Name:_____

CSE373 Fall 2013, Midterm Examination
October 18, 2013

Please do not turn the page until the bell rings.

Rules:

- The exam is closed-book, closed-note, closed calculator, closed electronics.
- **Please stop promptly at 3:20.**
- There are **93 points** total, distributed **unevenly** among **7** questions (many with multiple parts):

Question	Max	Earned
1	14	
2	15	
3	18	
4	17	
5	5	
6	12	
7	12	

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly circle your final answer.**
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:_____

1. (**14** points) For each function $f(n)$ below, give an asymptotic upper bound using "big-Oh" notation. You should give the tightest bound possible (so giving $O(2^n)$ for every question is unlikely to result in many points). **You must choose your answer from the following list.** The list is not in any particular order and any item in the list could be the answer for 0, 1, or more than 1 question.

   $O(n), O(n^2), O(2^n), O(n^{1/2}), O(n^{1/4}), O(n^3), O(n^4), O(n^5), O(n^6), O(n^7), O(n^8), O(n^9),$
   $O(\log^3 n), O(n^2 \log n), O(\log^8 n), O(\log^4 n), O(n \log^3 n), O(\log^2 n), O(\log n), O(1), O(n^3 \log n),$
   $O(n^n), O(n \log \log n), O(n \log^2 n), O(\log \log n), O(n \log n)$

   (a) $f(n) = 100n^3 - 7n^3 + 14n^2$ _____

   (b) $f(n) = 100n^3 - 100n^3 + 7n^2$ _____

   (c) $f(n) = \log(7n^2)$ _____

   (d) $f(n) = 5 \log \log n + 4 \log^2(n)$ _____

   (e) $f(n) = .001n + 100 \cdot 2^n$ _____

   (f) $f(n) = n^3(1 + 6n + 2014n^2)$ _____

   (g) $f(n) = (\log n)(n + n^2)$ _____

   Solution:

   (a) $O(n^3)$
   (b) $O(n^2)$
   (c) $O(\log n)$
   (d) $O(\log^2 N)$
   (e) $O(2^n)$
   (f) $O(n^5)$
   (g) $O(n^2 \log n)$

Name:_____

2. (**15** points) Describe the worst case running time of the following code in "big-Oh" notation in terms of the variable `n`. You should give the tightest bound possible.

(a)
```
void f1(int n) {
  for(int i=0; i < n; i++) {
    for(int j=0; j < n; j++) {
      for(int k=0; k < n; k++) {
        for(int m=0; m < n; m++) {
          System.out.println("!");
  } } } }
}
```

(b)
```
void f2(int n) {
  for(int i=0; i < n; i++) {
    for(int j=0; j < 10; j++) {
      for(int k=0; k < n; k++) {
        for(int m=0; m < 10; m++) {
          System.out.println("!");
  } } } }
}
```

(c)
```
int f3(int n) {
  int sum = 73;
  for(int i=0; i < n; i++) {
    for(int j=i; j >= 5; j--) {
       sum--;
    }
  }
  return sum;
}
```

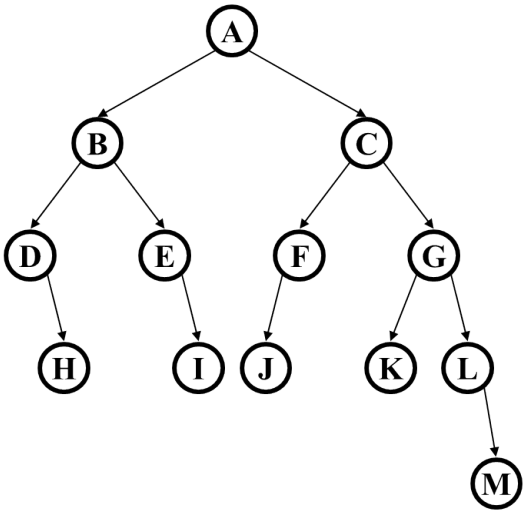
(d)
```
int f4(int n) {
  if (n < 10) {
    System.out.println("!");
    return n+3;
  } else {
    return f4(n-1) + f4(n-1);
  }
}
```

(e)
```
int f5(int n) {
  if (n < 10) {
    System.out.println("!");
    return n+3;
  } else {
    return f5(n-1) + 1;
  }
}
```

Solution:
(a) $O(n^4)$, (b) $O(n^2)$, (c) $O(n^2)$, (d) $O(2^n)$, (e) $O(n)$

Name:_____

3. (**18** points) Assume this tree is a binary search tree even though you cannot see what the keys and values are at the nodes (the letters we write below are just "names" for the nodes for the purpose of answering the questions).



(a) What node is the successor of node A?

(b) What node is the successor of node B?

