Name: ________Sample Solution ________________________

Email address: ______________________________________

CSE 373 Autumn 2012: Midterm #1

(closed book, closed notes, NO calculators allowed)

Instructions: Read the directions for each question carefully before answering. We may give partial credit based on the work you **write down**, so if time permits, show your work! Use only the data structures and algorithms we have discussed in class or which were mentioned in the book so far.

Note: For questions where you are drawing pictures, please circle your final answer for any credit.

Good Luck!

Total: 83 points. Time: 50 minutes.

Question	**Max Points**	**Score**
1	16	
2	8	
3	20	
4	15	
5	6	
6	8	
7	10	
Total	83	

1. (16 pts) **Big-O**
For each of the functions *f(N)* given below, indicate the tightest bound possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **You MUST choose your answer from the following** (not given in any particular order), each of which could be re-used (could be the answer for more than one of a) – h)):

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

You do not need to explain your answer.

a) $f(N) = 100 \log N^2 + 10 N^2 \log N$ $O(N^2 \log N)$

b) $f(N) = ((N + 1) \cdot (N + 2))/2$ $O(N^2)$

c) $f(N) = N^2 \cdot (2\log N + \log N) + N^3$ $O(N^3)$

d) $f(N) = N \log^2 N + N \log \log N$ $O(N \log^2 N)$

e) $f(N) = N^2 \cdot (N + 2N) + (N^3 \cdot N^3)$ $O(N^6)$

f) $f(N) = N^{1/4} + \log N$ $O(N^{\frac{1}{4}})$

g) $f(N) = (N^2)^3 + N^5$ $O(N^6)$

h) $f(N) = \log_8(N^8)$ $O(\log N)$

2. (8 pts) **Big-Oh and Run Time Analysis:** Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable `n`. ***Showing your work is not required***. You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of I. – IV.):

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

Runtime:

```
 I.  void smiley (int n, int sum) {
          for (int i = 0; i < n * 100; ++i) {
             for (int j = n; j > 0; j--)
                sum++;
             for (int k = 0; k < i; ++k)
                sum++;
          }
      }
```

$\boxed{O(n^2)}$

```
II.  int sunny (int n, int m, int sum) {
        if (n < 10)
          return n;
        else {
          for (int i = 0; i < n; ++i)
           sum++;
          return sunny (n - 2, m, sum);
        }
     }
```

$\boxed{O(n^2)}$

```
III.  void funny (int n, int x, int sum) {
         for (int k = 0; k < n; ++k) {
            if (x < 10) {
               for (int i = 0; i < n; ++i)
                  sum++;
            }
            for (int j = n; j > 0; j--)
               sum++;
         }
      }
```

$\boxed{O(n^2)}$

```
IV.  void happy (int n, int sum) {
        int j = n;
        while (j > 2) {
           sum++;
           j = j / 2;
        }
     }
```

$\boxed{O(\log n)}$

3. (20 pts total) **Trees**.

a) (4 pts) What is the minimum and maximum number of nodes in a balanced **AVL tree of height 5?** (Hint: the height of a tree consisting of a single node is 0) ***Give an exact number*** (with no exponents) for your answers – not a formula.

Minimum = 20

Maximum = 63

b) (4 pts) What is the minimum and maximum number of nodes in a **full tree of height 6?** ***Give an exact number*** (with no exponents) for your answers – not a formula.

Minimum = 13

Maximum = 127

c) (4 pts) What is the minimum and maximum number of **non-leaf** nodes in a **complete tree of height 5?** ***Give an exact number*** (with no exponents) for your answers – not a formula.

