CSE 373, Spring 2011 Midterm

1. Big-Oh (17 Points)

Calculate the approximate value of the variable `sum` after the following code fragment, in terms of variable `n`. Use summation notation to compute a closed-form solution (ignore small errors caused by `i` not being evenly divisible by 2). Then use this value to give a tightly bounded Big-Oh analysis of the runtime of the code fragment.

```
int sum = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j += 2) {
        sum += 4;
    }
}

for (int k = -50; k <= -1; k++) {
  sum--;
}
```

2. Sorting (18 Points)

Consider the follow array:

```
[32, 33, 5, 2, 14, -4, 22, 39, 34, -9]
```

Each of the following is a view of a sort in progress of the above array. Which sort is which? Each sort is used exactly once.

- Choose between **bubble sort**, **selection sort**, **insertion sort**, **shell sort**, **merge sort**, and **quick sort**.
- If the sorting algorithm contains multiple loops, the array is shown after a few of passes of the outermost loop has completed.
- If the shorting algorithm is **shell sort**, Shell's increments are used for the gap (i.e. the gap gets reduced by dividing by 2 each pass starting by dividing by the length of the array by 2).
- If the sorting algorithm is **merge sort**, the array is shown after the recursive calls have completed on each sub-part of the array.
- If the sorting algorithm is **quick sort**, the algorithm chooses the *first* element as its pivot. For quick sort, the array is shown before the recursive calls are made.

a. `[2, 5, 32, 33, 14, -4, 22, 39, 34, -9]`

Sorting Algorithm: ______________________________

b. `[2, 5, 14, 32, 33, -9, -4, 22, 34, 39]`

Sorting Algorithm: ______________________________

c. `[2, 5, -4, 14, 22, 32, -9, 33, 34, 39]`

Sorting Algorithm: ____

d. REMOVED

e. `[-9, 22, 5, 2, 14, -4, 32, 39, 34, 33]`

Sorting Algorithm: ____

f. `[-9, -4, 2, 5, 14, 33, 22, 39, 34, 32]`

Sorting Algorithm: ____

3. **AVL Trees** (25 Points)

a. Given the following list of integers:

   16, 35, 25, 7, 5, 20, 0

   Draw the **AVL tree** that results when all of the above elements are added (in the given order) to an initially empty AVL tree.

   Please show your work. You do not have to draw an entirely new tree after each element is added, but since the final answer depends on every add being done correctly, you may wish to show the tree at various important stages to help earn partial credit in case of an error.

b. What is the balance factor of the root node of the **AVL tree** that you drew for part 3(a)?

c. Draw a valid AVL tree of integers for which a single remove would cause the tree to become imbalanced in one of the two ways an insertion *cannot* cause an AVL tree to become imbalanced. For full credit, draw the **valid AVL tree** ***before*** **the remove** and say **which integer value** would have to be removed to cause the AVL tree to become imbalanced in a way not covered by the four insertion imbalance cases.

4. **Heaps** (20 Points)

Given the following integer elements:

85, 25, 45, 11, 3, 30, 1, 6

a. Draw the tree representation of the heap that results when all of the above elements are added (in the given order) to an initially empty **minimum binary heap**. ***Circle the final tree that results from performing the additions.***

b. After adding all the elements, perform **3 removes** on the heap. ***Circle the tree that results after the two elements are removed***.

Please show your work. You do not need to show the array representation of the heap. You do not have to draw an entirely new tree after each element is added or removed, but since the final answer depends on every add/remove being done correctly, you may wish to show the tree at various important stages to help earn partial credit in case of an error.

5. Heap Programming (20 Points)

In lecture, we implemented a class called `IntBinaryHeap`, a minimum binary heap for integers. Add a method to this class called `nodesAtLevel` that accepts as a parameter an integer representing a level in the heap and returns a `String` representation of all the values at that level in the order they are found when read left to right. The root level of the heap (i.e. the location of the minimum value) is level 0, level 1 contains the root's children, etc. An `IllegalArgumentException` should be thrown if `nodesAtLevel` is called with a level of less than 0, a level greater than the number of levels in the heap, or if the heap has no elements.

The following table shows some example calls to the `nodesAtLevel` and the `String`s returned when an `IntBinaryHeap` called `heap` is constructed and contains the values shown in Figure 1.

Call	**Returns**
`heap.nodesAtLevel(-1)`	*throws exception*
`heap.nodesAtLevel(0)`	`"-9"`
`heap.nodesAtLevel(1)`	`"-4 2"`
`heap.nodesAtLevel(2)`	`"33 5 32 22"`
`heap.nodesAtLevel(3)`	`"39 34 14"`
`heap.nodesAtLevel(4)`	*throws exception*

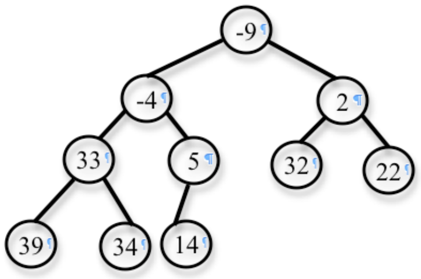

Figure 1 Heap

Index	0	1	2	3	4	5	6	7	8	9	10
Value	0	-9	-4	2	33	5	32	22	39	34	14

Figure 2 Array Representation of Above Heap

The class declaration of `IntBinaryHeap` along with its fields is given below. You should write `nodesAtLevel` as if your were adding it to this class. Recall that we implemented `IntBinaryHeap` with an array data structure (represented by the `array` field) and that the root (i.e. the minimum value) was stored at index 1. For an example of this representation, see Figure 1 and Figure 2.

You should take advantage of the fact `IntBinaryHeap` uses an array to implement your method in O(n) time. In other words, your method should not be recursive.

```
public class IntBinaryHeap implements IntPriorityQueue {
   private static final int DEFAULT_CAPACITY = 10;
   private int[] array;
   private int size;

   // THE nodesAtLevel METHOD GOES HERE
}
```

Write your `nodesAtLevel` method on the next page.