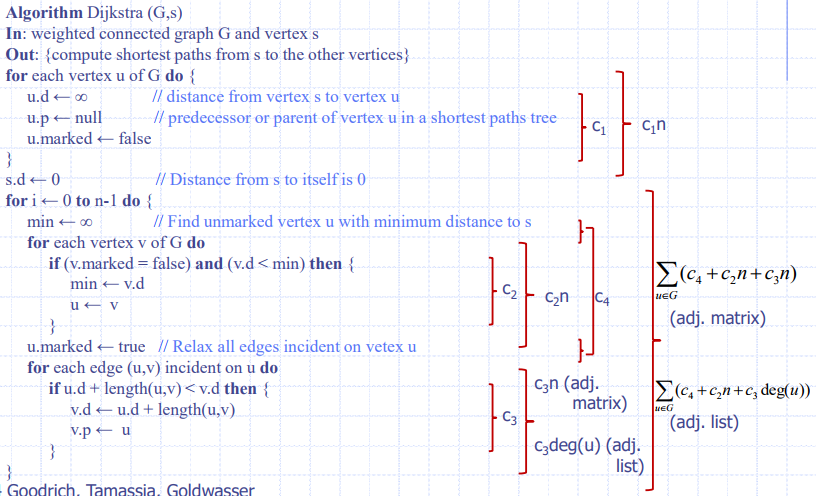
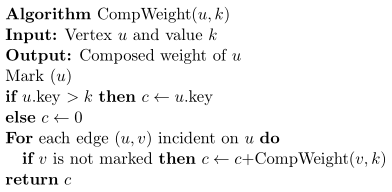
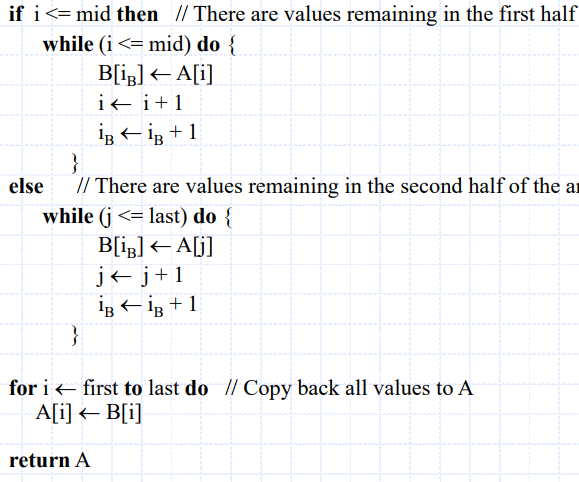
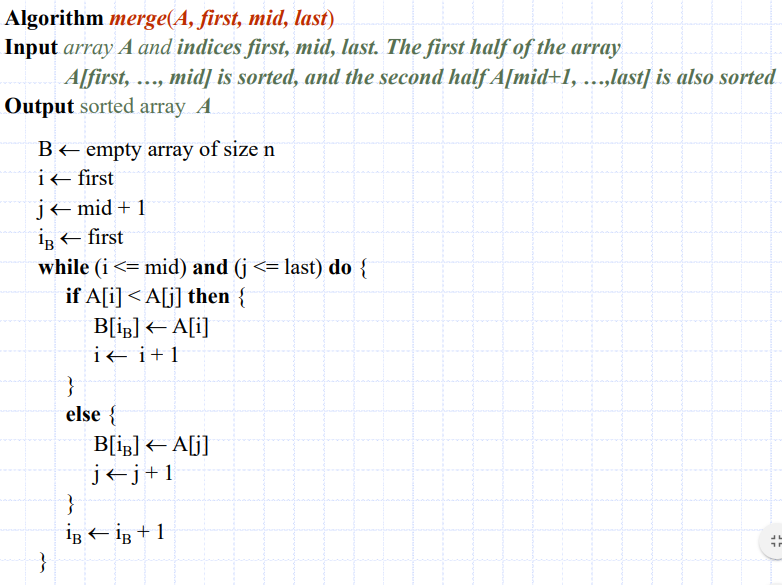
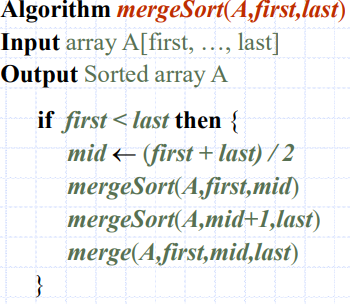
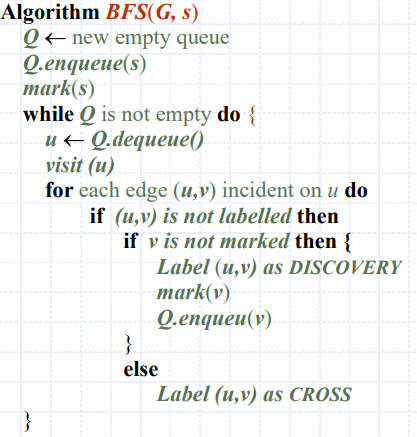
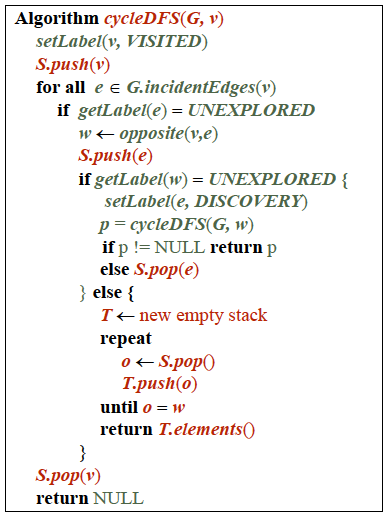
