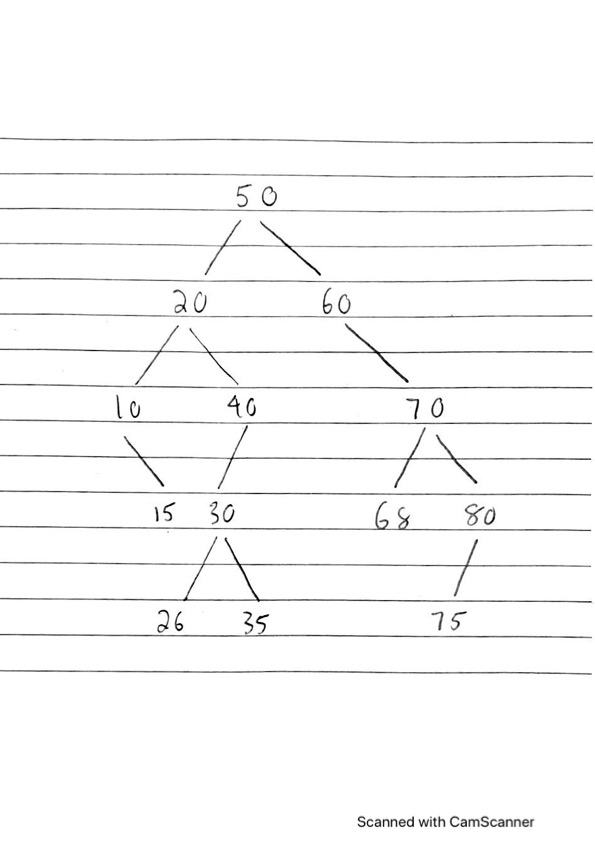
Eideen Mozaffari

UID : 105988436

Homework 5

**Problem 1a:**



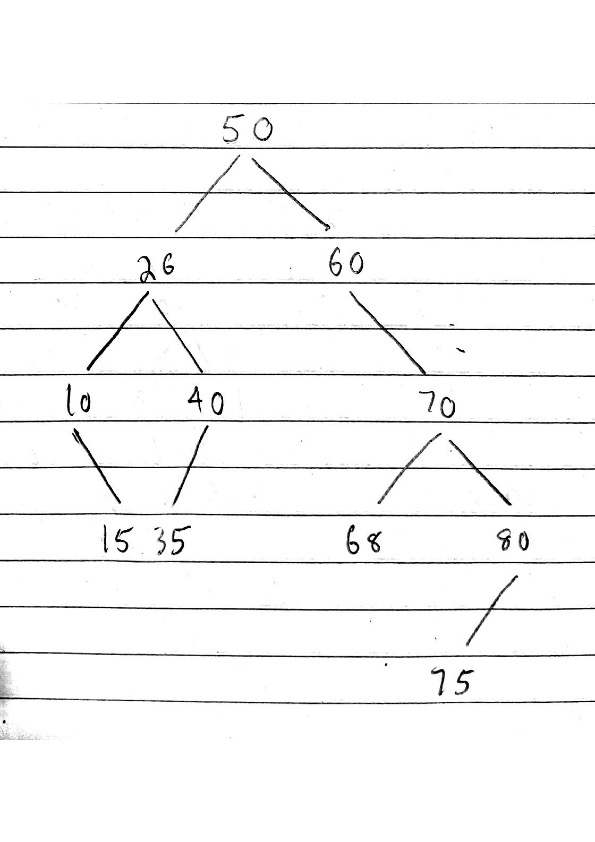
**Problem 1b:**

Preorder: 50, 20, 10, 15, 40, 30, 26, 35, 60, 70, 68, 80, 75

Inorder: 10, 15, 20, 26, 30, 35, 40, 50, 60, 68, 70, 75, 80

Postorder: 15, 10, 26, 35, 30, 40, 20, 68, 75, 80, 70, 60, 50

**Problem 1c:**



**Problem 2a:** Show a C++ structure/class definition for a binary tree node that has both child node pointers and a parent node pointer. Assume the data stored in each node is an int.

Solution:

struct Node

{

Node\* parent

Node\* left\_child

Node\* right\_child;

int value;

}

**Problem 2b:** Suppose this tree is being used to represent a set of integers. Write pseudocode to insert a new node into a binary search tree with parent pointers. Since this is for a set, attempting to insert a value already in the tree does nothing. It is your choice what parameters the function takes, whether it is recursive or not, whether it uses an auxiliary function you write, etc.

