COL106
Data Structures and Algorithms

Subodh Sharma and Rahul Garg

Binomial Heaps

Based on slides by: Kevin Wayne, Princeton University, Ananda Guna, CMU, Guy Kortsarz, Rutgers University

Prim's MST Improved:Runtime

```
Initialize: T = Ø; Pick an arbitrary u ∈ V; U = {u}
for all v: (u, v) in E do
    e = (u, v); D(v) = w(u,v)
    InsertHeap(w(u, v), e)          <- O(|V| log(|V|) total
while (U ≠ V) do
   (x, y) = DeleteMin()             <- O(|V| log(|V|) total
    U = U ∪ { y }
    T = T U (x, y)
    forall (y, w) in E st w in V-U do      <- O(|E|) iterations total
        if w(y, w) < D(w)
            D(w) = w(y, w)
            DecreaseWeight(w)       <- O(|E|) iterations total
                                       O(log(V)) per iteration
        endif
    endfor
end while
```

Total runtime: $O(|V| \log(|V|) + |E| \log(|V|))$

Using Fibonacci Heaps

Depending on the heap implementation, running time could be improved!

	EXTRACT-MIN	DECREASE-KEY	Total
binary heap	O(lgV)	O(lgV)	O(ElgV)
Fibonacci heap	O(lgV)	O(1)	O(VlgV + E)

From $O(|E|\log(|V|)$ to $O(|V|\log|V| + |E|)$

Why Binomial/Fibonacci Heaps?

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log *n*	log *n*	1	1
delete-min	*n*	log *n*	log *n*	log *n*	log *n*
decrease-key	*n*	log *n*	log *n*	1	1
delete	*n*	log *n*	log *n*	log *n*	log *n*
union	1	*n*	log *n*	1	1
find-min	*n*	1	log *n*	1	1

n = number of elements in priority queue

Runtime of Dijkstra's/Prim's Algorithm

O(|V|²)

O(|E|log(|V|)

O(|E| + |V|log(|V|)

Dijkstra's/Prim's

1 make-heap

|V| insert

|V| delete-min

|E| decrease

Why Binomial Heaps

- Will lead to Fibonacci heaps
- Very interesting data structure
- Allows for efficient merging of priority queues
- Very fascinating
- Warning: Be prepared for this exciting journey
- Get all your working memory to work for you, especially for Fibonacci heaps (next class)
- This class will be fun

Binomial Heaps

Programming Techniques — S.L. Graham, R.L. Rivest, Editors

A Data Structure for Manipulating Priority Queues

Jean Vuillemin
Université de Paris-Sud

A data structure is described which can be used for representing a collection of priority queues. The primitive operations are insertion, deletion, union, update, and search for an item of earliest priority.

Key Words and Phrases: data structures, implementation of set operations, priority queues, mergeable heaps, binary trees

CR Categories: 4.34, 5.24, 5.25, 5.32, 8.1

Efficiently Merging two Heaps

- Binary heap is a data structure that allows
  - insert in O(log n)
  - deleteMin in O(log n)
  - findMin in O(1)
- How about merging two heaps
  - complexity is O(n)
- So we discuss a data structure that allows merge in O(log n)

Applications of Heaps

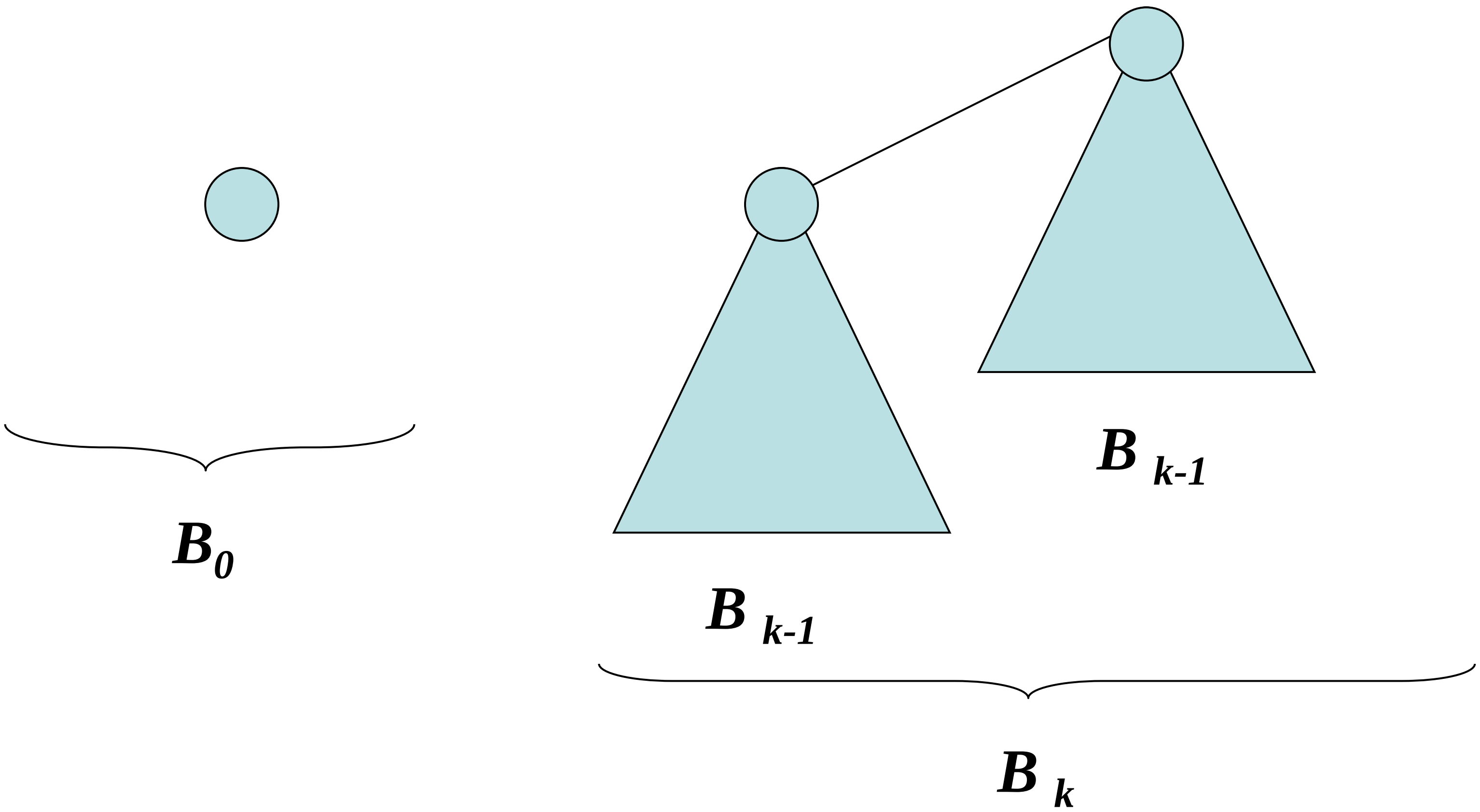
- Binary Heaps
  - efficient findMin, deleteMin
  - many applications
- Binomial Heaps
  - Efficient merge of two heaps
  - Merging two heap-based data structures
- Binomial Heap is build using a structure called Binomial Trees

