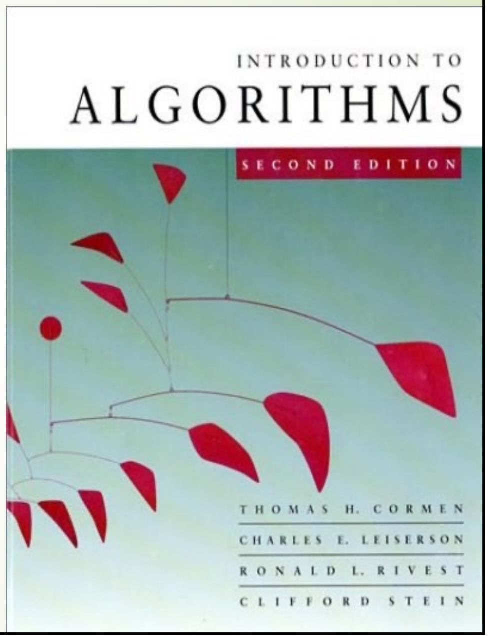
BINOMIAL HEAPS

SUBJECT-CODE : KCS-503

Dr. Ragini Karwayun



1

CONTENT

- Definitions
  - Binomial trees
  - Binomial heaps
- Operations on Binomial Heaps
  - Creating a new heap
  - Finding the minimum key
  - Union
  - Insert a node
  - Extract minimum
  - Deleting a key
  - Decreasing a key

2

OPERATIONAL COSTS ON HEAPS

Operation	Linked List	Heaps		
		Binary	Binomial	Fibonacci *
make-heap	1	1	1	1
insert	1	log N	log N	1
find-min	N	1	log N	1
delete-min	N	log N	log N	log N
union	1	N	log N	1
decrease-key	1	log N	log N	1
delete	N	log N	log N	log N
is-empty	1	1	1	1

INTRODUCTION

- Heap is a data structure that has enormous utility.
  - We saw how a binary heap was used to design an efficient heapsort algorithm.
- Heaps are used to implement priority queues.
  - Priority queue maintains elements according to their keys.
  - Min-priority queue supports the following operations:
    - Insert(Q,x) – insert an element x
    - Minimum(Q) – returns the element with the smallest key
    - Extract-Min(Q) – removes and return the min element
    - Decrease-Key(Q,x,k) – decrease the value of the key of x to k.
  - For example, priority queues are used to schedule jobs on shared computer resources.

INTRODUCTION

- Mergeable heaps, support the following operations:
  - Make-Heap() creates and returns an empty heap.
  - Insert(H,x) inserts a node x with a key into a heap H.
  - Minimum(H) returns a pointer to the node whose key is minimum.
  - Extract-Min(H) deletes the minimum node and returns its pointer.
  - Union(H1, H2) creates and returns a new heap that contains all nodes from H1 and H2.
  - DECREASE-KEY(H, x, k) - assigns to node x within heap H the new key value k, which is assumed to be no greater than its current key value.1
  - DELETE(H, x) - deletes node x from heap H.
- Binary heaps work well if we don't need the Union operation that takes Θ(n) time.
  - The Union on binomial heaps takes O(lg n) time.

5

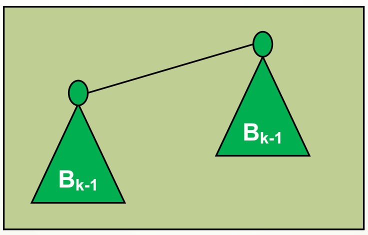
BINOMIAL TREES

- An ordered tree is a rooted tree, where the order of children nodes matters.
- **The binomial tree B_k is an ordered tree defined recursively**:
  - B_0 consists of a single node.

B_0

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

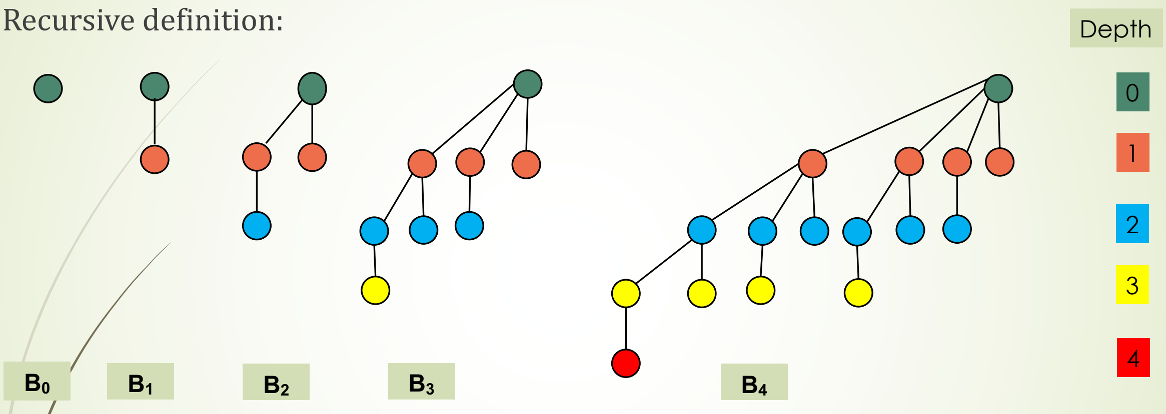
B_k



6

BINOMIAL TREE.

- Recursive definition:



7

Binomial Trees: Properties

- For the binomial tree B_k,

1. There are 2^k nodes,
2. The height of the tree is k
3. There are exactly $\binom{k}{i}$ nodes at depth i for $i=0,1,...,K$
4. The root has degree k, which is greater than any other node; moreover if children of the root are numbered left to right by $k-1,k-2,...,0$, the child i is the root of subtree b_i.

