Algorithms analysis

Algorithm

- problem solving method suitable for implementation.
- well-defined procedure that takes a set of values as input and produces a set of values as output
- central for most fields in computer science.

Example 1: the sorting problem

- Input: sequence A of n numbers \( a_1, a_2, \ldots, a_n \).
- Output: reordering of A in increasing value.

Algorithm analysis: determine the time/space required by an algorithm

Goals:

- minimize time
- minimize space

Note:

- The expected size of the input matters when comparing
- Trade off for time/space

How to measure running time

Experimental study

- write program
- run program and measure time

Note: Not a good method in general
- need to implement and test: preferable not to!
- need similar hardware/software to compare
- data set is limited and may not be relevant

General methodology

- high level description (pseudo-code)
- independent on hardware/software

Pseudo-code is a structured description of an algorithm: not as formal as a programming language.

Example: find the maximum element of an array.

Algorithm Max(A, n):
• Input: an array A storing \( n \) integers
• Output: maximum element in A.

1: \( \text{max} = a[1] \)
2: for \( i = 2 \) to \( n \) do
3: if \( \text{max} < A[i] \) then \( \text{max} = A[i] \)

**Primitive operations**

• arithmetic operations (+, -, *, /)
• comparisons
• indexing, etc.

Use pseudo-code to count the number of primitive operations.

**Worst case vs. average case**

Usually measure worst case complexity.

• crucial importance in some applications
• often algorithms with better worst case performance have worse average case performance in practice.

**Asymptotic Notations**

\( f, g \) functions over natural numbers

**Big-Oh:** \( f(n) = O(g(n)) \) if there are constants \( c > 0 \) and \( n_0 \geq 1 \) such that \( 0 \leq f(n) \leq cg(n) \) for all \( n \geq n_0 \) (\( g \) is an **upper bound** for \( f \)).

Example 1: \( 2n - 1 \) is \( O(n) \).
Proof (smart choice for now): \( n_0=1, c=2 \) satisfies \( 2n - 1 \leq cn \) for \( n \geq n_0 \).

Example 2: \( n^3 + n \log n \) is \( O(n^3) \).
Proof (smart choice for now): \( n_0=1, c=2 \) satisfies \( n^3 + n \log n \leq 2n^3 \) for \( n \geq n_0 \).
Some rules

- try to get the smallest **upper bound** \( g \) for \( f \).
- look at the dominant term of \( f \) to guess \( g \).

Common cases (\( n \) is the input size)

- \( O(1) \): constant
- \( O(\log n) \): logarithmic
- \( O(n) \): linear
- \( O(n \log n) \)
- \( O(n^2) \): quadratic
- \( O(2^n) \): exponential

**Big-Omega:** \( f(n) = \Omega(g(n)) \) if there are constants \( c>0 \) and \( n_0 \geq 1 \) such that \( f(n) \geq cg(n) \geq 0 \) for all \( n \geq n_0 \) (\( g \) is a **lower bound** for \( f \)).

Example 1: sorting of \( n \) real numbers is \( \Omega(n \log n) \).
Example 2: \( 2n - 1 \) is \( \Omega(n) \): \( n_0 = 1, c=1 \).

**Big-Theta:** \( f(n) = \Theta(g(n)) \) if there are constants \( c',c''>0 \) and \( n_0 \geq 1 \) such that \( 0 \leq c'g(n) \leq f(n) \leq c''g(n) \) for all \( n \geq n_0 \) (\( g \) is an **asymptotically tight** bound for \( f \)).

Example 1: some sorting algorithms are \( \Theta(n \log n) \).
Example 2: \( 2n - 1 \) is \( \Theta(n) \).

**Small-o:** denote an upper bound that is not asymptotically tight

\( f(n) = o(g(n)) \) if for any constant \( c>0 \) there is a constant \( n_0 \geq 1 \) such that \( 0 \leq f(n) < cg(n) \) for all \( n \geq n_0 \).

Example: \( 2n = o(n^2) \) but \( 2n^2 \neq o(n^2) \).

**Asymptotic notations:** \( T(n) = T(\frac{n}{2}) + \Theta(n) \)

Asymptotic analysis

- use big-Oh for the number of primitive operations in the algorithm.
- compare asymptotic running times of algorithms
  
  \( O(n) \) better than \( O(n^2) \)
  
  \( O(n \log n) \) better than \( O(n \sqrt{n}) \)
Some Rules

1. \( T_1(n) = O(f(n)), T_2(n) = O(g(n)) \):
   
   - \( T_1(n) + T_2(n) = \max\{O(f(n)), O(g(n))\} \)
   
   - \( T_1(n) \times T_2(n) = O(f(n) \times g(n)) \)

2. \( T(n) \) is a polynomial of degree \( d \) (\( d \) constant): \( T(n) = \Theta(n^d) \).

3. \( T(\text{loop}) = T(\text{loop body}) \times \text{number of iterations} \).

Solving Recurrences

1. **Substitution method**: guess a bound then prove by induction that the guess is correct.
   
   Example: \( T(n) = 2T\left(\frac{n}{2}\right) + n, T(1) = 1 \): guess \( T(n) = O(n \log n) \).

2. **Iteration method**: recursion trees.

   Example: \( T(n) = 3T(\frac{n}{3}) + n, T(1) = 1 \).

3. **Master method**: \( T(n) = aT\left(\frac{n}{b}\right) + f(n), a \geq 1, b > 1 \)
   
   (a) if \( f(n) = O(n^{\log_b a - \epsilon}), \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).
   
   Example: \( T(n) = 9T\left(\frac{n}{3}\right) + n \).

   (b) if \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \).
   
   Example: \( T(n) = T\left(\frac{2n}{3}\right) + 1 \).

   (c) if \( f(n) = \Omega(n^{\log_b a + \epsilon}), \epsilon > 0 \), and \( af\left(\frac{n}{b}\right) \leq cf(n), n \geq n_0 \), for some \( c < 1, n_0 \geq 1 \), then \( T(n) = \Theta(f(n)) \).
   
   Example: \( T(n) = 3T\left(\frac{n}{4}\right) + n \log n \).