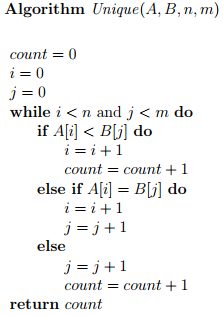
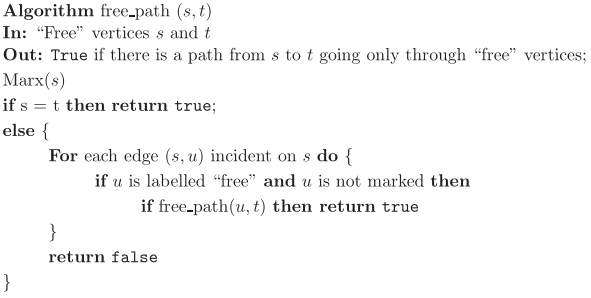
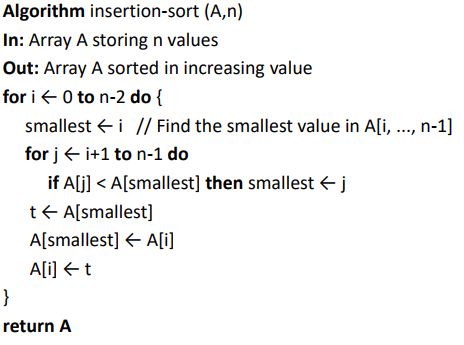
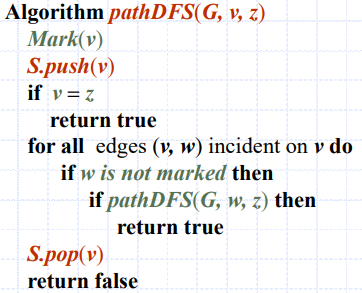
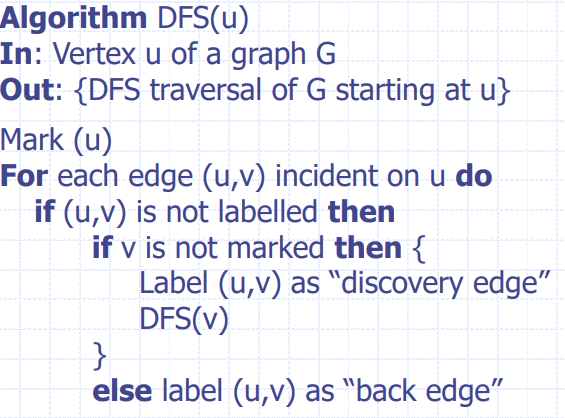
(**For Prims: remove u.d from last if statement and don’t add u.d to v.d in that if statement)**

