Binomial Heaps

Binary Heap

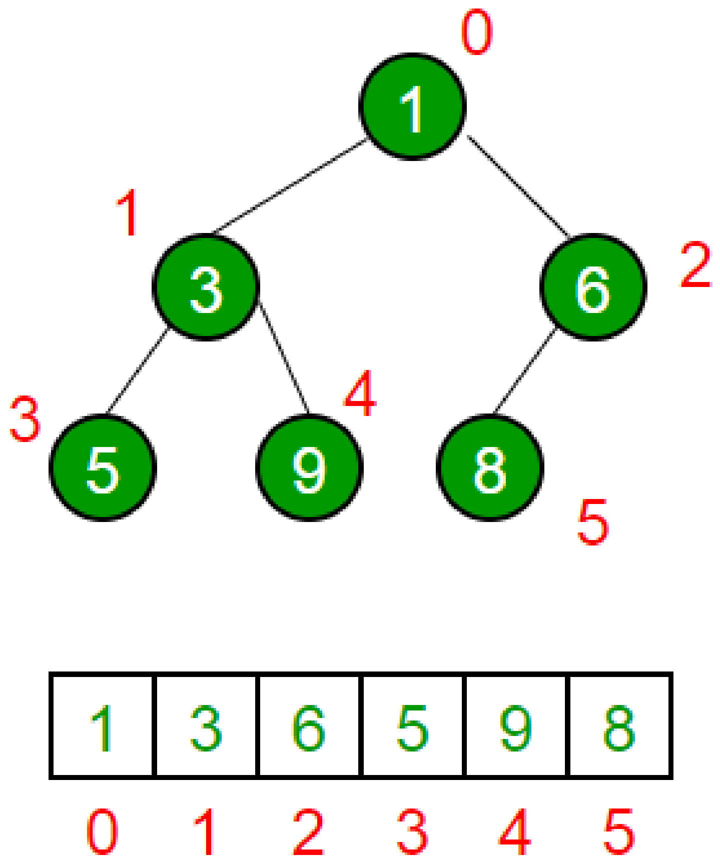
A Binary Heap is a Binary Tree with following properties.

1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

- A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.
- The root element will be at A[0]. Given the index of ith node:

A[(i-1)/2]	Returns the parent node
A[(2*i)+1]	Returns the left child node
A[(2*i)+2]	Returns the right child node


0
1
1
3
6
2
3
5
9
4
8
5
1 3 6 5 9 8
0 1 2 3 4 5

Operations on Min Heap:

1) minimum(): It returns the root element of Min Heap.
2) extractMin(): Removes the minimum element from MinHeap.
3) decreaseKey(): Decreases value of key.
4) insert(): Inserting a new key.
5) delete(): Deleting a key.

Why Binomial Heaps?

Binomial Heap presents data structures known as **mergeable heaps**. The worst case for the UNION operation is considerably much better than other data structures.

Procedure	Binary Heap Worst Case	**Binomial Heap (worst-case)**	Fibonacci heap (amortized)
Make Heap	$\Theta(1)$	$\boldsymbol{\Theta(1)}$	$\Theta(1)$
Insert	$\Theta(\lg n)$	**$O(\lg n)$**	$\Theta(1)$
Minimum	$\Theta(1)$	**$O(\lg N)$**	$\Theta(1)$
Extract-Min	$\Theta(\lg n)$	$\boldsymbol{\Theta(\lg n)}$	$O(\lg n)$
Union	$\Theta(n)$	**$O(\lg n)$**	$\Theta(1)$
Decrease-Key	$\Theta(\lg n)$	$\boldsymbol{\Theta(\lg n)}$	$\Theta(1)$
Delete	$\Theta(\lg n)$	$\boldsymbol{\Theta(\lg n)}$	$\Theta(\lg n)$

Plan

- Binomial Trees
- Binomial Heaps
- Representing Binomial Heaps
- Binomial Heap Operations
- Applications

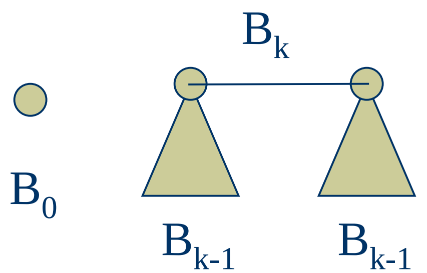
Binomial Trees

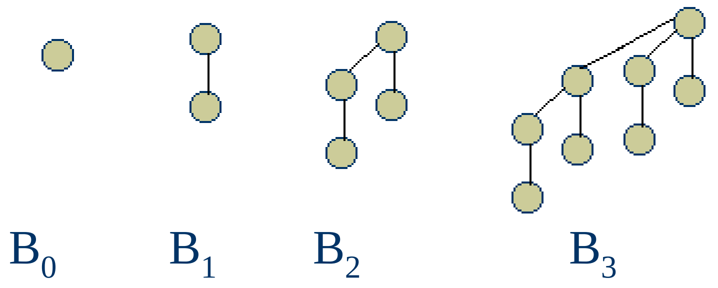
- A Binomial tree can be defined recursively where:

  A binomial tree B_k consists of two B_{k-1} binomial trees that are linked together. The root of one is the leftmost child of the root of the other.

