**Other Concepts:**

1. **Drop Non-Dominant Terms:** In O(n^2 + n), focus on O(n^2 ) as it will dominate for large n.
2. **Drop Constants:** O(2n) simplifies to O(n)
3. **Different Terms for inputs:** If there is are 2 separate loops with different parameters -> O(n+m)

Good reference: <https://www.bigocheatsheet.com/>





**Code for O(n)**

1. def print\_items(n):
2. # print\_items accepts one argument 'n'. It will print
3. # a sequence of numbers from 0 up to, but not including, 'n'.
5. for i in range(n):
6. # A for loop is initiated with 'i' iterating over
7. # the sequence of numbers produced by range(n).
8. # For each iteration, 'i' takes the current number in
9. # the sequence from 0 up to but not including 'n'.
11. print(i)
12. # Inside the loop, print(i) is called. This prints
13. # the current value of 'i' to the console. This
14. # action is performed 'n' times due to the for loop,
15. # resulting in printing of numbers from 0 to 'n - 1'.

**Other possible solutions:**

There are a few different ways the for loop in your **print\_items** function could be rewritten. Here are a few examples:

1. **Using a while loop instead of a for loop:**
   1. def print\_items(n):
   2. i = 0
   3. while i < n:
   4. print(i)
   5. i += 1
2. **Using a for loop with an explicitly defined list:**
   1. def print\_items(n):
   2. for i in list(range(n)):
   3. print(i)
3. **Using a for loop with iterator:**
   1. def print\_items(n):
   2. for i in iter(range(n)):
   3. print(i)
4. **Using list comprehension**

(though this isn't recommended if the goal is just to print the items, as it unnecessarily creates a list of **None** values):

* 1. def print\_items(n):
  2. \_ = [print(i) for i in range(n)]

1. **Using the map function.**

This method is also not recommended for printing purposes, as it's less readable and efficient than a simple loop, and it also unnecessarily creates a list of **None** values:

* 1. def print\_items(n):
  2. \_ = list(map(print, range(n)))

Each of these alternative solutions achieves the same result as the original function, but the simplest and most straightforward solution is the original for loop.

If there are 2 separate for loops

O(n) + O(n)

O(n + n)

O(2n)

O(n)

Lesson Learnt – Drop constants

**Code for O(n2):**

def print\_items(n):

for i in range(n):

for j in range(n):

print(i,j)

**Code for O(1)**

n = 1

n + n (Single Operation, no matter how n grows)

Code for O(log(n))