

## 1 Trees, Graphs, and Traversals, Oh My!

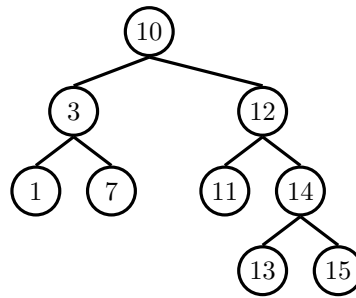
- (a) Write the following traversals of the BST below.

Pre-order:

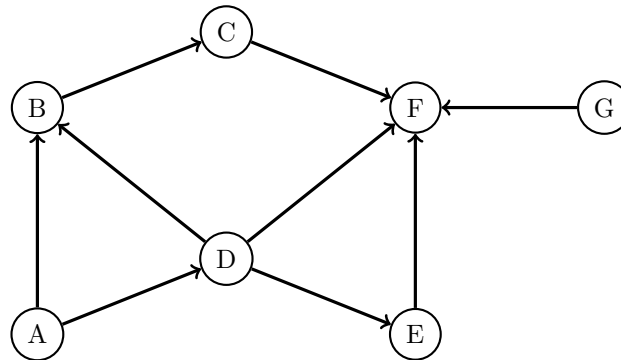
In-order:

Post-order:

Level-order (BFS):



- (b) Write the graph below as an adjacency matrix, then as an adjacency list. What would be different if the graph were undirected instead?



- (c) Write the order in which the DFS pre-order and post-order graph traversals would visit nodes in the same directed graph above, starting from vertex *A*. Break ties alphabetically.

Pre-order:

Post-order:

## 2 Absolutely Valuable Heaps

- (a) Assume that we have a binary min-heap (smallest value on top) data structure called `MinHeap` that has properly implemented the `insert` and `removeMin` methods. Draw the heap and its corresponding array representation after each of the operations below:

```
MinHeap<Character> h = new MinHeap<>();  
h.insert('f');  
h.insert('h');  
h.insert('d');  
h.insert('b');  
h.insert('c');  
h.removeMin();  
h.removeMin();
```

- (b) Your friendly TA Allen challenges you to create an integer max-heap without writing a whole new data structure. Can you use your min-heap to mimic the behavior of a max-heap? Specifically, we want to be able to get the largest item in the heap in constant time, and add things to the heap in  $\Theta(\log n)$  time, as a normal max heap should.

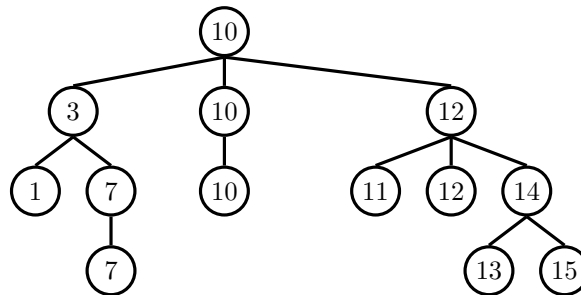
*Hint:* Although you cannot alter them, you can still use methods from `MinHeap`. The name of this question might also be a clue!

### 3 Trinary Search Tree

We'd like a data structure that acts like a BST (Binary Search Tree) in terms of operation runtimes but allows duplicate values. Therefore, we decide to create a new data structure called a TST (Trinary Search Tree), which can have up to three children, which we'll refer to as **left**, **middle**, and **right**. In this setup, we have the following invariants, which are very similar to the BST invariants:

1. Each node in a TST is a root of a smaller TST
2. Every node to the **left** of a root has a value "lesser than" that of the root
3. Every node to the **right** of a root has a value "greater than" that of the root
4. **Every node to the middle of a root has a value equal to that of the root**

Below is an example TST to help with visualization.



Describe an algorithm that will print the elements in a TST in **descending** order.