Disjoint Set.

Let's understand it by taking a question.

I/p: n = 4

makeFriends(0,1), makeFriends(1,3)

areFriends(0,2), areFriends(0,1), areFriends(0,3)

O/p: No Yes Yes.

→ Disjoint set is related to 2 operations Union & Find.

Naive Sol.

→ Not using Disjoint Sets.

→ Consider given elements as nodes in undirected graph, if 2 are friend of each other, edge exists.

Can we adj matrix (or) list.

O(n) O(n).

Eff sol (using Disjoint)

→ Can be used anywhere, when we have a (universe/set) & we have multiple subsets of it as disjoint subsets.

→ find(x): Returns a representative of x's set (i.e returns friends of x)

O(1)

union(x,y): Combines set x and y.

(same as makeFriends()).

O(1),

∴ To Solve this problem fns are:-

Ans ① boolean arefriends (x, y)

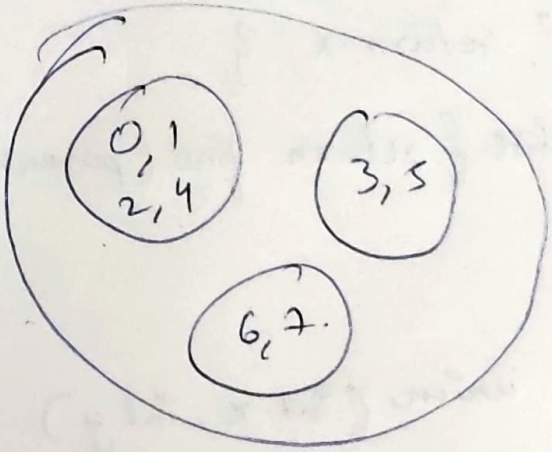
```
{
    return (find(x) == find(y));
}
```

② makefriends (x, y) {

```
    union(x, y);
}
```

Eg:

find(x) ⟶

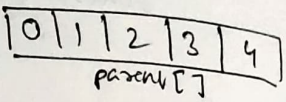


will return same value for 0, 1, 2, 4.

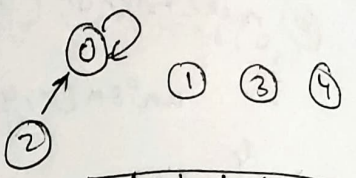
Implementation of Union and Find operations.

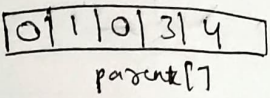
```
int parent[n];
void initialize()
{ for(int i=0; i<n; i++)
  { parent[i] = i; }
}
```

n = 5



union(0,2)






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

void union(int x, int y)
{ int x_rep = find(x);
  int y_rep = find(y);
  if(x_rep == y_rep) return;
  parent[y_rep] = x_rep;
}
```

Notes:-

* This is a simple implementation, having O(n) for both of you can include rank[], to make both O(1).
  (RANK)
* find can further be optimised using (PATH COMPRESSION)