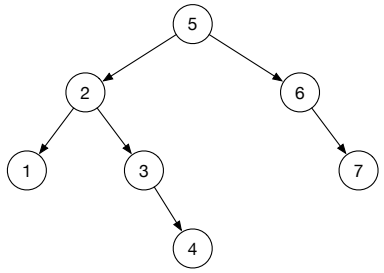
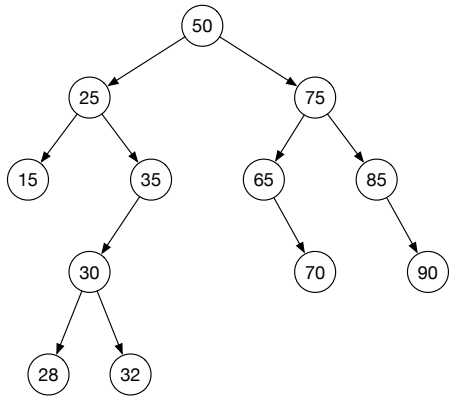
CS241
Homework 2: Binary trees, Hashing, O(n log n) sorting, priority queues/heaps

1. Assume you have the following binary search tree:



   (a) List the order of the nodes for a preorder traversal.
   (b) List the order of the nodes for an inorder traversal.
   (c) List the order of the nodes for a postorder traversal.

2. Assume you are given the following binary search tree:



   Show what the tree looks like after each of the following sequences of operations (draw the tree only after the last operation in each sequence).
   RESET THE TREE BACK TO THE DRAWING ABOVE AT THE BEGINNING OF EACH SEQUENCE.
   For deleting a node with two children, replace with the inorder *successor*.

   (a) Insert 11, Insert 64, Insert 31, Insert 80, Insert 38, Insert 37, Insert 39
   (b) Delete 35
   (c) Delete 25
   (d) Delete 50

3. Suppose a friend tells you they have a binary search tree with the following preorder traversal: 14, 7, 3, 10, 20, 16, 17, 22. Reconstruct the entire BST from this traversal and draw it. Hint: given a preorder traversal, where is the root?

4. Assume you have a node class that is used to implement a ***binary tree*** (note, not necessarily a binary search tree):

```
struct node {
  int data;
  node *left, *right;
};
```

   Write a function called `int getLevel(int lookfor, node * root)` that searches a binary tree for the integer called `lookfor`. If it's found, it will return the level of the tree at which the integer is found. If the

integer is not found, `getLevel()` returns `-1`. The root of the tree is at level 0, its children are at level 1, and so forth. Note that since this is a binary tree, not a binary search tree, you'll have to recurse on both children. You may not include any other arguments in this function.

5. Suppose we have a hash table with 11 locations that stores integers. Our hash function is h(i) = i % 11 and we use linear probing to resolve collisions. Show what the hash table looks like after doing the following insertions: 26, 42, 5, 44, 92, 59, 40, 36, 12, 60, 80.

6. Repeat exercise 5, but assume our hash table uses chaining instead. Show the table after all the numbers are inserted.

7. Run mergesort and show how it works on the array [13 57 39 85 99 70 22 48 64]. You must explicitly show the splitting and merging steps --- make it clear in what order these happen. Feel free to use a diagram like the one we did in class.

8. Run quicksort and show how it works on the array [30, 10, 40, 12, 50, 90, 20, 60, 57, 34, 55]. For each call to quicksort, explicitly show the partitioning step that happens (draw the array before and after the partitioning), then show what the two recursive calls produce. You can use a diagram like on the book's pages 758-759 if it's helpful.

9. Assume you have an empty max-heap (parent must be bigger than children). Insert the following numbers into the heap. Draw the heap (as a binary tree) after each insertion marked with a star: 1, 7*, 2, 6*, 3, 5*, 8*.

10. Assume you have the following array: [? 1 2 3 4 5 6 7], with array index 0 unoccupied as we normally do with heaps. So the array has 8 total positions, indexed 0-7, but we ignore position zero.

    (a) Draw the array as a binary tree (it won't be a heap).
    (b) Run the heapify algorithm (talked about in class, see page 752) on the binary tree to convert it to a max-heap. Show the binary tree (as a tree, not an array) after each percolate_down step is completed, including the last one, by which time the binary tree should be completed turned into a heap.