**1. Intro**-

(1) Why Study? a. Important for other branches b. driver for innovation c. lens on other science d. be better programmer e. sharpen analytical skills f. think algorithmic

(2) Program Workflow: 5 Stages- 1. Application Stage 2. Design Stage 3. Algorithm Analysis Stage 4. Implementation Stage 5. Experiment Stage.

**2. Algorithm Analysis-**

(1) Quest for Efficiency: a. Computational complexity: not based on speed of computers, but good algorithms and organization b. efficient program design c. Four Principles: 1. Each structure and algorithm has benefits and costs. 2. Tradeoffs 3. Should know enough about common practice 4. Data structure follow application needs d. Measuring performance

(2) Experimental studies: 1. Programming workflow. #1 is inefficient. #2 predicts resources (**time,** space) needed. Our focus is capability of the machine.

(3) Growth Rate Analysis:

- Primitive Operations: Assigning value, following object reference, performing arithmetic, comparing, accessing element, calling method, returning.

**- Speed from slowest to fastest:** Exponential, Cubic, Quadratic, NlogN, Linear, Log, Constant.

- The dominant term is the one that makes the program the slowest.

- Tips to do when analyzing: Go line by line, calculate cost and how many times it is repeated and comment on what it does. Reminder: loops are n+1 (for failure).

(4) Asymptotic analysis: **Big-O notation**- how fast does function grow in relation to input size. Upper bound, the worst case runtime. Max time.

**Properties:** speed from slow to fast: n!, 2n, n3, n2, nlogn, n, logn, 1.

To find constant, divide the two functions and its everything bigger than intersection.

**Big-Omega:** lower bound of an algorithm. Best case runtime

**Big-Theta:** tight bound, average case, between omega and o.

**3. Algorithm Design Patterns:** Recursive calls

**4. Linear Data Structures:** arrays (Static data), singly linked lists(Dynamic data). Static size has to be set and cant be changed, dynamic is free to change.

**5. Sorting Algorithms**:

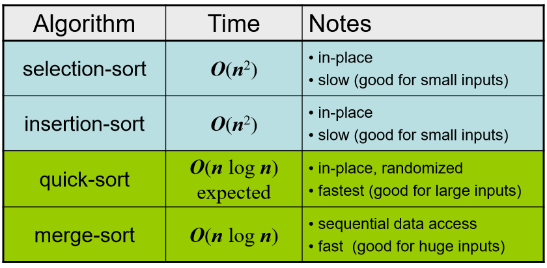
Selection Sort: finds the largest value, puts it in the last element, finds the next largest and puts it in the one before the last and so on. N2 for worst and average scenarios, uses n2/2 comparisons uses n exchanges.

Bubble Sort: bubbles up every two elements and swaps them to be in correct order. Repeats until the array is in order. (remember to have a flag to see if its already sorted). N2 for worst and average scenarios, uses n2/2 comparisons uses n2/2 exchanges.

Insertion Sort: Starts from left and moves the data over if a value on the right side is greater than the on the left. N2 for worst and average scenarios, uses n2/2 comparisons uses n2/2 exchanges for worst. N comparisons and 0 moves for best. uses n2/4 comparisons uses n2/4 exchanges for average

Merge Sort: halves the array into 2 repeatedly until it is left with one element then sorts it one by one. Recursive call. Nlogn complexity.

Quicksort: picks a partition then moves the elements accordingly. Divide and conquer. Recursive. Best case nlogn. Worst case n2(pivot is the smallest number in array)

SUMMARY:

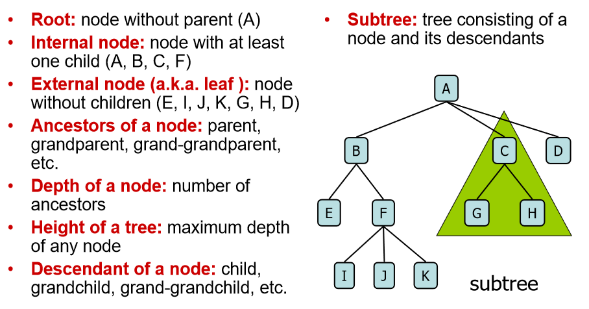
**6. Searching algorithms:**

Sequential: Best case 1, worst case n. average n. simply go one by one until it is found.

