ADS Assignment - 2

GOODLUCK | Page No.
Date / /

Name: Shruti Deepak Dhumne
Rollno : 227 PRN : 0220200161
Seatno : S204156 E1 - Batch

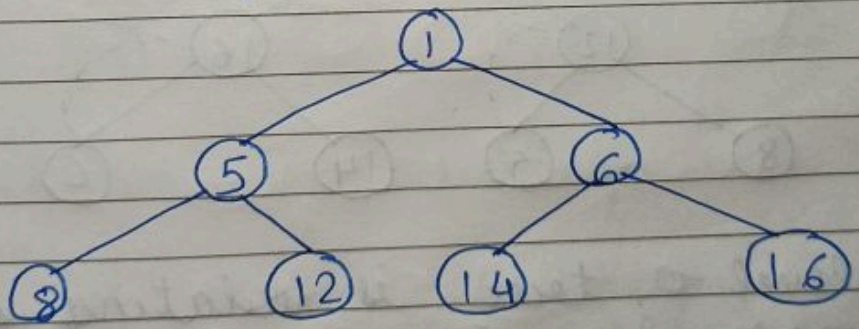
Q1. Construct a max heap for the given array of elements :- 1, 5, 6, 8, 12, 14, 16

→

As the given elements are of array so lets represent in an array.

	0	1	2	3	4	5	6
A	1	5	6	8	12	14	16

- We know heap basics, such as
i) A [(i-1)/2] returns parent node
ii) A [(i * 2)+1] returns left child
iii) A [(i * 2)+2] returns right child
directly we can construct heap from given array



- Constructed heap is of type min heap we need to convert it in max heap.
- To transfer given min heap to max heap
HEAPIFY (A, 9)
- If (A[i]) is violating heap rules
- bring its child to the position of A[i] (which is maximum one).

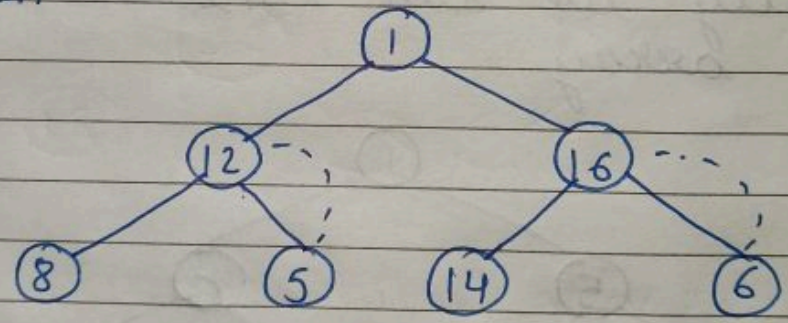
Date

* Procedure for building heap:
- Use the procedure HEAPIFY in bottom up manner to convert an array into heap
- Since, the elements in the sub array are all ~~leaves of~~ leafs of tree, each is 1-element heap to begin with.
- The procedure build heap goes through the remaining nodes of the tree and runs HEAPIFY on each one.

1 As stated going from bottom-up;

i) At level-1 both sub trees are violating properties of max heap [5<8 & 12; 6<14 & 16]

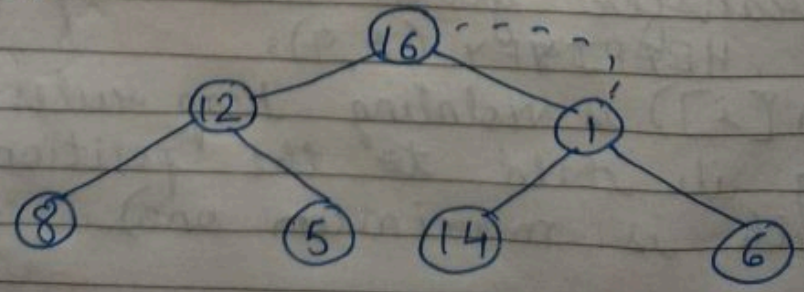
ii) Moving maximum children to parent's position