8

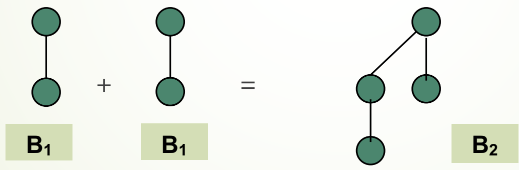
Binomial Trees Properties - Proof

- The proof is by induction on k. Verifying all properties for B_0 is trivial. For the inductive step, we assume that the lemma holds for B_{k-1}.

Property 1 : There are 2^k nodes in B_k tree.

- The Binomial tree B_k consists of two copies of B_{k-1}, hence B_k has

$$2^{k-1} + 2^{k-1} = 2^k \text{ nodes}$$

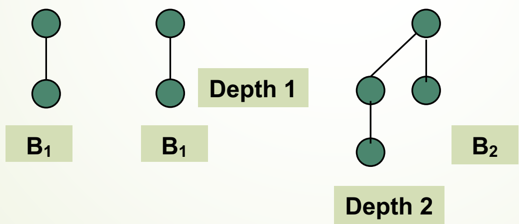


9

Binomial Trees Properties - Proof

Property 2: The height of the tree is k

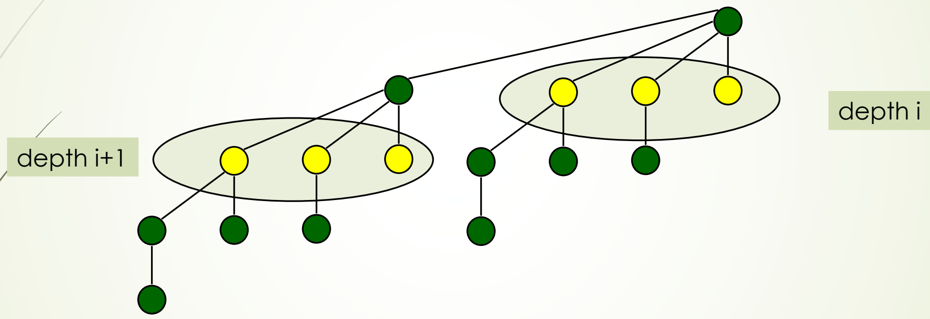
- Because of the way in which two copies of B_{k-1} are linked together to form B_k, the maximum depth of a node in B_k is one greater than the maximum depth in B_{k-1}.
- the height of B_k is increased by one compared to B_{k-1}. Hence its maximum depth is $k- 1 + 1 = k$



10

Binomial Trees Properties - Proof

Property 3: There are exactly $\binom{k}{i}$ nodes at depth i for i=0,1,...,K



11

Binomial Trees Properties - Proof

Property 3: There are exactly $\binom{k}{i}$ nodes at depth i for i=0,1,...,K

- Let D(k,i) the number of nodes at depth i of binomial tree B_k. Since B_k is composed of two copies of B_{k-1} linked together, a node at depth i in B_{k-1} appears in B_k once at depth i and once at depth i+1.
- In other words, the number of nodes at depth i in B_k is the number of nodes at depth i in B_{k-1} plus the number of nodes at depth i-1 in B_{k-1}.
- Thus D(k,i) = D(k-1,i) + D(k-1,i-1)

12

Binomial Trees Properties - Proof

Property 3: There are exactly $\binom{k}{i}$ nodes at depth i for i=0,1,...,K

So,

$$
\begin{aligned}
& D(k,i) \\
&= D(k-1,i) + D(k-1,i-1) \\
&= \binom{k-1}{i} + \binom{k-1}{i-1} \quad \text{, by ind. hyp.} \\
&= \frac{(k-1)!}{i!(k-1-i)!} + \frac{(k-1)!}{(i-1)!(k-i)!} \\
&= \frac{(k-1)!(k-i) + i(k-1)!}{i!(k-i)!} \\
&= \frac{k! - i(k-1)! + i(k-1)!}{i!(k-i)!} \\
&= \binom{k}{i}
\end{aligned}
$$

Binomial Trees Properties - Proof

4. The only node with the greater degree in B_k than B_{k-1} is the root, which has one more child.

Root degree of B_k = 1 + root degree of B_{k-1}
= 1 + k–1
= k

- By inductive hypothesis children of B_{k-1} are roots of B_{k-2}, B_{k-3}, ... , B_0. Once B_{k-1} is linked to B_{k-1} from the left, the children of the resulting root are B_{k-1}, B_{k-2}, ... , B_0.

Corollary -19.2

- **The maximum degree of any node in an n-node binomial tree is lg n.**
- **<u>Proof.</u>**
- Property 1 says :: There are 2^k nodes in B_k tree.
- Property 4 says : Root degree of B_k = k
- The root has the maximum degree k, and n = 2^k

Therefore k = lg n.

15

Binomial Heaps

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

1. Each binomial tree in H obeys the min-heap property:
   - the key of a node >= the key of its parent
   - this ensures that the root of the min-heap-ordered tree contains the smallest key.
2. For any k >= 0, there is at most one binomial tree in H whose root has degree k.
   - this implies that an n-node binomial heap H consists of at most (lg n + 1) binomial trees.
     - it can be observed by a binary representation of n as (lg n + 1) bits.
     - A bit i = 1 only if a tree B_i is in the heap.