Solution:

// Auxiliary Function to create a new Node

Node\* newNode(val)

create a pointer and assign it a new Node

set Node's value to val

set Node's parent, left\_child, and right\_child to nullptr

return the pointer to new Node

void insertNode(Node\* root, int val) // Insertion function

if root is nullptr

root = newNode(val)

return

current Node pointer = root

while current is not nullptr

if val is equal to current's value

return

if val is less than current's value

if current's left\_child is nullptr

set current's left\_child = newNode(val)

set new Node's parent = current

return

else current's left\_child is not nullptr

set current = current's left\_child

else val is greater than current's value

if current's right\_child is nullptr

set current's right\_child = newNode(val)

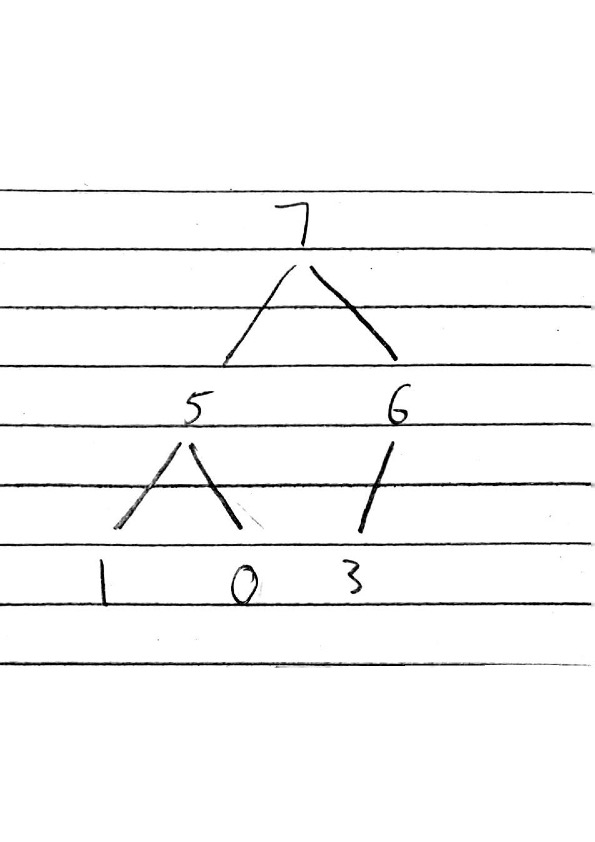
set new Node's parent = current

return

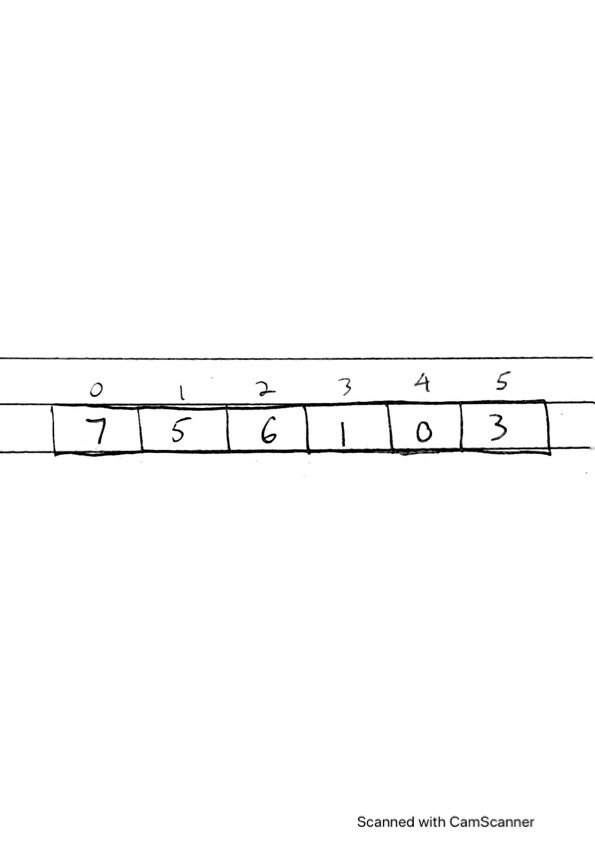
else current's right\_child is not nullptr

