Construct DisJoint Set for the below Graphs

Case 2: Design DisJoint Set with QuickUnion.

Here the number of Vertices are 8 and non of them are connected and we know that every individual vertex can act as root to ItSelf.

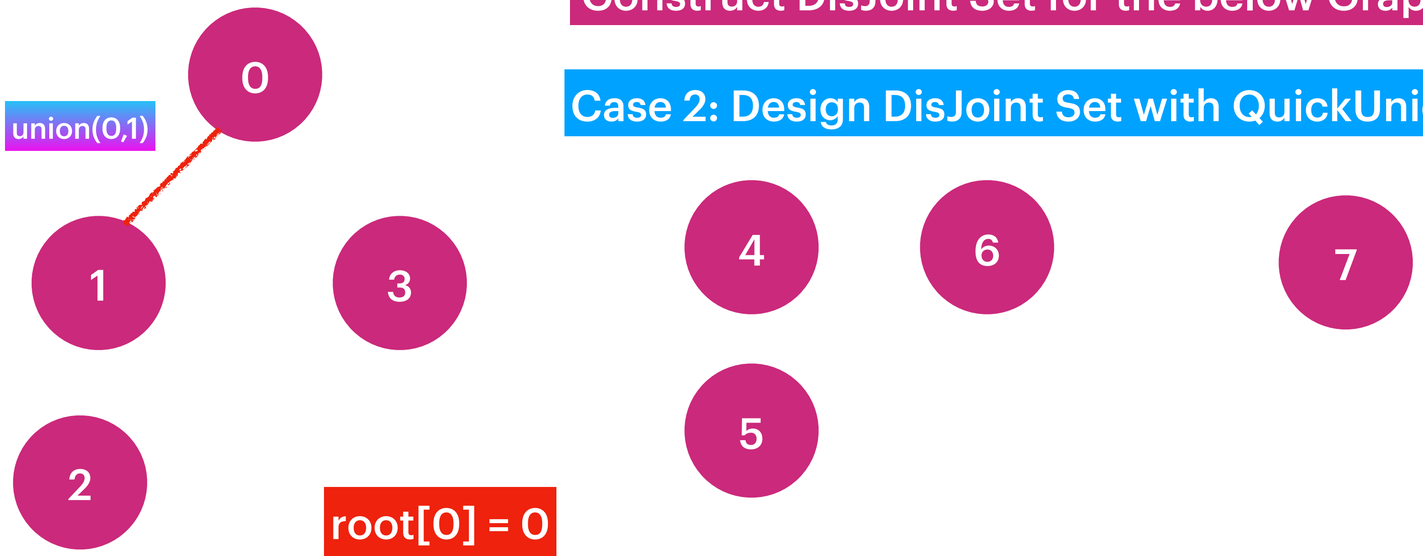
root	0	1	2	3	4	5	6	7
Index's/ Vertices	0	1	2	3	4	5	6	7

Array Index can be represented as Vertex.
Index value can be represented by root.

Construct DisJoint Set for the below Graphs

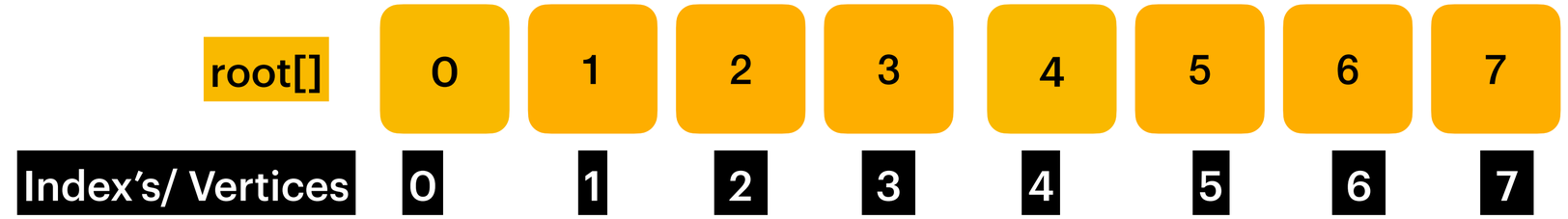
Case 2: Design DisJoint Set with QuickUnion.

