Fibonacci Heaps

→ Collection of trees with each tree following the heap ordering (min heap) property.

→ Trees may be in any order in the root list.
(Unlike Binomial Heaps)

→ A pointer to the min element of the heap always maintained.

→ Siblings are connected through a circular doubly linked list.

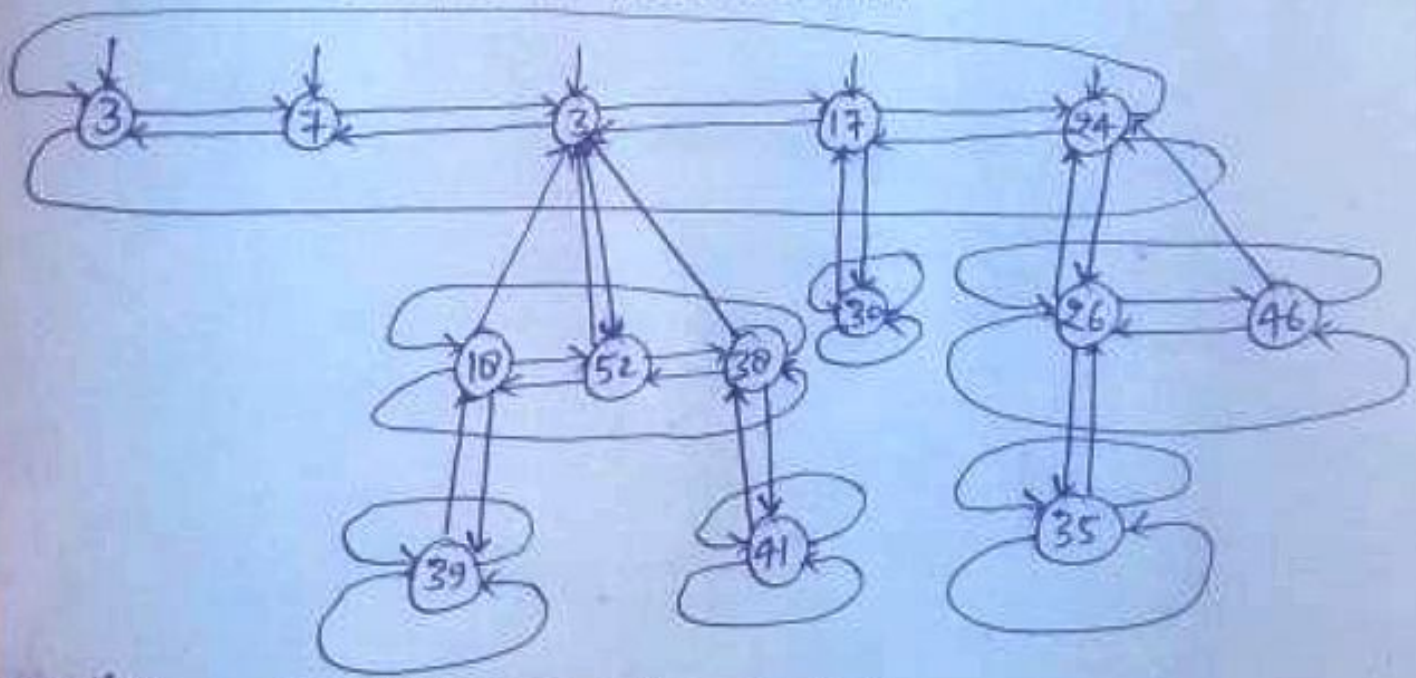
→ Each child points to its parent.

→ Each parent points to any one child.

→ degree(x): No. of children of root of a tree.

→ Mark(x): 1 – Lost one of its child
0 – Lost no child

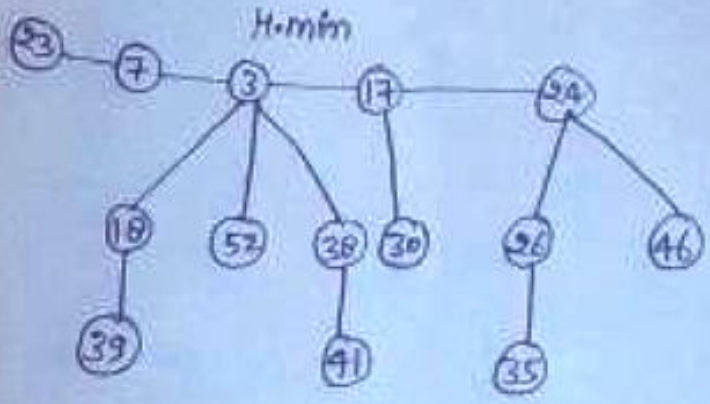
Structure of Fibonacci Heap



Notation:

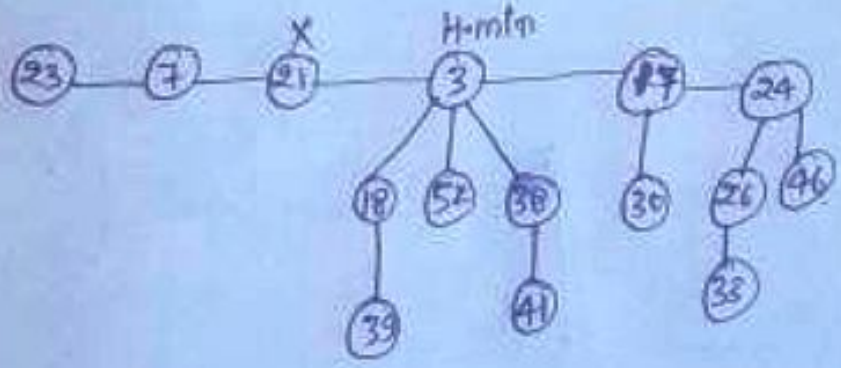
n: Number of nodes in heap.
rank(x): Number of children of node x.
rank(H): Max rank of any node in heap H.
trees(H): Number of trees in heap H.
marks(H): Number of marked nodes in heap H.

Insertion of key



Fib-Heap-Insert (H, x)

1. x.degree = 0
2. x.P = NIL
3. x.child = NIL
4. x.mark = FALSE
5. If H.min == NIL
6. create a root list for H containing just x.
7. H.min = x
8. else insert x into H's root list
9. If x.key < H.min.key
10. H.min = x
11. H.n = H.n + 1

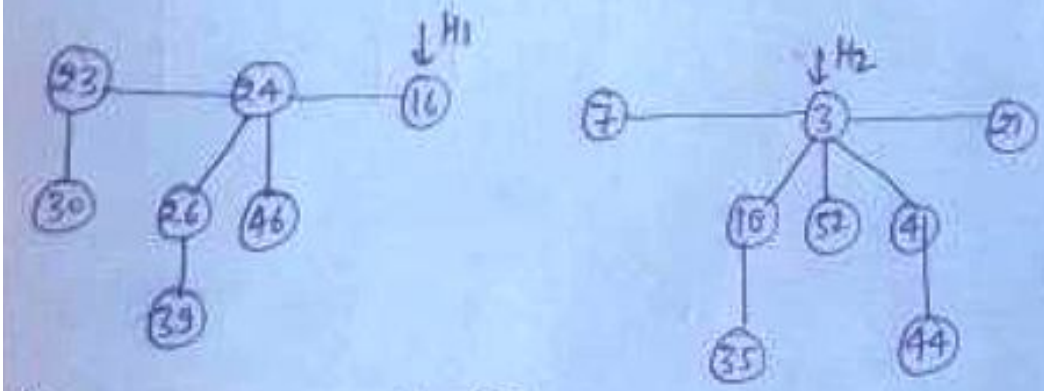


TC = O(1)

