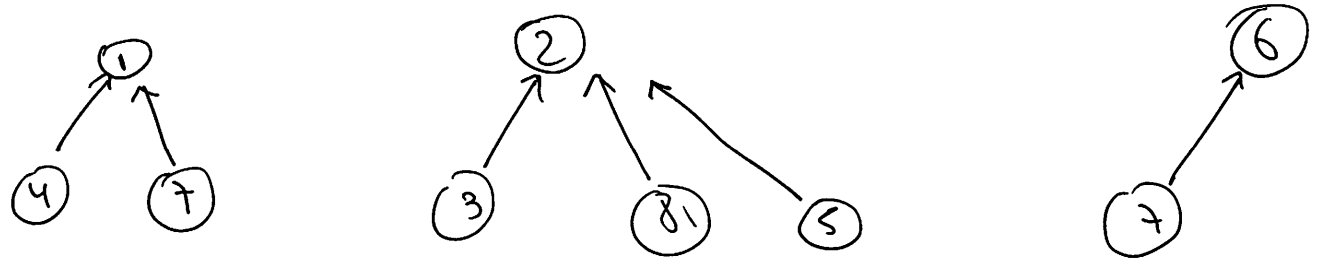
Union-find Data Tree based implementation [GT 7.3]

Represent each component as a star

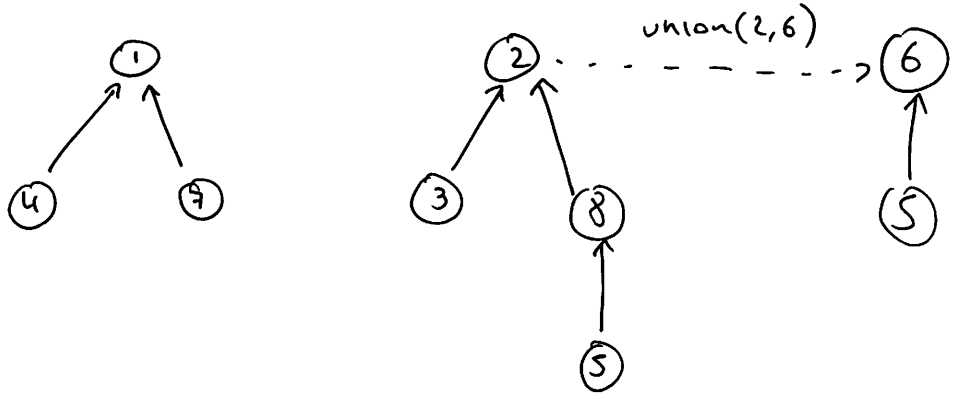


Pointers take $O(n)$ space using an array

- make_set(x) → $O(n)$ time
- find → $O(1)$ time
- union → $O(n)$ time

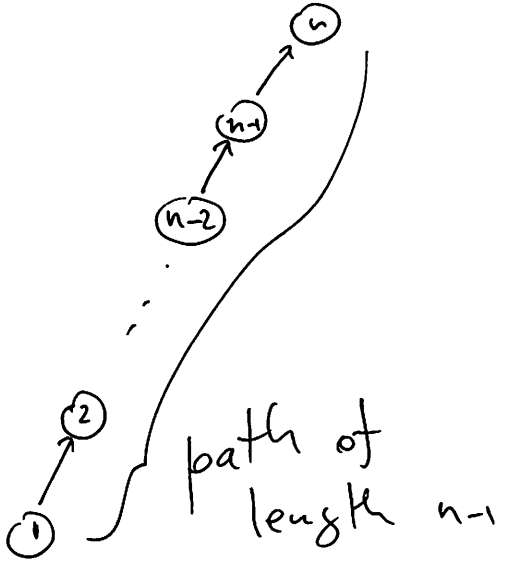
Maybe we can get faster unions at the expense of slower finds

Idea: Do laty unions:



- make set : O(n) time
- find : O(n) time
- union : O(1) time

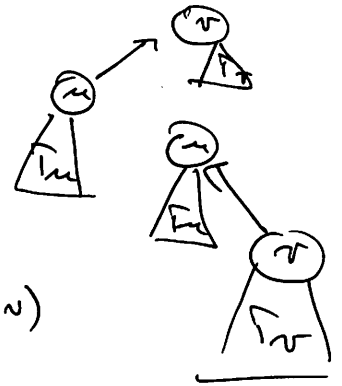
union (1,2)
union (2,3)
⋮
union (n-1,n)
find (1)



Idea: Union by rank

Every root has a rank, union(u,v) →
- if rank(u) < rank(v)
- → if rank(v) ≤ rank(u)

increase rank of new root by 1 if rank(u) = rank(v)

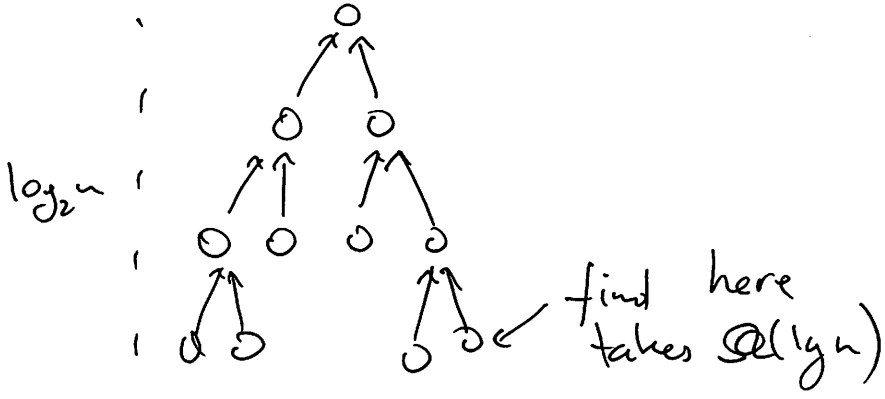


Obs: If T_u is set rooted at u then $|T_u| \geq 2^{\text{rank}(u)}$

Obs: $\text{rank}(u) \leq \log_2 n \quad \forall u$

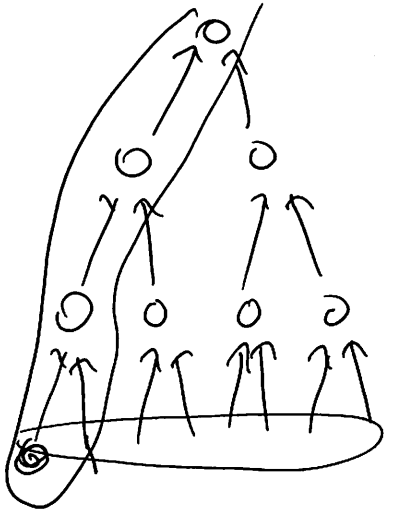
Obs: If $u \neq \text{parent}[u]$ then $\text{rank}[u] < \text{rank}[\text{parent}[u]]$

- make_set (A) → O(n) time // sets rank(u) = 0
- find → O(log n) time
- union → O(1) time



Idea: After find[u] set parent[u] = find(u)
so next time we call find(u) it runs faster

Obs Does not improve amortized running time



perform $\frac{n}{2}$ finds all take $\Omega(\log n)$

$\frac{n}{2} \log n$

Idea: Use path compression: ~~update~~ parent[v] = find(u) for every vertex v on path from v to u

Obs: At least on this example $\frac{n}{2}$ finds on leaves take $\Theta(n)$ time overall

Fact: m find operations and n union ops take $O(m\alpha(n))$ Ackerman function very very slow growing function