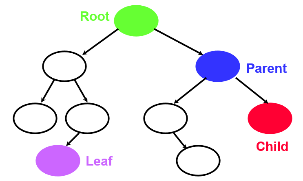
Binary: best case 1, worse case logn. Major assumption, array is sorted.

Take first, mid and last element. Check if what you are looking for is in first have or second half. Recursively call using either the first or second half repeat until found.

**7. Stacks**: First in, Last Out. Last in, First Out. Push inserts, pop removes. Use link stack rather than arrayed stack.



**8. Trees:**

Leaf: no children.

Path: number of edges to get from one node to another.

Root is at depth 0.

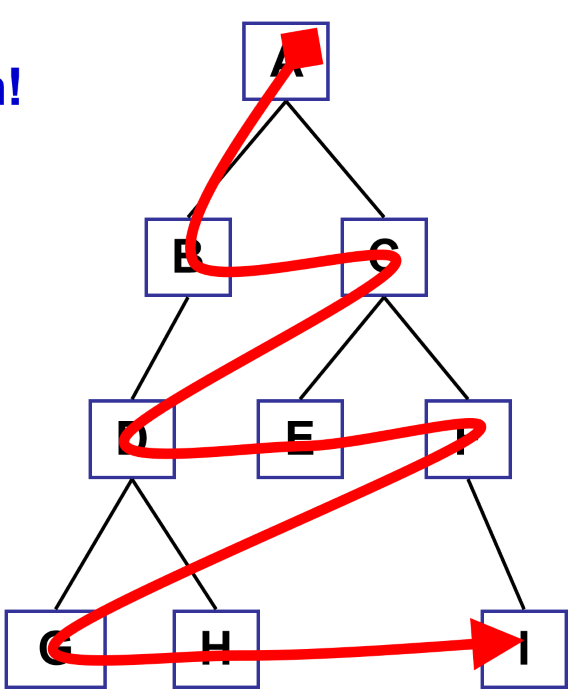
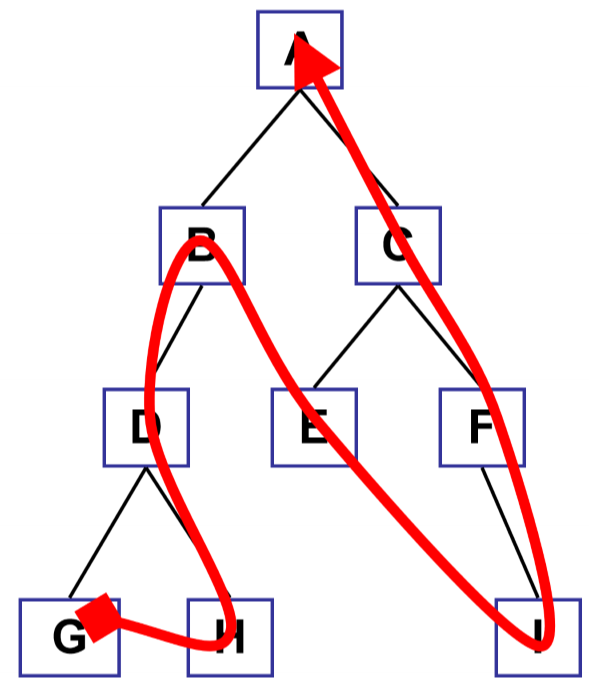
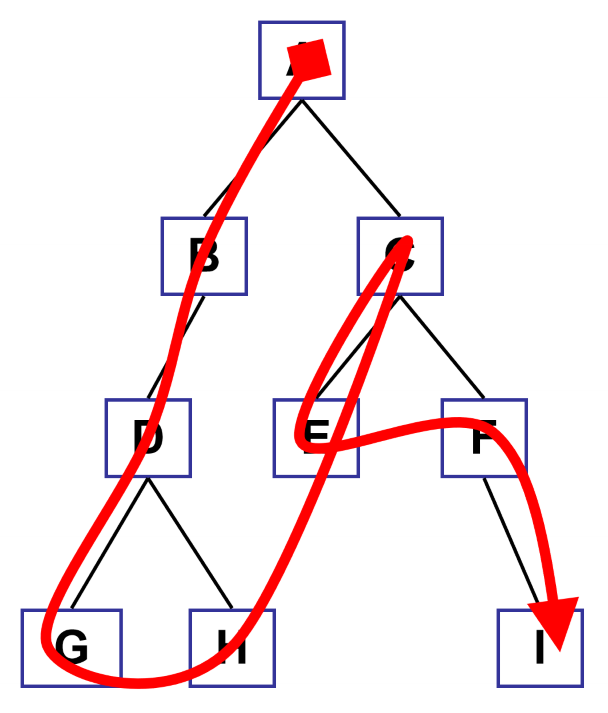
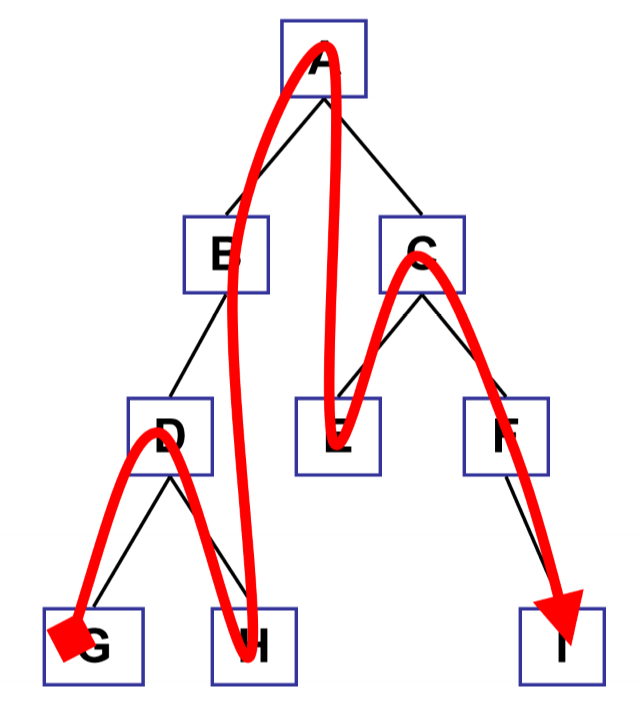
Siblings: (B,C,D)  
Height of node is from node to node, height of tree is the deepest path from root to node. A Binary tree is a tree with two children at most.

Number of external nodes: at least h+1. At most 2h. Number of internal nodes: at least h. at most 2h-1. Total number of nodes: at least 2h+1. At most 2h+1-1.

Height of tree: at least log(n+1)-1. At most (n-1)/2

**9. Binary Search Trees:**

InOrder: Left, Data, Right PreOrder: Data, Left, Right Post Order: Left, Right, Data Level Order: (Breadth First Search)



Four kinds of node deletion: a leaf node, one left child, one right child, node with two children.

**10. AVL:** Binary trees. Must be sorted. Balance factor: must be -1, 0, 1. Balance factor = right height – left height. To balance, label node, child and grandchild of the longest branch, Xa, Yc, Zb, where b becomes the new root node with a on its left and c on its right.

Simple Right Rotation: +2. Simple left rotation: -2. Guaranteed search time is logn.

**11. Heaps:** in a complete binary tree, every level except the lowest must be complete, and at deepest as far left as possible.