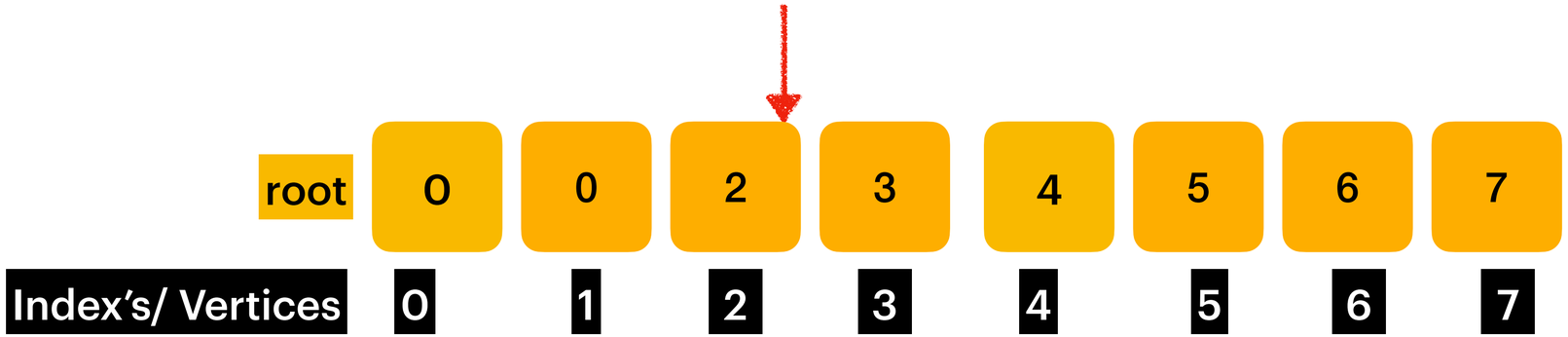
union(0,1)



root[0] = 0
root[1] = 1

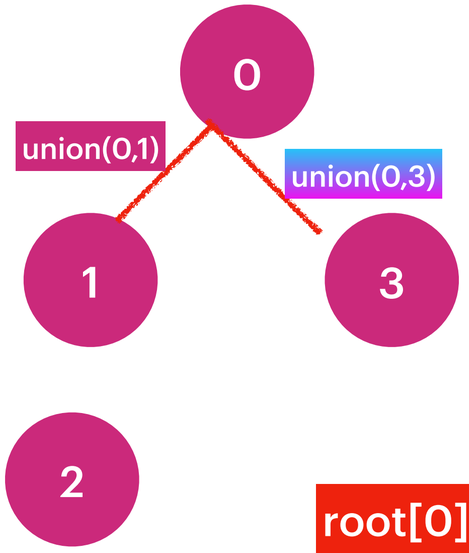
```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```

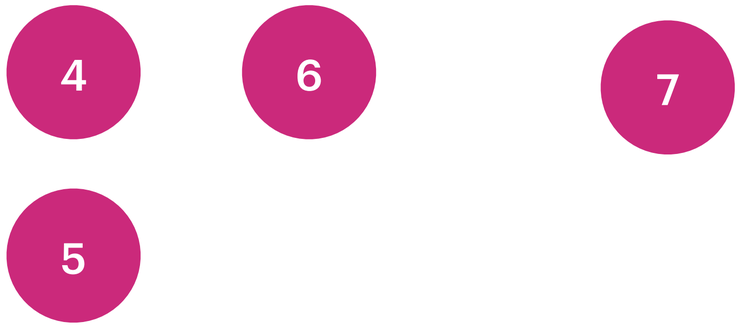




Construct DisJoint Set for the below Graphs

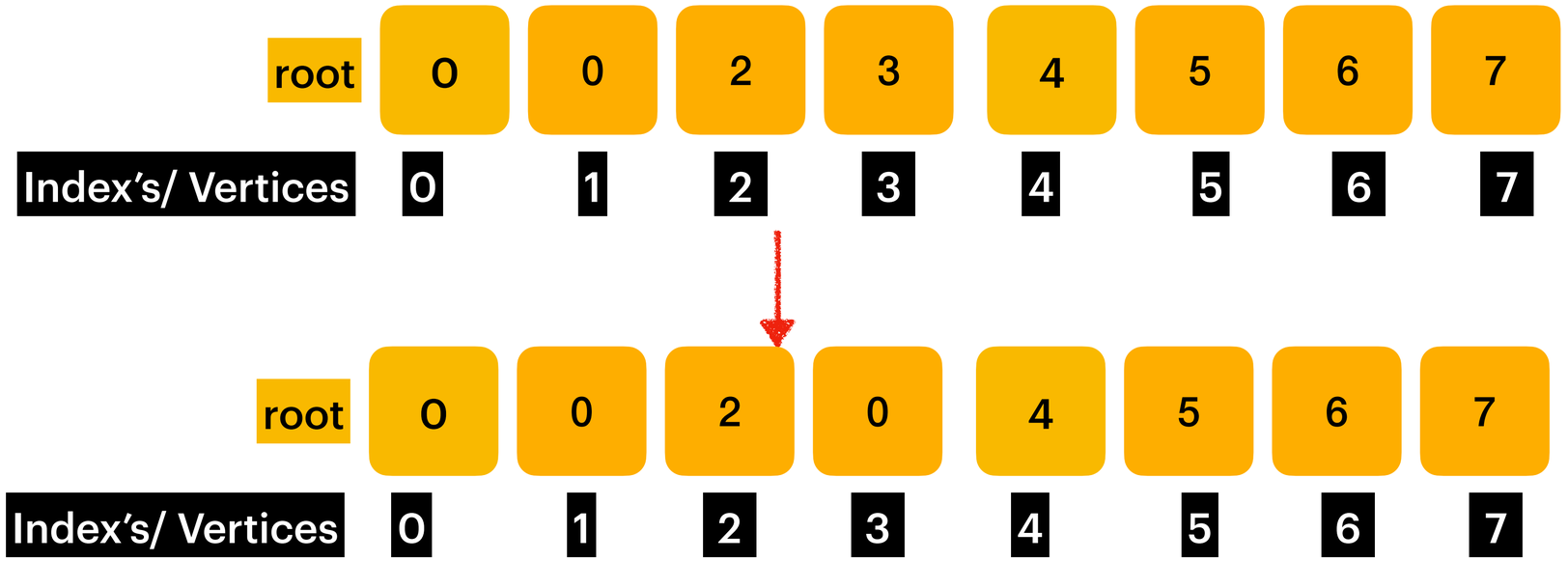
Case 2: Design DisJoint Set with QuickUnion.





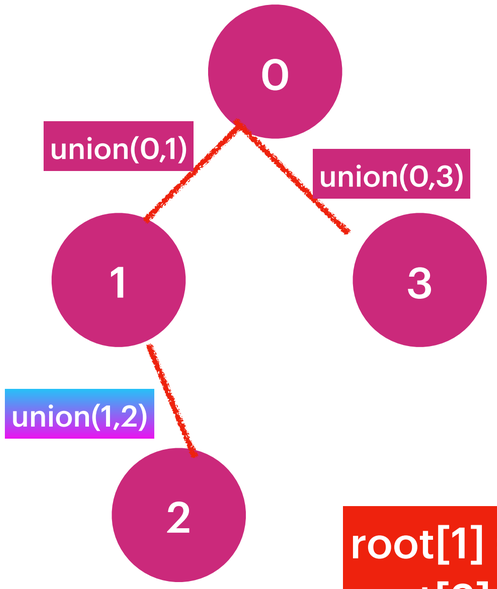

