ALGORITHM ANALYSIS / BIG-OH

WHY ANALYZE ALGORITHMS?

- CPU time (effort) is a valuable resource
- There can be multiple different algorithms and implementations to solve a problem
- Want to make sure we choose an efficient approach
    - We'll focus on efficiency with respect to operations performed by CPU
    - Can also choose / analyze with respect to others such as space on computer used
    - Often tradeoff between these

ALGORITHM

- Not the actual code (aka "implementation")
- The steps you take to solve the problem
- Analysis should happen:
  - at the algorithm stage
  - before you start coding

COMPLEXITY

- How does the time depend on the problem size?
  - Use the phrase "time", but we don't really care about exact execution time
  - Look at growth function that says how time behaves
  - Actually only care about **asymptotic complexity**
    - Long term behavior as n increases
    - Based on dominant term
- Need to defined problem size (n)
  - Ex: # elements in array

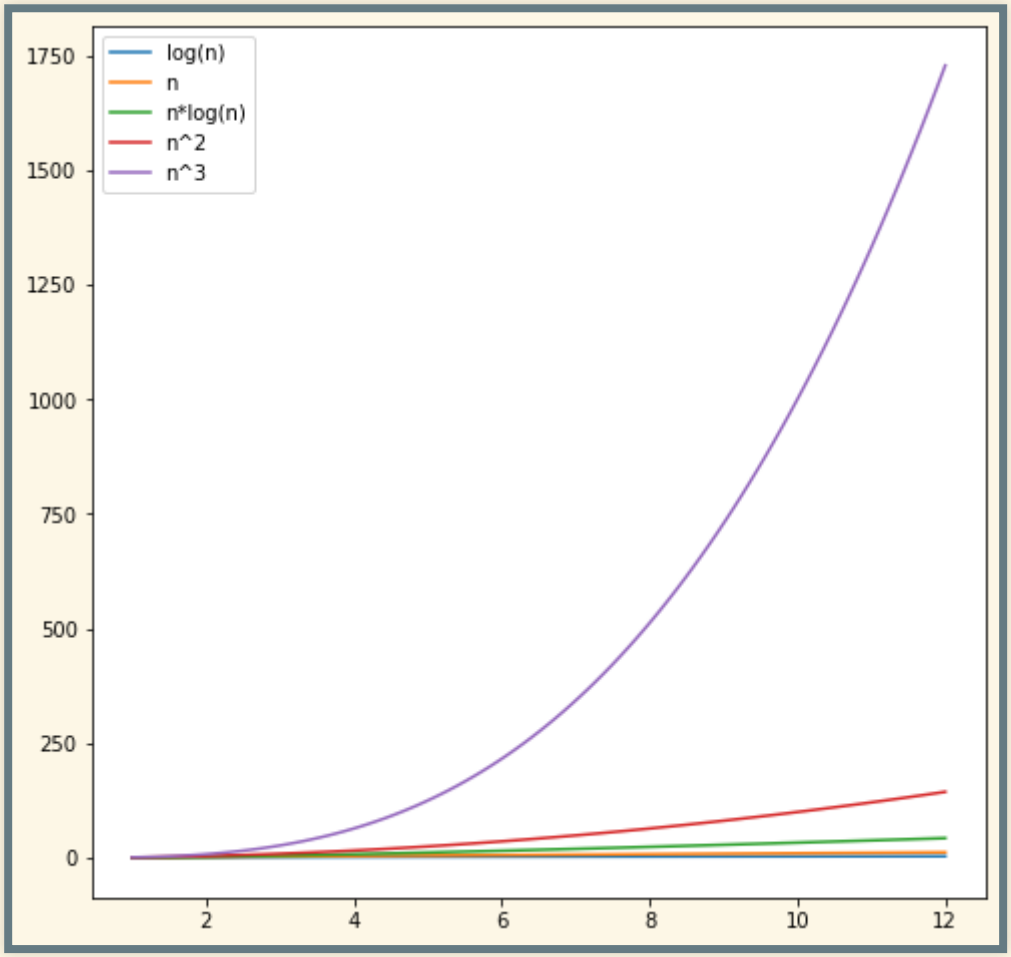
COMPLEXITY

- Aka **order** of algorithm
- Big-Oh notation
  - O(n^2) -> behaves like n^2 as n increased
- Big-Oh formally:
  - f(n) is O(g(n)) if there exists c and m s.t. f(n) < c g(n) for all n > m
- Basically, upper bound to growth function

EXAMPLES

- 2n^3 + 12n^2 - n + 4
- 4n^2 + 15n
- n + n*log(n) + 6
- log(n) + 12

COMPLEXITY VISUALIZATIONS



COMPLEXITY VISUALIZATIONS