Fibonacci & Binomial Heaps

~~Fib~~

Fibonacci Heaps:

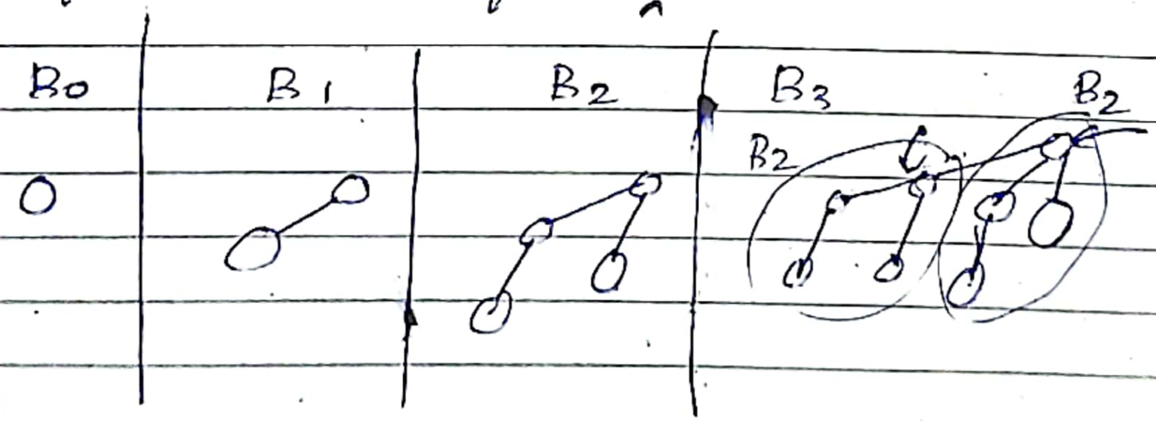
→ A collection of trees with each tree foll. the heap order property.

→ Trees may be in any order in the root list

Binomial Tree

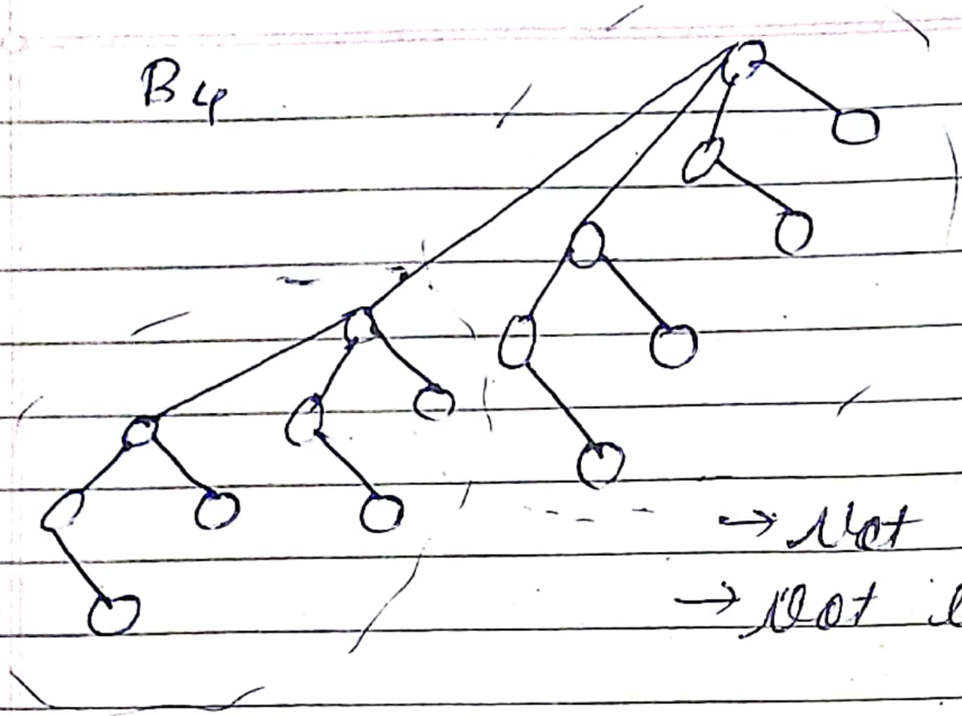
B_k: Binomial tree with order k.