After moving

2 At level-0, tree is violating property of max heap [1<12 & 16]

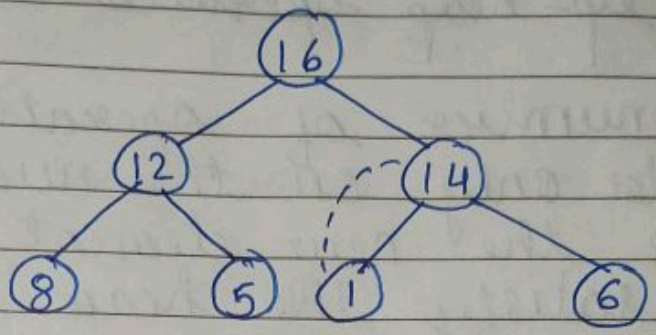
i) Moving maximum children to parent's position



After moving

3 again at level-1, righ sub-tree (heap) is violating property of max heap.

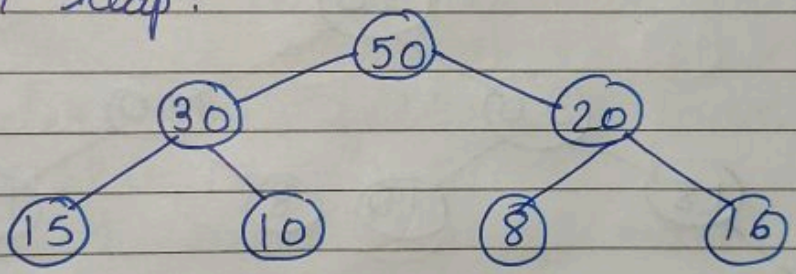
i) Move maximum children to parent position



Final max - heap is ready.

Q2. consider the following max heap 50, 30, 20, 15, 10, 8, 16 insert new node 60.

→ given heap:



* Insert operation on heap:

1 Add an element to heap, we can perform this algorithm:

i) Add the element to the bottom level of the heap at the leftmost open space

ii) Compare the added element with the parent, if they are in correct order, stop

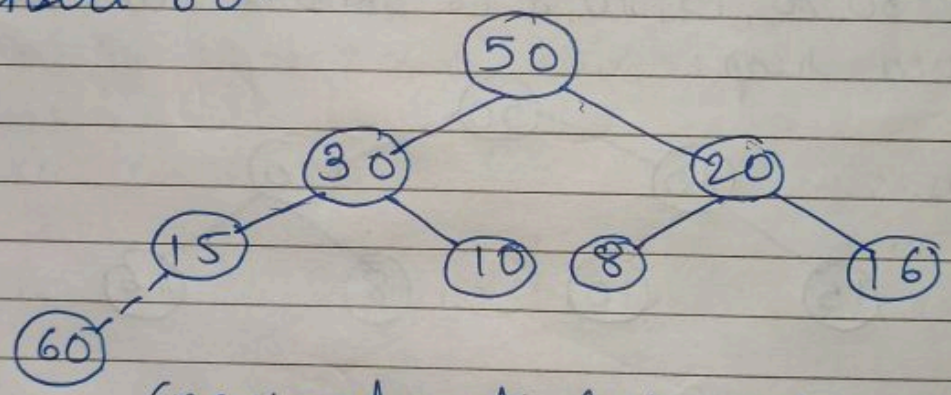
iii) If not, swap the elements with its parent and return to previous step.

2&3 Which restore the heap property by comparing and possibly swapping a node with its parent are called the up-heap operation.

4 The number of operations required depends only on the number of levels the new element must rise to satisfy the heap property.

