

## Syllabus: Design and Analysis of Algorithms (BCSC0012)

<b>Module No.</b>	<b>Content Credits:03 L-T-P-J: 3-0-0-0</b>	<b>Teaching Hours</b>
<b>I</b>	<p><b>Introduction:</b> Algorithms, Analyzing algorithms, Complexity of algorithms, Growth of functions, Performance measurements, Sorting and order Statistics - Shell sort, Quick sort, Merge sort, Heap sort, Comparison of sorting algorithms, Sorting in linear time.</p> <p><b>Advanced Data Structures:</b> Red-Black trees, B – trees, Binomial Heaps, Fibonacci Heaps.</p> <p>Divide and Conquer with examples such as Sorting, Matrix Multiplication, Convex hull and Searching.</p>	20
<b>II</b>	<p>Greedy methods with examples such as Optimal Reliability Allocation, Knapsack, Minimum Spanning trees – Prim's and Kruskal's algorithms, Single source shortest paths - Dijkstra's and Bellman Ford algorithms.</p> <p>Backtracking, Branch and Bound with examples such as Travelling Salesman Problem, Graph Coloring, n-Queen Problem, Hamiltonian Cycles and Sum of subsets</p> <p>Dynamic programming with examples such as Knapsack. All pair shortest paths – Warshall's and Floyd's algorithms, Resource allocation problem</p>	20

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 1: Introduction



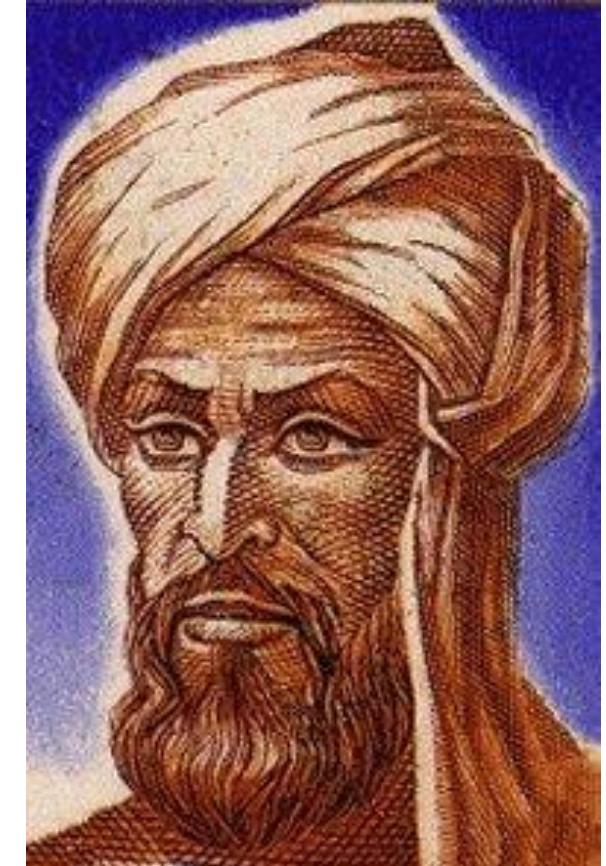
Prof. Anand Singh Jalal

**Department of Computer Engineering & Applications**

# The Men Behind the Word Algorithm

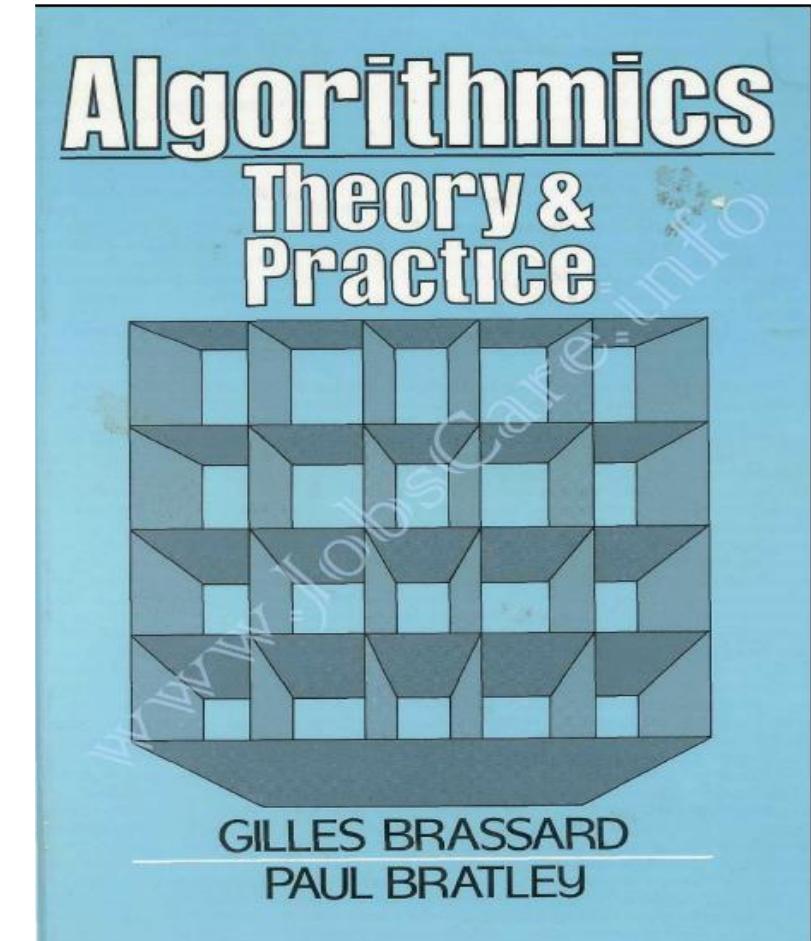
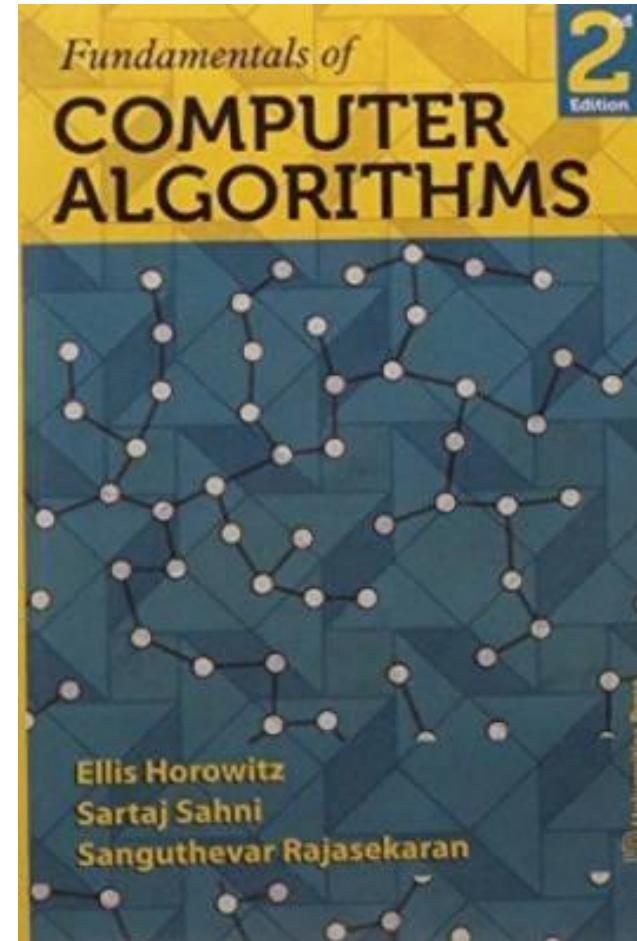
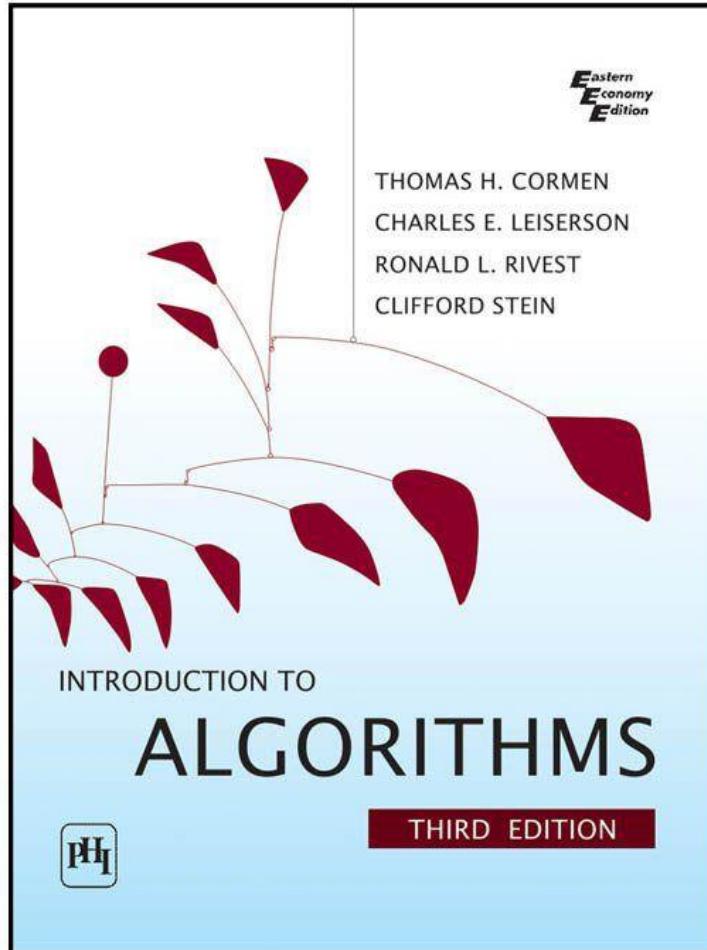
The word algorithm comes from the name of the 9th century Persian mathematician **Muhammad ibn Mūsā al-Khwārizmī**, Who was Mathematician, astronomer and geographer.

He is often considered one of the **fathers of Algebra**. Also he is the creator of the numbers which we use today.



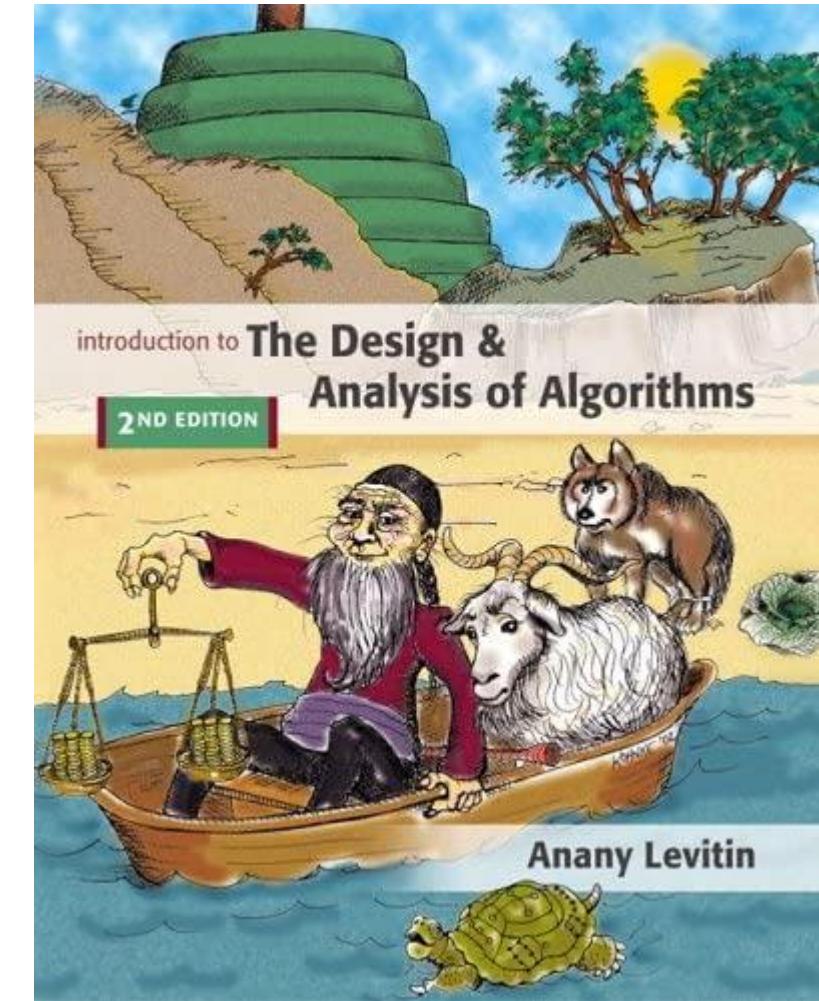
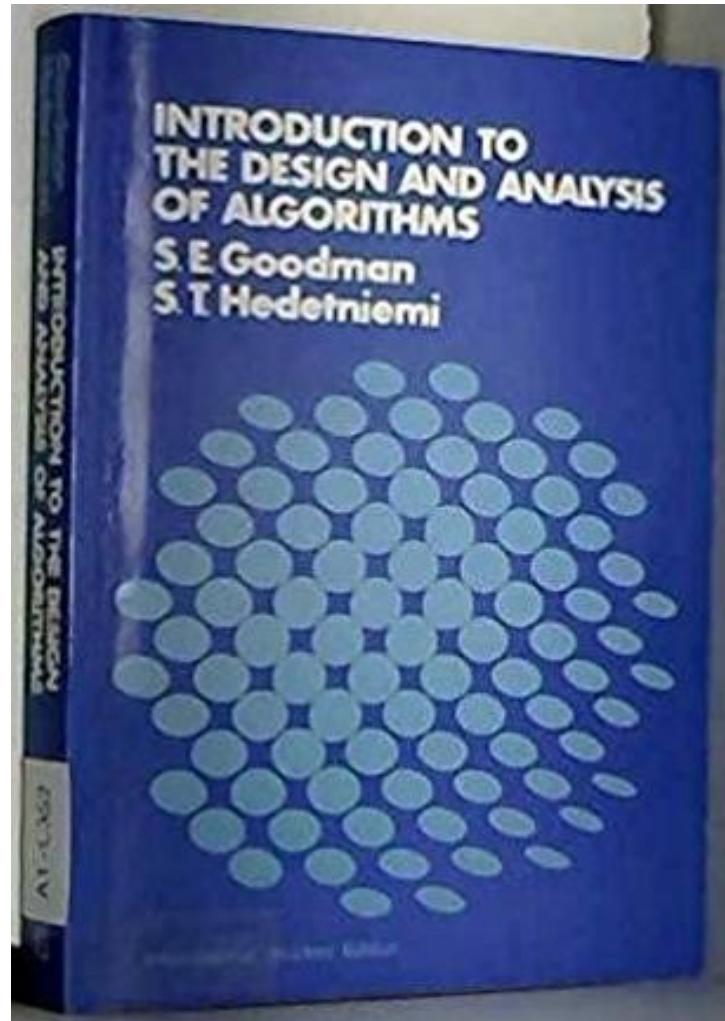
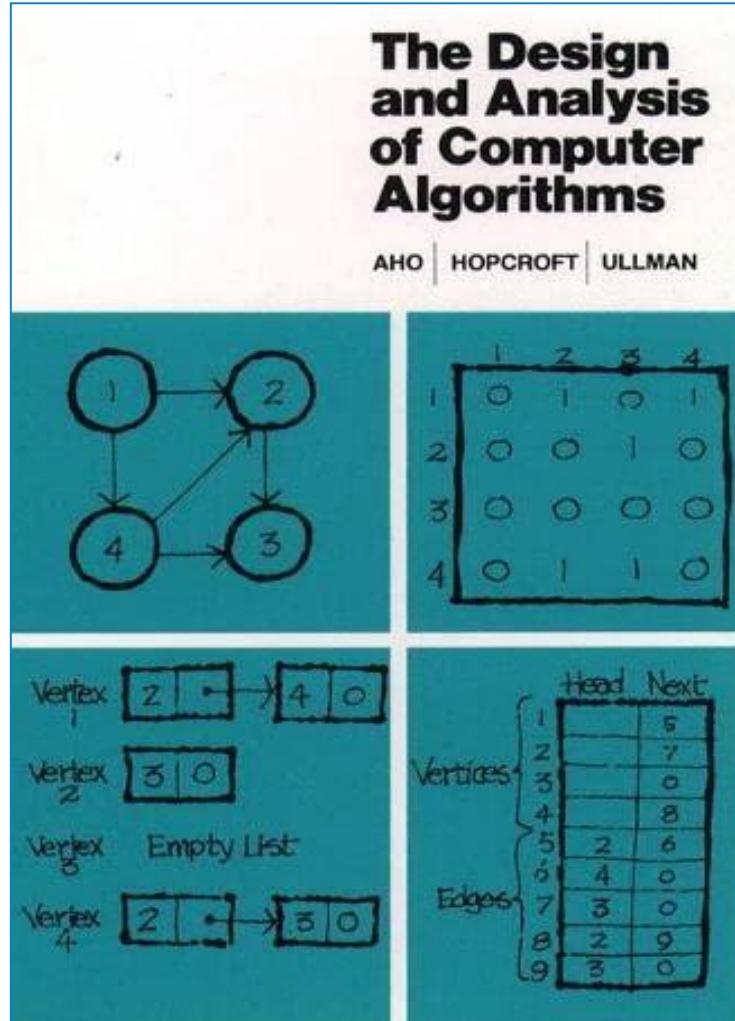
# Books

Accredited with **A** Grade by **NAAC**

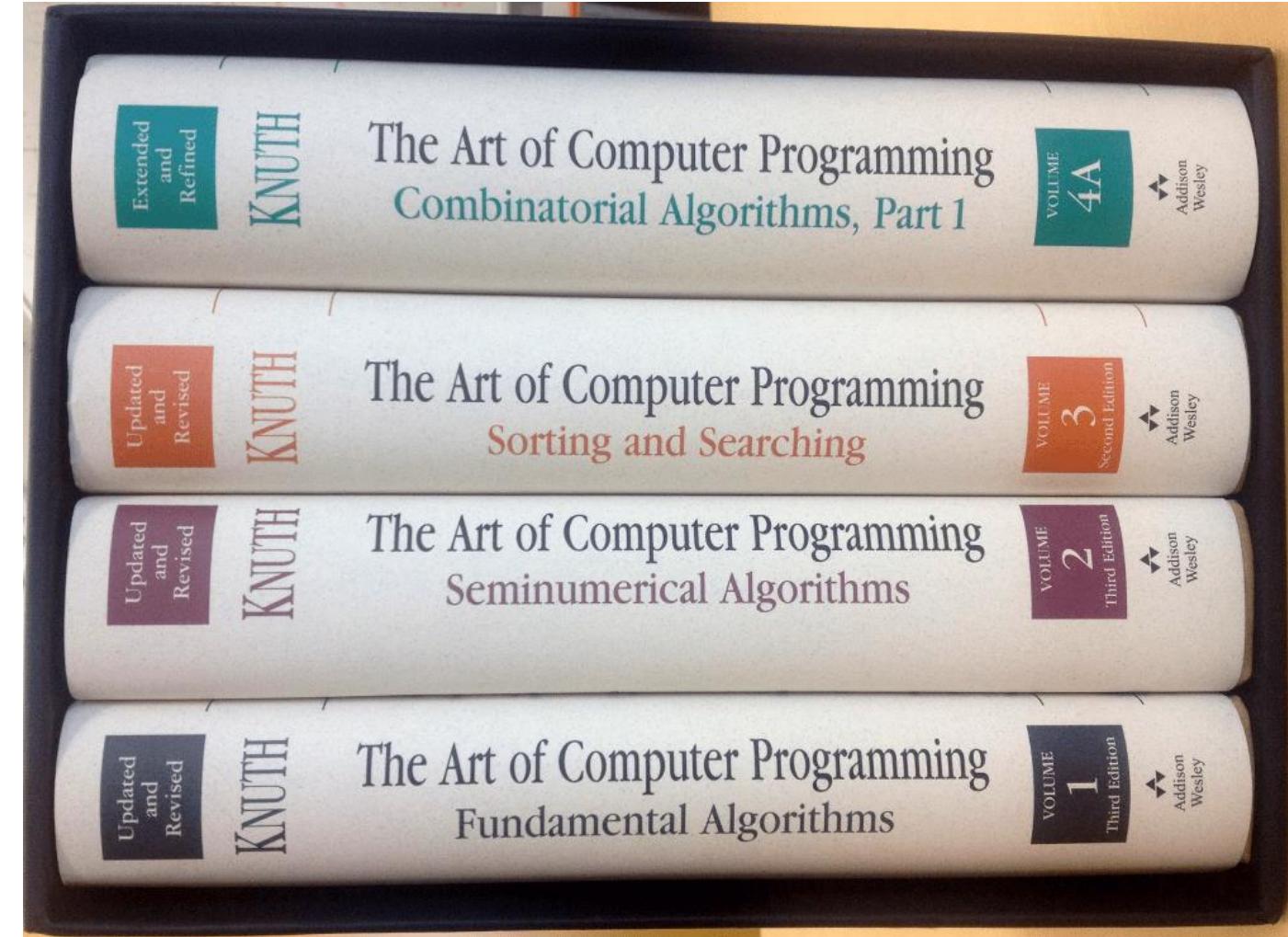
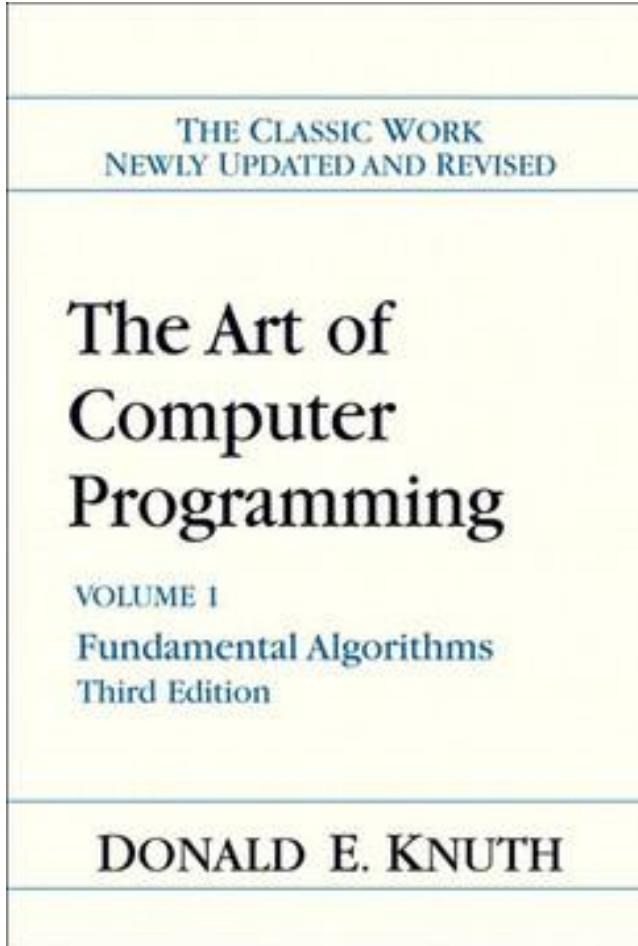