Three main categories: Max Heap- the parent is bigger than the children, Min Heap- the parent is smaller than the children, Max-Min: levels alternate

Not necessarily in order. It can be built in nlogn. Adding a new node: push it upwards, swapping with its parents until it reaches its proper location in a max heap. REHEAPIFACTION UPWARDS. Removing root node: move last node to root, then swap downwards until it reaches its proper location. REHEAPIFCATION DOWNWARDS

Also a max heap.

**12. Queue:** jobs with the same priority, first come first serve. Jobs with higher priority goes first.

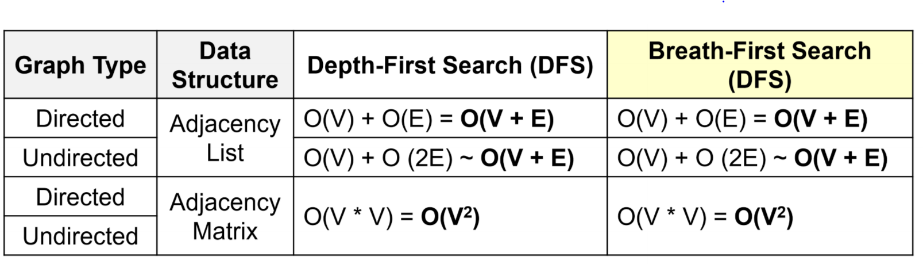
**13. Graphs: (1) Terminologies.** V- vertices. E- Edges. Directed graph- A🡪B.. (A,B) =/= (B, A). Undirected graph- A—B (A,B) = (B,A). Weighted graph- has value on the edge. Forest- graph with no cycles. Size – number of vertices in a graph. Adjacent vertices- two nodes connected to each other. Incident vertices- when an edge connected a node. Degree of a vertex- how many edges are connected to it. Regular graph- when every vertex has the same degree. Paths- to get from one node to another. Cycle- if you can go from one node to the same node using a different edge. Number of edges in a path or cycle is the length. Hamiltonian Path- a path that goes through every vertex once, but not necessarily all edges, a cycle is one that starts and ends in the same place. Eulerian path- touches every edge once, can touch a vertex more than once. Distance- how many edges it takes to get from one node to another. Isomorphic graphs- if you can redraw a graph so it looks like the other.

A complete graph- a graph where all nodes are connected to each other

Adjacency list- a linked list that lists all the nodes that are connected to a certain node.

Adjacency Matrix- a nxn matrix. Inputs a 1 or the weight when two nodes are connected.

(2)Traversals. Depth first search- visits a different node at first chance and then marks it as visited. Breadth first search – visited everything connected to a node first then others after it finishes.

Dijkstra’s Algorithm: Finding the shortest path to each node. Make sure to check each connection. There must be no cycles.

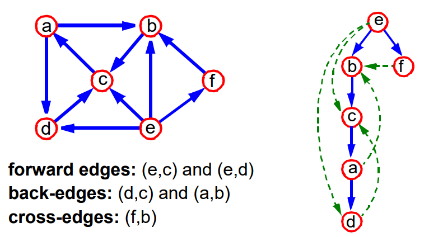
Minimum Spanning Tree: minimal total sum of edges. No cycles! You judge this one by picking the shortest distance between two nodes.

Prim’s algorithm: picking the shortest edge from one node to another.

**14. Digraphs:** a graph where all edges are directed. Only one direction

Source: all arrows go outwards. Sink: all arrows go inwards.

DFS: like an undirected graph. However, there are 3 types of edges that are not discover edges. back edges (to ancestor in DFS tree),

forward edges (to descendant in DFS tree), cross edges (to vertex that is neither ancestor nor descendant).

BFS: similar to undirected. Only two new types, back and cross, no forward edge.

Reachability: if one vertex can get to another vertex from a directed path. Ex) vertex c reaches vertices a, b and d.

