Pointer Analysis

Question 1. Consider a flow-insensitive, context-insensitive, but field-sensitive pointer analysis of the following program:

```
class Node {
        int v;
        Node left, right;
        Node(int v) { this.v = v; }
}


static void main() {
        Node root = new Node(5);    // h1
        for (int i = 0; i < 10; i++)
        add(root, i);
}
```

```
boolean add(Node this, int q) {
    int p = this.v;
    if (q < p) {
    Node n = this.left;
    if (n == null) {
            Node l = new Node(q);   // h2
            this.left = l;
            return true;
    }
    return add(n, q);
    }
    if (q > p) {
    Node m = this.right;
    if (m == null) {
            Node r = new Node(q);   // h3
            this.right = r;
            return true;
    }
    return add(m, q);
    }
    return false;
}
```

a. A flow-insensitive pointer analysis analyzes an (unordered) set of simple statements in the given program. Write the statements contained in this set for the above program (in the main and add functions together). Recall that this set includes only 4 kinds of statements: allocation (v = new h, where h is the commented label), copy (v1 = v2), field read (v1 = v2.f), and field write (v1.f = v2). The allocation statements are listed below for your convenience. You will lose points for listing statements not analyzed by this analysis.

Relevant allocation statements: `root = new h1,` `l = new h2,` `r = new h3`
Relevant copy statements: ______________________
Relevant field-read statements: ______________________
Relevant field-write statements: ______________________

Answer:

Relevant copy statements: `this = root,` `this = n,` `this = m`
Relevant field-read statements: `n = this.left,` `m = this.right`
Relevant field-write statements: `this.left = l,` `this.right = r`

b. For each pair of expressions below, write all correct choices among 1-3 below.
1. They do NOT point to the same object in any (concrete) run of the program.
2. The above pointer analysis proves using allocation-site based heap abstraction that they CANNOT point to the same object in any run of the program.

3. The above pointer analysis proves using the type based heap abstraction that they CANNOT point to the same object in any run of the program.

```
A. root, root.right                         1   2   3
B. root.left, root.right                    1   2   3
C. root.left.right, root.right.left         1   2   3
D. root.left.left, root.right.right         1   2   3
E. root.left.right, root.right.right        1   2   3
F. root.left.right.left, root.right.left    1   2   3
```

Answer:
A. 1, 2
B. 1, 2
C. 1, 2
D. 1, 2
E. 1
F. 1