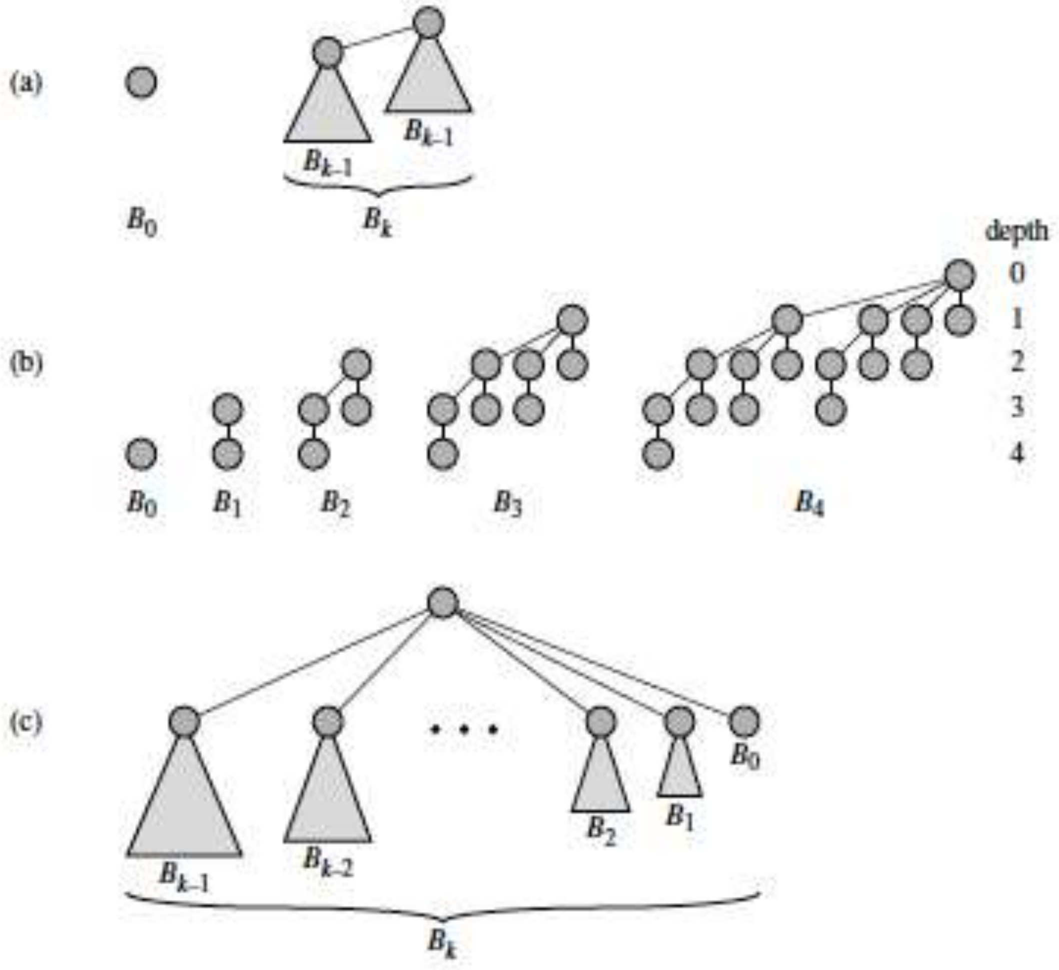
- Properties of a binomial tree are given by the following lemma:

  1. there are 2^k nodes
  2. the height of the tree is k
  3. there are exactly ${}^kC_i$ nodes at depth i for i = 0, 1, …, k and
  4. the root has degree k, which is greater than that of any other node





Binomial Trees



Binomial Heaps

A **binomial heap,** H, is a set of binomial trees that satisfies the following binomial-heap properties.

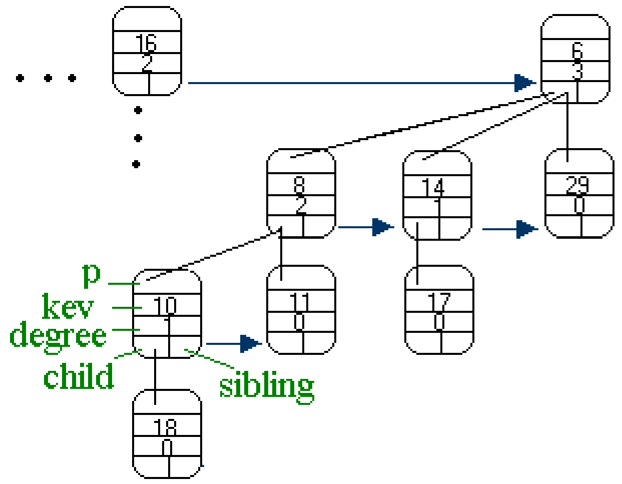
1. Each binomial tree in H is **min-heap-ordered**: the key of a node is greater than or equal to the key of its parent