- B_k has k children at root
- B_k can be formed using 2 B_{k-1} trees
- The root of ~~2~~ B_{k-1} will be the leftmost child of the root of other B_{k-1}



- There are 2^k nodes in a tree with order k

B_4



→ Not balanced
→ Not binary

Properties of ~~Binary~~ Binomial Trees:

1) There are 2^k nodes in a Binomial tree B_k

2) The ht. of the B_k tree is k

3) There are exactly ${}^kC_i$ nodes at depth i in Binomial tree B_k

4) The max. ~~depth~~ degree of any node in a n node binomials tree is $\log(n)$

Fibonacci Heap:

→ Collection of trees satisfying min-heap prop.

→ ~~Main~~ Maintain pointer to minimum element

→ Trees may be in any order in the root list

→ Siblings are ~~conn~~ connected through a circular doubly linked list

→ Each child points to its parent

→ Each parent points to any one child.

→ degree (node): No. of children of a node of a tree

→ ~~mark (x):~~

→ mark (node): 1 → lost one of its child
0 → -"- none -"-

~~Lar~~ Lazy approach (Fibonacci Heap)

→ Insert: Simply add to the list & update minimum if needed.

→ Union: Simply combine lists using pointers & update the minimum

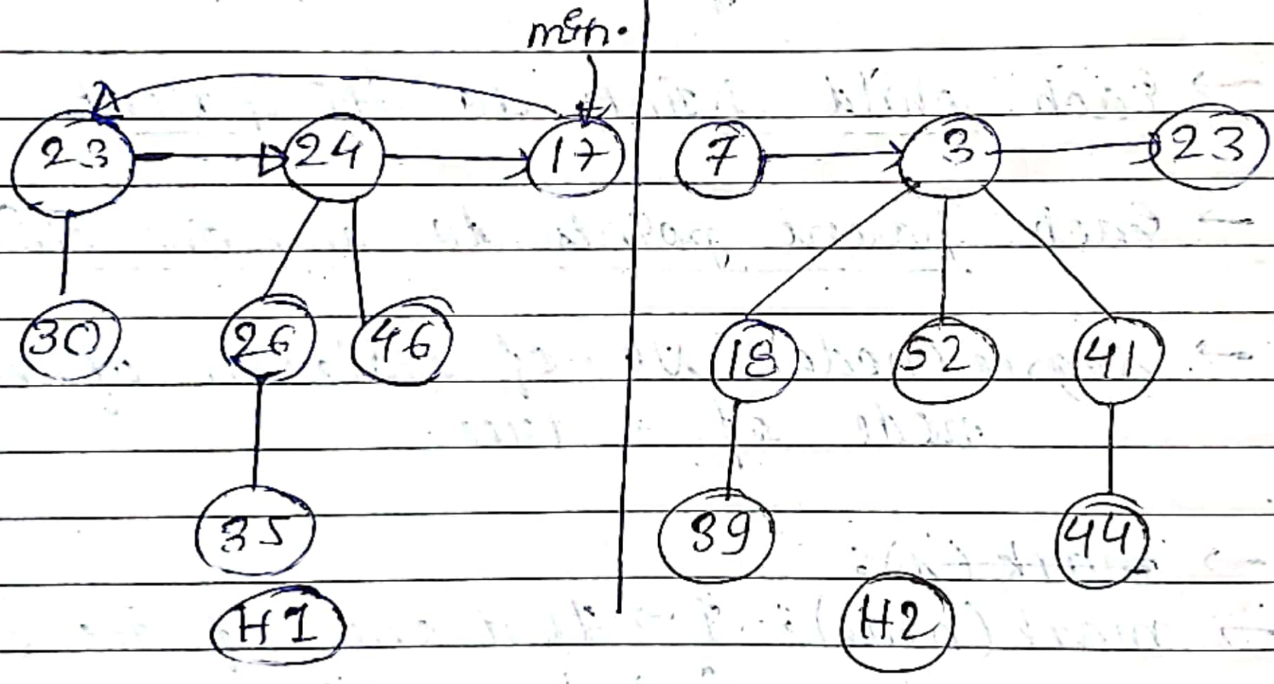
→ Delete : Delete the min., merge lists & consolidate

