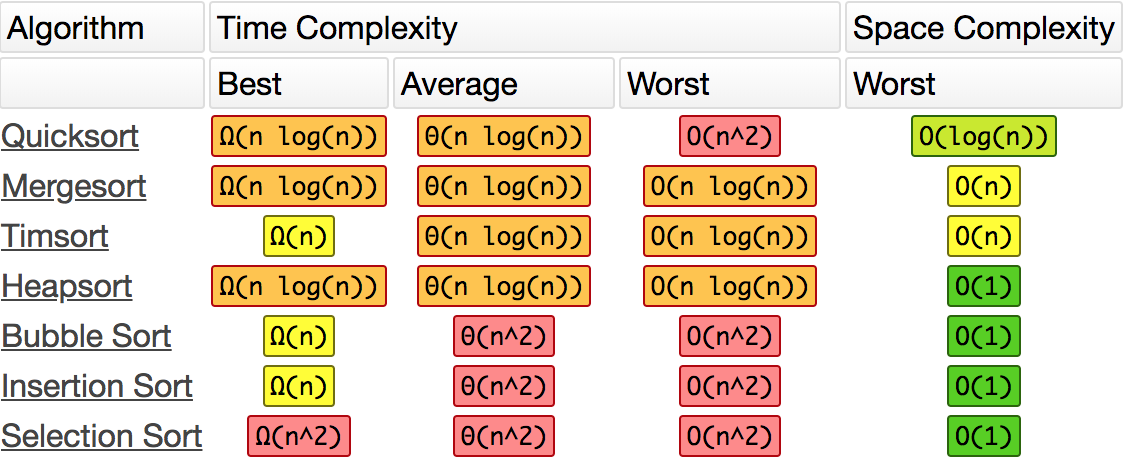
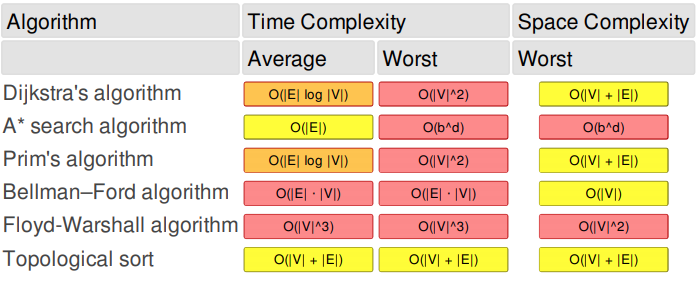
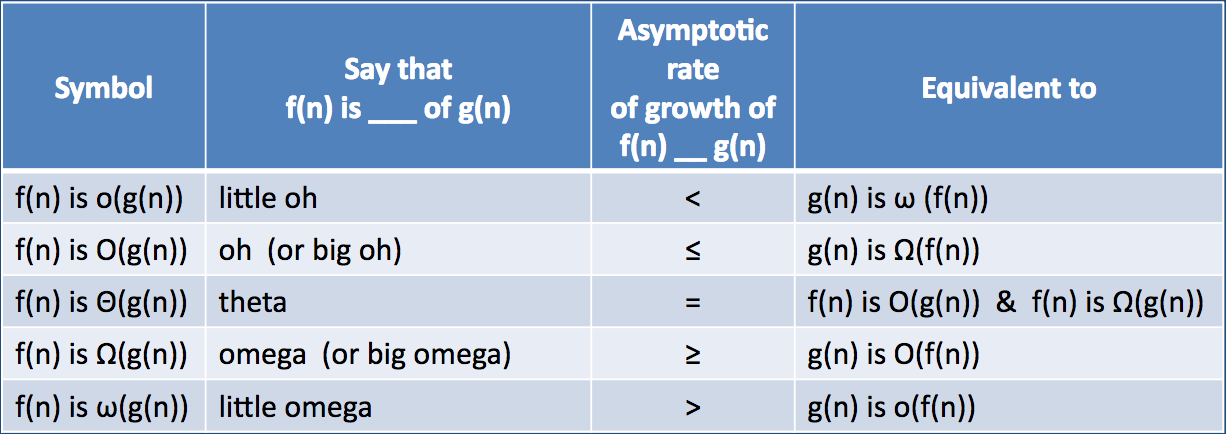
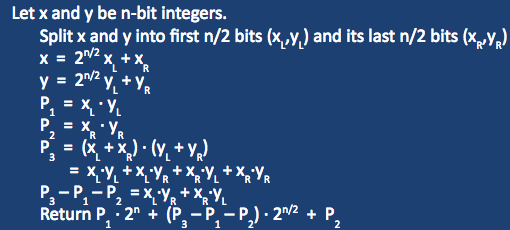
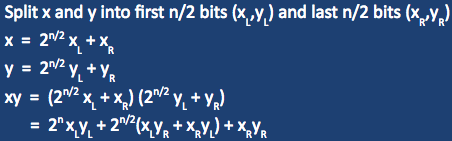
## /Users/brianrodriguez/Desktop/Screen Shot 2018-02-28 at 4.11.50 PM.png/Users/brianrodriguez/Desktop/Screen Shot 2018-02-28 at 4.17.04 PM 1.png/Users/brianrodriguez/Desktop/Screen Shot 2018-02-28 at 3.45.01 PM.png

