CS 163 Practice Final Exam
Winter 2012

The final exam is Saturday, 21 April.

Any problem from either midterm or either practice midterm may (and often does) appear again on the final. In addition, make sure you are comfortable solving the following problems:

1. Given the following array of integers:

0	1	2	3	4	5	6	7	8	9
3	7	9	11	23	37	51	67	100	150

Perform a binary search for the value 37. List out the search steps below. In each step, the `min`, `mid`, and `max` variables are zero-based integer indexes into the list provided above.

1. `min` = 0

   `mid` = ______ is 37 greater than or less than this item? _____ (adjust `min` or `max` in next step)

   `max` = 9

2. `min` = ______

   `mid` = ______ is 37 greater than or less than this item? _____ (adjust `min` or `max` in next step)

   `max` = ______

3. `min` = ______

   `mid` = ______ is 37 equal to this item? If not, you made a mistake somewhere.

   `max` = ______

When is it appropriate to use binary search?

2. Sort the items in the following array in ascending (increasing) order using the **selection** sort. This algorithm `n-1` iterations/passes to sort an array with `n` items. For each pass of the sorting algorithm, show the values in the array at the end that pass.

85	**25**	**60**	**55**	**40**	**90**	**45**	**20**

What input will allow the <u>insertion sort</u> to run fastest (i.e., best-case scenario)? Why?

3. Sort the items in the following array in ascending (increasing) order using the **insertion** sort. This algorithm `n-1` iterations/passes to sort an array with `n` items. For each pass of the sorting algorithm, show the values in the array at the end that pass.

85	**25**	**60**	**55**	**40**	**90**	**45**	**20**

What input will allow the insertion sort to run fastest (i.e., best-case scenario)? Why?

4. Sort the items in the following array in ascending (increasing) order using the **bubble** sort. This algorithm `n-1` iterations/passes to sort an array with `n` items. For each pass of the sorting algorithm, show the values in the array at the end that pass.

85	**25**	**60**	**55**	**40**	**90**	**45**	**20**

How can you modify bubble sort so that it runs in `O(n)` time when the input is already sorted.

5. Given the list of numbers shown below, demonstrate how the Quick Sort algorithm works by showing the successive stages of the algorithm.

- Sort the numbers from lowest to highest (ascending order).
- Choose the left-most number in each segment as the pivot.
- Underline the pivot
- Surround each set of numbers to be sorted in brackets
- Stop using recursion when the segment length is reduced to three elements.

Initial Array: 10 9 7 4 8 11 19 3 1 2 15 20

Step	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**
0	10	9	7	4	8	11	19	3	1	2	15	20
1	[3	9	7	4	8	2	1]	<u>10</u>	[19	11	15	20]
2	[2	1]	<u>3</u>	[4	8	7	9]					
3									[15	11]	<u>19</u>	[20]
4				<u>4</u>	[8	7	9]					
Done	1	2	3	4	7	8	9	10	11	15	19	20

Initial Array: 9 5 7 8 20 19 13 11 12 6 4 18

Step	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**
0	9	5	7	8	20	19	13	11	12	6	4	18
1												
2												
3												
4												
5												
Done												

Initial Array: 11 6 18 8 7 19 3 4 5 16 9 20

Step	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**
0	11	6	18	8	7	19	3	4	5	16	9	20
1												
2												
3												
4												
5												
Done												

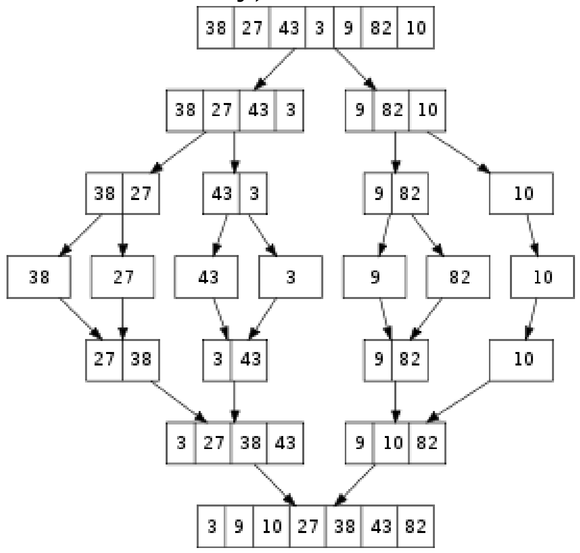
Initial Array: 6 7 12 9 4 16 15 8 19 1 14 20

Step	**[0]**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**	**[10]**	**[11]**
0	6	7	12	9	4	16	15	8	19	1	14	20
1												
2												
3												
4												
5												
Done												