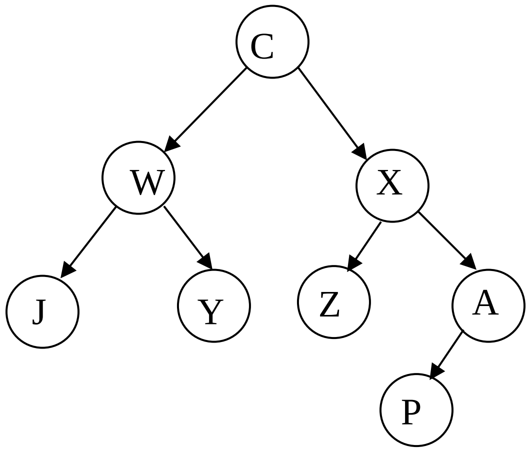
Minimum = 16

Maximum = 31

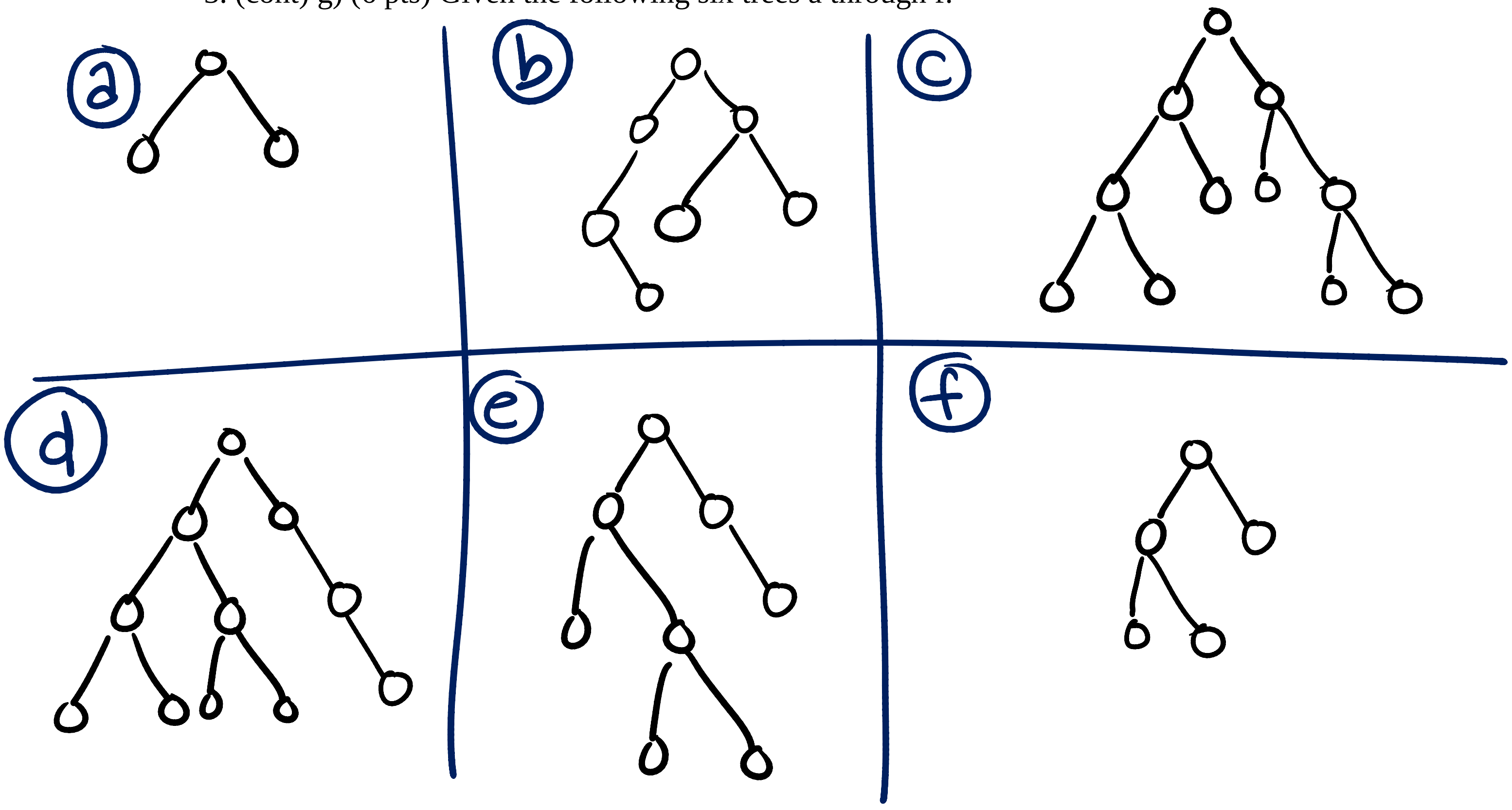
d) (1 pt) What is the **depth** of node X in the tree shown below: 1

e) (1 pt) Give a Post-Order traversal of the tree shown below:

J Y W Z P A X C



3. (cont) g) (6 pts) Given the following six trees a through f:



List the letters of all of the trees that have the following properties: (Note: It is possible that none of the trees above have the given property, it is also possible that some trees have more than one of the following properties.)

Full: a, c, f

Complete: a, f

AVL balanced: a, c, e, f

Perfect: a

4. (15 pts total) **Stacks – Please ANSWER PART (a) and PART (b).**
(a) (11 pts) Given the stack interface from homework 1, write Java code that will determine which value in the stack is the smallest and move that value to the top of the stack while otherwise leaving the remainder of the stack in its original order.