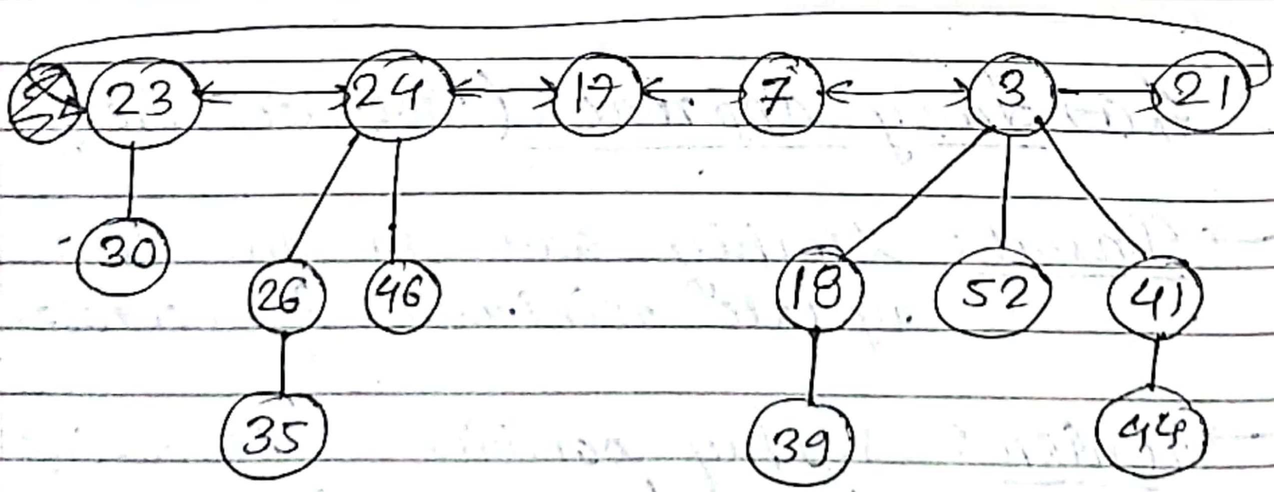
Union:

→ Concatenate the root list of H1 & H2 onto a new root list H

→ Set min. of H

→ Set n(H) equal to total no. of nodes





Decrease Key

Decrease Key (node)

Case-I : When parent[node] < node : ~~Cut~~
→ Decrease value

Case-II : When parent[node] > node and

parent is unmarked
→ Decrease value
→ Cut node & add to root list
→ Mark the parent