Binomial Trees

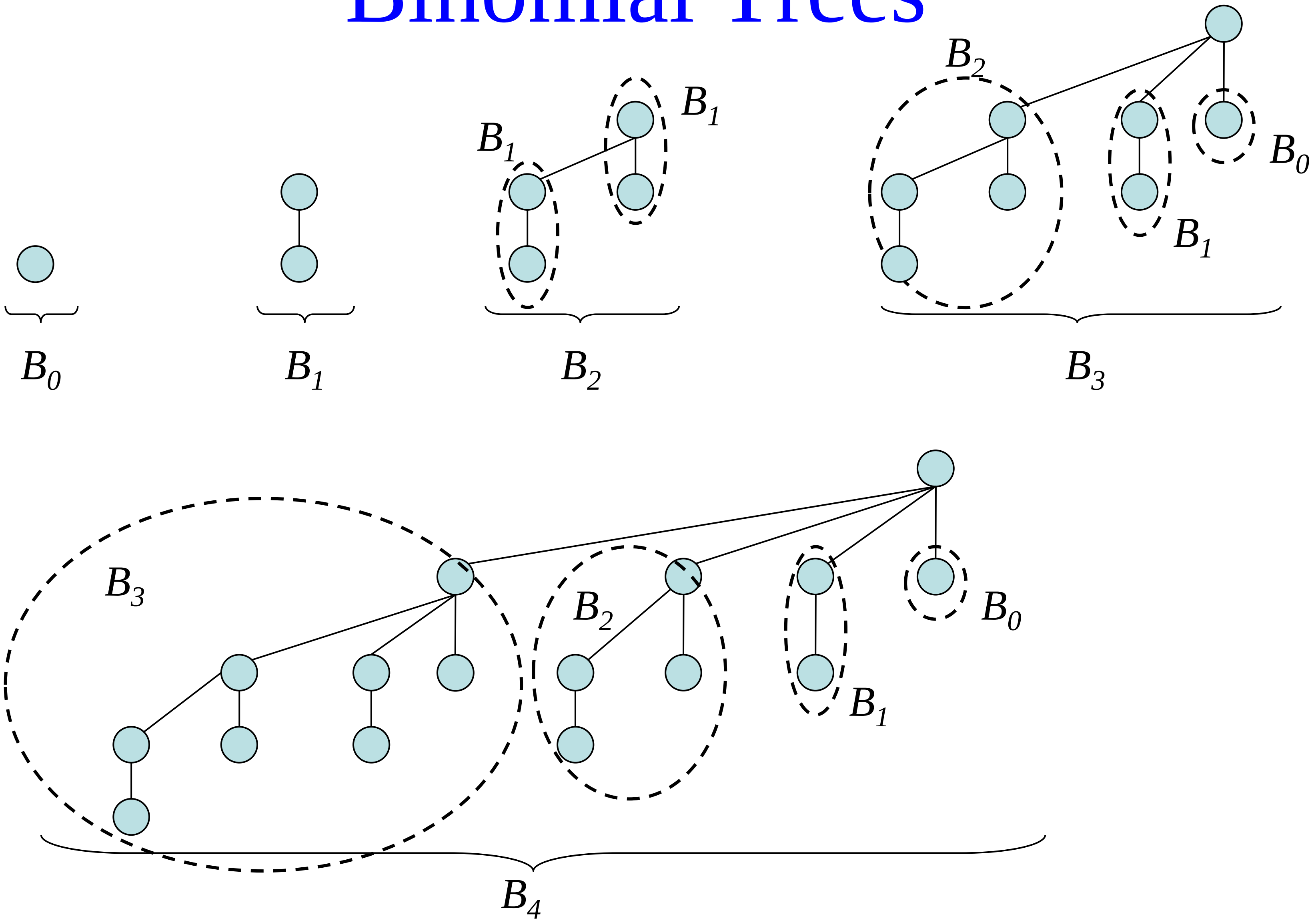
- The binomial tree B_k is an ordered tree defined recursively

    B_o Consists of a single node
    .
    .
    .
    B_k Consists of two binominal trees B_{k-1} linked together. Root of one is the leftmost child of the root of the other.

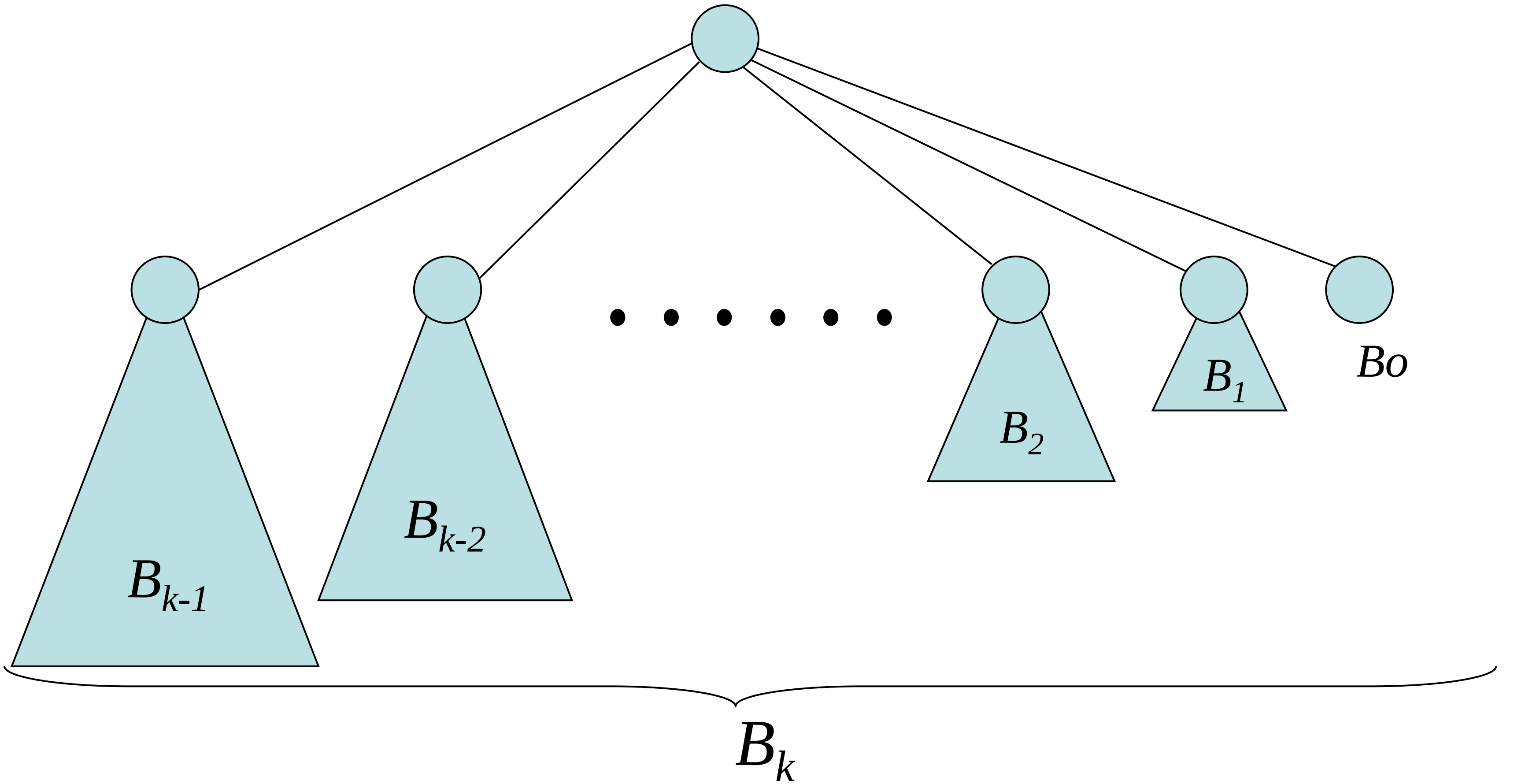
Binomial Trees



Binomial Trees



Binomial Trees



Properties of Binomial Trees

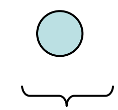
Lemma: For the binomial tree B_k

1. There are 2^k nodes
2. The height of 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 > degree of any other node and
5. If the children of the root are numbered from left to right as k-*1*, k-*2*,...,*0;* child i is the root of a subtree B_i.

Properties of Binomial Trees

Proof: By induction on *k*

Base case: k = 0



B_0

1. There are $2^0 = 1$ nodes
2. The height of tree is 0
3. There are exactly 1 nodes at depth 0
4. The root has degree 0
5. If the children of the root are numbered from left to right as k-1, k-2,…,0; child i is the root of a subtree B_i. Not applicable.