16

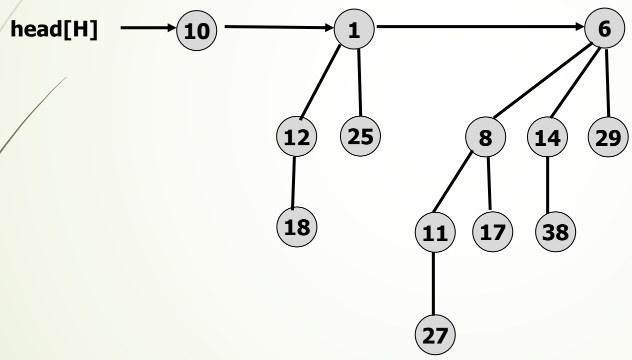
Example

The following heap has 13 keys, binary representation of 13 is 1101

so the heap has

1	1	0	1
B_3	B_2	0	B_0

trees.



17

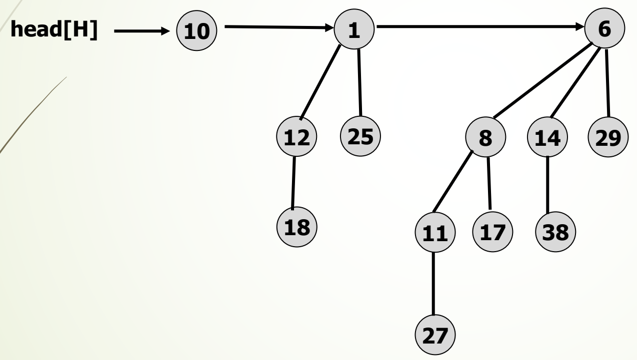
Binomial Heaps: Representation

- Each binomial tree within a binomial heap is stored in the left-child, right-sibling representation.
- Each node has the following fields:
  - key[x]
  - p[x] pointer to the parent.
  - child[x] pointer to the leftmost child. (left child)
  - sibling[x] pointer to the sibling immediately to the right. (right sibling)
  - degree[x] – the number of children of x.
- The roots of the binomial trees in the heap are organized in a linked list, -- root list.
  - The degrees of roots increase when traversing to the right.
- A heap H is accessed by the field head[H], which is the pointer to the first root in the root list of H.

18

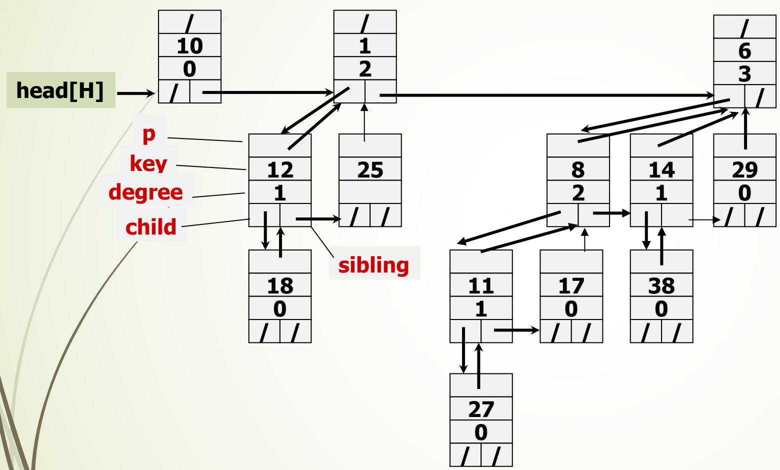
Binomial Heaps: Representation

- If node x is a root, then p[x] = nil
- If node x has no children, then child[x] = nil
- If node x is the right most child of its parent, then sibling[x] = nil



19

Binomial Heaps: Representation



20

Binomial Heaps: Representation

- **Suppose that x is a node in a binomial tree within a binomial heap and assume that sibling[x] != NIL.**
- If x is not a root, how does degree[sibling[x]] compare to degree[x]?

  degree[sibling[x]] degree[x]
- How about if x is a root?

  degree[sibling[x]] degree[x]
- **If x is a nonroot node in a binomial tree within a binomial heap, how does degree[x] compare to degree[p[x]]?**

  degree[x] degree[p[x]]

21

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.
- Running time = Θ(1).

22

Finding the Minimum

- This procedure returns a node with the minimum key in an *n*-node heap *H*. Note that in the min-heap-ordered heap, the minimum key must reside in a root node.

Binomial-Heap-Minimum(H)

```
y = NIL
x = head[H]
min = ∞
while x ≠ NIL
    if key[x] < min
        min = key[x]
        y = x
    x = sibling[x] // next root
return y
```

- Because there are at most $\lfloor \log_2 N \rfloor + 1$ roots to check, the running time of this algorithm is $O(\lg n)$.

Uniting Two Binomial Heap

- The operation of uniting two heaps is used as a subroutine by most of other operations.
- The procedure is implemented by repeatedly linking binomial trees whose roots have the same degree.
  - It uses an auxiliary Binomial-Link procedure to link two trees.
  - It also uses an auxiliary Binomial-Heap-Merge procedure to merge two root lists into a single linked list sorted by degree.
- Uniting two heaps *H1* and *H2* returns a new heap and destroys *H1* and *H2* in the process.

Binomial Link

- This procedure links the B_{k-1} tree rooted at node *y* to the B_{k-1} tree rooted at node *z*. Node *z* becomes the root of a B_k tree. The procedure is straightforward because of the left-child, right-sibling representation of trees.

Binomial-Link(y,z)

1 p[y] = z

2 sibling[y] = child[z]

3 child[z] = y

4 degree[z] = degree[z]+1

- The procedure simply updates pointers in constant time $\Theta(1)$.