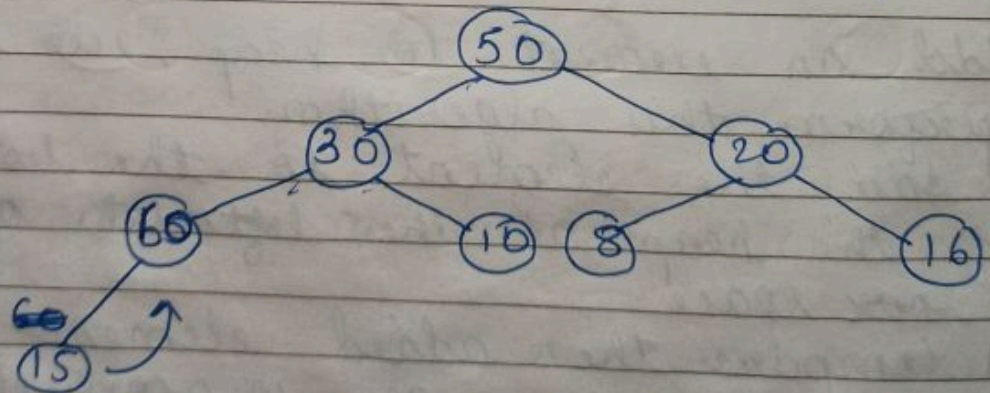
Worst case :- O log (n)
Average case :- O log (1)

1 Insert 60.



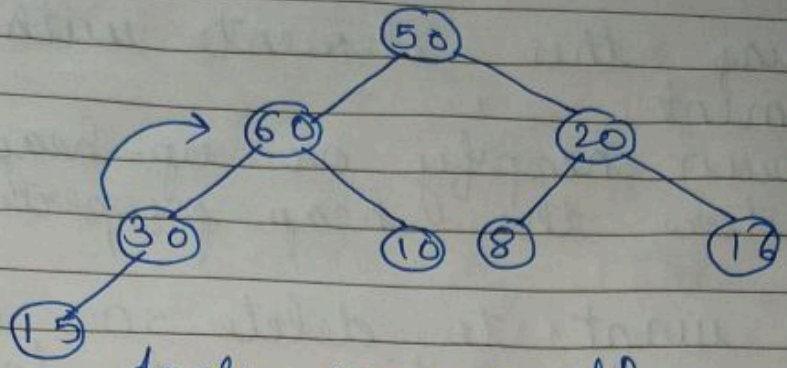
(new node at leftmost open source)

2



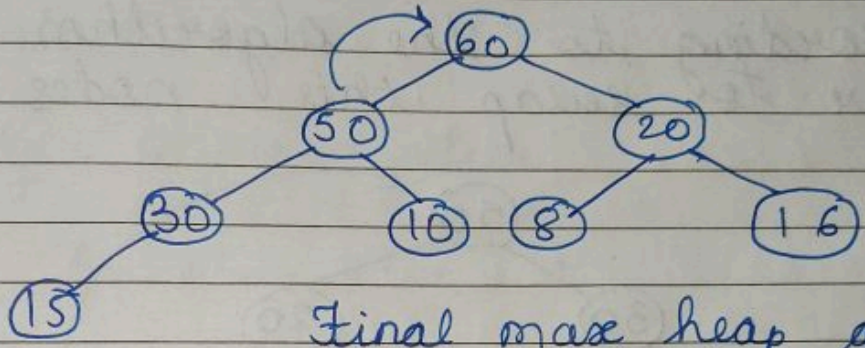
For max heap, swap greater child node with smaller & parent node.

3.



Apply same cond^n as used in ②

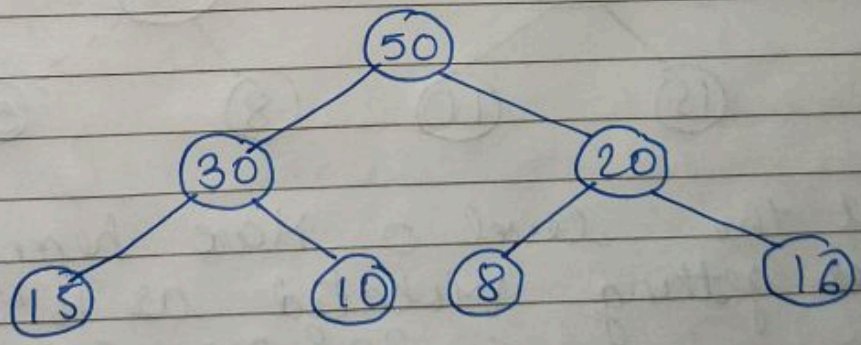
4



Final max heap after insertion

Q3. Consider the following max heap:
50, 30, 20, 15, 10, 8, 16
Delete a node with value 50