```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```

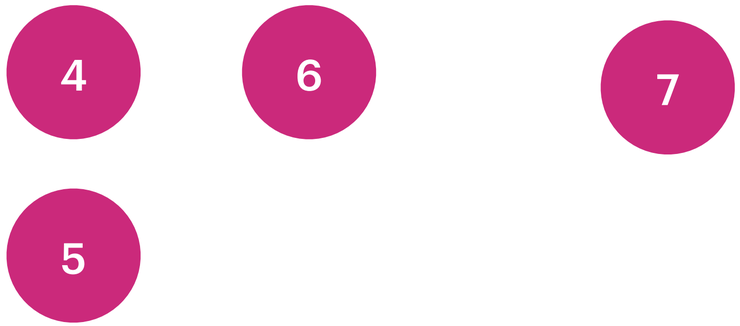
root[0] = 0
root[3] = 3



Construct DisJoint Set for the below Graphs

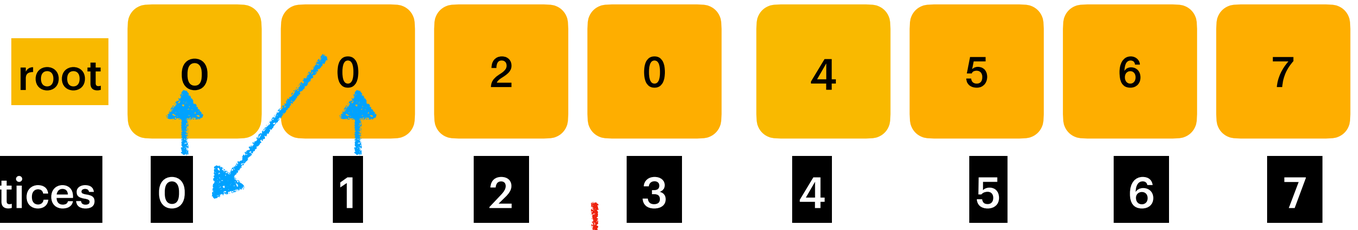
Case 2: Design DisJoint Set with QuickUnion.

