CS 202 Fundamental Structures of Computer Science II

Assignment 2 – Heaps and AVL Trees

Date Assigned: November 6, 2015
Due Date: November 21 23:55 (sharp), 2015

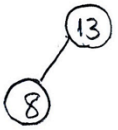
Name: Gülsüm
Surname: Güdükbay
ID: 21401148
Section: 1

1) Question Part

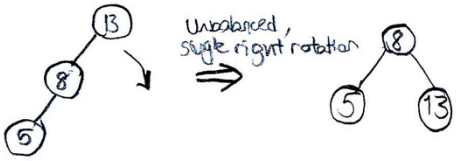
a. Result of inserting 13, 8, 5, 9, 4, 6, 12, 2, 1 and 3:

Inserting 13:

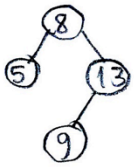
Inserting 8:



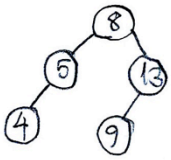
Inserting 5:



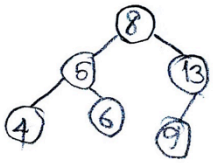
Inserting 9:



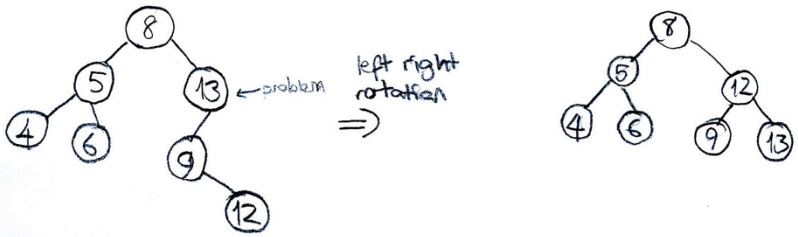
Inserting 4:



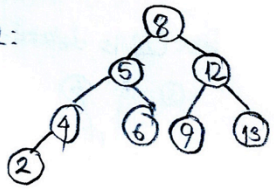
Inserting 6:

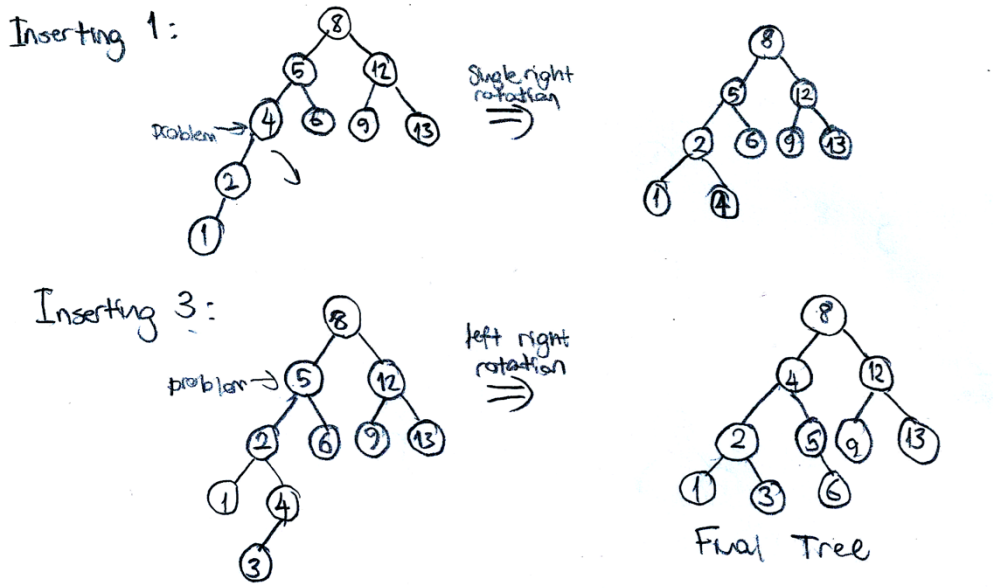


Inserting 12:

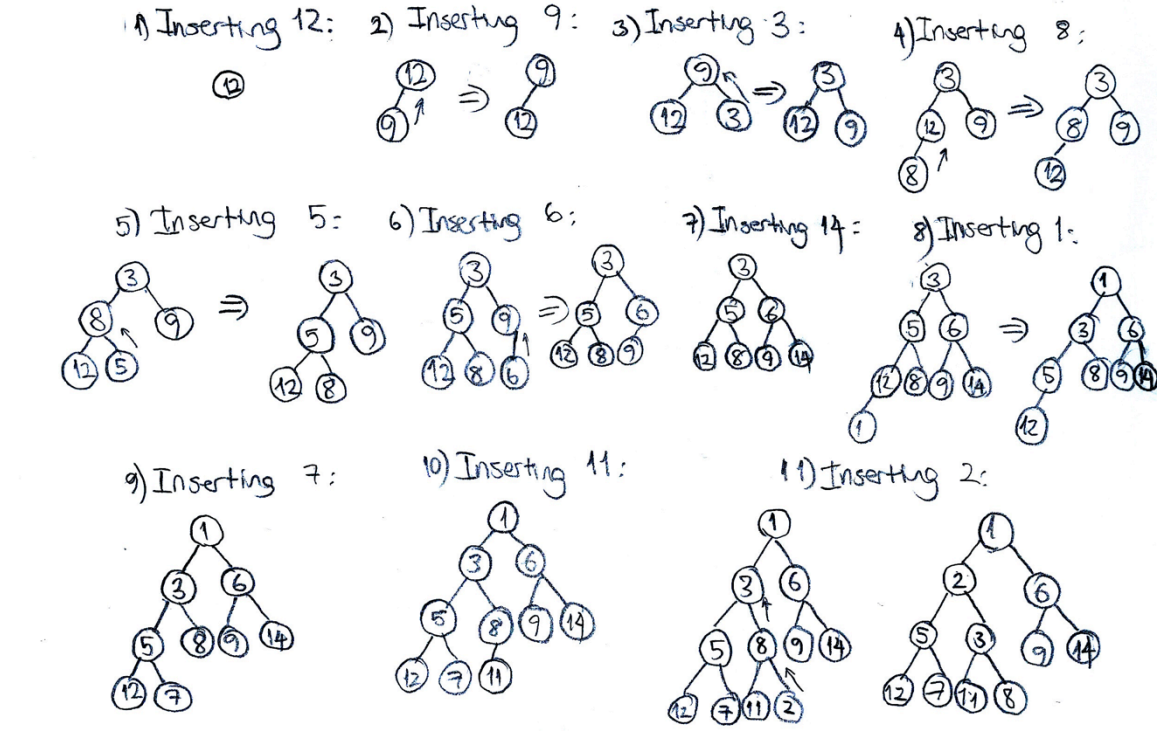


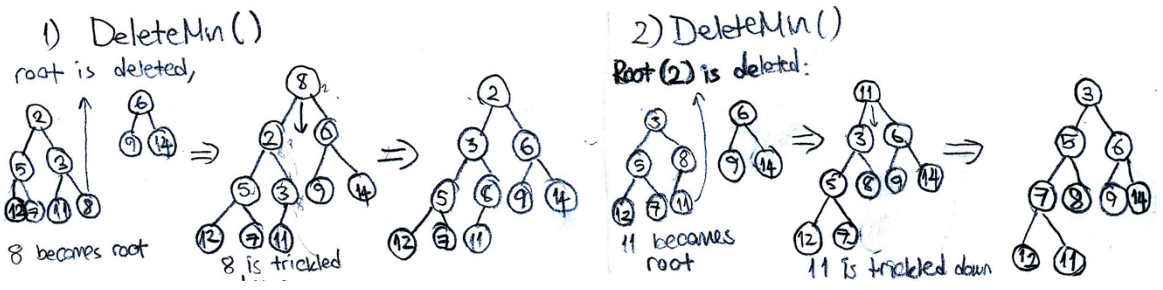
Inserting 2:



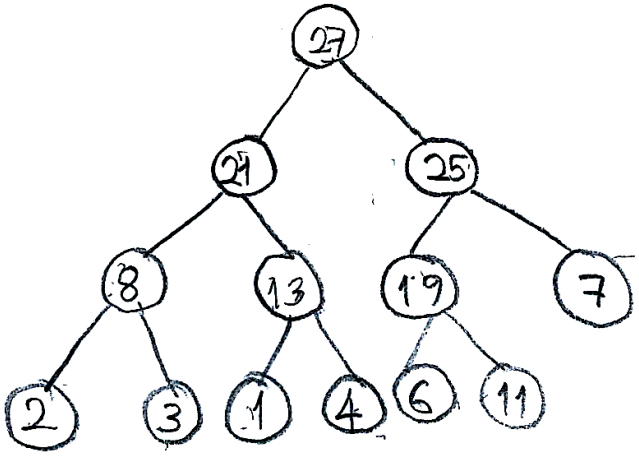


b. Basic operations on binary min heaps:





c. Binary heap traversals:



Binary max heap used in the traversals below, the sorted array (with heapsort algorithm) should be in decreasing order.

In the preorder traversal, the output is not sorted since traversing to the left and then the right increases the decreased value of elements as seen in the output below:

Preorder Traversal: 27, 21, 8, 2, 3, 13, 1, 4, 25, 19, 6, 11, 7

Since the value of the left child of any root is less than the value of that root because of the max heap property, the inorder traversal does not give a sorted list in a descending order as seen in the output below:

Inorder Traversal: 2, 8, 3, 21, 1, 13, 4, 27, 6, 19, 11, 25, 7

In the postorder traversal, the output is not sorted, since traversing to the left and then right and then root, and then again left and right decreases the increasing values of elements and does not give an output that should be in ascending order as seen in the output below:

Postorder Traversal: 2, 3, 8, 1, 4, 13, 21, 6, 11, 19, 7, 25, 27

d. Minimum number of nodes in an AVL tree of height h:

i. N(h) the minimum number of nodes in an AVL tree of height h. The left and right subtrees of this tree should also have minimum number of nodes. Since an AVL tree is a balanced tree type, when one of the subtrees is having a height of h-1, the other should have h-2, because the children heights can only differ by 1.

Then, we can construct the recurrence relation below (root itself is also included in the equation since it should be counted):

$$N(h) = N(h - 1) + N(h - 2) + 1.$$

By intuition, N(0) = 1 (the number of elements in a tree of height 0 is at least one) and N(1) = 2 (the number of elements in a tree of height 1 should be at least 2).

If we list the first 5 elements of the recurrence relation above, we get 1, 2, 4, 7, 12. We can clearly see that the number of elements for height h of a AVL tree can be found by adding up the previous 3 numbers minus 1 for the root itself, in the recursive series. Since this relation is similar to Fibonacci sequence, by intuition we can see that the expression is found to be N(h) = F(h + 3) – 1, where F(h) is Fibonacci sequence of h elements.

Proof by induction:
Showing that the expression holds for the base cases where h = 0:

$$N(0) = 1$$
$$F(3) = 2$$

So, N(0) = F(3) – 1.
For h = 1:

$$N(1) = F(4) – 1.$$
$$N(2) = F(5) – 1.$$

Going on by incrementing h's value, N(k) is assumed to be true for the equation for h >= 1 to h = k:

$$N(k) = F(k+3) – 1$$

Now, it is being proven that the relation holds for h = k + 1:

$$N(k + 1) = N(k) + N(k - 1) + 1$$

Substituting Fibonacci sequences instead of N(h) relations for each term we obtain:

$$N(k+1) = F(k+3) – 1 + F(k+3-1) – 1 + 1$$
$$N(k+1) = F(k+3) – 1 + F(k+2)$$

Since F(n) = F(n-1) + F(n-2),

$$F(k+3) + F(k+2) = F(k+4)$$

Therefore,

$$N(k+1) = F(k+4) – 1.$$

Replacing k with k – 1 to obtain N(k), we obtain N(k) = F(k+3) – 1, which shows that the recurrence relation holds for all h by the principles of mathematical induction.