→ Given max heap:

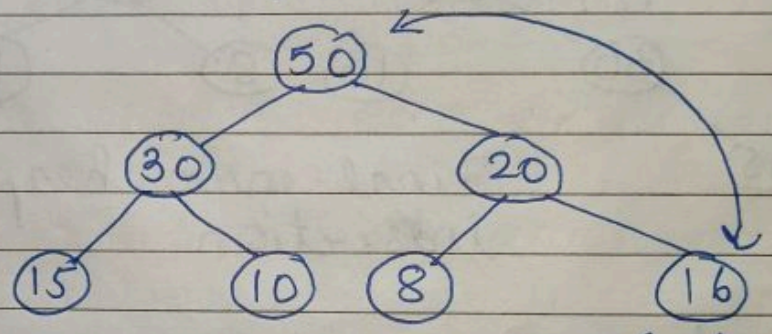


* Delete operation:
- Deleting an arbitary element can be done as follows:

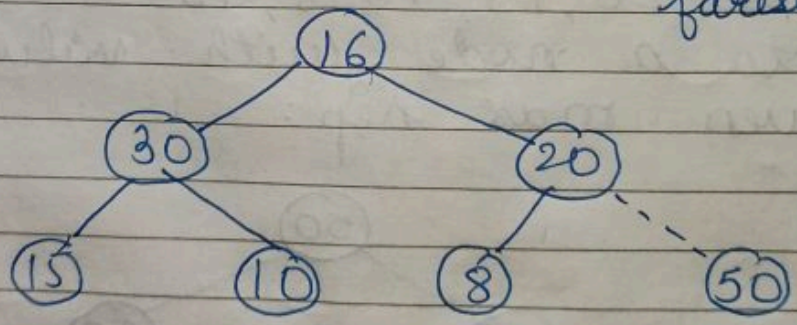
i) Find the index of the element ~~can be done~~ as we want to delete

II Swap this element with the last element.
III Down heapify or up-heapify to restore the heap property.

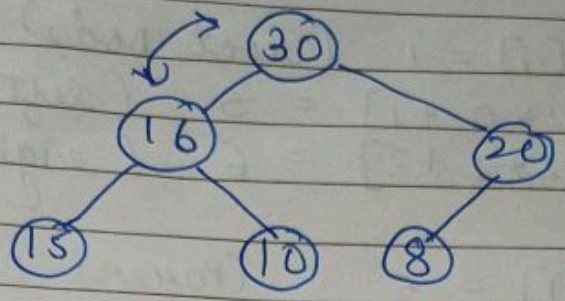
- We want to delete 50 which is at root position and last element is 16.
- According to the algorithm we have to swap this nodes.



last node / farest node



- At the level 0 Max heap property is getting violated as 16 (Parent) < 30 & 20 (child)
- To recover the max heap property we need to swap 30 & 16.



Max heap final

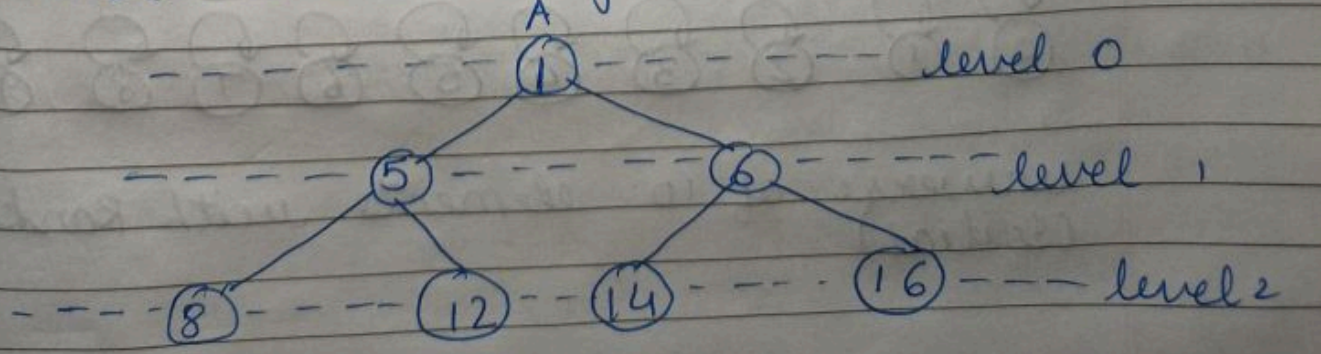
84. Construct a min heap for the given array of elements.
1, 5, 6, 8, 12, 14, 16

