is 3n-4 ∈ Ω(n²)?

assume that cn² ≤ 3n-4 ≤ 3n for all n ≥ n₀; cn² ≤ 3n

cn ≤ 3; n ≤ 3/c for all n ≥ n₀; impossible

**BRGC(3) =** **['000', '001', '011', '010', '110', '111', '101', '100'];** T(n) = Θ(n2ⁿ)

russian peasant multiplication: Θ(lg(min(n,m)))

runs faster if n < m, so swap if necessary

until n = 1: divide n by 2, multiply m by 2

for all even n, remove its corresponding value of m

sum the remaining values of m

Given T(n) = aT(n/b) + Θ(nᵈ):

A math equations on a white background

Description automatically generated

A screenshot of a computer

Description automatically generated

Selection finds the next smallest element and puts it in place; Insertion moves the first unsorted element

-quicksort: best/average case is Θ(nlgn), worst case is Θ(n²) but it's rare with good pivot selection

-mergesort: recursively splits and insertion sorts the array. always Θ(nlgn) (= amt of inversions) but needs a lot of memory

-radix sort: always Θ(n\*log₁₀(max element)), does a stable counting sort (preserves the order of seemingly equal elements) on each digit

- counting sort: Θ(n + (max element))

- BFS/DFS/topo sort: Θ(V²) with matrix; Θ(V+E) with list

LomutoPartition(A, l, r): pivot = A[0]; Θ(n)

A math equations on a white background

Description automatically generatedp ← A[l]

s ← l

for i ← l + 1 to r do

if A[i] < p

s ← s + 1

swap(A[s], A[i])

swap(A[l], A[s])

return s

Quicksort(A, l, r):

if(l < r)

s ← LomutoPartition(A, l, r)

Quicksort(A, l, s − 1)

Quicksort(A, s + 1, r)

**kruskal**: O(ElgV) sort edges by weight then node letters

include edges by incr. weight UNLESS they form a cycle

find(n) tells you the furthest node from n;

if find(x) = find(y) then don't link x, y

**Prim's Algorithm**: O(V²) with matrix or O(ElgV) with list shortest edges connecting all unvisited nodes to any visited node

**floyd-warshall:**  O(V³)

- add all given weights to D0 (diagonal is all 0s)

- vertical is "from", horizontal is "to"

- while the graph still contains distances of infinity

- for vertices i, j, k in V, if D[i][j] > D[i][k] + D[k][j] then D[i][j] = D[i][k] + D[k][j]

**Dijkstra's algorithm**:

Θ(|V|²) for weight matrix + priority Q as unordered array

O(|E|log|V|) for adj. list + priority Q as minheap

like prim, except we look at TOTAL DISTANCE from source to new vertex.

**coin-row** – C = actual list of coins, F = best running sum, S = coin index of last pickup

-F(n) = max(C[n] + F(n - 2), F(n - 1)) for n > 1;

otherwise F(0) = 0, F(1) = c₁

Θ(1) running, Θ(n) backtracking

**robot coin collection** - find path thru F in Θ(i+j) time;

for all i, j in C, F[i,j] = max(F[i-1, j], F[i, j-1], + C[i, j])

Θ (rows \* columns runtime), Θ(1) backtracking

**Candies:**

case 1: child i-1 ≥ child i ≤ child i+1;

child i is a valley. they get 1 candy.

case 2: child i-1 < child i ≤ child i+1

child i is rising. they get one more candy than the last child.

case 3: child i-1 ≥ child i > child i+1

child i is falling. they get one more candy than the next child.

case 4: child i-1 < child i > child i+1

child i is a peak. they get 1 more than max(next child, last child).

A math problem with numbers

Description automatically generated with medium confidencewe will also pretend child 0 and child n+1 are worth infinity

**max sum descent:** fill in each space of the pyramid with the sum of its max parent, retrace by following max child

**0-1 knapsack** - for item i with weight wᵢ and value vᵢ and knapsack capacity j,

F(i, j) = 0 if i = 0 or j = 0

F(i, j) = max(vᵢ + F(i - 1, j - wᵢ), F(i - 1, j)) if wᵢ ≤ j ≤ W (can fit)

i.e. max(use, lose)

F(i, j) = F(i - 1, j) if 0 ≤ j < wᵢ (can't fit)

Time & space both Θ(n\*W), worst case backtracking is Θ(n) ⇛ O(n)

A diagram of a diagram

Description automatically generated**RBT Insertion:** RBT height = Θ(lgn)

Properties that could be violated during insertion:

2. The root is not black. (Make it black and check back.)

4. A node is red, but both of its children are not black. In this case:

Z is a red child of a red parent z's grandparent exists and is black:

while z.p.color == RED

if z.p == z.p.p.left // If z's parent is left child

y = z.p.p.right // z's uncle y is right child

if y.color == RED

A diagram of a network

Description automatically generated with medium confidence z.p.color = BLACK // case 1a

y.color = BLACK // case 1a

z.p.p.color = RED // case 1a

z = z.p.p // case 1a

z's uncle y is black and z is a right child:

if z == z.p.right

z = z.p // case 2a

LEFT-ROTATE(T, z) // case 2a

z's uncle y is black and z is a left child

z.p.color = BLACK // case 3a

z.p.p.color = RED // case 3a

RIGHT-ROTATE(T, z.p.p) // case 3a

**Karatsuba**:

A math equations on a white background

Description automatically generated

**Ford-Fulkerson** is O(E\*(value of max flow)), looks like E-K without specifying BFS

**Edmonds-Karp Algorithm** - O(VE²)

- G\_f (residual network) is made of all edges (u, v) where c\_f(u, v) > 0

- where c\_f = 0 unless…

- capacity of (u,v) - current flow of (u,v) if (u,v) exists {i.e. how much more we can push \*into\* the real edge}

- current flow (v,u) if the backward edge (v,u) exists {i.e. how much we can push \*out of\* the real edge}

edmonds-karp(flow network G, source s, sink t): initialize all flows to 0

while there exists a path from s to t:

do BFS to find shortest path from s to t on residual network

augment real flows on that path by the bottleneck (add to forwards, subtract from backwards)

update residual network

**stair climber:** get\_ways(n) = 1 prepended to each solution of get\_ways(n - 1) + 2 prepended to each solution of get\_ways(n - 2) + 3 prepended to each solution of get\_ways(n - 3); T(n) = 3ⁿ

