BINOMIAL HEAP

A binomial heap is a collection of binomial trees that are linked together. So firstly we should define binomial trees. A Binomial tree B_k is made up of two ~~dime~~ binomial trees B_{k-1} that are merged together such that root of one is the left most child of root of other.

(i) A Binomial tree of order 0 is a single node.
i.e. (3) (5) ...

(ii) A binomial tree of order 1 has two nodes

(iii) A binomial tree of order 2 has 4 nodes.

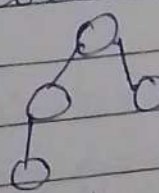
(iv) Similarly bin. tree of order k has 2^k nodes

- k represents height of tree.

No. of nodes in level $L = {}^kC_L$

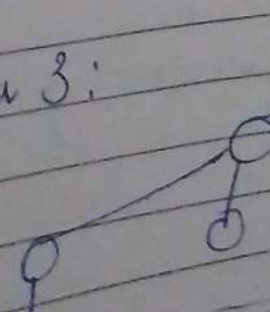
k = height of tree

Foreg: We consider Bin. tree of ~~k=~~ orders 2



No. of nodes in level $0 = {}^2C_0 = 1$
" " " $1 = {}^2C_1 = 2$
" " " $2 = {}^2C_2 = 1$

- Order 3:



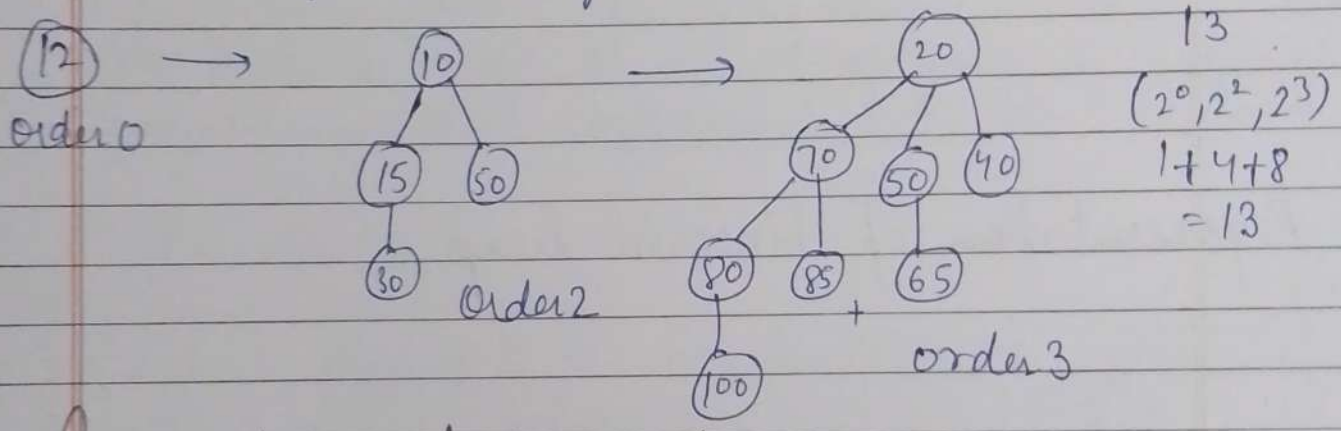
No. of nodes level 0 → ${}^3C_0 = 1$
L-1 → ${}^3C_1 = 3$
L-2 → ${}^3C_2 = 3$
L-3 → ${}^3C_3 = 1$

DATE: __/__/__
PAGE: _____

Binomial heap properties:

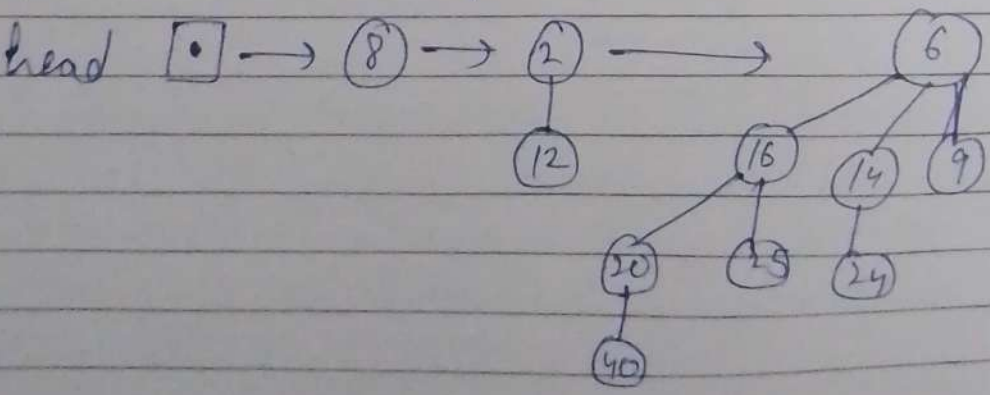
1. Each binomial tree in a bin. heap will follow min-heap property i.e. root of tree holds smallest value in tree.
2. In a heap of order k, there is only one binomial tree; whose root has degree k. (height)

for eg. A binomial heap with 13 nodes. A collection of 3 binomial trees of orders 0, 2, & 3 from left to right