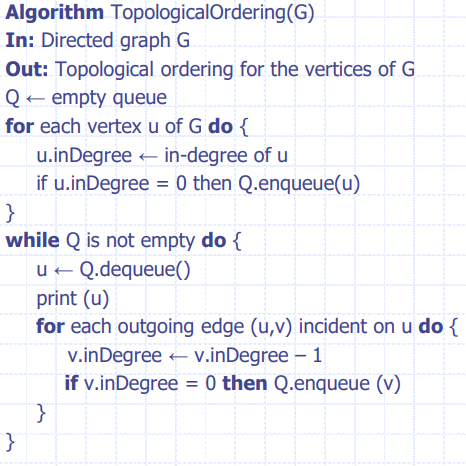
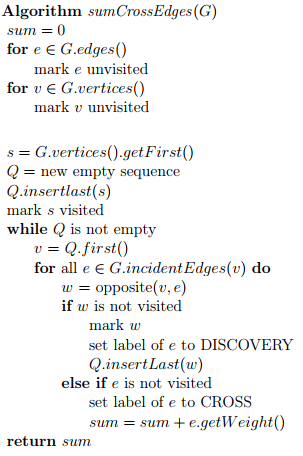
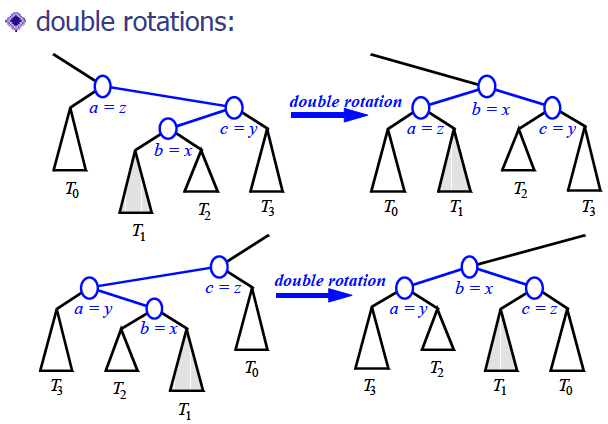
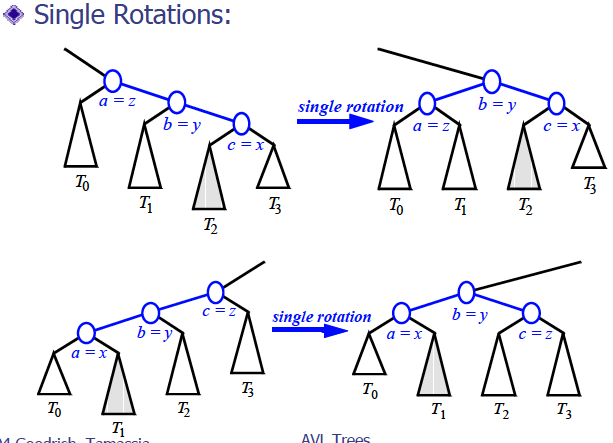
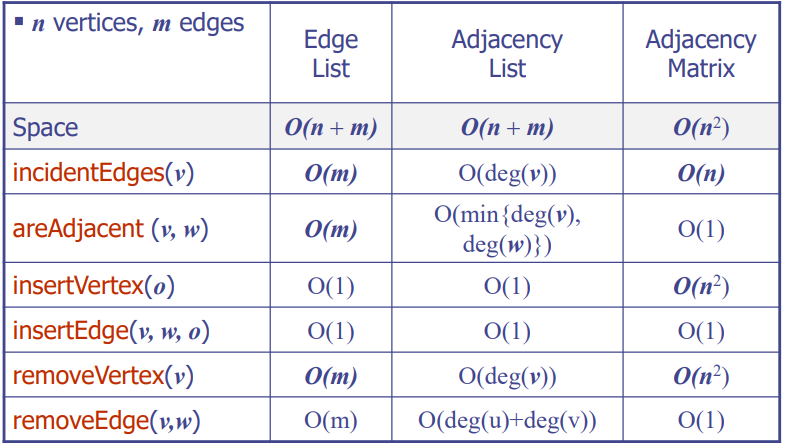
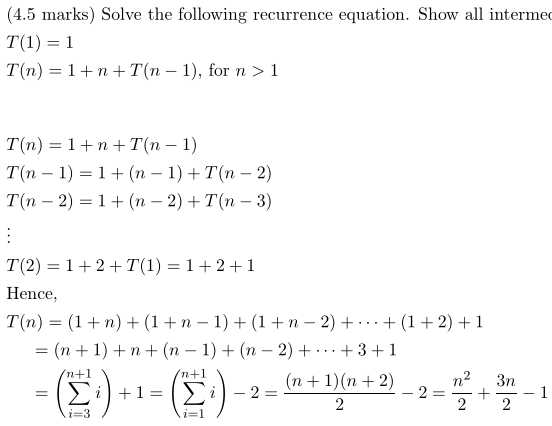
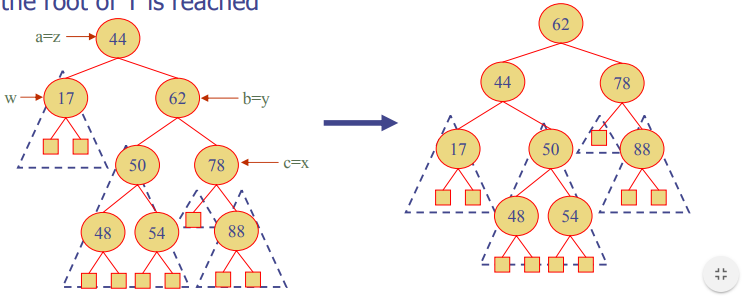


