# CSC130 Mid-Term Practice Exam

1. (10 Points) Given $f(n) = 3^n$, $g(n)= 1 + 2 + 4 + 8 + . . . + 2^n$, which of the following is(are) correct:

1) f(n) = O(g(n)) (big Oh)
2) f(n) = Ω(g(n))
3) f(n) = Ɵ(g(n))
4) f(n) = o(g(n)) (small Oh)
5) f(n) = Ɯ(g(n))

2. (10 Points) Answer the following questions regarding memory of a process.

1) In which section is compiled program code stored?

2) In which section are function local variables stored?

3) In which section are static variables stored?

4) In which section does Java garbage collection work?

3. (10 Points) Given the input array has n elements, only contains integers from 0 to m, and is in reverse sorted order, provide the running time analysis in big-Oh notation for the following sorting algorithms.

| Sorting Algorithm | Time Complexity |
|---|---|
| Merge Sort | |
| Quick Sort | |
| Heap Sort | |
| Count Sort | |

4. (10 Points) Mark the following statements True or False.

1) An input array in ascending order represents a min binary heap.
2) Randomizing input can improve merge sort algorithm performance.
3) Radix sort sorts numbers from the most significant digit to the least significant digit.
4) For comparison-based sorting algorithms, the lower bound for average-case running time is Ω(nlogn), where n is the number of elements to be sorted.

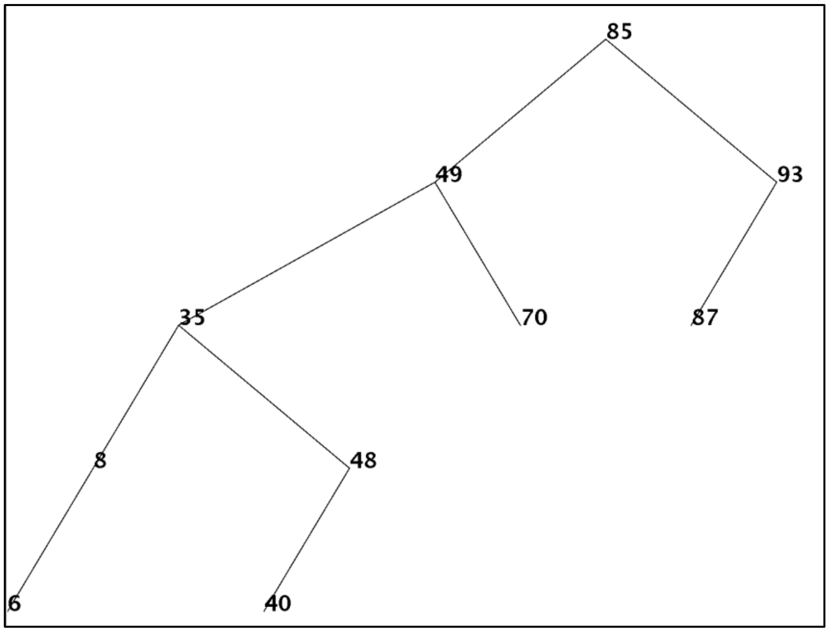
5. (10 Points) Which of the following is (are) balanced tree(s)?

1) Binary max heap
2) Binary search Tree
3) Binary tree
4) Complete binary tree

6. (10 Points) Given the following binary tree, list keys in the following order:

a) In-order traversal order

b) Post-order traversal order



7. (10 Points) Apply the partition algorithm for the following array, assume lo is 0 and hi is 16. Provide the partition results.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| original | 85 | 53 | 38 | 44 | 32 | 97 | 99 | 65 | 46 | 66 | 57 | 68 | 13 | 63 | 98 | 42 | 50 |
| Partition Result | | | | | | | | | | | | | | | | | |

8. (10 Points) Sort the following array into **non-decreasing** order using heap sort with heap root at **index 0**. Provide the heap construction result and the first swap & sink result. Use sink method for heap construction.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| original | 85 | 53 | 38 | 44 | 32 | 97 | 99 | 65 | 46 | 66 | 57 | 68 | 13 | 63 | 98 | 42 | 50 |
| Heap Construction | | | | | | | | | | | | | | | | | |
| swap & sink | | | | | | | | | | | | | | | | | |

9. (10 Points) Provide worst-case running time and space analysis in big O notation 1) for the following function, 2) for the following function with BST replaced by red-black BST. Briefly explain your answer.

```
// assume keys and vals have the same length and are not null
public static void client(Integer[] keys, Object[] vals) {
        BST<Integer, Object> bst = new BST<Integer, Object>();
        int n = keys.length;
        for (int i=0; i<n; i++) {
                bst.put(keys[i], vals[i]);
        }
}
```

10. (10 Points) Implement **getNumLeaves(Node node)** method inside the given BST class. 1) Describe your algorithm. 2) Implement your algorithm in Java.

```
public class BST<Key extends Comparable<Key>, Value> extends AbstractOrderedSymbolTable<Key, Value> {
        public class Node {
                private Key key; // key
                private Value val; // associated value
                private Node left, right; // links to subtrees
                private int n; // # nodes in subtree rooted here

                public Node(Key key, Value val, int n) {
                        this.key = key;
                        this.val = val;
                        this.n = n;
                }
        }
        private Node root; // root of BST
        /**
         * @return the number of leaves in the BST
         */
        public int getNumLeaves() {
                return getNumLeaves(root);
        }
        /**
         * @param node the root of the sub-tree
         * @return the number of leaves in the sub-tree
         */
        private int getNumLeaves(Node node) {
                // provide your implementation here
        }
}
```