Thus if the `stack1` contains the `double` values: (bottom 4 2 8 3 6 5 top) then a call to `moveSmallestToTop(stack1)` would leave `stack1` as follows: (bottom 4 8 3 6 5 2 top)

In your solution, you may only declare extra `DStacks` or scalar variables (e.g. ints, doubles – not arrays or linked lists) if needed. You may not use other methods or classes from Java collections. You can assume that any `DStacks` you use will never become full and that the provided stack will not contain duplicate values. Rather than throwing an exception, if the provided stack is empty `moveSmallestToTop(DStack)` will do nothing. `moveSmallestToTop` does not return any values.

Unlike homework 1, assume that the only implementation of `DStack` that currently exists is the class `QuadStack`. `QuadStack` implements all of the operations of the `DStack` interface, but does them each at a cost of O(N^2). If you need to create any temporary `DStacks` in your code, you will need to create new `QuadStacks`. The interface for `DStack` is given below:

```
public interface DStack {
    // Returns true if stack is empty, false otherwise.
    public boolean isEmpty();

    // Adds d to the top of the stack.
    public void push(double d);

    // Returns and removes the value on top of stack.
    // @throws EmptyStackException if stack is empty
    public double pop();

    // Returns (but does not remove) the value on top of stack.
    // @throws EmptyStackException if stack is empty
    public double peek();
}
```

Please write your code for `moveSmallestToTop (DStack myStack)` on the next page.

(b) (4 pts) Since the only implementation of `DStack` that currently exists is the class `QuadStack`, all `DStack` parameters passed to `moveSmallestToTop (DStack myStack)` must be `QuadStack`s. Thus **all** stack operations `(isEmpty, push, pop, peek)` on any stacks in your code take time O(N^2) each.

What is the worst case Big-O running time of your implementation of `moveSmallestToTop`? For full credit give your answer in the most simplified form and the tightest bound possible. (No credit if your solution to part a) is not mostly correct.)

Runtime:

$\boxed{O(N^3)}$

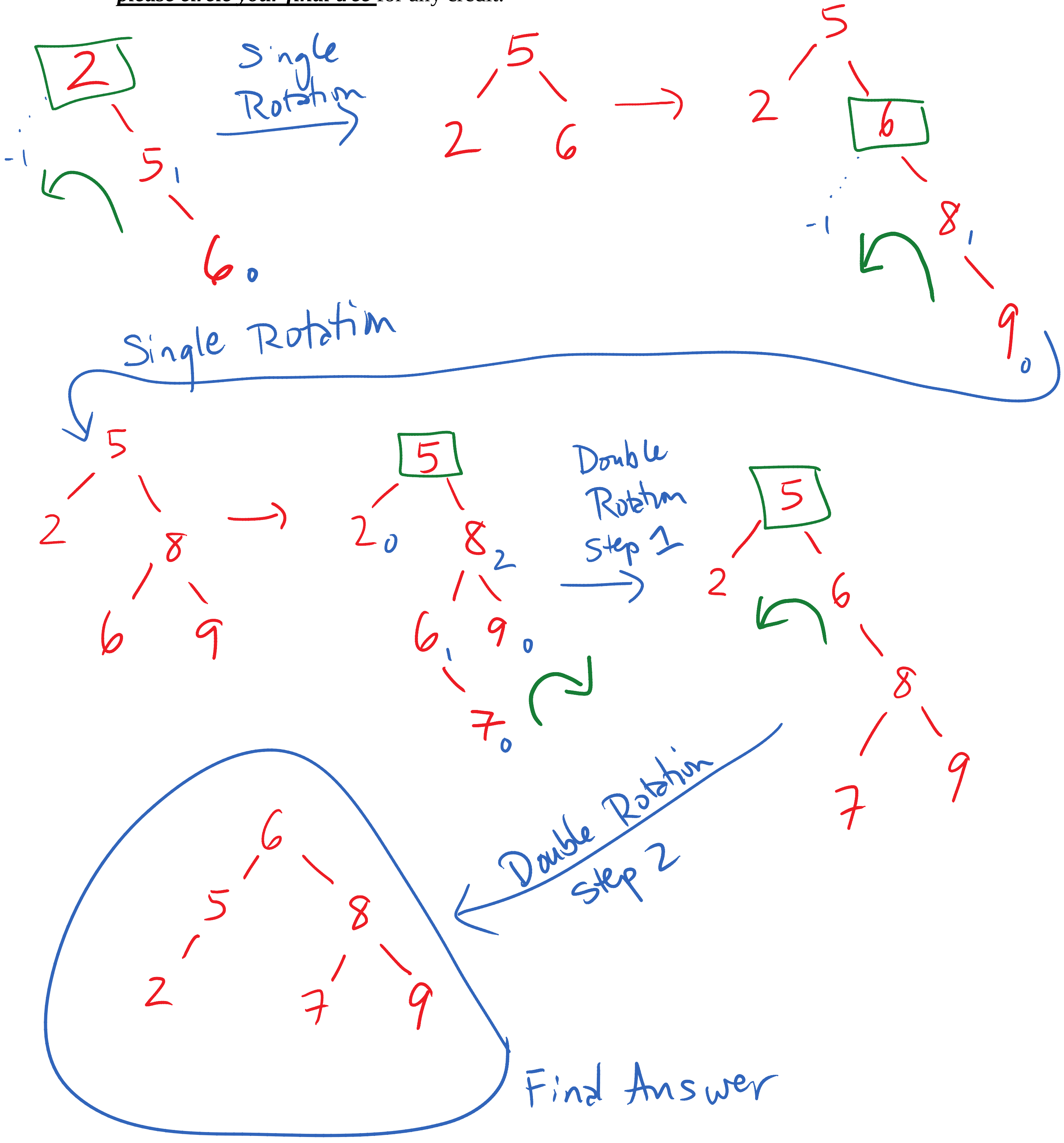
```
// Finds the smallest value in myStack and moves it to the top of the stack,
// otherwise leaves myStack in original order.
// Does nothing if myStack is empty.
public static void moveSmallestToTop (DStack myStack) {
        // Add your code here:

public static void moveSmallestToTop(DStack myStack) {
        if (!myStack.isEmpty()) {
                DStack tempStack = new QuadStack();
                double min = myStack.peek();
                // Find the smallest value, place values on tempStack
                while (!myStack.isEmpty()) {
                        double temp = myStack.pop();
                        if (temp < min) min = temp;
                        tempStack.push(temp);
                }
                // Put all values except min back on orig stack
                while (!tempStack.isEmpty()) {
                        double temp = tempStack.pop();
                        if (temp != min) myStack.push(temp);
                }
                myStack.push(min);
        }
}
```

