Algorithm Analysis

Running Time

- Run time of some code is measured in terms of the “size” of the input data.
- Denoted “n”
- Run time is reported as proportional to some function of n
  - Sometimes called the “order” of the code

Big-O Notation

- Mathematically, we use “big-O” notation.
- For example, we write $O(n^2)$ to mean: (at most) proportional to n^2

Combining Big-Os

- Adding Big Os, only larger survives
  - $O(n) + O(n^2) = O(n^2)$
- Multiplying Big Os
  - $O(n) * O(n^2) = O(n^3)$

Special Cases

- Linear means proportional to n
- Constant means time does not depend on the input size. (O(1))

Combining Code: Succession

- If two separate pieces are run in succession, the overall order is their **sum**

Combining Code: Loops

- If a loop is executed, then the overall order is at most the number of times the loop executes **times** the worst-case order of the body

Asymptotic Analysis

Type	Pronounced	Meaning (order of)	Examples
$O(n)$	Big-O	$\leq c*n$	$5n$, $\log(n)$, 1
$o(n)$	Little-o	$< c*n$	$\log(n)$, 1
$\Omega(n)$	Big-Omega	$\geq c*n$	$5n$, $100n^2$, $n!$
$\omega(n)$	Little-Omega	$> c*n$	$100n^2$, $n!$
$\Theta(n)$	Theta	$= c*n$	$5n$, $100n$, $0.01n$

$O(n^2)$	$\leq k*n^2$
$o(n)$	$< k*n$
$\Omega(n^3)$	$\geq k*n^3$
$\omega(n^{1.5})$	$> k*n^{1.5}$
$\Theta(\log(n))$	$= k*\log(n)$

Type	Pronounced	Examples
$o(n)$	Little-o	$\log(n)$, 1
$O(n!)$	Big-O	$5n! + 4n^2$, $\log(n)$, 1
$\Theta(n^3)$	Theta	$16n^3 + 4n^2 - 2n$, $12n^3$
$\Omega(2^n)$	Big-Omega	$4*2^n$, $n!$
$\omega(n^2)$	Little-Omega	$100n^3$, n^4, $6n^{100}$, $n!$

Complexity	**Name**	**Example**
$\Theta(1)$	Constant	Insert element at end of array
$\Theta(\log(n))$	Logarithmic	Binary search (sorted)
$\Theta(n)$	Linear	Naive search for maximum value (unsorted)
$\Theta(n \log(n))$	n log n	Mergesort
$\Theta(n^2)$	Quadratic	Two nested loops
$\Theta(n^k)$ - really, $O(n^k)$	Polynomial	Matrix multiplication
$\Theta(2^n)$ - really, $O(2^n)$	Exponential	Check for subset
$\Theta(n!)$ - really, $O(n!)$	Factorial	Generate all permutations

Best-Worst-Average Case

- Best - fastest case possible
- Worst - worst case possible
- Average - average case

Amortized Analysis

- Aggregate method
  - Find an upper bound T(n) that holds for every sequence of n operations
  - The cost of a single operation is T(n)/n
- Any n operations take O(n) time:
  - O(1) amortized
- Any n operations take n^3 time:
  - $O(n^2)$ amortized
- Definition: Amortized analysis assesses the average performance of an algorithm or data structure over a sequence of operations, rather than focusing on individual operations.
- Purpose: It provides a more accurate understanding of the overall performance characteristics, especially for data structures and algorithms with varying costs for different operations.
- Example: Consider dynamic arrays or vectors. Resizing operations (e.g., doubling the array size) occur infrequently, but their cost is distributed across multiple insertions, resulting in amortized constant time complexity for each insertion.