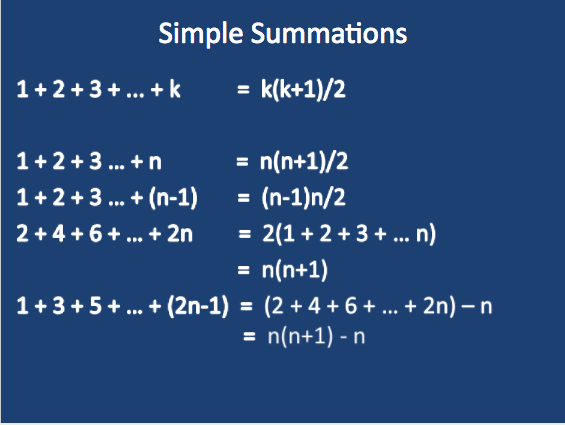


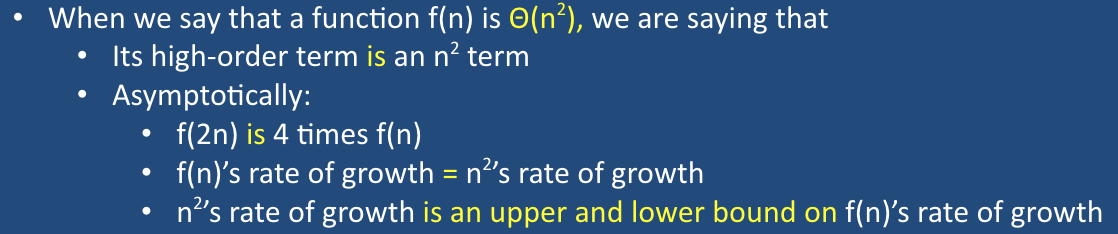


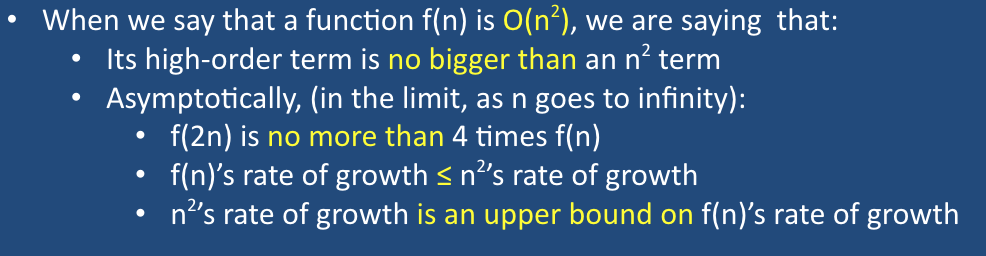


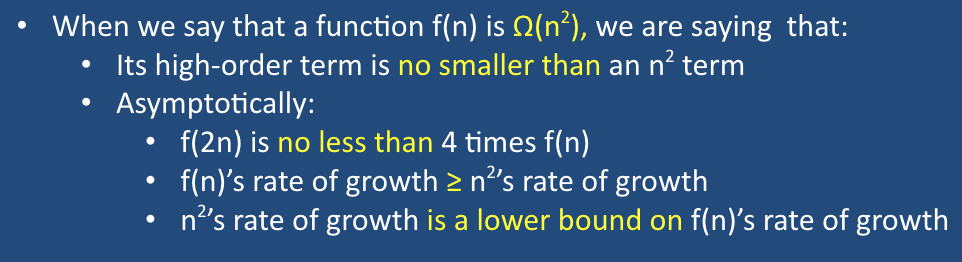
../../../Desktop/Screen%20Shot%202018-02-28%20at%206.06.04%20PM.png