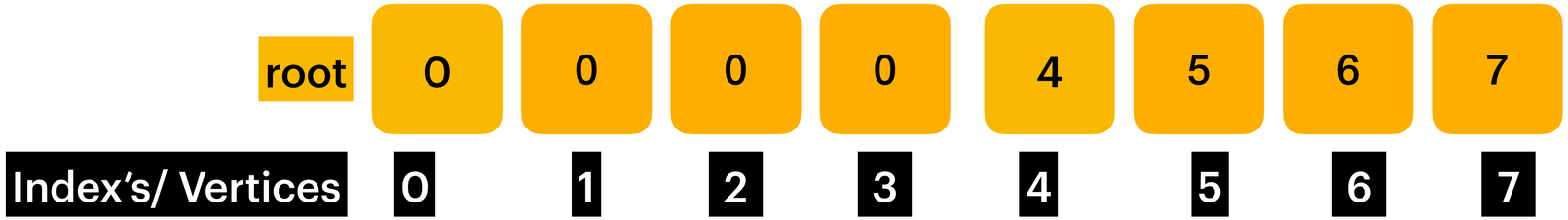





```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```

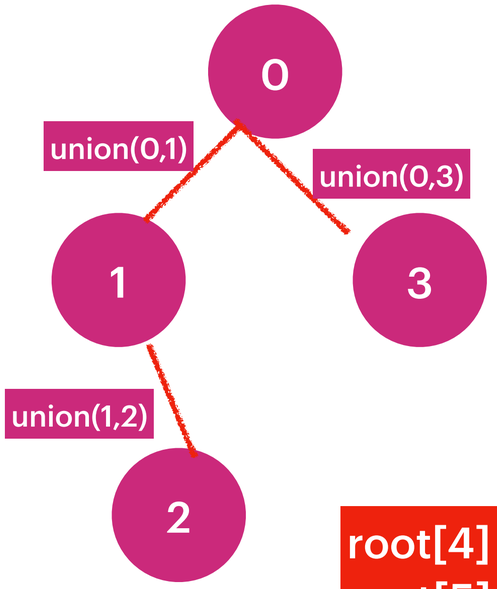
root[1] = 0
root[2] = 2





Construct DisJoint Set for the below Graphs

Case 2: Design DisJoint Set with QuickUnion.




