Week 1 Solutions

Solutions for week 1 of Algorithms and Data Structures.

1.1.14 Algorithm Design

Divide N by 2, and update N to be to floor of the result (automatic in Java, by calling int(result) in Python). Do this as long as N is larger than 1 and simply count the number of divisions made.

```
# Because Python is pseudo-code as code:

def lg(n: int) -> int:
  current_lg = 0
  while n > 1:
    current_lg +=1
    n = int(n / 2)

  return current_lg
```

1.5.1 Quick-find

```
Initialization
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 3 4 5 6 7 8 9

9-0
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 3 4 5 6 7 8 0
Array accesses:
13 (2 in 2x find() + 10 for checking parents + 1 for updating parents)

3-4
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 5 6 7 8 0
Array accesses:
13 (2 in 2x find() + 10 for checking parents + 1 for updating parents)

5-8
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 8 6 7 8 0
Array accesses:
13 (2 in 2x find() + 10 for checking parents + 1 for updating parents)

7-2
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 8 6 2 8 0
Array accesses:
13 (2 in 2x find() + 10 for checking parents + 1 for updating parents)
```

```
2-1
        0 1 2 3 4 5 6 7 8 9
array = 0 1 1 4 4 8 6 1 8 0
Array accesses:
14 (2 in 2x find() + 10 for checking parents + 2 for updating parents)

5-7
        0 1 2 3 4 5 6 7 8 9
array = 0 1 1 4 4 1 6 1 1 0
Array accesses:
14 (2 in 2x find() + 10 for checking parents + 2 for updating parents)

0-3
        0 1 2 3 4 5 6 7 8 9
array = 4 1 1 4 4 1 6 1 1 4
Array accesses:
14 (2 in 2x find() + 10 for checking parents + 2 for updating parents)

4-2
        0 1 2 3 4 5 6 7 8 9
array = 1 1 1 1 1 1 6 1 1 1
Array accesses:
16 (2 in 2x find() + 10 for checking parents + 4 for updating parents)
```

1.5.2 Quick-union

```
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 3 4 5 6 7 8 9

9-0
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 3 4 5 6 7 8 0
Array accesses: 3 (1 for find(9), 1 for find(0) and 1 for updating parent)

Forest:
0 1 2 3 4 5 6 7 8
9

3-4
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 5 6 7 8 0
Array accesses: 3 (1 for find(3), 1 for find(4) and 1 for updating parent)

Forest:
0 1 2 4 5 6 7 8
9     3

5-8
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 8 6 7 8 0
Array accesses: 3 (1 for find(5), 1 for find(8) and 1 for updating parent)

Forest:
0 1 2 4 6 7 8
9     3     5

7-2
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 4 4 8 6 2 8 0
Array accesses: 3 (1 for find(7), 1 for find(2) and 1 for updating parent)

Forest:
0 1 2 4 6 8
9   7 3   5
```

```
2-1
        0 1 2 3 4 5 6 7 8 9
array = 0 1 1 4 4 8 6 2 8 0
Array accesses: 3 (1 for find(2), 1 for find(1) and 1 for updating parent)

Forest:
0 1 4 6 8
9 2 3   5
  7

5-7
        0 1 2 3 4 5 6 7 8 9
array = 0 1 1 4 4 8 6 2 1 0
Array accesses: 9 (3 for find(5), 5 for find(7) and 1 for updating parent)

Forest:
0   1   4 6
9  2 8  3
   7 5

0-3
        0 1 2 3 4 5 6 7 8 9
array = 4 1 1 4 4 8 6 2 1 0
Array accesses: 5 (1 for find(0), 3 for find(3) and 1 for updating parent)

Forest:
 1    4   6
2 8  3 0
7 5    9

4-2
        0 1 2 3 4 5 6 7 8 9
array = 4 1 1 4 1 8 6 2 1 0
Array accesses: 5 (1 for find(4), 3 for find(2) and 1 for updating parent)

Forest:
  1      6
2 8  4
7 5 3 0
      9
```