Case-III : When parent[node] < ~~node~~ node and

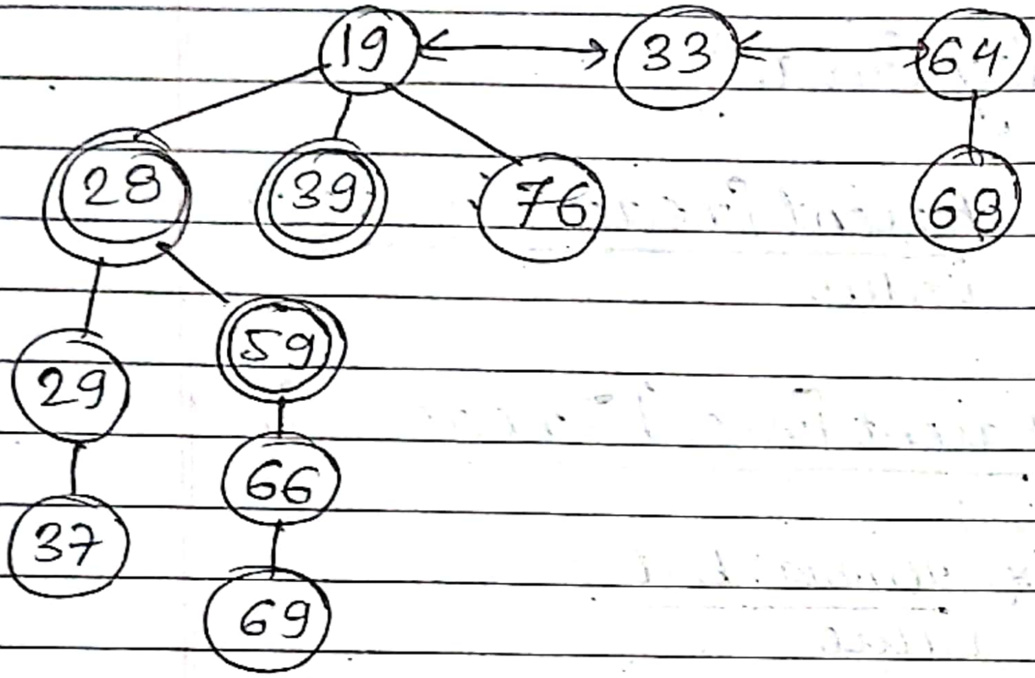
parent[node] is marked

→ Cut node & add to root list

→ do {
→ Cut & add parent to root list
→ Unmark the parent
} while (parent is unmarked OR reached root list)

E.g.

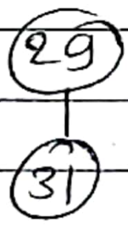
Eg



Op 1: Dec. 37 to 31.

~~Cons: only this sub this subtree.~~

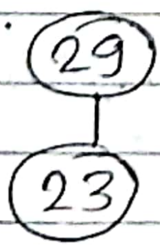
Case-I



As parent (29) is less than 31 min-heap prop. isn't violated & the heap remains as it is

