

## 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

D&C: Show recurrence is optimal inductively by showing sub-problems generate optimal solutions

DP: Show recurrence is optimal by description of optimal substructure then show algorithm implements the recurrence

## 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

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}:$$

0 if  $f(n)$  is  $o(g(n))$ ,  $\infty$  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:  $T(n) = aT(\frac{n}{b}) + f(n)$ ,  $\epsilon > 0$

$f(n) = O(n^{\log_b(a)-\epsilon}) \rightarrow T(n) = \Theta(n^{\log_b a})$

$f(n) = \Theta(n^{\log_b a}) \rightarrow T(n) = \Theta(n^{\log_b a} \log n)$

$f(n) = \Omega(n^{\log_b(a)+\epsilon}) \wedge af(\frac{n}{b}) \leq cf(n)$  for some  $c < 1 \rightarrow T(n) = \Theta(f(n))$

## Divide and Conquer

Divide: Break problem into smaller independent sub-problems

Conquer/Combine: Solve sub-problems recursively and combine

## Recurrence Relation

A function defined in terms of itself(recursively)

Analysis of D&C generally involves a recurrence relation

1,2,3 Method:  $T(n) = \sum^{levels} \text{time per level}$  or  $\sum^{levels-1} \text{time per level} + (\text{base case value} \cdot \# \text{ of leaves})$

## 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	$\frac{n}{4}$	$4 \frac{n}{4} = n$
$\vdots$	$\vdots$	$\vdots$
$k$	$\frac{n}{2^k} = 1$	$2^k \frac{n}{2^k} = n$

$$\rightarrow \left( \sum_{i=0}^{k-1} n \right) + 0 \cdot 2^{\log_2 n} \rightarrow \sum_{i=0}^{\log_2 n - 1} n \rightarrow n \log_2 n$$

$$\text{or } \rightarrow \sum_{i=0}^k n \rightarrow \sum_{i=0}^{\log_2 n} n \rightarrow n \log_2 n$$

## Closest Pair

D&C by splitting plane in half by median then checking in rectangles

$O(n \log n)$

## Dynamic Programming

Overlapping and dependant sub-problems

## Weighted Interval Scheduling

j- intervals sorted by latest finishing time

p(j)- interval that immediately precedes j without overlap

$$\text{OPT}(j) = \max(\text{OPT}(j-1), v_j + \text{OPT}(p(j)))$$

Can use recursive, top-down approach(memoization) or iterative, bottom-up(tabulation)

n = number of intervals

$O(n \log n)$  to sort,  $O(n)$  for algorithm

## Subset Sum

i- index of item, W- max weight

$\text{OPT}(i, W) = \text{OPT}(i-1, W)$  if  $W < w_i$

else  $\max(w_i + \text{OPT}(i-1, W - w_i), \text{OPT}(i-1, W))$

n = number of items, W = sum

$O(n \cdot W)$ , pseudo-polynomial

## 0/1 Knapsack

Maximize value instead of weight

i- index of item, v- value, W- max weight

$\text{OPT}(i, W) = \text{OPT}(i - 1, W)$  if  $W < w_i$   
 else  $\max(v_i + \text{OPT}(i - 1, W - w_i), \text{OPT}(i - 1, W))$

### Sequence Alignment

$\delta$ - gap cost,  $\alpha$ - alignment cost  
 $\text{OPT}(i, j) = \min(\alpha_{x_i y_j} + \text{OPT}(i - 1, j - 1), \delta + \text{OPT}(i - 1, j), \delta + \text{OPT}(i, j - 1))$   
 $m, n$  = length of strings  
 $O(mn)$

### Bellman-Ford

$i$ - number of usable edges,  $v$ - start node,  $w$ - intermediate node  
 $\text{OPT}(i, v) = \min(\text{OPT}(i - 1, v), \min_{w \in V}(\text{OPT}(i - 1, w) + c_{vw}))$   
 $m$  = number of edges,  $n$  = number of nodes  
 $O(mn)$   
 $\text{OPT}(n, v) \neq \text{OPT}(n - 1, v) \rightarrow$  path has negative cycle

### Network Flow

Source(s): Only outgoing edges  
 Sink(t): Only incoming edges  
 Capacity( $c_e$ ):  $\in \mathbb{N}$   
 $f(e)$ : Flow through edge  $e$ ,  $\geq 0$   
 $v(f)$ : Value of flow  $f$ ,  $\sum_{e \text{ out } s} f(e)$

### Ford-Fulkerson

Implementation of max-flow problem  
 $m$  = number of edges,  $C$  = max possible flow  
 $O(mC)$

### Max Flow/Min Cut

Cut: Partition of nodes  
 $A$  = subset with  $s$ ,  $B$  = subset with  $t$   
 Cut Capacity:  $c(A, B) = \sum_{e \text{ out } A} c_e$   
 $v(f) = f^{\text{out}}(A) - f^{\text{in}}(A) = f^{\text{in}}(B) - f^{\text{out}}(B)$   
 Min cut = Max flow