Proof of Correctness

Total Correctness: Termination and Partial Correctness
Partial Correctness: Loop invariants and induction
Loop Invariant: A property that holds before and after each iteration of a loop
Initialization: The loop invariant holds before the first iteration
Maintenance: If the loop invariant holds before an iteration, it holds after the iteration
Termination: When loop terminates, invariant gives useful property to show the algorithm is correct
Iterative: Usually loop invariants
Recursive: Usually induction

Complexity

$a^{\log_b x} = x^{\log_b a}$
$\log_b x = \frac{\log_c x}{\log_c b}$
$\log_b M \cdot N = \log_b M + \log_b N$
$\log_b \frac{M}{N} = \log_b M - \log_b N$
$\log_b M^k = k \log_b M$
Big Oh: $f(n)$ is $O(g(n))$ if $f(n) \leq cg(n)$ for $n \geq n_0 : c, n_0 > 0$
Big Omega: $f(n)$ is $\Omega(g(n))$ if $f(n) \geq cg(n)$ for $n \geq n_0 : c, n_0 > 0$
Big Theta: $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$
Little Oh: Strict Big Oh
Little Omega: Strict Big Omega
$\lim_{n\to\infty} \frac{f(n)}{g(n)}$:
0 if $f(n)$ is $o(g(n))$, ∞ if $f(n)$ is $\omega(g(n))$
$< \infty$ if $f(n)$ is $O(g(n))$, > 0 if $f(n)$ is $\Omega(g(n))$
$0 < \infty$ if $f(n)$ is $\Theta(g(n))$
Growth Rates: $1 < \log(n) < \sqrt{n} < n < n\log(n) < n^2 < n^c < 2^n < c^n < n! < n^n$
Harmonic: $\sum_{k=1}^{n} \frac{1}{k} \sim \ln n$
Triangular: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \sim \frac{n^2}{2}$
Squares: $\sum_{k=1}^{n} k^2 \sim \frac{n^3}{3}$
Geometric: $\sum_{k=0}^{n} ar^k = \frac{a(r^{n+1}-1)}{r-1}$
Stirling's Approximation: $\log_2(n!) \sim n \log_2 n$
Master's Theorem: $T(n) = aT(\frac{n}{b}) + f(n)$, $\epsilon > 0$
$f(n) = O(n^{\log_b(a)-\epsilon}) \to T(n) = \Theta(n^{\log_b a})$
$f(n) = \Theta(n^{\log_b a}) \to T(n) = \Theta(n^{\log_b a} \log n)$
$f(n) = \Omega(n^{log_b(a)+\epsilon}) \wedge af(\frac{n}{b}) \leq cf(n)$ for some $c < 1 \to T(n) = \Theta(f(n))$

Divide and Conquer

Divide: Break problem into smaller subproblems
Conquer/Combine: Solve subproblems recursively and combine

Recurrence Relation

A function defined in terms of itself
Mirrors the recursive algorithm it represents
Analysis of the running time of a divide and conquer algorithm generally involves solving a recurrence relation
1,2,3 Method

Merge Sort

$T(n) = 2T(\frac{n}{2}) + n - 1$, $T(1) = 0$

Level	Problem Size	Total Time
0	n	n
1	$\frac{n}{2}$	$2\frac{n}{2} = n$
2	$\frac{n}{4}$	$4\frac{n}{4} = n$
$\vdots$	$\vdots$	$\vdots$
k	$\frac{n}{2^k} = 1$	$2^k \frac{n}{2^k} = n$

$\to (\sum_{i=0}^{k-1} n) + 0 \cdot 2^{log_2 n} \to \sum_{i=0}^{log_2 n - 1} n \to n \log_2 n$
or $\to \sum_{i=0}^{k} n \to \sum_{i=0}^{log_2 n} n \to n \log_2 n$

Closest Pair

Dynamic Programming

Weighted Interval Scheduling

Memoization

Subset Sum/Knapsacks

Sequence Alignment

Bellman-Ford

Network Flow

Maximum Flow Problem

Ford-Fulkerson

Max Flow/Min Cut