Strong Connectivity: each vertex can reach all other vertex. To test pick a vertex and if something is not reached its not strongly connected. O(n+m) for this algorithm.

Transitive Closure: given G, it is a digraph G\* such that G\* has same vertices as G and if G has directed path from u to v then G\* has a directed EDGE from u to v (essentially connects the graph directly).Floyd-Warshall:an algorithm to findG\*

Directed Cycle: starts and ends at the same place. Directed Acyclic Graph (DAG): no directed cycles.

Topological ordering: a numbering vi,..,vn such that for every edge (vi,vj) i<j. Theorem: digraph admits a topological ordering iff it is a DAG. Ex) a prerequisite graph.

**15. Hash Table:** Hash function: k%(number of spaces in table).

Buckets: Having more than one space for entry in a hashed value.

Chaining: connecting each index with a linked list.

Truncation: ignore part of the tree and use the rest as an array index. Ex) taking key 1237821937 and using 937 as the hashed value

Folding: partitioning a key into several parts and combining them in a unique way. Ex) take 123123 and adding 123 and 123 to each other so hashed value is 66.

Modular arithmetic: use the modulus function to get hashed value. Ex) 1230%1000 = 230.

Generic hashed function does not exist it depends on your values.

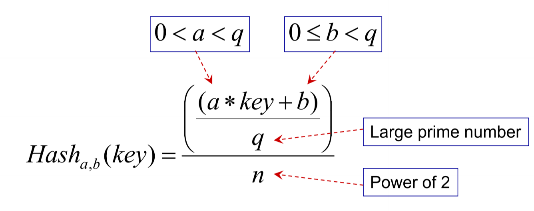
Typical has functions: modulus hash function- key%size of table. Folding: first half, last half, add, modulo. If names or letters, assign number values to them.

A collision: when two different keys are trying to be inserted into the same hashed table. A strategy: inserting it in the next open spot.

How to handle collisions: hashing with buckets- make the table size a PRIME NUMBER. Linear probing – keep increasing hash value by a set number until an open spot is found. Use a wraparound array. If returned to the same place, table is full. Rehashing: creating a new table, twice the size. Transfer old values to the new table and recomputing their positions.

Hash functions that use all the key is most of the time better than those that don’t. Worst hash function maps all the keys to the same place. Best hash maps to different addresses.

Bloom Filter:

The proportion of the table which is full is called the load factor.

Reasons to use chaining: When the number of unsuccessful searches is large. And when the record is large

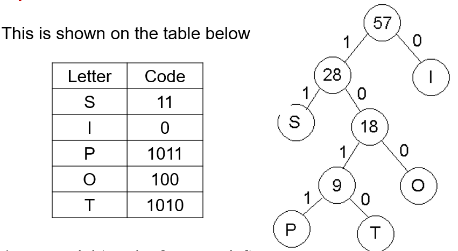
Reasons to use open addressing: most searches are likely to be successful, load factor is moderate, records are relatively small

Coalesced hash table: one column for keys, the other for the location of a key if it was forced to occupy another space.

Lessons about hashing: hash function must be carefully selected. Varies with data. Range of input should be larger than the table size. Table size should be a prime number.

Purpose of hash tables: allows extremely fast lookup given a key value. Reduce the address space of the information to the table space of the hash table.

Hash function: the reduction function. Collision: Hash(a)=Hash(b) but a!=b. collision resolution: multiple element buckets. Chaining is most general solution

**16. Text Processing:** optimal binary tree- binary trees in which the weighted path length is minimized. The weighted path length of a tree is the sum of the value in a node multiplied be the number of edges traversed to get to that node. In such tree the nodes which are used most are closest to the root of the tree and the ones that are used less appear in the bottom of the tree. For example, letters. In a given table, the frequency is all considered a node. The two smallest are combined with their parent node being the addition of the two. And the next node is connected to the parent node by addition again and so on.

Huffman Code: Label left edges with 1 and right edges with 0.