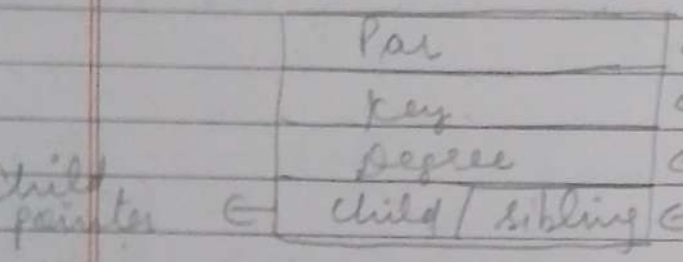
Representation of Binomial Heaps

A binomial heap with 11 nodes. binary representation of 11 is (1011 i.e. $2^3, 2^1, 2^0$). The bin-heap consists of 3 min heap of order 1 & 3

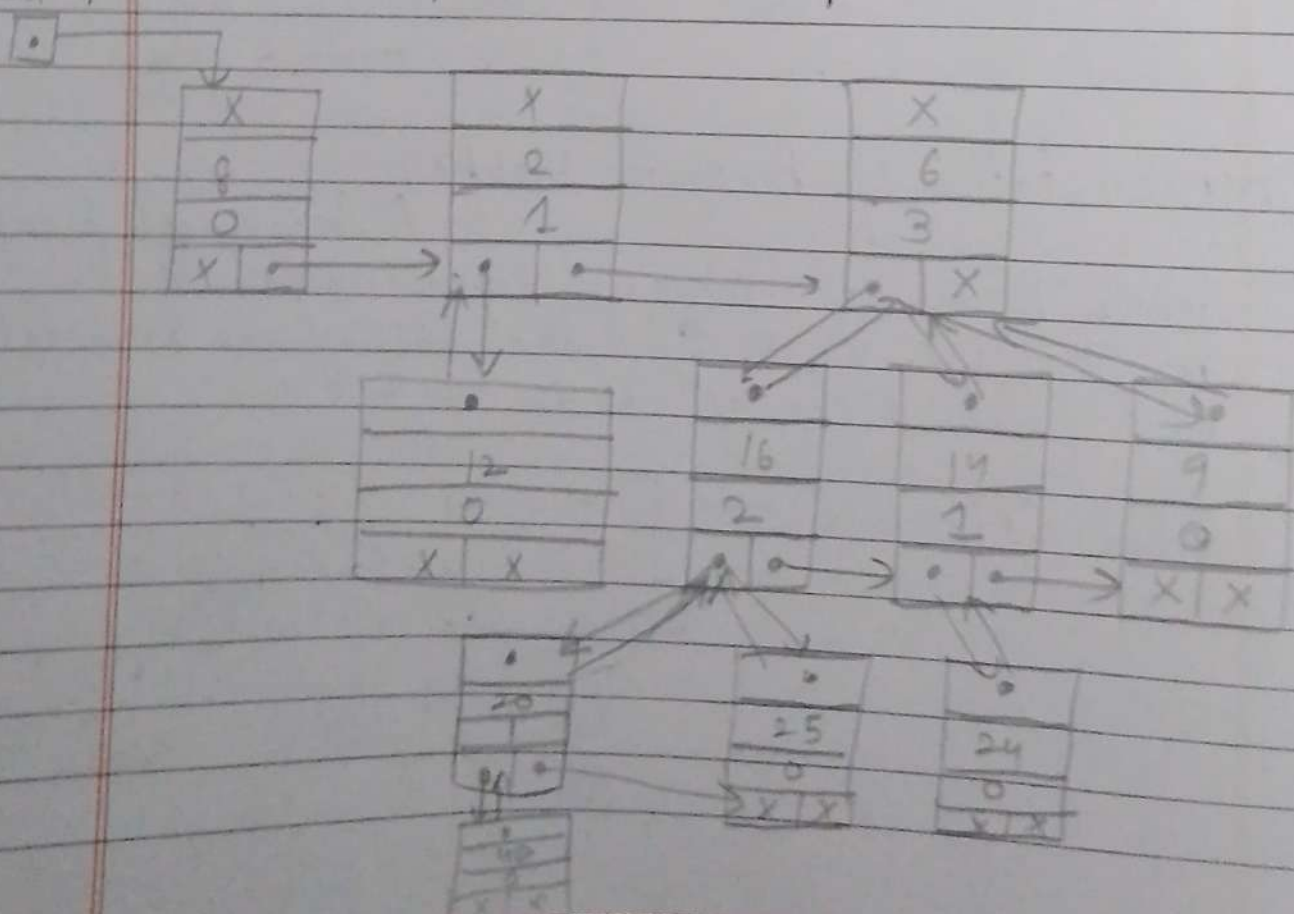


Representation of heap node: node in A Binomial heap with 14 nodes R consists of following parts

1. A pointer containing address of parent node.
2. A pointer containing address of child and sibling node.
3. Key value of node
4. Degree i.e. no. of children of node



Representation of binomial heap



Operations on binomial heap: Different operations performed on binomial heap are :-

(a) Finding Minimum key (root nodes).
(b) Uniting Binomial heaps
(c) Insertion
(d) Deletion
(e) Extract min
(f) decreasing key value

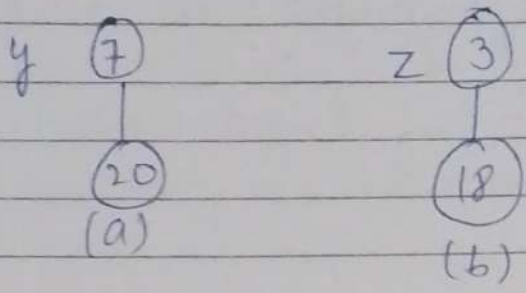