```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```

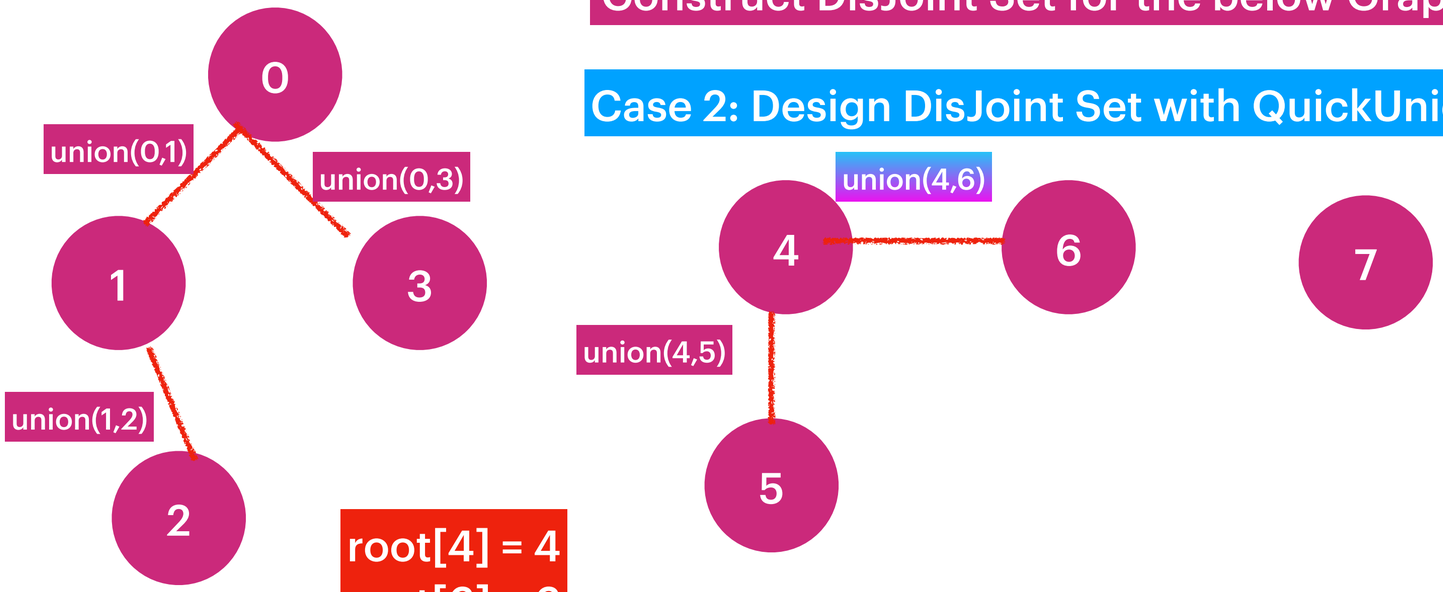
root[4] = 4
root[5] = 5

root	0	0	2	0	4	5	6	7
Index's/ Vertices	0	1	2	3	4	5	6	7

root	0	0	0	0	4	4	6	7
Index's/ Vertices	0	1	2	3	4	5	6	7

Construct DisJoint Set for the below Graphs

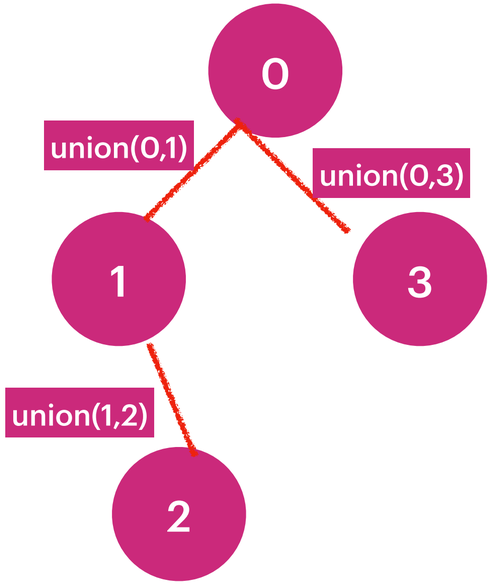
Case 2: Design DisJoint Set with QuickUnion.



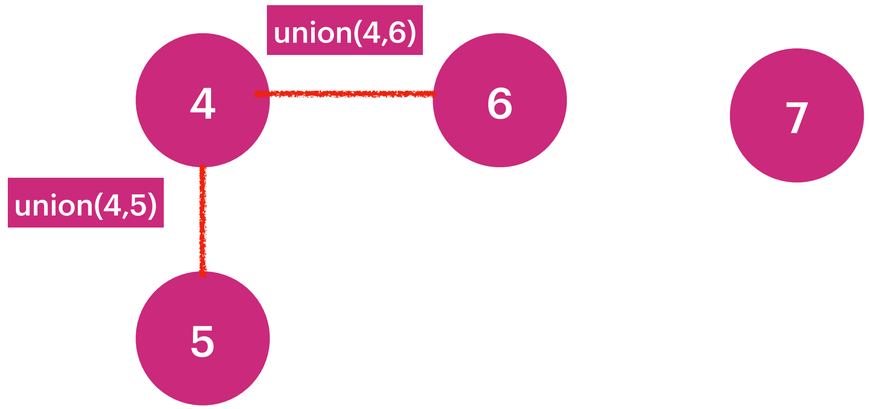
root[4] = 4
root[6] = 6

root	0	0	2	0	4	4	6	7
Index's/ Vertices	0	1	2	3	4	5	6	7

root	0	0	0	0	4	4	4	7
Index's/ Vertices	0	1	2	3	4	5	6	7



Case 2: Design DisJoint Set with QuickUnion.