Properties of Binomial Trees

Proof: By induction on *k*

Induction hypothesis: Assume that the lemma holds for B_{k-1}

Induction step: To prove that lemma holds for B_k

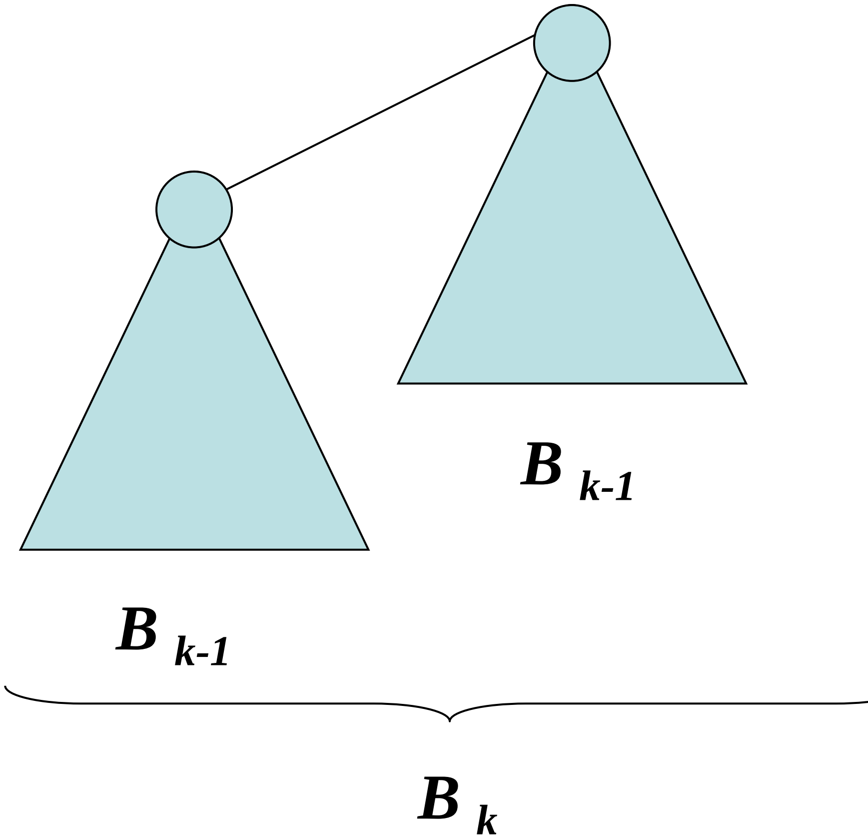
Properties of Binomial Trees

1. B_k consists of two copies of $B_{k\text{-}1}$

∴ $|\ B_k\ | = |\ B_{k\text{-}1}\ | + |\ B_{k\text{-}1}|$
$= 2^{k\text{-}1} + 2^{k\text{-}1} = 2^k$

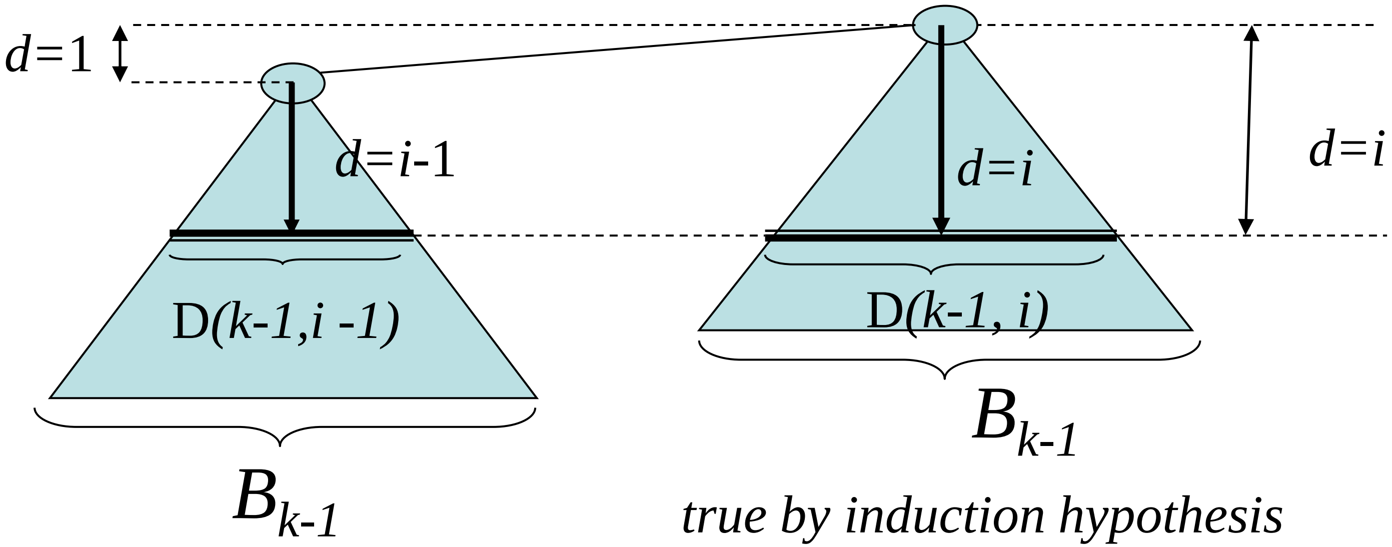
2. $h_{k\text{-}1}$ = Height ($B_{k\text{-}1}$) = k-1 by induction

$h_k = h_{k\text{-}1} + 1 = k\text{-}1 + 1 = k$



Properties of Binomial Trees

3. Let D(k,i) denote the number of nodes at depth i of a B_k ;



true by induction hypothesis

$$D(k,i)=D(k-1,i-1) + D(k-1,i) = \binom{k-1}{i-1} + \binom{k-1}{i} = \binom{k}{i}$$