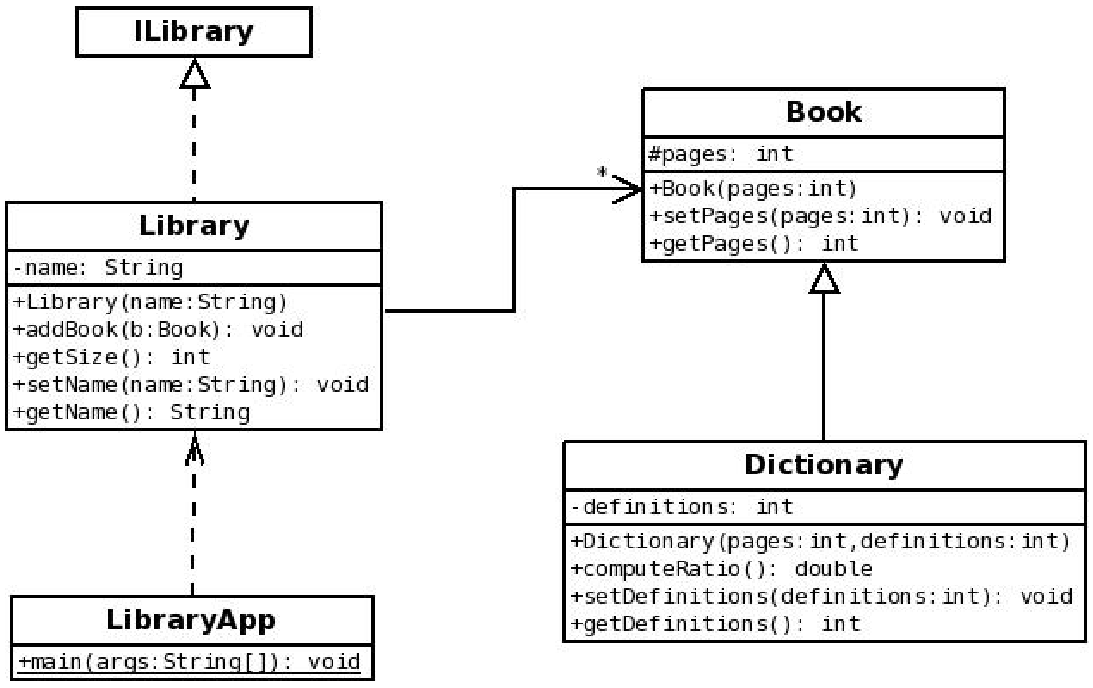
6. Trace the execution of merge sort on this array:

```
{90, 8, 7, 56, 123, 235, 9, 1, 653}
```

Your answer should be a diagram that is structured something like this (but with different values, obviously)



7. **[UML – 20 points]** Generate template/skeletal Java classes from the following class diagram. Make sure to include fields, methods, and relationships present in the class diagram.



8. Determine the Big-Oh running time for the method below. Assume `n` is the length of the array.

```
1    public static void isOrdered(int[] data) {
2      boolean answer = true;
3      for (int j = 0; j < array.length; j++) {
4        if (array[j] > array[j+1) {
5          answer = false;
6        }
7      }
8      return answer;
9    }
```

a) Count the number of times each of the following statements in the table is executed with respect to `n` (the length of the array). Assume the worst-case scenario.

Line/Statement	**# of Times Executed**
2	
3	
4	
5	
8	

b) The running time function, `T(n)`, of an algorithm is computed as the sum of the counts in the table above. Compute `T(n)`.

`T(n)` =

c) What is the Big-Oh of this program?

9. Determine the Big-Oh running time for the method below. Assume `n` is the length of the array.

```
1     public static int countSingletons(int[] data) {
2        int answer = data.length;
3        int j = 0;
4        while (j < data.length) {
5          int count = 0;
6          int k = j + 1;
7          while (k < data.length) {
8            if (data[j] == data[k]) {
9              count++;
10           }
11         }
12         if (count > 0) {
13            answer--;
14         }
15       }
16       return answer;
```

a) Count the number of times each of the following statements in the table is executed with respect to `n` (the length of the array):

Line/Statement	**# of Times Executed**
2	
3	
4	
5	
6	
7	
8	
9	
12	
13	
16	

b) The running time function, `T(n)`, of an algorithm is computed as the sum of the counts in the table above. Compute `T(n)`.

`T(n)` =

c) What is the Big-Oh of this program?

10. Add the following methods to the `SingleLinkedList` class we wrote during lecture.

   a) `int indexOf(T data)`
   b) `T removeFirst()`
   c) `T removeLast()`
   d) `T remove(int place)`
   e) `T remove(T object)`
   f) `void insertInOrder(T data)`. This method requires you to implement a new *unchecked* exception to throw if the user calls `insertInOrder` on an unsorted list.