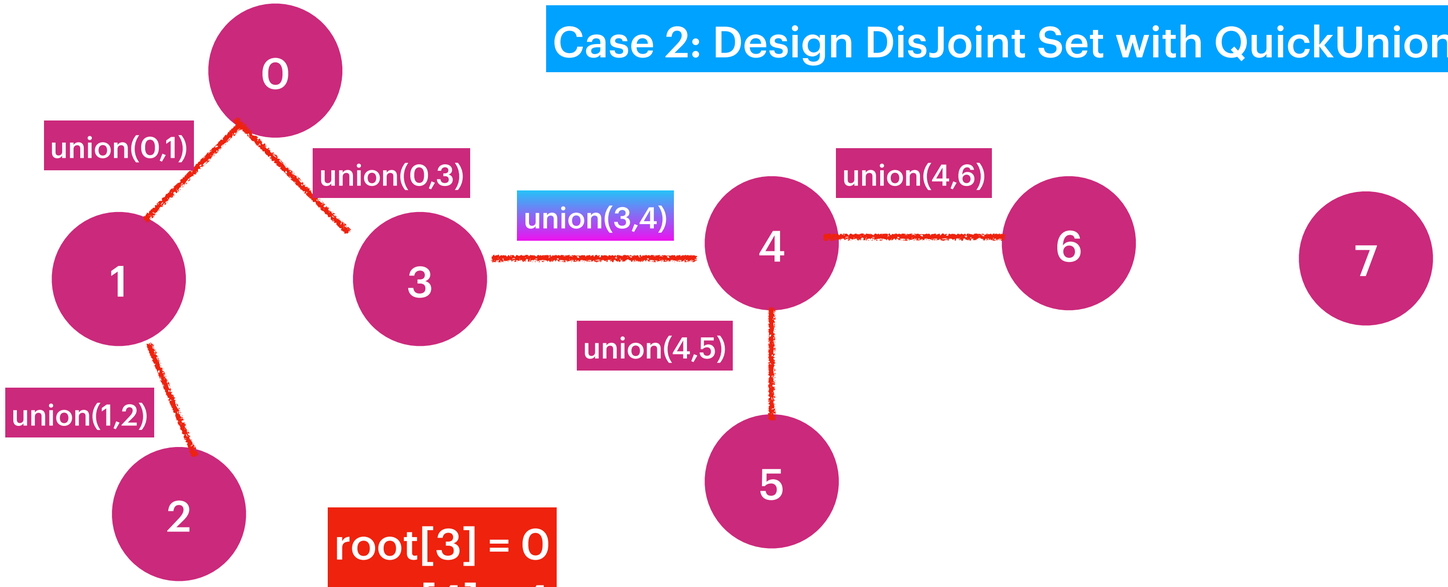
root	0	0	0	0	4	4	4	7
Index's/ Vertices	0	1	2	3	4	5	6	7

```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```

V(2) & V(3) connected. —> True (find(2) == find(3))

V(2) & V(5) are connected. —> False (find(2) != find(5))

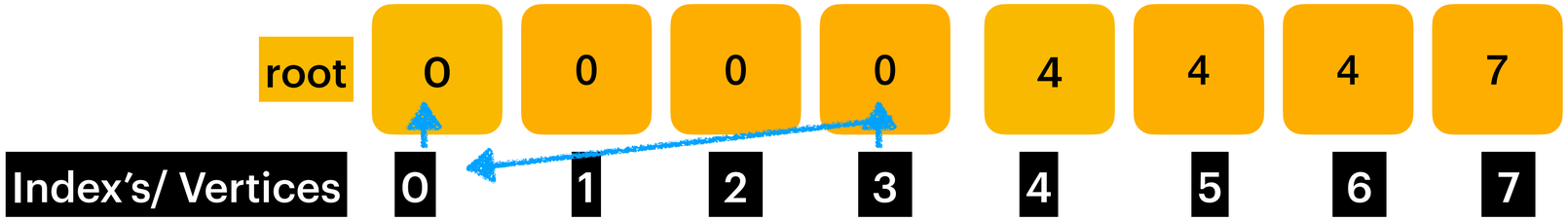
Case 2: Design DisJoint Set with QuickUnion.



Time Complexity : O(n)

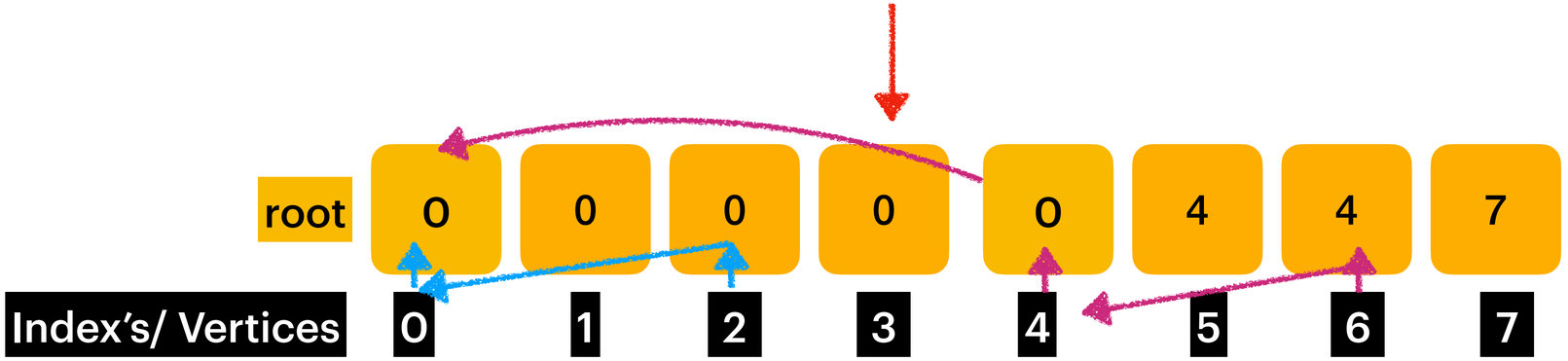
```
public void union(int vx, int vy)
{
int rootX = find(vx);
int rootY = find(vy);