# Books ...

Accredited with **A** Grade by NAAC

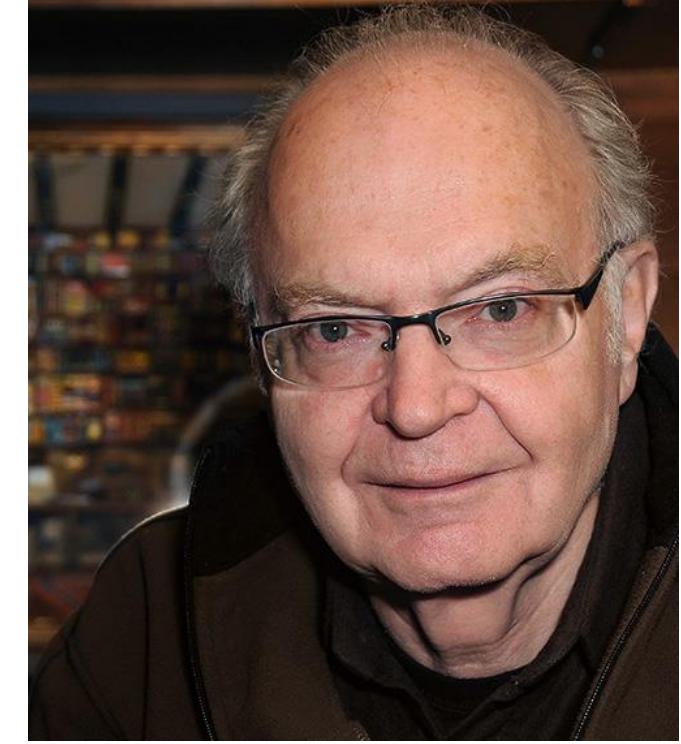


# Books ...



# Books ...

**Donald Ervin Knuth** has been called the "father" of the **Analysis of Algorithms**. He contributed to the development of the rigorous analysis of the computational complexity of algorithms and systematized formal mathematical techniques for it. In the process he also popularized the asymptotic notation.



Donald Ervin Knuth  
Professor, Stanford University  
Recipient of Turing Award

# Course Objective

**This course introduces the basics of computational complexity analysis and various algorithm design paradigms.**

The goal is to provide students with solid foundations to deal with a wide variety of computational problems, and to provide a thorough knowledge of the most common algorithms and data structures.

# Algorithm: Introduction

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.



# Algorithm: Introduction ...

- An algorithm is a sequence of computational steps that transform input to desired output.

**(Thomas H. Cormen)**

- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task

**(Sartaj Sahni)**

# Algorithm: Introduction ...

## Difference between Algorithm and Program

Algorithm	Program
Written at Design stage	Written at implementation stage
Need domain expert	Need programmer
Written in any language	Written in programming language
H/W or OS independent	H/W or OS dependent
Can be Analyze	Can be tested

# Algorithm: Introduction ...

## Criteria (properties) of an algorithm

- All algorithms must satisfy the following five criteria:
  - **Input** – zero or more quantities are externally supplied
  - **Output** – Atleast one quantity is produced
  - **Definiteness** – Each instruction should be clear and unambiguous
  - **Finiteness** – The algorithm should terminate after a finite number of steps
  - **Effectiveness** – Every instruction must be very basic. It is also always feasible.

# Pseudo-Code

- Pseudo-code is a description of an algorithm that is more structured than usual prose but less formal than a programming language.
- **Example: swapping two numbers.**

Algorithm SWAP (a, b)

Begin

Temp :=a;

a :=b;

b :=temp;

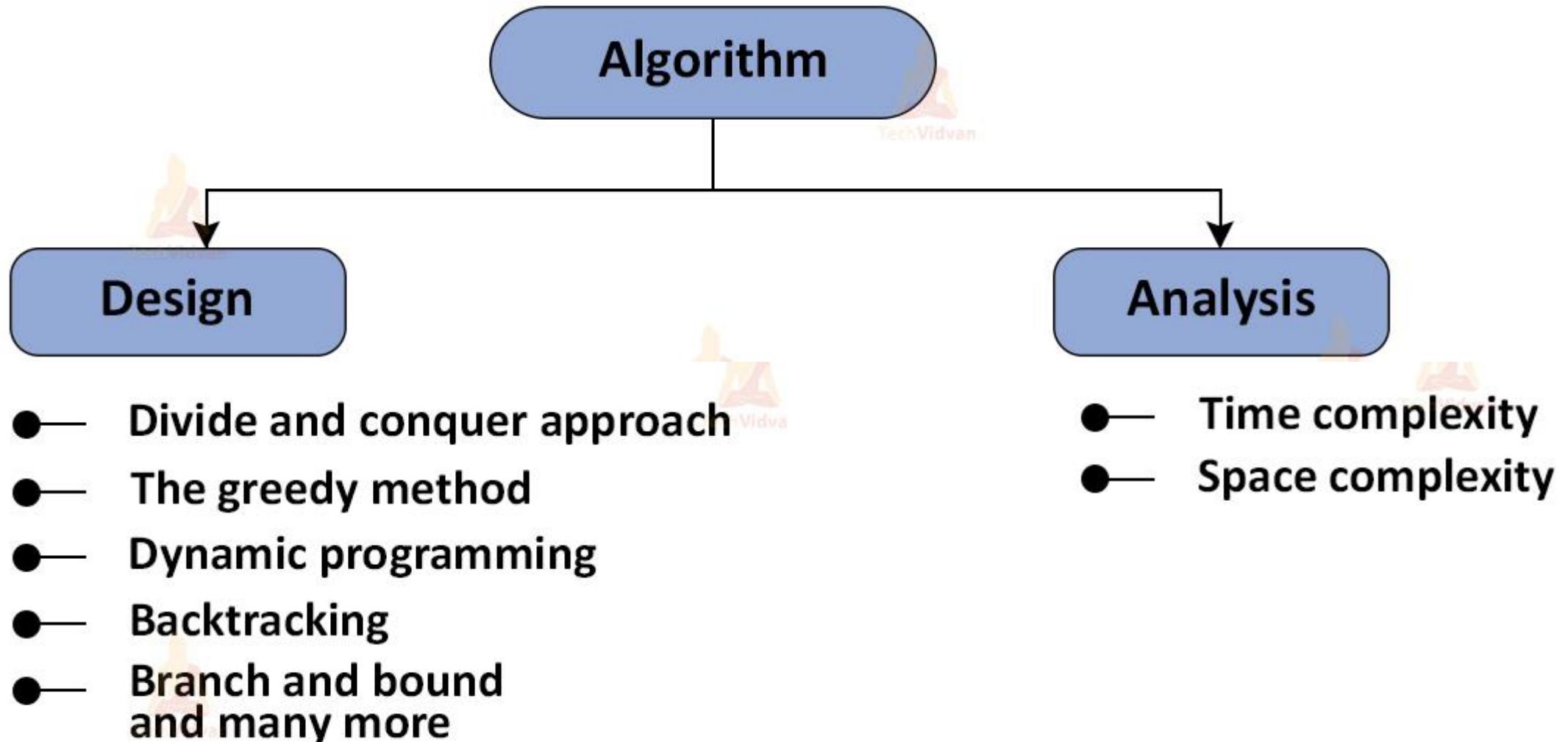
End

- Pseudo-code is our preferred notation for describing algorithms.
- However, pseudo-code hides program design issues.

# What is Pseudo-Code ?

- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm.
  - Expressions: use standard mathematical symbols to describe numeric and boolean expressions   -use  $\leftarrow$  for assignment (“=” in Java)
    - use  $=$  for the equality relationship (“==” in Java)
  - Method Declarations:
    - Algorithm name(*param1, param2*)
  - Programming Constructs:
    - decision structures: **if ... then ... [else ... ]**
    - while-loops: **while ... do**
    - repeat-loops: **repeat ... until ...**
    - for-loop: **for ... do**
    - array indexing: **A[i]**
  - Methods:
    - calls: object method(args)
    - returns: **return value**

# Two Aspects of Algorithm



# Analysis of Algorithms

- **Elementary operation (Primitive Operation):** It is an operation whose execution time can be bounded above by a constant depending only on the particular implementation used (machine, programming language, and so on).
- Examples:
  - calling a method and returning from a method
  - arithmetic operations (e.g. addition)
  - comparing two numbers, etc.
- **By inspecting the pseudo-code, we can count the number of primitive operations executed by an algorithm.**

# Algorithm: Introduction ...

## How to Analyze an Algorithm

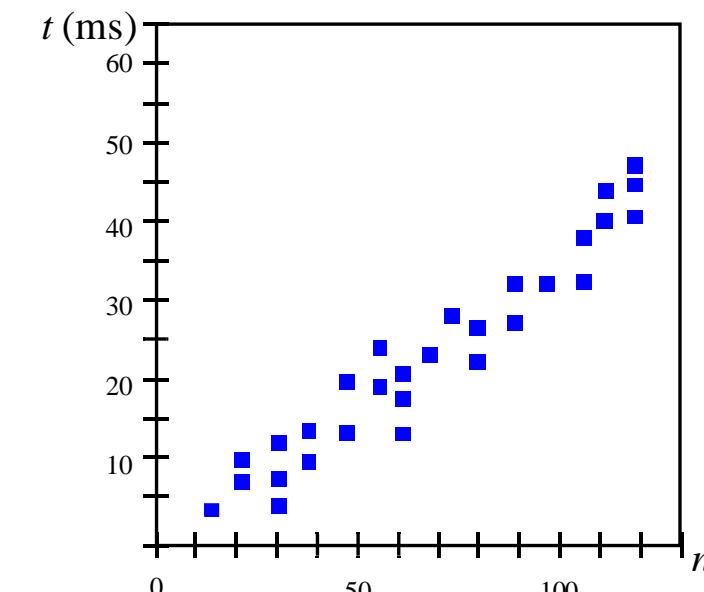
1. Time
2. Space
3. N/W
4. Power
5. CPU

Time complexity of an algorithm is the amount of computer time it needs to run the completion.

Space Complexity is an amount of memory requirement for an algorithm to completion of its execution.

# Measuring the Running Time

- How should we measure the running time of an **algorithm**?
- **Approach 1: Experimental Study**
  - Write a **program** that implements the algorithm
  - Run the program with data sets of varying size and composition.
  - Use a method like **System.currentTimeMillis()** to get an accurate measure of the actual running time.



# Limitations of Experimental Studies

- Experimental studies have several limitations:
  - It is necessary to **implement** and test the algorithm in order to determine its running time.
  - Experiments can be done only on a **limited set of inputs**, and may not be indicative of the running time on other inputs not included in the experiment.
  - In order to compare two algorithms, the same **hardware and software environments** should be used.

# Beyond Experimental Studies

- We will now develop a **general methodology** for analyzing the running time of algorithms. In contrast to the "experimental approach", this methodology:
  - Uses a **high-level description** of the algorithm instead of testing one of its implementations.
  - Takes into account **all possible inputs**.
  - Allows one to evaluate the efficiency of any algorithm in a way that is **independent from the hardware and software environment**.

# Algorithm: Introduction ...

## Frequency Count Method

**Ex 1)**

```

1. for (i=0; i<n; i++)
2. {
3.     Stat;
4. }
```

Time Analysis	
Instruction	Time Unit
1	$n+1$
3	$n$
<b>Total</b>	<b><math>2n+1</math></b>
<b>Degree</b>	<b><math>O(n)</math></b>

# Algorithm: Introduction ...

## Frequency Count Method

**Ex 2)**

```
1. for (i=n; i>0; i--)  
2. {  
3.     Stat;  
4. }
```

Time Analysis	
Instruction	Time Unit
1	$n+1$
3	$n$
Total	$2n+1$
Degree	$O(n)$

# Algorithm: Introduction ...

## Frequency Count Method

**Ex 3)**

```
1. for (i=0; i<n; i=i+2)
2. {
3.     Statement;
4. }
```

Time Analysis	
Instruction	Time Unit
1	$n/2+1$
3	$n/2$
<b>Total</b>	<b><math>n+1</math></b>
<b>Degree</b>	<b><math>O(n)</math></b>

# Algorithm: Introduction ...

## Ex 4)

```

1. for (i=0; i<n; i++)
2. {
3.     for (j=0; j<i; j++)
4.     {
5.         Statement;
6.     }
7. }
```

$$\text{Total Time} = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$f(n) = O(n^2)$$

Time Analysis		
i	j	No. of Time
0	0	0
	0	
1	0	1
	1	
	0	2
2	1	
	2	
	-	-
n	-	
	-	

# Algorithm: Introduction ...

**Ex 5)**

```

1. P=0;
2. for (i=1; P<=n; i++)
3. {
4.     P=P+i;
5. }
```

**Assume that the loop will terminate when  $P>n$**

$$P = \frac{k(k+1)}{2} \Rightarrow \frac{k(k+1)}{2} > n \Rightarrow k^2 > n$$

$$k > \sqrt{n} \Rightarrow f(n) = O(\sqrt{n})$$

Time Analysis	
i	P
1	$0+1=1$
2	$1+2=3$
3	$1+2+3$
4	$1+2+3+4$
.	.
.	.
.	.
k	$1+2+3....+k$

# Algorithm: Introduction ...

**Ex 6)**

```

1. for (i=1; i<n; i=i*2)
2. {
3.     Statement;
4. }
```

**Assume that the loop will terminate when  $i \geq n$**

$$2^k \geq n \Rightarrow 2^k = n \Rightarrow k = \log_2 n$$

$$f(n) = O(\log_2 n)$$

Time Analysis	
iteration	i
1	1
2	2
3	$2^2$
4	$2^3$
.	.
.	.
.	.
k-1	$2^k$

# Algorithm: Introduction ...

## Frequency Count Method

- Ex 7) Algorithm for sum of elements of an array

### **Algorithm\_Sum(A, n)**

```
//A is an array of size n
1. {
2.     S=0;
3.     for (i=0; i<n; i++)
4.     {
5.         S=S+A[i]
6.     }
7.     return S;
8. }
```

Time Analysis		Space Analysis	
Instruction	Time Unit	Variable	Space Unit
2	1	A	n
3	$n+1$	n	1
5	n	S	1
7	1	i	1
<b>Total</b>	<b><math>2n+3</math></b>	<b>Total</b>	<b><math>n+3</math></b>
<b>Degree</b>	<b><math>O(n)</math></b>	<b>Degree</b>	<b><math>O(n)</math></b>

# Algorithm: Introduction ...

## Frequency Count Method

- Ex 8) Algorithm for sum of two matrix

**Algorithm\_Sum\_MAT(A, B, n)**

```
//A, B is an array of size
1. {
2. for (i=0; i<n; i++)
3. {
4.   for (j=0; j<n; j++)
5.   {
6.     C[i, j]=A[i, j]+B[i, j]
7.   }
8. }
```

Time Analysis	
Instruction	Time Unit
2	$n+1$
4	$n(n+1)$
6	$n*n$
<b>Total</b>	<b><math>2n^2+2n+1</math></b>
<b>Degree</b>	<b><math>O(n^2)</math></b>

Space Analysis	
Variable	Space Unit
A	$n^2$
B	$n^2$
C	$n^2$
n	1
i	1
J	1
<b>Total</b>	<b><math>3n^2+3</math></b>
<b>Degree</b>	<b><math>O(n^2)</math></b>

# Algorithm: Introduction ...

## Frequency Count Method

### Ex 9) Algorithm for multiplication of two matrix

```
Algorithm_Mul_MAT(A, B, n)
```

```
//A is an array of size n
```

```
1. {
2.   for (i=0; i<n; i++)
3.   {
4.     for (j=0; j<n; j++)
5.     C[I,j]=0;
6.     for (k=0; k<n; k++)
7.     {
8.       C[i,j]=C[i,j] + A[i,k]*B[k,j]
9.     }
10.  }
11. }
```

Time Analysis		Space Analysis	
Instruction	Time Unit	Variable	Space Unit
2	$n+1$	A	$n^2$
4	$n(n+1)$	B	$n^2$
5	$n*n$	C	$n^2$
6	$(n+1)n*n$	n	1
8	$n*n*n$	i	1
Total	$2n^3+3n^2+2n$	j	1
Degree	$O(n^3)$	k	1
		Total	$3n^2+4$
		Degree	$O(n^2)$

# Algorithm: Introduction ...

## Ex 10)

```

1. for (i=n; i>=1; i=i/2)
2. {
3.     Statement;
4. }
```

**Assume that the loop will terminate when  $i < 1$**

$$n / 2^k < 1 \Rightarrow n / 2^k = 1 \Rightarrow n = 2^k$$

$$k = \log_2 n \Rightarrow f(n) = O(\log_2 n)$$

Time Analysis	
iteration	i
1	n
2	$n/2$
3	$n/2^2$
4	$n/2^3$
.	.
.	.
.	.
k-1	$n/2^k$

# Algorithm: Introduction ...

## Ex 11)

```

1. for (i=0; i<n; i++)
2. {
3.     Statement;
4. }
5. for (j=0; j<n; j++)
6. {
7.     Statement;
8. }
```

Time Analysis	
Instruction	Time Unit
1	$n+1$
3	$n$
5	$n+1$
7	$n$
<b>Total</b>	<b><math>4n+2</math></b>
<b>Degree f(n)</b>	<b><math>O(n)</math></b>

# Algorithm: Introduction ...

## Ex 12)

```
1. for (i=1; i<n; i++) _____ n+1
2. {
3.     for (j=1; j<n; j=j * 2) _____ n*log2n
4.     {
5.         Statement; _____ n*log2n
6.     }
```

$$\text{Total Time} = 2n \log_2 n + n$$

$$f(n) = O(n \log_2 n)$$

# Algorithm: Introduction ...

## Ex 14) While loop

```
1 . i=0; _____ 1
2 . while (i<n) _____ n
3 . {
4 .   Statement; _____ n
5 .   i++; _____ n
6 . }
```

While loop and for loop are similar.

Do while is different from these two. As it execute at least one time even the condition is false.

$$\text{Total Time} = 3n + 2$$

$$f(n) = O(n)$$



# Algorithm: Introduction ...

## Ex 13) While loop

```
1 . a=1;  
2 . while (a<b)  
3 . {  
4 .   Statement;  
5 .   a=a*2  
6 . }
```

Value of a
1
$1*2=2$
$2*2=2^2$
$2^2*2=2^3$
$2^k$

the loop will terminate when  $a \geq b$

$$\because a=2^k \Rightarrow 2^k \geq b \Rightarrow$$

$$2^k = b \Rightarrow k=\log_2 b \Rightarrow f(n)=O(\log_2 n)$$

# Algorithm: Introduction ...

**Ex 14)** GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.

```

1.while  (m!=n)
2. {
3.   if  (m>n)
4.     m=m-n;
5.   else
6.     n=n-m;
7. }
```

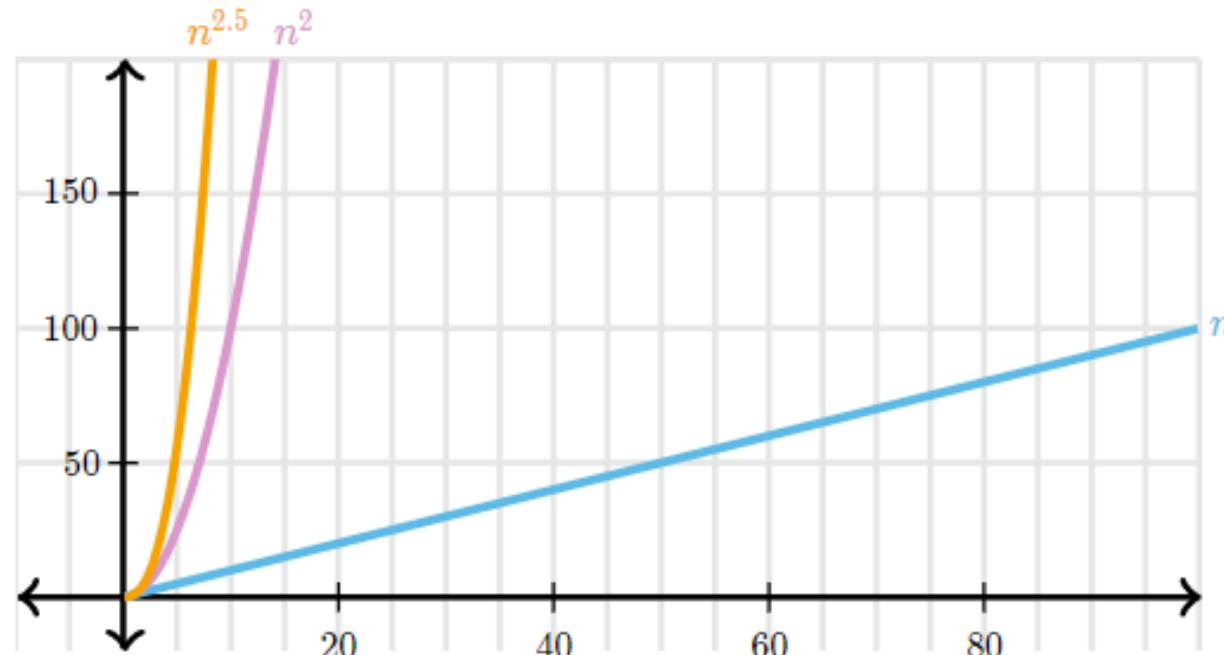
Since it a linear method so  
 $f(n) = O(n)$

Example	
m	n
36	60
36	24
12	24
12	12

This is the worst-case complexity, because the value  $x + y$  decreases with every step.

# Algorithm: Growth of a Function

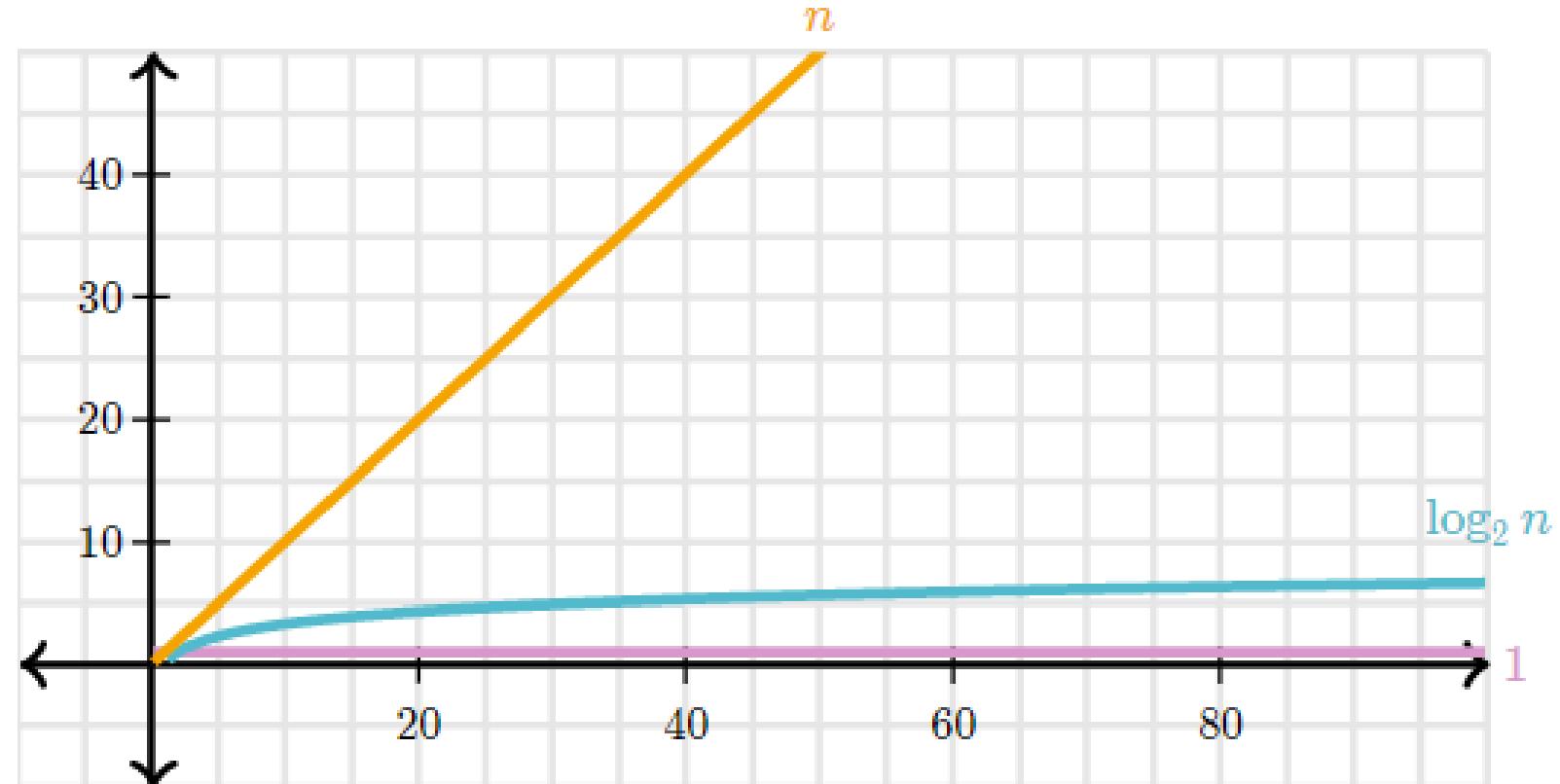
The following graph compares the growth of  $n$ ,  $n^2$ , and  $n^{2.5}$ :



Logarithms grow more slowly than polynomials. That is,  $\Theta(\log_2 n)$  grows more slowly than  $\Theta(n^a)$  for any positive constant  $a$ . But since the value of  $\log_2 n$  increases as  $n$  increases,  $\Theta(\log_2 n)$  grows faster than  $\Theta(1)$ .

# Algorithm: Growth of a Function ...

The following graph compares the growth of  $1$ ,  $n$ , and  $\log_2 n$ :





# Algorithm: Growth of a Function ...

$$1 < \log_2 n < \sqrt{n} < n < n \log_2 n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$$

$O(1)$	Constant
$O(\log_2 n)$	Logarithmic
$O(n)$	Linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential

n	$\log_2 n$	$n^2$	$2^n$
1	0	1	2
2	1	4	4
4	2	16	16
8	3	64	256

# Algorithm: Growth of a Function ...

## Example

- \* Sorting an array with  $n$  elements
  - \* Naïve algorithms : time proportional to  $n^2$
  - \* Best algorithms : time proportional to  $n \log n$
- \* How important is this distinction?
- \* Typical CPUs process up to  $10^8$  operations per second
- \* Useful for approximate calculations

# Algorithm: Growth of a Function ...

## Example

- \* Telephone directory for mobile phone users in India
  - \* India has about 1 billion =  $10^9$  phones
  - \* Naïve  $n^2$  algorithm requires  $10^{18}$  operations
    - \*  $10^8$  operations per second  $\Rightarrow 10^{10}$  seconds
    - \* 2778000 hours
    - \* 115700 days
    - \* 300 years!
  - \* Smart  $n \log n$  algorithm takes only about  $3 \times 10^{10}$  operations
    - \* About 300 seconds, or 5 minutes



# Algorithm: Introduction ...

## General steps to develop an Algorithm

**There are four steps:**

- To devise an algorithm
- Validate an algorithm
- Analyze an algorithm
- Test an algorithm
  - ✓ Debugging
  - ✓ Profiling

# Review on Summations

Why do we need summation formulas?

- For computing the running times of iterative constructs
- **Constant Series:** For integers  $a$  and  $b$ ,  $a \leq b$ ,

$$\sum_{i=a}^b 1 = b - a + 1$$

- **Linear Series (Arithmetic Series):** For  $n \geq 0$ ,

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

- **Quadratic Series:** For  $n \geq 0$ ,

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

# Review on Summations ...

- **Cubic Series:** For  $n \geq 0$ ,

$$\sum_{i=1}^n i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- **Geometric Series:** For real  $x \neq 1$ ,

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For  $|x| < 1$ ,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

# Review on Summations ...

- **Linear-Geometric Series:** For  $n \geq 0$ , real  $c \neq 1$ ,

$$\sum_{i=1}^n ic^i = c + 2c^2 + \cdots + nc^n = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^2}$$

- **Harmonic Series:**  $n$ th harmonic number,  $n \in \mathbb{I}^+$ ,

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &= \sum_{k=1}^n \frac{1}{k} = \ln(n) + O(1) \end{aligned}$$

Which kind of growth best characterizes each of these functions?

**Q1)**

	Constant	Linear	Polynomial	Exponential
--	----------	--------	------------	-------------

$(3/2)^n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
-----------	-----------------------	-----------------------	-----------------------	-----------------------

1	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
---	-----------------------	-----------------------	-----------------------	-----------------------

$(3/2)n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
----------	-----------------------	-----------------------	-----------------------	-----------------------

$2n^3$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
--------	-----------------------	-----------------------	-----------------------	-----------------------

$2^n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
-------	-----------------------	-----------------------	-----------------------	-----------------------

$3n^2$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
--------	-----------------------	-----------------------	-----------------------	-----------------------

1000	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
------	-----------------------	-----------------------	-----------------------	-----------------------

$3n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
------	-----------------------	-----------------------	-----------------------	-----------------------

Which kind of growth best characterizes each of these functions?

	Constant	Linear	Polynomial	Exponential
--	----------	--------	------------	-------------

$(3/2)^n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
-----------	-----------------------	-----------------------	-----------------------	----------------------------------

1	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
---	----------------------------------	-----------------------	-----------------------	-----------------------

$(3/2)n$	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
----------	-----------------------	----------------------------------	-----------------------	-----------------------

$2n^3$	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
--------	-----------------------	-----------------------	----------------------------------	-----------------------

$2^n$	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
-------	-----------------------	-----------------------	-----------------------	----------------------------------

$3n^2$	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
--------	-----------------------	-----------------------	----------------------------------	-----------------------

1000	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
------	----------------------------------	-----------------------	-----------------------	-----------------------

$3n$	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
------	-----------------------	----------------------------------	-----------------------	-----------------------

1 / 4

A function has "constant" growth if its output does not change based on the input, the  $n$ . The easy way to identify constant functions is find those that have no  $n$  in their expression anywhere, or have  $n^0$ . In this case, 1 and 1000 are constant.

2 / 4

A function has "linear" growth if its output increases linearly with the size of its input. The way to identify linear functions is find those where  $n$  is never raised to a power (although  $n^1$  is OK) or used as a power. In this case,  $3n$  and  $(3/2)n$  are linear.

3 / 4

A function has "polynomial" growth if its output increases according to a polynomial expression. The way to identify polynomial functions is to find those where  $n$  is raised to some constant power. In this case,  $2n^3$  and  $3n^2$  are polynomial.

4 / 4

A function has "exponential" growth if its output increases according to an exponential expression. The way to identify exponential functions is to find those where a constant is raised to some expression involving  $n$ . In this case,  $2^n$  and  $(3/2)^n$  are exponential.

## Q2)

Rank these functions according to their growth, from slowest growing (at the top) to fastest growing (at the bottom).

$n$

$n^3$

$1$

$(3/2)^n$

$n^2$

$2^n$

$1$

$n$

$n^2$

$n^3$

$(3/2)^n$

$2^n$

Rank these functions according to their growth, from slowest growing to fastest growing.

**Q3)**

$4n$

$8n^2$

$6n^3$

$64$

$n \log_2 n$

$n \log_6 n$

$\log_2 n$

$\log_8 n$

$8^{2n}$

$64$

$\log_8 n$

$\log_2 n$

$4n$

$n \log_6 n$

$n \log_2 n$

$8n^2$

$6n^3$

$8^{2n}$

“Thank you”

*Any Questions ?*

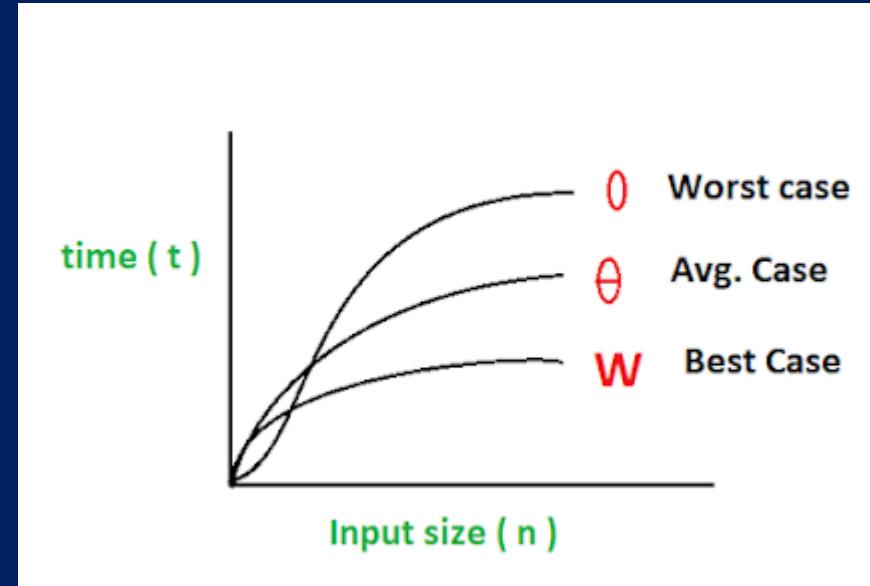


**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 2: Asymptotic Notations



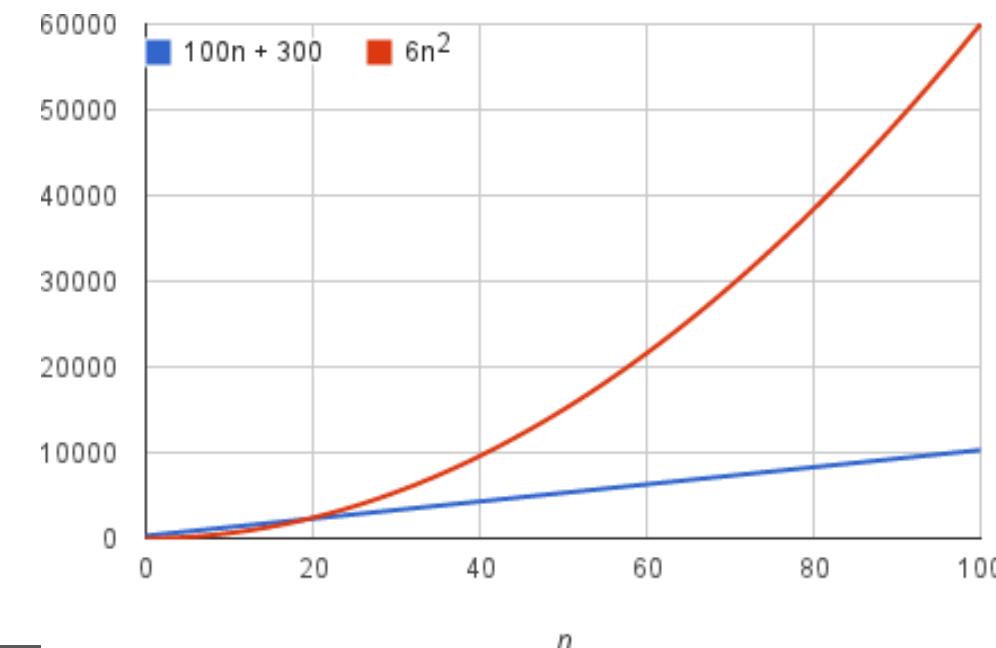
Prof. Anand Singh Jalal

**Department of Computer Engineering & Applications**

# Rate of Growth

$6n^2 + 100n + 300$  machine instructions. The  $6n^2$  term becomes larger than the remaining terms,  $100n + 300$ , once  $n$  becomes large enough, 20 in this case. Here's a chart showing values of  $6n^2$  and  $100n + 300$  for values of  $n$  from 0 to 100:

By dropping the less significant terms and the constant coefficients, we can focus on the important part of an algorithm's running time—its rate of growth



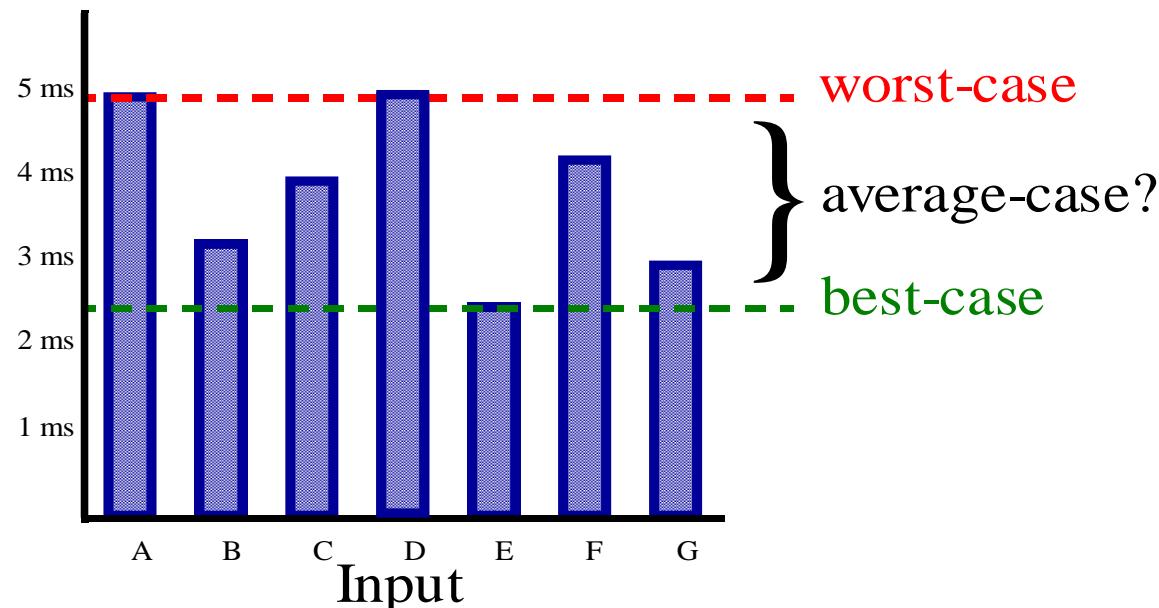


# Algorithm: Introduction

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

# Average Case vs. Worst Case Running Time of an algorithm

- An algorithm may run faster on certain data sets than on others.
- Finding the **average case** can be very difficult, so typically algorithms are measured by the **worst-case** time complexity.
- Also, in certain application domains (e.g., air traffic control, surgery, IP lookup) knowing the **worst-case** time complexity is of crucial importance.



# Asymptotic Notation

The main idea of asymptotic analysis is to have a measure of efficiency of algorithms that doesn't depend on machine specific constants, and doesn't require algorithms to be implemented and time taken by programs to be compared.

Asymptotic notations are the mathematical notations used to describe the running time (time complexity) of an algorithm when the input tends towards a particular value or a limiting value.

**Asymptotic notations show the class of a function**

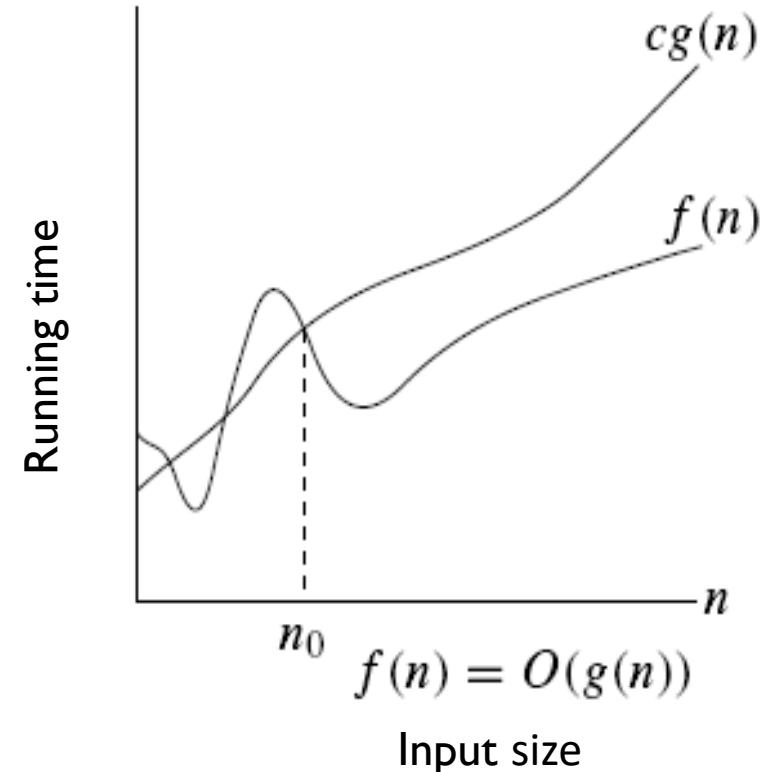


# Asymptotic Notation ...

- **Practical Significance**
- **Big Omega :**
  - Best case
  - Never achieve better than this
- **Big Theta:**
  - Average case
- **Big O:**
  - Worst case
  - Upper bound
  - Time must not exceed

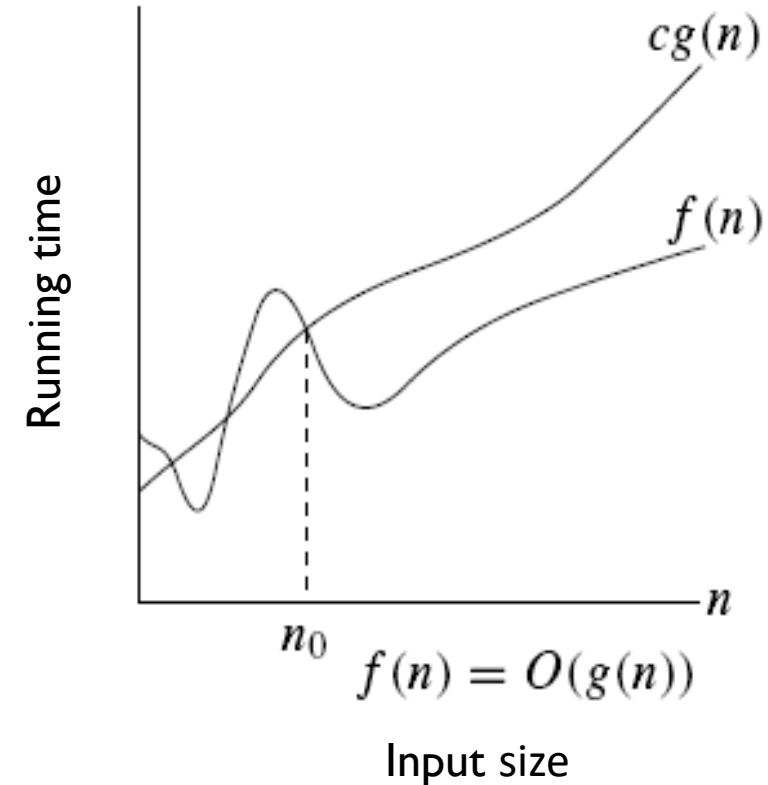
# Asymptotic Notation ...

- **The Big-Oh (O) Notation**
  - **Asymptotic upper bound**
  - $f(n)=O(g(n))$  , if there exists constants  $c > 0$  and  $n_0 \geq 1$  , s.t.  $f(n) \leq c g(n)$  for  $n \geq n_0$
  - $f(n)$  and  $g(n)$  are functions over non-negative integers.
  - **Used for worst case analysis**



# Asymptotic Notation ...

- **The Big-Oh ( $O$ ) Notation**
  - Asymptotic upper bound
  - if  $f$  &  $g$  be the functions then  
if  $\lim_{n \rightarrow \infty} f(n)/g(n) = c < \infty$   
Then  
 $f(n) \in O(g(n))$



# Asymptotic Notation ...

For example:  $f(n) = 3n+2$     $g(n)=n$

$$f(n)=O(g(n))$$

$$f(n) \leq c \cdot g(n)$$

$$3n+2 \leq c \cdot n$$

$$c=4, n_0 = 2$$

$$\therefore f(n) = O(n)$$

$$\begin{aligned} 3n+2 &\leq 3n+2n & \forall n \geq 1 \\ 3n+2 &\leq 5n \\ 3n+2 &\leq 5n^2 \end{aligned}$$

$$\therefore f(n) = O(n^2)$$

Since  $O(n)$  is closest bound so we will take  $O(n)$

$$1 < \log_2 n < \sqrt{n} < n < n \log_2 n < n^2 < n^3 < \dots < 2^n < 3^n \dots < n^n$$

# Asymptotic Notation ...

- **For example:  $f(n) = 3n+2 \quad g(n)=n$**

- $f(n)= O(g(n))$

$$\begin{aligned}\lim_{n \rightarrow \infty} f(n)/g(n) &= c < \infty \\ &= \lim_{n \rightarrow \infty} (3n+2)/n \\ &= \lim_{n \rightarrow \infty} (3+2/n) \\ &= \lim_{n \rightarrow \infty} (3+2/\infty) \\ &= 3 + 2/(1/0) \\ &= 3 + 2*0/1 \\ &= 3+0 \\ &= 3 < \infty\end{aligned}$$

Hence, we can say  $f(n)=O(g(n))$   
 $f(n)=O(n)$

# Big-Oh and Growth Rate

- The big-Oh notation gives an **upper bound on the growth rate** of a function
- The statement “ $f(n)$  is  $O(g(n))$ ” means that the growth rate of  $f(n)$  is no more than the growth rate of  $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

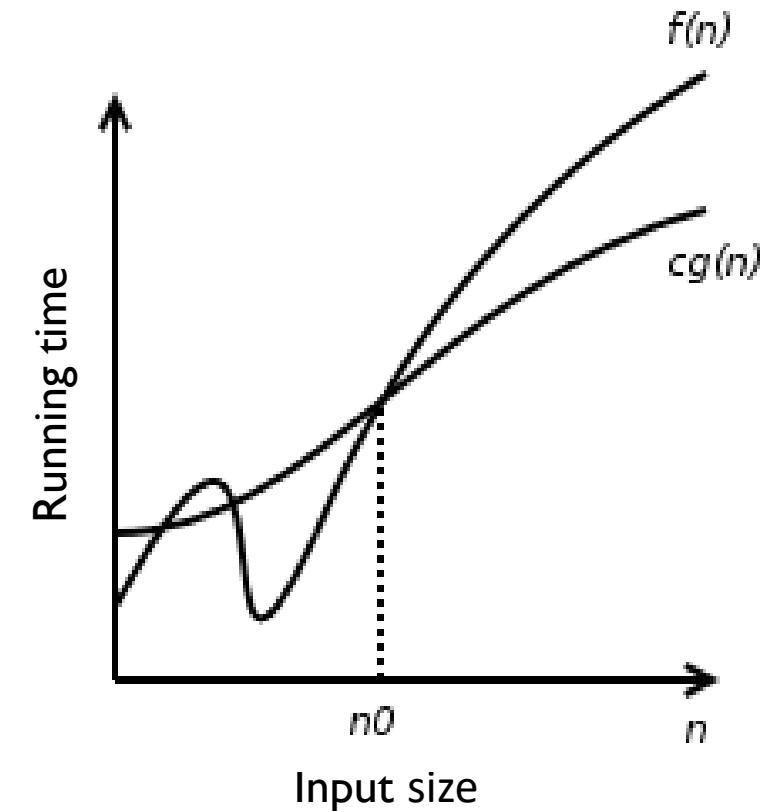
	$f(n)$ is $O(g(n))$	$g(n)$ is $O(f(n))$
$g(n)$ grows more	Yes	No
$f(n)$ grows more	No	Yes
Same growth	Yes	Yes

# Big-Oh Rules

- If  $f(n)$  is a polynomial of degree  $d$ , then  $f(n)$  is  $O(n^d)$ , i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say “ $2n$  is  $O(n)$ ” instead of “ $2n$  is  $O(n^2)$ ”
  - Use the simplest expression of the class
    - Say “ $3n + 5$  is  $O(n)$ ” instead of “ $3n + 5$  is  $O(3n)$ ”

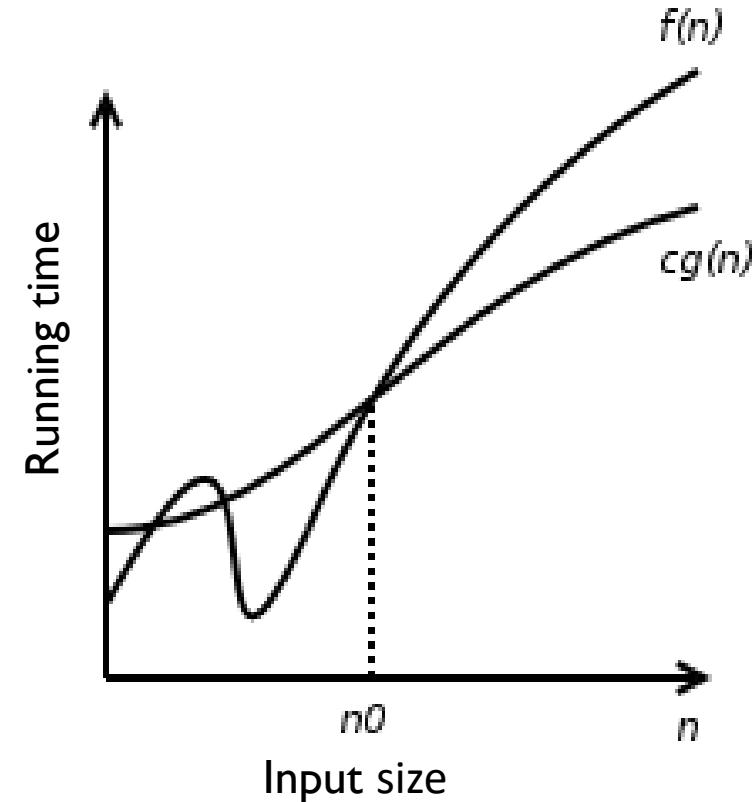
# Asymptotic Notation ...

- **The Big-Omega ( $\Omega$ ) Notation**
  - Asymptotic lower bound
  - $f(n) = \Omega(g(n))$  , if there exists constants  $c > 0$  and  $n_0 \geq 1$  ,  
s.t.  $f(n) \geq c g(n) \geq 0$  for  $n \geq n_0$
  - $f(n)$  and  $g(n)$  are functions over non-negative integers.
  - **Used for best case running time or lower bound of algorithmic problem.**



# Asymptotic Notation ...

- **The Big-Omega ( $\Omega$ ) Notation**
  - Asymptotic lower bound
  - if  $f$  &  $g$  be the functions then  
if  $\lim_{n \rightarrow \infty} f(n)/g(n) > 0$   
Then  
 $f(n) \in \Omega(g(n))$

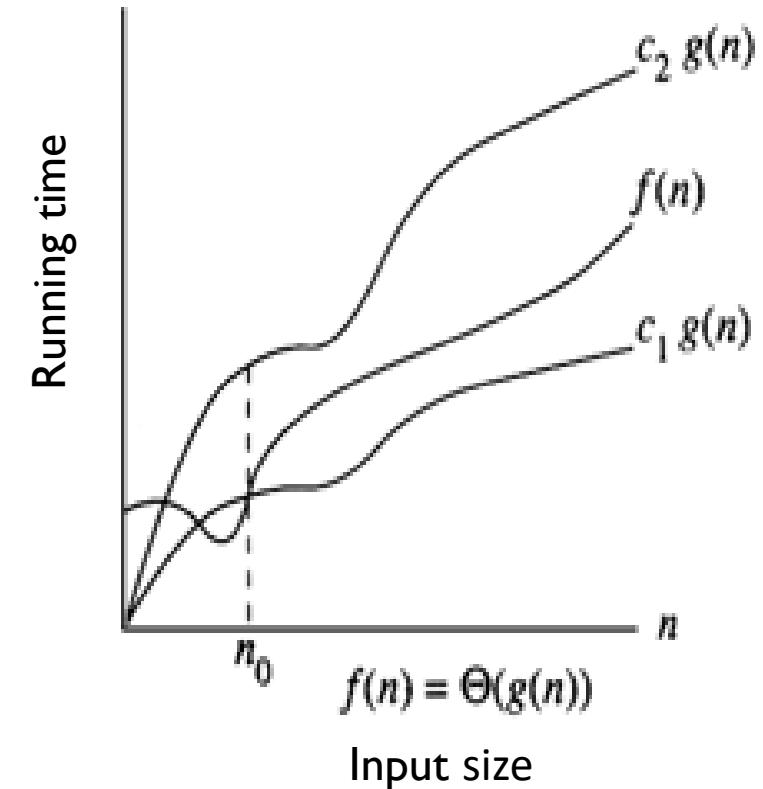


# Asymptotic Notation ...

- For example:  $f(n) = 3n+2 \quad g(n)=n$
- $f(n)=\Omega(g(n))$   
 $f(n) \geq c \cdot g(n)$   
 $3n+2 \geq c \cdot n$   
 $c=1, n_0 = 1 \quad n \geq 1$

# Asymptotic Notation ...

- **The Big-Theta ( $\Theta$ ) Notation**
  - Asymptotically tight bound
  - $f(n) = \Theta(g(n))$ , if there exists constants  $c_1$  &  $c_2$  and  $n_0$ , s.t.  
 $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$   
for  $n \geq n_0$
  - $f(n)$  and  $g(n)$  are functions over non-negative integers.
  - **Used for average case running time**



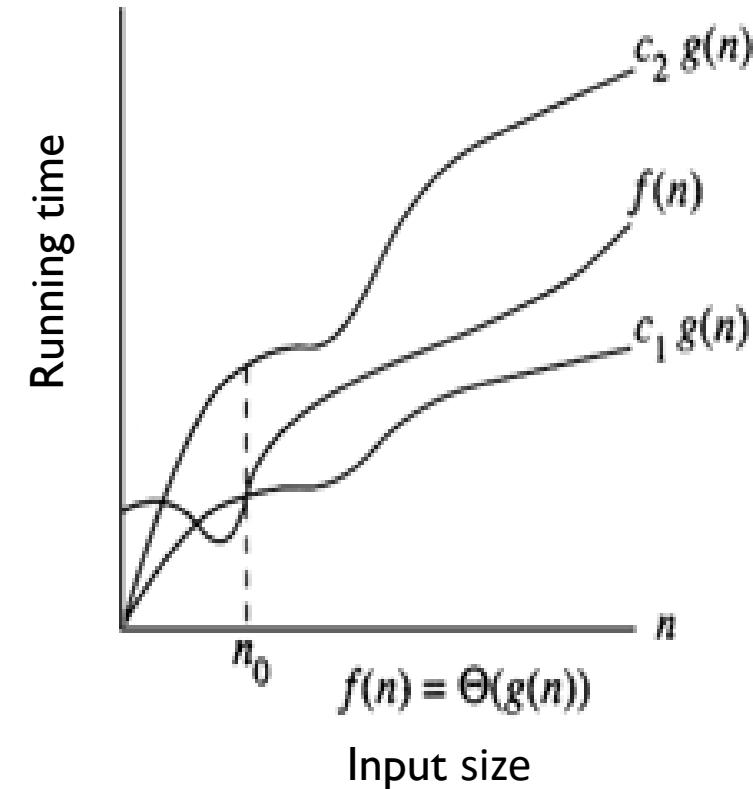
# Asymptotic Notation ...

- **The Big-Theta ( $\Theta$ ) Notation**

- Asymptotically tight bound
- if  $f$  &  $g$  be the functions then  
if  $\lim_{n \rightarrow \infty} f(n)/g(n) < \infty$

Then,

$f(n) \in \Theta(g(n))$



# Asymptotic Notation ...

- **For example:**  $f(n) = 3n+2 \quad g(n)=n$

- $f(n) = \Theta(g(n))$

- $c_1.g(n) \leq f(n) \leq c_2. g(n)$

$$f(n) \leq c_2 . g(n)$$

$$3n+2 \leq c_2 . n$$

$$c_2=4, n_0 \geq 1$$

$$f(n) \geq c_1. g(n)$$

$$3n+2 \geq c_1. n$$

$$c_1=1, n_0 \geq 1$$

$$\therefore f(n) = \Theta(n)$$

# Asymptotic Notation ...

Match each function with an equivalent function, in terms of their  $\Theta$ . Only match a function if  $f(n) = \Theta(g(n))$ .

$f(n)$	$g(n)$
$n + 30$	$n^2 + 3n$
$n^2 + 2n - 10$	$n^4$
$n^3 * 3n$	$\log_2 2x$
$\log_2 x$	$3n - 1$

$f(n)$	$g(n)$
$n + 30$	$3n - 1$
$n^2 + 2n - 10$	$n^2 + 3n$
$n^3 * 3n$	$n^4$
$\log_2 x$	$\log_2 2x$

“Thank you”

*Any Questions ?*



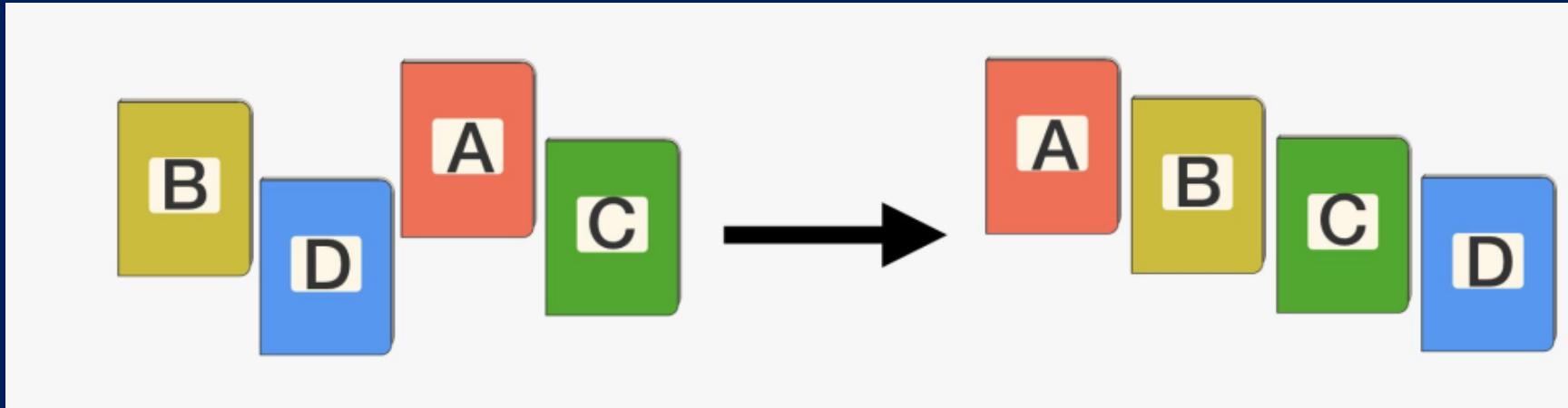
**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 4: Sorting

## Shell Sort



Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

# Shell Sort

- Invented by Donald Shell in 1959.
- 1<sup>st</sup> algorithm to break the quadratic time barrier but few years later, a sub quadratic time bound was proven
- Shell sort works by comparing elements that are **distant** rather than adjacent elements in an array.
  
- Shell sort uses a sequence  $h_1, h_2, \dots, h_t$  called the ***increment sequence***. Any increment sequence is fine as long as  $h_1 = 1$  and some other choices are better than others.

# Shell Sort

- Shell sort makes multiple passes through a list and sorts a number of equally sized sets using the insertion sort.
- Shell sort improves on the efficiency of insertion sort by shifting values to their destination.

Shell sort is also known as ***diminishing increment sort***.

The distance between comparisons decreases as the sorting algorithm runs until the last phase in which adjacent elements are compared

# Empirical Analysis of Shell sort (Advantage)

- Advantage of Shell sort is that its only efficient for medium size lists. For bigger lists, the algorithm is not the best choice. Fastest of all  $O(N^2)$  sorting algorithms.
- 5 times faster than the bubble sort and a little over twice as fast as the Insertion sort, its closest competitor.

# Empirical Analysis of Shell sort (Disadvantage)

- Disadvantage of Shell sort is that it is a complex algorithm and its not nearly as efficient as the **merge, heap, and quick sorts**.
- The shell sort is still significantly slower than the merge, heap, and quick sorts, but its relatively simple algorithm makes it a good choice for sorting lists of less than 5000 items unless speed important. It's also an excellent choice for repetitive sorting of smaller lists.

# Shell Sort

- Comparison based sorting technique.
- Starts with gap =  $\lfloor n/2 \rfloor$ , where n = no. of elements.
- One by one select elements to the right of the gap and place them at their appropriate position.

## Time Complexity

Best  $O(n \log n)$

Worst  $O(n^2)$

Average  $O(n \log n)$

**Space Complexity**  $O(1)$

# Shell Sort: Example

19	63	2	6	7	18	60	16	9	4
----	----	---	---	---	----	----	----	---	---

**Solution:**

$$N=10 ; \text{gap} = \lfloor n/2 \rfloor = 5$$

**PASS I:**

19	63	2	6	7	18	60	16	9	4
					↓				
18	63	2	6	7	19	60	16	9	4
18	60	2	6	4	19	63	16	9	7
18	60	2	6	4	19	63	16	9	7

# Shell Sort: Example ...

**Pass II:**

$$\text{gap} = \lfloor 5/2 \rfloor = 2$$

18	60	2	6	4	19	63	16	9	7
----	----	---	---	---	----	----	----	---	---



18	60	2	6	4	19	63	16	9	7
----	----	---	---	---	----	----	----	---	---



2	6	4	7	9	16	18	19	63	60
---	---	---	---	---	----	----	----	----	----



2	6	4	7	9	16	18	19	63	60
---	---	---	---	---	----	----	----	----	----

# Shell Sort: Example ...

**Pass III:**

$$\text{gap} = \lfloor 2/2 \rfloor = 1$$

2	6	4	7	9	16	18	19	63	60
---	---	---	---	---	----	----	----	----	----



2	6	4	7	9	16	18	19	63	60
---	---	---	---	---	----	----	----	----	----



2	4	6	7	9	16	18	19	60	63
---	---	---	---	---	----	----	----	----	----

# Shell Sort: Example 2

## EXAMPLE 2:

18	32	12	5	38	33	16	2
----	----	----	---	----	----	----	---

# Shell Sort: Example 2 ...

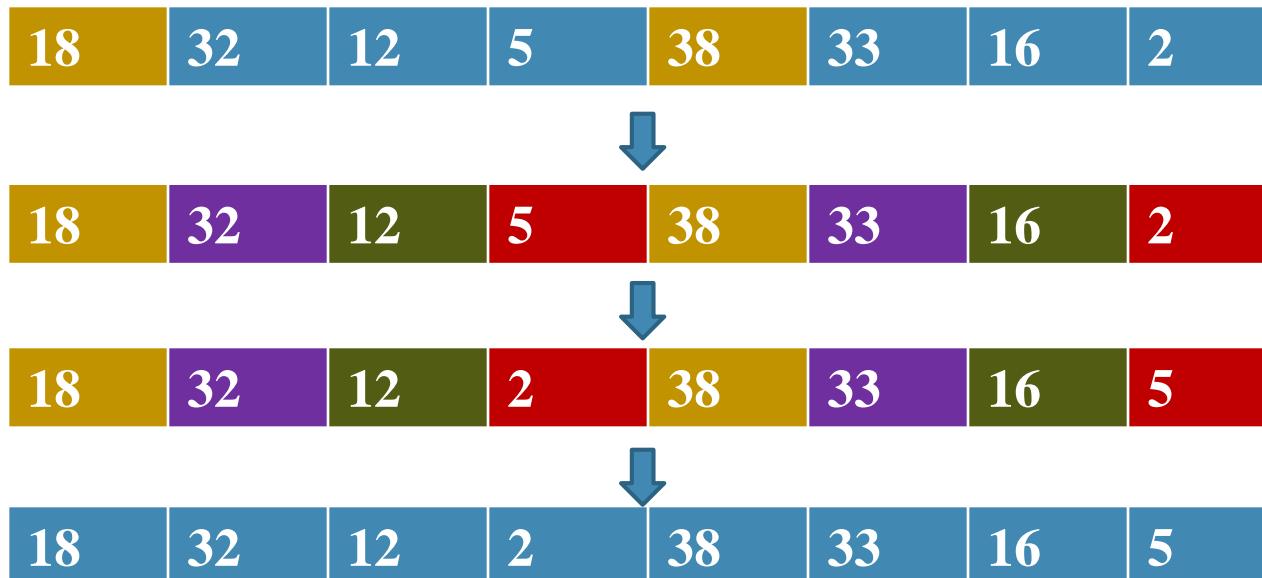
## EXAMPLE 2:

18	32	12	5	38	33	16	2
----	----	----	---	----	----	----	---

**Solution:**

$$N=8 ; \text{gap} = \lfloor n/2 \rfloor = 4$$

**PASS I:**



# Shell Sort: Example 2 ...

PASS II:

$$\text{gap} = \lfloor 4/2 \rfloor = 2$$

18	32	12	2	38	33	16	5
----	----	----	---	----	----	----	---



18	32	12	2	38	33	16	5
----	----	----	---	----	----	----	---



12		16		18		38	
----	--	----	--	----	--	----	--



	2		5		32		33
--	---	--	---	--	----	--	----



12	2	16	5	18	32	38	33
----	---	----	---	----	----	----	----

# Shell Sort: Example 2 ...

**PASS III:**

$$\text{gap} = \lfloor 2/2 \rfloor = 1$$

12		2		16		5		18		32		38		33
----	--	---	--	----	--	---	--	----	--	----	--	----	--	----



12		2		16		5		18		32		38		33
----	--	---	--	----	--	---	--	----	--	----	--	----	--	----



2		5		12		16		18		32		33		38
---	--	---	--	----	--	----	--	----	--	----	--	----	--	----



# Shell Sort: Algorithm

**Shell sort( A, n)**

**Step1:-** for ( $i = n/2$ ;  $i > 0$ ;  $i = i/2$ )

**Step2:-** for( $j=i$ ;  $j < n$ ;  $j++$ )

**Step3:-** for( $k=j-i$ ;  $k \geq 0$ ;  $k=k-i$ )

**If**( $A[k+i] \geq A[k]$ )

**break**;

**else exchange( A[k], A[k+i] )**

**Step4:-** End

Let the index is start from 0

2	1	5	8	10	3	6	16
---	---	---	---	----	---	---	----

“Thank you”

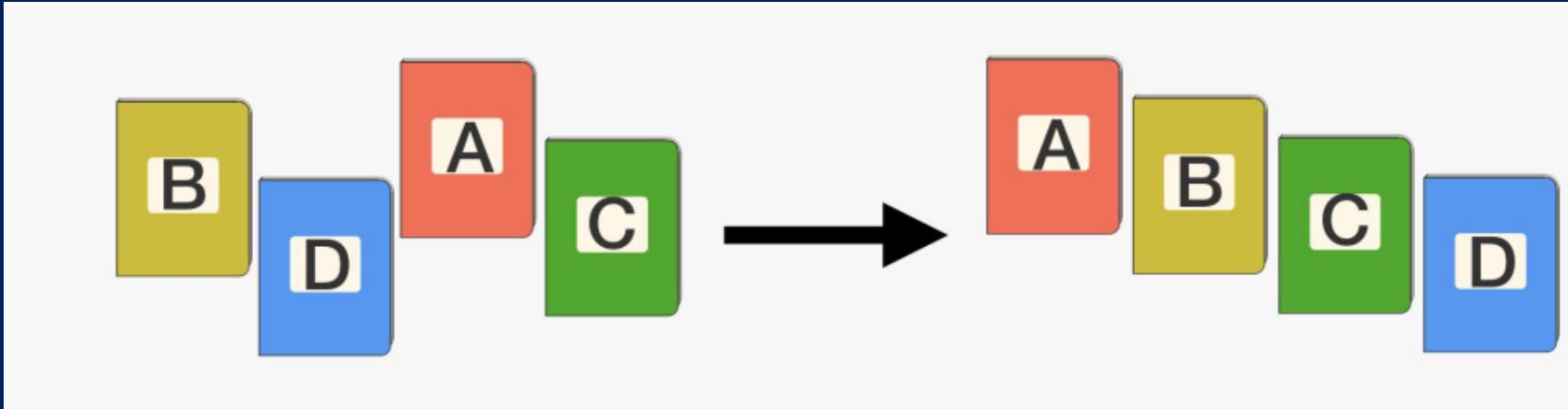
*Any Questions ?*



**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# Chapter 5: Divide and Conquer

## Merge Sort



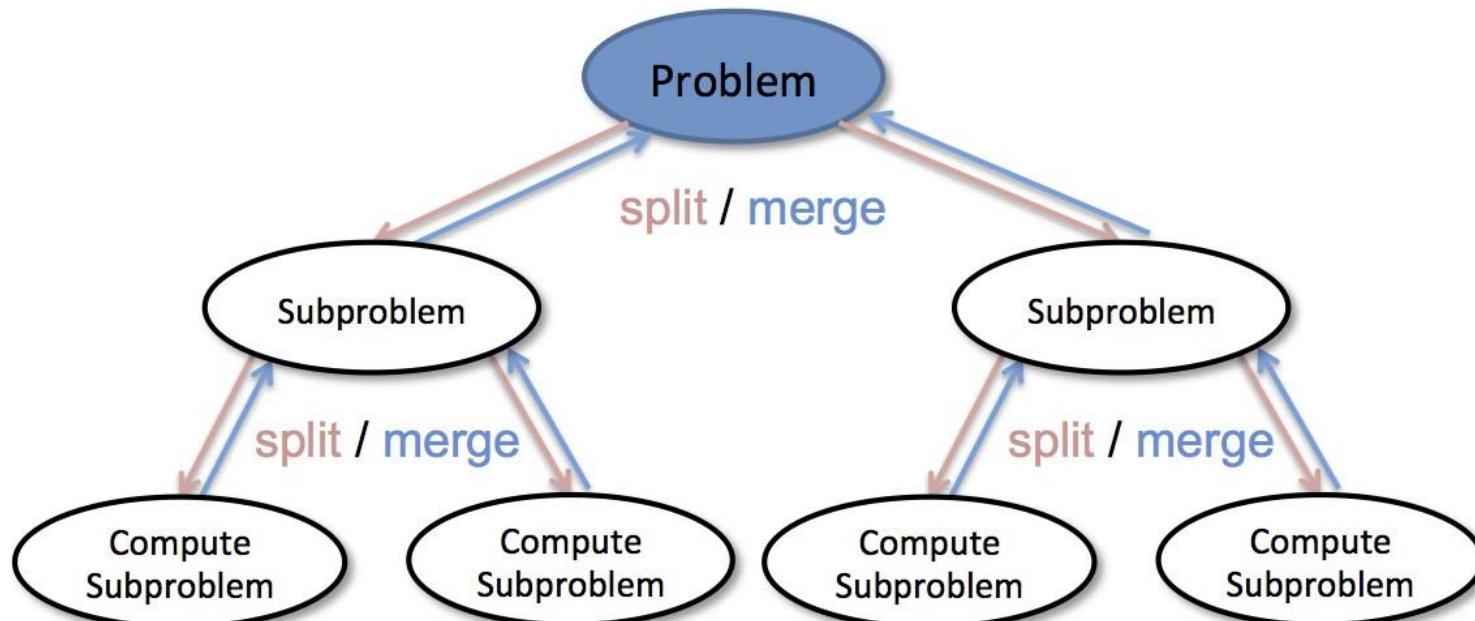
Prof. Anand Singh Jalal

# Divide and Conquer

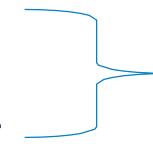
**Divide** the problem into a number of sub-problems that are smaller instances of the same problem.

**Conquer** the sub-problems by solving them recursively.

**Combine** the solutions to the sub-problems into the solution for the original problem.



# Divide and Conquer

- Sorting Algorithms
    - Bubble sort
    - Selection sort
    - Insertion Sort
    - **Quick Sort**
    - Merge Sort
- 
- Divide-Conquer Approach**

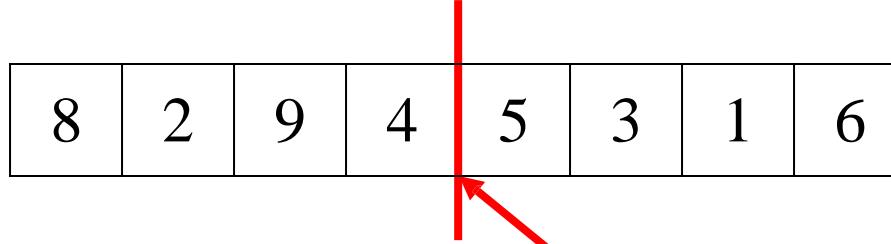
# Divide and Conquer: Merge Sort

- **DIVIDE** : Divide the unsorted list into two sub lists of about half the size.
- **CONQUER** : Sort each of the two sub lists recursively. If they are small enough just solve them in a straight forward manner.
- **COMBINE** : Merge the two-sorted sub lists back into one sorted list.

# Divide and Conquer: Merge Sort ...

- MergeSort is a recursive sorting procedure that uses at most  **$O(n \lg(n))$**  comparisons.
- To sort an array of **n** elements, we perform the following steps in sequence:
  - If  **$n < 2$**  then the array is already sorted.
  - Otherwise,  **$n > 1$** , and we perform the following three steps in sequence:
    1. **Sort** the **left half** of the the array using MergeSort.
    2. **Sort** the **right half** of the the array using MergeSort.
    3. **Merge** the sorted left and right halves.

# Divide and Conquer: Merge Sort ...

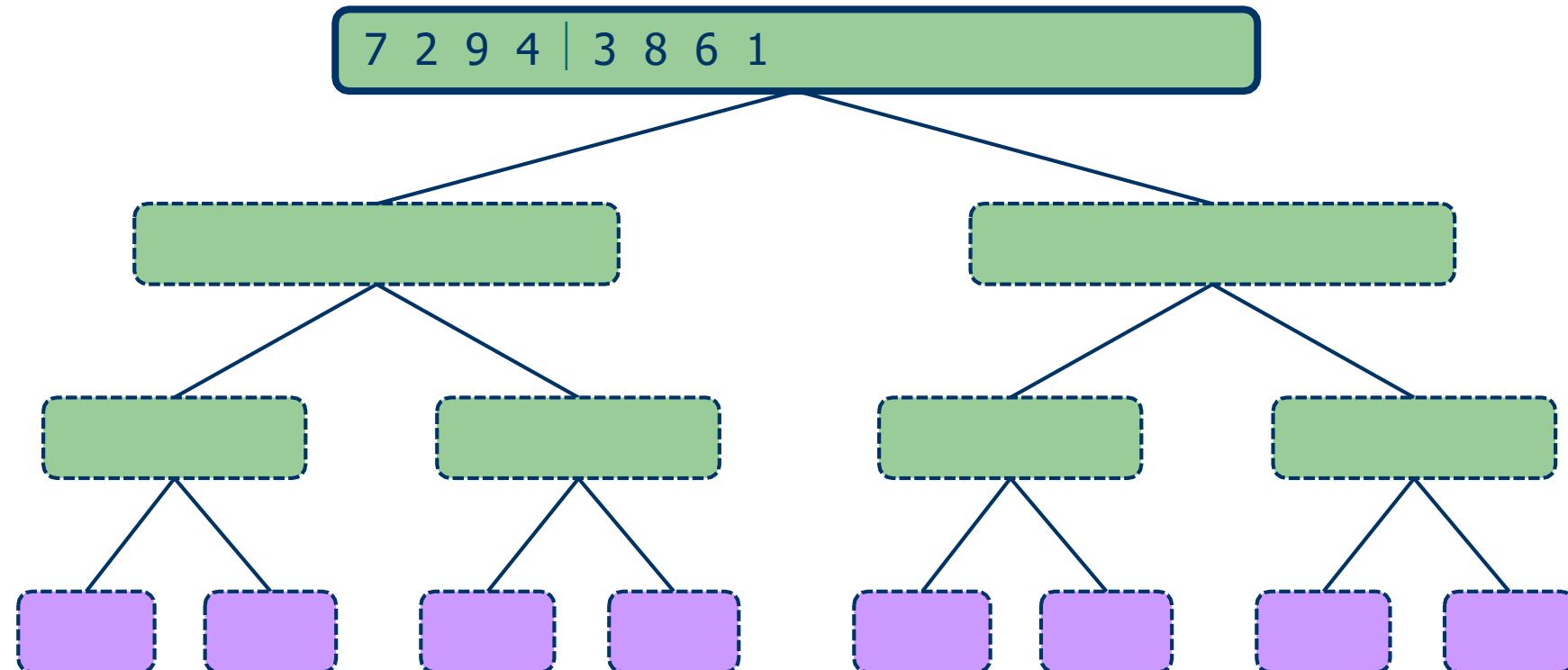


- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)
- Merge two halves together

# Divide and Conquer: Merge Sort ...

## Execution Example

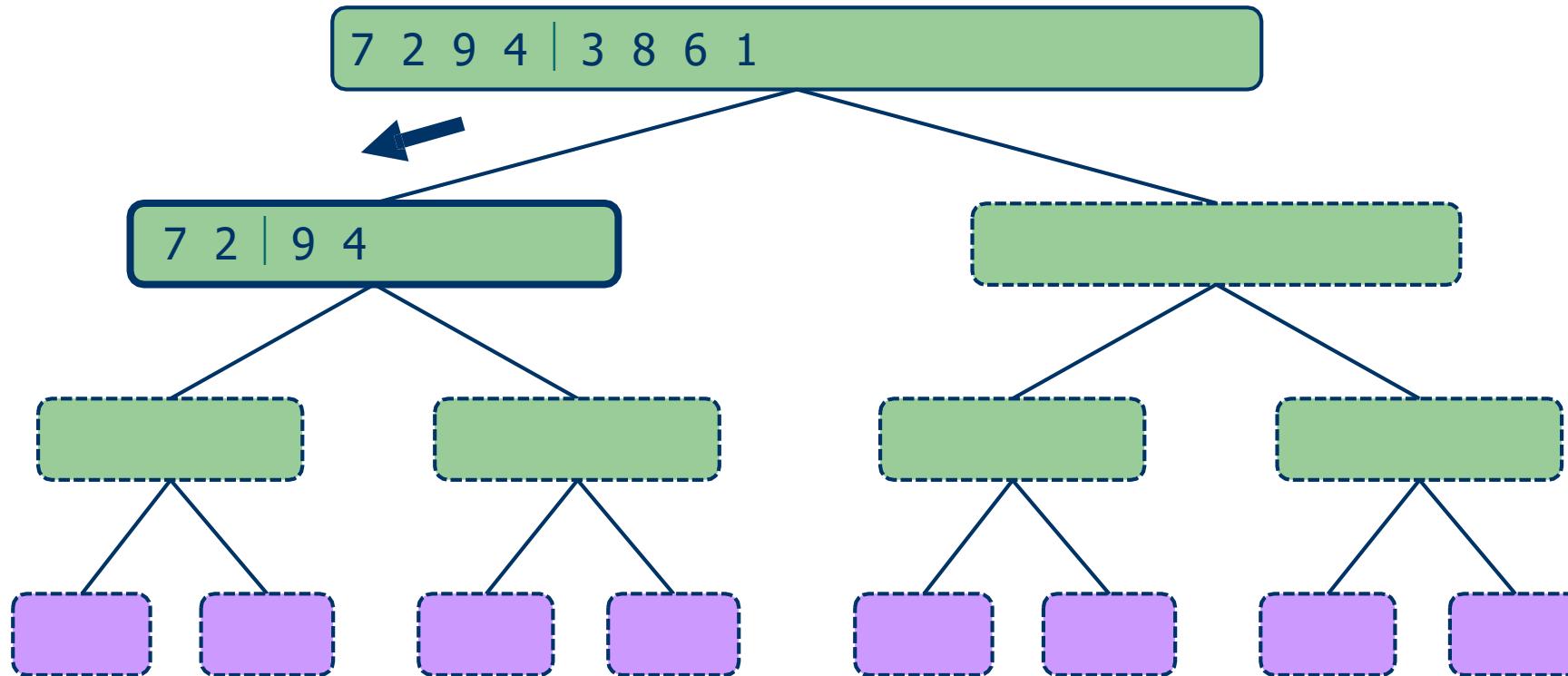
- Partition



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

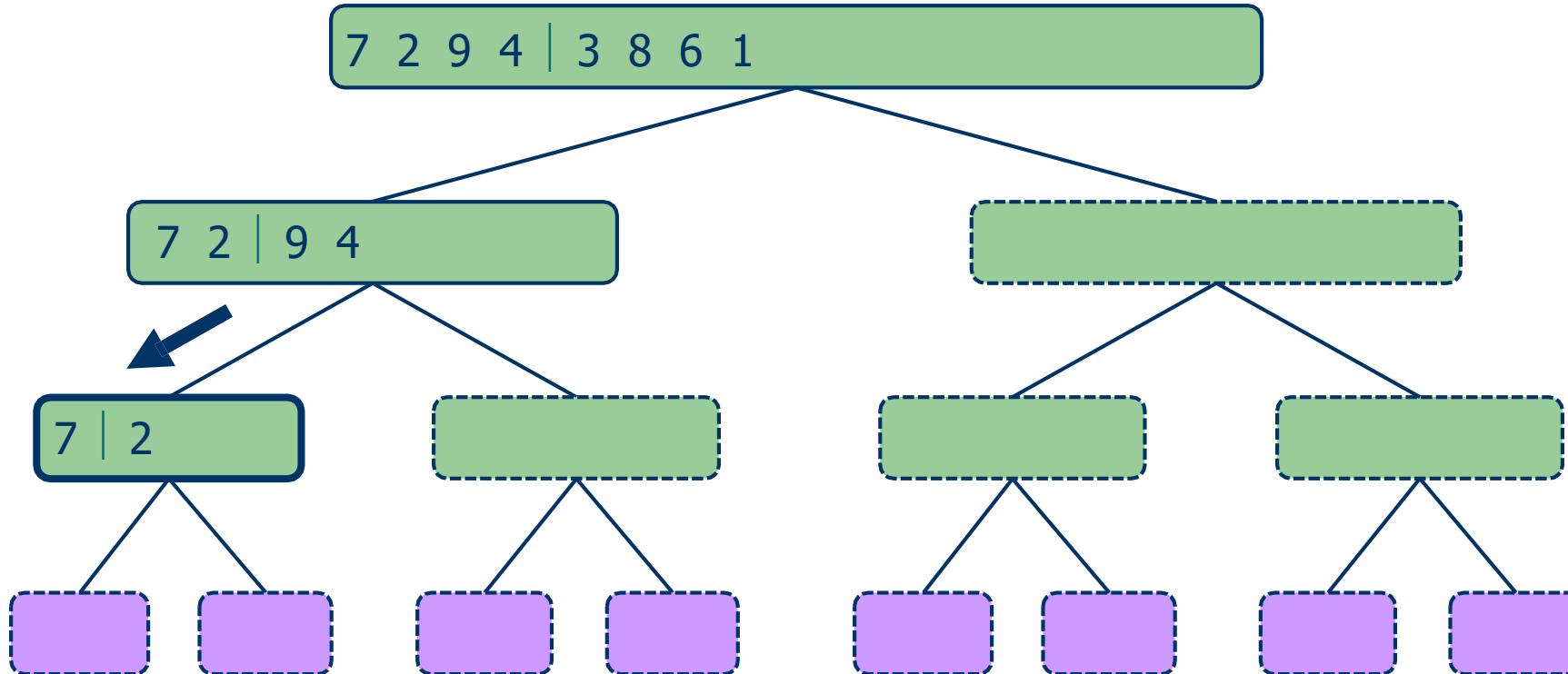
- Recursive call, partition



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

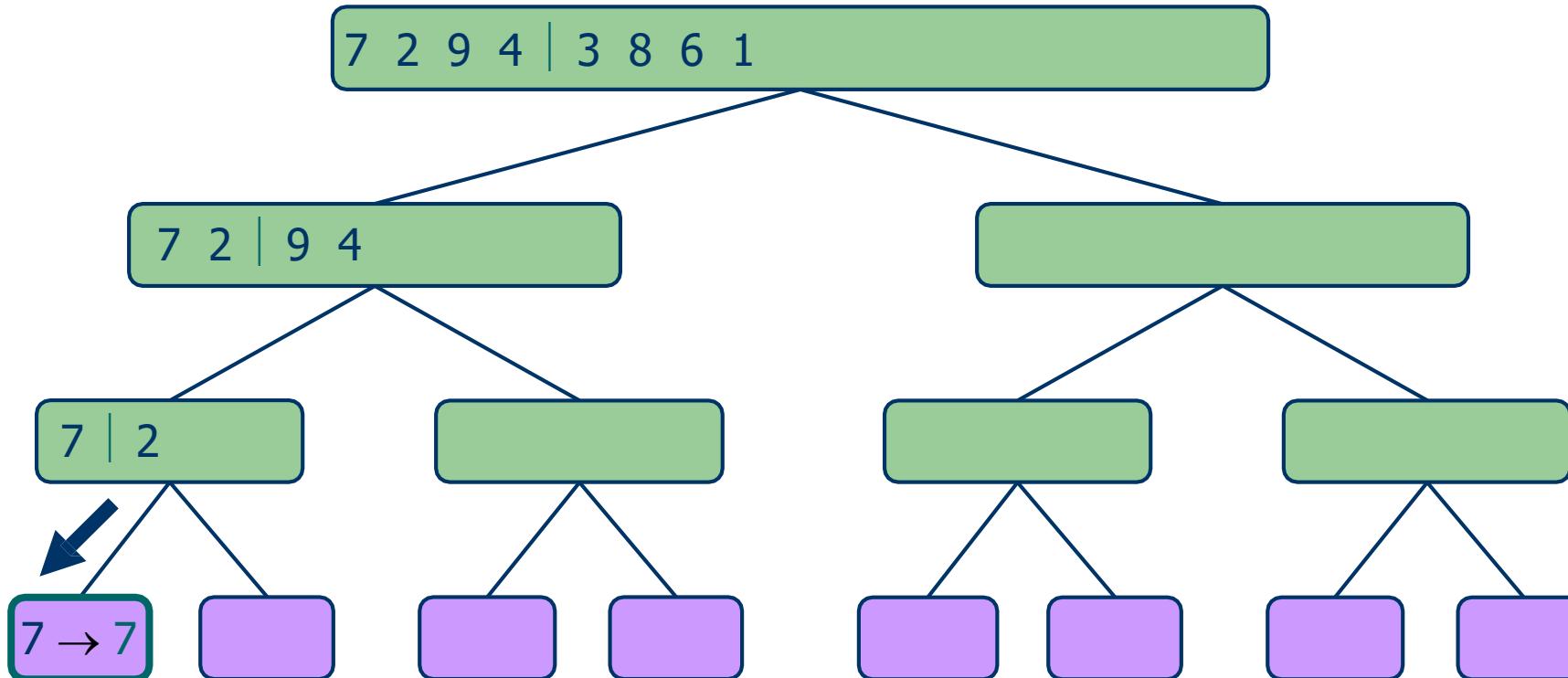
- Recursive call, partition



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

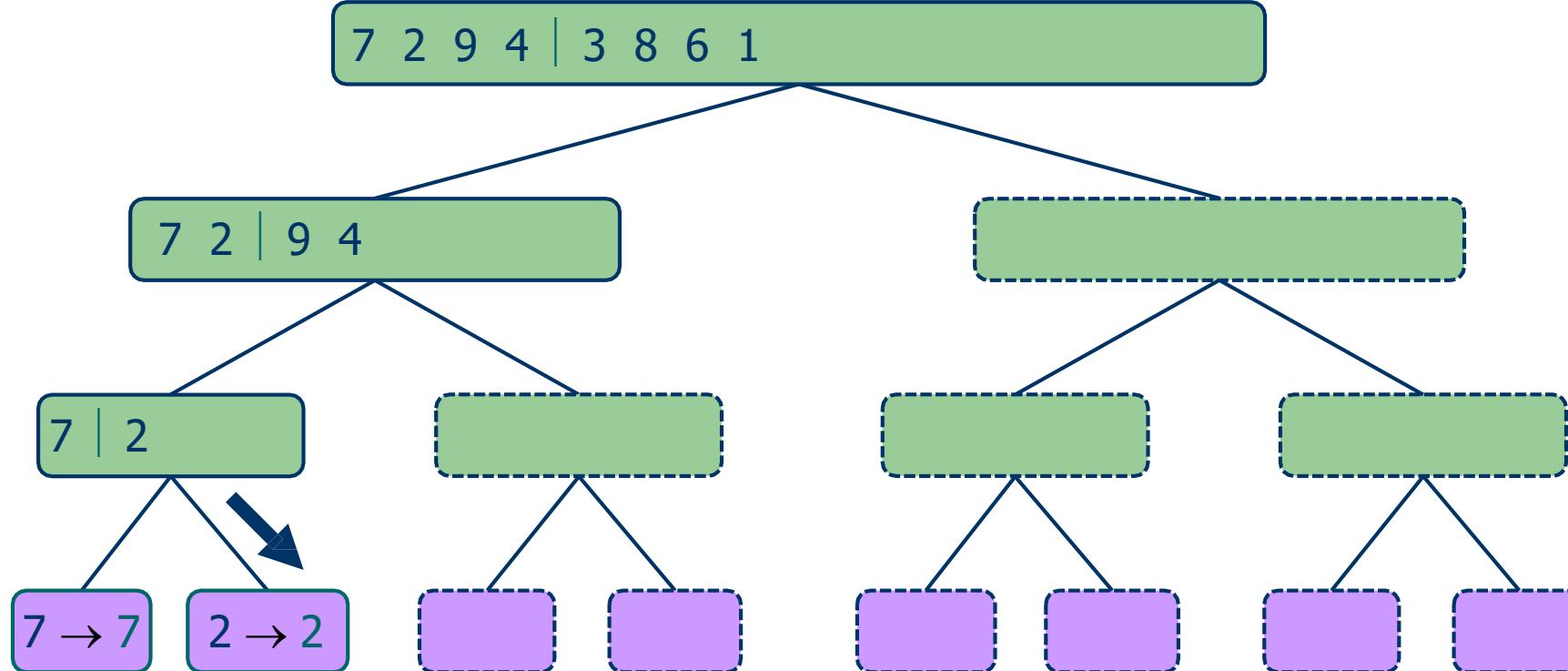
- Recursive call, base case



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

- Recursive call, base case

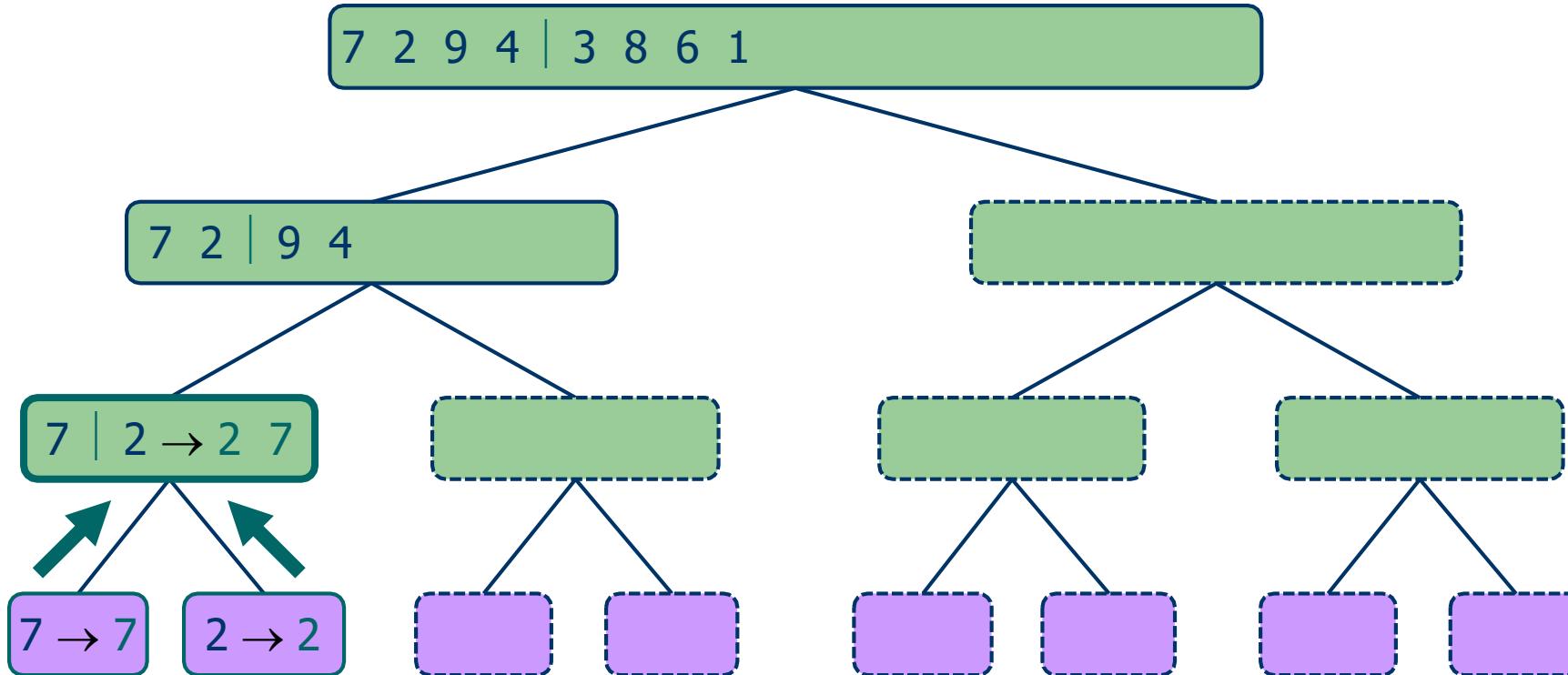




# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

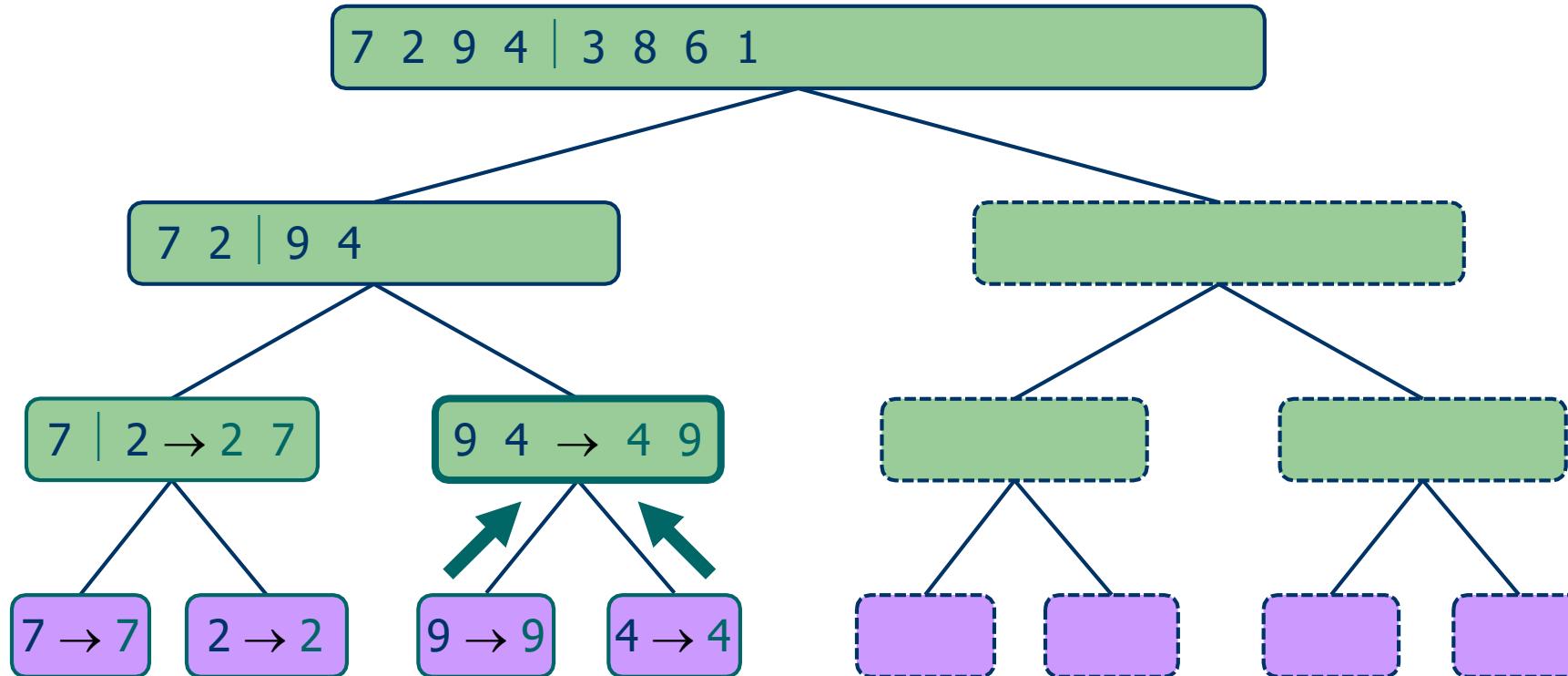
- Merge



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

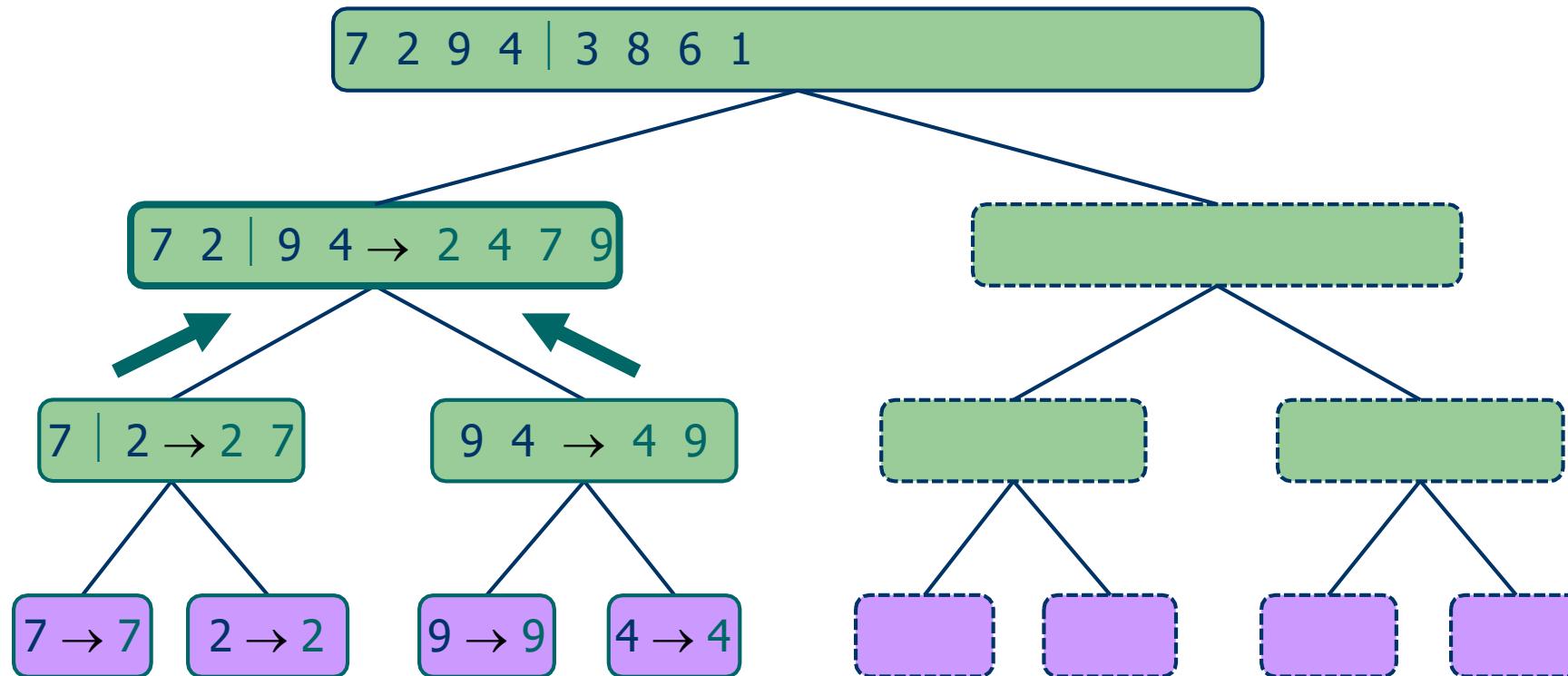
- Recursive call, ..., base case, merge



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

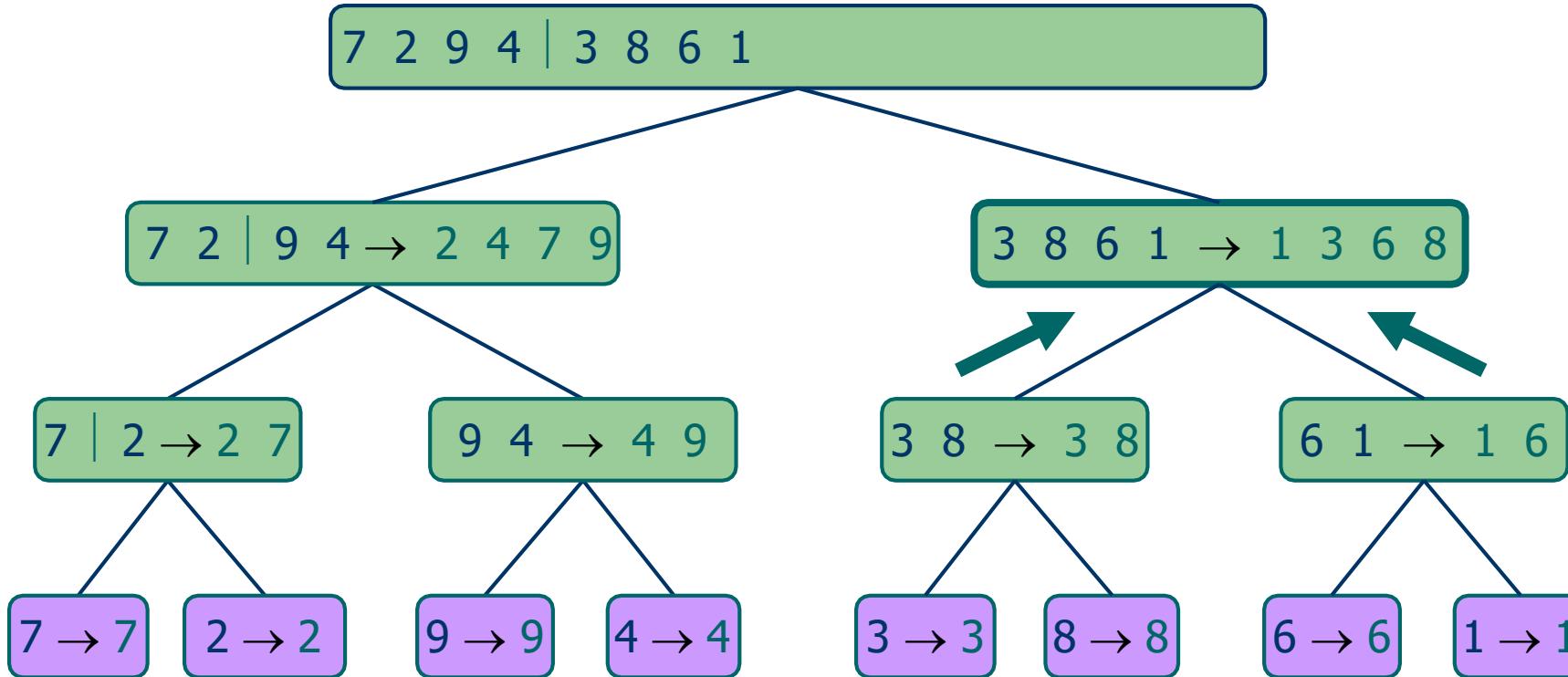
- Merge



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

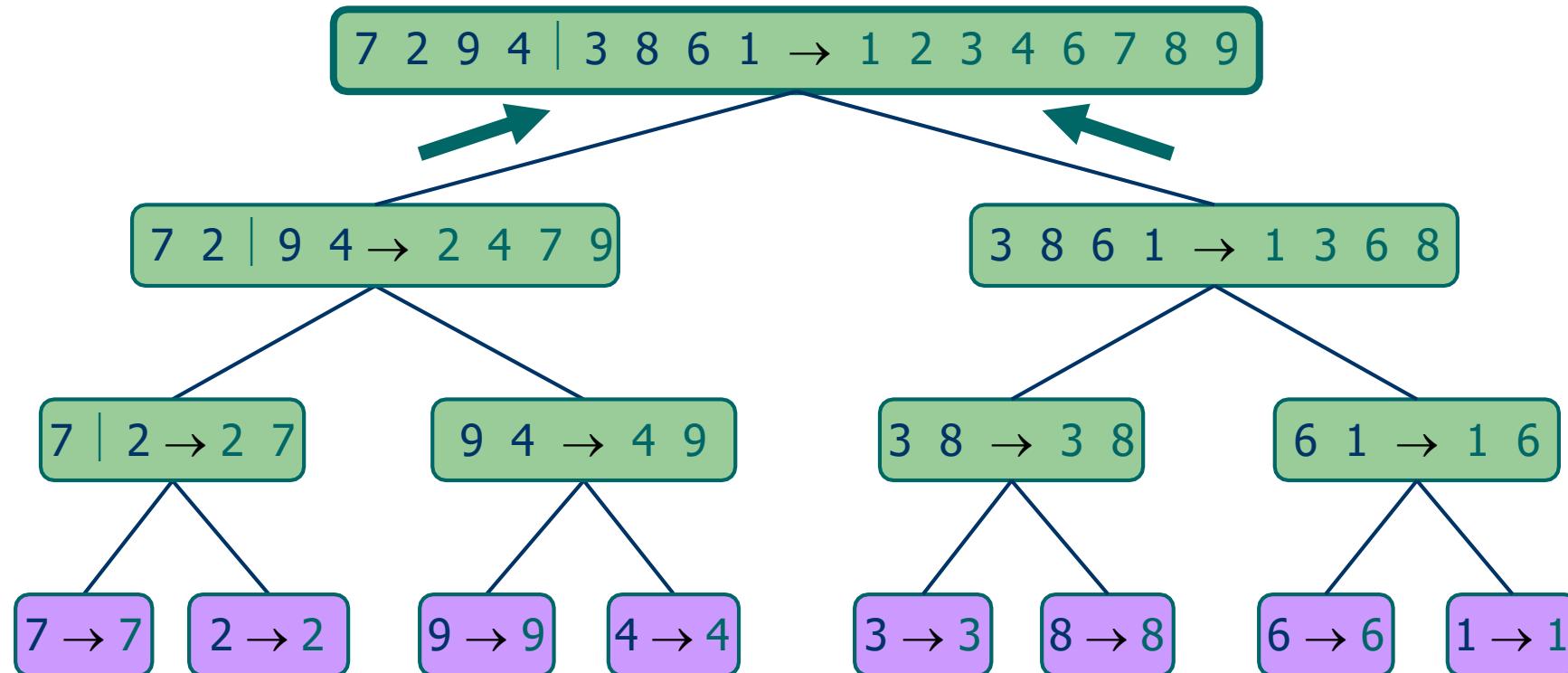
- Recursive call, ..., merge, merge



# Divide and Conquer: Merge Sort ...

## Execution Example (cont.)

- Merge



# Divide and Conquer: Merge Sort ...

**INPUT:** a sequence of  $n$  numbers stored in array A

**OUTPUT:** an ordered sequence of  $n$  numbers

***MergeSort (A, p, r)*** // sort A[p..r] by divide & conquer

```
1  if  $p < r$ 
2    then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3      MergeSort (A, p, q)
4      MergeSort (A, q+1, r)
5      Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

**Initial Call:** *MergeSort(A, 1, n)*

# Divide and Conquer: Merge Sort ...

**Merge( $A, p, q, r$ )**