→ heap representation

	0	1	2	3	4	5	6
A	1	5	6	8	12	14	16

* Min heap:
- In a min-heap the key present at the root node must be minimum among the keys present at all of its children.
- The same property must be recursively true for all sub-trees in that binary tree.

A[i/2] : Parent node
A[(2*i)+1] : Left child node
A[(2*i)+2] : Right child node



Date

1 $i=0$; $A[i] = 1$ (root-node)
$A[2*0+1] = 5$ (left child-node)
$A[2*2+2] = 6$ (right child-node)

2 $i=1$; $A[1] = 5$ (Parent node)
$A[2*1+1] = 8$ (left child-node)
$A[2*1+2] = 12$ (right child-node)

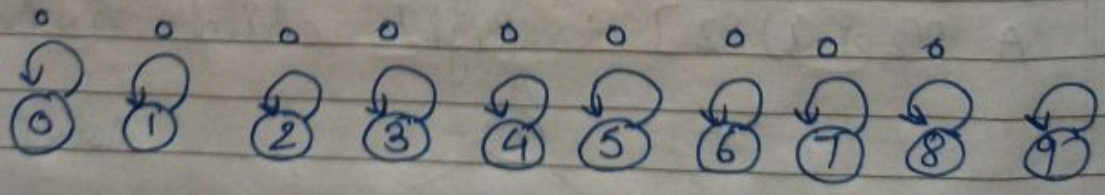
3 $i=2$; $A[2] = 6$ (Parent node)
$A[2*2+1] = 14$ (left child-node)
$A[2*2+2] = 16$ (right child-node)

Q5 Perform the following union-by-rank operations on a universe of 10 elements (0-9, each initially in their own set)

→

Draw the forest of trees that result union(1,5), union(3,7), union(1,3,4) union(5,7), union(0,8), union(6,9) union(3,9)

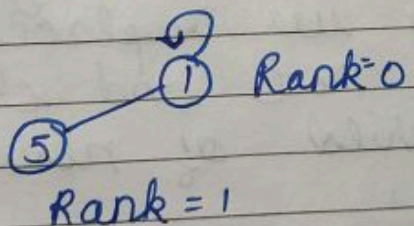
- If set have equal weight, use the root with the smaller value as the root of the new set.



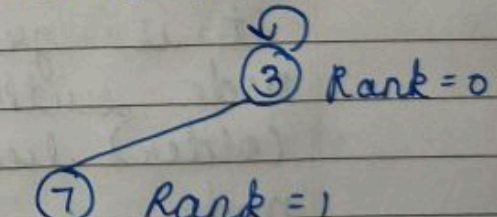
universe of 10 elements with Ranks (Initial)

* Union by rank:
- It always attaches the shorter tree to the root of the taller tree.
- To implement union by rank, each element is associated with a rank
- Initially a set has one element and a rank of zero.
- If we union two sets and both trees have the same rank; the resulting set's rank is one larger
- Both trees have the different ranks - the resulting set's rank is the larger of the two.
- Ranks are used instead of height or depth because path compression will change the tree's height over time.
- Worst case complexity: O log (N)

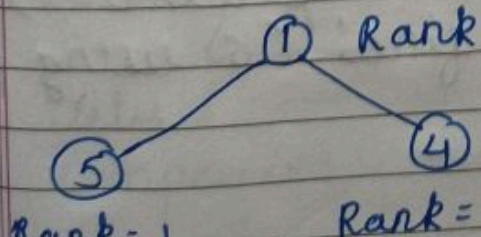
1] Union (1, 5)



