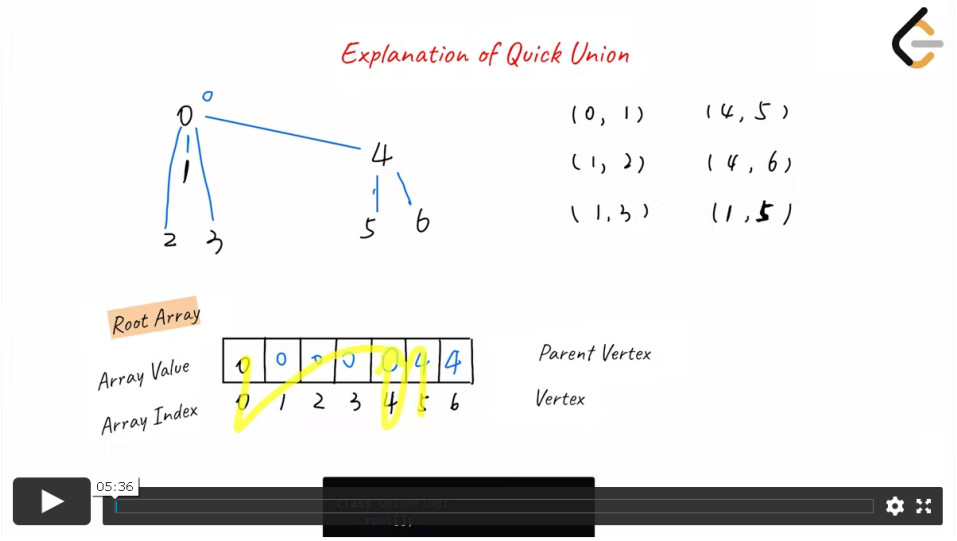
Quick Union - Disjoint Set

Report Issue

Explanation of Quick Union

In the following video we'll take a look at Quick Union implementation of a Disjoint Set and show the difference between the Quick Union implementation and the Quick Find implementation we talked about earlier. As previously done for the Quick Find implementation, we'll also derive the time complexity of the Quick Union operations so you can compare them.



Why is Quick Union More Efficient than Quick Find?

Generally speaking, Quick Union is more efficient than Quick Find. We'll explain the reason in the below video.

▶ Clarifying Notes

Algorithm

Here is a sample quick union implementation of the Disjoint Set.

C++ Java Python3 Copy Run Playground

```
class UnionFind {
```

```
public:
    UnionFind(int sz) : root(sz) {
        for (int i = 0; i < sz; i++) {
            root[i] = i;
        }
    }

    int find(int x) {
        while (x != root[x]) {
            x = root[x];
        }
        return x;
    }

    void unionSet(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            root[rootY] = rootX;
        }
    }

    bool connected(int x, int y) {
        return find(x) == find(y);
    }
```

Time Complexity

	Union-find Constructor	Find	Union	Connected
Time Complexity	$O(N)$	$O(N)$	$O(N)$	$O(N)$

Note: N is the number of vertices in the graph. In the worst-case scenario, the number of operations to get the root vertex will be H where H is the height of the tree. Because this implementation does not always point the root of the shorter tree to the root of the taller tree, H can be at most N when the tree forms a linked list.

- The same as in the quick find implementation, when initializing a `union-find constructor`, we need to create an array of size N with the values equal to the corresponding array indices; this requires linear time.
- For the `find` operation, in the worst-case scenario, we need to traverse every vertex to find the root for the input vertex. The maximum number of operations to get the root vertex would be no more than the tree's height, so it will take $O(N)$ time.
- The `union` operation consists of two `find` operations which (**only in the worst-case**) will take $O(N)$ time, and two constant time operations, including the equality check and updating the array value at a given index. Therefore, the `union` operation also costs $O(N)$ in the worst-case.
- The `connected` operation also takes $O(N)$ time in the worst-case since it involves two `find` calls.

Space Complexity

We need $O(N)$ space to store the array of size N.