Runtime Complexity
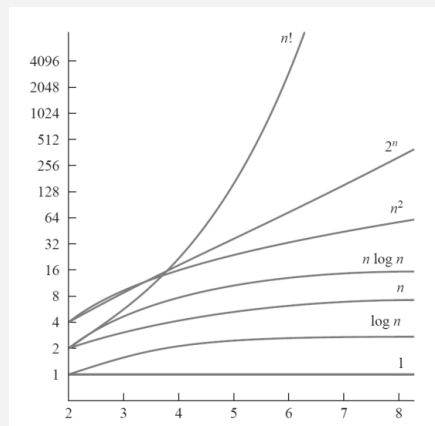

Formally: there exist constants c and n0 such that for all
sufficiently large n: $f(n) \leq c \cdot g(n)$
$c, n0\ n : n \geq n0, f(n) \leq c \cdot g(n)$

Master theorem

$T(n) = a \cdot T(n/b) + f(n)$, $a \geq 1$ and $b > 1$
let $c = \log_b a$
Case 1: (only leaves) if $f(n) = O(n^{c-e})$, then $T(n) = \Theta(n^c)$ for some $e > 0$
Case 2: (all nodes) if $f(n) = \theta(n^c \log^k n)$, $k \geq 0$, $T(n) = \theta(n^c \log^{k+1} n)$
Case 3: (only internal nodes) if $f(n) = \omega(n^{c+e})$, then $T(n) = \theta(f(n))$ for some $e > 0$

Kruskals Algorithm

Sort all edges by their weights
Loop:
- Choose the minimum weight edge and join correspondent vertices (subject to cycles).
- Go to the next edge.
- Continue to grow the forest until all vertices are connected
Runtime Complexity:
Sorting edges – O(E log E)
Cycle detection – O(V) for each edge
Total: O(V * E + E * log E)

Depth-First-Search (DFS)

It starts at a selected node and explores as far as possible along each branch before backtracking. DFS uses a stack for backtracking

Breadth-First-Search (BFS)

It starts at a selected node and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS uses a FIFO queue for bookkeeping

Amortized Analysis

Aggregate method: The amortized cost of an operation is given by $T(n) / n$
Accounting Method: We assign different charges to each operation; some operations may charge more or less than they actually cost.

Topological Sort

1. Select a vertex that has zero in-degree.
2. Add the vertex to the output.
3. Delete this vertex and all its outgoing edges.
4. Repeat

Coin Change

$opt[k,x] = \min(opt[k-1,x], opt[k,x - dk] + 1)$
Base : $opt[1,x] = x$, $opt[k,0] = 0$

01 knapsack

$opt[k,x] = \max(vk + opt[k-1, x - wk], opt[k-1,x])$
base: $opt[0,x] = 0$, $opt[k,0] = 0$
$opt[k,x] = opt[k-1,x]$ if $wk > x$

Djikstra's Algorithm

When algorithm proceeds, all vertices are divided into two groups
- vertices whose shortest path from the source is known
- vertices whose shortest path from the source is NOT discovered yet.
Move vertices one at a time from the undiscovered set of vertices to the known set of the shortest distances, based on the shortest distance from the source.
Runtime: O(V.log V + E.log V)

Heap

	Binary	Binomial	Fibonacci
findMin	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
deleteMin	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$ (ac)
insert	$\Theta(\log n)$	$\Theta(1)$ (ac)	$\Theta(1)$
decreaseKey	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)
merge	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$ (ac)

Karatsuba

$a \times b = (x1 \cdot 10^{n/2} + x0) \cdot (y1 \cdot 10^{n/2} + y0)$

Strassen Algorithm

1968

Strassen's Algorithm

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} s_1 + s_2 - s_4 + s_6 & s_4 - s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{pmatrix}$$

Prim's Algorithm

1) Start with an arbitrary vertex as a sub-tree C.

2) Expand C by adding a vertex having the minimum weight edge of the graph having exactly one end point in C.

3) Update distances from C to adjacent vertices.

4) Continue to grow the tree until C gets all vertices.

Runtime:

binary heap : $O(V.\log V + E. \log V)$

Fibonacci heap: $O(V. \log V + 1)$ (ac)

Greedy Algorithm

It is used to solve optimization problems

It makes a local optimal choice at each step

Earlier decisions are never undone

Does not always yield the optimal solution

Longest Common Subsequence

$LCS[i,j] = (1 + LCS[i-1,j-1])$ if $s[i] = s[j]$

$LCS[i,j] = (max(lcs[i-1,j] , max[i,j-1])$ if $s[i] != s[j]$

base case: $lcs[i,0] = lcs[0,j] = 0$