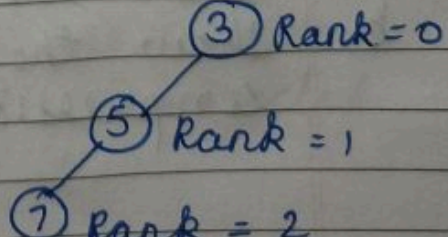
2] Union (3, 7)



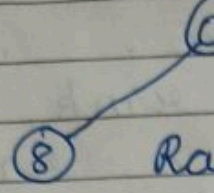
3] Union (1, 4)



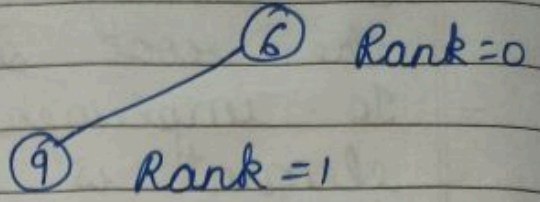
4] Union (5, 7)



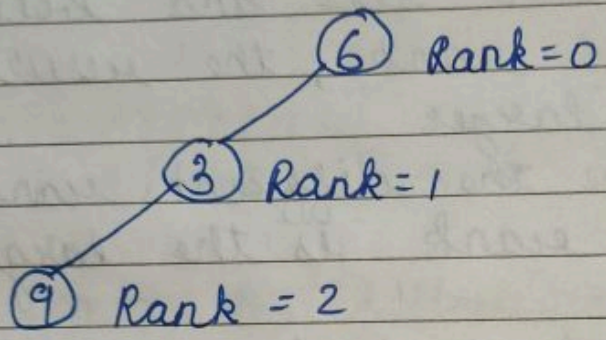
5] Union (0,8)



6] Union (6,9)

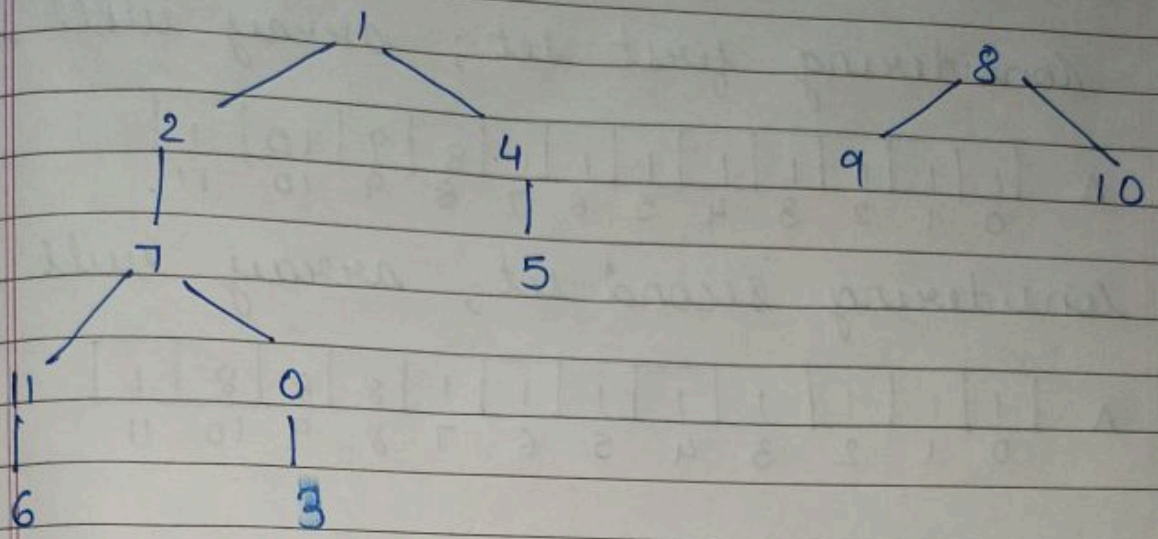


7] Union (3,9)



- In the above forest, inital rank of individual sets was (0)
- we kept them as parent and the child node's rank increased by one as given in the statment.
- Whenever, we were inserting new node it is greater so we replaced new node with child node and child node (older) become child of newly inserted node.

