15-122: Principles of Imperative Computation

Recitation 21a

Nivedita Chopra, Josh Zimmerman

Heaps/Priority Queues

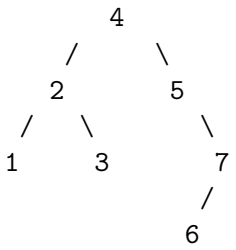
Insert elements from 1 to 15 into a heap in such a way to get the smallest possible number in the last level:

Solution: Since the heap shape invariant is so strict, we will always end up with the same shape, regardless of the order of insertion (even though the exact ordering may change). So, there will be 8 values in the last level.

Binary Search Trees

Construct a BST by inserting 4,2,3,1,5,7,6 in the given order

Solution:



Write code to print out the elements in ascending order
(Assume that you have a print_elem function)

Solution:

```
1     void print_tree(tree T) {
2       if (T == NULL)
3         return;
4       print_tree(T->left);
5       print_elem(T->data);
6       print_tree(T->right);
7     }
```

What was the cost of printing the entire tree in ascending order?

Solution: $O(n)$

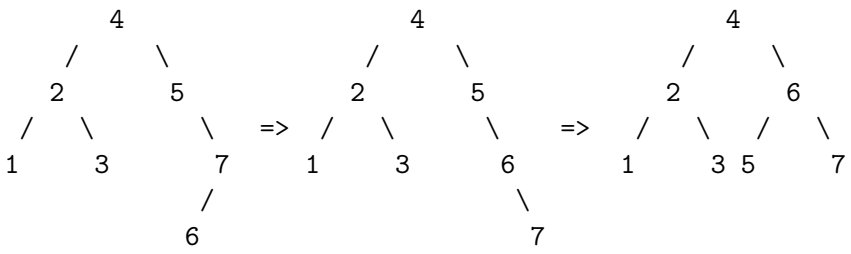
So we printed out the elements in sorted order in less than $O(n \log n)$ time!! Did we just come up with a more efficient sorting algorithm?

Solution: No. It took $O(n \log n)$ work to build this tree in the first place, which did the work of actually ordering the tree. What we did wasn't very different from printing out an array that was already sorted.

AVL Trees

Is the tree in the previous part balanced? Is it left heavy or right heavy? Perform operations to balance the tree

Solution: It is not balanced. It requires a double rotation to fix: first around 7, then around 5.



Let's C

What does the following function do?
(Hint: In C, the assignment statement (=) returns the assigned value)

```
void mystery(char* p, char* q){ while (*p++ = *q++); }
```

Solution: It copies the contents of `q` into `p`. If it isn't clear why this is the case, don't spend too much time worrying about it. You *can* do a lot of things you should never do in C.