Binomial Heap Union Algorithm

```
Binomial – Heap – Union(H1, H2)
{
    H ← Make - Binomial - Heap( )
    head[H] ← Binomial - Heap - Merge(H1, H2)
    free H1 and H2 but not the lists they point to
    if head[H] = NIL then return H
    prev-x ← NIL
    x ← head[H]
    next-x ← sibling[H]

    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            Case1 and 2
  .......
```

Binomial Heap Union Algorithm

```
……
      } 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
}
```

Case 3

Case 4

4 cases of BINOMIAL-HEAP-UNION

1. if (degree[x] ≠ degree[next-x])

   x = next-x;

2. If (degree[x] = degree[next-x]] = degree[sibling[next-x]) then

   x = next-x;

if degree[x] = degree[next-x] != degree[sibling[next-x]] then case 3 or case 4

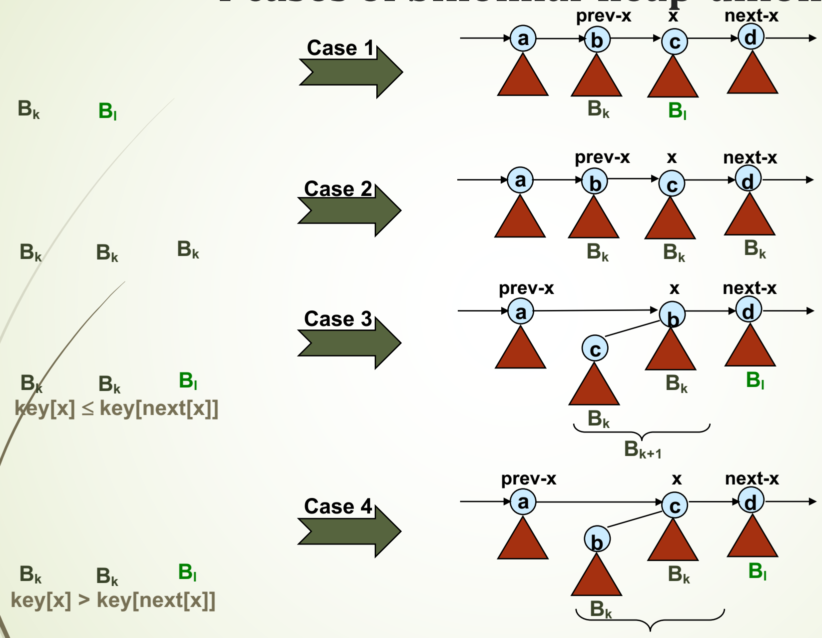
3. if key[x] ≤ key[next-x] then

   Link(next-x, x)

4. if key[x] > key[next-x] then

   Link(x, next-x);

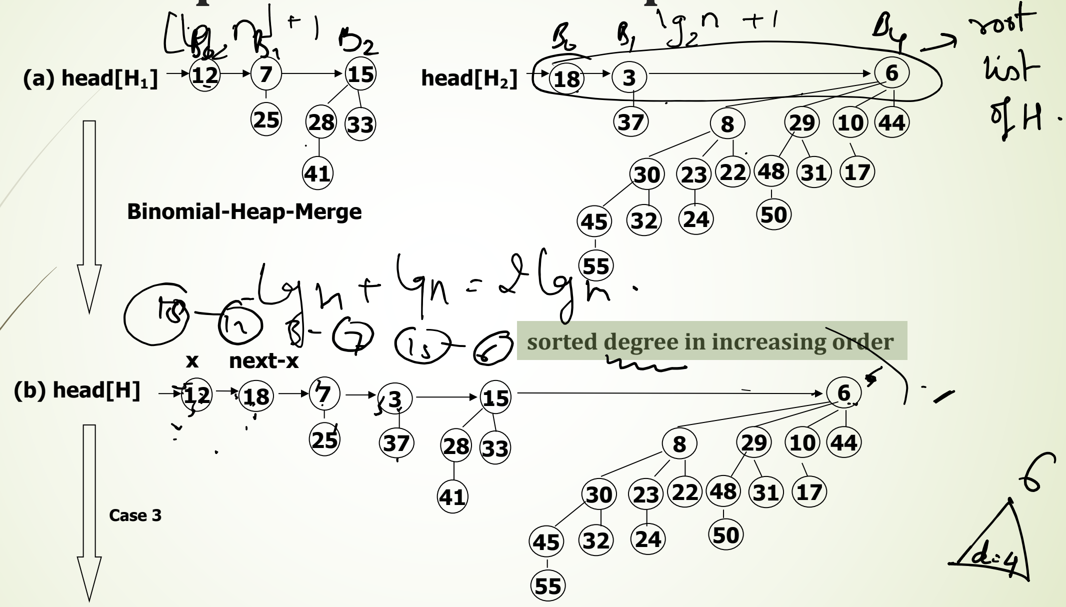
4 cases of binomial-heap-union



Binomial Heap Union Procedure

- The Binomial Heap union procedure has two phases :
- First phase is performed by Binomial-Heap-Merge, that merges the root list of the heaps H1 and H2 into a single linked list H that is sorted by degree into monotonically increasing order. There might be as many as two roots (but no more of each degree).
- The second phase links roots of equal degree until at most one root remains of each degree.
- Throughout the union procedure, we maintain three pointers to root list:
  - x points to the root currently being examined
  - prev_x points to the root preceding x on the root list
  - next_x points to the root following x on the root list