   - the root of a heap-ordered tree contains the smallest key in a given sub-tree

2. There is at most one binomial tree in H whose root has a given degree

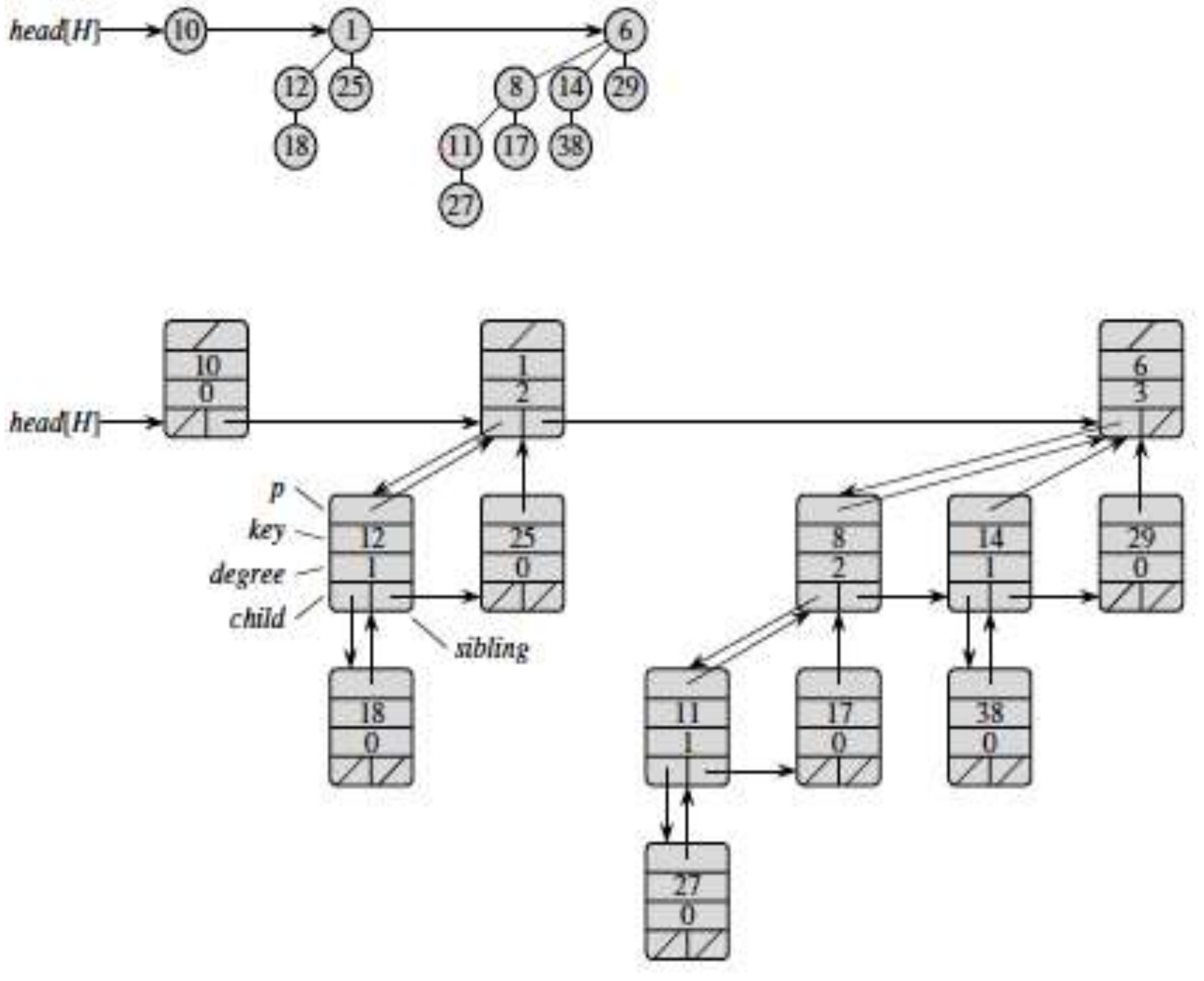
   - an n-node binomial heap H consists of at most log(*n*) + 1 binomial trees.

Representing Binomial Heaps



x:	key
head[x]:	pointer to the root (if NIL, binomial heap is empty)
p[x]:	pointer to the parent(if x is the root, then NIL)
child[x]:	leftmost child (if node has no children,then NIL)
sibling[x]:	sibling of x immediately to its right (rightmost child of parent, then NIL)

Representing Binomial Heaps



- The roots of the binomial trees within a binomial heap are organized in a linked list, which we refer to as the root list.
- The degrees of the roots strictly increase as we traverse the root list.
- The sibling field has a different meaning for roots than for non-roots. If x is a root, then sibling[x] points to the next root in the root list.
- A given binomial heap H is accessed by the field head[H],. If binomial heap H has no elements, then head[H] = NIL.

Make-Binomial-Heap

Make-Binomial-Heap()

- Allocates and returns an object H, where head[H] = NIL
- Running time: **O(1)**

Binomial-Heap-Minimum

-returns a pointer to the node with the minimum key in an n-node binomial heap

Binomial-Heap-Minimum(H)

```
1.    $y \leftarrow NIL$
2.    $x \leftarrow head[H]$
3.    $min \leftarrow \infty$
4.    while $x \neq NIL$
        do if $key[x] < min$
             then $min \leftarrow key[x]$
                  $y \leftarrow x$
           $x \leftarrow sibling[x]$
return $y$
```

Running time: *O(log n)*

Binomial-Heap-Union

- The BINOMIAL-HEAP-UNION procedure repeatedly links binomial trees whose roots have the same degree.

- The following procedure links the B_{k-1} tree rooted at node y to the B_{k-1} tree rooted at node z; that is, it makes z the parent of y. Node z thus becomes the root of a B_k tree.