(a) Finding Min key value: It returns the min value in the binomial heap
(H). Since it follows min heap property, root of heap tree will have min value.

Steps:

1. X = start [H]
2. mini = ∞
3. Repeat step 4 & 5 while X != NULL
4. If key [x] < mini
5. mini = key[x]
   LOC = X
   [end if]
6. X = sibling [X]
   (End loop)
7. Return (mini, Loc)

(b) Uniting Binary Heap: Uniting Binomial heap is performed by using two procedures. First procedure we link binomial trees, whose roots have ~~procedure~~ same degree.

Consider two binomial tree of order 1.



(a) Binomial trees y & z.
Now since out of both trees the tree Z has min root value so we will make root of y as child of root of tree Z.

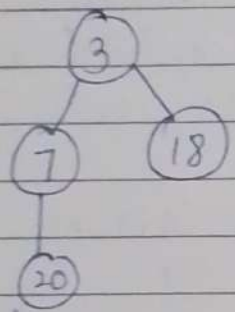


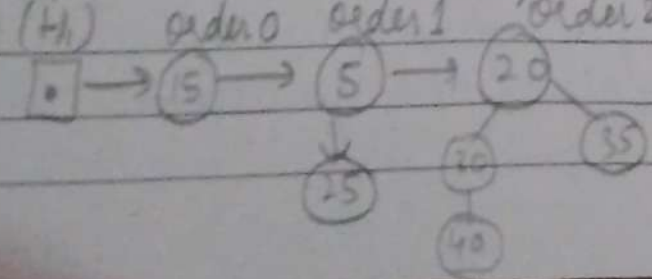
fig: linking operation on binomial heap.

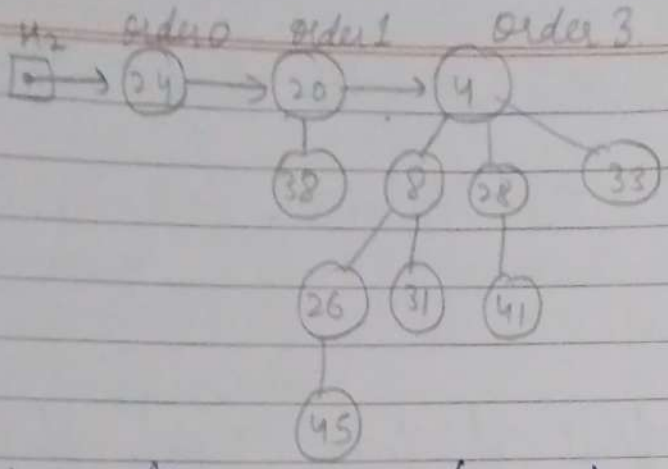
Steps:

1. Par [y] = z
2. Si [y] = child [z]
3. child [z] = y
4. Deg (z) = Deg [z] + 1
5. Return

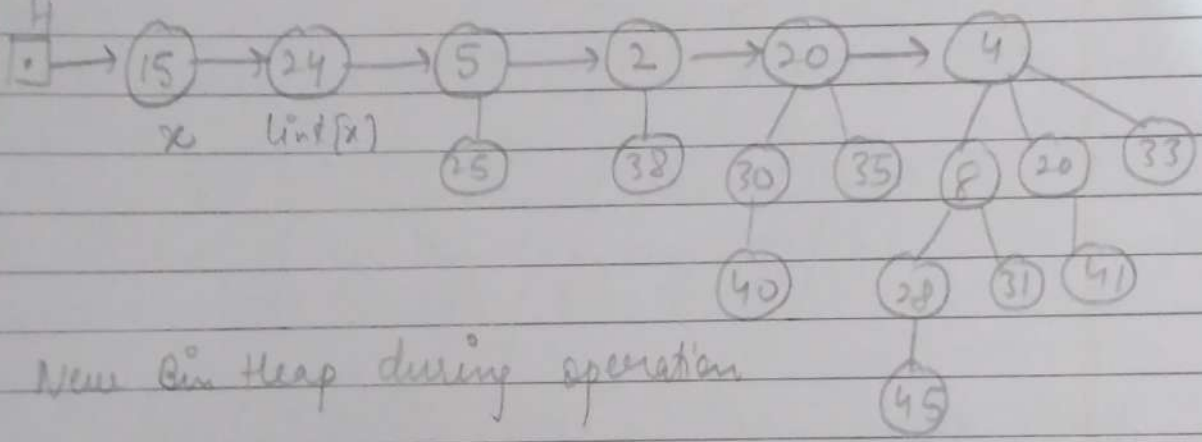
The second procedure, we unite binomial heaps say H_1 & H_2. First we merge two heap then we link roots & equal degree

For eg: Take two bin heaps H_1 & H_2



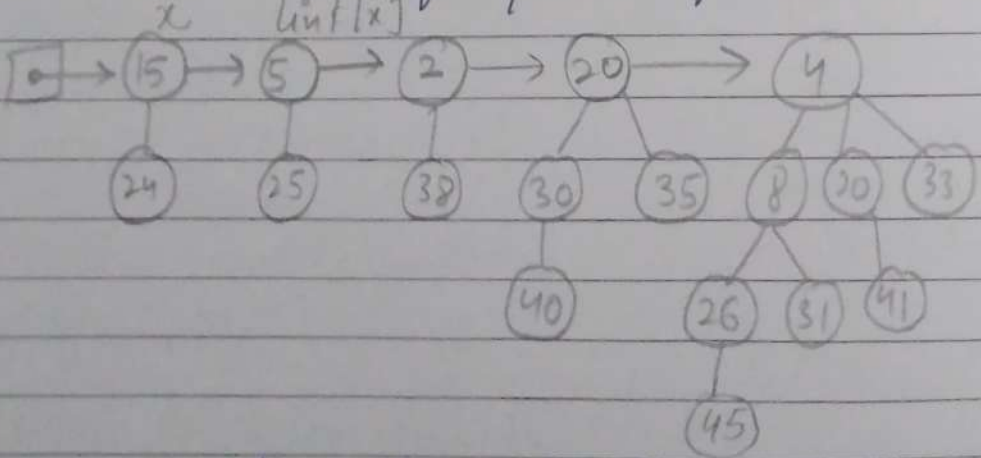


After calling merge (H_1, H_2) we get binomial heap.