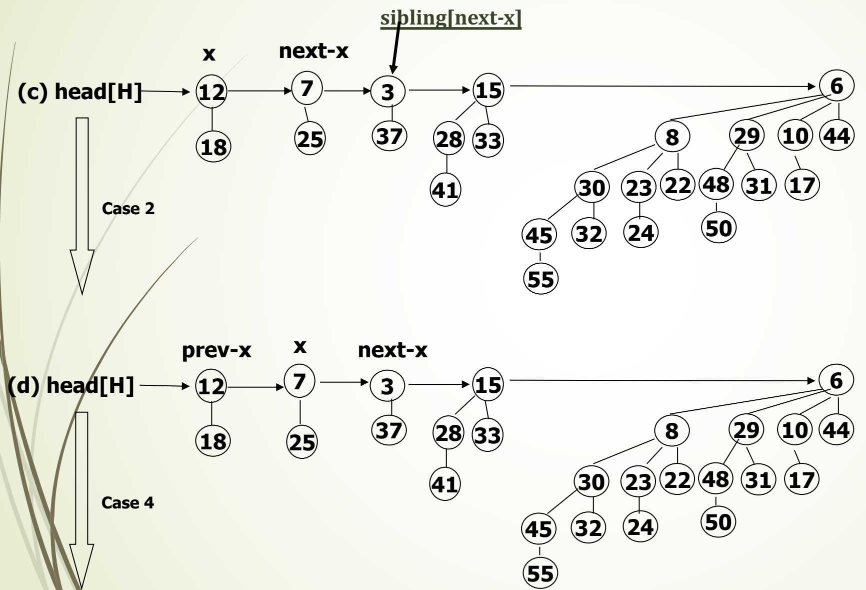
Example - Binomial-heap-union



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

31

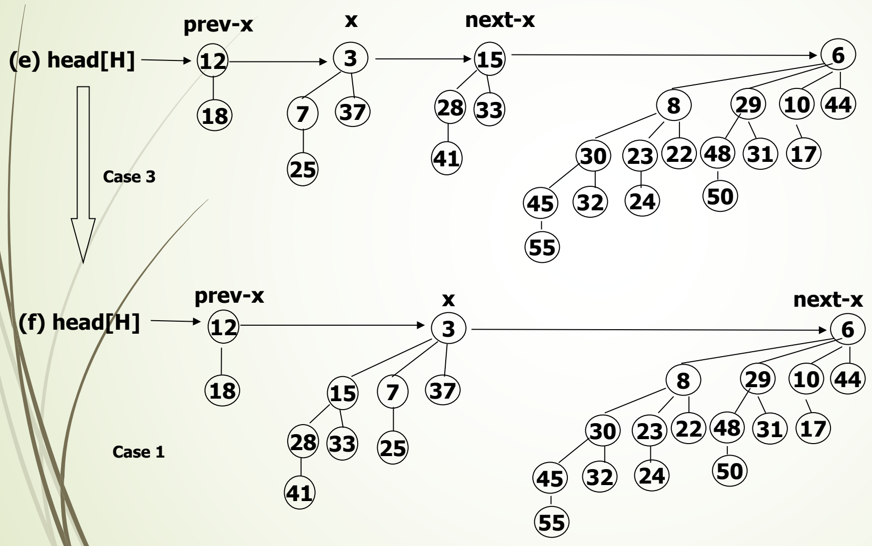
Example - Binomial-heap-union



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

32

Example - Binomial-heap-union



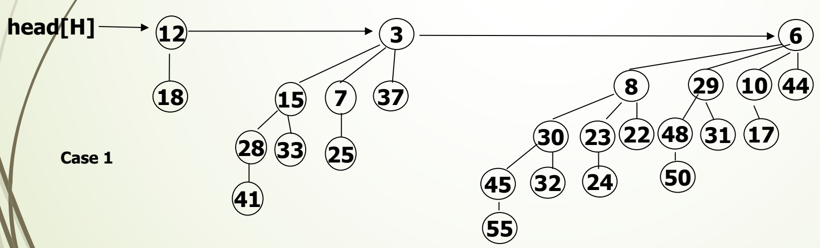
33

Example - Binomial-heap-union

1. Heap H_1 has 7 keys : 111 → $B_2\ B_1\ B_0$
2. Heap H_2 has 19 keys : 10011 → $B_4\ B_1\ B_0$
3. Heap H has 26 keys : 11010 → $B_4\ B_3\ B_1$

Created heap H that is union of heaps H_1 and H_2 contains number of nodes that are equal to the sum of nodes of both heaps.

Binomial Heap Union is analogous to binary addition.



34

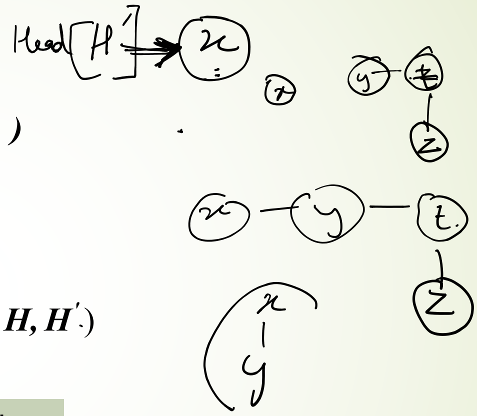
Heap Union: Performance

- 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.
  - Let H_1 contain n_1 nodes and H_2 contain n_2 nodes: $n=n_1+n_2$.
  - H_1 contains at most lg n_1 +1 roots
  - H_2 contains at most lg n_2 +1 roots
  - After merging two heaps, H contains at most lg n_1 + lg n_2 + 2 <= 2 lg n + 2 = O(lg n) roots.
  - Hence the time to perform Binomial-Heap-Merge is O(lg n).
  - Each iteration of the while loop takes Θ(1) time, and there are at most lg n1 + lg n2 + 2 = O(lg n) iterations.
  - Each iteration either advances the pointers one position down the root list, or removes a root from the root list.