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

Binomial-Heap-Union: Merging Heaps

Binomial-Heap-Union(H_1, H_2)

```
1.   H ← Make-Binomial-Heap()
2.   head[H] ← Binomial-Heap-Merge($H_1$, $H_2$)
3.   free the objects $H_1$ and $H_2$ but not the lists they point to
4.   if  head[H] = NIL
5.           then return  H
6.   prev-x ← NIL
7.   x ← head[H]
8.   next-x ← sibling[x]
9.   while next-x $\neq$ NIL
10.      do if (degree[x] $\neq$ degree[next-x]) or
     (sibling[next-x] $\neq$ NIL and degree[sibling[next-x]] = degree[x])
11.    then prev-x ← x
12.         x ← next-x
13.    else if key[x] <= key[next-x]
14.              then sibling[x] ← sibling[next-x]
•                     Binomial-Link(next-x, x)
1.               else if prev-x = NIL
2.                        then head[H] ← next-x
3.                        else sibling[prev-x] ← next-x
4.                     Binomial-Link(x, next-x)
5.                     x ← next-x
6.     next-x ← sibling[x]
7.   return H
```

Stage 1

Stage 2

Stage 3

Stage 4

Case 1 & 2

Case 3

Case 4

Binomial-Heap-Union

Stage 1: Start merging root lists of binomial heaps H_1 and H_2 into a single root list H. The root list is strictly in increasing degree

Stage 2: Check for empty heap

Stage 3: Initialize prev-x, x and next-x

Stage 4: decide which sub-trees to link to each other depending types of cases:

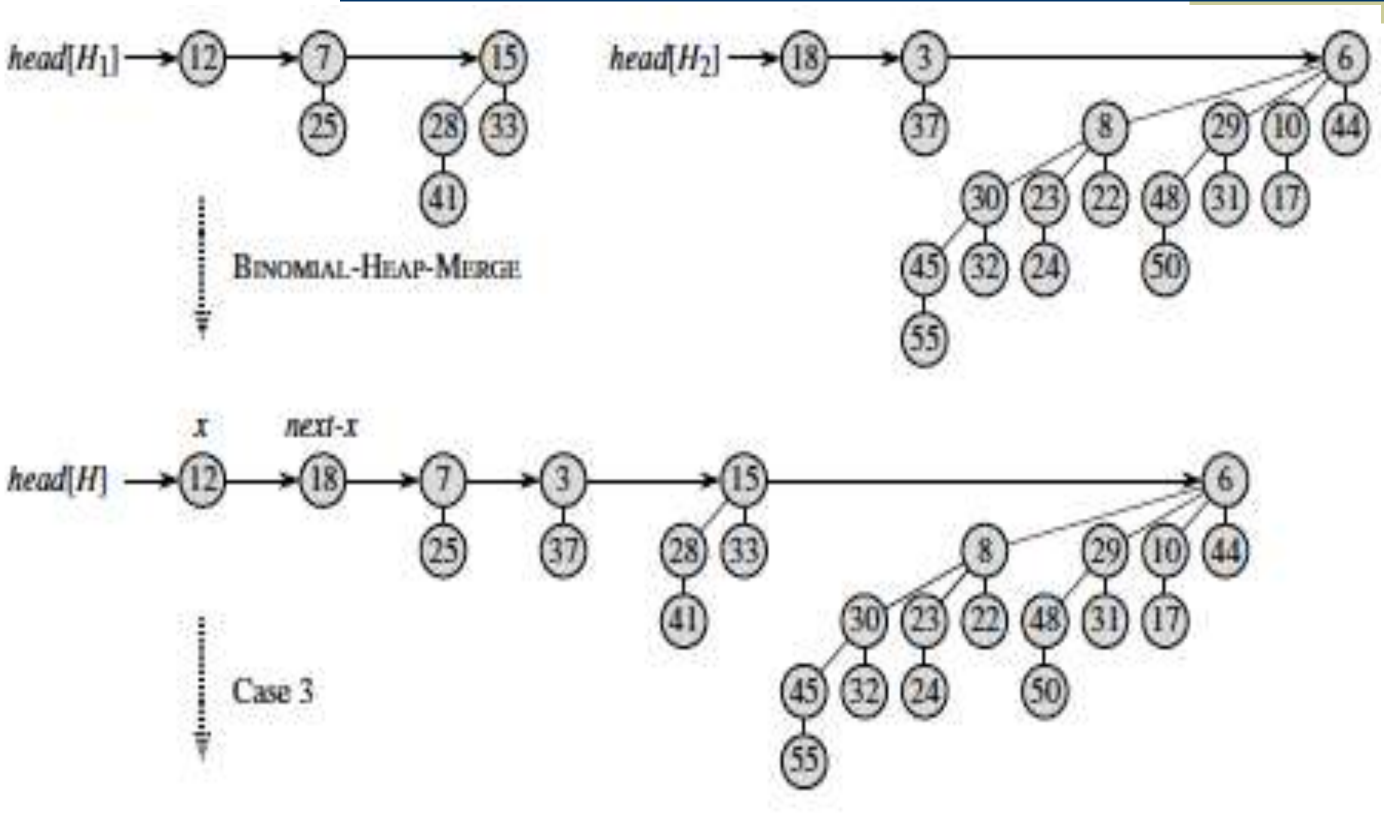
Case 1: degree[x] $\neq$ degree[next-x]

Case 2: degree[x] = degree[next-x] =degree[sibling[next-x]]