set current = current's right\_child

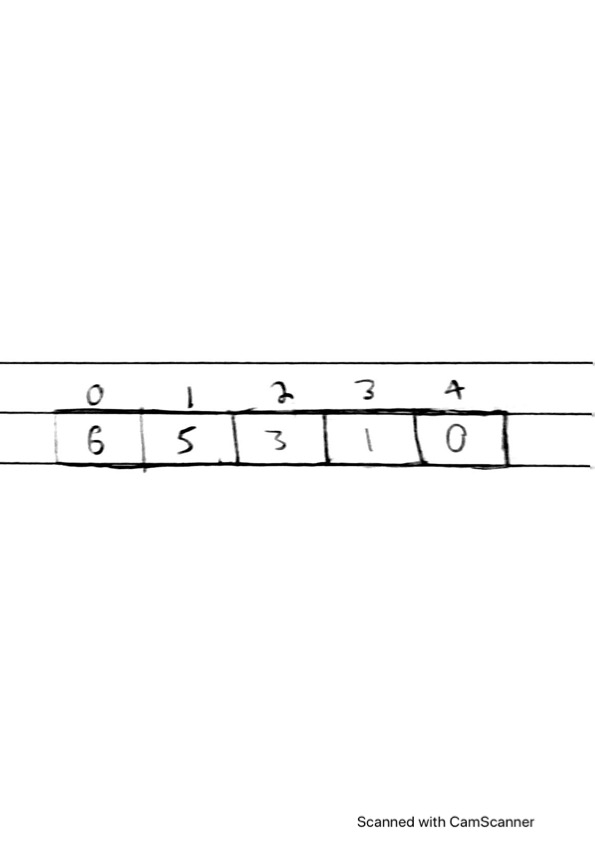
**Problem 3a:** Show the resulting heap (As in problem 1a, show the tree in some recognizable form.)



**Problem 3b:** Show how your heap from part a would be represented in an array.



**Problem 3c:** After executing h.remove(item); one more time, show the array that results.



Note: A pair<T1, T2> is a simple struct with two data members, one of type T1 and one of type T2. A set<K> and a map<K, V> are organized as approximately balanced binary search trees; an unordered\_set<K> and an unordered\_map<K, V> are organized as hash tables that never allow the load factor to exceed some constant, and a loop that visits every item in a hash table of N items is O(N). For the keys to be hashed, the hash function used produces uniformly distributed results.

**Problem 4:** Suppose UCLA has C courses each of which has on average S students enrolled. For this problem, courses are represented by strings (e.g. "CS 32"), and students by their int UIDs. We will consider a variety of data structures, and for each determine the big-O time complexity of the appropriate way to use that data structure to determine whether a particular student s is enrolled in course c. For example, if the data structure were vector<pair<string, vector<int>>>, where each pair in the outer vector represents a course and all the students in that course, with those students being sorted in order, then if the pairs are in no particular order in the outer vector, the answer would be O(C + log S). (The reason is that we'd have to do a linear search through the outer vector to find the course, which is O(C), and then after that we could do a binary search of the S students in the sorted vector for that course, which is O(log S).) In these problems, we're just looking for the answer; you don't need to write the reason.

1. vector<pair<string, list<int>>>, where each pair in the outer vector represents a course and all the students in that class, with those students being sorted in order. The pairs are in no particular order in the outer vector. What is the big-O complexity to determine whether a particular student s is enrolled in course c?
2. map<string, list<int>>, where the students in each list are in no particular order. What is the big-O complexity to determine whether a particular student s is enrolled in course c?
3. map<string, set<int>>. What is the big-O complexity to determine whether a particular student s is enrolled in course c?
4. unordered\_map<string, set<int>>. What is the big-O complexity to determine whether a particular student s is enrolled in course c?
5. unordered\_map<string, unordered\_set<int>>. What is the big-O complexity to determine whether a particular student s is enrolled in course c?
6. Suppose we have the data structure map<string, set<int>> and we wish for a particular course c to write the id numbers of all the students in that course in sorted order. What is the big-O complexity?
7. Suppose we have the data structure unordered\_map<string, unordered\_set<int>> and we wish for a particular course c to write the id numbers of all the students in that course in sorted order (perhaps using an additional container to help with that). What is the big-O complexity?
8. Suppose we have the data structure unordered\_map<string, set<int>> and we wish for a particular student s to write all the courses that student is enrolled in, in no particular order. What is the big-O complexity?

Solution:

1. O(C) + O(S) 🡺 O(C + S)
2. O(log C) + O(S) 🡺 O(log C + S)
3. O(log C) + O(log S) 🡺 O(log C + log S)
4. O(1) + O(log S) 🡺 O(log S)
5. O(1) + O(1) 🡺 O(1)
6. O(log C) + O(S) 🡺 O(log C + S)
7. O(1) + O(S log S) + O(S) 🡺 O(S log S + S) 🡺 O(S log S)
8. O(C) \* O(log S) 🡺 O(C log S)