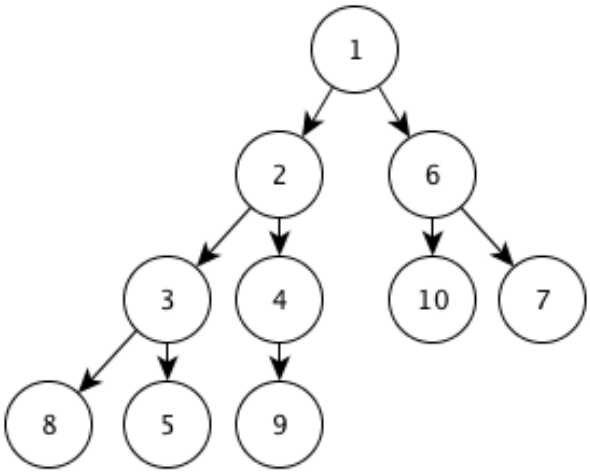
CECS 277 – Lecture 11 – Heaps

Heaps are great for when you want to consistently keep track of the minimum (min-heap) or maximum (max-heap) of a list of elements, or for quickly (O(n log n)) sorting a list of elements. Heaps are stored in an array or list, but they are still arranged as a binary tree. The tree is kept balanced and almost completely filled. In a min-heap the minimum value is kept at the root of the tree.

Nodes – A heap is made up of nodes. The first node in the tree is called the root. Each node keeps track of its own data.

Inserting Nodes – Each node is initially placed at the end of the tree. If the new node's value is less than the parent's, then the nodes are swapped. This process repeats until the child is greater than the parent.

Example: Given the following values: 4, 8, 6, 1, 3, 10, 7, 2, 5, 9.



Accessing Elements – Since the binary tree is stored in a list, accessing the location of a particular node's parent or child is simple:

1. A Node's Parent's Location – `(i - 1) / 2`
2. A Node's Left Child's Location – `(2 * i) + 1`
3. A Node's Right Child's Location – `(2 * i) + 2`

Removing a Node – There are three steps for removing a node:

1. Remove the root node and replace it with the last value in the tree.
2. If the node has children, find the smallest child and swap it with the parent as long as the parent is greater than the child.
3. Repeat this step until the parent is no longer larger than either of its children.

The Heapsort Algorithm – sorts the elements.

1. Place all elements into the heap.
2. Remove all of the elements one at a time. The elements are removed in sorted order.

Example:

```
public class Node {
	private int data;
	public Node(int d){
		data = d;
	}
	public int getData(){
		return data;
	}
	public void setData(int d){
		data = d;
	}
}


import java.util.ArrayList;
public class Heap {
	private ArrayList<Node> heap;
	public Heap(){
		heap = new ArrayList<Node>();
	}
	public int getSize(){
		return heap.size();
	}
	public boolean isEmpty(){
		return heap.isEmpty();
	}
	public int getPLoc(int i){
		return (i-1)/2;
	}
	public int getLCLoc(int i){
		return 2*i+1;
	}
	public int getRCLoc(int i){
		return 2*i+2;
	}
	public Node getNodeAt(int i){
		if(heap.get(i)==null){
			System.out.println("Item does not exist.");
			return null;
		}else{
			return heap.get(i);
		}
	}
```

```
public void addNode(Node n){
      heap.add(null);
      int index = heap.size()-1;
      while(index > 0 && getNodeAt(getPLoc(index))
                                    .getData()>n.getData()){
            heap.set(index, getNodeAt(getPLoc(index)));
            index = getPLoc(index);
      }
      heap.set(index, n);
}

public Node removeMin(){
      Node min = heap.get(0);
      int index = heap.size()-1;
      Node last = heap.remove(index);
      if(index > 0){
            heap.set(0, last);
            Node root = heap.get(0);
            int end = heap.size()-1;
            index = 0;
            boolean done = false;
            while(!done){
                  if(getLCLoc(index)<=end){//left exists
                        Node child = getNodeAt(getLCLoc(
                                                index));
                        int childLoc = getLCLoc(index);
                        if(getRCLoc(index)<=end){//rt exists
                              if(getNodeAt(getRCLoc(index))
                              .getData()<child.getData()){
                                    child = getNodeAt(getRCLoc(
                                                index));
                                    childLoc = getRCLoc(index);
                              }
                        }
                        if(child.getData()<root.getData()){
                              heap.set(index, child);
                              index = childLoc;
                        }else{
                              done = true;
                        }
                  }else{//no children
                        done = true;
                  }
            }
```

```
                heap.set(index, root);
            }
            return min;
        }

        public void printHeap(){
            for(int i=0;i<heap.size();i++){
                System.out.print(heap.get(i).getData()+" ");
            }
            System.out.println();
        }
}



public class TestHeap {
        public static void main(String[] args) {
            Heap h = new Heap();
            for(int i= 1;i<=10;i++){
                int rand =(int)(Math.random()*100)+1;
                Node n = new Node(rand);
                System.out.print(rand+" ");
                h.addNode(n);
            }
            System.out.println();
            h.printHeap();
            for(int i=0;i<10;i++){
                h.removeMin();
                h.printHeap();
            }
        }
}
/* Output
84 96 38 63 57 49 85 36 65 10
10 36 49 57 38 84 85 96 65 63
36 38 49 57 63 84 85 96 65
38 57 49 65 63 84 85 96
49 57 84 65 63 96 85
57 63 84 65 85 96
63 65 84 96 85
65 85 84 96
84 85 96
85 96
96 */
```