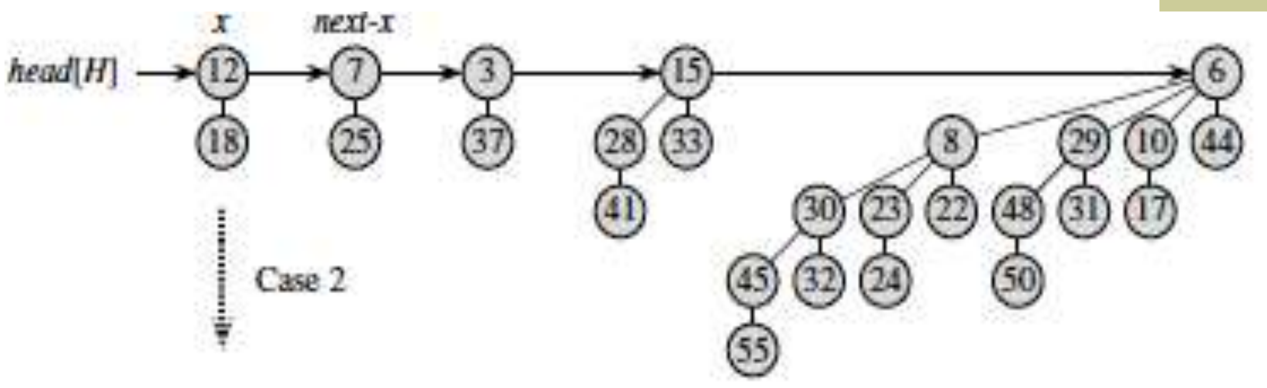
Case 3 & 4: degree[x] = degree[next-x] $\neq$ degree[sibling[next-x]]

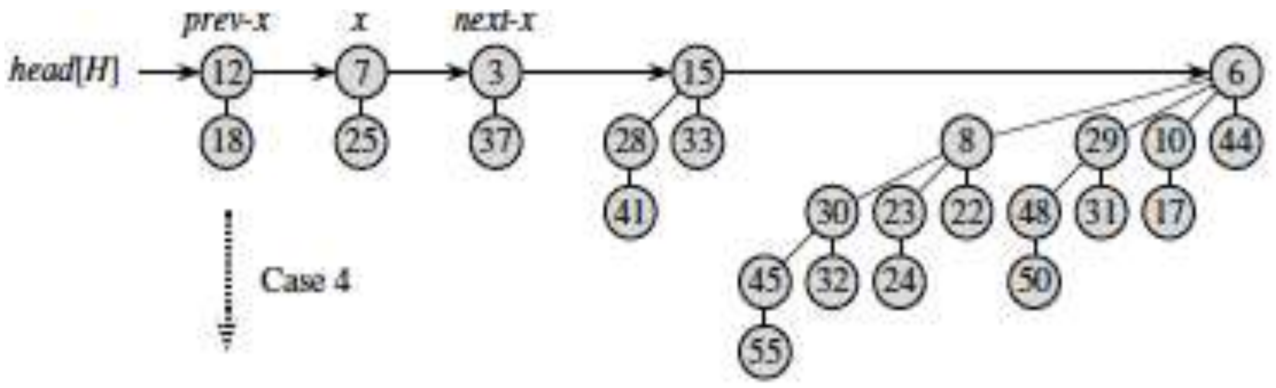
Running time for examining the sum of roots of heaps H_1 and H_2 ,
$\log n_1 + \log n_2$: *O(log n)*

Binomial-Heap-Union

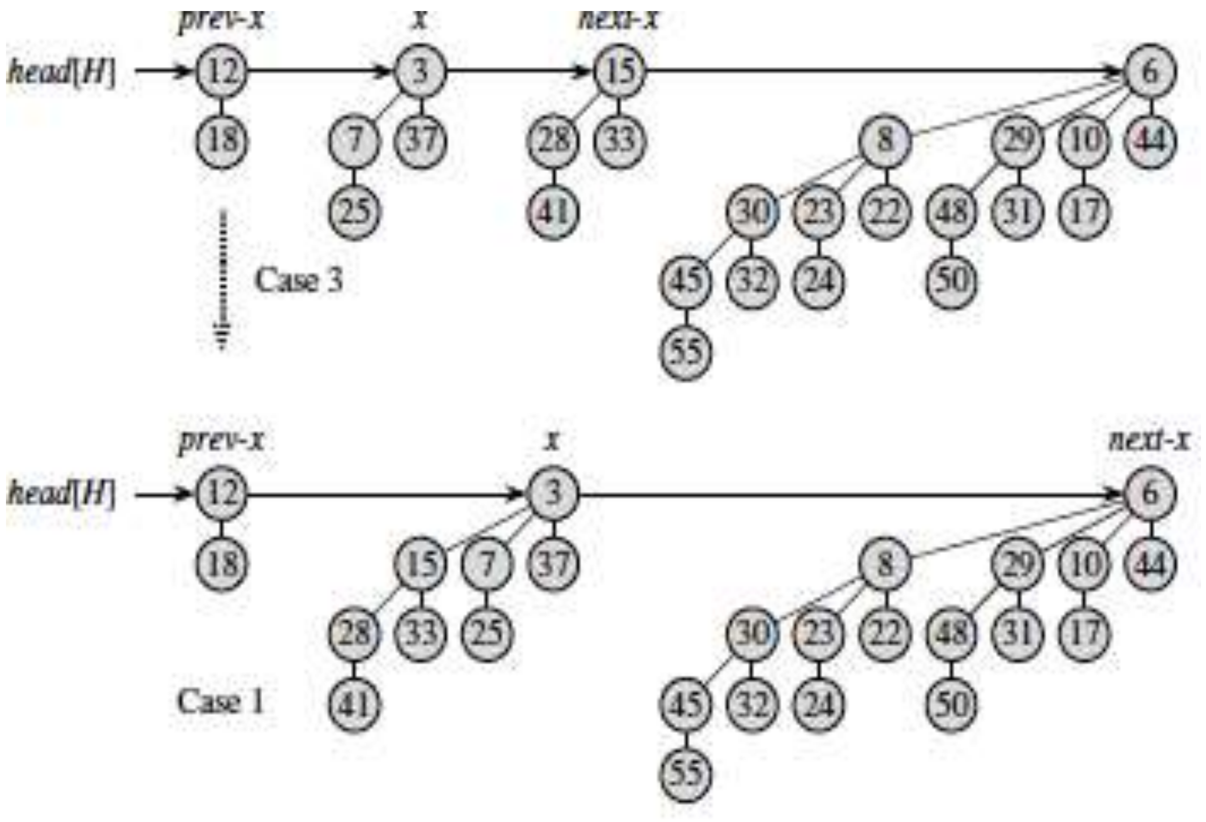


Binomial-Heap-Union

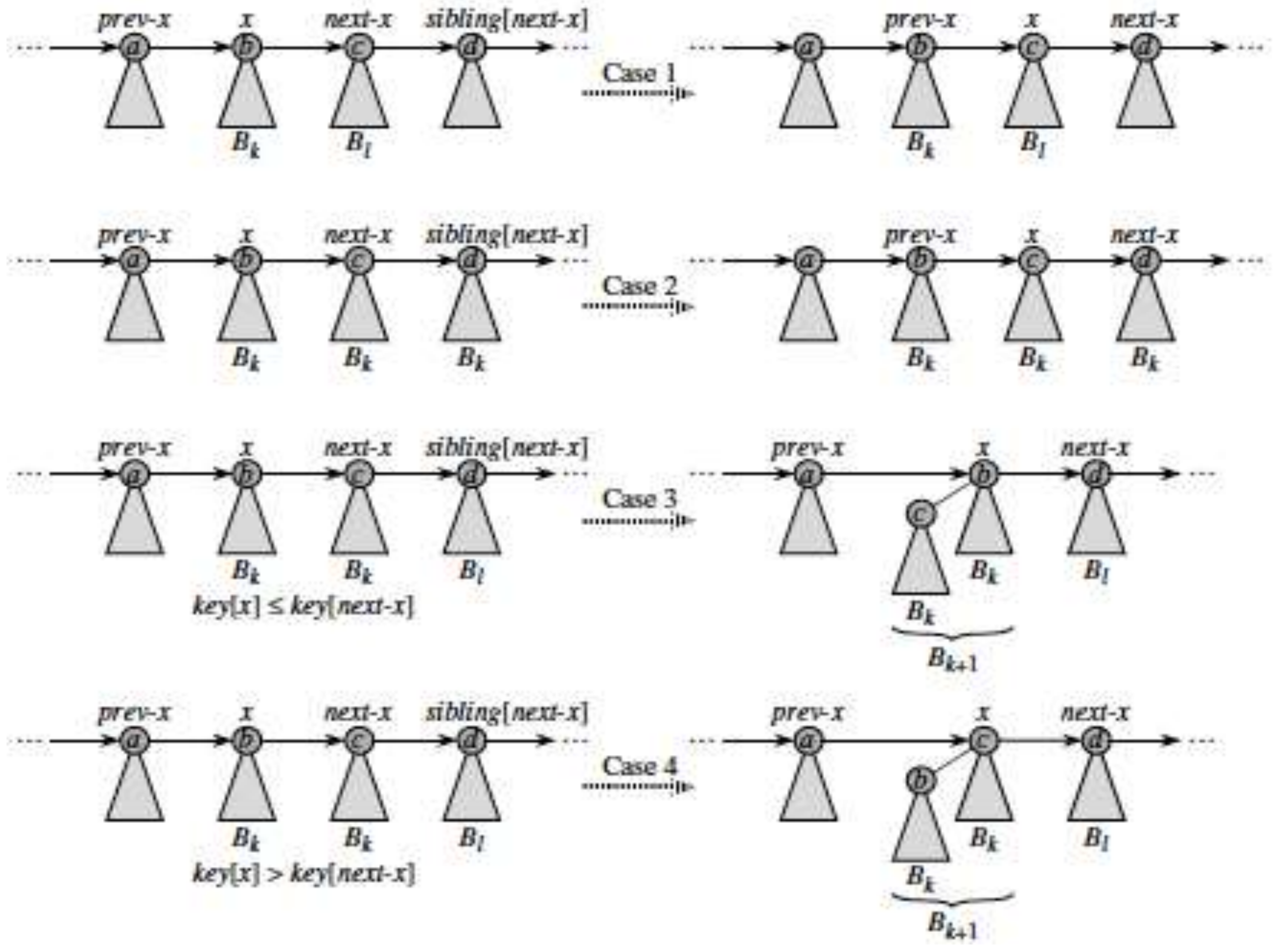




Binomial-Heap-Union



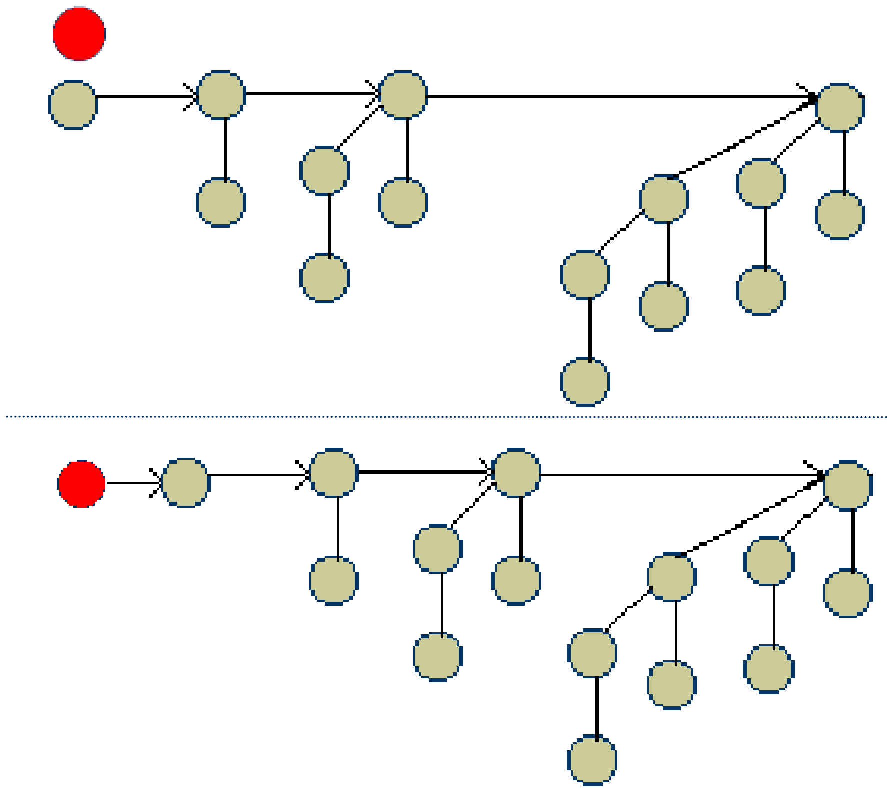
Binomial-Heap-Union



Binomial-Heap-Insert

-inserts node x into binomial heap H

```
Binomial-Heap-Insert(H, x)
1.      H’ ← Make-Binomial-Heap()
2.      p[x] ← NIL
3.      child[x[  ← NIL
4.      sibling[x] ← NIL
5.      degree[x] ← 0
6.      head[H’] ← x
7.      H ← Binomial-Heap-Union(H, H’)
```



Running time:

Make-Binomial-Heap() = O(1)

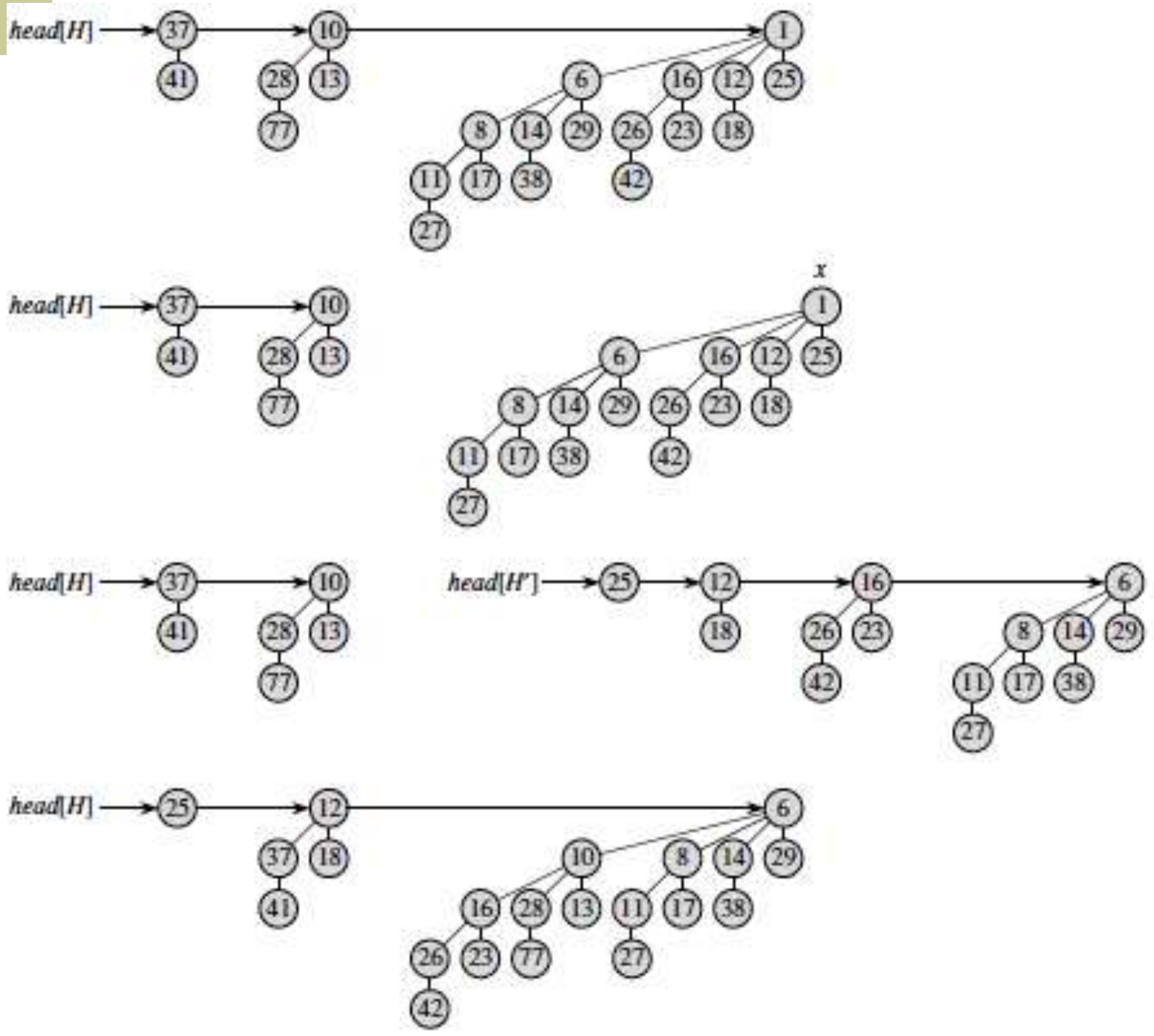
Binomial-Heap-Union = O(logn)

