Analysis Tools

Characterizing Performance

Running time
Memory usage

Depends partly on hardware platform, implementation, operating system, etc.

Goals of Characterization

Predict performance on any input
Compare relative performance of algorithms/data structures
Do it without having to implement first

Experimental Analysis

Implement data structure and algorithm
Run on many inputs of different sizes and other characteristics
  • Record running time, memory usage, etc.
Perform statistical analysis
  • Plot data, find a best fitting curve

Limitations of Experimental Analysis

Requires implementations of each algorithm/data structure to be compared
Fair comparison must be on same hardware/software platform
Difficult to make good predictions
  • Test inputs may not fully characterize all possible inputs
  • Extrapolation of input sizes may not be accurate
    — Difficult to know what input range must be tested

Asymptotic Analysis

Express algorithm as pseudo-code
Count maximum number of primitive operations
  • As function of input size, \( n \)
Report analysis results in “Big-Oh” notation
Pseudo-code

Looks like generic high-level language
Designed for human readability
Express algorithm concisely
  • But don’t skip important details

Pseudo-code Example

Algorithm: arrayMax(A,n)
Input: An array A storing n >=1 integers
Output: Maximum element value in A
currentMax ← A[0]
for i ← 1 to n-1 do
  if currentMax < A[i] then
    currentMax ← A[i]
return currentMax

Primitive Operations

Determine “running time” of pseudo-code algorithm
Assume each operation takes same time or some constant multiple
Just count operations
  • assignment
  • procedure call, return
  • arithmetic operation, comparison
  • indexing array, following reference

Counting Operations Example

currentMax ← A[0] 2 ops
for i ← 1 to n-1 do 2n-2 ops
  if currentMax < A[i] then n-1 ops
    currentMax ← A[i]
  ~n ops (max)
return currentMax 1 op

Total operations: 4n ops

Exact constants will not matter for the asymptotic analysis

Asymptotic Analysis

Provides bounds on worst (or average) case behavior of algorithm
Emphasizes behavior “in the limit”, as n grows to be very large
Constant factors are ignored

“Big-Oh” Notation

Given two functions, f(n) and g(n),
f(n) is O(g(n)) if there is are constants c > 0 and n₀ ≥ 1 such that f(n) ≤ cg(n) for all n ≥ n₀
  • “f(n) is order g(n)”
g(n) provides upper bound on f(n)
  • in some sense, f(n) ≤ g(n)
Analysis of maxArray

Let's say number of operations was exactly $4n = f(n)$
Choose $c=5$ and $n_0=1$, and try $g(n) = n$
$f(n)$ is $O(n)$

Proving Big-Oh by Example

1. Choose likely value for $c$
2. Find intersection of $f$ and $cg$
   - set equal and find roots
3. Choose largest intersection as $n_0$
4. Show that $cg > f$ for a value $> n_0$

Some useful $g(n)$ functions

logarithmic
  - constant
  - linear $1 \log n$
  - quadratic $n\log n$
  - cubic $n^2\log n$
  - polynomial $n^k$
  - exponential $2^n$
  - in increasing order

Other useful notations

big-Oh $O \leq$
  - “upper bound”
little-o $o <$
little-omega $\omega >$
big-Omega $\Omega \geq$
  - “lower bound”
big-Theta $\Theta =$