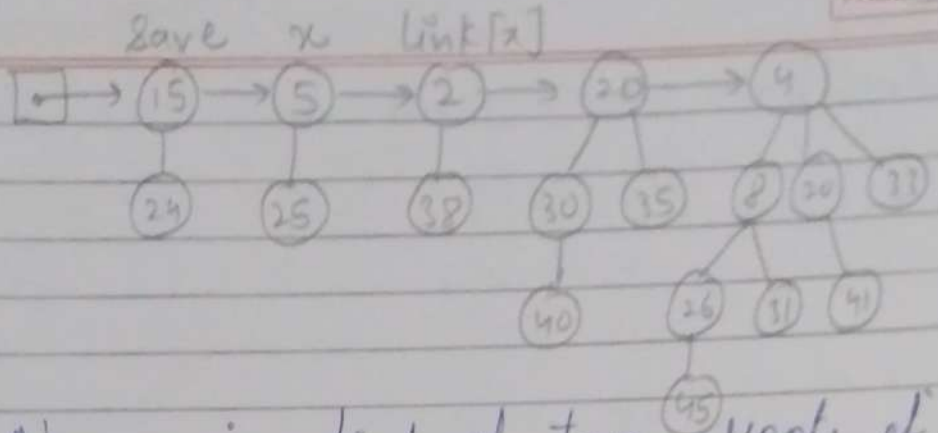


New Bin Heap during operation

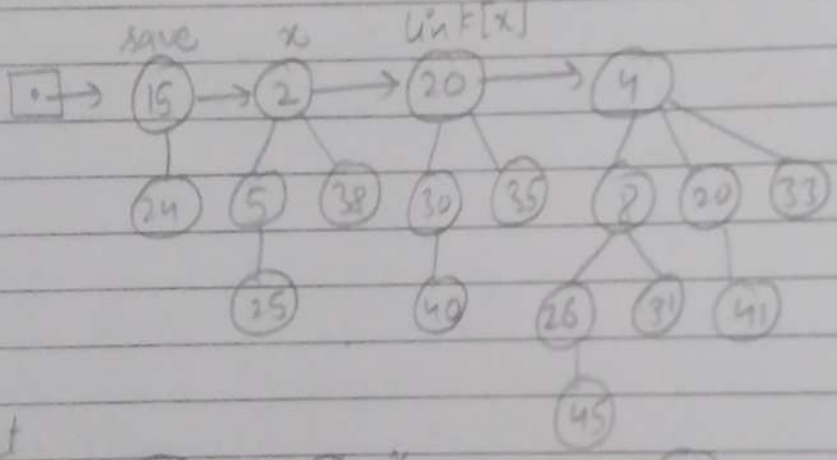
Here x is first root on root list of H. Both x & link [x] have same degree (i.e. 0) & key [x] < key link [x]. So we will links both roots of equal degree.

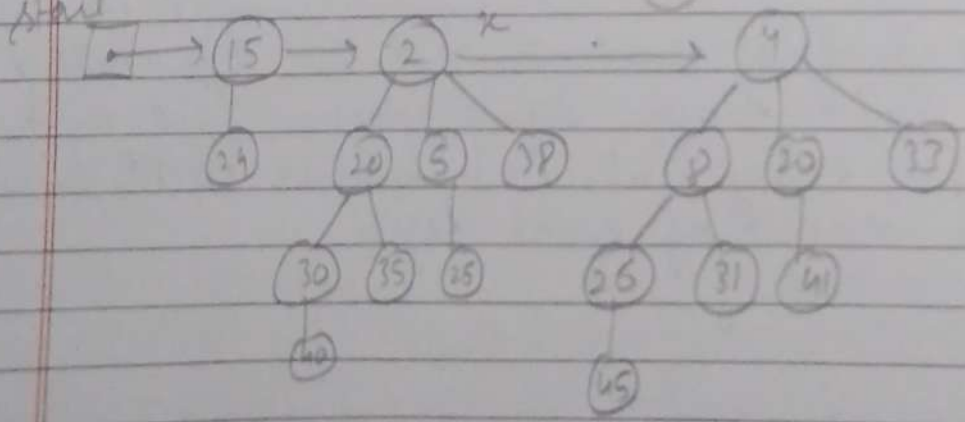


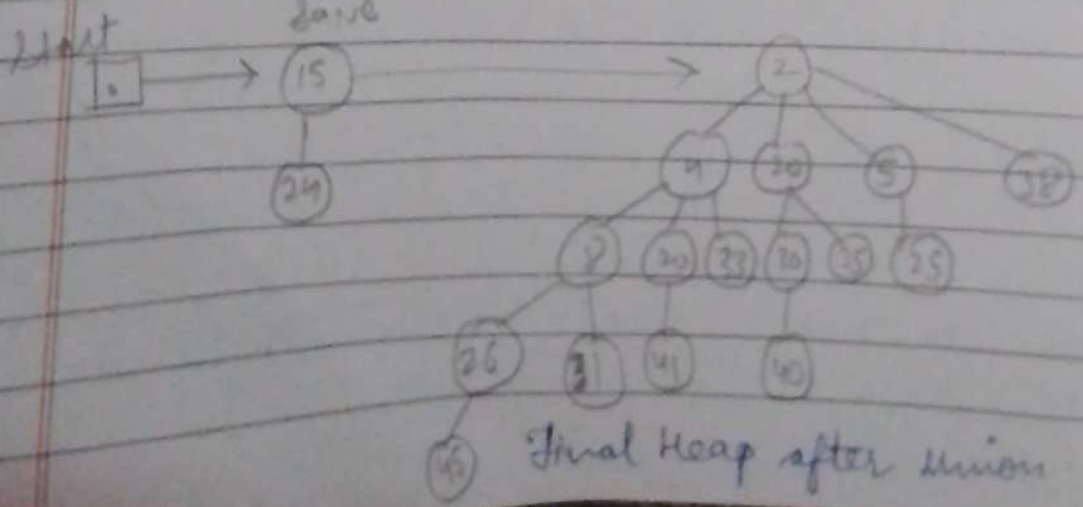
Now we have 3 roots of equal degree So update pointers.