Q6. Given the following forest of up-trees,
a Show the array which represents them
b Show the result of find (6) using path compression.



→a In the above forest, range between 0-11, are 12 elements, so we, need to create array of size 12.

A :	0	1	2	3	4	5	6	7	8	9	10	11

Above disjoin sets represents sets:
first has elements: {0, 1, 2, 3, 4, 5, 6, 7, 11}
second has elements: {8, 9, 10}

- All the subsets are said to be ~~tom~~ connected components
- One can also relate these elements with nodes of a graph.
- The elements in one subset can be considered as the nodes of the graph which are connected to each other directly or indirectly. Therefore, each subset can be considered as connected component.

considering first set; array will be

A	1	1	1	1	1	1	1	1	8	9	10	1
	0	1	2	3	4	5	6	7	8	9	10	11

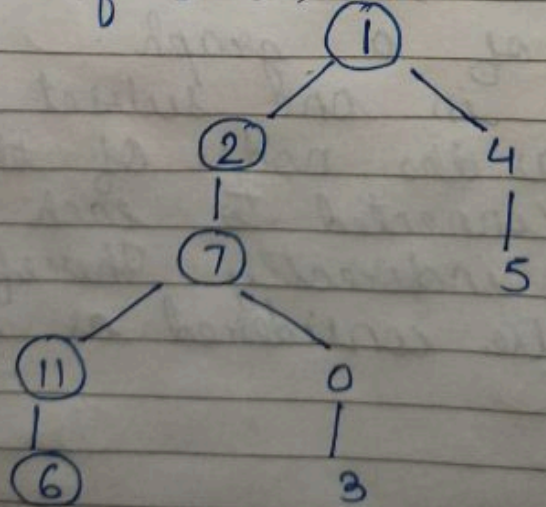
considering second set; array will be

A	1	1	1	1	1	1	1	1	8	8	8	1
	0	1	2	3	4	5	6	7	8	9	10	11

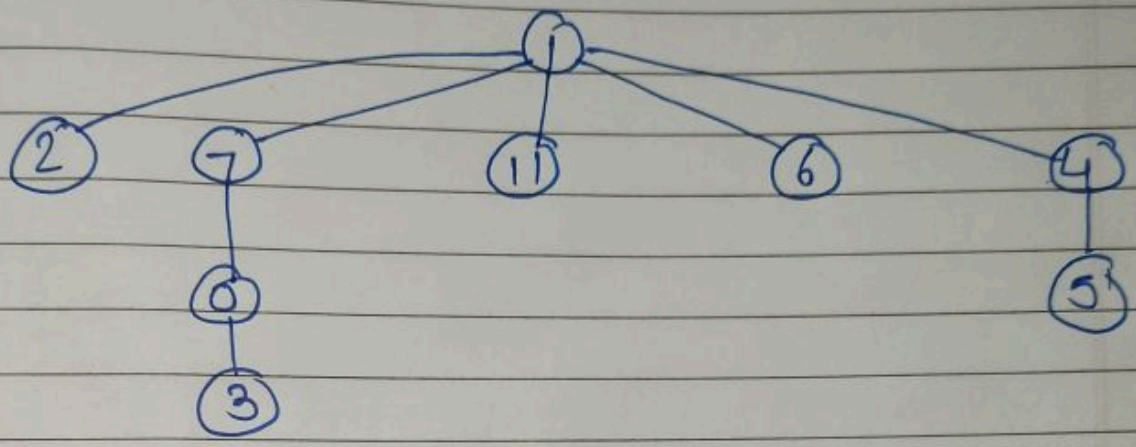
b →

Path compression:

- Path compression is a way of ~~fa~~ flattening the structure of the tree whenever find is used on it.
- Since, each element visited on the way to a root is part of the same set.
- All of these visited elements can be reattached directly to the root.
- The resulting tree is much flatter, speeding up future operations not only on these elements but also on those referencing them.
- result find (6)



- After calling find(6) tree is compressed which shortens the paths for the visited nodes 6, 11, ~~12~~, 7, 2



Compressed tree after calling find(6)