T.A. Help Session: Union Find

ANSWER KEY

1. a. Show the resulting up Up-Tree data structure after the following unions (no union-by-size):

union(1,5), union(3,7), union(5,6), union(1,4), union(6,2), union(5,3)

b. Fill in the resulting up-tree array implementation with

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>.weight</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

What is the worst-case runtime of a union operation? O(1)
find operation? O(N)

c. Now show the resulting up-tree data structure after the following finds, utilizing path compression:

find(7)
find(2)
2. Pseudocode
a. Write pseudocode for a basic union operation:
   ```java
   int[] up;
   public void union(int x, int y) {
       up[y] = x;
   }
   ```

b. Write pseudocode for a union-by-weight operation:
   ```java
   int[] up;
   int[] weight;
   // Assume x and y are roots
   public void unionByWeight(int x, int y) {
       if (weight[x] > weight[y]) {
           up[y] = x;
           weight[x] += weight[y];
       } else {
           up[x] = y;
           weight[y] += weight[x];
       }
   }
   ```

   // OPTIMIZED, assumption that x and y are roots
   ```java
   int[] up;
   public void unionByWeight(int x, int y) {
       if (up[x] < up[y]) { // x has heavier *negative* weight
           up[x] += up[y];
           up[y] = x;
       } else {
           up[y] += up[x];
           up[x] = y;
       }
   }
   ```

c. Write pseudocode for a basic find operation:
   ```java
   int[] up;
   public int find(int x) {
       while (up[x] > 0) {
           x = up[x];
       }
       return x;
   }
   ```
d. Write pseudocode for a find operation, implementing path compression:

```java
int[] up;
public int findWithPathCompression(int x) {
    int root = x;
    while (up[root] > 0)
        root = up[root];

    if (root == x)
        return root;

    int oldParent = up[x];
    while (oldParent != root) {
        up[x] = root;
        x = oldParent;
        oldParent = up[x];
    }
    return root;
}
```