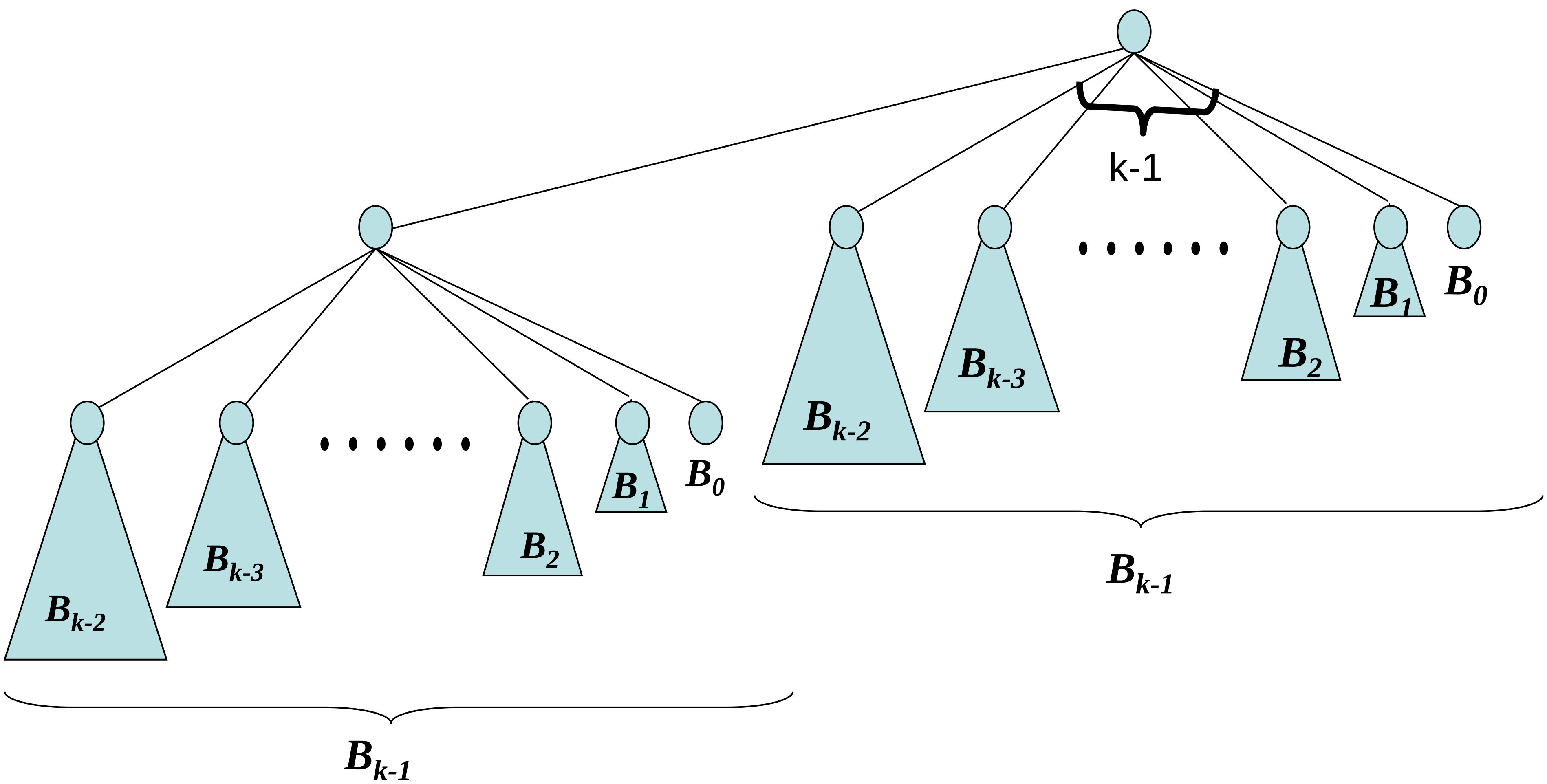
Properties of Binomial Trees

4. Only node with greater degree in B_k than those in B_{k-1} is the root,

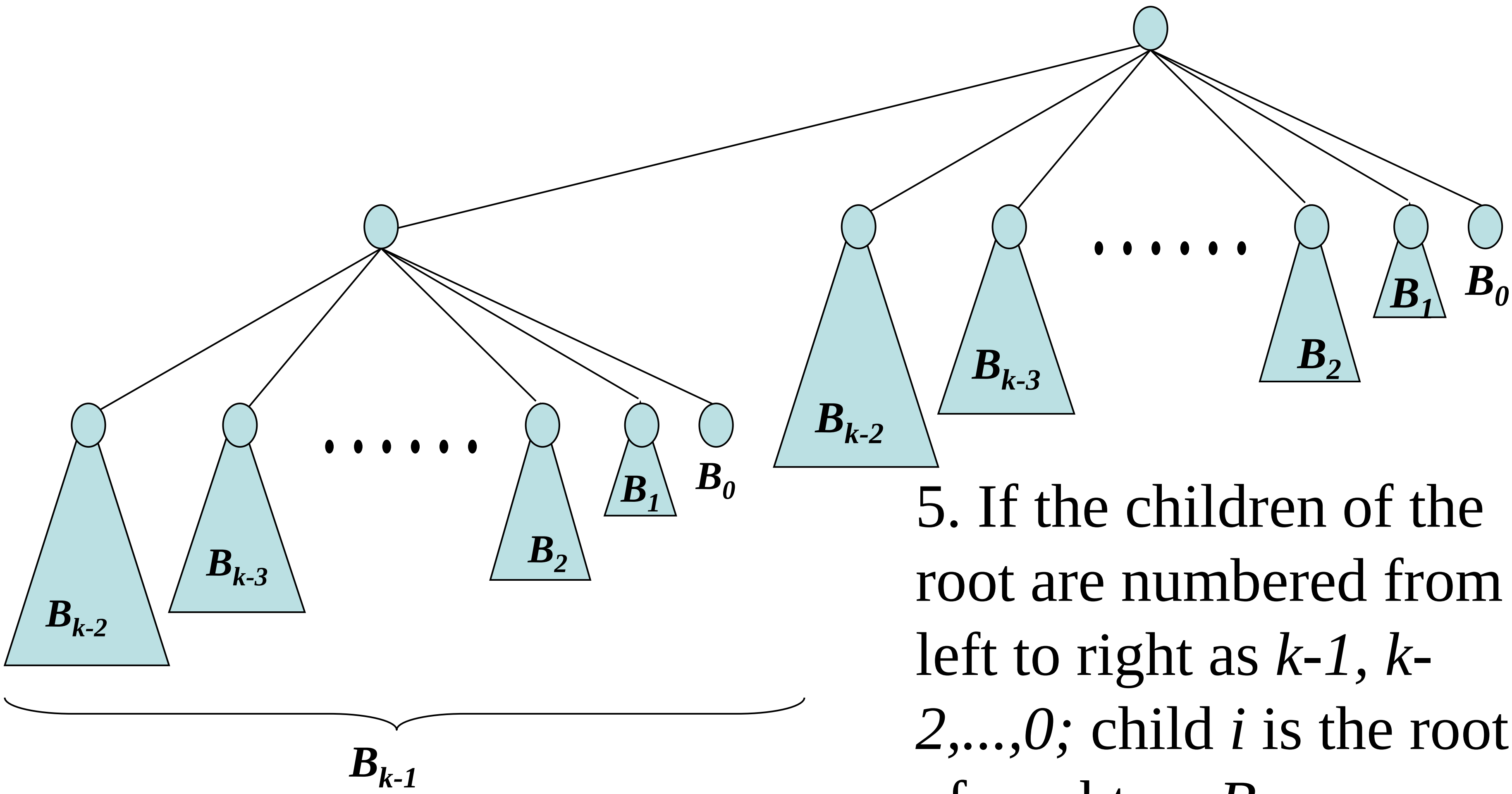
- The root of B_k has one more child than the root of B_{k-1},

∴ Degree of root B_k=Degree of B_{k-1}+1=(k-1)+1=k

Properties of Binomial Trees



Properties of Binomial Trees



5. If the children of the root are numbered from left to right as *k-1*, *k-2*,...,*0;* child *i* is the root of a subtree B_i.

Properties of Binomial Trees

The maximum degree of any node in an n-node binomial tree is log(n)

The term Binomial Tree comes from the 3rd property

i.e. There are $\begin{pmatrix} k \\ i \end{pmatrix}$ nodes at depth i of a B_k

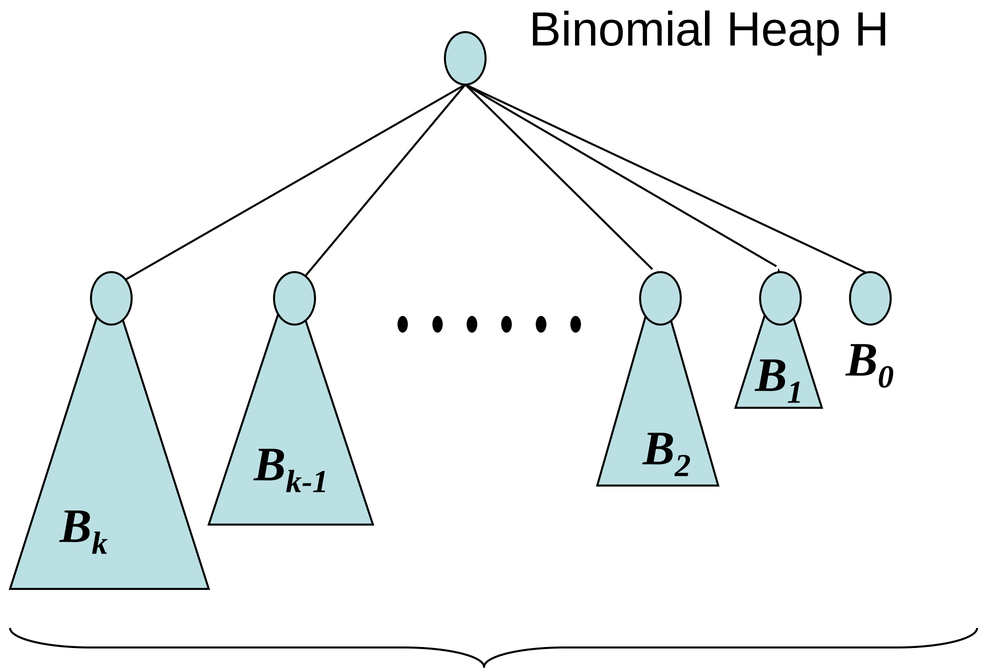
terms $\begin{pmatrix} k \\ i \end{pmatrix}$ are the binomial coefficients.

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 HEAP-ORDERED
   - the key of a node is $\geq$ the key of the parent
   - Root of each binomial tree in *H* contains the smallest key in that tree
2. There is at most one binomial tree B_k for every k

