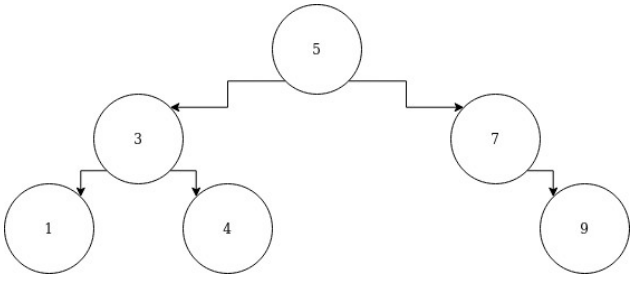
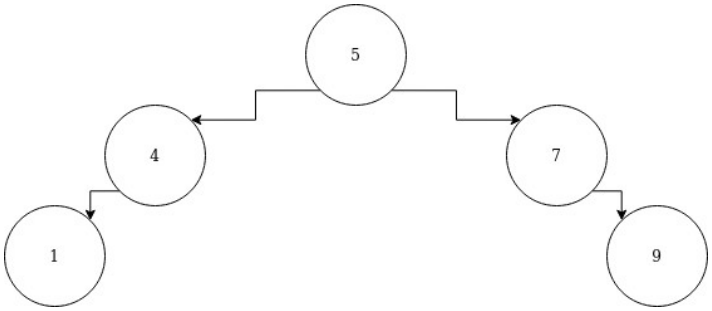
Daniel Vergilis
CS435-002
Project 1

GitHub: https://github.com/dv258/cs435-project1/

1. Binary Search Trees
   a. The properties of a binary search tree are:
      - Each node must have at most two children
      - The left subtree of every node must only contain values less than its parent
      - The right subtree of every node must only contain values greater than its parent
   b. Worst case time complexity for (n = # of inputs):
      - Insert: O(n)
        - If the input is inserted in order, the binary search tree acts like a linked list, where the worst-case insertion is O(n)
      - Delete: O(n)
        - If the input is inserted in order, the binary search tree acts like a linked list, where the worst-case deletion is O(n)
      - Find-next: O(log n)
        - The algorithm picks the right subtree and traverses leftward each level, unless it is null, where it will then compare the parent.
      - Find-prev: O(log n)
        - The algorithm picks the left subtree and traverses rightward each level, unless it is null, where it will then compare the parent.
      - Find-min: O(n)
        - If the input is inserted in reverse order, the binary search tree acts like a linked list, where finding the minimum is O(n)
      - Find-max: O(n)
        - If the input is inserted in reverse order, the binary search tree acts like a linked list, where finding the minimum is O(n)
   c. .
      i. Make a binary search tree with insertion, deletion, and can find the next, previous, min, and max nodes. Assume the inputs are integers, and the tree does not have any self-balancing features.
      ii. Some edge cases: inserting into an empty tree, inserting a duplicate value, deleting the root node, deleting a value not in the tree.
      iii. Insert 5, 3, 7, 1, 4, 9:



Delete 3:



iv. insertRec:
- compare value with current node, if less than, go down left child, if greater than, go down right child. If the child is null, then insert a new node there.

deleteRec:
- compare value with current node, if less than, go down left child, if greater than, go down right child. If value is equal, replace the node with the successor and delete the old node.

findNextRec:
- find min node on right subtree, if right child is null, check if parent is next node.

findPrevRec:
- find max node on left subtree, if left child is null, check if parent is next node.

FindMinRec:
- find leftmost leaf in subtree.

findMaxRec:
- find rightmost leaf in subtree.

v. I don't see any optimization issues with the code. All function have a space complexity of O(1), and the average time complexity for the functions are $O\left(log\ n\right)$.

vi. https://github.com/dv258/cs435-project1/bst-recursive.cpp
https://github.com/dv258/cs435-project1/bst-iterative.cpp

vii. The biggest issue I see with this code is that, because it is recursive, a very deep tree will require a lot of function calls and will take up more space on the call stack.

2. Sort It!
    a. 0005, 0006, 0007, 0010, 0011, 0012, 0016, 0017, 0018, 0019, 0020
    b. Given an unsorted list of elements, they can be sorted by using a Binary Search Tree. Inserting each element into the Binary Search Tree and doing an inorder traversal yields the inserted elements in sorted order.
    c. Located in https://github.com/dv258/cs435-project1/bst-recursive.cpp
3. Arrays of Integers
    a. Located in https://github.com/dv258/cs435-project1/int-arrays.cpp
    b. Located in https://github.com/dv258/cs435-project1/int-arrays.cpp