1. (10 points):
Which of the following data structures is most appropriate if speed of storage and removal of arbitrary items is most important?

(a) HashSet
(b) LinkedList
(c) Queue
(d) TreeSet

Which of the following list structures is most appropriate if additions and removals are at both the beginning and the end of the list?

(a) ArrayList
(b) Heap
(c) LinkedList
(d) Stack

What does the following method do?

```java
void f(HashMap theMap) {
    Set s = theMap.keySet();
    PriorityQueue p = new PriorityQueue();
    for (Object o:s)
        p.add(o);
    while (p.size() > 0) {
        Object key = p.poll(); // poll() removes the highest priority item
        System.out.println(key + " " + theMap.get(key));
    } // while
} // f()
```

(a) Empty the map.
(b) Shuffle the map.
(c) Sort the map by key and display the result.
(d) All of the above.
(e) None of the above.
2. (10 points): Show an inorder traversal of the following tree.

```
A
  / \  
B   C
  /   /
D   E
  /   /
F   G  
  /    /
H     
```
3. (10 points): Show the binary search tree that results from inserting the following numbers in the given order. The search order is numerical.

33 58 78 63 74 19 22 3 88 18
4. (10 points): Show the heap that results by inserting the following numbers into an empty heap in the given order.

\[3 \ 5 \ 8 \ 7 \ 6 \ 4 \ 1 \ 9 \ 2 \ 0\]
5. **(10 points)**: Show the output produced by the following Java program:

```java
public class PhoneNumberHasher {
    private int tableSize = 5;
    private String[] table = new String[tableSize];

    public int h(int value) {
        return sumOfDigits(value) % tableSize;
    } // h()

    public static int sumOfDigits(int n) {
        if (n < 10)
            return n;
        else
            return (n % 10) + sumOfDigits(n / 10);
    } // sumOfDigits()

    public void run() {
        int[] phoneNumbers = {3526, 1073, 2620, 3802, 6238, 3940};
        String[] offices = {"413", "423", "404J", "424", "429", "431"};
        for (int index = 0; index < phoneNumbers.length; ++index)
            table[h(phoneNumbers[index])] = offices[index];
        for (int index = 0; index < tableSize; ++index)
            if (table[index] != null)
                System.out.println(index + ": " + table[index]);
    } // run()

    public static void main(String[] args) {
        PhoneNumberHasher pnh = new PhoneNumberHasher();
        pnh.run();
    } // main()
} // class PhoneNumberHasher
```