←←←

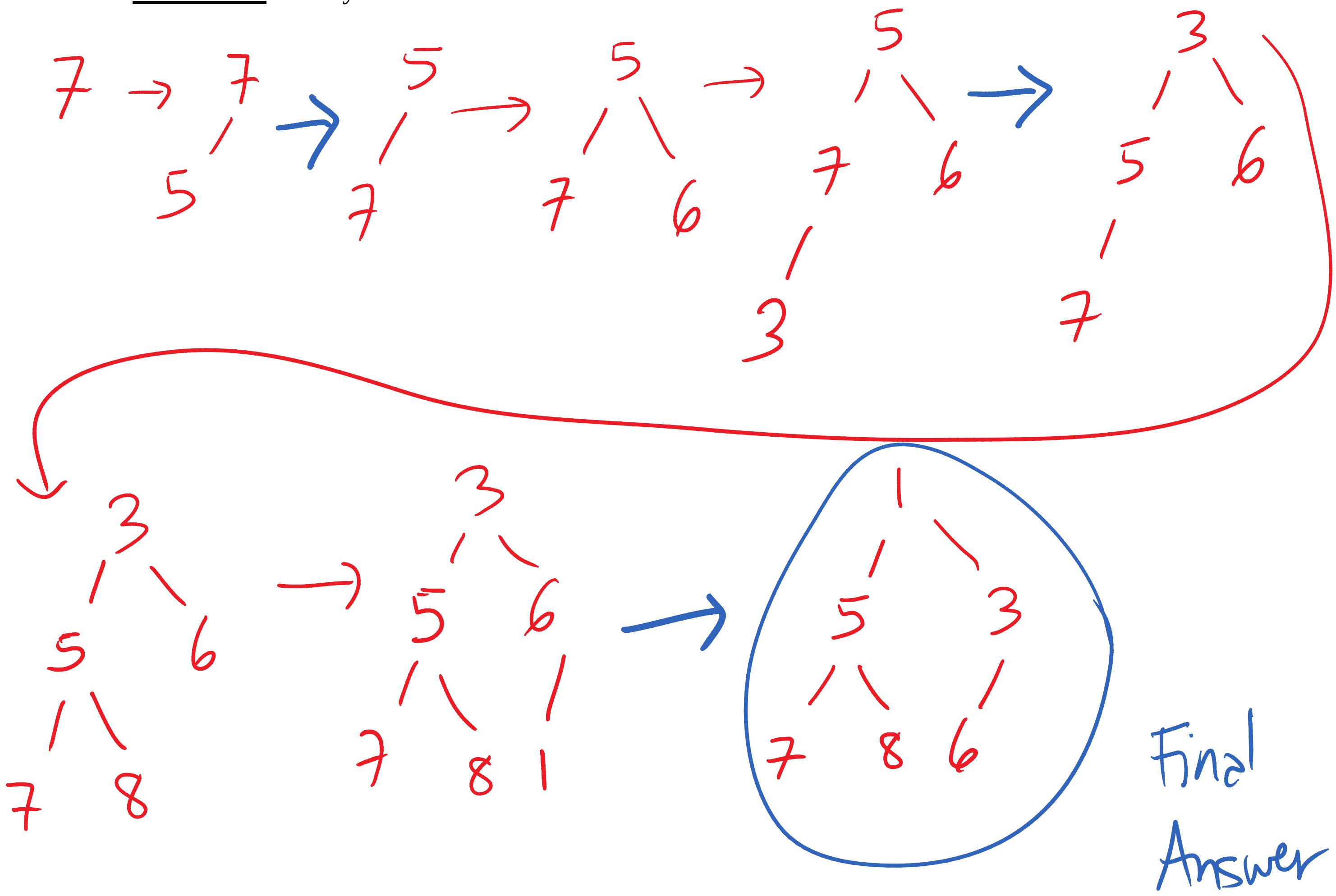
Don't forget to answer part b!!!!!