Karatsuba: n^1.59 Recursive Mult: n^2





**Explore**: only reaches every vertex that is reachable through the edges of the vertex being visited.

**DFS**: Does an explore on all vertices that haven’t been visited by explore or DFS. Complexity: Matrix (V^2) List O(V+E)

**Connected** **Components on Undirected Graphs**: Identifies the largest connected “clumps” in the graph by assigned an ID of the same value to all the connected components in the “clump”

**Strongly Connected Components on Directed Graphs**: Largest possible “chunks” of a directed graph in which every vertex is connected to every other vertex is a SCC. To determine (meta-sinks and meta-sources)/SCCs reverse the graph and do a DFS. The meta source vertexes in the reversed graph will be meta sink vertexes in the original.

**BFS**: Does a search of a source vertex and adds the edges to a queue. Popping form the queue, once the BFS has searched all reachable vertices and recording the distance from the start, the algorithm stops. Complexity: Matrix O(V^2) List (V+E)

**Shortest Path Algorithms**:

**Update Operation**: update(u, v, w) **->** if dist[v] > dist[u] + w **->** dist[v] = dist[u] + w **->** prev[v] = u

**Bellman-Ford**: For all edges and vertices set dist to infinity and prev to null. Set starting vertex dist to 0. Then repeat for V-1 times **-->** for each (u, v, w) in E **-->** update(u,v,w)…. Complexity: Matrix(V^3) List (V^2 + VE)

**DAG:** Linearize the DAG and update the edges out of each vertex, considering the vertices in linearized order. Complexity: Matrix O(V^2) List (V+E)

**Dijkstra**: Use a PQ (either a min or max heap) and for each (u,v, w) in U update depending on weight if update then insert or change the value of v, dist[v]. This operation changes the PQ, be careful to update the values in the PQ.

**Graphs in the Wild**:

**Implementation**: Don’t build graph completely and then search it, start with on vertex and generate the graph and edges as necessary. **Pouring problem**: Directed edge between two vertices if a single pouring transforms into the other then create the edge and vertex respectively.

**Modular Arithmetic**:

x mod N = x + kN, where k is an int <= x + kN < N. In other words, the result of adding a multiple of N to x to get into the right range. **Ex**: 101 mod 4 = 101 – 25(4) = 1. **Ex**: -101 mod 4 = -101 mod 26(4) = 3. Answer must be between 0 <= ans <= N

**Addition**: (x+y) mod N, compute x+y then subtract N if sum is >= N complexity if O(logn)

**Multiplication**: (xy) mod N, compute xy, divide by N, report remainder. Complexity O(n^2)

**Exponentiation**: x^y mod N, start with x and multiply x by y-1 times, take result mod N. **Fast Tech**: 4^8 mod 7 => x^8 = ((x^2)^2)^2 => 4^2 mod 7 = 16 mod 7 = 2 => 2^2 mod 7 = 4 mod 7 = 4 => 4^2 mod 7 = 2

**Blended Algorithms:** Cannot use blended algorithms if the first algorithm is not divide and conquer. The second algorithm can be either not divide and conquer or it can.

**Tail Recursion:** If a method returns immediately after a recursive call. T tail recursive call can be mechanically converted into a loop. => Quickselect on avg is O(n) worst case (n^2).

**Majority Elements**: An element x is an array’s majority element if more than half of the array’s elements are equal to x.

**Graph asides:** Cyclic means graph contains at least one cycle and Acyclic means graph contains no cycles. Sparse graph means E is O(V) and dense means E is Ω(V^2)

**Asides**: first k powers of 2: 2^(k+1)-1. Pre-times mean nothing. Log(n!) = nLogn