KRUSKAL'S ALGORITHM.

```
Kruskal(<V, E, w>):
  Sort E so that
    w(e1) <= w(e2) <= ... <= w(e|E|)
  Et ← {}
  count ← 0.
  k ← 0.
  while count < |V| - 1
      k ← k+1
    if Et union {ek} is acyclic
          Et ← Et union {ek}
          count ← count + 1
  return Et.
```

Et — denotes min spanning tree.

V - set of vertices

E - set of edges

w - set of weights

K - keep track of edges.

Count - No. of vertices.

Given a graph with 'n' vertices,

Make disjoint sets.

① Creation of sets.

② Find if it creates a cycle.

③ If it does not result in a cycle - Union.

eg - S = {1, 2, 3, 4, 5}

Create sets - {1}, {2}, {3}, {4}, {5}

↓

Done by the operation "makeset()"

↓

makeset(P). Creates - {P}

makeset returns singleton sets.

Union (1,2) = {1,2}

For every set there is a representative. Representative can be any element in the set. Generally 1st elt is taken as representative.

U(1,2) → representatives of 1 & 2 should be returned.

If representative of 1 & representative of 2 are same, we should not add. Bcz it will result in a cycle.

⇒ "Find" operation return representative of a set.

eg: find{1} → 1

eg: makeset(5)

{1} {2} {3} {4} {5} — Disjoint sets

Union {1,2} Find{1} → 1

{1,2} {3} {4} {5} → representative.

1 — representative

3 — representative

Union {2,3}

→ Representative of 2 is 1

" 3 is 3

Both are not same. So u can perform Union operation.

→ {1,2,3} {4} {5} → representative.

1 — representative of set

4 — representative

Linked List Representation

makeset (1) → [1 |] → NULL

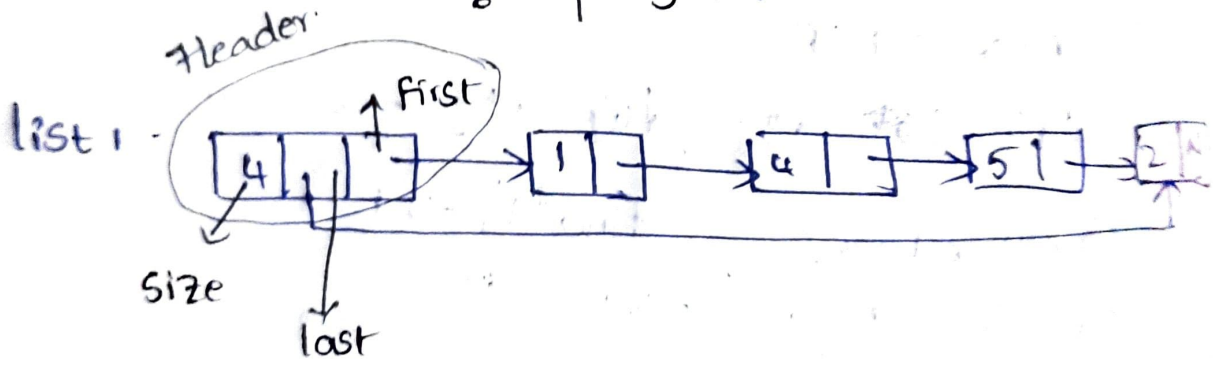
Makeset (2) → [2 |] → NULL

Complexity of makeset : O(1)

Bcz it creates a single element list.

Eg: Subset representatives

element index	Representative
1	1
2	1
3	3
4	1
5	1
6	3



list 2 - [0 | N | N]

List 3 - [2 | 1] → [3] → [6 | N]

list 4 - [0 | N | N]

list 5 - [0 | N | N]

list 6 - [0 | N | N]

Complexity of find (or) search - O(n)

Retrieve the elt - O(1).

Union - Traverse to the end of the list and attach the elt.

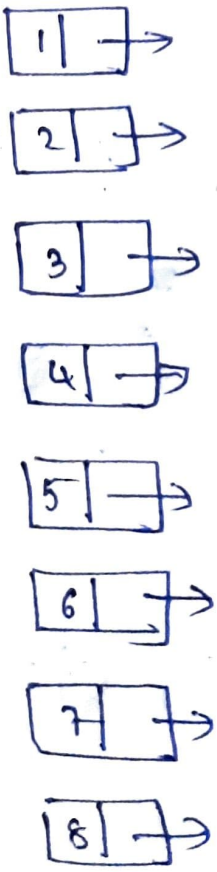
eg:- Union (1,2)

Union (1,3) ⇒ ① ⇒

②

~~Sometimes~~ When we have to append 2 lists compare the sizes. Traverse the smaller list and attach the larger list there.

eg:-



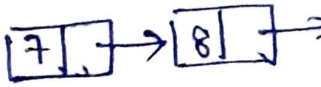
(a) ⇒ Union(1,2) -

1 & 2 singleton sets are deleted.