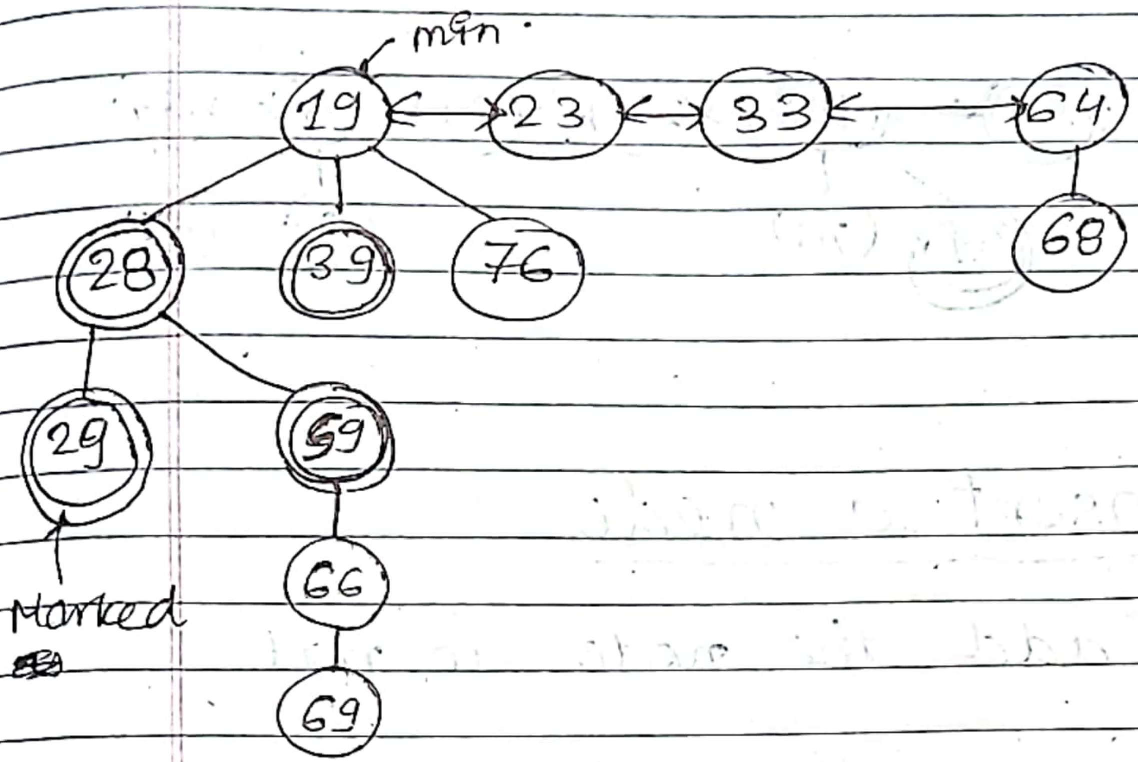
Case-II

Dec. 31 to 23

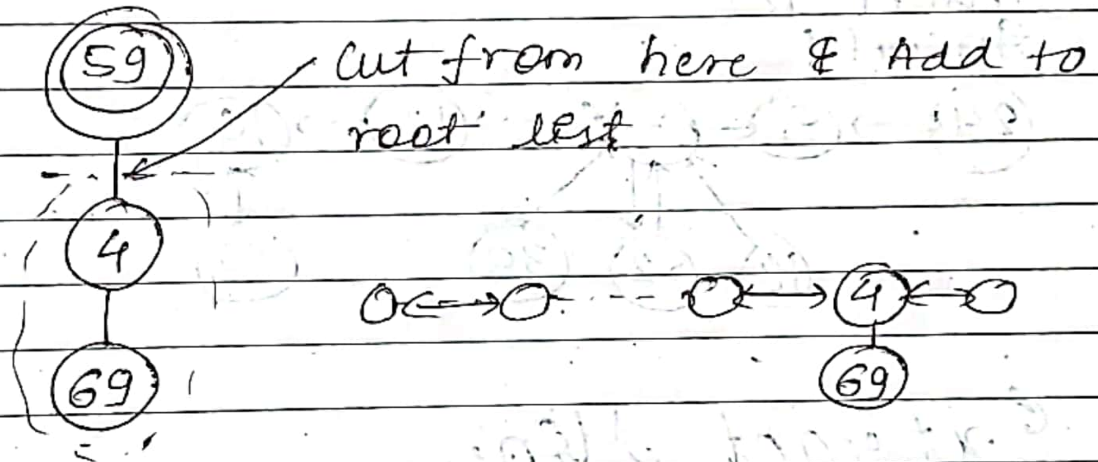


→ Min-heap prop. is violated
→ ∴ Cut 23 & add to root list

View now:



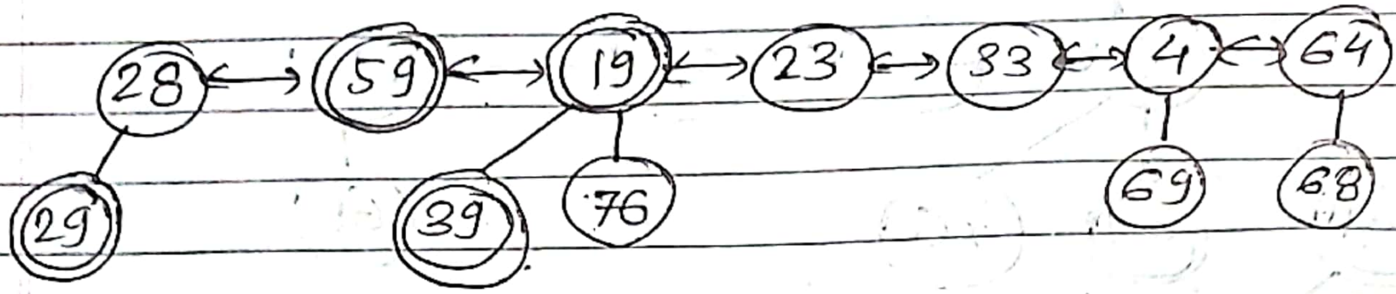
Case-3: Dec. 66 to 4



Now as 59 is marked, cut & add 59 to root list

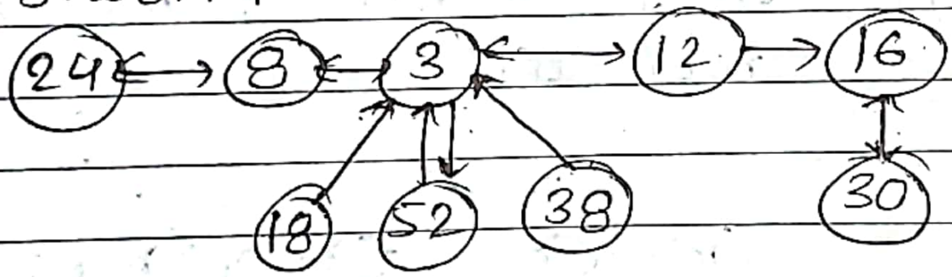
Now 28 is also marked so cut & add to root list

Final view:



~~In~~ Insert a node

→ Just add the node to root list
→ Update min. if reqd.
→ Change pointers
→ Insert (12)
~~Insert 1~~



Extract Min.