root[rootY] = rootX;
}
```

root[3] = 0
root[4] = 4

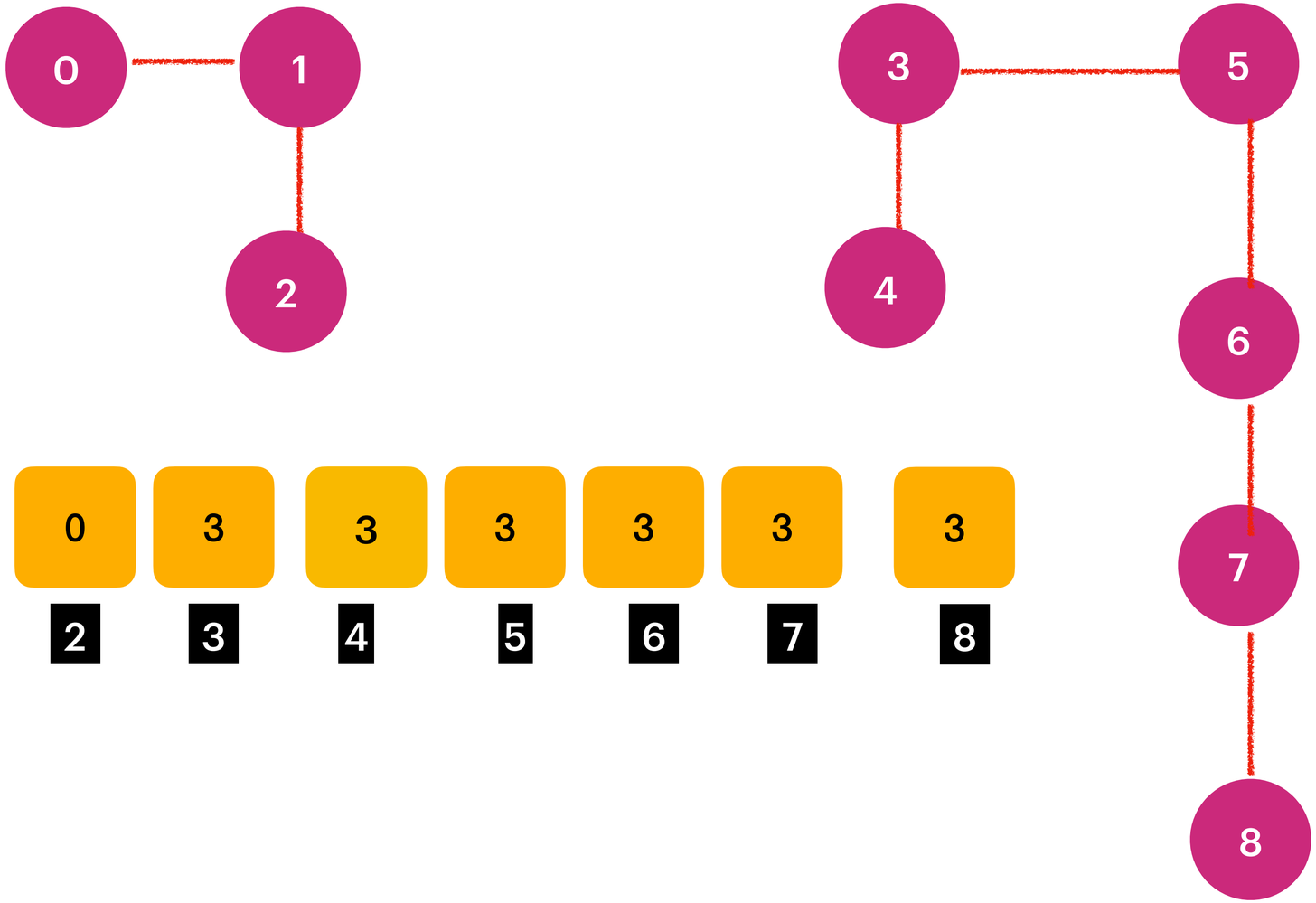


Time Complexity : O(n)

