CSE 373, Winter 2011 Midterm

1. **Big-Oh** (18 Points)

Calculate **the exact value** of the variable `sum` after the following code fragment, in terms of variable `n`. Use summation notation to compute a closed-form solution. Then use this value to give a tightly bounded Big-Oh analysis of the runtime of the code fragment.

```
int sum = 0;
for (int i = 1; i <= n + 3; i++) {
    for (int j = 1; j <= n * n; j++) {
        sum += 2;
    }

    sum--;
}
return sum;
```

2. **Sorting** (12 Points)

For the following questions, indicate which sorting algorithm will successfully sort the elements in the least amount of time.

- Choose between **bubble sort**, **selection sort**, **insertion sort**, **merge sort**, **quick sort**, and **heap sort**.
- The algorithm chosen should be the one that completes fastest, without crashing.
- Assume that bubble sort is *not* optimized in any way.
- Assume that the quick sort algorithm chooses the first element as its pivot at each pass.
- Assume stack overflow occurs on 5000+ stacked method calls.

a. array size 50000, random order

Sorting Algorithm: ____

b. array size 1000000, descending order, no extra memory may be allocated

Sorting Algorithm: ____

c. array size 350000, ascending order

Sorting Algorithm: ____

3. **Trees** (25 Points)

a. Given the following list of integers:

   20, 30, 90, 50, 40, 35, 10, 37, 31, 34

   Draw the **binary search tree (BST)** that results when all of the above elements are added (in the given order) to an initially empty BST.

b. What is the balance factor of the root node of the **BST** that you drew for part 3(a).

c. Given the same list of integers from 3(a):

   20, 30, 90, 50, 40, 35, 10, 37, 31, 34

   Draw the **AVL tree** that results when all of the above elements are added (in the given order) to an initially empty tree.

   Please show your work. You do not have to draw an entirely new tree after each element is added or removed, but since the final answer depends on every add/remove being done correctly, you may wish to show the tree at various important stages to help earn partial credit in case of an error. The next page is blank to give you space to write.

d. What is the balance factor of the root node of the **AVL tree** that you drew for part 3(c).

4. **Heaps** (20 Points)

Given the following integer elements:

30, 65, 22, 40, 15, 70, 80, 60, 55, 10

a. Draw the tree representation of the heap that results when all of the above elements are added (in the given order) to an initially empty **maximum binary heap**. ***Circle the final tree that results from performing the additions.***

b. After adding all the elements, perform **2 removes** on the heap. ***Circle the tree that results after the two elements are removed***.

Please show your work. You do not need to show the array representation of the heap. You do not have to draw an entirely new tree after each element is added or removed, but since the final answer depends on every add/remove being done correctly, you may wish to show the tree at various important stages to help earn partial credit in case of an error.

5. Priority Queue Implementation

Part A: Implementation

In lecture, we implemented a minimum priority queue using a heap data structure. This problem implements a *minimum priority queue* using a *queue*. The queue is already implemented with a linked list in the `LinkedListQueue` class.

Elements are added to the priority queue by being put at the end of the queue. The `add` method of the queue-based priority queue is implemented below.

Finish the implementation of the class `IntQueuePQ`, a priority queue implemented using a queue, by writing the **`remove`** method. The `remove` method finds, removes, and returns the minimum element from the priority queue. If the priority queue contains no elements and the remove method is called an `IllegalStateException` is thrown.

In order to write `remove` you may call any of the methods declared by the `IntQueue` interface on the `queue` instance variable found in `IntQueuePQ`. However, you may not use any additional data structures or abstract data types other than `queue` declared in the `IntQueuePQ` class in order to solve this problem.

```
public interface IntPriorityQueue {
    public void add(int value);
    public int remove();
}

public interface IntQueue {
    public boolean isEmpty();
    public void enqueue(int i);
    public int dequeue();
    public int size();
}

public class IntQueuePQ implements IntPriorityQueue {
    private IntQueue queue;

    public IntQueuePQ() { queue = new LinkedListQueue(); }

    public void add(int value) { queue.enqueue(value); }

    // THE REMOVE METHOD WOULD GO HERE
}
```

Write your `remove` method below.

Part B: Analysis

Given an efficient implementation of the `LinkedListQueue` class, what is the average-case runtime of your `remove` method? Give your answer in Big Oh notation.