1) 10 Points

Compute an appropriately tight O (big-O) bound on the running time of each code fragment, in terms of *n*. Assume integer arithmetic. Circle your answer for each fragment.

a)
```
for(i = 0; i < n; i++) { for(j
     = 0; j < n; j++) {
         for(k = 0; k < i * j; k++) { sum++;
         }
     }
 }
```

b)
```
for(i = 1; i < n; i = i * 2) {
     for(j = 1; j < i; j++) {
         sum++;
     }
 }
```

c)
```
for(i = 0; i < n; i++) {
     myArray = new array[i];
     for(j = 0; j < i; j++) {
         myArray[j] = random();
     }
     mergeSort(myArray);
 }
```

d)
```
for(i = 0; i < n; i++) {
     tree = new UnbalancedBinarySearchTree(); for(j
     = 0; j < n; j++) {
         tree.insert(j);
     }
 }
```

e)
```
tree = new AVLTree();
for(i = 0; i < n; i++) {
     for(j = 0; j < n; j++) {
         tree.insert(random());
     }
}
```

Q3: O, Ω, and Θ (9 pts)

For each of the following statements, indicate whether it is always true, sometimes true, or never true. You do not need to include an explanation. Assume that the domain and codomain of all functions in this problem are natural numbers(1, 2, 3 ...).

a) A function that is O(n) is _____ $O(n^2)$.

○ Always ○ Never ○ Sometimes

b) A function that is $\Theta(n)$ is _____ $O(n^2)$

○ Always ○ Never ○ Sometimes

c) A function that is O(n) is _____ O(log n).

○ Always ○ Never ○ Sometimes

d) A function that is O(n) is _____ $\Theta(n^2)$

○ Always ○ Never ○ Sometimes

e) A function that is $\Omega(\log(n^{1/\log n}))$ is _____ $\Omega(n^{2/3})$.

○ Always ○ Never ○ Sometimes

f) A function that is Ω(log n) is _____ Ω(n)

○ Always ○ Never ○ Sometimes

g) A function that is O(n) is _____ $\Omega(n^2)$.

○ Always ○ Never ○ Sometimes

h) If f(n) is $\Theta(g(n))$ and g(n) is Ω(h(n)), then f(n) is Ω(h(n)).

○ Always ○ Never ○ Sometimes

i) If f(n) is both O(g(n)) and Ω(h(n)), then **g(n)** is $\Theta(h(n))$.

○ Always ○ Never ○ Sometimes

Q4: Write a Recurrence (5 pts)

Give a base case and a recurrence for the runtime of the following function. Use variables appropriately for constants (e.g. c_1, c_2, etc.) in your recurrence (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence.

```
public static int fun(int n) {
    if (n < 5) {
        return n;
    } else if (fun(n / 2) < 10) {
        int a = fun(n - 3);
        int b = fun(n / 2);
        return b;
    } else {
        int j = fun(n - 7);
        for (int i = 0; i < n * n; i += 4) {
            j += i;
        }
        return j;
    }
}
```

$T(n) =$ _____ For $n < 5$

$T(n) =$ _____ For $n > 5$

Yipee!!!! YOU DO NOT NEED TO SOLVE *this* recurrence…

Q6: Heaps (11 pts)

.

Here's an array presentation of a 0-indexed binary heap:

23	25	55	81	49	64	79	98	95	70	68		

a) (1 pt) This is a _____ (choose one: max/min) heap.
b) (2 pts) Draw the visual representation of the given heap.

c) (2 pts each) For each statement below, if there's an **insertion** that can achieve the goal, list the element to insert; otherwise, justify why it cannot be achieved (1-2 sentences max). Each statement is independent; you are allowed to insert one element for each statement.

i) 70 is a child of 64

ii) 79 is no longer a child of 55

iii) 49 only has one child

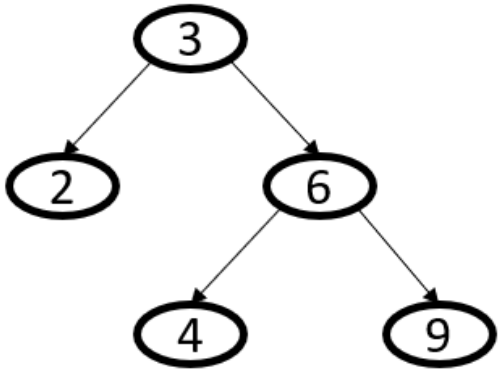
iv) No percolation is done for the insertion

Q7: AVL (12 pts)

a) (3 pts) Draw the AVL tree that results after **inserting 5** into this AVL tree. Be sure to draw your final tree in the box below AND **indicate the total number of single and double rotations required for this insertion.**

b) (1 pts) This insertion required: (SELECT ONE)

- ◯ One single rotation
- ◯ One double rotation
- ◯ One single rotation AND One double rotation
- ◯ More than one single rotation and one double rotation



Final Tree: