Computer Science 1 Exam 2

Linked Lists

Terminology

- Node - An element in the linked list that holds data and a pointer to the next node in the list
- Head - The first node in the linked list
- Tail - The last node in the linked list

General Runtimes

Function	Best Case	Average Case	Worst Case
Insert	O(1)	O(N)	O(N)
Delete	O(1)	O(N)	O(N)
Search	O(1)	O(N)	O(N)

Types

Type	Insert Head	Insert Tail	Remove Head	Remove Tail
Basic (Holds a head pointer)	O(1)	O(n)	O(1)	O(n)
Head and Tail (Holds head and tail pointers)	O(1)	O(1)	O(1)	O(n)
Circular (Holds a tail pointer that points to head)	O(1)	O(1)	O(1)	O(n)
Double Linked (Each node also points to previous node)	O(1)	O(1)	O(1)	O(1)

Stacks

Can be implemented with arrays, array lists, or linked lists

Runtime

Function	Best Case	Average Case	Worst Case
Push	O(1)	O(1)	O(n)
Pop	O(1)	O(1)	O(1)
Top	O(1)	O(1)	O(1)

Infix to Postfix

When converting from an infix expression to a postfix expression, use a stack to hold onto the operators and open parenthesis, while traversing through the expression

Scan the infix expression from left to right:

- If the element is an operand, add it to postfix expression
- If the element is an operator or parenthesis, add it to the stack maintaining the precedence of them, moving any operators at the top of the stack that take precedence over the current element to the postfix expression

Add any remaining elements in the stack to the postfix expression

Postfix Evaluation

When evaluating a postfix expression, use a stack to hold onto the operands, while traversing through the expression

Scan the postfix expression from left to right

- If the element is an operand, add it to the stack
- If the element is an operator, pop the top two elements from the stack, perform the operation on them and add the result to the stack

There will be only one element in the stack at the end if done correctly which will be the result

Queues

Can be implemented with arrays, array lists, or linked lists

Function	Best Case	Average Case	Worst Case
Enqueue	O(1)	O(1)	O(1)
Dequeue	O(1)	O(1)	O(1)
Front	O(1)	O(1)	O(1)

Flood Fill Algorithm

When implementing a flood fill algorithm, use a queue to add all locations that need to be changed and explored

1. Start with the starting location in the queue
2. Repeat process until queue is empty
    - Explore nearby elements of next location in queue and add them to queue if they are valid to be changed
    - Change current location's value to new value and pop from queue

Rooted Trees

Terminology

- Root - Start of tree that branches out to other nodes
- Parent of Node C - The first node traversed on the path from C to the root
- Child of Node P - A node that branches out from P
- Leaf - A node with no children
- Height - The highest depth in a tree
- Binary Tree - A tree where every node has either 0, 1, or 2 children and every child is either a left or right child

Traversals

Pre order

1. Check root
2. Explore left branch
3. Explore right branch

In order

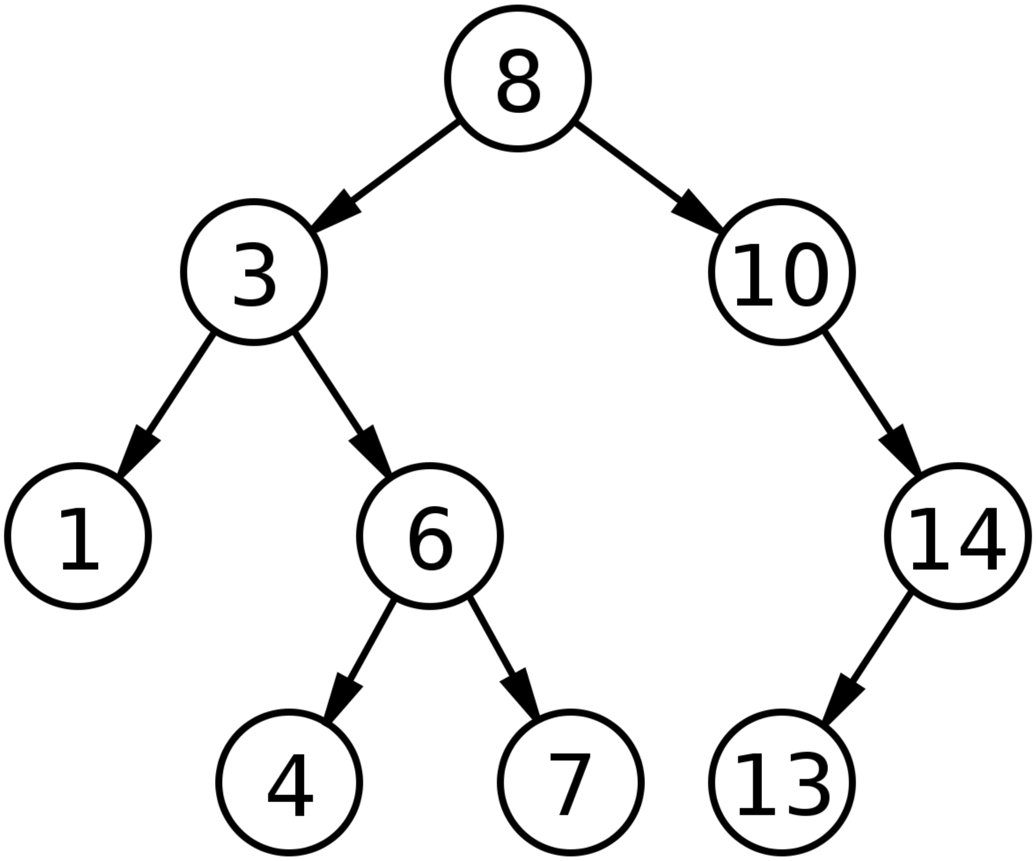
1. Explore left branch
2. Check root
3. Explore right branch

