# ICS 211 Fall 2014 Final Exam, Dec 19th, 2014

Clearly write your name on both the **back** and **front** of this exam.

This exam is closed-book, closed note, closed computer, closed neighbor. No calculators are allowed. There are a total of 100 points.

Be sure to answer all parts of each question.

**Question 1** (10 points): Implement this method to do a selection sort.

**public static <T extends Comparable<T>> void selectionSort(T [] table) {**

**int n = table.length;**

**for (int fill = 0; fill < n – 1; fill++) {**

**int minPos = fill;**

**for (int next = fill + 1; next < n; next++) {**

**if (table[next].compareTo(table[posMin]) < 0) {**

**posMin = next;**

**}**

**}**

**T temp = table[fill];**

**table[fill] = table[posMin];**

**table[posMin] = temp;**

**}**

**}**

**Question 2** (10 points): Implement this method to add a value to a max heap (the value is **Comparable**). Assume there is room in the array to add the element.

**public class Heap<E implements Comparable<E>> {**

**protected E[] data;**

**int size; // invariant: size <= data.length**

**…**

**// precondition: size < data.length**

**public void add(E value) {**

int child = size++;

data[child] = value;

int parent = (child – 1) / 2;

// reheap

while (child != 0 && data[parent].compareTo(data[child]) < 0) {

E temp = data[parent];

data[parent] = data[child];

data[child] = temp;

child = parent;

parent = (child – 1) / 2;

}

}

**Question 3** (10 points): Implement this method to merge two sorted arrays into a result array. The result array is large enough (so don’t check array sizes).

**public static <T extends Comparable<T>> void merge(T[] left,**

**T[] right,**

**T[] result) {**

**int i = 0;**

**int j = 0;**

**int k = 0;**

**while (i < left.length && j < right.length) {**

**if (left[i].compareTo(right[j]) < 0) {**

**result[k++] = left[i++];**

**else {**

**result[k++] = right[j++];**

**}**

**}**

**while (i < left.length) {**

**result[k++] = left[i++];**

**}**

**while (j < right.length) {**

**result[k++] = right[j++];**

**}**

**}**

**Question 4** (10 points): Discuss the advantages and disadvantages of chained hashing vs. open addressing.

|  |  |
| --- | --- |
| Open Addressing | Chaining |
| + Single Array | + No collisions |
| + Easy to implement | + Store more items than table size |
| - Collisions | - More complex data structure |
| - Deletion is complicated | - Requires linked list/Additional class |
| - Can have probing performance hit | - Must traverse linked list |

**Question 5** (10 points): Explain how stacks are used by Java to implement recursion.

Java pushes each method call onto the Runtime Activation stack. The activation record contains the parameters and local variables, so when the recursive method returns Java keeps track of the previous state of the method from the stack. This allows Java to combine the results and eventually return the correct value.

**Question 6** (5 points): How does using a Huffman Tree and encoding achieve compression? Explain briefly.

Items with higher frequency are higher in the Huffman tree, thus producing a shorter Huffman code for high frequency items.

**Question 7** (5 points): Explain the differences between: **a Binary Tree, a Binary Search Tree,** and **a Binary Search**.

Binary Tree is a root node, possibly null, with a binary tree as a left child, and a binary tree as a right child.

Binary Search Tree is a binary tree whose left child is a binary search tree and whose right child is a binary search tree. All the values in the left child are less than the root value, all the values in the right child are greater than the root value.

Binary Search is a method of searching for a value in a sorted array. Start from the middle then recursively search the left side or right side depending on the comparison of the middle value.

**Question 8** (10 points): Explain the difference between: **x == y** and **x.equals(y)**. Be very clear.

== compares the object references, not values in the objects.

.equals() is a method implemented by the class that compares member variables or values of the class instance.

**Question 9** (10 points): Give the worst-case, average case, and best-case runtimes for these sorting algorithms.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Worst | Average | Best |
| Selection  sort | N2 | N2 | N2 |
| Bubble  sort | N2 | N2 | N |
| Insertion  sort | N2 | N2 | N |
| Heap  sort | N log N | N log N | N log N |
| Merge  sort | N log N | N log N | N log N |

**Question 10** (10 points): Implement this method **poll()** to remove the first element from this queue, implemented using a circular array.

**public class CircularArrayQueue<E> {**

**protected E[] data;**

**protected int size;**

**protected int first; // index of first element, if size > 0**

**protected int last; // index of last element, if size > 0**

**…**

**public E poll() {**

**if (size == 0) {**

**return null;**

**}**

**E result = data[first];**

**first = (first + 1) % data.length;**

**size++;**

**return result;**

**}**

**Question 11** (10 points): Write this recursive method to find the depth of the tree. **Do not use a helper method.**

**private static int depth(BinaryNode<T> node) {**

**if (node == null) {**

**return 0;**

**}**

**int left = depth(node.left);**

**int right = depth(node.right);**

**if (left > right) {**

**return 1 + left;**

**}**

**else {**

**return 1 + right;**

**}**

**}**