35

Insert a Node

Binomial – Heap – Insert(H, x)
{
H′ ← Make - Binomial - Heap( )
p[x] ← NIL
child[x] ← NIL
sibling[x] ← NIL
head[H′] ← x
H ← Binomial - Heap - Union(H, H′)
}

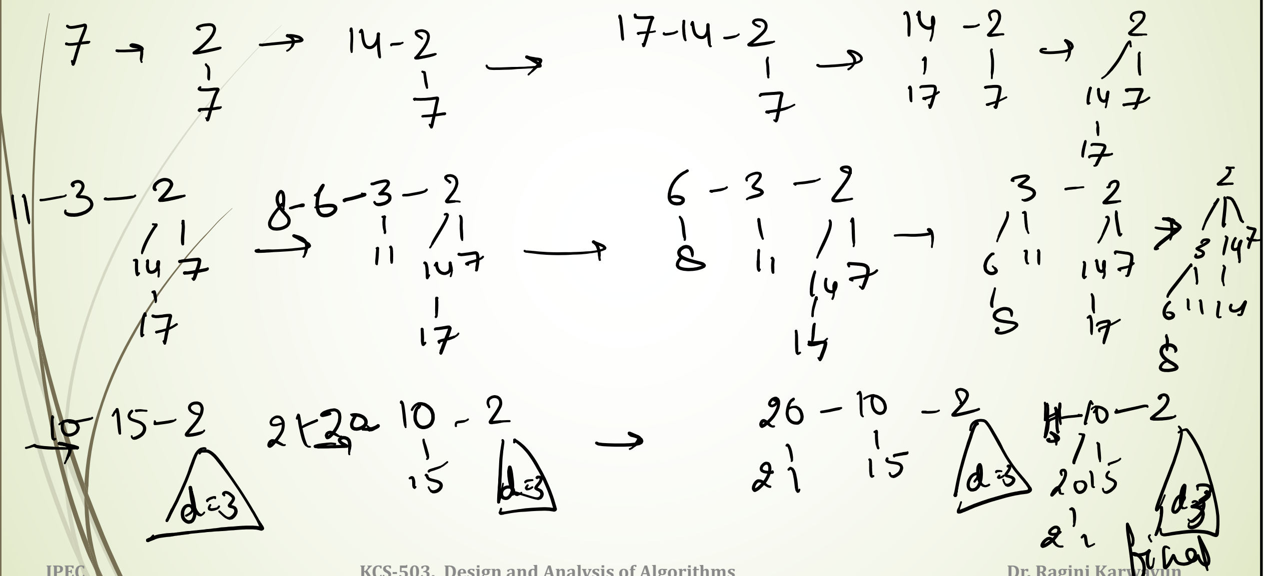


Inserting node x into H, takes O(lg n) time

36

Example

- Insert the following keys in an empty Binomial Heap [7,2,14,17,3,11,6,8,15,10,20,21]



37

38

Extracting the node with minimum key

Binomial – Heap – Extract - Min(H)

{

1. **find and remove the root with the min key in the root list of H**
2. **$H' \leftarrow$ 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. **$H \leftarrow$ Binomial - Heap - Union(H, H')**
5. **return x**