(c) What node is the successor of node C?

(d) What is the height of the tree?

(e) What is the maximum number of nodes that could be added to the tree without increasing its height?

(f) Is the tree an AVL tree?

(g) If we remove only node H, is the result an AVL tree?

(h) If we remove only node J, is the result an AVL tree?

(i) If we remove only node K, is the result an AVL tree?

Solution:

(a) J

(b) E

(c) K

(d) 4

(e) 18

(f) yes

(g) yes

(h) no

(i) no

Name:_____

4. (**17** points) In this problem, you will write some **Java code** for simple operations on **binary search trees** where keys are ints. Assume you already have the following code. Assume the method bodies, even though not shown, are correct and implement the operations as we defined them in class.

```
public class BinarySearchTreeNode {
   public int key;
   public SomeType value;
   public BinarySearchTreeNode left;
   public BinarySearchTreeNode right;
}
public class BinarySearchTree {
   private BinarySearchTreeNode root;
   public void insert(int key, SomeType value) { ... }
   public void delete(int key) { ... }
   public SomeType find(int key) { ... }
}
```

Don't overlook part (c).

(a) Add a method `public int treeHeight()` to the `BinarySearchTree` class that computes the height of the tree. You will need a helper method.

(b) Add method `public void deleteMax()` to the `BinarySearchTree` class that deletes the maximum element in the tree (or does nothing if the tree has no elements). You may want a helper method.

(c) For your answers in part (a) and part (b), give a worst-case big-Oh running time in terms of n, where n is the number of nodes in the tree. Do not assume the tree is balanced.

There is room below and on the next page for solutions.

Name:_____

More space for solutions.

Solution:

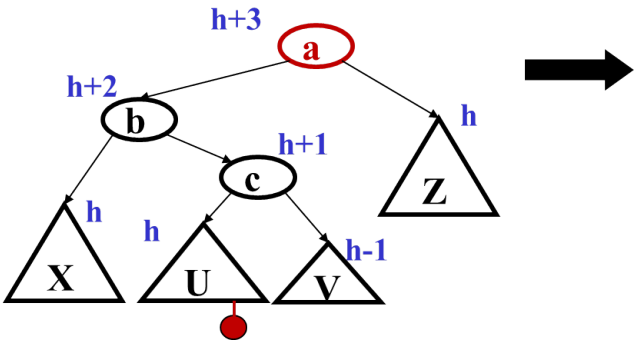
(a)
```
public int treeHeight() {
    return nodeHeight(root);
}
int nodeHeight(BinarySearchTreeNode n) {
  if(n==null)
    return -1;
  return 1 + Math.max(nodeHeight(n.left),nodeHeight(n.right));
}
```

(b)
```
public void deleteMax() {
    if(root == null)
       return;
    else if(root.right == null) // root is max
       root = root.left;
    else
       deleteMaxNode(root);
}
// recursive version
void deleteMaxNode(BinarySearchTreeNode prev) {
    if(prev.right.right == null)
      prev.right = prev.right.left; // partial credit: prev.right = null;
    else
      deleteMaxNode(prev.right);
}
// iterative version [helper method not needed]
void deleteMaxNode(BinarySearchTreeNode prev) {
    while(prev.right.right != null)
      prev = prev.right;
  prev.right = prev.right.left; // partial credit: prev.right = null;
}
// A third version that uses the provided delete method
public void deleteMax() {
    if(root == null)
      return;
    BinarySearchTreeNode n = root;
    while(n.right != null)
       n = n.right;
    delete(n.key);
}
```

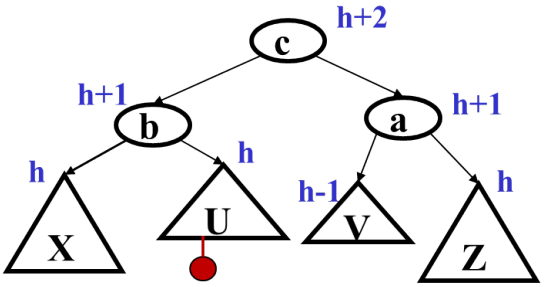
(c) Both operations are $O(n)$ in the worst case.

Name:_____

5. (**5** points) Below is part of the picture from lecture for how to do the double-rotation when an AVL-tree insertion causes the node **a** to violate the balance property because of an insertion in its left-right grandchild. Complete the picture to show the shape of the tree after the double-rotation. (You can just show the final result.) Include the heights of each node/subtree in the result similar to how they are included in the picture before the double-rotation.