Now x is first of two roots of equal degree







Final Heap after union

Explain above steps in words:

1. Merge two heaps.
2. Traverse list of merged roots. Keep track of 3 pointers, prev(x), next x & x.
   4 cases arise

Case 1: Orders of x & next x are not same, simply move ahead following cases have orders of x and next x same.

Case 2: If order of next_next x is also same, move ahead.

Case 3: If key of x key[x] ≤ key[next x] then make next x as child of x.

Case 4: If key[x] ≥ key[next x] then make x as child of next[x].

Insertion: Algo

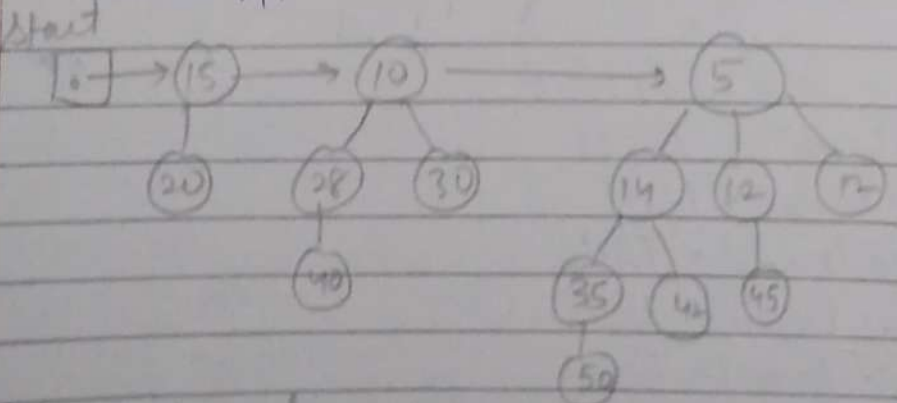
1 Node = Get Node()
2. key (Node) = ITEM
3. Par (Node) = Null
4. Child (Node) = Null
5. Sibling (Node) = Null
6. Deg (Node) = 0
7. Start [H] = Node — [created New heap]
8. Call Bin Heap Union (H, H')
9. Return H.

We set all pointers of new node equal to null. This node has 0 dg. Add this node to empty heap H'. Then call proc. Bin Heap Union Hence node inserted

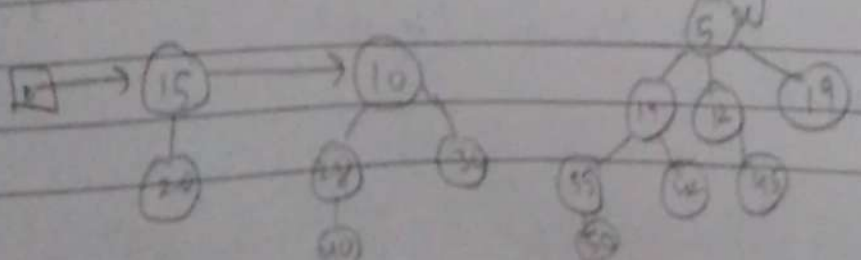
Extracting min key: To extract min key first of all we have to find min key value. To do so call proc. Bin Heap min. Then extract key & reverse order of linked list of child nodes of that key. Call heap Union to rearrange heap.

Algo:
1. Call Bin heap min (H)
2. x = loc
3. Remove x from root list of H.
4. Reverse order of linked list of child nodes of x and set start [H'] points to start of resulting list.
5. Call Bin Heap Union (H, H').
6. Return H.