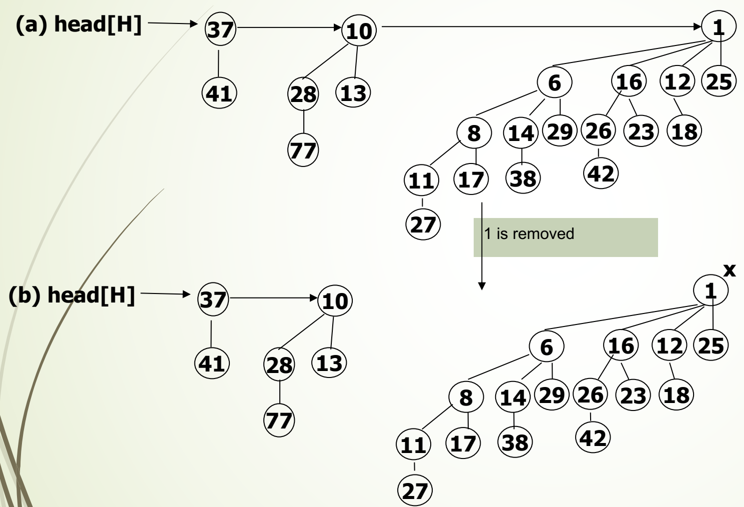
}

Runs in O(lg n) time

39

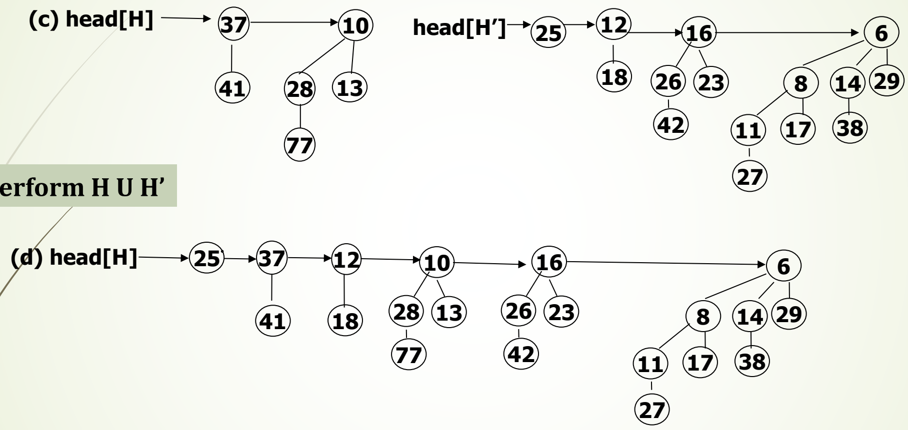
BINOMIAL-HEAP-EXTRACT-MIN

1 is minimum key



40

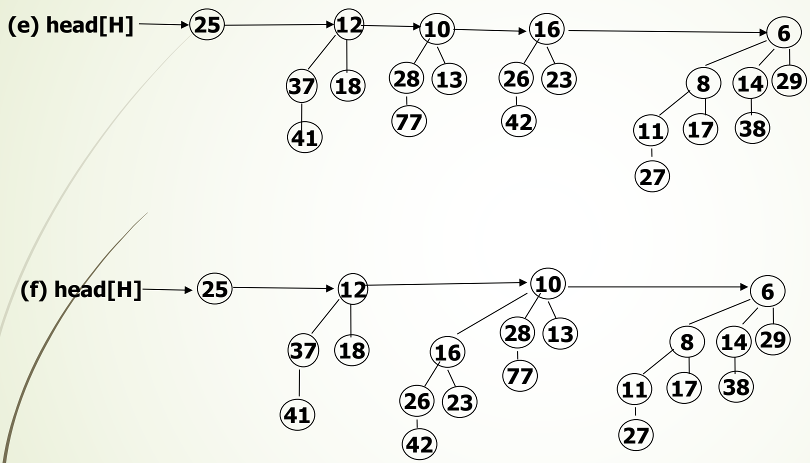
BINOMIAL-HEAP-EXTRACT-MIN



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

41

BINOMIAL-HEAP-EXTRACT-MIN

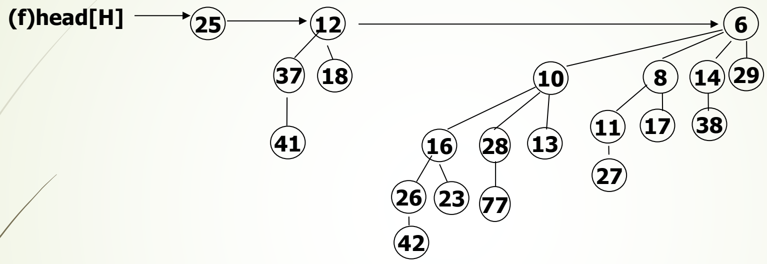


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

42

BINOMIAL-HEAP-EXTRACT-MIN

- Final Heap after extracting the minimum



43

Decreasing a key

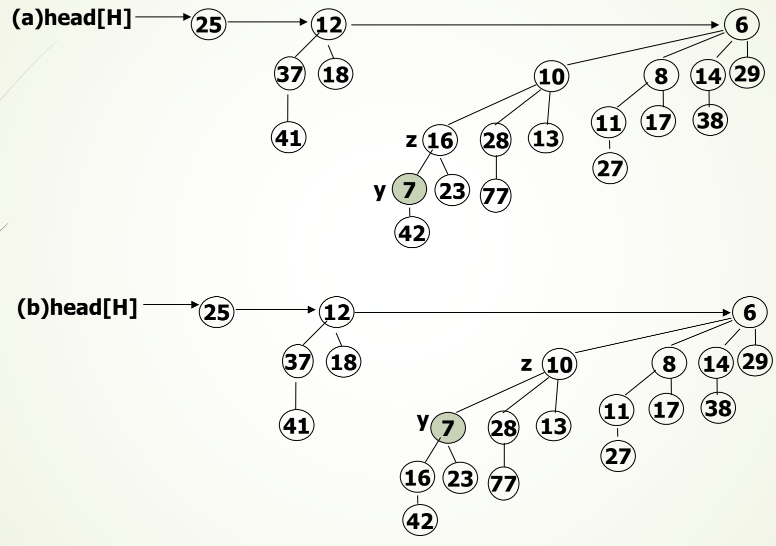
```
Binomial – Heap – Decrease - Key(H, x, k)
{
   if  k > key[x] then   error "k  >  key[x]"
   key[x]  ←  k
   y  ←  x
   z  ←  p[y]
   while ( z ≠ NIL and key[y] < key[z]  ) do
   {
        exchange key[y]  ↔ key[z] and other  fields
        y  ←  z
        z  ← p[y]
   }
}
```

Running time = O(lg n)