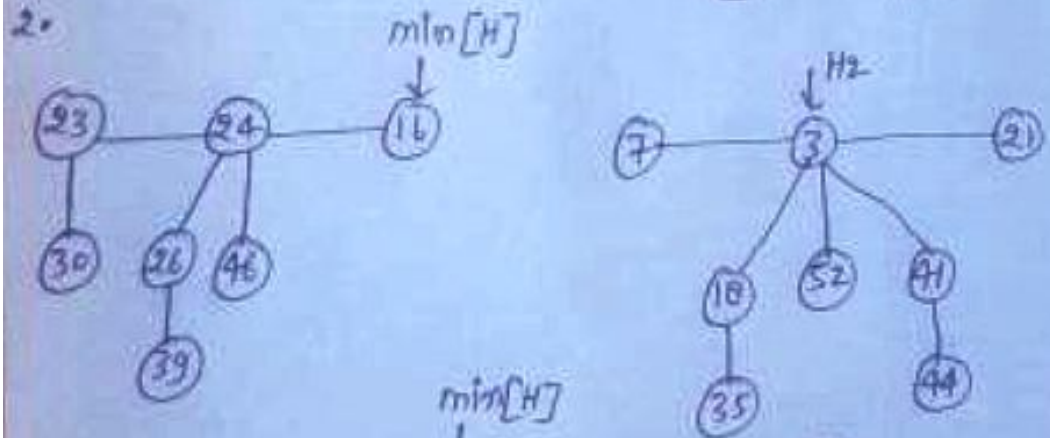
UNION of Fibonacci Heaps

Fib_Heap_UNION(H_1, H_2)

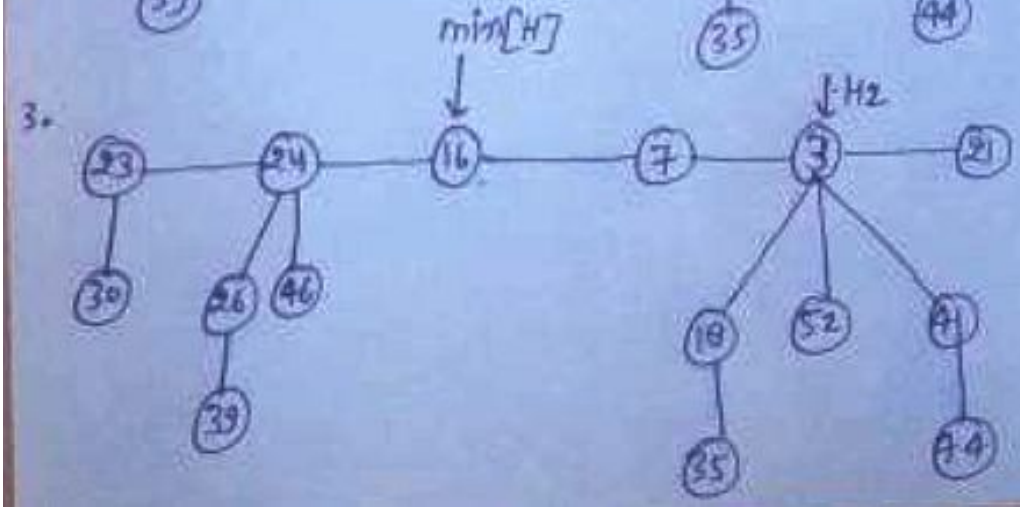
1. $H \leftarrow$ Make_Fib_Heap(●)
2. $min[H] \leftarrow min[H_1]$
3. Concatenate the root list of H_2 with the root list of H
4. If ($min[H_1] = NIL$) or ($min[H_2] \neq NIL$ and $min[H_2] < min[H_1]$)
5. then $min[H] \leftarrow min[H_2]$
6. $n[H] \leftarrow n[H_1] + n[H_2]$
7. free the objects H_1 and H_2
8. return H

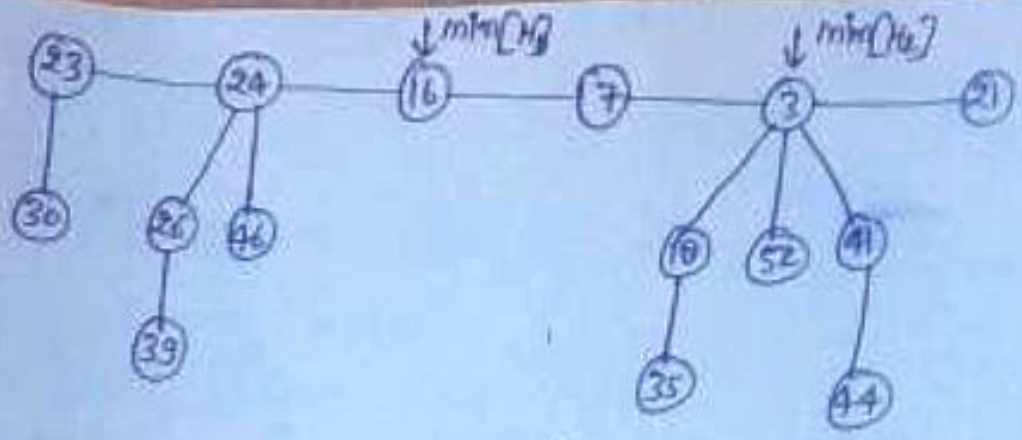


2.

