CSE 373 Section Handout #3

Heaps and PriorityQueues

1. Implement a method *isHeap* to verify whether a given array of integers is a valid binary heap. isHeap should return `true` if the array is a binary heap, `false` otherwise. You may assume that the array provided represents a valid structure (in other words, check to see if the *order property* is satisfied

   For example, suppose a variable *array* contains the following sequence of values: [-, 2, 4, 6, 10] `isHeap(array)` would return true.

2. Implement the *contains* method for the IntPriorityQueue class. The method should return true if and only if the given key is contained within the priority queue.

   What is the runtime of the method? How does this compare to that of other PriorityQueue operations, as discussed in class?

   Note: **heap** is the array representation of a binary heap.

```
public class IntPriorityQueue {
      private int[] heap;
      private int size;

      public boolean contains(int key) {
            // your code here









      }
}
```

3. Write pseudocode to sort an array using a minHeap. What is the asymptotic worst case runtime of your method?

Binary Search Trees

Assume we are implementing the following IntTree class, assuming it is a **binary search tree**.

```
public class IntTree {
	private IntTreeNode root;

	private class IntTreeNode {
		IntTreeNode left;
		IntTreeNode right;
		int data;
	}

	// Add your methods here....
}
```

4. Implement the *contains* method for the IntTree class. What is the runtime of this method? How does this compare to the contains method for a heap/priority queue?

5. Implement *printSortedOrder*, a method which given the root of a BST, print all elements in sorted order.

Dictionaries and Amortized Analysis

6. Given some text, write pseudocode to show how to find the most frequently used word in the text. What is the runtime of your solution?

7. Consider the Stack ADT and a Stack implementation that starts with < N open spaces. What is the amortized cost of a series of N pushes? Do your analysis assuming push is O(1).

8. Consider augmenting the Stack ADT with an extra operation:

   multipop(k): Pops up to k elements from the Stack and returns the number of elements it popped

   What is the amortized cost of a series of pushes, pops, and multipops? Do your analysis assuming push and pop are both O(1) and the initial Stack is empty.