```
public void find(int v)
{
while(v != root[v])
{
V = root[v];
}
return v
}
```



V(2) & V(6) connected. —> True (find(2) == find(6))

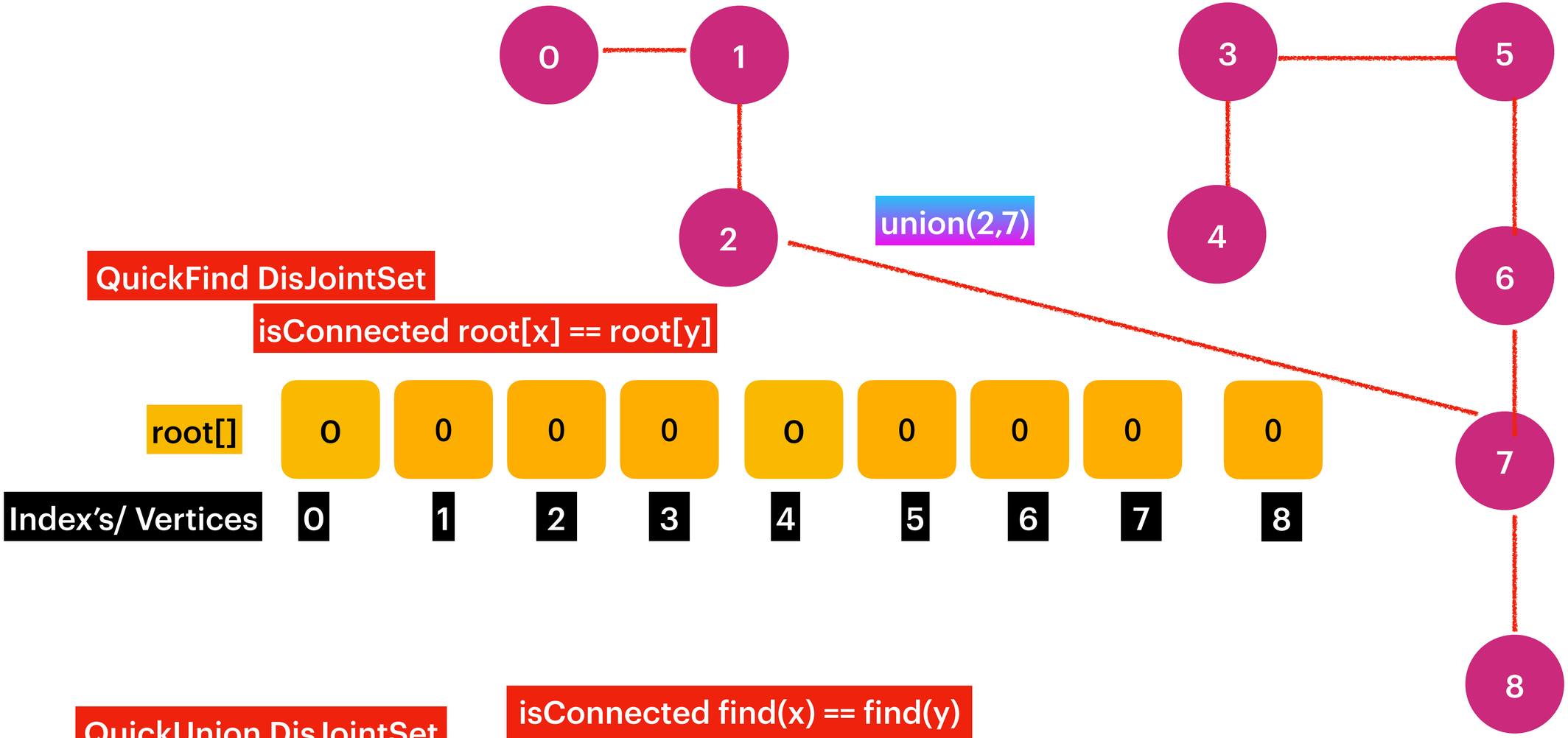


QuickFind DisJointSet

root[]	0	0	0	3	3	3	3	3	3
Index's/ Vertices	0	1	2	3	4	5	6	7	8

QuickUnion DisJointSet

root[]	0	0	0	3	3	3	3	3	3
Index's/ Vertices	0	1	2	3	4	5	6	7	8


0
1
2
3
4
5
6
7
8
union(2,7)
QuickFind DisJointSet
isConnected root[x] == root[y]
root[]
0 0 0 0 0 0 0 0 0
Index's/ Vertices
0 1 2 3 4 5 6 7 8
QuickUnion DisJointSet
isConnected find(x) == find(y)
Ex: find(2) == find(7) —> 0 == 0
root[]
0 0 0 0 3 3 3 3 3
Index's/ Vertices
0 1 2 3 4 5 6 7 8

Case 3: Design DisJoint Set with PathOptimisation.