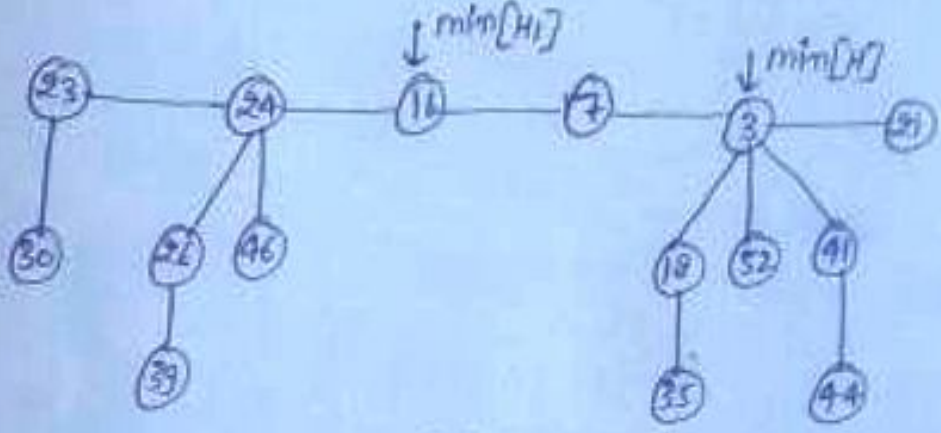


3.

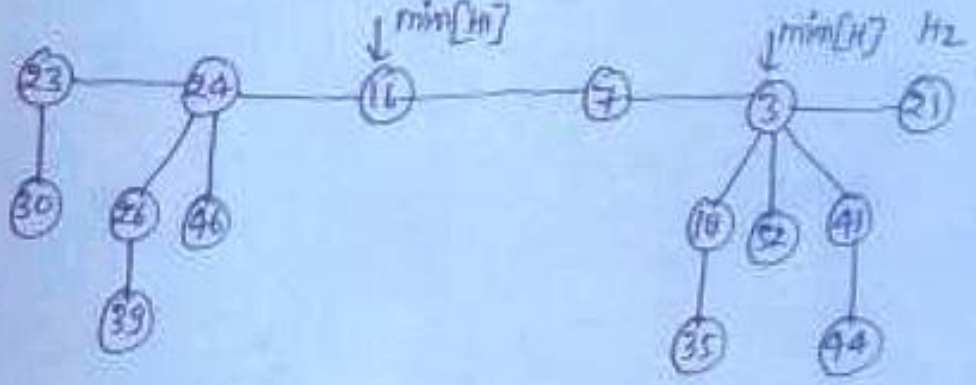




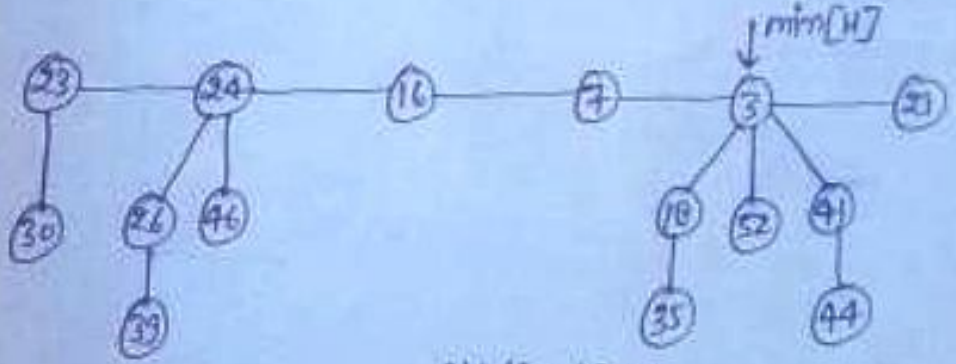
5.



6.



7.



$Tc = O(1)$

Applications

1. Priority Queue Implementation.
2. Large amount of data representation.
3. Decrease key operation is used in minimum spanning tree algorithm.
4. Single source shortest path.