CS 225

C++

(a) C syntax

(b) Object Oriented

(c) Big three

(d) Templates

(e) Inheritance

(f) Generic Programming (Functors)

Linear Data Structures

(a) Array

(b) Linked List

(c) Linear ADT's (usually linear)

(a) Lists

i. Runtimes:

	SLL	Array
Insert/Remove at front:	$O(1)$	$O(1)$
Insert at curr:	$O(n)$	$O(n)$
Remove at curr:	$O(1)$ Hack	$O(n)$
Insert at i:	$O(n)$	$O(n)$
Remove at i:	$O(n)$	$O(n)$

(b) Stacks (LIFO)

Push: $O(1)$
Pop: $O(1)$
Find: $O(n)$

(c) Queues (FIFO)

Push(enqueue): $O(1)$
Pop(dequeue): $O(1)$
Find: $O(n)$

Trees (asymmteric graphs)

(a) Terminology
   i. Leaf: Terminal node with not children
   ii. Root: Base node in a directed tree

(b) Traversals
   i. All tree traversals are min $O(n)$
   ii. Preorder: Parent - Child - Child
   iii. Inorder: Child - Parent - Child
   iv. Postorder: Child - Child - Parent
   v. Depth-first order: In depth-first order, we always attempt to visit the node farthest from the root that we can, but with the caveat that it must be a child of a node we have already visited. Unlike a depth-first search on graphs, there is no need to remember all the nodes we have visited, because a tree cannot contain cycles. Pre-order is a special case of this. See depth-first search for more information.
   vi. Breadth-first order: Contrasting with depth-first order is breadth-first order, which always attempts to visit the node closest to the root that it has not already visited. See breadth-first search for more information. Also called a level-order traversal.

(c) N-airy Tree
   i. Root: is a tree with a root node in which every node has at most two children.
   ii. Full: Every non-leaf node has 2 leafs
   iii. Perfect: is a full binary tree in which all leaves are at the same depth or same level, and in which every parent has two children.
   iv. Complete: is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
   v. Balanced: is commonly defined as a binary tree in which the height of the two subtrees of every node never differ by more than 1,[3] although in general it is a binary tree where no leaf is much farther away from the root than any other leaf.
   vi. BST:
      A. The left subtree of a node contains only nodes with keys less than the node's key.
      B. The right subtree of a node contains only nodes with keys greater than the node's key.
      C. Both the left and right subtrees must also be binary search trees.
   vii. AVL:
      A. In computer science, an AVL tree is a self-balancing binary search tree, and it was the first such data structure to be invented.[1] In an AVL tree, the heights of the two child subtrees of any node differ by at most one. Lookup, insertion, and deletion all take $O(logn)$ time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.
      B. Rotations $O(1)$: Look at the way you fingers move in a right hand curl
      C. Left Rotation: out the page right hand rule
      D. Right Rotation: into of the page right hand rule

viii. B-Trees (m):

A. In computer science, a B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. (Comer 1979, p. 123) Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. It is commonly used in databases and filesystems.

B. $m - 1$ children

ix. KD-tree

x. Heap:

A. Heap Sort

Graphs

(a) MST: In computer science, a B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. (Comer 1979, p. 123) Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. It is commonly used in databases and filesystems.

(b) In mathematics, a graph is an abstract representation of a set of objects where some pairs of the objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges. Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. Graphs are one of the objects of study in discrete mathematics.

(c) Traversals:

i. BST

A. Enqueue the root node.

B. Dequeue a node and examine it.

C. ——If the element sought is found in this node, quit the search and return a result.

D. ——Otherwise enqueue any successors (the direct child nodes) that have not yet been discovered.

E. If the queue is empty, every node on the graph has been examined quit the search and return "not found".

F. If the queue is not empty, repeat from Step 2.

ii. DST:

iii. Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

iv. Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that

forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm was developed in 1930 by Czech mathematician Vojtch Jarnk and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959. Therefore it is also sometimes called the DJP algorithm, the Jarnk algorithm, or the PrimJarnk algorithm.

v. Dijkstras Algorithm: conceived by Dutch computer scientist Edsger Dijkstra in 1956 and published in 1959,[1][2] is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.

Other ADTS

(a) Map(Associative Array)(Hash Maps)

(b) Priority Queues: Either Heap or Unsorted Array, Always give the min or max of the set

(c) DistJoint Sets (Up-Trees)