Post order

1. Explore left branch
2. Explore right branch
3. Check root

Binary Search Trees

A binary tree but for each node in the tree, every node in its left branch has a lower value and every node in its right branch has a higher value



Runtime

Function	Best Case	Average Case	Worst Case
Insert	O(1)	O(log n)	O(n)
Delete	O(1)	O(log n)	O(n)
Search	O(1)	O(log n)	O(n)
Minimum	O(log n)	O(log n)	O(log n)
Maximum	O(log n)	O(log n)	O(log n)

AVL Trees

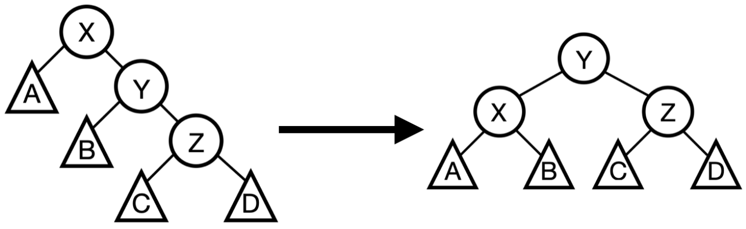
A self-balancing binary search tree where the balance factor (left branch's height - right branch's height) of each node is either 1, 0, or -1

Runtime

Function	Best Case	Average Case	Worse Case
Insert	O(log n)	O(log n)	O(log n)
Delete	O(log n)	O(log n)	O(log n)
Search	O(1)	O(log n)	O(log n)

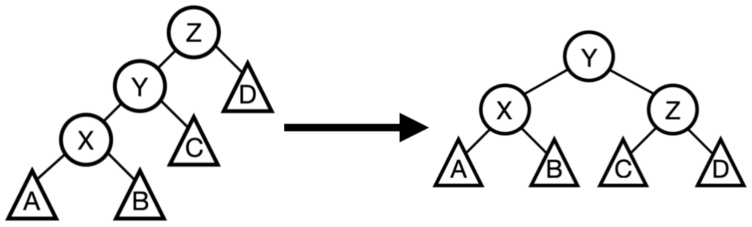
Left Rotation

- Save pointer to right child of root as rightChild
- Set right child of root to the left child of rightChild
- Set the left child of rightChild to the root so that rightChild is the new root



Right Rotation

- Save pointer to left child of root as leftChild
- Set left child of root to the right child of leftChild
- Set the right child of leftChild to the root so that leftChild is the new root



Balancing

Each unbalanced node will be one of four cases

Case 1 (Left-Left Heavy)

The left child of the root has a higher height than the right child of the root

The left branch of the left child has a higher height than the right branch of the left child

To balance: Right Rotation on root

Case 2 (Left-Right Heavy)

The left child has a higher height than the right child of the root

The right branch of the left child has a higher or equal height than the left branch of the left child

To balance: Left Rotation on left child and Right Rotation on root

Case 3 (Right-Left Heavy)

The right child has a higher height than the left child of the root

The left branch of the right child has a higher or equal height than the right branch of the right child

To balance: Right Rotation on right child and Left Rotation on root

Case 4 (Right-Right Heavy)

The right child has a higher height than the left child of the root

The left branch of the left child has a higher height than the right branch of the left child

To balance: Left Rotation on root