→ 1) Delete the min. node

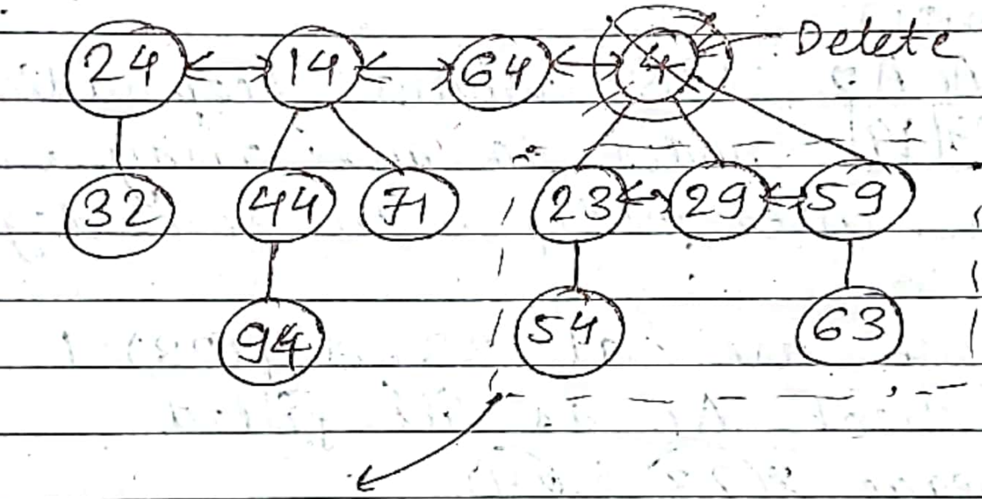
2) Join the root list of descendants of deleted node to the fibo. heap.

3) Go from left to right in the ~~heap~~ root list

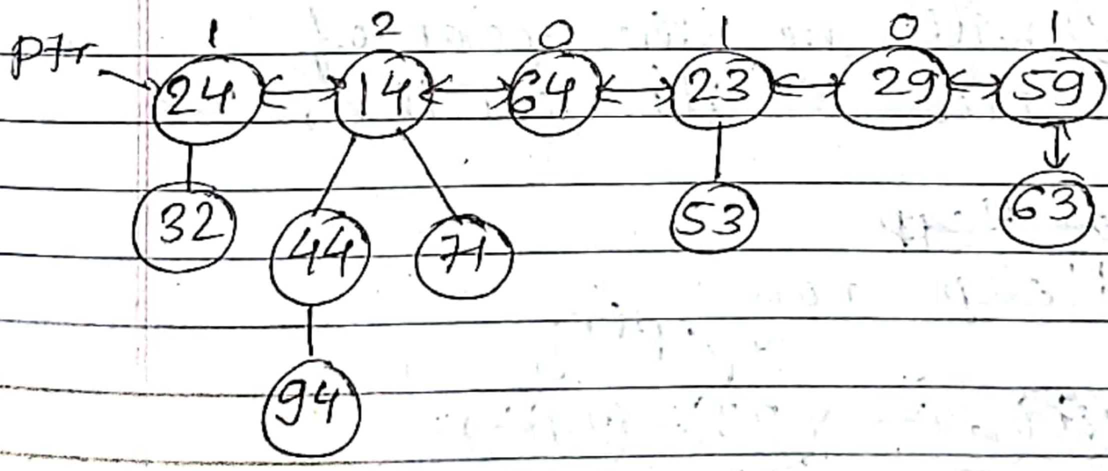
- Find new min.
- Merge trees with same degree

E.g.
Extract min() → i.e. here 4

S-1:



Join with root list



Now we will store the degrees of nodes of root list in auxiliary array

Auxiliary Array:

Size of array: D(n) [Maximum degree among all the nodes of a Fibonacci tree]

A0 A1 A2 ......

Now we keep a ptr. at left-most elem. of the root list & add the deg. of node to aux. array.

