1) O(2^n)
2) n/2
3) FALSE, the largest element may have a left child and still be the largest
4) OHHIMARK
5) FALSE. It only works if you reheap down when what is below it is also a heap. Consider this tree: [1 10 9 15]

```
    1
 10   9
15
```

After you reheap each element, you get [10 15 9 1]:

```
   10
 15 9
1
```

Which is not a heap.

6) N-1
7) O(n log n)
8) 9
9) 2
10) O(n^2)

Section 2 (Heap):

```
            // Easiest way: A full heapify would work just fine to fix
      anything
            for(i=nElems/2-1; i>=0; i--) {
                  bubbleDown(i);
            }

            // Harder way: Actually figure out what is broken and bubble them
            down, not necessary because bubbleDown does that for you
            for(i=nElems/2-1; i>=0; i--) {
                  if ((heap[i] < heap [i*2+1) ||
                        ((i*2+2 < nElems) && (heap[i] < heap[i*2+2])){

                        bubbleDown(i);
                  }
            }

            // This looks at all of the elements, including the leaves, not
            as good
```

```
            for(i=nElems-1; i>=0; i--) {
                  if (((i*2+1 < nElems) && (heap[i] < heap [i*2+1)) ||
                        ((i*2+2 < nElems) && (heap[i] < heap[i*2+2])){
                        bubbleDown(i);
                  }
            }


            // You can't go from the front and bubble down, bubble down
            depends on the node's children being the root of a valid heap
```

Section 3:

```
private void constructList(BSTNode<T> node){
            if (node == null) return;
            constructList(node.left);
            Node<T> newNode = new LLNode(node.data);
            if (head == null){
                  head = newNode;
            } else {
                  tail.next = newNode;
            }
            tail = newNode;
            constructList(node.right);

}
```

Section 4:

```
      public void printNodes (T start) {
            clearVisits();
            if (start == null) {return};
            visitVertex(start);
            Queue<T> q = new Queue<T> ();
            q.enqueue(start);
            System.out.println(start);
            T b = null;
            while(!queue.isEmpty()) {
                  T v = queue.dequeue();
                  while((b=getNextUnvisitedNeighbor(v)) != null) {
                        visitVertex(b);
                        System.out.println(b);
                        queue.enqueue(b);
```

```
}}}
```