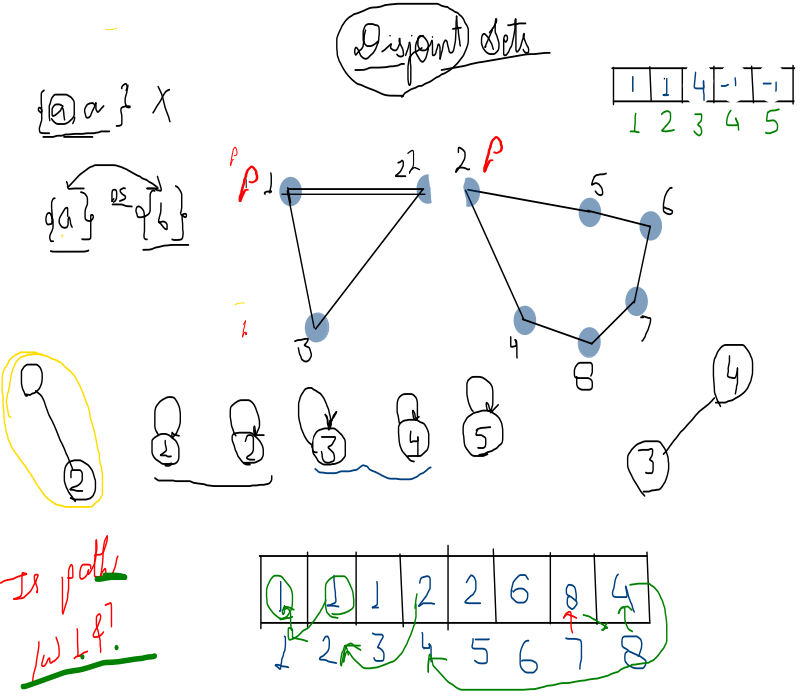


1. find
2. union

```
class UnionFind:                    quick find
    root[];

    UnionFind(size):
        root = [0] * size;
        for i in [0, size):
            root[i] = i;

    int find(x):
        return root[x];

    void union(x, y):
        rootX = find(x);
        rootY = find(y);
        if rootX ≠ rootY:
            for i in [0, root.length):
                if (root[i] == rootY):
                    root[i] = rootX;

    boolean connected(x, y):
        return find(x) == find(y);
```

Source : Leetcode...for any discrepency contact repo owner, it will be removed.

Quick Find

```
class UnionFind:
    root[];

    UnionFind(size):
        root = [0] * size;
        for i in [0, size):
            root[i] = i;

    int find(x):
        return root[x];

    void union(x, y):
        rootX = find(x);
        rootY = find(y);
        if rootX ≠ rootY:
            for i in [0, root.length):
                if (root[i] == rootY):
                    root[i] = rootX;

    boolean connected(x, y):
        return find(x) == find(y);
```

	Union-find Constructor	Find	Union	Connected
Time Complexity	$O(N)$	$O(1)$	$O(N)$	$O(1)$

Efficiency

Quick Union

```
class UnionFind:
    root[];

    UnionFind(size):
        root = [0] * size;
        for i in [0, size):
            root[i] = i;

    int find(x):
        if x == root[x]:
            return x;
        return root[x] = find(root[x]);

    void union(x, y):
        rootX = find(x);
        rootY = find(y);
        if rootX ≠ rootY:
            root[rootY] = x;

    boolean connected(x, y):
        return find(x) == find(y);
```

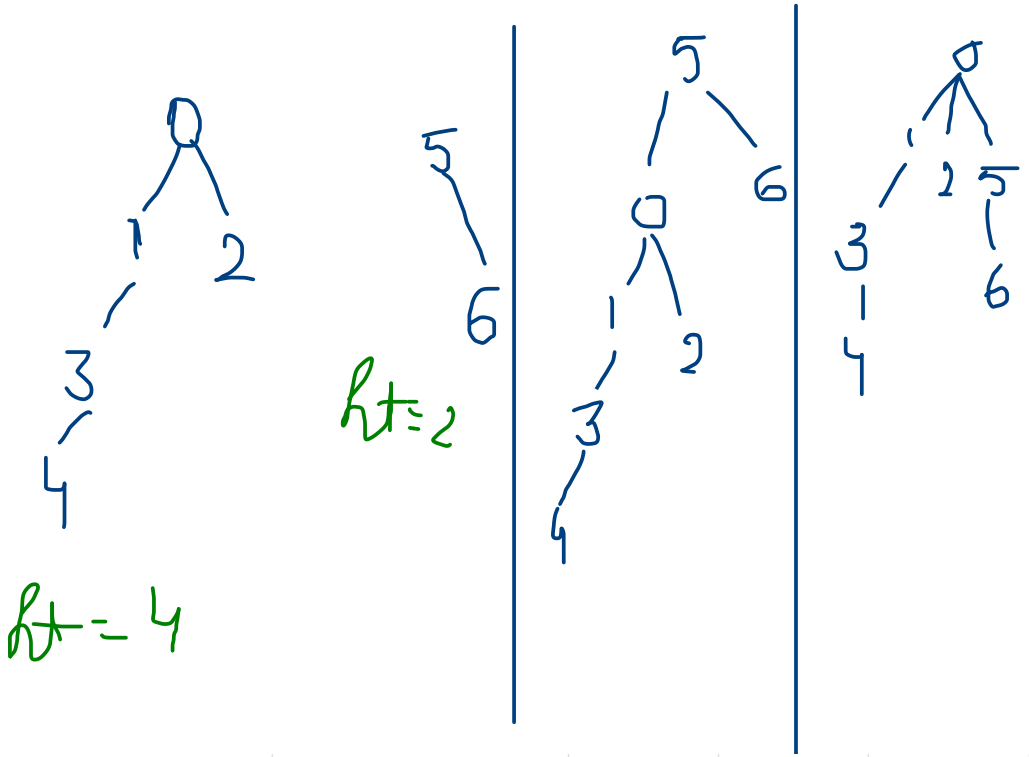
	Union-find Constructor	Find	Union	Connected
Time Complexity	$O(N)$	$O(N)$	$O(N)$	$O(N)$

```cpp
class UnionFind {
public:
    /* Optimised Quick Union */
    /* Rank stores height if each set */
    UnionFind(int sz) : root(sz), rank(sz) {
        for (int i = 0; i < sz; i++) {
            root[i] = i;
            rank[i] = 1;
        }
    }

    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) {
            if (rank[rootX] > rank[rootY]) {
                root[rootY] = rootX;
            } else if (rank[rootX] < rank[rootY]) {
                root[rootX] = rootY;
            } else {
                root[rootY] = rootX;
                rank[rootX] += 1;
            }
        }
    }

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

private:
    vector<int> root;
    vector<int> rank;
};
```

if both sets have same height, then make rootX root node and increment it's height

Union by Rank (ht)

Clearly, more balanced the tree less time it takes to find the height



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

Path Compression Optimization

After finding the root node, we can update the parent node of all traversed elements to their root node. When we search for the root node of the same element again, we only need to traverse two elements to find its root node

```
int find(int x) {
    if (x == root[x]) {
        return x;
    }
    return root[x] = find(root[x]);
}
```

simply...storing the result of root node

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

Memorize Optimized Implementation

$\alpha \rightarrow$ Inverse Ackerman Funtion