Binomial Heap



Zero or one copies of binomial trees B_k

Let $b_i = 1$ if B_i is present and $b_i = 0$ of B_i is absent

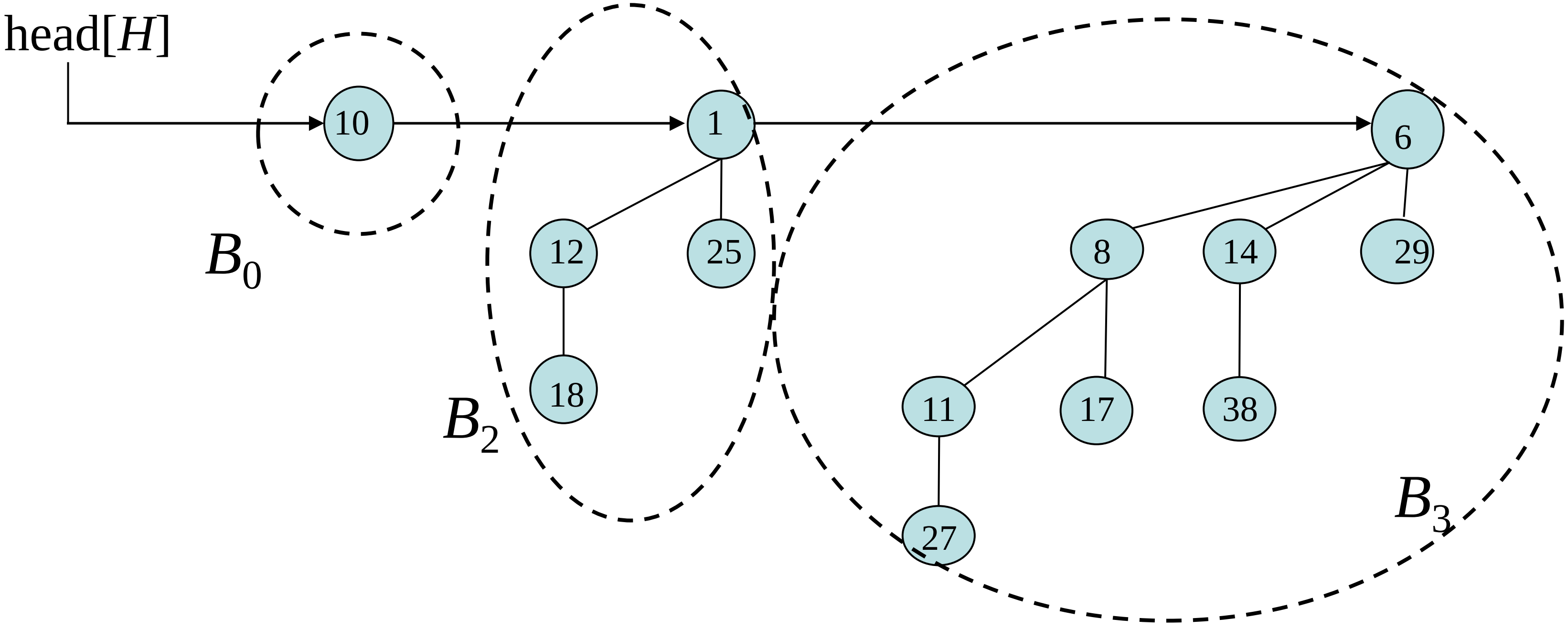
Number of nodes in the heap H = $\sum_{i=0}^{k} b_i 2^i$

Binomial Heaps

Example: A binomial heap with n = 13 nodes

$$13 = [\,\overset{3}{1}\ \overset{2}{1}\ \overset{1}{0}\ \overset{0}{1}\,]_2 = [\,1*2^3 + 1*2^2 + 0*2^1 + 1*2^0] = [\,8 + 4 + 0 + 1]$$

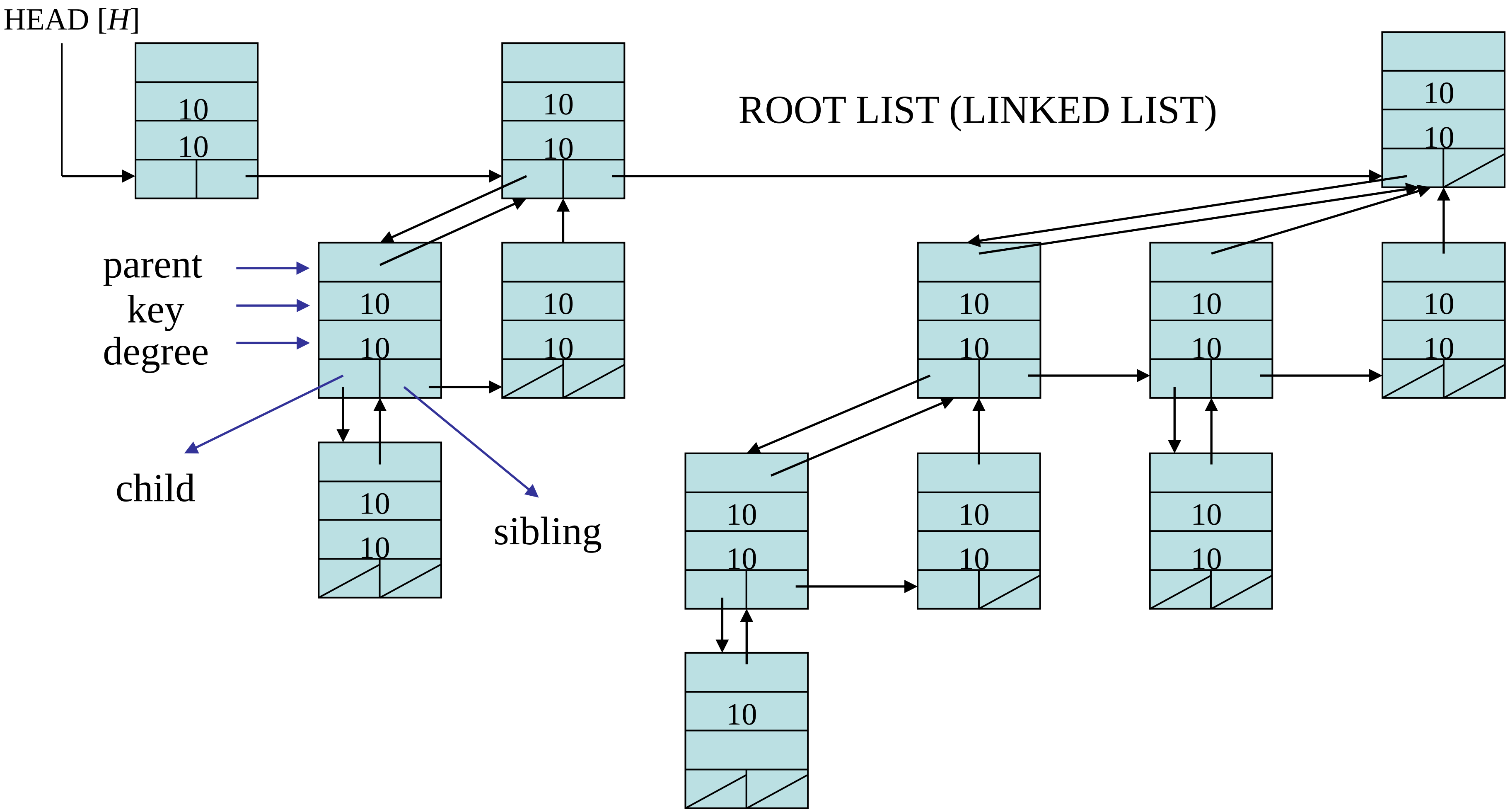
Consists of B_0, B_2, B_3



Representation of Binomial Heaps

- Each binomial tree within a binomial heap is stored in the left-child, right-sibling representation
- Each node *X* contains POINTERS
  - *p*[*x*] to its parent
  - child[*x*] to its leftmost child
  - sibling[*x*] to its immediately right sibling
- Each node *X* also contains the field degree[*x*] which denotes the number of children of *X*.

Representation of Binomial Heaps



Operations on Binomial Heaps

CREATING A NEW BINOMIAL HEAP

MAKE-BINOMIAL-HEAP ( )

allocate H

head [H] $\leftarrow$ NIL

return H

end

RUNNING-TIME= $\Theta(1)$

Operations on Binomial Heaps

BINOMIAL-HEAP-MINIMUM (H)

```
    x ← Head [H]
    min ← key [x]
    x ← sibling [x]
    while   x ≠ NIL do
        if   key [x] < min then
                min ← key [x]
                y ← x
        endif
                x ← sibling [x]
    endwhile
    return y
end
```

