ECE242
Fall 2009

2nd Midterm Examination
(120 Minutes, closed book)

Name: ______________________
Student ID: ______________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
</tr>
</tbody>
</table>

NOTE: Any questions on writing code must be answered in Java using Data Structures topics covered in lectures -
1. [10 points]
   Note: You do not need to write any code, but you do need to show each step in the sorting process for full credit. Clearly identify any pivot elements.
   a. [5 points] Sort the array below using selection sort.

   Sol:

   38  27  43  3  9  82  10  19
   3  27  43  38  9  82  10  19
   3  9  43  38  27  82  10  19
   3  9  10  38  27  82  43  19
   3  9  10  19  27  82  43  38
   3  9  10       19  27  38  43  82

   b. [5 points] Sort the array below using quick sort.

   94  21  89  30  4  82  45  1
   1  21  89  30  4  82  45  94
   1  21  4  30  89  82  45  94
   1  4  21  30  45  82  89  94
2. **[10 points]** You have a database consisting of a large array of objects and it performs the following functions:
   - It adds new objects.
   - When queried for an object it searches for that object, and if found returns a reference to the object.

   The priority of the database is to search for the object and return it as soon as possible.

   a. **[3 points]** In the first case, you are using linear search to search for the existence of a particular object in the database. What is the total complexity involved in inserting and searching for an object in the database.

   **Sol:**

   The average complexity of linear search is $O(n)$, while that of inserting an object into the database is $O(1)$. Hence the total complexity is $O(n)$.

   b. **[3 points]** Now you are using binary search to search for an object. What is the total complexity involved in searching for an object if the objects must also be sorted.

   **Sol:** The average complexity of binary search is $O(\log n)$, now when we insert an object we also need to sort the database. The complexity of sorting is either $O(n^2)$ (selection, bubble and insertion) or $O(n\log n)$ (merge and quick sort). Thus the total complexity is $O(n\log n)$ or $O(n^2)$.

   c. **[4 points]** Does it make sense to use a binary search algorithm to search a linked list? If yes, how would you go about it, if not, why?

   **Sol:** No. A separate search starting from the beginning of the list is required to find each intermediate binary search starting point.
3. **[20 points]** Consider a tree where each node has 3 children. The first element that you insert forms the root. The next three elements go to the left child, the middle child and the right child. The diagram below explains the concept. We get that tree by inserting the following sequence 39, 23, 34, 50, 20, 24, 25, 29, 40, 47, 49, 53, 99.

```
39
  23   50
 /  \ /  \
20  34 49  99
 /  \\  /  \\
24 25 40 53
```

a. **[3 points]** Print out the node labels using a post order traversal of the nodes

The output sequence is 20 24 25 29 40 47 34 49 53 99 50 39.

b. **[17 points]** Write a method in Java `postOrder` that will traverse such a tree in post order fashion. You can assume the root (a Node) is passed to the method.

The class definition of the Node is

```java
private class Node {
    private int id;
    private Node left;
    private Node mid;
    private Node right;

    public Node(int key, Node l, Node m, Node r) {
        id = key;
        left = l;
        mid = m;
        right = r;
    }
}
```
public void postorder(Node r) {
    if(r.left != null) postorder(r.left);
    if(r.mid != null) postorder(r.mid);
    if(r.right != null) postorder(r.right);
    System.out.println(r.id);
}

4. [20 points] You are given a populated BST which stores integers as elements.
   a. [17 points] Write a method void ntheelement (int n) that will locate and print
      the nth smallest element in the tree.

      The class definition of the BST Node is

      private class BSTNode
      {
        private int id;
        private BSTNode left;
        private BSTNode right;
        int count = 0;

        public BSTNode(int key, BSTNode l, BSTNode r) {
          id = key;
          left = l;
          right = r;
        }
      }
public void nthelement(int n) {
    inorder(root, n);
}

public void inorder(BSTNode r, int n) {
    if (r == null) return;
    inorder(r.left);
    count++;
    if (count == n) {
        System.out.print(r.id);
        return;
    }
    inorder(r.right);
}

b. [3 points] What is the complexity of this method? Provide a brief discussion supporting your answer.

Sol: Complexity is O(n), as you need to get to each element in the tree.
5. **[15 points]** Write one or more methods that perform merge sort on an input integer array. The result of the sort should be an array which has the biggest values at the start of the array (e.g. the sorted array should be in descending order). The top level `mergesort` method should return the value at the middle index in the sorted array if there are odd number of values, or the average of the two middle elements if there are even number of values.

**Sol:**

```java
public int mergesort (int[] array){
    mergeSort(array,0,array.length-1);
    if (array.length % 2 == 0){
        System.out.println("The array has even number of values");
        int count = (array.length / 2);
        return ((array[count] + array[count - 1])/2);
    } else{
        System.out.println("The array has odd number of values");
        int count = (array.length / 2);
        return array[count];
    }
}

public void mergeSort (int[] array, int min, int max)
{
    int [] temp = new int [array.length];
    int index1, left, right;

    /** return on list of length one */
    if (min==max)
        return;

    /** find the length and the midpoint of the list */
    int size = max - min + 1;
    int pivot = (min + max) / 2;
    //temp = (T[]) (new Comparable[size]);

    mergeSort(array, min, pivot); // sort left half of list
    mergeSort(array, pivot + 1, max); // sort right half of list

    /** copy sorted data into workspace */
    for (index1 = 0; index1 < size; index1++)
        temp[index1] = array[min + index1];

    /** merge the two sorted lists */
    left = 0;
    right = pivot - min + 1;
    for (index1 = 0; index1 < size; index1++)
    {
        if (right <= max - min)
            if (left <= pivot - min)
```
if (temp[left]<(temp[right]) )
    array[index1 + min] = temp[right++];

else
    array[index1 + min] = temp[left++];
else
    array[index1 + min] = temp[right++];
else
    array[index1 + min] = temp[left++];
}
}

6. **[25 points]** You are given a hash table that has 100 entries. Each entry is a reference that is a root of a binary search tree. A series of integer keys are to be stored in the hash table. When a new integer key arrives, its hash value is calculated by squaring the integer and considering the last two decimal digits as the index to the hash table. The key then needs to be inserted into the BST at that index. Each tree cannot have more than 5 elements (including the root). If an insertion must be performed for a BST with 5 elements, the smallest value in the tree should be replaced by the newest value to be inserted. Otherwise the new value can be inserted directly in the BST.
   
   a. **[5 points]** Create a method int hashCode(int) that implements the hashing function.
b. [20 points] Create a method insertKey(int) that inserts a key into the hash table while conforming to the above restrictions. Note: You do not need to implement the replaceminvalue and numOfKeys methods. You can assume that these methods are given.

The BST interface is as follows

```java
public class BST implements BSTInterface {
    public void insert(int id); // inserts a new key into the BST
    public void replaceminvalue(int id); // Replaces the min key
    // with the new key
    public int numOfKeys(); // Returns the number of keys
    // in the BST
}
```

```java
void insertKey(int value){
    System.out.println("Inserting the element "+value);
    int hashvalue = hashCode(value);
    if (table[hashvalue] != null){
        //System.out.println("an element already exists");
    }
}
```
System.out.println("The number of keys are "+table[hashvalue].numOfKeys());

if(table[hashvalue].numOfKeys() < 5){
    System.out.println("The number of keys are less than 5, inserting value ");
    table[hashvalue].insert(value);
}
else{
    System.out.println("The number of keys are equal to 5, replacing min value");
    table[hashvalue].replaceminvalue(value);
}
else{
    table[hashvalue] = new BST();
    table[hashvalue].insert(value);
}
}