UNIT-2

CHAPTER-4

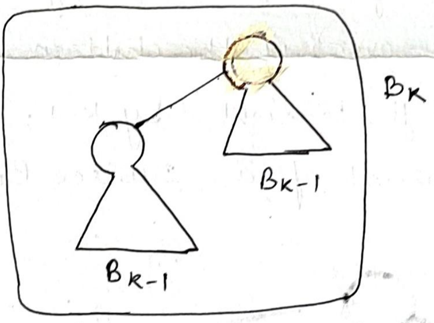
BINOMIAL HEAPS

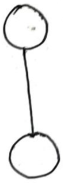
A Binomial heap is a collection of Binomial trees. A Binomial tree B_K is an ordered tree defined recursively:-

(i) The Binomial tree B_0 consists of a single node

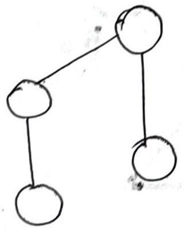
(ii) For $K \geq 1$, the Binomial tree B_K consists of two Binomial trees B_{K-1} that are linked together: the root of one is the left most child of the root of the other.

B_0

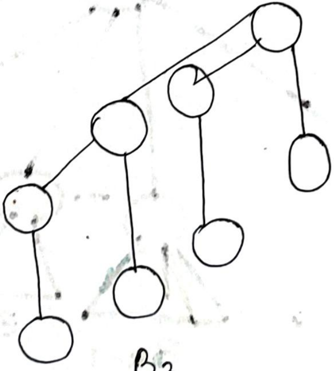




B_1



B_2



B_3.

. In terms of depth it is clear that

B_0 has node $= 2^{depth} = 2^0 = 1$ node

B_1 has node $= 2^1 = 2$ nodes

B_2 has node $= 2^2 = 4$ nodes.

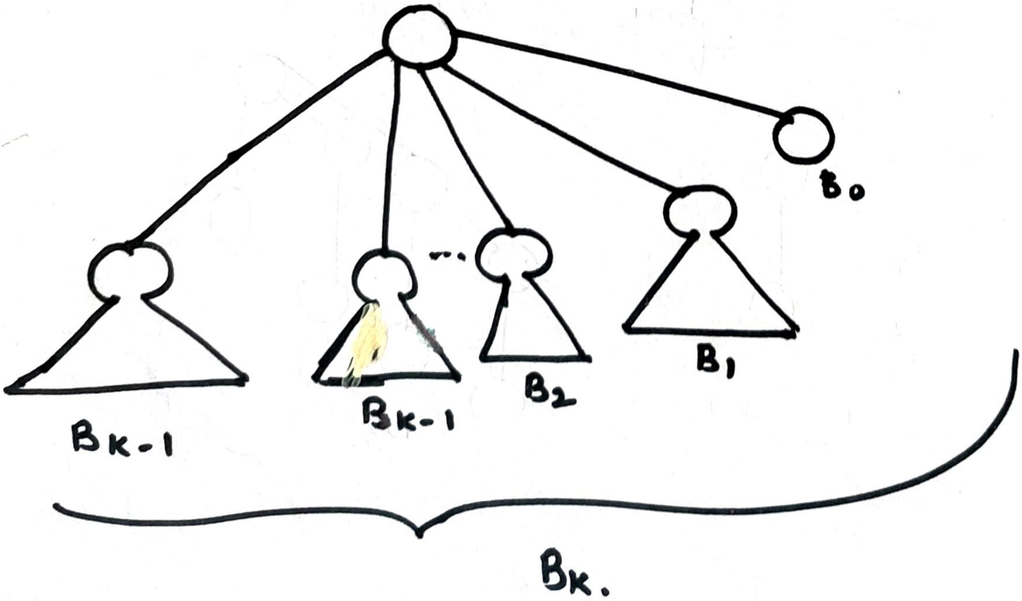
Properties of Binomial Trees

For the binomial tree B_K.

1. There are 2^k nodes
2. The height of the tree is k.

$$\frac{K!}{i!(K-i)!}$$

3. There are exactly nodes $\binom{K}{i}$ at depth i for $i = 0, 1, \ldots, k$.
4. The root has degree k, which is greater than that of any other node ; more-over if the children of the root are numbered from left to right by $k-1, k-2, \ldots, 0$ child i is the root of a subtree B_i



Binomial Heaps

A Binomial heap H is a set of binomial trees that satisfies the following binomial-heap properties :-

1. Each binomial tree in H obeys the min-heap property :- the key of a node is greater than or equal to the key of its parent.

2. For non negative integer k, there is at most one binomial tree in H whose root has degree k.

3. Binomial trees will be joined by a linked list of roots.

(a) The first property implies that the root of each binomial tree contains the smallest element in that tree.

(b) The second property implies that, There can be at most $\lceil \log n \rceil + 1$ binomial trees in a binomial heap with n nodes.

Representation of Binomial Heaps

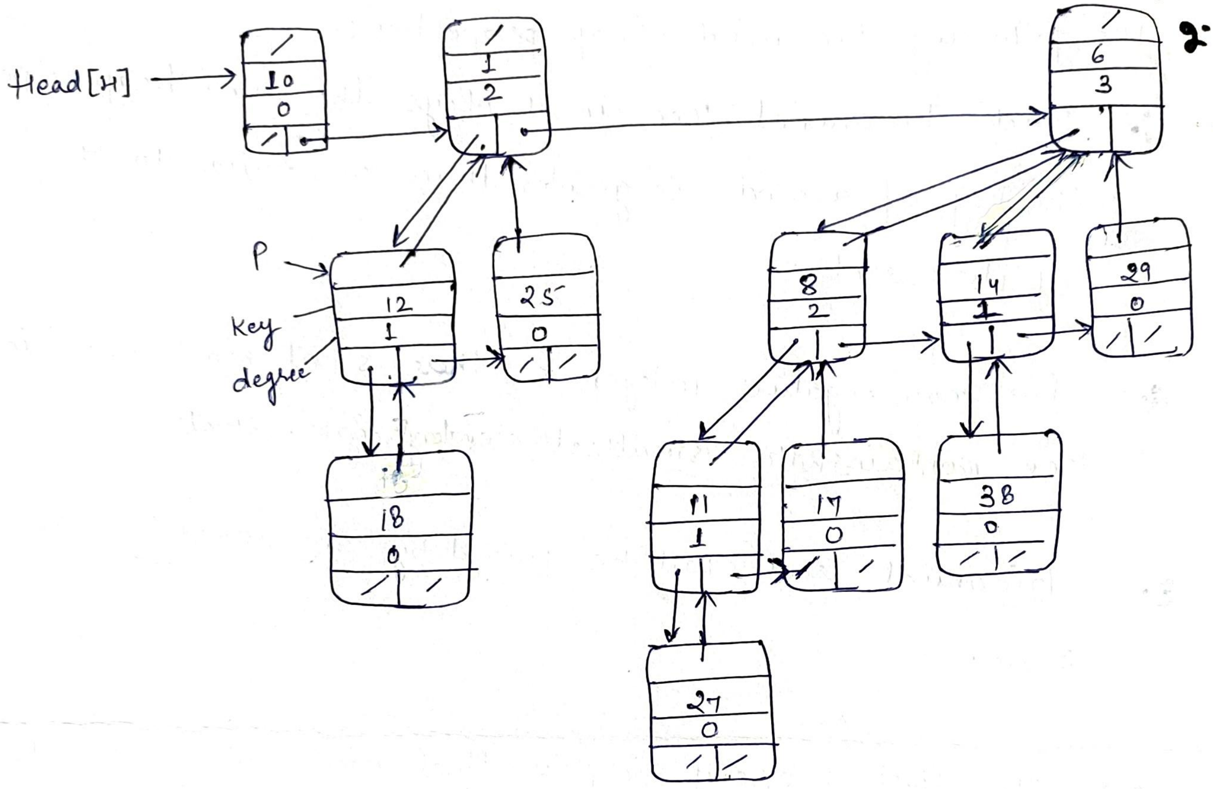


figure:- A Binomial Heap with n=13 nodes.

(a) The Heap consists of Binomial trees B_0, B_2, B_3 which have 1, 4, and 8 nodes respectively. Since each binomial tree is min-heap-ordered, the key of any node is no less than the key of its parent. Also shown is the root list, which is a linked list of roots inorder increasing degree.

(b) In Binomial heap H, Each binomial tree is stored in the left child, right sibling representation & each node stores its degree.

Operations on Binomial Heaps :- Firstly create a Binomial heap after that the following operations are performed.

1. finding the minimum key
2. Union of two binomial heaps
3. Inserting a node
4. Extracting a node with minimum key
5. Decrease a key.

Creating a new binomial heap

To make an empty binomial heap, the MAKE-BINOMIAL-HEAP procedure simply allocates and returns an object H, where head[H] = NIL. The running time is $\Theta(1)$

Finding the minimum key

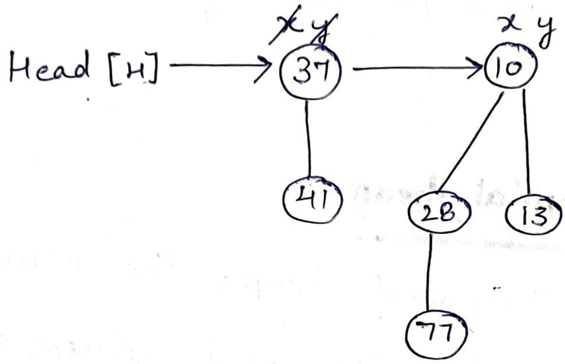
The procedure BINOMIAL-HEAP-MINIMUM returns a pointer to the node with the minimum key in an n-node binomial heap H.

```
BINOMIAL-HEAP-MINIMUM (H)
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL
      do if key[x] < min
            then min ← key[x]
                 y ← x
         x ← sibling[x]
return y
```

Analysis of Binomial-Heap-minimum (H)

There are at most $\lfloor \lg n \rfloor + 1$ roots to check, the running time of BINOMIAL- HEAP-MINIMUM is $O(\lg n)$.

Example



```
y ← NIL
 x ← HEAD [H]
  min ← ∞
  while x ≠ NIL (TRUE)
          if key [x] < min
              37 < ∞ (True)
              min ← key [x]
              min ← 37
              y ← x
          x ← sibling [x]
  while x ≠ NIL (True)
          if key[x] < min
              10 < 37 ( True)
              min ← 10
              y ← x
          x ← sibling [x]
  while nil ≠ nil ( false)

  return min 10
```

Proved.

Union of Two Binomial Heaps

```
BINOMIAL-HEAP-UNION (H1, H2)
H ← MAKE-BINOMIAL-HEAP()
head [H] ← BINOMIAL-HEAP-MERGE (H1, H2)
If Head[H] = NIL
        return H
prev-x ← NIL
x ← head [H]
next-x ← sibling [x]
while next-x ≠ NIL
    do if (degree [x] ≠ degree [next-x]) or
          (sibling [next-x] ≠ NIL and degree [sibling[next-x]] =
                                            degree[x])
        then prev-x ← x
             x ← next-x
    else if key [x] ≤ key [next-x]
        then sibling [x] ← sibling [next-x]
             BINOMIAL-LINK ( next-x, x)

    else if prev-x = NIL
             then head [H] ← next-x
         else sibling [prev-x] ← next-x
             BINOMIAL-LINK (x, next-x)
             x ← next-x

    next-x ← sibling [x]

return H
```

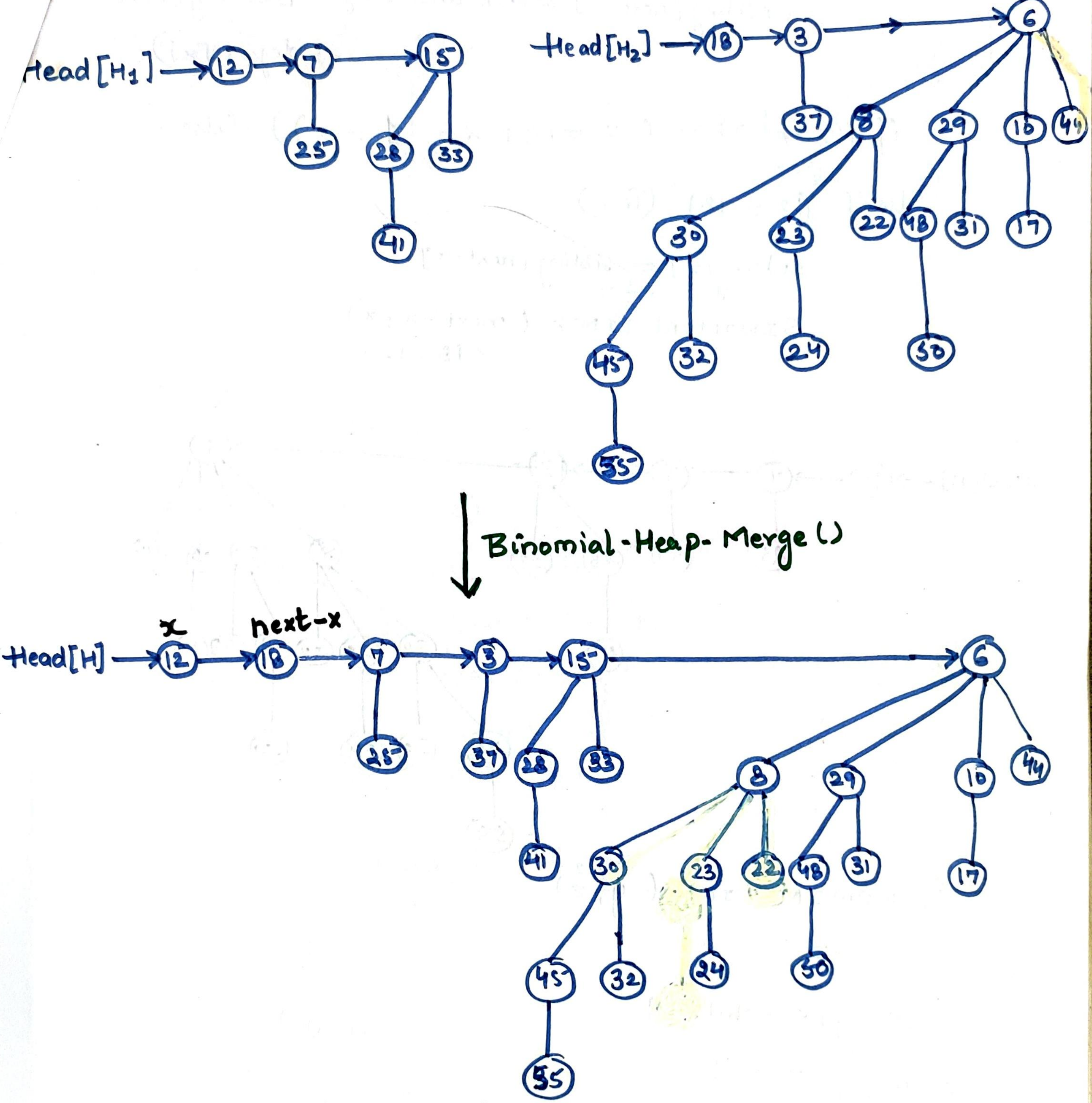
Binomial-Heap-Merge (H_1, H_2)

```
a ← head[H1]
b ← head[H2]
head[H1] ← Min-Degree(a,b)
if head[H1] = NIL
    return
if head[H1] = b
    then b ← a
a ← head[H1]
while b ≠ NIL
    do if sibling[a] = NIL
        then sibling[a] ← b
            return
    else if degree[sibling[a]] < degree[b]
        then a ← sibling[a]
    else c ← sibling[b]
        sibling[b] ← sibling[a]
        sibling[a] ← b
        a ← sibling[a]
        b ← c
```

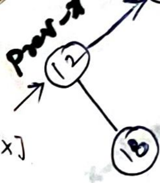
Analysis of Binomial-Heap-Union()

The running time of BINOMIAL-HEAP-UNION is $O(\lg n)$, where n is the total number of nodes in binomial heaps H_1 and H_2

Example of Binomial Heap-Union



```
If Head[H] = NIL (false)
        return

prev-x ← NIL
x ← head[H]
next-x ← sibling[x]
while next-x ≠ NIL
        18 ≠ NIL (True)
```



if (degree[x] ≠ ~~d~~degree [next-x]) or

(sibling [next-x] ≠ NIL and degree [sibling [next-x]

= degree [x])

if ((0 ≠ 0) or (7 ≠ NIL && (1 = 0)) false.

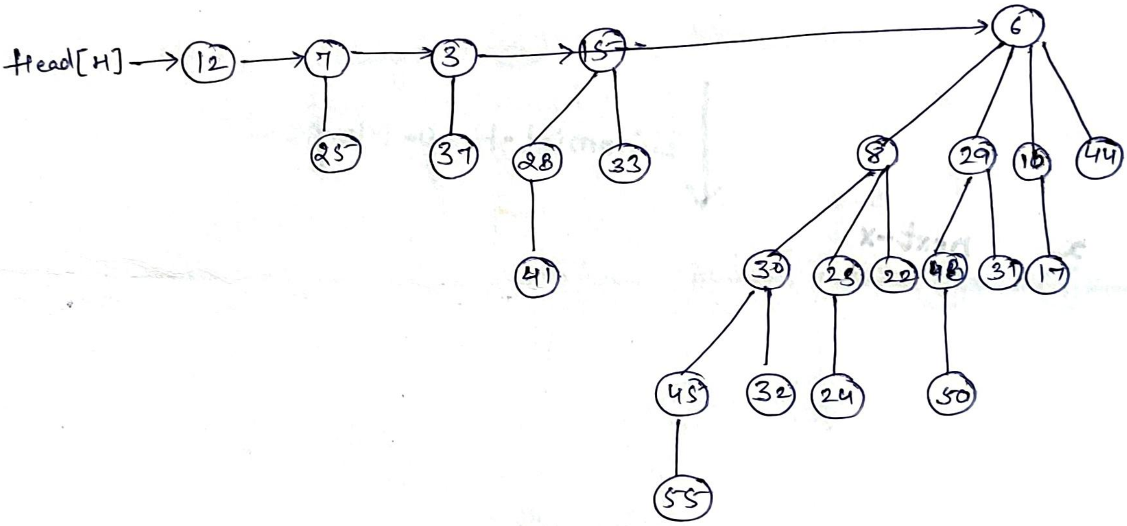
else if (12 ≤ 18) (True)

sibling[x] ← sibling [next-x]

← 7

BINOMIAL LINK (next-x, x)

(18, 12)



BINOMIAL LINK ($\overset{18}{y}$, $\overset{12}{z}$)

P[y] ← z

Sibling[y] ← child[z]

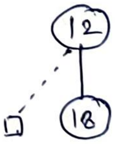
sibling[y] ← nil

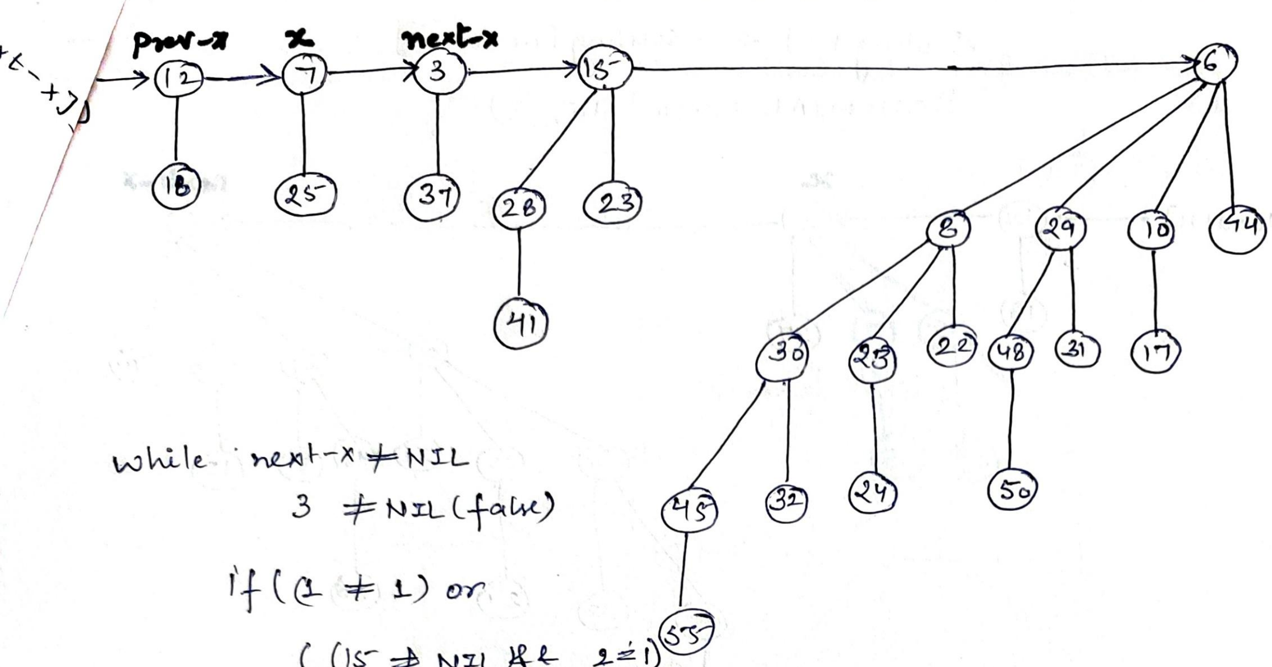
child[z] ← y.

degree [z] ← degree [z] + 1

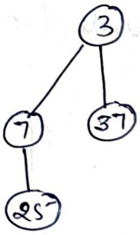
degree [z] ← 0 + 1

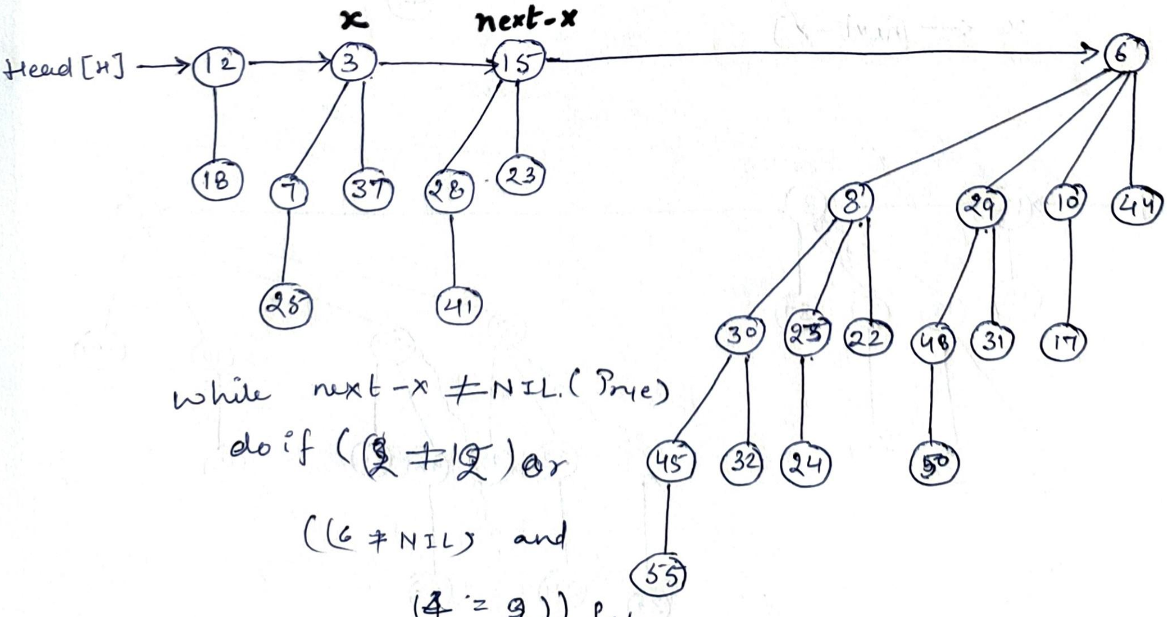
← 1.





```
while   next-x ≠ NIL
          3 ≠ NIL (false)
    if (1 ≠ 1) or
        ((15 ≠ NIL && 2 = 1) false

  else if  7 ≤ 3  (false)
    else if prev-x = NIL  (false)
      else    sibling [prev-x] ← next-x
                             y, z
              BINOMIAL LINK (7, 3)

              x ← next-x
              x ← 3
```

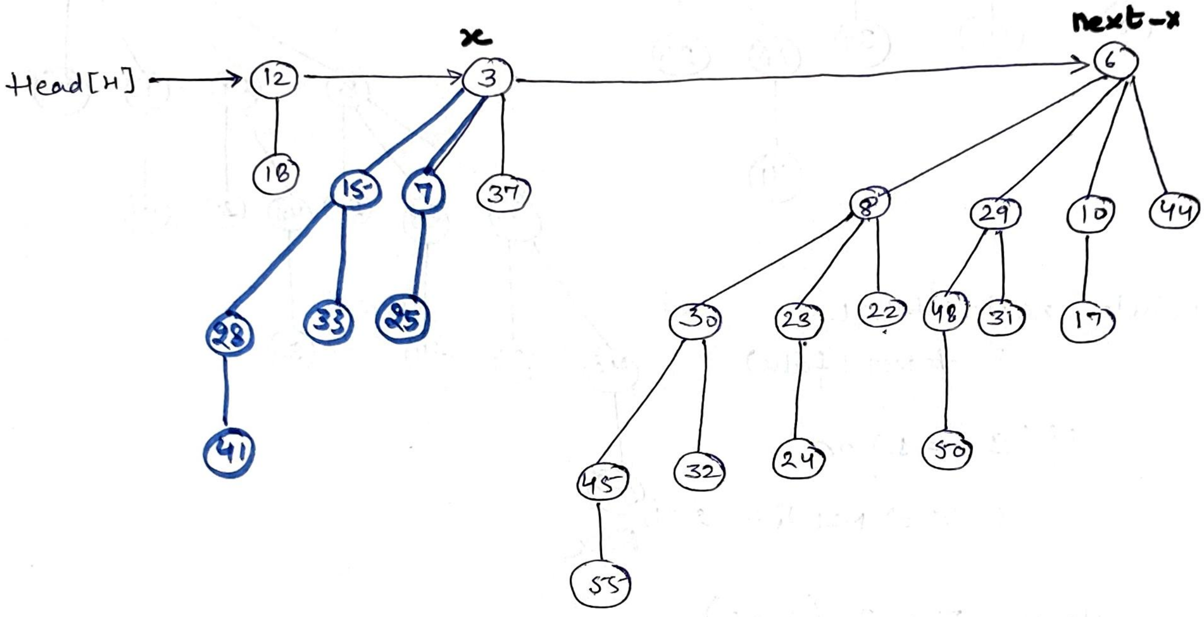




```
while  next-x ≠ NIL. (True)
  do if (2 ≠ 2) or
      ((6 ≠ NIL) and
          (4 = 2)) false.
  else if  3 ≤ 15  (True)
```

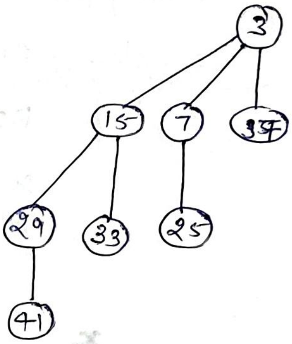
Sibling [x] ← sibiling [next-x]

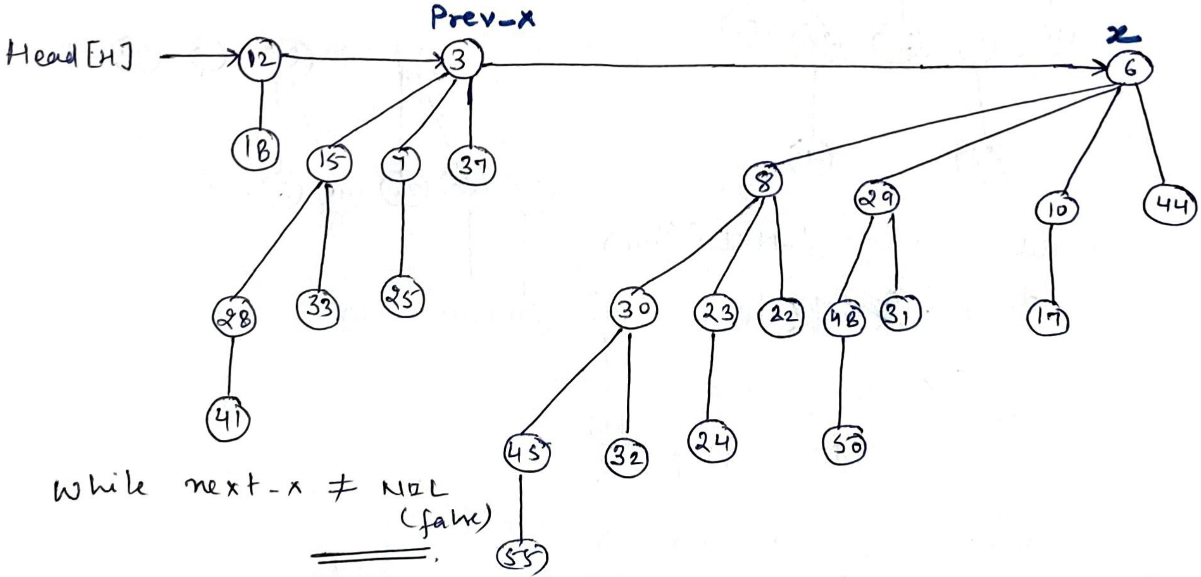
BINOMIAL LINK (15, 3)



BINOMIAL LINK ($\overset{y}{15}, \overset{z}{3}$)

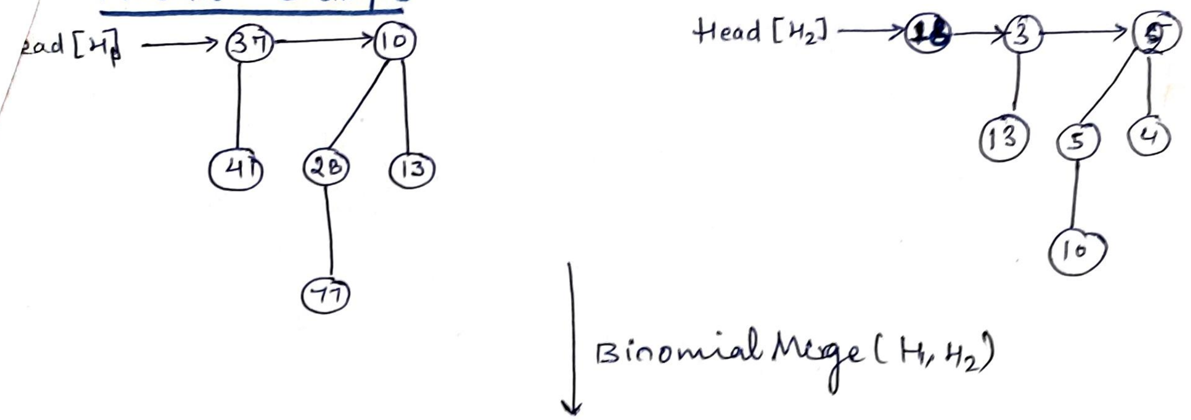
```
while ( next-x ≠ NIL )
        6 ≠ NIL ( True)

    if ( 3 ≠ 4)   True.

      [prev-x] ← x

        x ← [next-x]
```

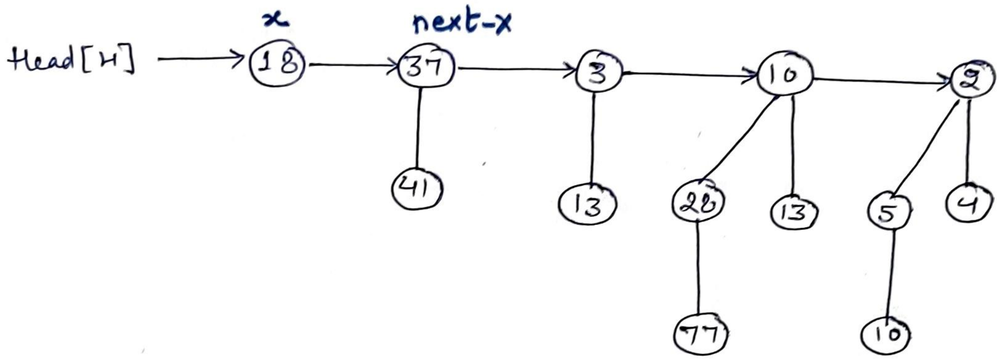




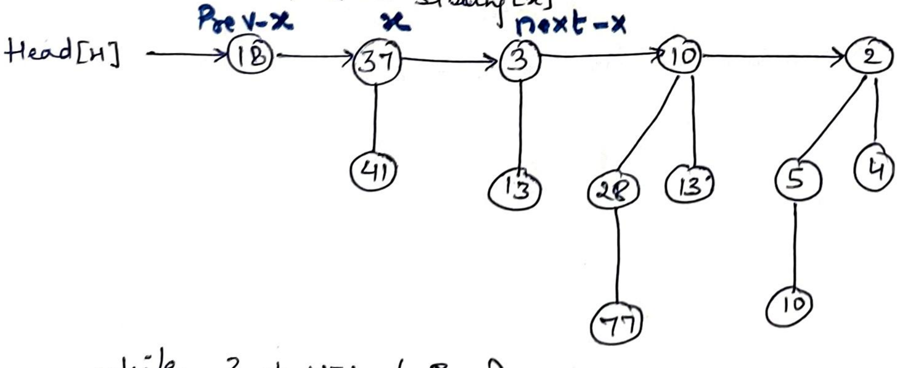
While next-x ≠ NIL
(false)

Another Example



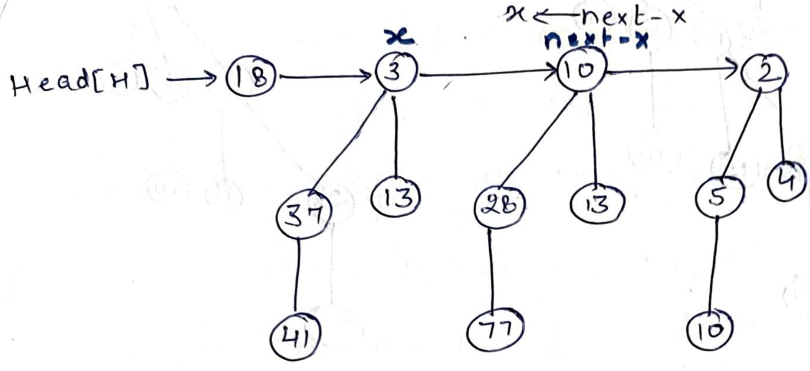


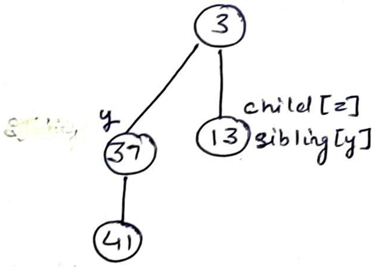
```
prev-x ← NIL

x ← head[H]

while 37 ≠ NIL ( True)
        if (0 ≠ 1) (True)
            prev-x ← x
            x ← next-x
    next-x ← sibling [x]
```



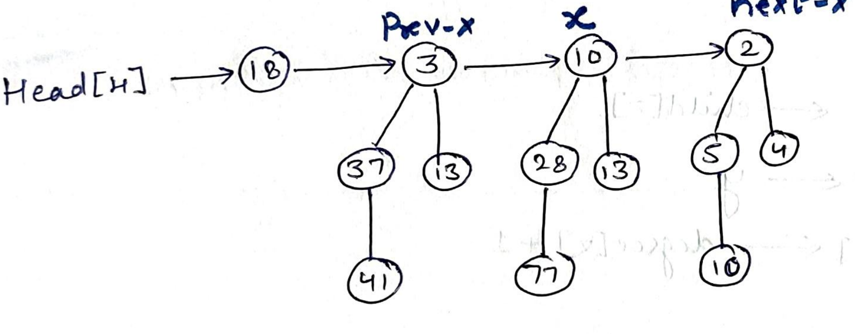
```
while 3 ≠ NIL ( True)
    if ((1 ≠ 1) or( sibling [next-x] ≠ NIL and 2 = 1) false
    else if 37 ≤ 3 (false)
        else if prev-x = NIL (false)
```

```
else
    sibling[prev-x] ← next-x
    BINOMIAL LINK( $\overset{y}{37}$ , $\overset{z}{3}$ )
    x ← next-x
```

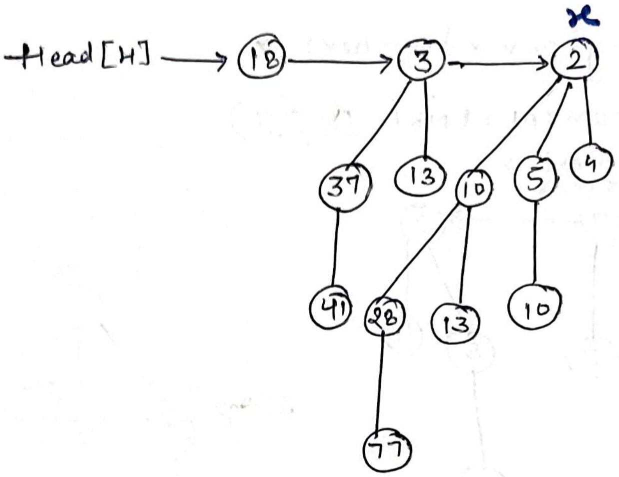


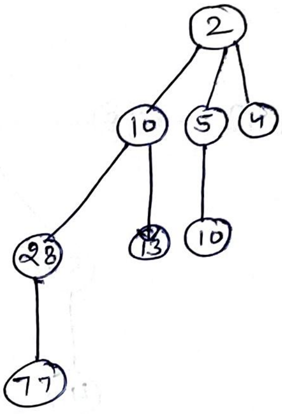


```
while 10 ≠ NIL (True)
    if ((2 ≠ 2)' or (2 ≠ NIL and 2 = 2) (True)

        prev-x ← x
        x ← next-x
        next-x ← sibling[x]
```



```
while 2≠NIL (false)
    if (2 ≠ 2 or nil = 2) false

    else if 10 ≤ 2 (false.

        else if prev-x = NIL (false)

            else sibling [prev-x] ← next-x
                BINOMIAL LINK( $\overset{y}{10}$,$\overset{z}{2}$)

                x ← next-x

        next-x ← nil
```





while nil ≠ nil (false)

Algorithm for BINOMIAL LINK

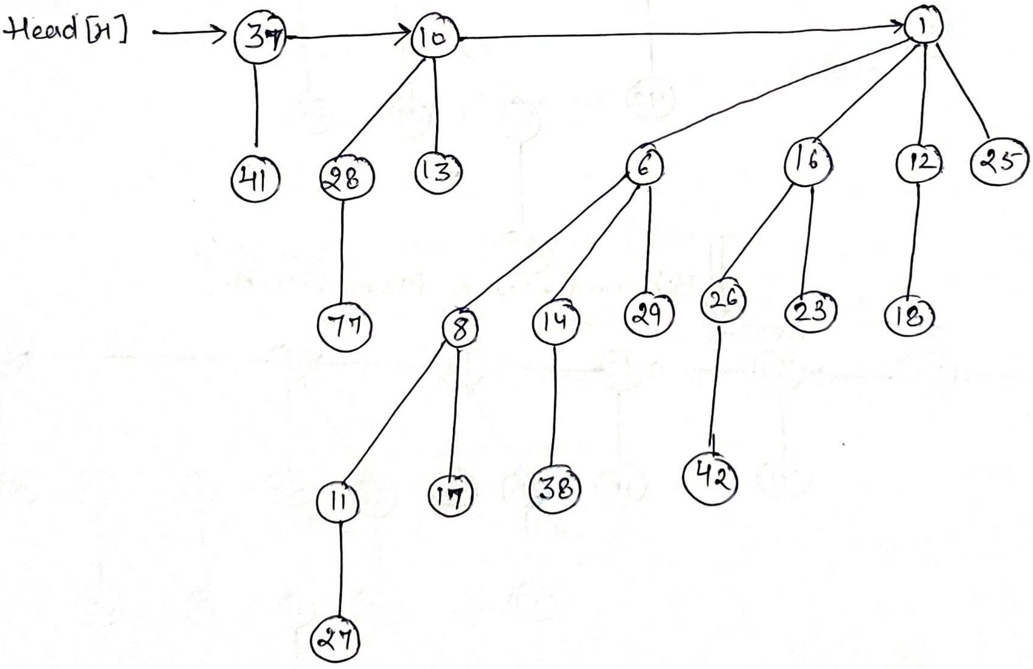
```
BINOMIAL-LINK(y, z)
1. p[y] ← z
2. sibling[y] ← child[z]
3. child[z] ← y
4. degree[z] ← degree[z] + 1
```

Extracting a Node with minimum key

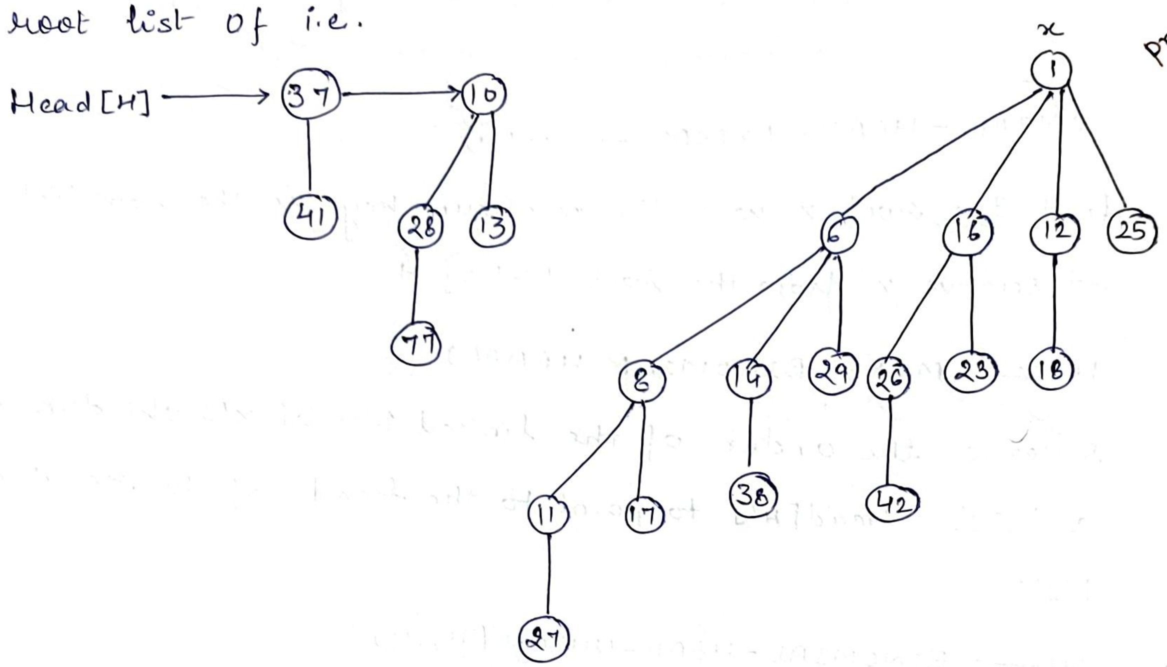
```
BINOMIAL - HEAP - EXTRACT - MIN (H)
1. find the root x with the minimum key in the root list of H,
   and remove x from the root list of H
2. H' ← MAKE BINOMIAL HEAP()
3. reverse the order of the linked list of x's children,
   and set head[H'] to point to the head of the resulting
   list
4. H ← BINOMIAL - HEAP - UNION (H, H')
5. return x
```

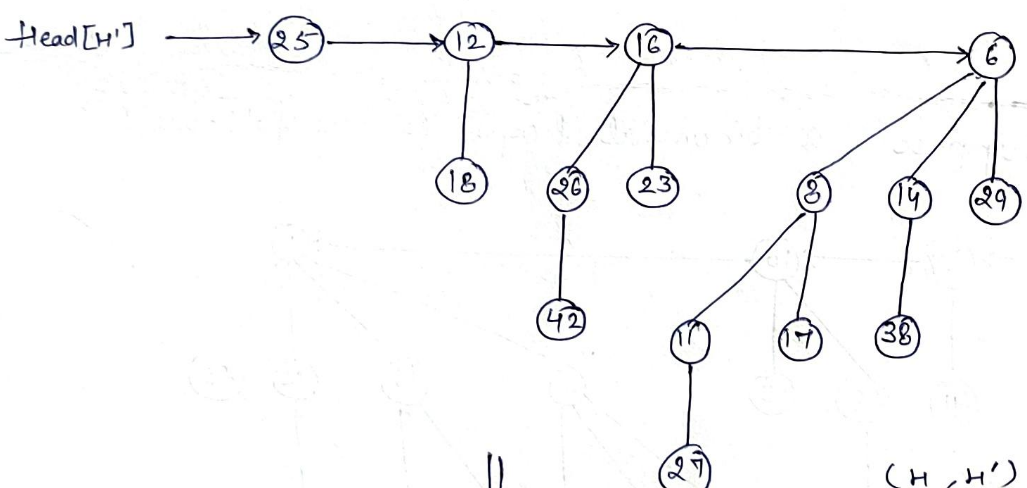
example.

Suppose a binomial heap H is as follows

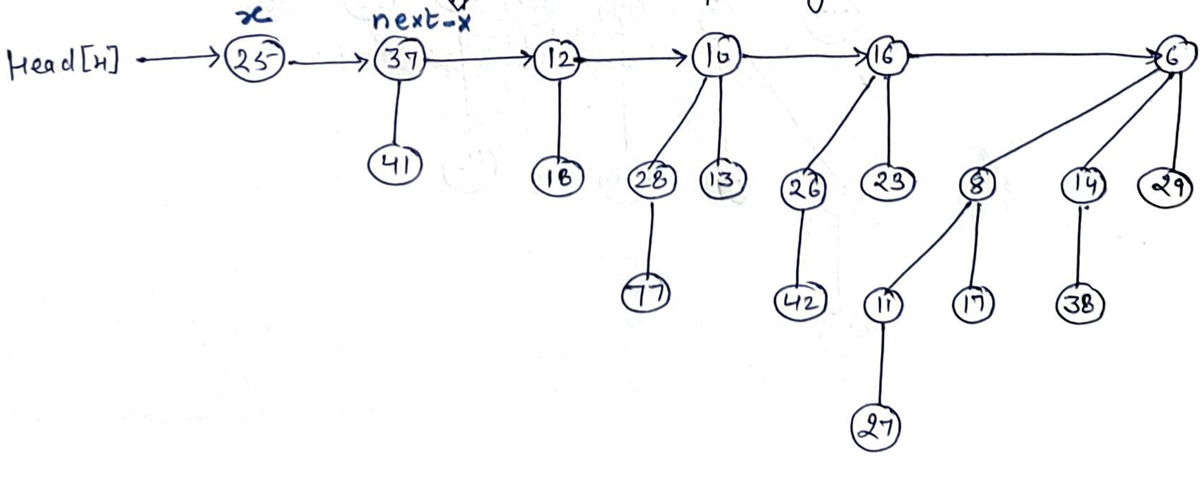


The root x with minimum key is 1. x is removed from root list of i.e.

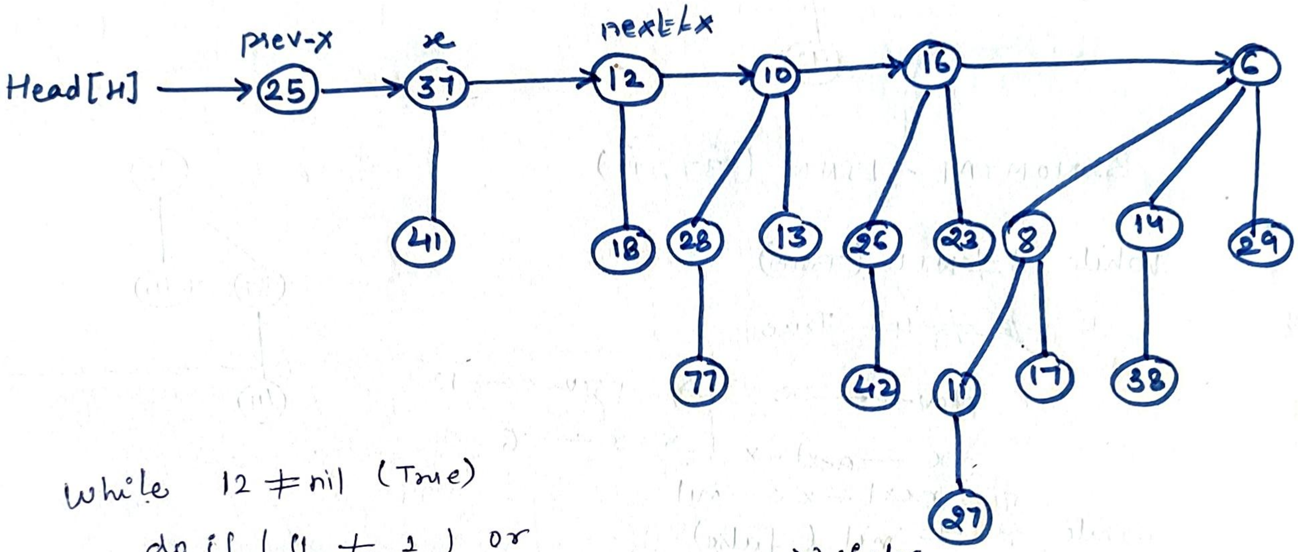




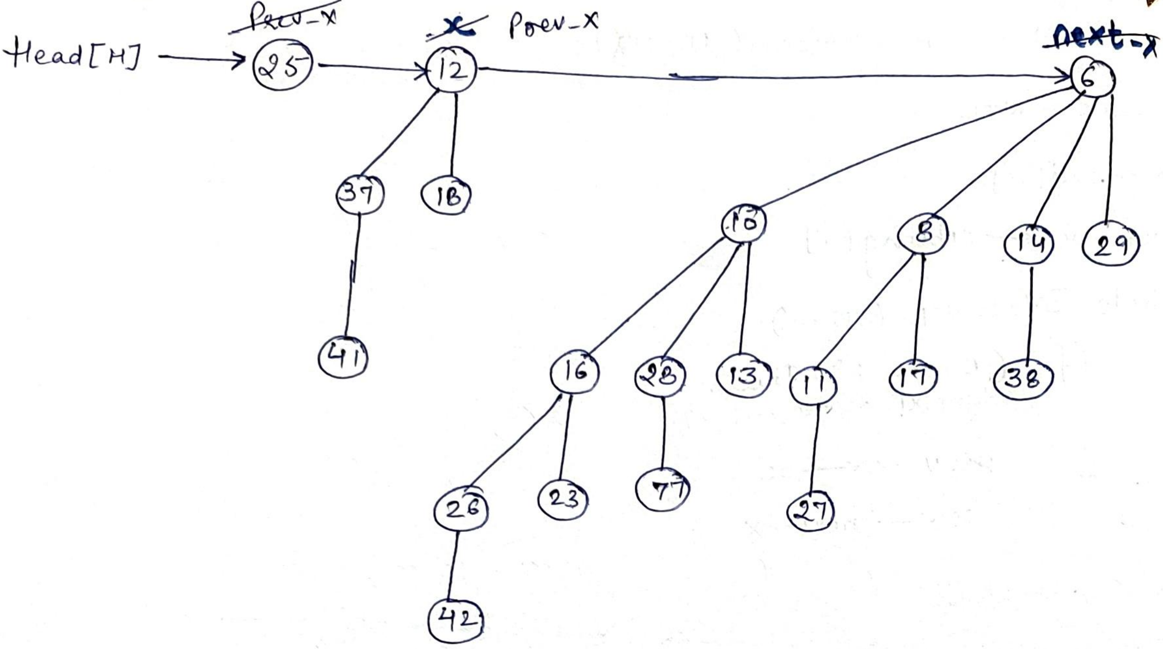
⇓ Binomial-Heap-Merge $\overset{(H, H')}{(H_1, H_2)}$

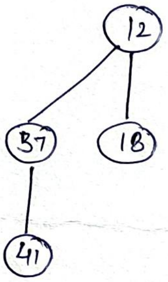


```
BINOMIAL-HEAP-UNION(H, H')
prev-x ← NIL
x ← Head[H]
next-x ← sibling[x]
while 37 ≠ NIL (True)
        if (0 ≠ 1) True.
           degree[x] degree[next-x]
           prev-x ← x
           x ← next-x
```



```
while 12 ≠ nil (True)
    do if (1 ≠ 1) or
               (10 ≠ nil and 2 = 1)) false
        else if (37 ≤ 12) false
            else if prev-x = NIL (false)
                else sibling[prev-x] ← next-x
                     BINOMIAL LINK(x, next-x)
                x ← next-x
        next-x ← sibling[x]
```





```
BINOMIAL-LINK ( $\overset{y}{37}$, $\overset{z}{12}$ )
while 6 ≠ NIL (True)
    if ( 2 ≠ 4) True
        prev-x ← x    ⇒    prev-x ← 12
        x ← next-x    ⇒    x ← 6
        next-x ← nil
while nil ≠ nil ( false)

    return H
```

Inserting a Node in Binomial Heap

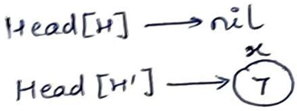
```
BINOMIAL-HEAP-INSERT( H, x)
1. H' ← MAKE-BINOMIAL-HEAP()
2. p[x] ← nil
3. child[x] ← nil
4. sibling[x] ← 0
5. degree [x] ← 0
6. Head[H'] ← x
7. H ← BINOMIAL-HEAP-UNION ( H, H')
```

xample A = { 7, 2, 4, 17, 11, 6, 8 }

Insert these keys into Binomial Heap.

Insert 7

BINOMIAL-HEAP-INSERT (H, 7)

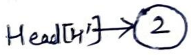


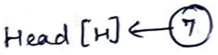
```
1. H' ← MAKE-BINOMIAL HEAP()
   P[x] ← nil
   child[x] ← nil
   sibling[x] ← 0
   degree [x] ← 0
   Head[H'] ← x
   H ← BINOMIAL-HEAP-UNION(H, H')
```

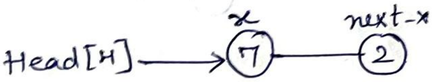
Insert 2

BINOMIAL-HEAP-INSERT(H, 2)

BINOMIAL-HEAP-UNION(H, H')



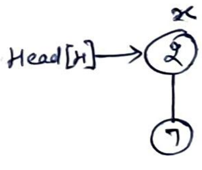




{ Binomial-Heap-Merge (H, H')

```
prev-x ← nil
x ← Head [H]
next-x ← sibling [x]
while (2 ≠ NIL)   True.
    if ((0 ≠ 0) ~~or~~ ((nil ≠ nil))) (False)
              0 = 0
    elseif 7 ≤ 2 (false)
      else if prev-x = NIL (True)
                Head [H] ← next-x
                BINOMIAL LINK (7, 2)
                x ← ~~nil~~ 2
```

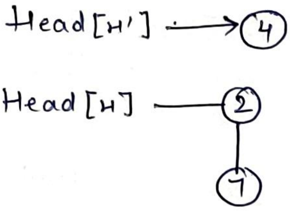
y z (over 7, 2)



next-x ← nil

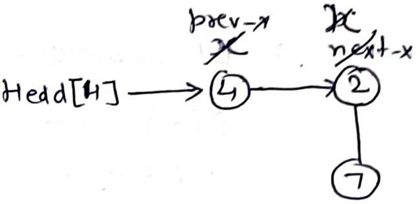
while nil ≠ nil (false)

Insert 4



⇓

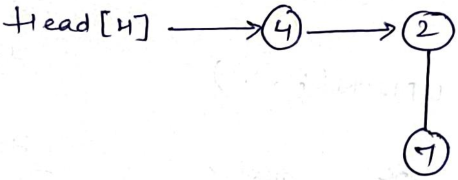
BINOMIAL - MERGE (H, H')



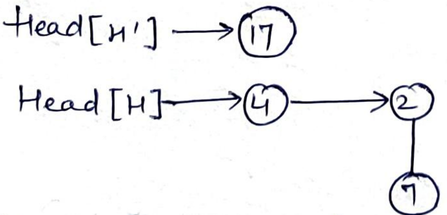
while 2 ≠ nil

```
do · if ( 0 ≠ 1) True
        prev-x ← 2
        x ← nil.
```

while nil ≠ nil (False)

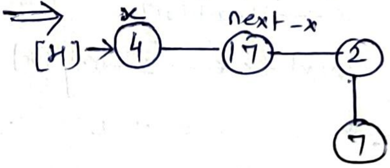


Insert 17

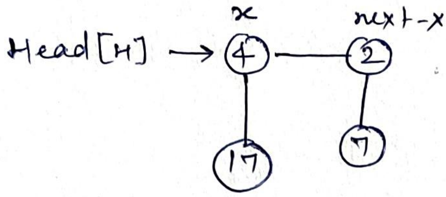


Binomial Merge ($\overset{H_1}{\overset{\uparrow}{H}}$, $\overset{H_2}{\overset{\uparrow}{H'}}$)

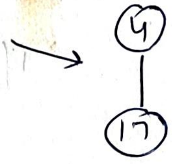
⟹



```
while 17 ≠ NIL (True)
    if ((0 ≠ 0) ((2 ≠ NIL) and (1 = 0)) False
    else if (4 ≤ 17) True
        sibling[x] ← sibling[next-x]
        BINOMIAL LINK($\underset{y}{17}$, $\underset{z}{4}$)
```



$\text{BINOMIAL-LINK}(\underset{y}{17},\underset{z}{4})$



while 2 ≠ nil (True)

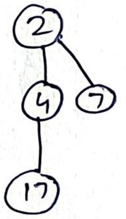
if ((1 ≠ 1) or((nil ≠ nil)) false

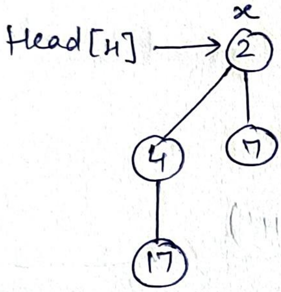
else if 4 ≤ 2 (False

else if prev-x = NIL (True)

Head[H] ← next-x

$\text{BINOMIAL LINK}(\underset{}{\overset{y}{4}},\overset{z}{2})$

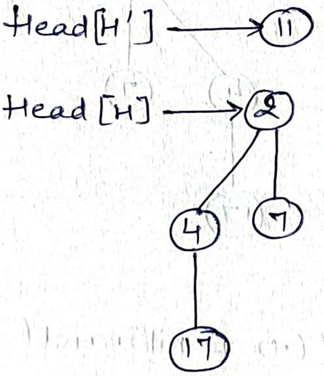




next-x ← nil

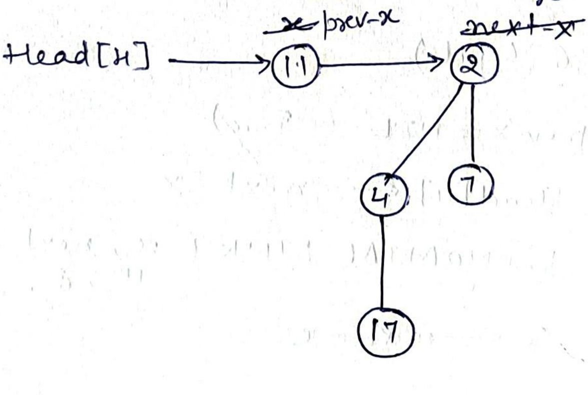
while nil ≠ nil (False)

Insert 11



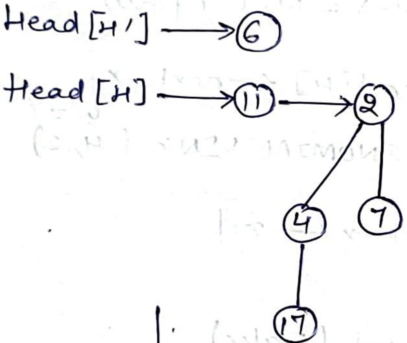
Binomial Merge(H, H')

⇒

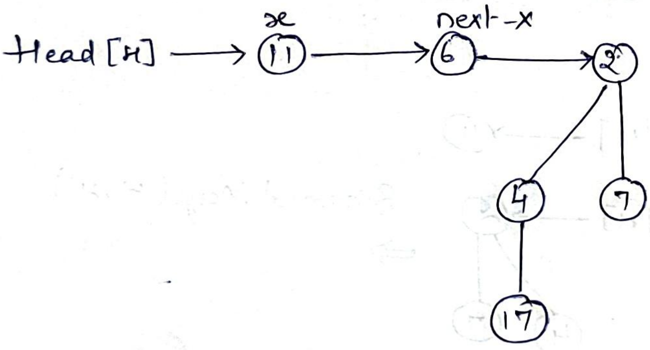


```
while 2 ≠ nil (True)
    if ( 0 ≠ 2) True.
        prev-x ← x            prev-x ← 11
        x ← next-x            x ← x ← 2 .
                              next-x ← nil
while nil ≠ nil (false)
```

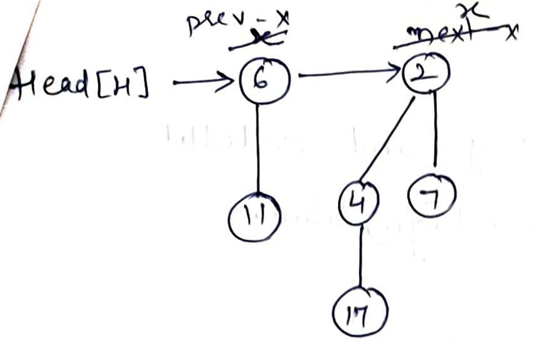
Insert 6

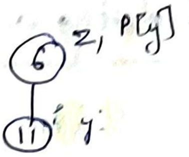


BINOMIAL - MERGE (H, H')



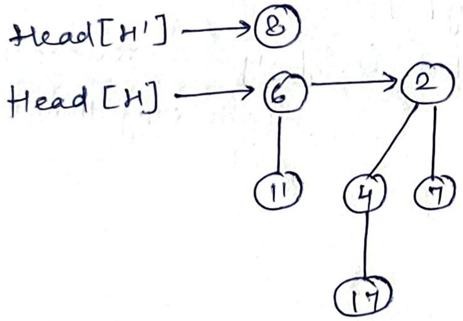
```
while 6 ≠ nil ( True)
    if ((0 ≠ 0) and ((0 ≠ nil) and ( 2 = 0)) (false)
    else if 11 ≤ 6 (false)
        elseif prev-x = NIL ( True)
            Head [H] ← next-x
            BINOMIAL LINK ( x, next-x)
                            11 , 6 .
                            y    z
            x ← next-x
```

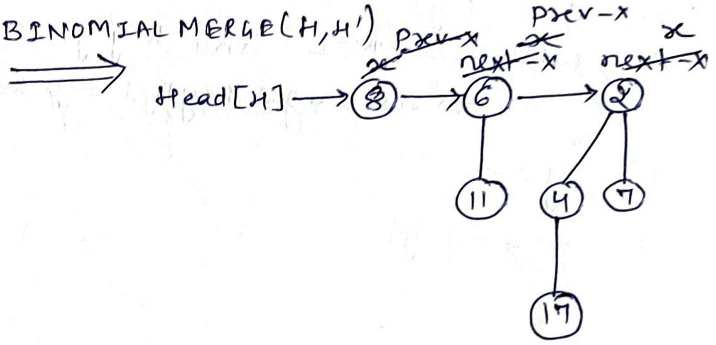




```
while 2 ≠ nil
    if (1 ≠ 2) True.
        prev-x ← 6
        x ← 2
        next-x ← nil
while nil ≠ nil (false)
```

Insert 8





```
while 6 ≠ nil (True)
    if (0 ≠ 1) (True)
        prev-x ← x
        prev-x ← 8
        x ← 6
        next-x ← 2
while 2 ≠ nil (True)
    if (1 ≠ 2) True
        prev-x ← 6
        x ← 2
        next-x ← nil
```

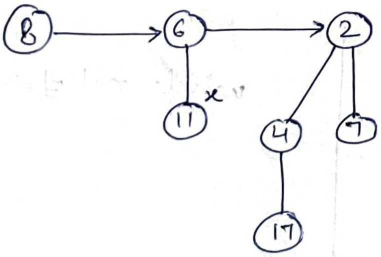
while nil ≠ nil

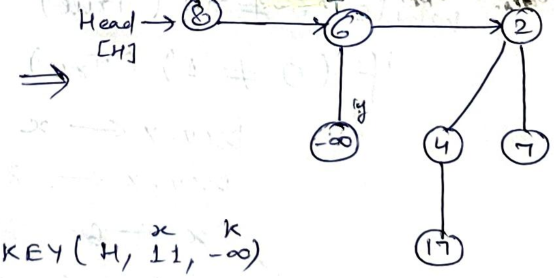
Deleting a key

It is easy to delete a node x's key and satellite information from binomial heap H is O(lgn) time.

```
BINOMIAL-HEAP-DELETE(H, x)
1. BINOMIAL-HEAP-DECREASE-KEY(H, x, -∞)
2. BINOMIAL-HEAP-EXTRACT-MIN(H)
```

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
1. if k > key[x]
2.     then error "new key is greater than current key"
3. key[x] ← k
4. y ← x
5. z ← p[y]
6. while z ≠ NIL and key[y] < key[z]
7.     do exchange key[y] ↔ key[z] { If y & z have other
                                      fields copy them too}
8. y ← z
9. z ← p[y]
```





BINOMIAL - HEAP - DECREASE - KEY(H, $\overset{x}{11}$, $\overset{k}{-\infty}$)

1. If -∞ > 11 (False)

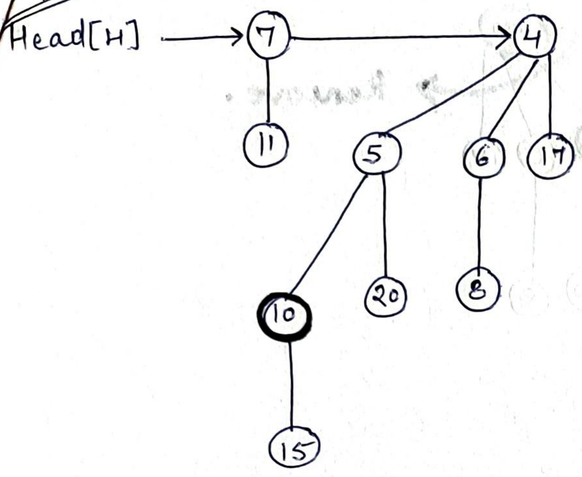
key[x] ← k

y ← x

z ← p[y]

ic for

uple

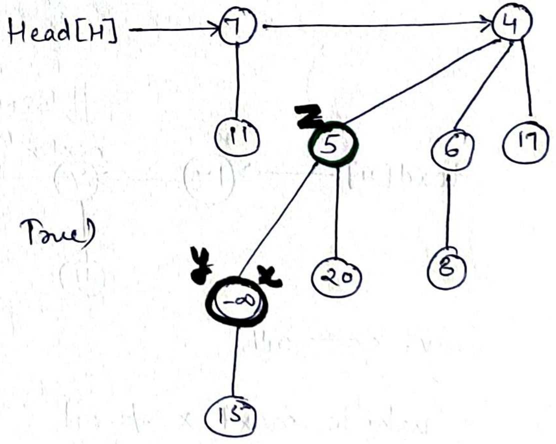


BINOMIAL-HEAP-DELETE(H, 10)

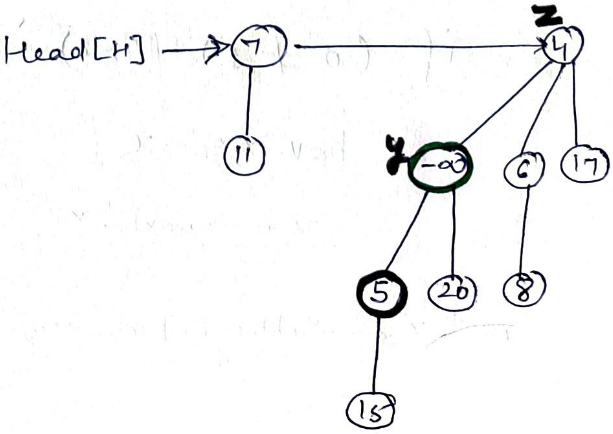
1. BINOMIAL-HEAP-DECREASE-KEY(H, 10, -∞)
2. BINOMIAL-HEAP-EXTRACT-MIN(H)

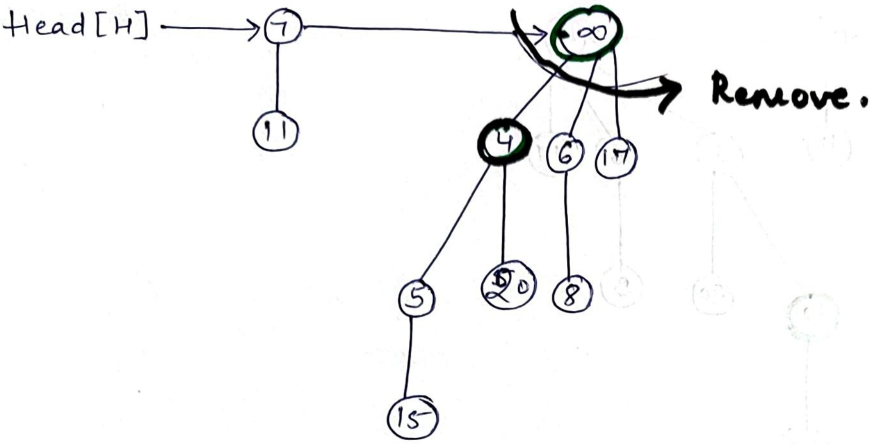
BINOMIAL-HEAP-DECREASE-KEY(H, 10, -∞)

```
If  -∞ > 10 ( false)

 key[x] ← -∞
 y ← x
 z ← p[y]

 while z ≠ NIL and -∞ < 5 ( True)
       key[y] ↔ key[z]
       y ← z
       z ← p[y]
```

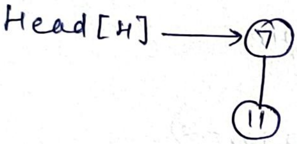


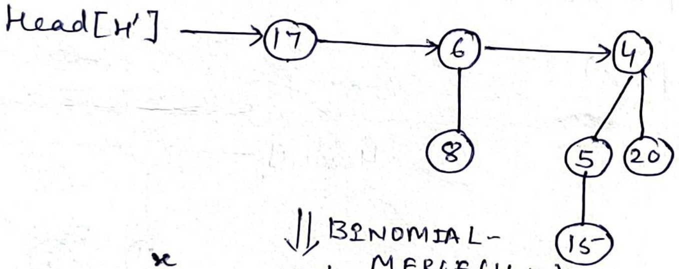
```
while  z ≠ NIL and
       -∞ < 4 ( True)
       key[y] ↔ key[z]
       y ← z
       z ← p[y]
```



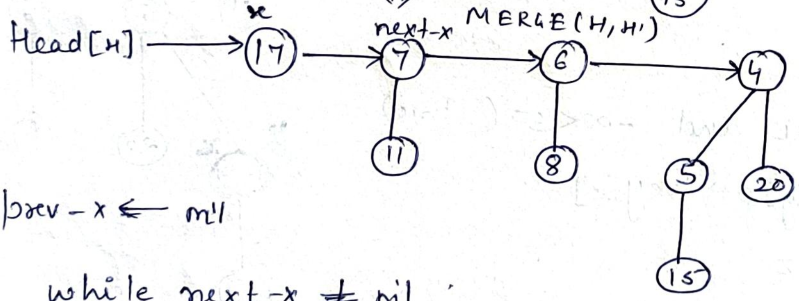


BINOMIN-HEAP - EXTRACT - MIN (H).

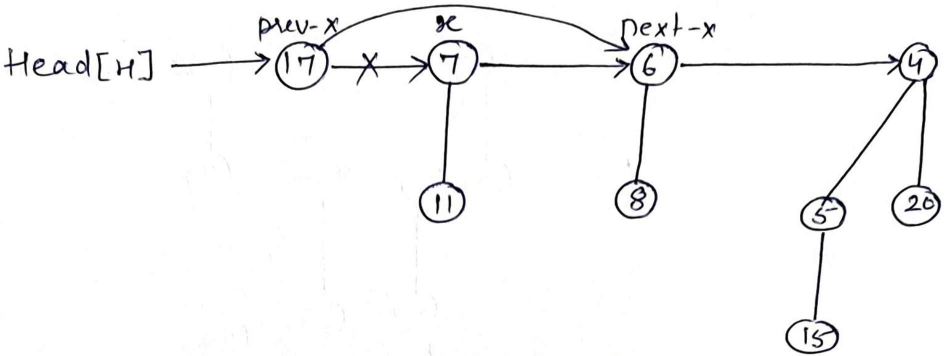




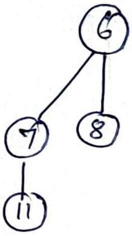
⇓ BINOMIAL-MERGE(H, H')

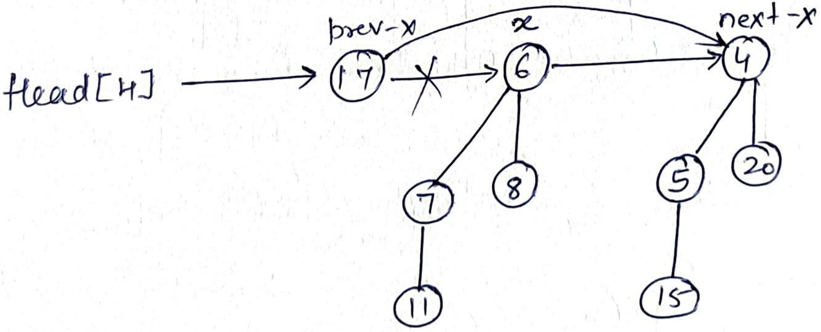


```
prev - x ← nil

    while next-x ≠ nil
              7 ≠ nil ( True)

          if (0 ≠ 1) (True)

                prev-x ← x

                  x ← next-x

      next-x ← Sibling[x]
```



```
while  next-x ≠ nil (True)
  if ((1 ≠ 1)  or (4≠nil and 2 = 1)) (false)

  else
      if 7 ≤ 6 (false)

      else if prev-x = nil (false)

          else
                Sibling[prev-x] ← next-x

                BINOMIAL LINK( x, next-x)

                x ← next-x

  next-x ← Sibling[x]
```



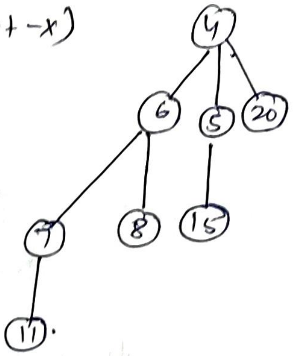


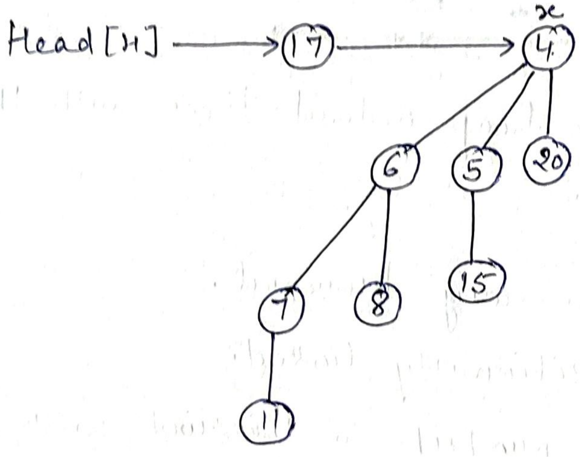
```
while  4 ≠ nil (True)
    if ((2 ≠ 2)  or ( nil ≠ nil  and        ) false.
  else if 6 ≤ 4 (false)
      else if 17 ≠ nil (false)
          else     sibling[prev-x] ← next-x
```

BINOMIAL LINK ($\overset{4}{\overset{6}{x}}$, $\overset{2}{\overset{4}{n}}$ext-x)

x ← next-x

next-x ← sibling[x]





next-x ← nil

while nil ≠ nil (false)

This is the final tree after Delete 10

Analysis of Binomial Heap Deletion

To delete a node x's key and satellite information from binomial heap H the Binomial-Heap-Delete procedure takes O(lgn) time.

UNIT-2

CHAPTER-4

FIBONACCI HEAPS

Fibonacci Heaps are min-heap ordered trees with the following characteristics.

1. The trees are not necessary binomial.
2. Siblings are bi-directionally linked.
3. There is a pointer min[H] to the root with the minimum key.
4. The root degrees are not unique.
5. The special attribute n[H] maintains the total number of nodes.
6. Each node has an additional Boolean label mark, indicating whether it has lost a child since the last time it was made a child of another node.

Structure of fibonacci Heaps:-

1. Like a binomial heap, a fibonacci heap is a collection of heap-ordered trees.

Unlike trees within binomial heaps, which are Ordered, trees within Fibonacci heaps are rooted but unordered.