~~Array~~ Array now:

A0	A1	A2
64	24	14

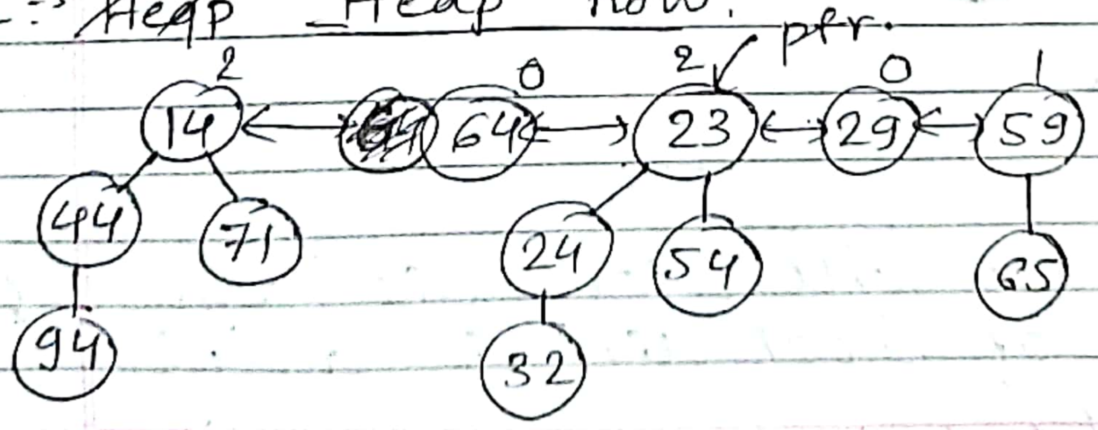
→ We would keep adding nodes to aux. array until we encounter nodes with same deg.

Now we are at 23, deg(23)=1 & we check A1 as it's filled we merge 23 & 24

Now 23 will be the root of merged tree ∵ it's less than 24

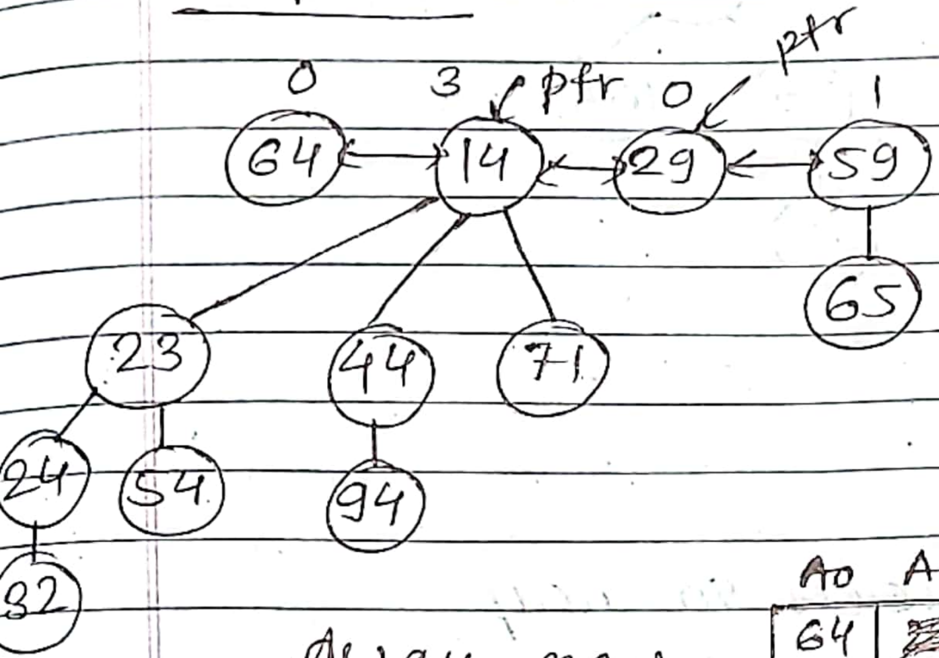
∴ ~~Tree now Heap~~

∴ ~~Heap~~ Heap now:



Now ||ly, we merge 23 & 14 of deg. 2, with 14 as root

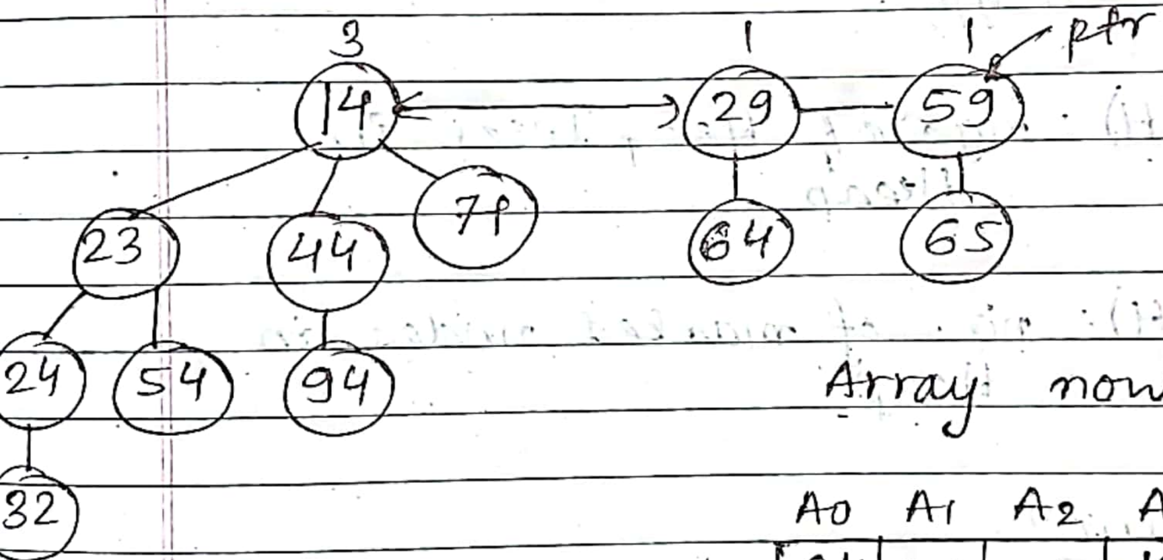
Heap now :



Array now

A_0	A_1	A_2	A_3
64			14

Now merge 64 & 29 with root 29

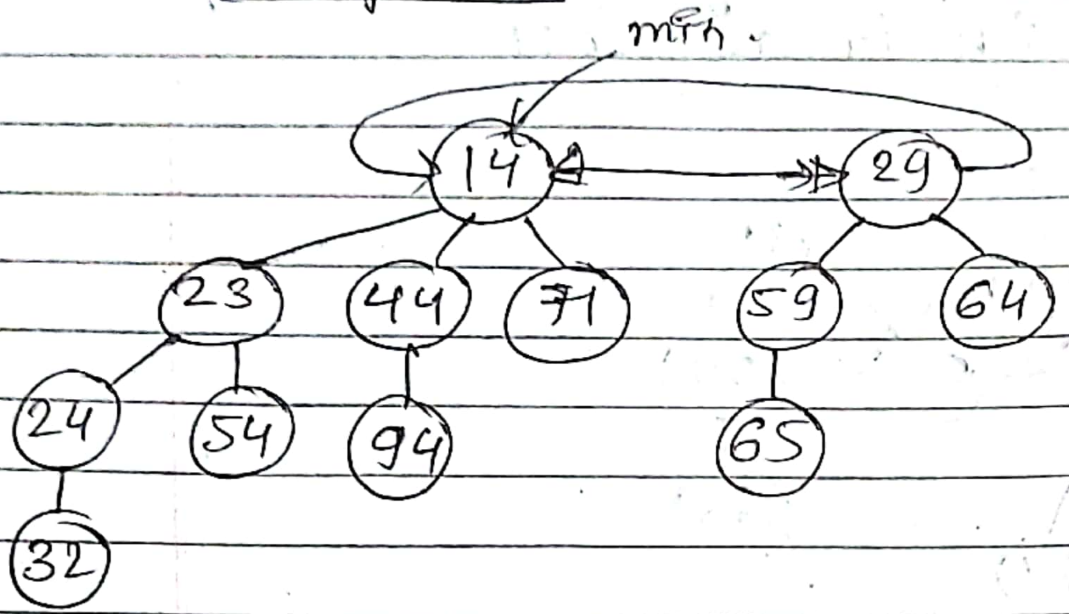


Array now

A_0	A_1	A_2	A_3
64	29		14

Now merge 29 & 59, with root 29

Heap now:



Notations:

→ n = no. of nodes in heap

→ rank(x) = no. of children of node x

→ rank(H) = max. rank of any node in Heap

→ trees(H) : no. of ~~Heap~~ Trees in Heap

→ marks(H) : no. of marked nodes in Heap

Pot. funcⁿ:

$\phi(H)$ = trees(H) + 2 * marks(H)