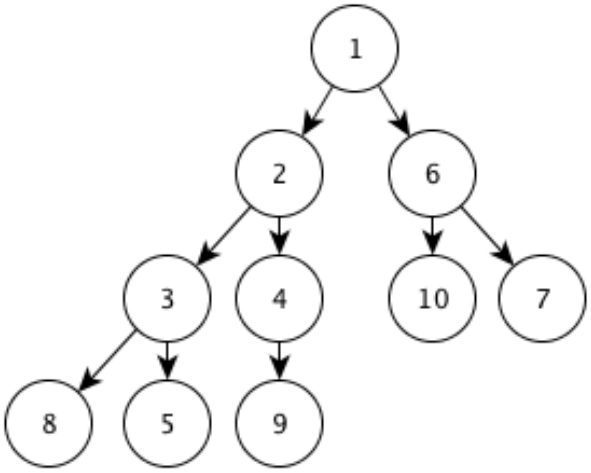
CECS 277 – Lecture 11 – Heaps

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.

Inserting Elements – Each new element is initially placed at the end of the tree. If the new element's value is less than the parent's, then the elements 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 element's parent or child is simple:

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

Removing an Element – There are three steps for removing an element:

1. Remove the root element and replace it with the last value in the tree.
2. If the element 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.

```
/** Integer Heap – only works for integers */
import java.util.ArrayList;
public class Heap {
	private ArrayList<Integer> heap;
	public Heap() {
		heap = new ArrayList<Integer>();
	}
	public int size() {
		return heap.size();
	}
	public boolean isEmpty() {
		return heap.isEmpty();
	}
	public int getCurrent() {
		return heap.get( 0 );
	}
	private int getPLoc( int i ) {
		return ( i - 1 ) / 2;
	}
	private int getLCLoc( int i ) {
		return 2 * i + 1;
	}
	private int getRCLoc( int i ) {
		return 2 * i + 2;
	}
	public void addItem( int i ) {
		heap.add( 0 );
		int index = heap.size() - 1;
		while( index > 0 && heap.get( getPLoc(index)) > i ) {
			heap.set( index, heap.get( getPLoc( index ) ) );
			index = getPLoc( index );
		}
		heap.set( index, i );
	}
	@Override
	public String toString( ) {
		//returns string in tree order not sorted order
		String s = "";
		for( int i = 0; i < heap.size( ); i++ ) {
			s += heap.get( i ) + " ";
		}
		return s;
	}
```

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

```
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;
                  System.out.print( rand + " " );
                  h.addItem( rand );
            }
            System.out.println( );
            System.out.println( );
            System.out.println( h );
            for( int i = 0; i < 10; i++ ) {
                  h.removeItem( );
                  System.out.println( h );
            }
      }
}

/* 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 */
```