CSE 373, Winter 2011
Practice Midterm

Name: _______________________________________

Student ID #: ____________________________

- You have 50 minutes to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This exam is open-book. You may not use any notes, slides, calculators, or other computing devices.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.
  The only abbreviations that are allowed for this exam are:
  - S.o.p for System.out.print,
  - S.o.pln for System.out.println, and
  - S.o.pf for System.out.printf.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.

Good luck!

Score summary: (for grader only)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Earned</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Big-Oh</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Sorting</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Trees and Heaps</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>Set Programming</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>TOTAL</td>
<td>Total Points</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

1 of 8
1. Big-Oh

Calculate the exact value of the variable \texttt{sum} after the following code fragment, in terms of variable \texttt{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.

```java
int sum = 0;
for (int i = 1; i <= n - 3; i++) {
    for (int j = 1; j <= i + 4; j += 2) {
        sum += 3;
    }
    sum++;
}
for (int i = 1; i <= 100; i++) {
    sum++;
}
```
2. Sorting

Consider the following array of int values.

\[7, 1, 6, 12, -3, 8, 4, 21, 2, 30, -1, 9\]

a. Write the contents of the array after 3 passes of the outermost loop of bubble sort.
b. Write the contents of the array after 4 passes of the outermost loop of selection sort.
c. Write the contents of the array after 5 passes of the outermost loop of insertion sort.
d. Write the contents of the array after all the recursive calls of merge sort have finished (before the very last merge begins).
e. Write the contents of the array after the first partitioning of quick sort has finished (before recursive calls).

Assume that the first element is chosen as the pivot.

Please show your work. You do not have to write an entirely new array after each pass of the algorithm, but since the final answer depends on every add/remove being done correctly, you may wish to show the array at various important stages (i.e. after each pass or recursive call) to help earn partial credit in case of an error.
3. Trees and Heaps

Given the following String elements:

"m", "x", "z", "s", "d", "b", "i", "t", "e", "g", "w", "k", "h"

Draw the tree that results when all of the above elements are added (in the given order) to each of the following initially empty data structures:

a. A binary search tree (BST).
   i. Draw the tree that results after adding all the elements.
   ii. Draw the tree that results after removing "k".
   iii. Draw the tree that results after removing "t".
   iv. Draw the tree that results after removing "x".
   v. Draw the tree that results after removing "m".

b. An AVL tree. Draw the tree that results after inserting all elements. Draw a new tree each time a rotation is necessary and say which kind of rotation was needed.

c. A minimum binary heap.
   i. Draw the heap after adding all the elements.
   ii. Perform three removes on the heap. Draw a new heap after each remove.

Please show your work. The next page is blank to give you space to write.
4. Set Programming

Part A: Implementation

One of the operations commonly performed on sets is intersection. The intersection of two sets contains all the items that the two sets have in common. The StringSet interface has been altered to have an intersect method. This method takes a StringSet as a parameter and returns a new StringSet that contains all the Strings that are in both the StringSet on which the method is called and the StringSet parameter. This method should not alter in any way the StringSet on which the method is called or the StringSet parameter.

The StringTreeSet class must now implement the intersect method. The public method has been written below. Write the recursive helper method intersect that this public method calls to populate the intersection StringSet to contain all the Strings that both this StringTreeSet instance and the other StringSet have. You may assume all of the other methods are implemented as discussed in lecture.

```java
public interface StringSet {
    public boolean add(String value);
    public boolean contains(String value);
    public StringSet intersect(StringSet other);
    public void print();
    public boolean remove(String value);
    public int size();
}

// A binary search tree implementation of a Set for Strings.
public class StringTreeSet implements StringSet {
    protected StringTreeNode root;

    ...

    public StringSet intersect(StringSet other) {
        StringSet intersection = new StringTreeSet();
        intersect(root, other, intersection);
        return intersection;
    }

    // YOUR RECURSIVE INTERSECT HELPER SHOULD GO HERE

    ...
}
```
Part B: Analysis

Consider the following code fragment:

```
StringSet s3 = s1.intersect(s2);
```

Assuming that \( s_2 \) has about half of the items that \( s_1 \) has, give the worst case running time of this method call if both \( s_1 \) and \( s_2 \) are of type `StringTreeSet` (i.e. a binary search tree implementation). Explain how you arrived at this running time.