**O(n2)**



|  |  |
| --- | --- |
| **DFS/BFS adjacency list** | O(n+m) |
| **DFS/BFS adjacency matrix** | O(n2) |
| **Selection sort (unsorted array)** | O(n2) |
| **Insertion sort (sorted array)** | O(n2) |
| **Either sort ^ using AVL/(2,4) tree** | O(nlogn) |
| **Merge sort** | O(nlogn) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **BST** | **AVL** | **Multi-Way** | **(2,4)** | **B** |
| **find** | o(height) | o(logn) | o(logd\*height) | o(logn) | O(logd\*logdn) |
| **smallest** | o(height) | o(logn) | o(height) | o(logn) | O(logdn) |
| **largest** | o(height) | o(logn) | o(height) | o(logn) | O(logdn) |
| **Success.** | o(height) | o(logn) | o(logd\*height) | o(logn) |  |
| **predeces** | o(height) | o(logn) | o(logd\*height) | o(logn) |  |
| **insert** | o(height) | o(logn) | o(logd\*height) | o(logn) | O(d\*logdn) |
| **remove** | o(height) | o(logn) | o(logd\*height) | o(logn) | O(d\*logdn) |



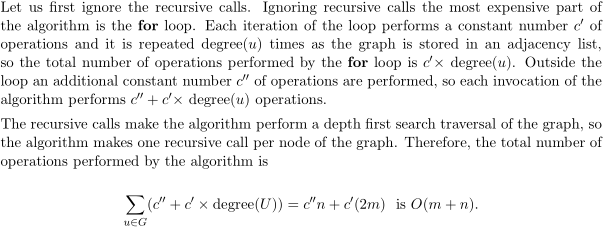
**Rebalancing after removal:**w=parent of removed nodez=first unbalanced node encountered while travelling up the tree from w.  
y=child of z with larger height  
x=child of y with larger height.

**RL**

**LR**

**RR**

**LL**



CompWeight

**GRAPH TERMINOLOGY:**

**Path:** sequence of adjacent vertices

**Simple path:** path s.t. all vertices and edges are distinct

**Simple cycle:** cycle s.t. all vertices are distinct except 1st/last

**n vertices, m edges, deg(v) degree of vertex v  
∑verticesdeg(v) = 2m**

**Complete graph:** all vertex r connected to every other vertex

**Tree:** connected graph without cycles (m = n – 1)

**Forest:** set of trees (m <= n-1)

**Subgraph:** subset of vertices/edges that forms a graph

**Connected Graph:** path from each vertex to each vertex

**Connected component:** Maximal connected subgraph

**Spanning:** contains all vertices of original graph

**Digraphs:** if simplem ≤ n(n-1).

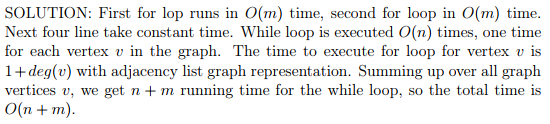
keep in-edges and out-edges in separate adj. lists to perform edge listing in time proportional to their size.   
Directed DFS determines vertices reachable from s.

**Strong connectivity:** each vertex can reach all other vertices

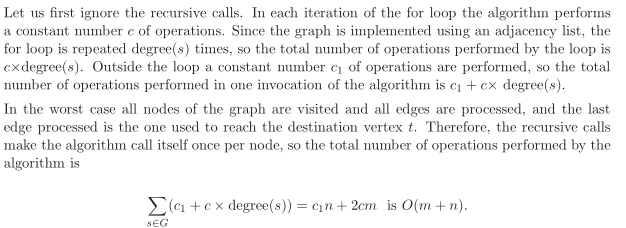
**DFS:** path between two vertices, spanning forest, connected components, paths, cycles  
**BFS:** shortest path between two vertices, spanning forest, connected components, paths, cycles

**Dijkstra:** minimum weight path between start vertex and all other vertices. Only works if graph is connected, and edges are undirected and non negative.

**Prim’s Algorithm:** computes minimum spanning tree (reach every vertex with minimum total edge weight)  
**Topological Sorting:** sorting s.t. for every directed edge (vi,vj), i < j. Vertices with no in-edges will be outputted first.



sumCrossEdges



free\_path