1.5.3 Weighted Quick-union

```
        0 1 2 3 4 5 6 7 8 9
array = 0 1 2 3 4 5 6 7 8 9

9-0
        0 1 2 3 4 5 6 7 8 9
array = 9 1 2 3 4 5 6 7 8 9
Array accesses: 3 (1 for find(9), 1 for find(0) and 1 for updating parent)

Forest:
1 2 3 4 5 6 7 8 9
                0

3-4
        0 1 2 3 4 5 6 7 8 9
array = 9 1 2 3 3 5 6 7 8 9
Array accesses: 3 (1 for find(3), 1 for find(4) and 1 for updating parent)

Forest:
1 2 3 5 6 7 8 9
    4         0

5-8
        0 1 2 3 4 5 6 7 8 9
array = 9 1 2 3 3 5 6 7 5 9
Array accesses: 3 (1 for find(5), 1 for find(8) and 1 for updating parent)

Forest:
1 2 3 5 6 7 9
    4 8     0

7-2
        0 1 2 3 4 5 6 7 8 9
array = 9 1 7 3 3 5 6 7 5 9
Array accesses: 3 (1 for find(7), 1 for find(2) and 1 for updating parent)

Forest:
1 3 5 6 7 9
  4 8   2 0
```

```
2-1
        0 1 2 3 4 5 6 7 8 9
array = 9 7 7 3 3 5 6 7 5 9
Array accesses: 5 (2 for find(3), 1 for find(1) and 1 for updating parent)

Forest:
3 5 6  7  9
4 8   2 1 0

5-7
        0 1 2 3 4 5 6 7 8 9
array = 9 7 7 3 3 7 6 7 5 9
Array accesses: 3 (1 for find(5), 1 for find(7) and 1 for updating parent)

Forest:
3 6   7    9
4   2 1 5  0
        8

0-3
        0 1 2 3 4 5 6 7 8 9
array = 9 7 7 9 3 7 6 7 5 9
Array accesses: 5 (3 for find(0), 1 for find(3) and 1 for updating parent)

Forest:
6   7     9
  2 1 5  0 3
      8    4

4-2
        0 1 2 3 4 5 6 7 8 9
array = 9 7 7 9 3 7 6 7 5 7
Array accesses: 9 (5 for find(4), 3 for find(2) and 1 for updating parent)

Forest:
6     7
   2 1 5  9
       8 0 3
           4

Please note that the size array is also accessed 4 times for each
union operation.
```

1.5.8 Incorrect union()

In the loop, id[p] will eventually be set to id[q], losing the reference to the original parent. This will make the next elements with id[i] == id[p] to not have their values updated.

Counterexample:

Original Array
0 1 2 3 4 5 6
0 1 2 2 4 4 6

Perform union(2,5)
p = 2
q = 5
id[p] = 2
id[q] = 4

Array[0]: 0 != 2, is not updated
Array[1]: 1 != 2, is not updated
Array[2]: 2 == 2, is updated to 4 (after this, all comparisons are incorrectly made with 4 instead of 2)

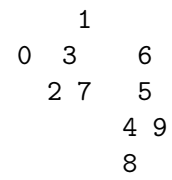
Updated Array
0 1 2 3 4 5 6
0 1 4 2 4 4 6
id[p] = 4

Array[3]: 2 != 4 is not updated (and should have been updated)
Array[4]: 4 == 4 is updated to 4 (again)
Array[5]: 4 == 4 is updated to 4 (again)
Array[6]: 6 != 4 is not updated

Final Array
0 1 2 3 4 5 6
0 1 4 2 4 4 6

id[3] is still 2, but should have been 4.

1.5.9 Weighted Quick-union Tree?



This forest cannot be the result of running weighted quick-union.

Consider the tree composed of nodes 1, 6, 5, 4, 9, 8
Its height is 4, which is higher than lg N
lg 6 = 3 < 4

Therefore it does not hold the proposition that the depth of any node in a forest built by weighted quick-union for N sites is at most lg N.