Binomial-Heap-Extract-Min

```
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
```

Running time : O(log n)

Binomial-Heap-Extract-Min



Binomial-Heap-Decrease-Key

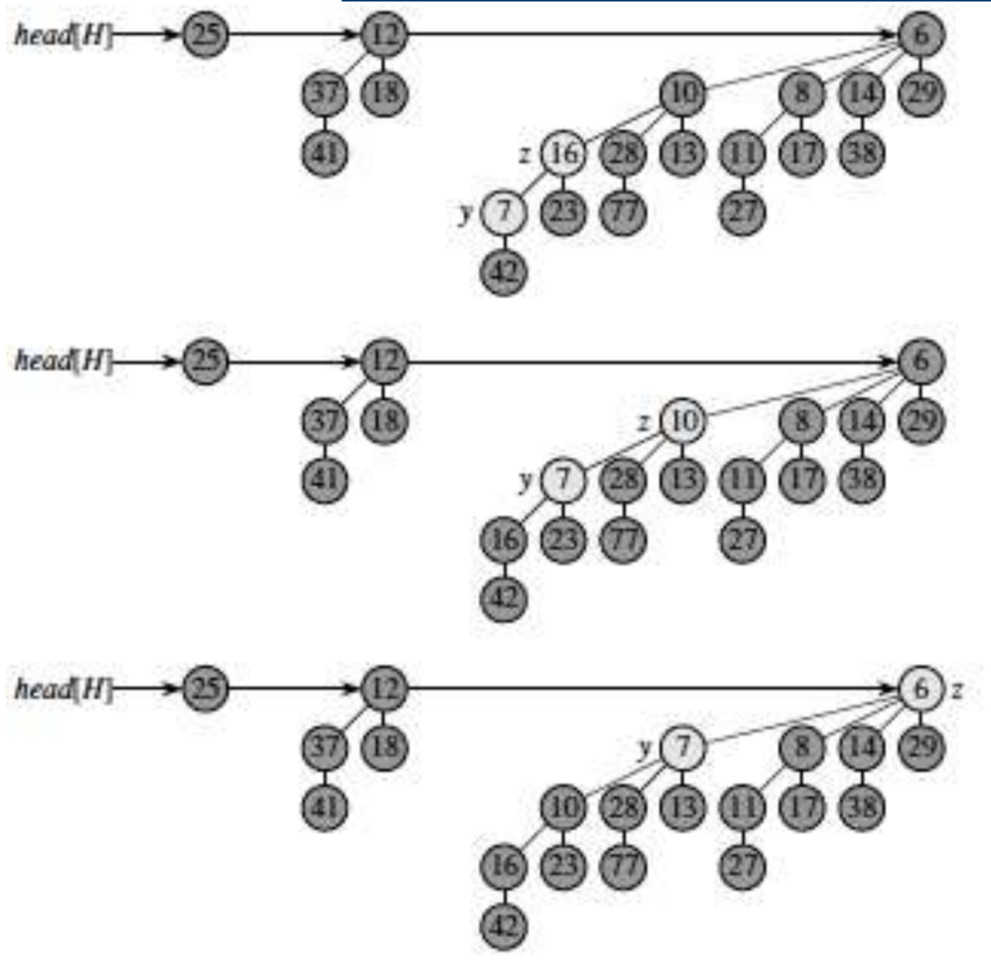
-decreases the key of a node x in a binomial heap H to a new value k

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]
8.      y ← z
9.      z ← p[y]
```

Running time : *O(log n)*

Binomial-Heap-Decrease-Key



Binomial-Heap-Delete-Key

The following implementation assumes that no node currently in the binomial heap has a key of $-\infty$.

```
BINOMIAL-HEAP-DELETE(H, $x$)
1  BINOMIAL-HEAP-DECREASE-KEY(H, $x$, $-\infty$)
2  BINOMIAL-HEAP-EXTRACT-MIN(H)
```

Running time : *O(log n)*

Applications

Mergeable heaps are used:

- In priority queues:

  -scheduling user tasks on a network.

- to find minimum spanning trees of undirected graphs

  -highways

  -computer networks

  -telephone lines

  -television cables