Solution:



Name:_____

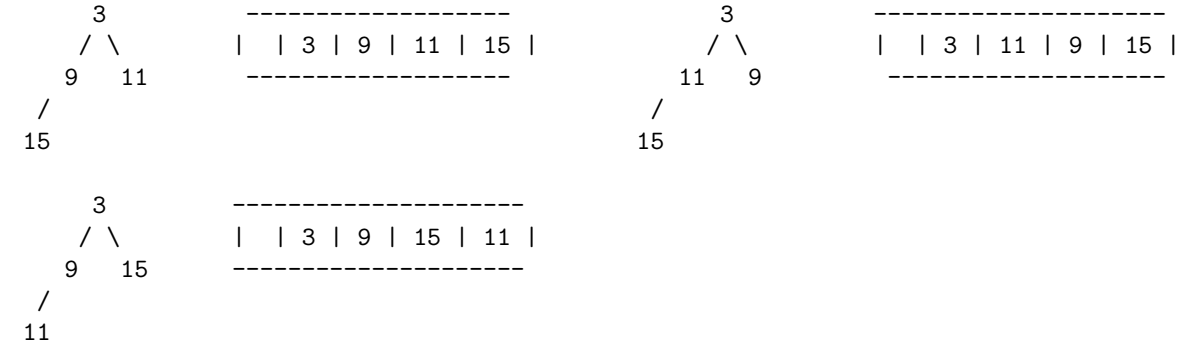
6. (**12** points) Suppose there is a binary min-heap with exactly 4 nodes, containing items with priorities 3, 9, 11, and 15.

   (a) Show **every** possible binary min-heap that could match this description. For each, draw the appropriate tree *and* the array representation. (You can show just the priorities, not the corresponding items.)

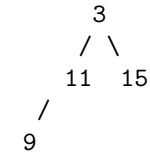
   (b) For **one** of your answers to part (a), show what happens with 4 `deleteMin` operations. Clearly indicate which heap you are starting with and show the heap after *each* `deleteMin`. You can just draw the tree (not the array) after each step.

Solution:

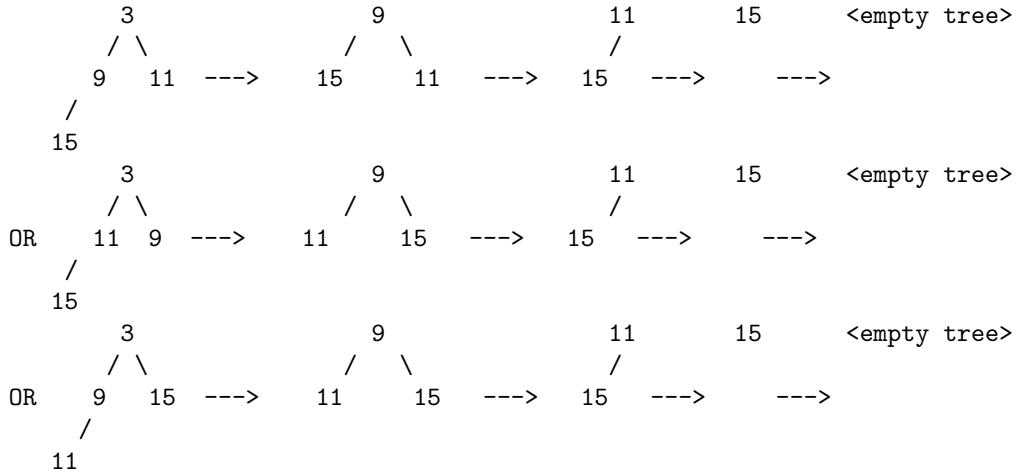
(a) Three possibilities:



Explanation: All 4-node heaps must have the shape of the 3 heaps in part (a). The minimum node, 3, must be at the root. The maximum node, 15, must be at a leaf. The only heap meeting these rules and not listed above does not satisfy the heap property because 11 is above 9:



(b)



Name:_____

7. (**12** points) Answer each of the following; no explanations required.

   (a) What is the worst-case asymptotic running-time for the best algorithm for finding something in a dictionary implemented with a sorted array?

   (b) What is the best-case asymptotic running-time for the best algorithm for finding something in a dictionary implemented with a sorted array?

   (c) What is the worst-case asymptotic running-time for the best algorithm for finding something in a dictionary implemented with a sorted linked list?

   (d) What is the best-case asymptotic running-time for the best algorithm for finding something in a dictionary implemented with a sorted linked list?

   (e) True or false: Every AVL tree is a binary search tree.

   (f) Complete this table appropriately:

2^{10}	about a thousand
2^{20}	
2^{30}	

Solution:
Note: We give answers here with O, but Θ would technically be more precise.

(a) $O(\log n)$
(b) $O(1)$
(c) $O(n)$
(d) $O(1)$
(e) true
(f)

2^{10}	about a thousand
2^{20}	about a million
2^{30}	about a billion