Operations on Binomial Heaps

Since binomial heap is HEAP-ORDERED

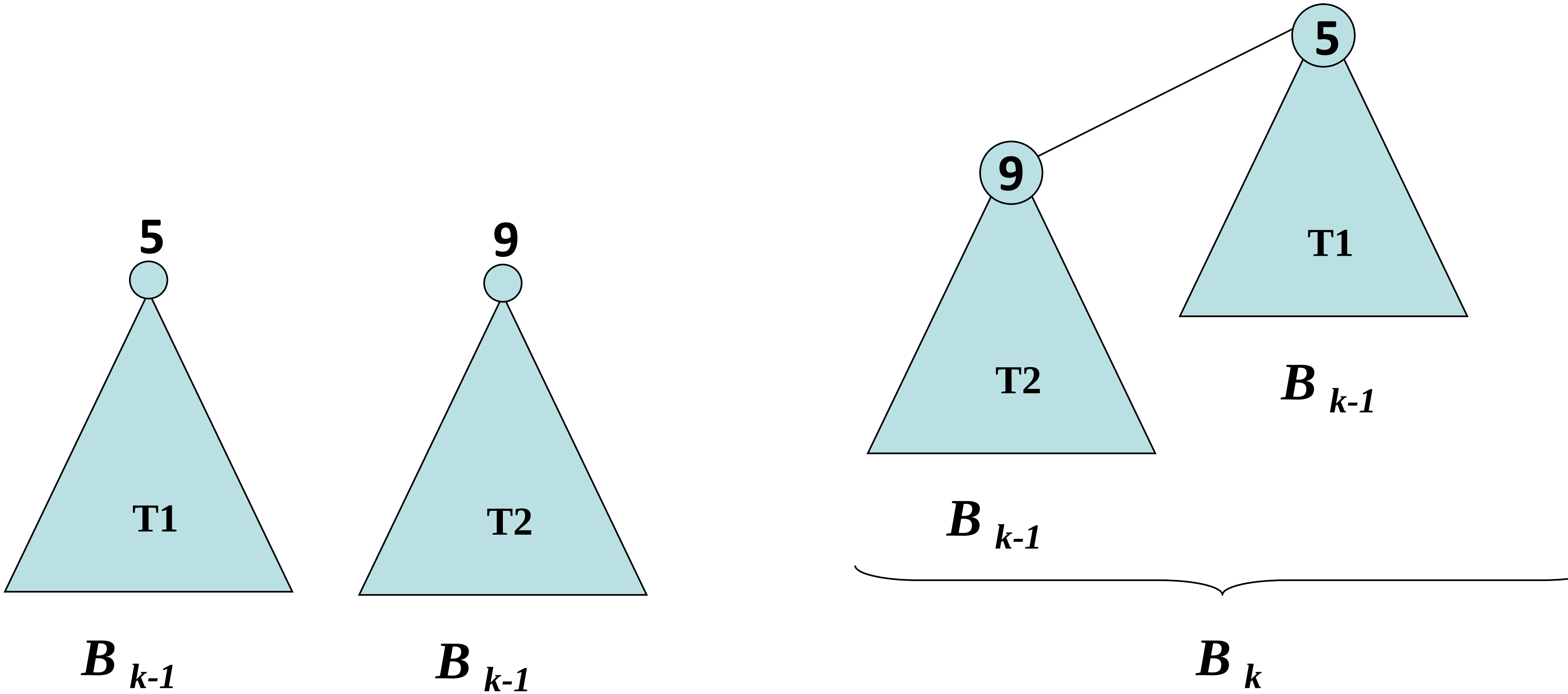
The minimum key must reside in a ROOT NODE

Above procedure checks all roots

NUMBER OF ROOTS $\leq \lfloor \log n \rfloor + 1$

$\therefore$ RUNNING–TIME = O ($\log n$)

Merging Heap Ordered Binomial Trees



Two heap ordered binomial trees B_{k-1} can be merged in O(1) time to give a heap ordered binomial tree B_k

Merging Binomial Heaps

- Just like adding numbers in binary
- Merging binomial heaps H1 with 13 and H2 with 7 nodes

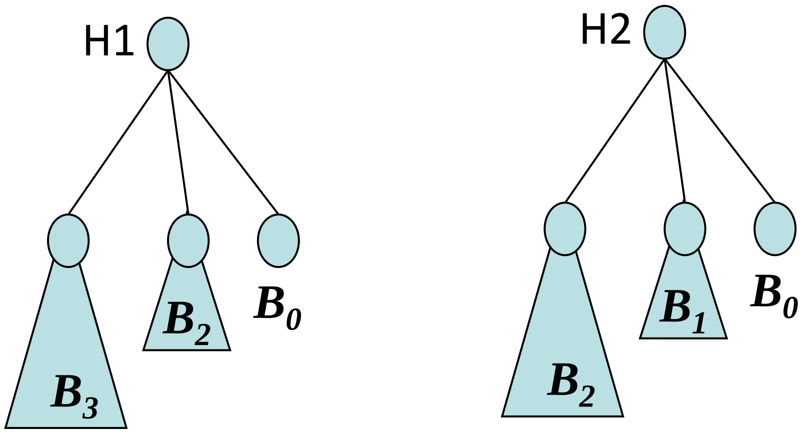
$13 = [\ 1\ 1\ 0\ 1\ ]_2 = [\ 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0] = [\ 8 + 4 + 0 + 1]$

H1 has B_3 (8 nodes) + B_2 (4 nodes) + B_0 (1 node)

$7 = [\ 0\ 1\ 1\ 1\ ]_2 = [\ 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0] = [\ 0 + 4 + 2 + 1]$

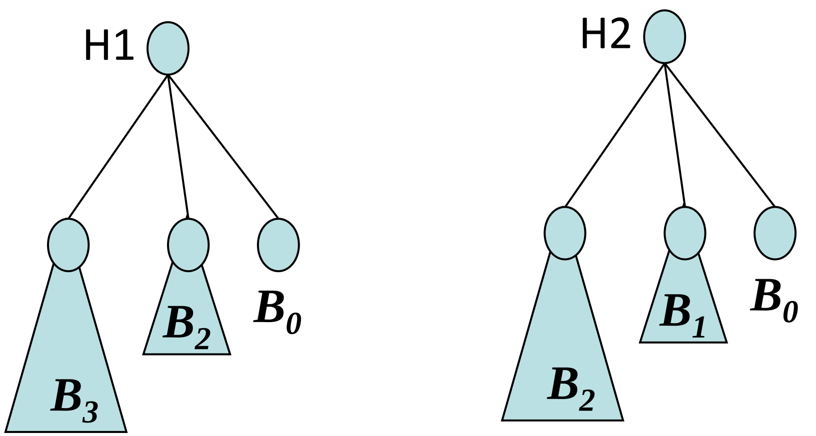
H2 has B_2 (4 nodes) + B_1 (2 nodes) + B_0 (1 node)

