02-613 Week 4

Algorithms and Advanced Data Structures

Aidan Jan

September 19, 2025

Asymptotics

We need a formal definition of algorithm efficiency.

- Must be concrete and falsifiable.
- We care about large problems.

Example: Heap Operations

Proposal for runtime: count the exact number of operations
Insert to heap:

- Find leaf node (1 operation)
- Compare two and swap (4 operations)
  - Worst case number of swaps ($\log n$ operations)

Delete from heap:

- Find leaf (1 operation)
- Swap (3 operations)
- Delete leaf (1 operations)
- Bubbling ($\max(4 \log n, (2+3) \log n)$)
  - (2 from compare, 3 from swap)

The thing is, we don't care about the exact number of operations. Sometimes, implementations may be abstracted out of the programming language too. However, what we do care about is how it scales.

- In terms of the heap example, the only term we will ever care about is the $\log(n)$ term, because with a large amount of nodes, it will grow larger than everything else.

Big-O

A running time $T(n)$ is $O(f(n))$ if $\exists$ constants $n_0 \geq 0$, $c \geq 0$, such that

$$T(n) \leq c \cdot f(n) \quad \forall n \geq n_0$$

For example, if $1 + 4 \log n \leq c \log n$, then $c = 5$. Or, if $6 + \max(4, 5) \log m \leq cn$, then $c = 7$.
In other words, $\mathrm{O}(f(n))$ is the set that contains all functions $cf(n)$, where c is a constant.

Big-Ω

$T(n) \in \Omega(f(n))$ for constant $\varepsilon \geq 0$, $n_0 \geq 0$, if $T(n) \geq \epsilon f(n) \quad \forall n \geq n_0$.
Big-Omega gives a upper bound for running speed. Unlike Big-O, we consider the best-case scenario.

- From our heap deletion example earlier, the Big-Omega of the delete operation would be O(1). This is because if we delete a leaf, we are instantly done.

Big-Θ

$T(n) \in \Theta(f(n))$ if and only if $T(n) \in O(f(n)) \wedge T(n) \in \Omega(f(n))$.

- For example, Big-Theta of BFS is $\Theta(n+m)$. This is because in the worst case, we have $O(n+m)$. However, in the best case, we still must visit every node, therefore $\Omega(n+m)$.

Asymptotic Limit

Theorem (Θ): if $\lim_{n\to\infty} \frac{T(n)}{g(n)} = c$, for some constant $c \geq 0$, then $T(n) \in \Theta(g(n))$.

Proof:

There exists a n_0 such that $\frac{c}{2} \leq \frac{T(n)}{g(n)} \leq 2c$, for all $n > n_0$. This implies that $T(n) \leq 2c \cdot g(n)$ for $n > n_0$, which gives $T_n \in O(g(n))$.

Example Asymptotics:

- O($\log n$): Binary search
- O($n \log n$): Sorting
- O(n^2): 2D arrays, all pairs of objects
- O(n^3): Matrix multiplication (sometimes)
- O(n^k): Tensor Multiplication, independent sets, finding all k-groups of objects
- O(2^n): (these are evil) Largest independent set in a graph
- O(n^n): (aka $n!$, also evil) List all permutations of a set