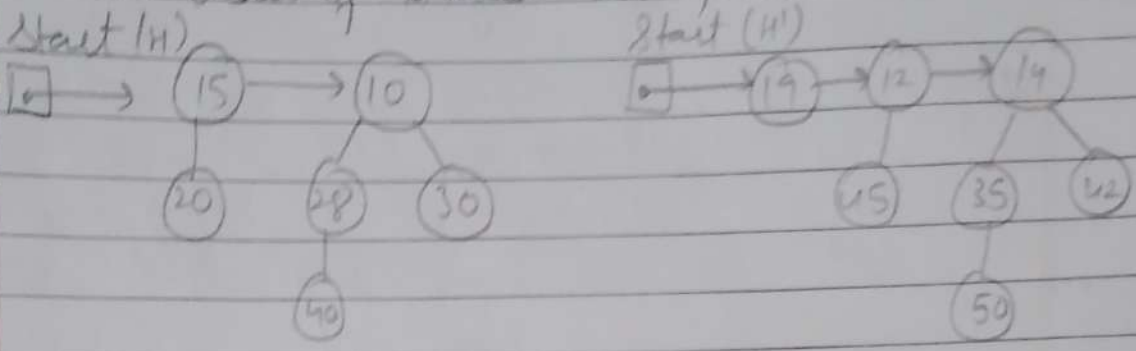
Ex.



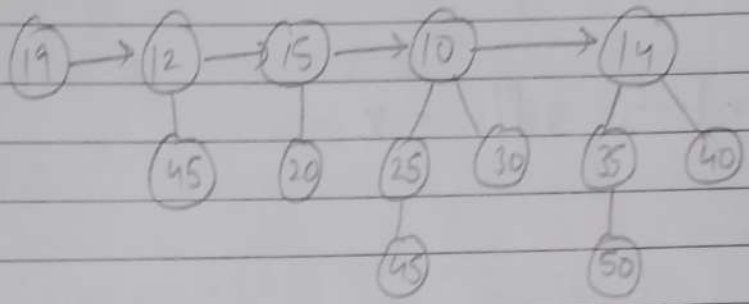
Min value for above Bin Heap is 5.



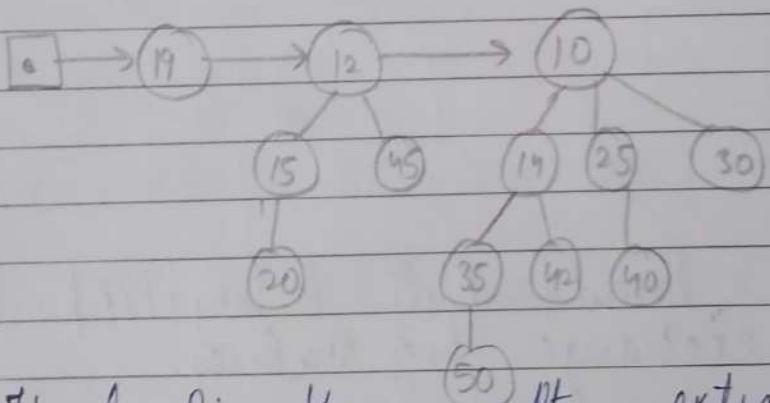
Reverse order of linked list of child



Unite H & H'



Start (H)



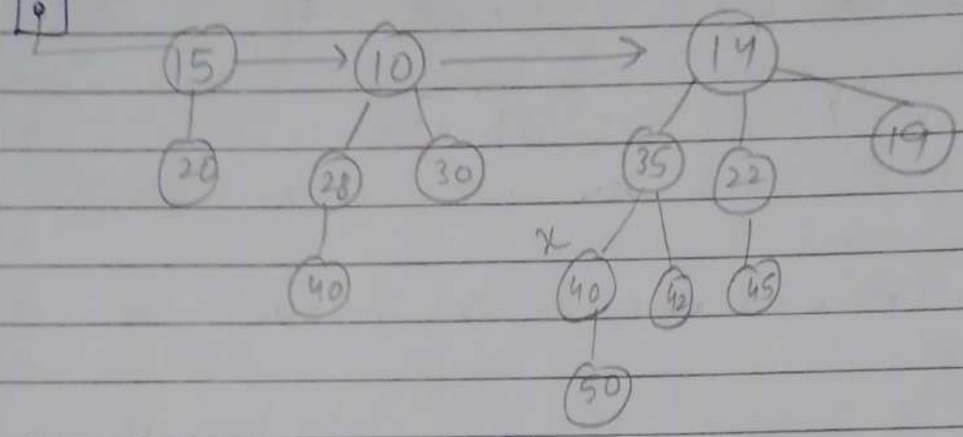
Final Bin. Heap after extracting min.