Merging Binomial Heaps



H1 $[\,1\ 1\ 0\ 1\,]_2$
H2 $[\,0\ 1\ 1\ 1\,]_2$

Merging Binomial Heaps



? ? 1 _

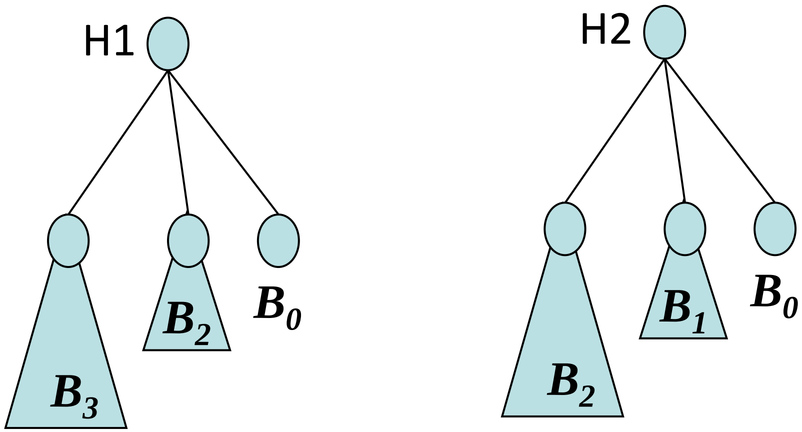
H1 [1 1 0 1]$_2$

H2 [0 1 1 1]$_2$

[? ? ? 0]$_2$

Step 1: Merge B_0 of H1 and H2 to get B_1

Merging Binomial Heaps



? 1 1 _

H1 $[\,1\ 1\ 0\ 1\,]_2$

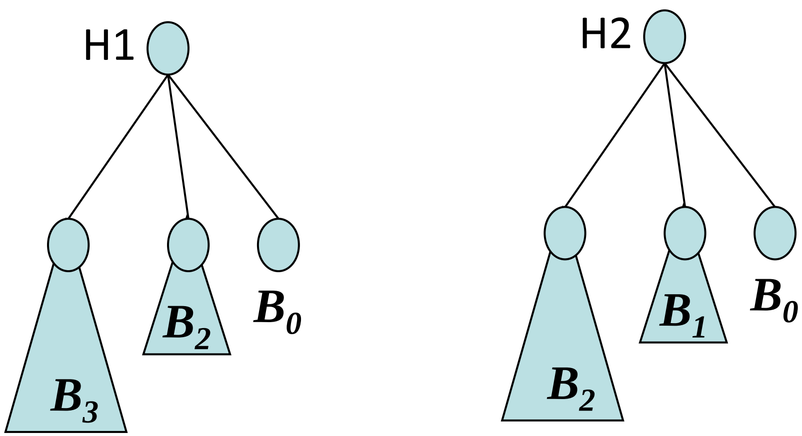
H2 $[\,0\ 1\ 1\ 1\,]_2$

$[\,?\ ?\ 0\ 0\,]_2$

Step 1: Merge B_0 of H1 and H2 to get B_1

Step 2: Merge B_1 of H2 with B_1 of step 1 to get B_2

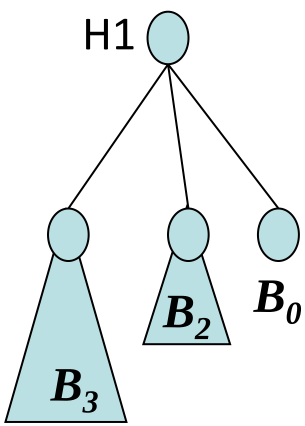
Merging Binomial Heaps

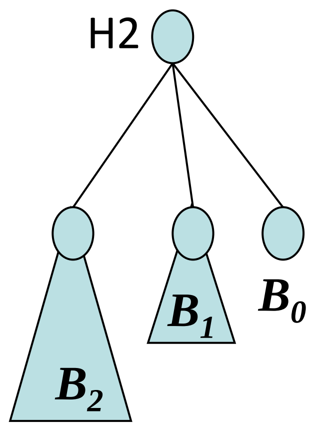


$$
\begin{array}{rl}
 & \;\;1\;1\;1\;\_ \\
\text{H1} & [\;1\;1\;0\;1\;]_2 \\
\text{H2} & [\;0\;1\;1\;1\;]_2 \\
 & \\
 & [\;?\;1\;0\;0\;]_2
\end{array}
$$

Step 1: Merge B_0 of H1 and H2 to get B_1
Step 2: Merge B_1 of H2 with B_1 of step 1 to get B_2
Step 3: Merge B_2 of H1 and B_2 of H2 to get B_3; Keep B_2 of step 2 intact

Merging Binomial Heaps





1 1 1 _

H1 [1 1 0 1]$_2$

H2 [0 1 1 1]$_2$

[1 0 1 0 0]$_2$

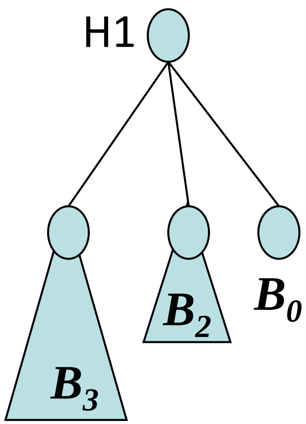
Step 1: Merge B_0 of H1 and H2 to get B_1

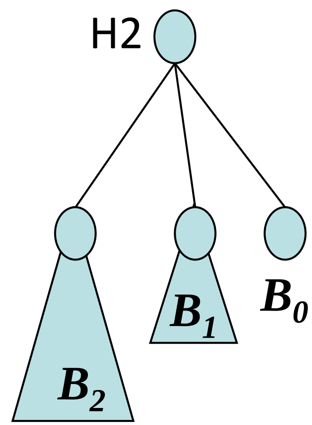
Step 2: Merge B_1 of H2 with B_1 of step 1 to get B_2

Step 3: Merge B_2 of H1 and B_2 of H2 to get B_3; Keep B_2 of step 2 intact

Step 4: Merge B_3 of H1 and B_3 of step 3 to get B_4

Merging Binomial Heaps

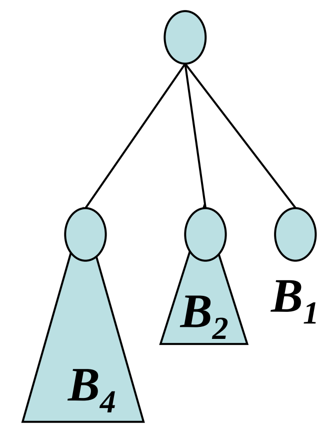




1 1 1 _
H1 [1 1 0 1] $_2$
H2 [0 1 1 1] $_2$

[1 0 1 0 0] $_2$

Merged Tree (H1+H2)



Step 1: Merge B_0 of H1 and H2 to get B_1
Step 2: Merge B_1 of H2 with B_1 of step 1 to get B_2
Step 3: Merge B_2 of H1 and B_2 of H2 to get B_3; Keep B_2 of step 2 intact
Step 4: Merge B_3 of H1 and B_3 of step 3 to get B_4

Time Taken to Merge Two Heaps

Since time taken to merge two binomial trees is O(1), the time taken to merge two binomial heaps is O(log(n)), number of bits needed to represent n

Inserting a Node

Create a single node binomial tree B0

Merge B0 with the current binomial heap

Total time is O(log n) in the worst case

Hint: How many operations may be needed to increment a k-bit binary number?

BINOMIAL-HEAP-INSERT (H,x)

H' ← MAKE-BINOMIAL-HEAP (H, x)

P [x] ← NIL

child [x] ← NIL

sibling [x] ← NIL

degree [x] ← O

head [H'] ← x

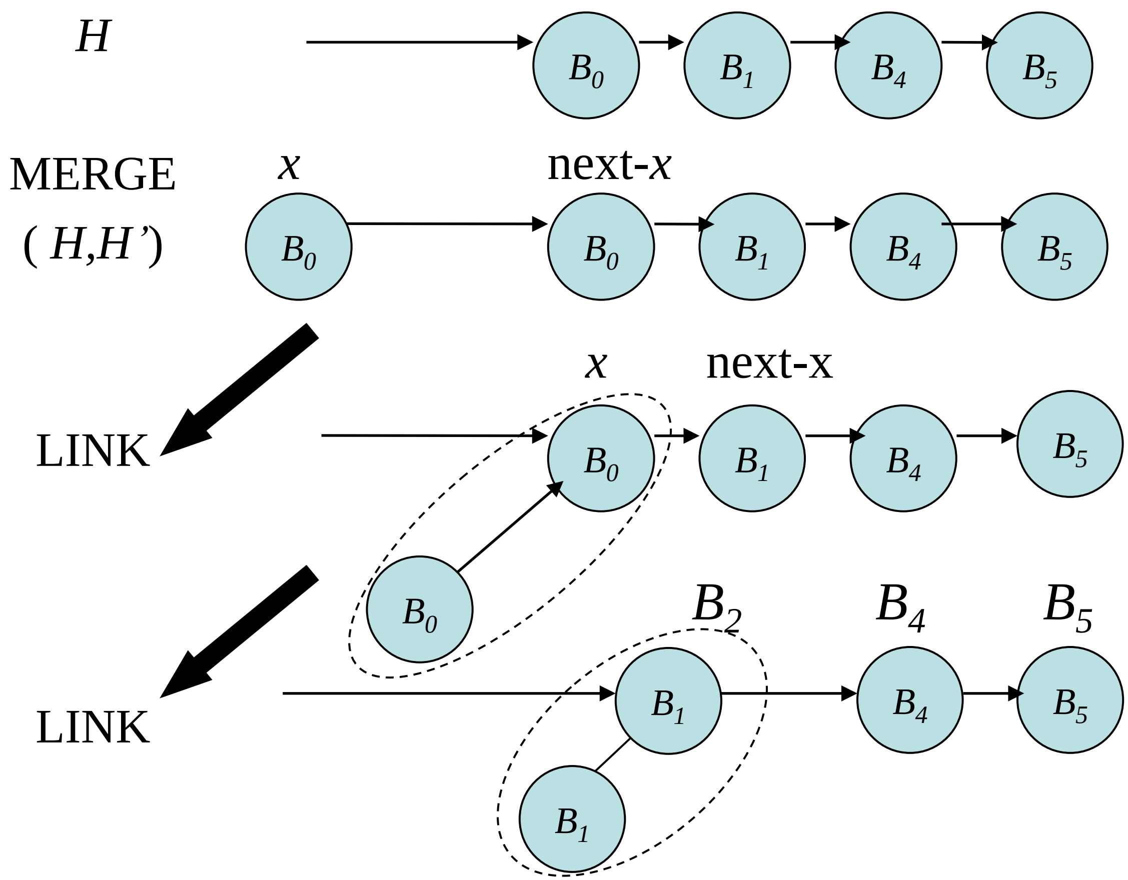
H ← BINOMIAL-HEAP-UNION (H, H')

end

RUNNING-TIME= O(lg n)

Relationship Between Insertion & Incrementing a Binary Number

$H : n_1 = 51 \qquad H = <110011> \quad = \quad \{ B_0, B_1, B_4, B_5 \}$



	5	4	3	2	1	0
					1	
	1	1	0	0	1	1
+						1
	1	1	0	1	0	0

Extracting the Node with the 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) $H' \leftarrow$ MAKE-BINOMIAL-HEAP ( )

(3) reverse the order of the linked list of x' children and set head $[H'] \leftarrow$ head of the resulting list