ii. The minimum number of nodes in an AVL tree of height 15 is:

$$N(15) = F(18) – 1 \text{ (According to the recurrence relation found in part i.)}$$
$$N(15) = 2583$$

e. Function in pseudocode that determines whether a given binary tree is a min heap:

```
//returns true if a tree is min heap, otherwise false
bool isMinHeap (root: HeapNode pointer) {

//Base case 1: if the root is null, an empty tree is assumed to be a min heap
    if (root is null) {
        return true;
    }

//Base case 2: if there is a left child or right child and the left/right child has a smaller value
//than the root returns false
    if (root has a left child AND root's left child has a smaller value than the root's value OR
       root has a right child AND root's right child has a smaller value than the root's value) {
        return false;
    }

//Recursive step, calls the method itself for the left and right subtrees and if the method comes
//till the leaf nodes they are both true, so the function returns true.
    return isMinHeap (root's left child) AND isMinHeap (root's right child);
}
```

2) Programming Part Report

A heap is a binary tree that is complete. Heaps have two types: max heap or min heap. Complete means that the tree is either an empty tree or depending on its type, the key of its children has less (if it is a max heap) or larger (if it is a min heap) value than the root, and recursively, both of the children are also heaps of same type.

The popMaximum() function (a.k.a. heapDelete()) deletes the root of the tree and replaces it with the last element in the heap.
The insert(a) function inserts an element to the last empty position in the heap and then trickles it up to its position.
The maximum() function returns the maximum element in a max heap, in which it is the first element in the heap array if it is an array based implementation.

The difference between a Binary Search Tree and a heap is that when you do an inorder traversal with a BST, the output is sorted, however this doesn't apply for heaps. For heaps, sort is done by swapping the root with the last element in the heap (deleting the root), rebuilding the two semi heaps into a heap, and repeating these steps until the element count in the heap is larger or equal to 1.

For the data1.txt input file, the number of elements was 1000 and the number of comparisons made was 15012.
For the data2.txt input file, the number of elements was 2000 and the number of comparisons made was 33994.
For the data3.txt input file, the number of elements was 3000 and the number of comparisons made was 54670.
For the data4.txt input file, the number of elements was 4000 and the number of comparisons made was 75924.
For the data5.txt input file, the number of elements was 5000 and the number of comparisons made was 98300.

Sample output to console for the data points above:

```
gulsum.gudukbay@dijkstra:~/hw3> ./heapsort data1.txt output.txt

Number of elements in the file is 1000
Number of comparisons done 0

Number of elements in the file is 1000
Number of comparisons done 15012
gulsum.gudukbay@dijkstra:~/hw3> ./heapsort data2.txt output.txt

Number of elements in the file is 2000
Number of comparisons done 0

Number of elements in the file is 2000
Number of comparisons done 33994
gulsum.gudukbay@dijkstra:~/hw3> ./heapsort data3.txt output.txt

Number of elements in the file is 3000
Number of comparisons done 0

Number of elements in the file is 3000
Number of comparisons done 54670
gulsum.gudukbay@dijkstra:~/hw3> ./heapsort data4.txt output.txt

Number of elements in the file is 4000
Number of comparisons done 0

Number of elements in the file is 4000
Number of comparisons done 75924
gulsum.gudukbay@dijkstra:~/hw3> ./heapsort data5.txt output.txt

Number of elements in the file is 5000
Number of comparisons done 0

Number of elements in the file is 5000
Number of comparisons done 98300
```