(b) ⇒ Union (3,4)

(c) ⇒ Union (5,6)

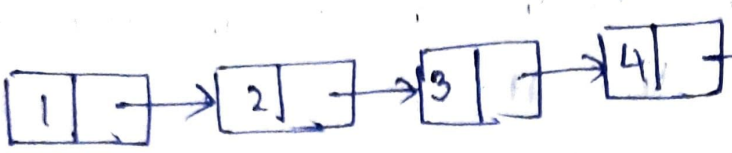
(d) ⇒ Union (7,8)



In (a) rep of 2 is modified to 1

elt	No. of times rep changed
1	0
2	1
3	0
4	1
5	0
6	1
7	0
8	1

⇒ Union (1, 4) - Rep of 1 → -1
Rep of 4 → -3
} Diff
So, union operation can be performed



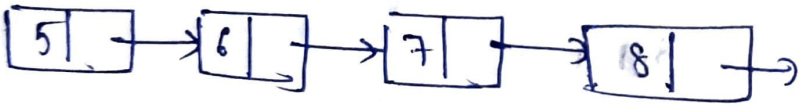
3 & 4 representatives are changed.

elt	1	2	3	4	5	6	7	8
No of times rep changed	0	1	1	2	0	1	0	1

⇒ If no. of elts = 4
representative is updated 2 times.

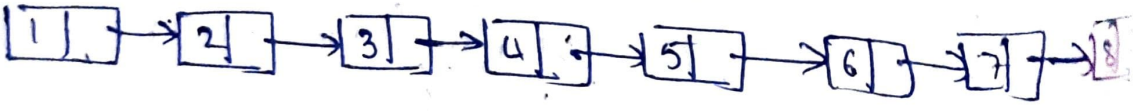
$\log_2 4$

⇒ Union (5 7)



elt	1	2	3	4	5	6	7	8
No. of times rep updated	0	1	1	2	0	1	1	2.

⇒ Union (1, 5)



elt	1	2	3	4	5	6	7	8
No of times rep is updated	0	1	1	2	1	2	2	3.

No. of elts = 8.

Max num of rep can be updated is 3.

So,

No. of elts = n

Max num of rep can be updated is $\log_2 n$

→ Set of union operations → $\log n$

for 'n' union operations → $n \log n$

→ m find operations & (n-1) union operations.

$m(\theta(1)) + (n-1)\log n \approx n\log n + m$

Complexity -

→ Makeset (x) - $\theta(1)$

→ Find (i) - $\theta(1)$ (Quick Find)

→ Union (x, y) - $\theta(n \log n)$

→ For (n-1) unions & m find - $\theta(m + n\log n)$

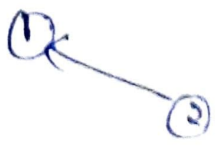
TREE REPRESENTATION

child → parent.

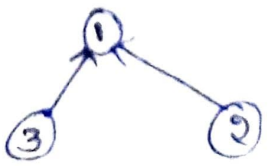
{1} {2} {3}

① ② ③

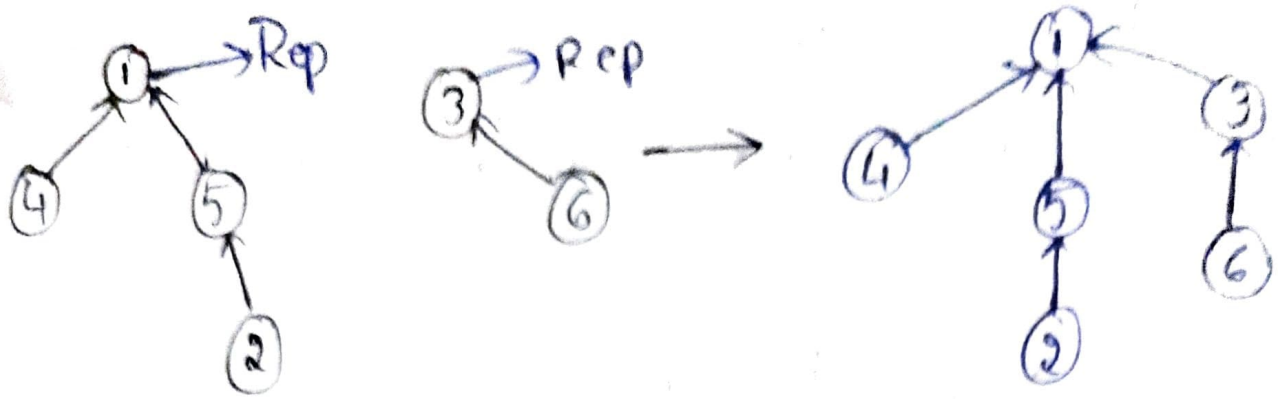
Union (1, 2) -



Union (1, 3) -



Eg



Representative table -

1	2	3	4	5	6
-1	5	-1	1	1	3

If that elt is rep give it as '-1'

Find rep of ②.

Go to index ②.

Rep of 2 is 5.

Go to ⑤. At posⁿ ⑤, value is 1.

At ①, value is -1.

So, rep of ② is 1.

Complexity -

→ Makeset (x) - $O(1)$

→ Find (x) - $O(\log n)$ (max hyt of tree is $\log n$)

→ Union (x, y) - $O(1)$ (Quick union)

→ For (n-1) unions and m finds - $O(n + m \log n)$