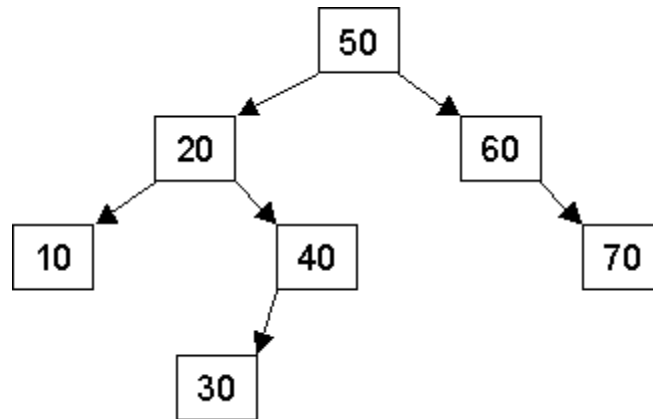


Homework 5

Time due: 11:00 PM Thursday, March 15

This homework is a good study guide for the final. The final will be open book, open notes. We don't expect you to memorize every last detail of every algorithm, so a skill you should develop is the ability to locate an algorithm in a reference source, trace through it, and understand it. If it's not in exactly the form your application requires, you should be able to adapt it.

1. Consider the following binary search tree, ordered using the $<$ relationship:



- a. Using the simplest binary search tree (BST) insertion algorithm (no balancing), show the tree that results after inserting into the above tree the nodes 80, 65, 74, 15, 36 and 25 in that order. (If you're not skilled with a drawing tool, use a simple text form of the tree. For example, the tree depicted above could be shown as

```

      50
     /  \
    20    60
   /  \  /  \
  10   40 70
   /
  30
  
```

Use enough space to distinguish left children from right children. Another way to represent the tree in text form (that distinguishes left children from right children) is

```

50
 20
  10
  40
   30
   xx
 60
  xx
  70
  
```

- b. After inserting the nodes mentioned in part a, what would be printed out by in-order, pre-order, and post-order traversals of the tree (assume your traversal function prints out the number at each node as it is visited)?
 - c. After inserting the nodes mentioned in part a, what is the resulting BST after you delete the node 30, then the node 20? (Again, just use a simple deletion algorithm with no balancing. If you have an option of making a choice, any correct choice is acceptable.)
2. In some binary search trees, each node has a left child pointer, a right child pointer and a parent pointer. The parent pointer of a node points to its parent (duh!), or is nullptr if the node is the root node. This problem will examine such trees.
 - a. 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.
 - b. Write pseudocode to insert a new node into a binary search tree with parent pointers. (Hint: You can find binary search tree insertion code on pp. 471-473).
3. Consider the following operations on an initially empty heap `h`; this heap is a maxheap, so the biggest item is at the top. The heap is represented as a binary tree:

```

h.insert(3);
h.insert(6);
h.insert(2);
h.insert(0);
h.insert(10);
h.insert(4);
int item;
h.remove(item); // Removes the biggest item from the heap, and puts it in item
h.insert(9);
h.insert(8);
h.remove(item);

```

- a. Show the resulting heap (As in problem 1a, show the tree in some recognizable form.)
 - b. Show how your heap from part a would be represented in an array.
 - c. After executing `h.remove(item);` one more time, show the array that results.
4. 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 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 $O(N)$.

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 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.

- a. `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*?
- b. `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*?
- c. `map<string, set<int>>`. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
- d. `unordered_map<string, set<int>>`. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
- e. `unordered_map<string, unordered_set<int>>`. What is the big-O complexity to determine whether a particular student *s* is enrolled in course *c*?
- f. 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?
- g. 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?
- h. 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?

Turn it in

By Wednesday, March 14, there will be a link on the class webpage that will enable you to turn in this homework. Turn in one zip file that contains your solutions to the homework problems. The zip file should contain one file:

- hw.doc, hw.docx, or hw.txt, a Word document or a text file with your solutions to the problems.