Decreasing key value: Dec key value is process of decreasing key value of node say x in bin. heap say H, to given value, say k. If k is greater than x, decreasing operation not possible

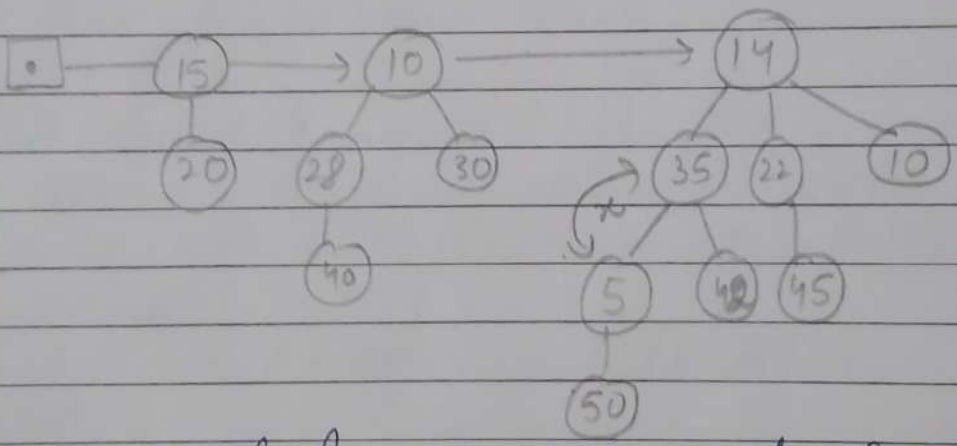
Syntax: Bin - Heap - decrease (H, x, k)

DATE: __/__/__
PAGE: _____

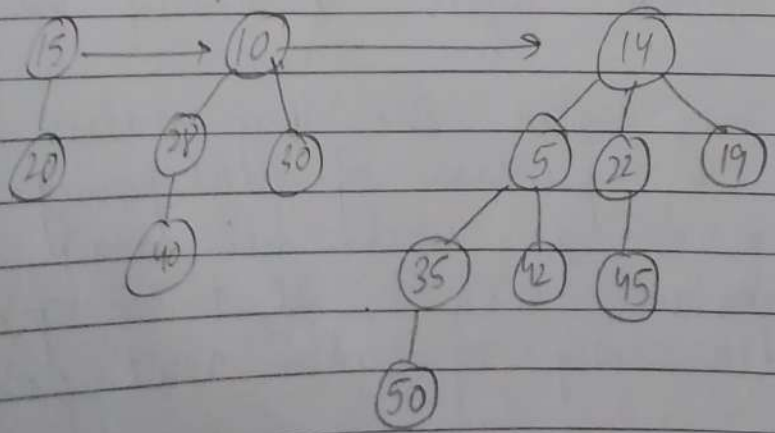
Ex:- Let k = 5 & key(x) = 40



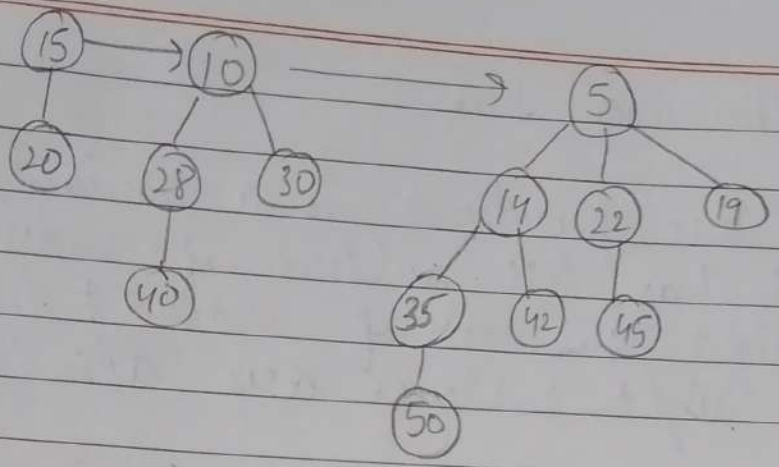
Let : key[x] = 5 ⇒ Ptr = x



Now while Parent != Null & key[PTR] < key [Parent] exchange key values.



again 5 < 14 exchange



Now min value is at root.

Deletion: To delete key from binary heap, we have to call 2 procedure Bin-Heap-decrease & Bin heap extract. Here we decrease key value up to first proc. will bring this key $-\infty$ to root of Bin. heap and then we call extract min because $-\infty$ will be min value in whole Binary Heap.

Syntax: Delete-Bin-Heap [H, x]

Procedure to delete given:

1. Call Bin-Heap-Decrease (H, x, $-\infty$)
2. Call Bin-Heap-Extract min (H)
3. Exit.