

Max. Circ Subarray: O(n)

Use kidane’s with min to find min subarray

Rest is max subarray

Max. Circ. Subarray: O(n^2)

Dynamic Prog.:

Top down or bottom up

Often optimal

Overlapping subproblems of same type

Recursive Algos:

Top down

Often optimal

Divide problem into subproblems of the same type

Combine subproblems

Greedy Algorithms:

Tend to be fast

Often not optimal

Divide problem into steps

Pick local optimum at each step

Zero sum games:

Sum of gains – sum of losses = 0

Duality Theorem:

If a linear program has a bounded optimum, then so does its dual, and they’re equal

Bipartite Graph:

Vertices can be partitioned into two sets V1,V2 such that there are no edges between vertices in the same group

Max Flow Min Cut Theorem:

Size of the maximum flow in a network equals the capacity of the smallest (s,t) cut

Ford-Fulkerson Algorithm:

Start with zero flow

Choose s🡪t path and increase flow as much as possible O(|V||E|^2)

Traveling Salesman Problem:

Brute force every tour takes O(n!)

Suggested algo: O(n^2 \* 2^n)

Network Flows:

Graph with source, sink

Every edge has max capacity

Find flow that maximizes problem

Flow in = flow out

Standard Form:

Variables all nonnegative

Constraints all equalities

Objective function to be minimized

Simplex Method:

Start at a vertex

Walk to adjacent vertex with higher value

Stop when neighbors all have lower value

Any sign 🡪 nonnegative:

replace x with (x+, x-), both >= 0

Linear Programming:

Optimization problems with an objective function and constraints that are linear functions

Equality -- > Inequality:

Convert ax = b to:

Ax <= b, ax >= b

Inequality -- > Equality:

Convert AiXi <= b into AiXi + s = b where s >= b

Max < --- > Min:

Multiply coefficient of objective function by -1

Coin Change:

Build table with 0,1,2,…value as columns, different sets with coins as rows

Cell Ai,j = Ai-1,j + Ai,j-v

Edit Distance:

Minimum number of edits to transform s1 into s2

Independent Sets:

Subset that has no edges between its nodes

Floyd Warshall Algorithm:

All Pairs Shortest Paths:

Could use bellman-ford on every vertex, but this takes O(|V|^2|E|)

Instead, use the Floyd-warshall algorithm which takes O(|V|^3)

Chain Matrix Multiplication: O(n^3)

Knapsack without repetition:

Knapsack with repetition:

Longest Increasing Subsequence:

Dynamic Programming:

Break problem into collection of subproblems

Solve smallest subproblem

Move on to bigger subproblems

Shortest Path In DAG:

Linearize DAG and then:

Graph Properties:

Undirected graph is connected if there is a path from any vertex to any other

In a disconnected graph, each connected subgraph is called a connected component

Two nodes (u,v) are strongly connected if there is u->v path and v->u path

Tree with n nodes has n-1 edges

Connected undirected graph with |E|=|V|-1 is a tree

Undirected graph is tree IFF unqieu path btw any pair of nodes

Kruskal’s Algorithm:

Start with empty set

Add next lightest edge that does not create a cycle O(|E|log|V|)

Quick Sort:

Pick pivot value

Partition array such that 1st half <= pivot, 2nd >= pivot

Avg/Best: O(nlogn)

Worst: T(n) = T(n-1)+O(n) = n^2 (max or min value is pivot)

Asymptotics:

ω(…): “greater-than” limit f(n)/g(n) = 0, f(n) = o(g(n))

Ω(…): “greater than or equal” limit f(n)/g(n) = ∞, f(n) = ω(g(n))

θ(…): “equal” limit f(n)/g(n) ∈ R, f(n) = θ(g(n))

O(…): “less than or equal”

o(…): “less than”

Merge Sort:

Split into subproblems of size n/2 until units of 1, then recursively merge back

T(n) = 2T(n/2) + O(n)

Stable sorting algorithms:

Merge, counting, radix, insertion

Fibonacci:

Fn = Fn-1 + Fn-2

T(n) = T(n-1) + T(n-2) + 3

Iterative: O(n)

Recursive: O(2^n)

Prims Algo:

Edge set x is always subtree of G

X grows by lightest edge each time

Matrix Multiplication:

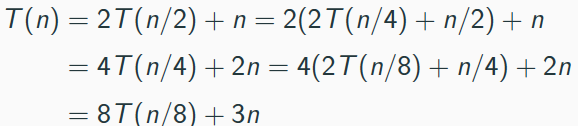
Divide into 8 (n/2 x n/2) then recursively multiply

Add resulting matrices

T(n) = 8T(n/2) + O(n^2)

Strassen’s Method:

T(n) = 7T(n/2) + O(n^2)



Huffman Encoding:

O(nlogn)

Radix Sort:

N numbers, each d digits long

Sort right to left 1 col at a time

Sort 1 col: O(n+k)

Sort d cols: O(d(n+k))

Have to use stable sort to sort columns

Horn Formulae:

Facts take form: 🡪X (empty LHS)

Find assignment of variables that make all clauses true

Adjacency Matrix:

|V| = n, then we have nxn matrix

Aij = 1 if edge from Vi to Vj, 0 otherwise

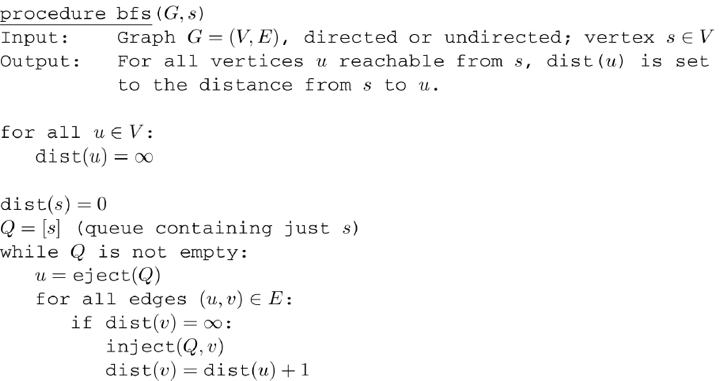
Check an edge: O(1)

Space complexity: O(n^2)

Set Cover:

Finding set of vertices that cover all vertices in graph

Greedy: until everything in graph is covered, pick node with largest # of uncovered elements



Sorting Lower Bound:

Comparison based sort has lower bound: Ω(nlogn)

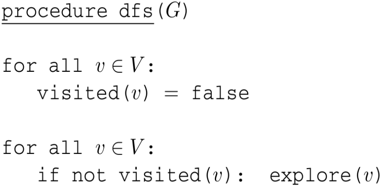
dddd

Adjacency List:

Each vertex has list of its neighbors

Check for edge: O(n)

Space complexity: O(|E|)



Insertion Sort:

Best: O(n) Worst: O(n^2)

DFS:

Only visit nodes reachable from the starting vertex

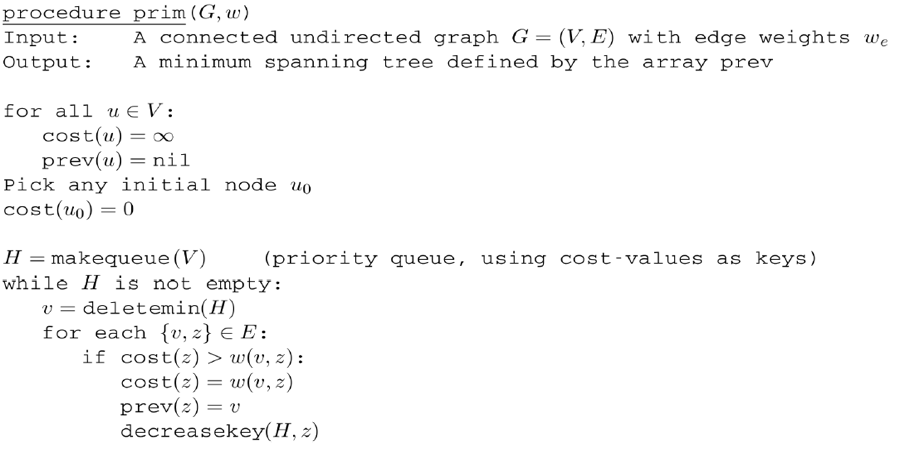
O(|V|) to mark nodes visited, call pre/post visit

O(|E|) to explore edges

Total: O(|V|+|E|)

Selection Sort:

Best/worst: O(n^2)



Unstable sorting algorithms:

Selection, quick

Types of Edges:

Tree edges: part of DFS forest

Forward-edge: node to non-child descendant

Back-edge: node to ancestor

Cross-edge: lead to neither descendant nor ancestor

Medians:

