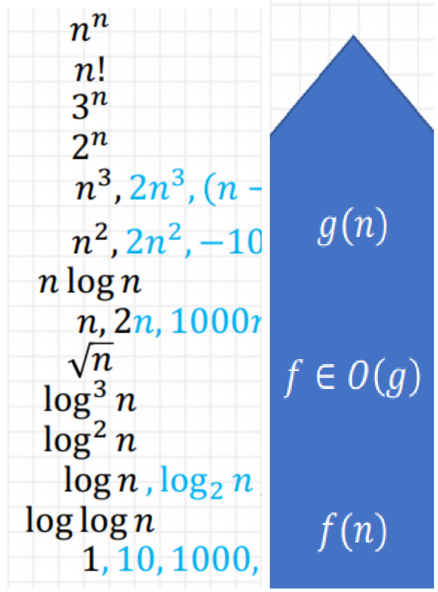
- $$T(n) = \begin{cases} \Theta(n^{\log_b a}), & \text{if } a > b^d \text{ (case} \\ \Theta(n^d \log n), & \text{if } a = b^d \text{ (case} \\ \Theta(n^d), & \text{if } a < b^d \text{ (case} \end{cases}$$
- To solve T(n), we take the result from the geometric series stuff, use K+1 instead of K, and multiply by f(n).
- Geometric series reminders
  - R =/= 1
    - $\frac{1-r^k}{1-r}$
  - R = 1
    - K
  - R < 1
    - $\frac{1}{1-r}$
  - R is $\frac{a}{b^d}$ where d is the exponent of f(n)
- $$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$
  - A is number of subproblems we make
  - B is factor by which subproblem size decreases
  - $K = \log_b n$
  - f(n) is difficulty to divide and recombine subproblems
- Limitations: Master Theor no work
  - If T(n) is not monotone
    - T(n) = sin(n)
  - If f(n) is not polynomial
    - f(n) = 2^n
  - If b cannot be expressed as a constant
- Runtimes
  - BFS: O(|V|+|E|)
  - DFS: O(|V|+|E|)
  - Rod cutting: O(n^2)
  - Bellman-Ford: O(VE)
  - Kruskal & Prim: O(|E|log|V|)
  - Dijkstra: O(V^2)
  - Floyd-Warshall: O(V^3)
- 

- **Array Sorting Algorithms**

Algorithm	Running time			Space Comple
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Big-O Properties

- Constants don't affect
- $f_1 * f_2 \text{ is } O(g_1 g_2)$
- $f_1 + f_2 \text{ is } O(\max(g_1, g_2))$
- If f is O(g), and g is O(h), then f is O(h)

Solving NP-Complete:

- Show problem A is within NP
- Find an NP-complete problem B
- Reduce A to be B

Adjacency Matrix

- Space: n^2 elements for n vertices

Adjacency List

- Space: Number of edges [2*(number of edges) if undirected] + number of vertices

Breadth First Search (BFS)

- Iterative
- Stored in Queue
- Runs in O(|V|+|E|)
- Shortest path (unweight)
- Testing bipartiteness
- Tree Traversal
  - Level-order

Depth First Search (DFS)

- Recursive or Iterative
- Stored in Stack
- Runs in O(|V|+|E|)
- Topological sorting
- Strongly Connected Components
- Tree Traversal
  - In-order, Pre-order, Post-order

Definitions

- Strongly Connected Components (SCC):
  - Only in directed graphs, things are considered

strongly connected if you can reach u from v and v from u. (mutually reachable).
    - Can be determined by running DFS on every vertex, if every vertex is found on every DFS, then strongly connected
- DAG: Directed Acyclic Graph, there are no directed cycles
    - If graph G has a topological ordering, then it also is a DAG and vice versa.

BFS: Testing Bipartiteness

- An undirected graph G = (V,E) can be called bipartite if the nodes on the graph can be colored with 2 colors in such a way that no nodes of the same color directly connect to one another.
- If there is an odd-length cycle in the graph, it cannot be bipartite
- One of the two following is true for determining bipartiteness
    - No edge of G joins two nodes of the same layer, and G is bipartite.
    - An edge of G joins two nodes of the same layer, and G contains an odd-length cycle (and hence is not bipartite).
- We can modify the BFS algorithm to color each neighbor with the opposite color when it explores a node.
- If a neighbor has already been colored (i.e., visited), and has the same color, then return false.
- If the BFS can traverse the entire graph and color all nodes, then return true.

How to Obtain Topological-sort:

- Call DFS to compute finishing times for each vertex v.
- As each vertex is finished, insert it onto the front of a linked list
- Return the linked list of vertices
- Can also be done using BFSstarting from a node with no entering edge (no edge goes into this node, it is like a base root for the rest of the graph)
- Both of these run in O(m+n)

Minimum Spanning Tree (MST)

- Kruskal's
    - Add a safe edge to the tree that is the lightest edge connecting two distinct components (one of them must not already be in the tree)
- MSTs are always acyclic
- Prim's
    - Add a safe edge to the tree by selecting the least-weight edge connecting the tree to a vertex not already in the tree.
- Floyd-Warshall
    - Run the img shown after setting every vertex to 0 and edge to its weight. Rest to inf

```
for k from 1 to V
  for i from 1 to V
    for j from 1 to V
      if dist [i][j] > dist [i][k] + dist [k][j]
        dist [i][j] ← dist [i][k] + dist [k][j]
      end if
```

Bellman-Ford

- Iterate at most V - 1 times
- Measures distance from one node to all nodes
- On each iteration, check if the distance we have listed to a neighboring node, or the distance to the examined node plus the distance between neighbor and examined in less, and mark down the minimum of the two.
- If nothing changes on a given iteration, the algo is finished.

Ford-Fulkerson

- Max flow is equal to capacity of min cut in graph.
- Residual Graph
    - At a given step, direct back all used up capacity in graph, leave rest going forward. Augment path to get more flow into graph.

Random Facts:

- BFS and DFS same if ran on tree
- Minimum weight edge always in some MST
-