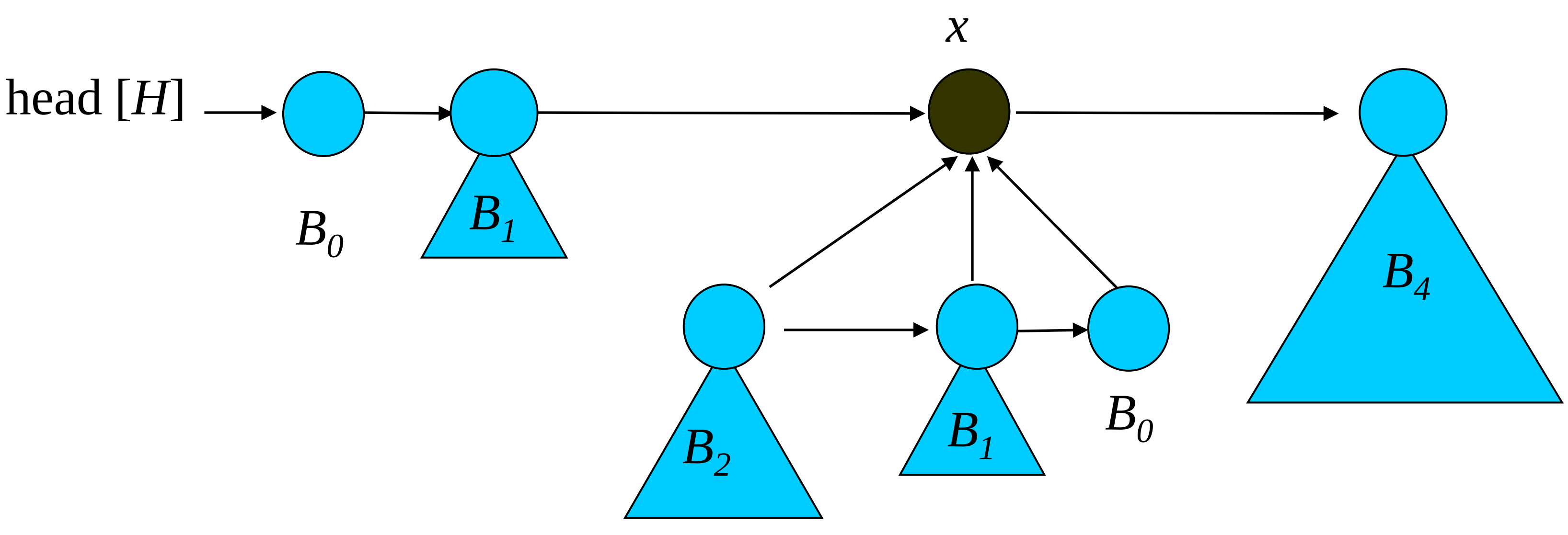
(4) $H \leftarrow$ BINOMIAL-HEAP-UNION (H, H')

return x

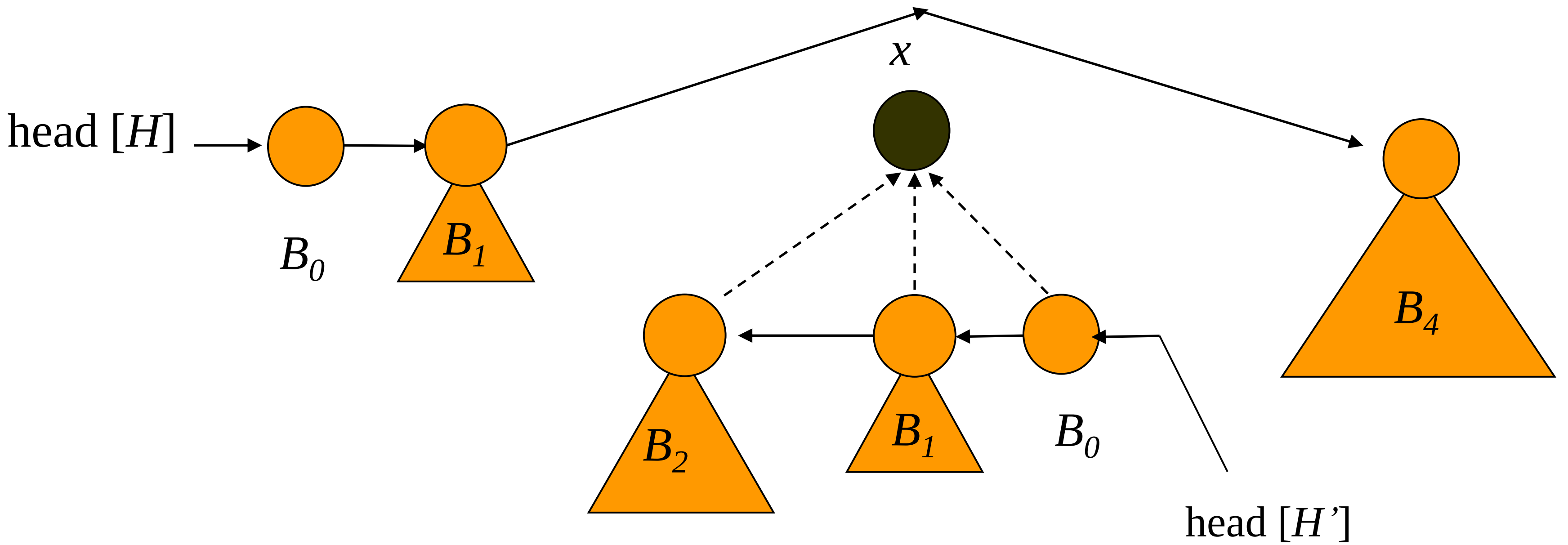
end

Extracting the Node with the Minimum Key

Consider H with n = 27, H = <1 1 0 1 1> = $\{B_0, B_1, B_3, B_4\}$
assume that x = root of B_3 is the root with minimum key



Extracting the Node with the Minimum Key



Extracting the Node with the Minimum Key

- Unite binomial heaps $H = \{B_0, B_1, B_4\}$ and $H' = \{B_0, B_1, B_2\}$
- Running time if H has n nodes
- Each of lines 1-4 takes $O(\lg n)$ time

  it is $O(\lg n)$.

Decreasing a Key

Keep swapping with the parent until parent is larger

BINOMIAL-HEAP-DECREASE-KEY (H, x, k)

key [x] ← k

y ← x

z ← p[y]

while z ≠ NIL and key [y] < key [z] do

exchange key [y] ← key [z]

exchange satellite fields of y and z

y ← z

z ← p [y]

endwhile

end

Decreasing a Key

- Similar to DECREASE-KEY in BINARY HEAP
- BUBBLE-UP the key in the binomial tree it resides in
- RUNNING TIME: O(log n)

Deleting a Key

BINOMIAL- HEAP- DELETE (H,x)

$y \leftarrow x$

$z \leftarrow$ p $[y]$

while $z \neq$ NIL do

key $[y] \leftarrow$ key $[z]$

satellite field of y $\leftarrow$ satellite field of z

$y \leftarrow z$; $z \leftarrow$ p $[y]$

endwhile

$H' \leftarrow$ MAKE-BINOMIAL-HEAP

remove root z from the root list of H

reverse the order of the linked list of z's children

and set head $[H'] \leftarrow$ head of the resulting list

$H \leftarrow$ BINOMIAL-HEAP-UNION (H, H')

RUNNING-TIME= O(lg n)

Deleting a Key (Cont.)

$H' \leftarrow$ MAKE-BINOMIAL-HEAP

remove root z from the root list of H

reverse the order of the linked list of z's children

set head $[H'] \leftarrow$ head of the resulting list

$H \leftarrow$ BINOMIAL-HEAP-UNION (H, H')

end

Binomial Heaps: Summary

Operation	Linked List	Binary Heap	Binomial Heap	Fibonacci Heap †	Relaxed Heap
make-heap	1	1	1	1	1
is-empty	1	1	1	1	1
insert	1	log *n*	log *n*	1	1
delete-min	*n*	log *n*	log *n*	log *n*	log *n*
decrease-key	*n*	log *n*	log *n*	1	1
delete	*n*	log *n*	log *n*	log *n*	log *n*
union	1	*n*	log *n*	1	1
find-min	*n*	1	log *n*	1	1

n = number of elements in priority queue

Runtime of Dijkstra's/Prim's Algorithm

$O(|V|^2)$

$O(|E|\log(|V|)$

$O(|E| + |V|\log(|V|)$

Thank You