5. (6 pts) **AVL Trees** Draw the AVL tree that results from inserting the keys: 2, 5, 6, 8, 9, 7 in that order into an initially empty AVL tree. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, ***please circle your final tree*** for any credit.

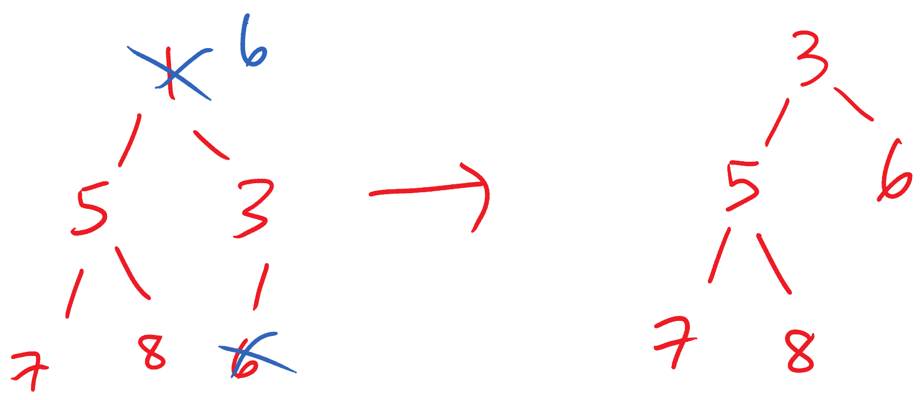


6. (8 pts) **Binary Min Heaps**

(a) [6 points] Draw the binary min heap that results from inserting the integers: 7, 5, 6, 3, 8, 1 in that order into an initially empty binary min heap.. *You do not need to show the array representation of the heap*. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, please **circle your final result** for any credit.



(b) [2 points] Draw the result of one deletemin call on your heap drawn at the end of part (a).



7. (10 pts) **Running Time Analysis:**

- Give the tightest possible upper bound for the ***worst case*** running time for each operation listed below in terms of *N*. Assume no duplicate values and that you can implement the operation as a member function of the class – with access to the underlying data structure, including knowing the number of values currently stored (N). For array-based implementations, assume that the array is large enough – does not need to be grown.
- **You do *NOT* need to give an explanation.** Although you may do so if you have time.
- You must choose your answer from the following (not listed in any particular order), each of which could be re-used (could be the answer for more than one item).

  $O(N^2)$, $O(N^{½})$, $O(\log^2 N)$, $O(N \log N)$, $O(N)$, $O(N^2 \log N)$, $O(N^5)$, $O(2^N)$, $O(N^3)$, $O(\log N)$, $O(1)$, $O(N^4)$, $O(N^{12})O(N^N)$, $O(N^6)$, $O(N^8)$, $O(\log \log N)$

a) Insert into a Dictionary implemented with an AVL tree. O(log N)

b) Findmin in a Priority Queue implemented with a binary search tree. O(N)

c) Pop from a Stack implemented with linked list nodes. O(1)

d) Deletemin from a Priority Queue implemented with a binary min heap. O(log N)

e) Find the maximum value in a binary min heap. O(N)