Data Structures and Algorithms

Complexity Analysis

Week 1

Time:
2 min

Counting Operations

- Every algorithm takes a certain amount of time
- A proxy for the amount of time is the number of ‘elementary operations’
- An elementary operation is a computer instruction executed in a single time unit.
    - +,-,x,/, %,>,<, | , &, ==, =
- The running time of an algorithm is a linear function of the number of operations

Time:
5 min

Example 1

```
Algorithm contains(a, x):
    for i = 0 to len(a)-1
        if x == a[i]
            return true
        return false
```

- Worst-case
- Best-case
- Average-case

Generally, we look at worst-case

when analysing algorithms.

Time:
5 min

Example 2

```
Algorithm max(a):
    max = a[0]
    for i = 1 to len(a)-1
        if a[i] > max
            max = a[i]
    return max
```

What is the runtime?

How much space is used?

Time:
5 min

Asymptotic Analysis

- Asymptotic - what is the ‘limiting behaviour’. What happens as n -> infinity?
- We wish to classify algorithms into smaller groups rather than computing the exact number of operations each time.
- There are 3 important notations here - **Big-O, Big-Omega, Big-Theta**.
- **Big-O** (upper bound for the algorithm) - it will take at most this time
- **Big-Ω** (lower bound for the algorithm) - it will take at least this time
- **Big-θ** (lower and upper bound for the algorithm) - it will take roughly this time

Time:
5 min

Big-O Definition

- Let's call the runtime function T(n)
- We can say that the runtime T(n) is O(f(n)) for a function f(n) (e.g. n, log n, n^2, 2^n, n!) if:
  - In **maths**: T(n) ≤ Mf(n) for all x > c for some constants M and c
  - In **words:** T(n) is eventually bounded above by a multiple of the function f(n).
- For example T(n)= $5n^2+2n$ is O(n^2) since $5n^2+2n \leq 7n^2$ for all n > 0
- Or T(n) = 18 n log n + 100 n is O(n log n) since 18 n log n + 100 n ≤ 200 n log n for all n > 1

Intuition - What is the dominant term in the expression? Which term grows the fastest?

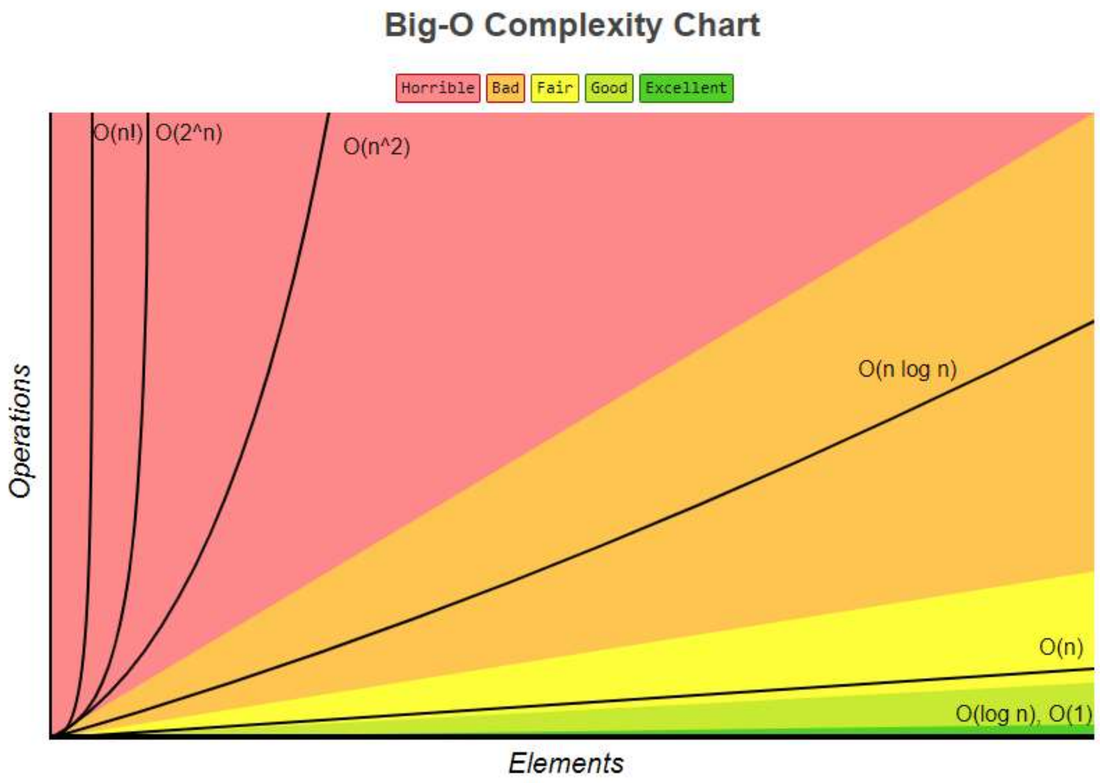
Time:
5 min

Big-O Categories

- There are various **Big-O** categories but the most used ones are
  - $O(1)$ - constant time
  - $O(\log n)$ - log time
  - $O(n)$ - linear time
  - $O(n \log n)$ - loglinear time
  - $O(n^2)$ - quadratic time
  - $O(n^3)$ - cubic time
  - $O(2^n)$ - exponential time
  - $O(n!)$ - factorial time

Time:
5 min

Big-O Complexity Chart



Source: [Medium]

Time:
20 min

Big O Notation - Exercise

- Get into groups of 3.
- Each group should have only one laptop open.
- Follow the link below to access the instructions and prompt for the exercise.
- Make sure to discuss the asymptotic runtime and space complexity of the algorithms provided.

bit.ly/DSAW1BigO

Time:
5 min

Big O Notation - Exercise Solutions

index	Time Complexity	Space Complexity
Alg1	$O(n^2)$	$O(n)$
Alg2	$O(n)$	$O(n)$
Alg3	$O(n^2)$	$O(n)$
Alg4	$O(\log(n))$	$O(n)$
Alg5	$O(ab)$	$O(a + b)$

Look up ***Auxiliary Space*** and see how that differs from space complexity

Time:
7 min

Questions

Time:
3 min

Next Steps

1. Commit implementations for the week
2. Readings for Week 1 - Make sure you understand time and space complexity
3. Review Python Basic Data Structures -
4. Brainstorm Project Topics
5. Post questions on Piazza