# CS Fundamentals
# Big O Notation

# Introduction

- An algorithm's Big-O notation is determined by how it responds to different sizes of a given dataset. For instance how it performs when we pass to it 1 element vs 10,000 elements.
- Big-O puts the **number of steps** in the spotlight. The hardware factor is taken out of the equation. Therefore we are not talking about *run time,* but about *time complexity*.
- Estimating the growth of the function without having to worry about constant multiplier or smaller order terms.
- used extensively to estimate the number of operations an algorithm will use as its inputs grows
- can be used to compare two algorithms to determine which one is more efficient as the size of the inputs grows

# Worst case scenario

- The Big-O notation takes a **pessimistic** approach to performance and refers to the worst case scenario.

  [12, 56, 3, 31, 56, 24, 6, 75]

  Searching for 24 -> 6 steps
  Searching for 56 -> 2 steps
  ...

  Big O Notation -> 8 steps
  If the array contains 300 elements -> 300 steps
  As the size of the array grows, the steps of the algorithm grow as well.

# O(1)

- O(1) means that the algorithm takes the same number of steps to execute regardless of how much data is passed in. It describes an algorithm that will always execute same amount of steps regardless of the size of the input data set.

```
def print_first(list):
    print(list[0])
```

```
def convert_to_farenheit(c):
    return (c*9/5) + 32
```

# O(N)

- An algorithm that is O(N) will take as many steps as there are elements of data. So when an array increases in size by one element, an O(N) algorithm will increase by one step.

```
def print_all_elements(list):
    for item in list:
        print (item)
```

```
def contains_value(list, value):
    for item in list:
        if item == value:
            return True
    return False
```

# O(N²)

- O(N²) represents an algorithm whose performance is directly proportional to the square of the size of the input data set. It is generally quite slow: If the input array has 1 element it will do 1 operation, if it has 10 elements it will do 100 operations, and so on.
- Two nested loops are usually O(N²)

```
def contains_duplicates(list):
    for i in range(len(list)):
        for j in range(len(list)):
            if i == j: #Don't compare with self
                continue
            if list[i] == list[j]:
                return True
    return False
```

# O(logN)

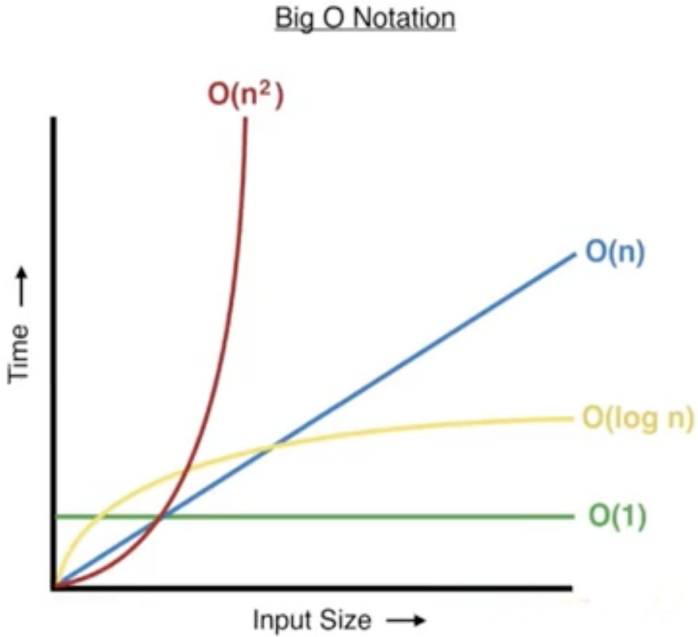
- It describes an algorithm that its number of operation increases by one each time the data is increased by a certain fold.
- In binary search case, O(logN) describes an algorithm that its number of operations increases by one each time the data is doubled.
- Binary search
  - Open the dictionary in the middle to check the word you are looking for.
  - If our word is alphabetically more significant, look in the right half, else look in the left half.
  - Divide the remainder in half again, and repeat steps 2 and 3 until we find our word.
- Very common algorithm for searching elements in an ordered array.
  10 elements < 2 * 2 * 2 * 2 -> 2^4 -> 4 steps to find the number
  100 elements < 2*2*2*2*2*2*2 -> 2^7 -> steps to find the number
  1000 elements < 2*2*2*2*2*2*2*2*2*2 -> 2^10 -> 10 steps to find the number

# O(logN)

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return True
    return False
```

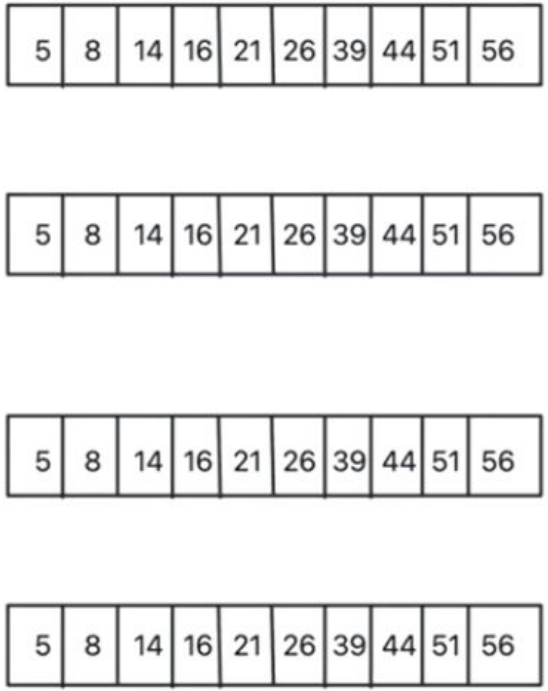
# Big O Notation comparison.

- O(N logN) A log linear algorithm of this complexity class is doing log(N) work N times and therefore its performance is slightly worse than O(N). Many practical algorithms belong in this category (from sorting, to pathfinding, to compression)
- $O(2^N)$ = Exponential growth means that the algorithm takes twice as long for every new element added. Poor performance.
- O(N!) — Factorial This class of algorithms has a run time proportional to the factorial of the input size. Very poor performance.




$O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(2^N) < O(N!)$

Binary search



# Revision

```
  10101
+ 1110
```

```
  1110101
+ 101111
```

# 140 base 18 to base 4

!P ^ Q -> P

| P | Q | !P | !P ^ Q | !P ^ Q -> P |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# (P -> Q) -> !(Q -> R)

| P | Q | R | p -> Q | Q -> R | !(Q -> R) | (P -> Q) -> !(Q -> R) |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

$\neg p \wedge r \Rightarrow q$ is false when...

15 >> 2 XOR 13 << 3

What are the elements of the set expressed as:

$A = \{ x \mid x \in Z, -5 < x < 6, x^2 > 5 \}$

$A = \{ x \mid x \in N, x < 6\}$

$B = \{ x \mid x \in Z, -3 < x < 3\}$

$U = \{ x \mid x \in Z, -10 < x < 10\}$

$(A \cup B)' = ?$

# THE END