Pick v and split list into three (less than, equal to, greater)

Best: T(n)=T(n/2)+O(n) (even split)

Worst cast: v is min/max value in array

Avg: T(n)=T(3n/4)+O(n)

BFS:

Proceed layer by later

Find layer d+1 from neighbors of node at layer d

Each vertex is placed onto queue

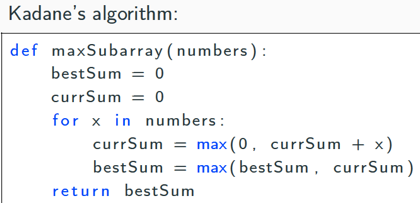
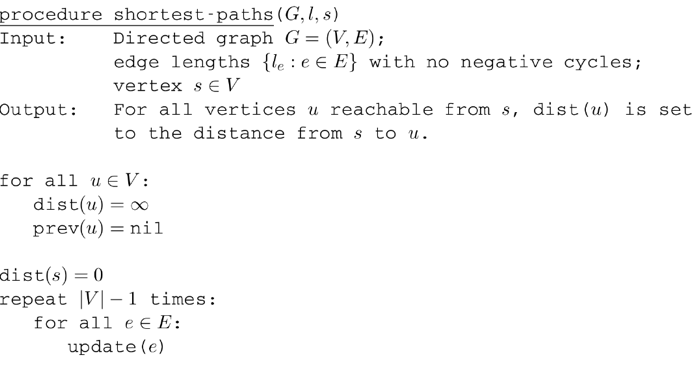
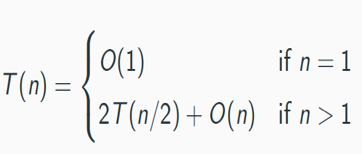
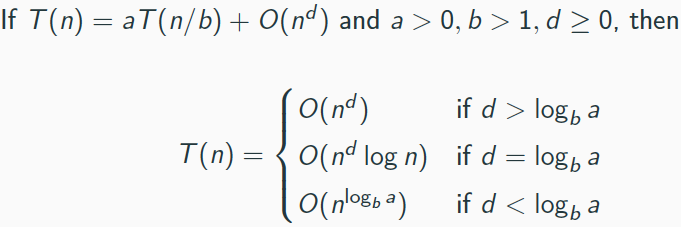
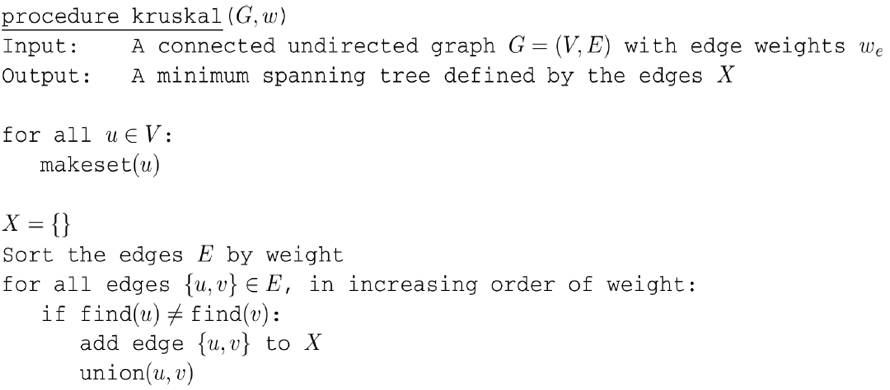
Total: O(|V|+|E|)

Divide and conquer:

Divide: split problem into subproblems

Conquer: solve subproblems recursive

Combine: combine results for solution



Kadane’s Algo:

Bellman Ford:

Computes shortest path for graphs (negative weights)

O(|V|\*|E|)

Linearize DAG:

Topologically sort graph – O(|V|+|E|)

List nodes in decreasing order of postvisit number

Minimum Spanning Tree:

Has no cycles

Connects all vertices minimum cost

Binary Search:

T(n)=T(n/2)+O(1)

Master theorem:

Maximum subarray:

Dijkstra’s Algorithm:

Given graph and starting vertex, find shortest path to all reachable vertices

|V| inserts into priority queue

|V| delete minimum from queue

|V|+|E| insert/decrease keys

Total: O((|V|+|E|)log|V|)

Bellman Ford:

Towers of Hanoi:

T(n)=2T(n-1)+O(1)

Greedy Algorithm:

Builds solution piece by piece

Picks best option at each stage