44

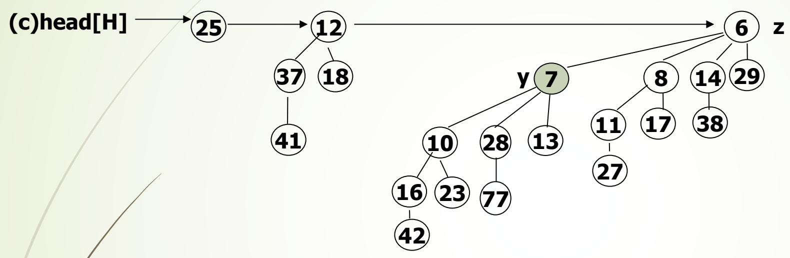
Decreasing a key

- K = 7, x = 26 ie. 26 is decreased to 7



Decreasing a key

- K = 7, x = 26 ie. 26 is decreased to 7



Deleting a key

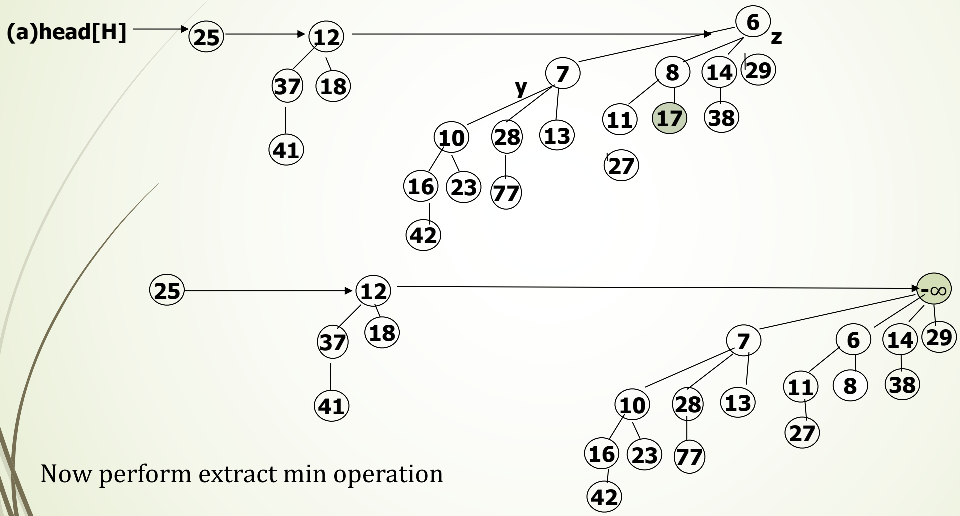
Type equation here.

$$\text{Binomial} - \text{Heap} - \text{Delete}(H, x)$$
$$\{$$
$$\text{Binomial-Heap-Decrease-Key}(H,\ x,\ -\infty)$$
$$\text{Binomial-Heap-Extract-Min}(H)$$
$$\}$$

Running time = O(lg n)

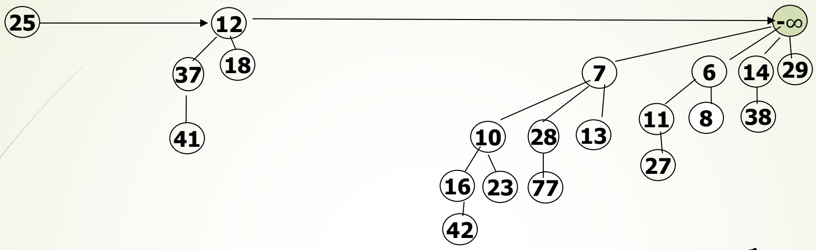
Deleting a key- Example

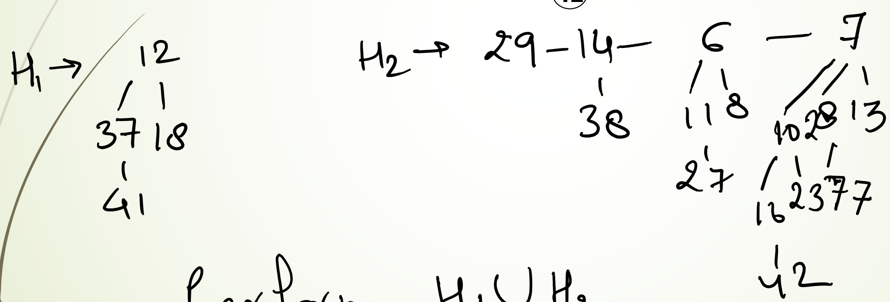
Delete key 17



Now perform extract min operation

Deleting a key- Example

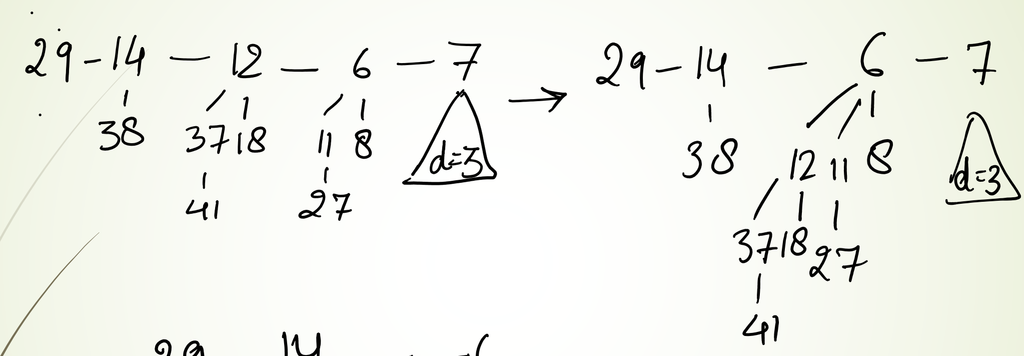


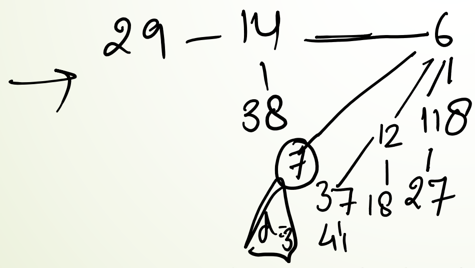


Perform $H_1 \cup H_2$.

49

Questions





final Heap after deleting 17

50

Questions

- Discuss the relationship between inserting into a binomial heap and incrementing a binary number and the relationship between uniting two binomial heaps & adding two binary numbers.