```

1  $n_1 \leftarrow q - p + 1$ 
2  $n_2 \leftarrow r - q$ 
3   for  $i \leftarrow 1$  to  $n_1$ 
4     do  $L[i] \leftarrow A[p + i - 1]$ 
5   for  $j \leftarrow 1$  to  $n_2$ 
6 do  $R[j] \leftarrow A[q + j]$  7
     $L[n_1+1] \leftarrow \infty$ 
8    $R[n_2+1] \leftarrow \infty$ 
9    $i \leftarrow 1$ 
10   $j \leftarrow 1$ 
11  for  $k \leftarrow p$  to  $r$ 
12    do if  $L[i] \leq R[j]$ 
13      then  $A[k] \leftarrow L[i]$ 
14         $i \leftarrow i + 1$ 
15      else  $A[k] \leftarrow R[j]$ 
16         $j \leftarrow j + 1$ 

```

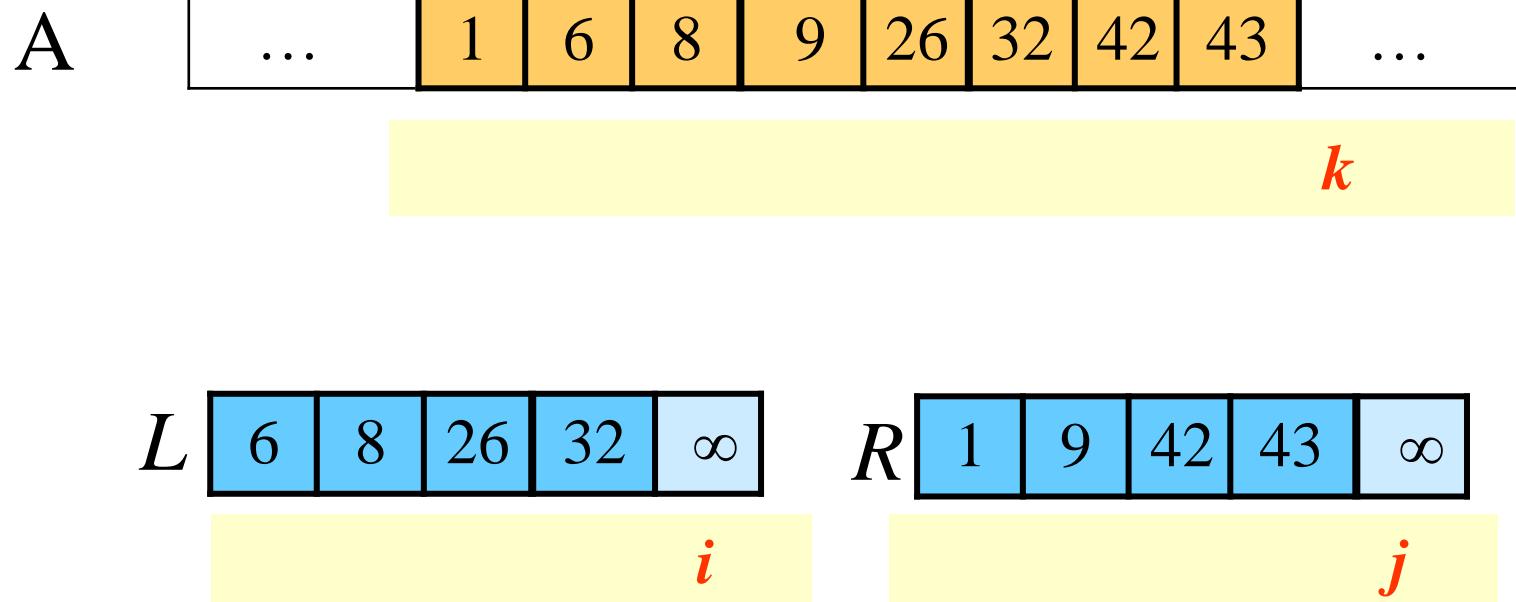
Input: Array containing sorted subarrays  $A[p..q]$  and  $A[q+1..r]$ .

Output: Merged sorted subarray in  $A[p..r]$ .

**Sentinels**, to avoid having to check if either subarray is fully copied at each step.

# Divide and Conquer: Merge Sort ...

## Merge – Example



# Divide and Conquer: Merge Sort ...

## Analysis of Merge Sort

**Running time  $T(n)$  of Merge Sort:**

- ◆ Divide: computing the middle takes  $\Theta(1)$
- ◆ Conquer: solving 2 subproblems takes  $2T(n/2)$
- ◆ Combine: merging  $n$  elements takes  $\Theta(n)$
- ◆ Total:

$$T(n) = \Theta(1) \quad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \quad \text{if } n > 1$$

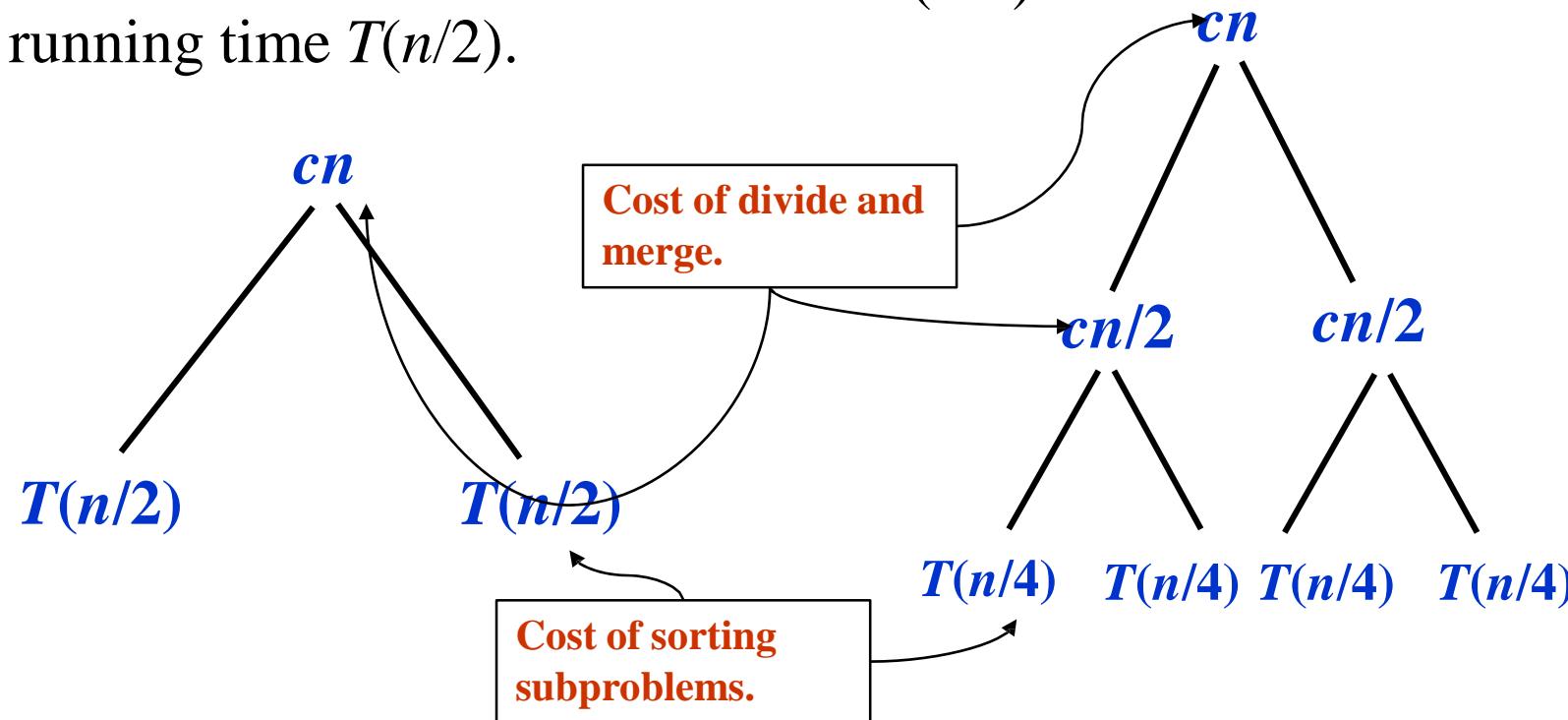
$$\Rightarrow T(n) = \Theta(n \lg n)$$

# Divide and Conquer: Merge Sort ...

## Recursion Tree for Merge Sort

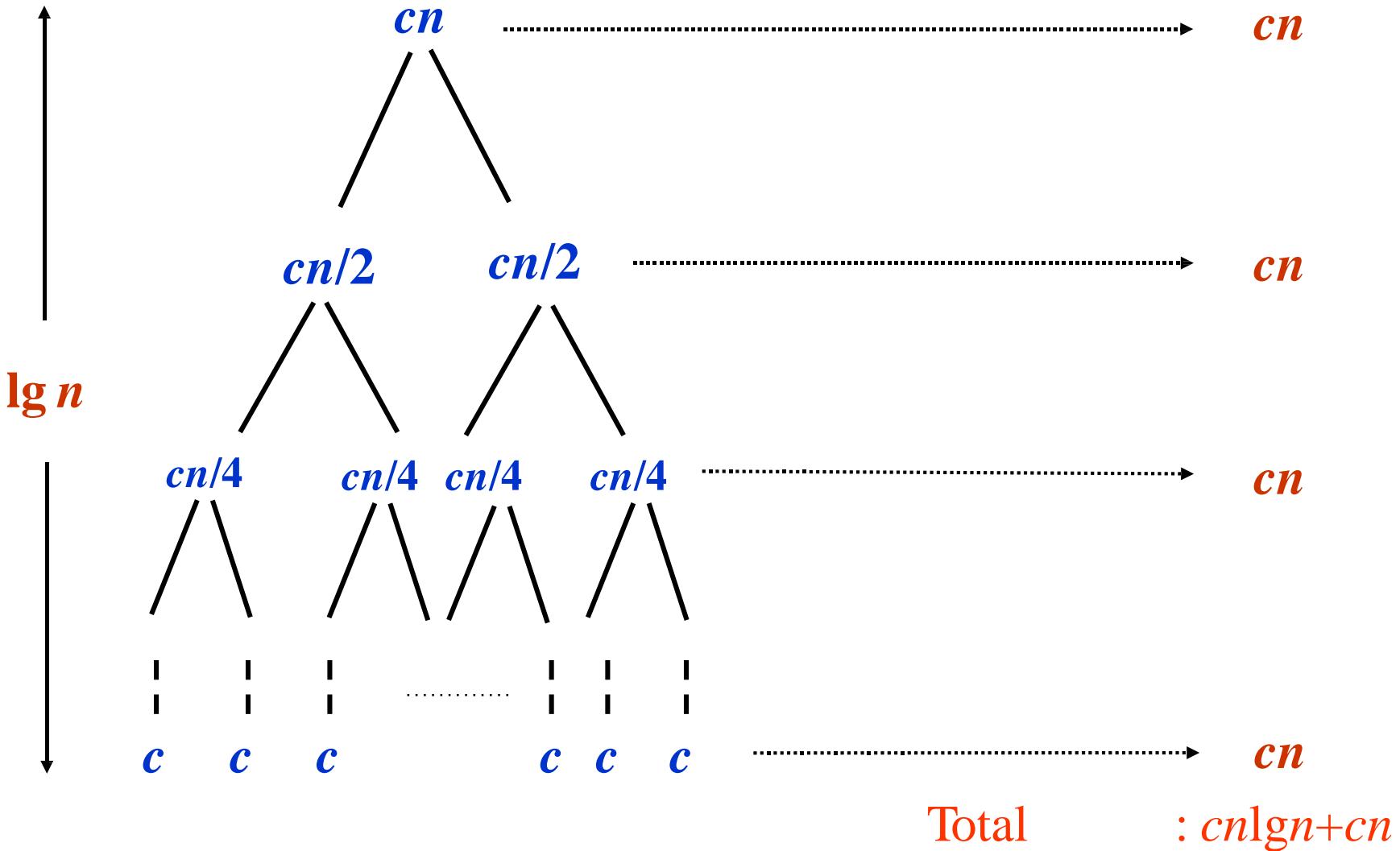
For the original problem, we have a cost of  $cn$ , plus two subproblems each of size  $(n/2)$  and running time  $T(n/2)$ .

Each of the size  $n/2$  problems has a cost of  $cn/2$  plus two subproblems, each costing  $T(n/4)$ .



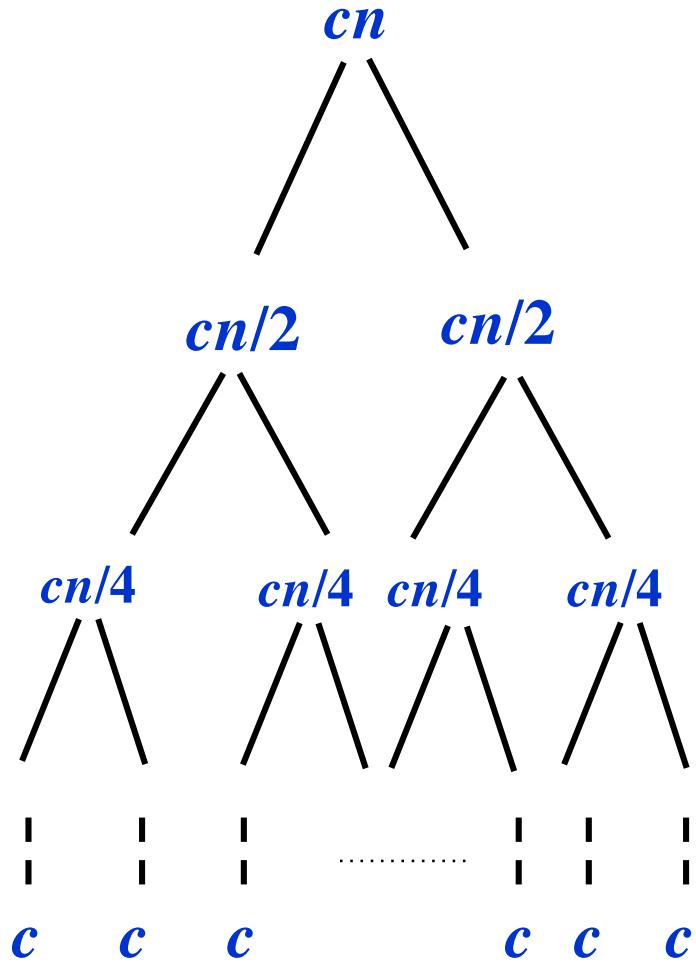
# Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



# Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



- Each level has total cost  $cn$ .
- Each time we go down one level, the number of subproblems doubles, but the cost per subproblem halves  
 $\Rightarrow$  *cost per level remains the same*.
- There are  $\lg n + 1$  levels, height is  $\lg n$ . (Assuming  $n$  is a power of 2.)
  - Can be proved by induction.
- Total cost = sum of costs at each level =  $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$ .

“Thank you”

*Any Questions ?*



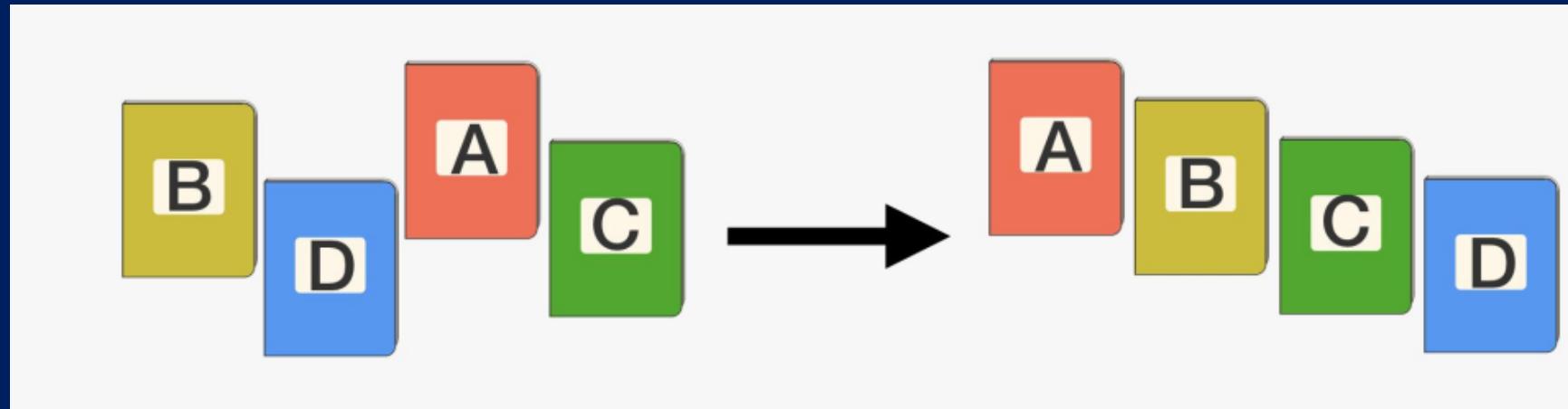
**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 5: Divide and Conquer

## Quick Sort



Prof. Anand Singh Jalal

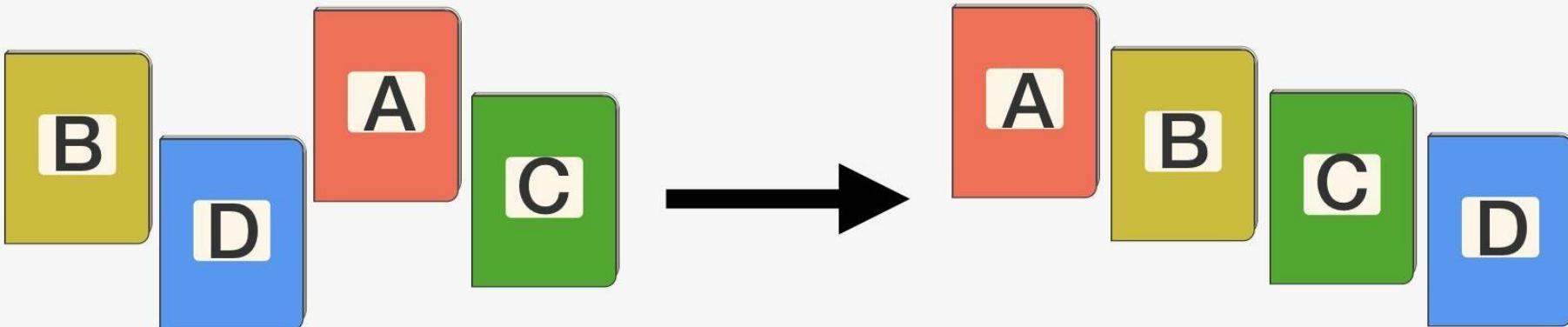
# The Sorting Problem

- **Input:**

- A sequence of  $n$  numbers  $a_1, a_2, \dots, a_n$

- **Output:**

- A permutation (reordering)  $a'_1, a'_2, \dots, a'_n$  of the input sequence such that  $a'_1 \leq a'_2 \leq \dots \leq a'_n$



# The Sorting Problem

- **Sorting takes an unordered collection and makes it an ordered one.**

1      2      3      4      5      6

77	42	35	12	101	5
----	----	----	----	-----	---



1      2      3      4      5      6

5	12	35	42	77	101
---	----	----	----	----	-----

# The Sorting Problem: Some Definitions

- **Internal Sort**

- The data to be sorted is all stored in the computer's main memory.

- **External Sort**

- Some of the data to be sorted might be stored in some external, slower, device.

- **In Place Sort**

- The amount of extra space required to sort the data is constant with the input size.

# The Sorting Problem: Some Definitions

## Stable sort

A **STABLE** sort preserves relative order of records with equal keys

Sorted on first key:

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Sort file on second key:

Records with key value 3 are not in order on first key!!

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Furia	3	A	766-093-9873	22 Brown
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Gazsi	4	B	665-303-0266	113 Walker
Aaron	4	A	664-480-0023	097 Little

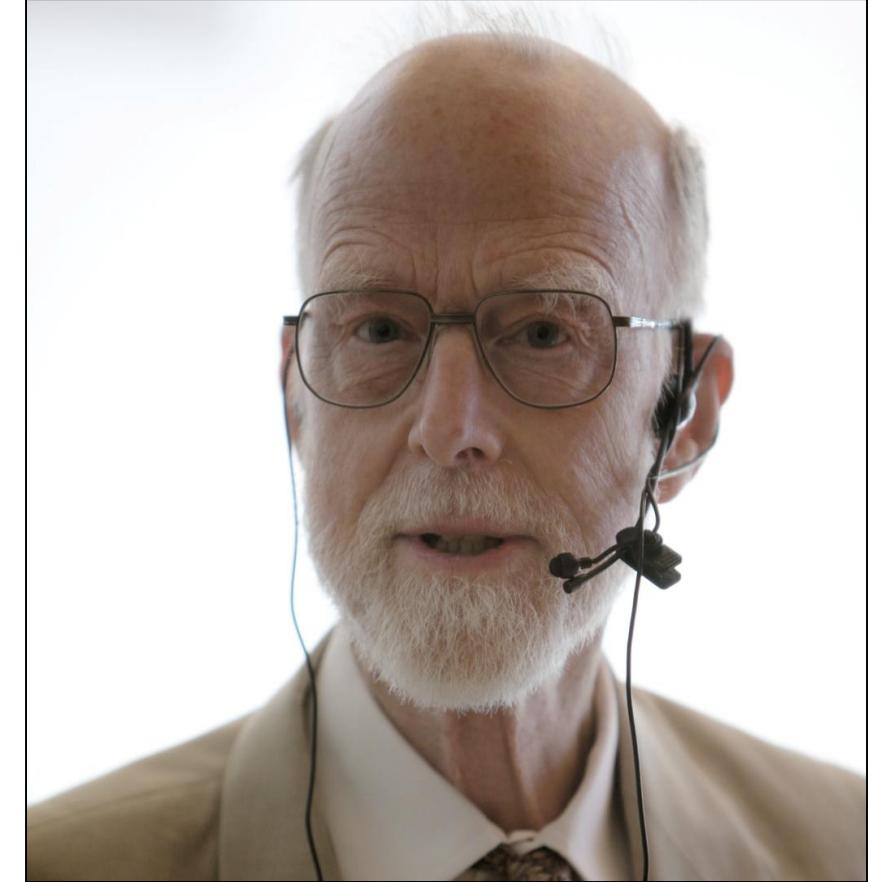
# Divide and Conquer

- Sorting Algorithms
    - Bubble sort
    - Selection sort
    - Insertion Sort
    - **Quick Sort**
    - **Merge Sort**
- Divide-Conquer Approach**

# Divide and Conquer: Quick Sort

**Quicksort** is an in-place sorting algorithm. Developed by British computer scientist Tony Hoare in 1959

January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society, **selected quicksort as one of the 10 algorithms** “with the greatest influence on the development and practice of science and engineering in the 20th century.”



**Tony Hoare**  
**(DEVELOPER OF QUICK SORT)**

# Divide and Conquer: Quick Sort

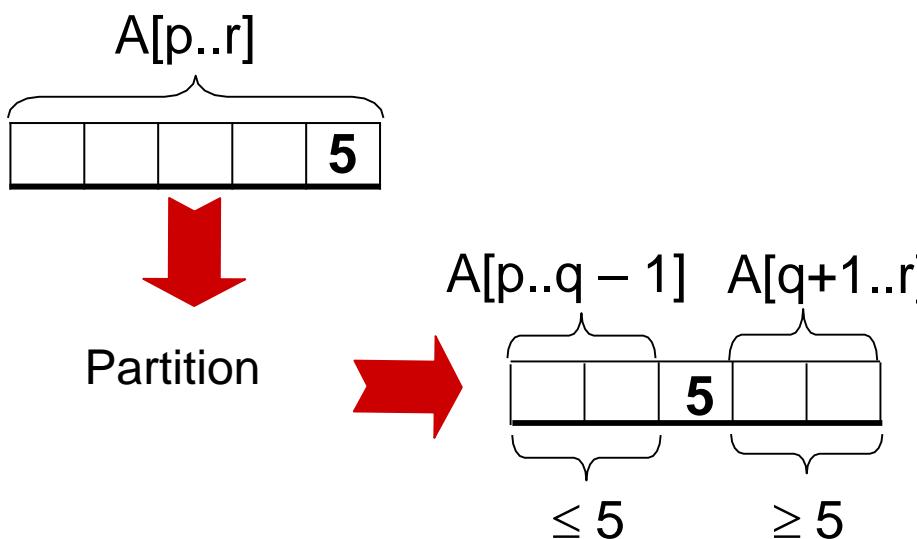
- Follows the **divide-and-conquer** paradigm.
- **Divide:** Partition (separate) the array  $A[p..r]$  into two subarrays  $A[p..q-1]$  and  $A[q+1..r]$ .
  - Each element in  $A[p..q-1] < A[q]$ .
  - $A[q] <$  each element in  $A[q+1..r]$ .
  - Index  $q$  is computed as part of the partitioning procedure.
- **Conquer:** Sort the two subarrays by recursive calls to quicksort.
- **Combine:** The subarrays are sorted in place – no work is needed to combine them.
- How do the divide and combine steps of quicksort compare with those of merge sort?

# Divide and Conquer: Quick Sort ...

Quicksort(A, p, r)

```

if p < r then
    q := Partition(A, p, r);
    Quicksort(A, p, q - 1);
    Quicksort(A, q + 1, r)
  
```



Partition(A, p, r)

```

x:= A[r], i:=p - 1;
for j := p to r - 1 do
    if A[j]  $\leq$  x then
        { i := i + 1;
          A[i]  $\leftrightarrow$  A[j]; }
    A[i + 1]  $\leftrightarrow$  A[r];
return i + 1
  
```

# Divide and Conquer: Quick Sort ...

**initially:**

p	r
2 5 8 3 9 4 1 7 10 6	
i j	

**note:** pivot ( $x$ ) = 6

**next iteration:**

2 5 8 3 9 4 1 7 10 6	
i j	

**next iteration:**

2 5 8 3 9 4 1 7 10 6	
i j	

**next iteration:**

2 5 8 3 9 4 1 7 10 6	
i j	

**next iteration:**

2 5 3 8 9 4 1 7 10 6	
i j	

**Partition( $A, p, r$ )**

```

x := A[r], i := p - 1;
for j := p to r - 1 do
    if A[j] ≤ x then
        { i := i + 1;
          A[i] ↔ A[j]; }
    A[i + 1] ↔ A[r];
return i + 1
  
```

# Divide and Conquer: Quick Sort ...

next iteration:      2 5 3 8 9 4 1 7 10 6

i            j

next iteration:      2 5 3 8 9 4 1 7 10 6

i            j

next iteration:      2 5 3 4 9 8 1 7 10 6

i            j

next iteration:      2 5 3 4 1 8 9 7 10 6

i            j

next iteration:      2 5 3 4 1 8 9 7 10 6

i            j

next iteration:      2 5 3 4 1 8 9 7 10 6

i            j

after final swap:      2 5 3 4 1 6 9 7 10 8

i            j

Partition(A, p, r)

```

x:= A[r], i:=p - 1;
for j := p to r - 1 do
    if A[j] ≤ x then
        { i := i + 1;
          A[i] ↔ A[j];}
    A[i + 1] ↔ A[r];
return i + 1
  
```

# Divide and Conquer: Quick Sort ...

**CASE:**  
**First element as a Pivot**

**ALGORITHM** *HoarePartition( $A[l..r]$ )*

//Partitions a subarray by Hoare's algorithm, using the first element  
// as a pivot

//Input: Subarray of array  $A[0..n - 1]$ , defined by its left and right  
// indices  $l$  and  $r$  ( $l < r$ )

//Output: Partition of  $A[l..r]$ , with the split position returned as  
// this function's value

$p \leftarrow A[l]$   
 $i \leftarrow l; j \leftarrow r + 1$

**repeat**

**repeat**  $i \leftarrow i + 1$  **until**  $A[i] \geq p$

**repeat**  $j \leftarrow j - 1$  **until**  $A[j] \leq p$

swap( $A[i]$ ,  $A[j]$ )

**until**  $i \geq j$

swap( $A[i]$ ,  $A[j]$ ) //undo last swap when  $i \geq j$

swap( $A[l]$ ,  $A[j]$ )

**return**  $j$

# Divide and Conquer: Quick Sort ...

0	1	2	3	4	5	6	7
5	3	1	9	8	2	4	7
5	3	1	9	8	2	4	7
5	3	1	4	8	2	9	7
5	3	1	4	8	2	9	7
5	3	1	4	2	8	9	7
5	3	1	4	2	8	9	7
2	3	1	4	5	8	9	7
2	3	1	4				
2	3	1	4				
2	1	3	4				
2	1	3	4				
1	2	3	4				
1							

## Example



3	4	
j	4	
3	4	
4		
8	9	7
8	7	9
8	7	9
8	7	9
7	8	9
7		9

# Divide and Conquer: Quick Sort ...

## Analysis of quicksort—best case

- Algorithm always chooses best pivot and splits sub-arrays in half at each recursion
  - $T(0) = T(1) = O(1)$ 
    - constant time if 0 or 1 element
  - For  $N > 1$ , 2 recursive calls plus linear time for partitioning
  - $T(N) = 2T(N/2) + O(N)$ 
    - Same recurrence relation as Mergesort
  - $T(N) = \underline{O(N \log N)}$

# Divide and Conquer: Quick Sort ...

## Analysis of quicksort - Worst Case

- Algorithm always chooses the worst pivot – one sub-array is empty at each recursion
  - $T(N) \leq a$  for  $N \leq C$
  - $T(N) \leq T(N-1) + bN$
  - $\leq T(N-2) + b(N-1) + bN$
  - $\leq T(C) + b(C+1) + \dots + bN$
  - $\leq a + b(C + (C+1) + (C+2) + \dots + N)$
  - $T(N) = O(N^2)$
- Fortunately, *average case performance* is  $O(N \log N)$

“Thank you”

*Any Questions ?*



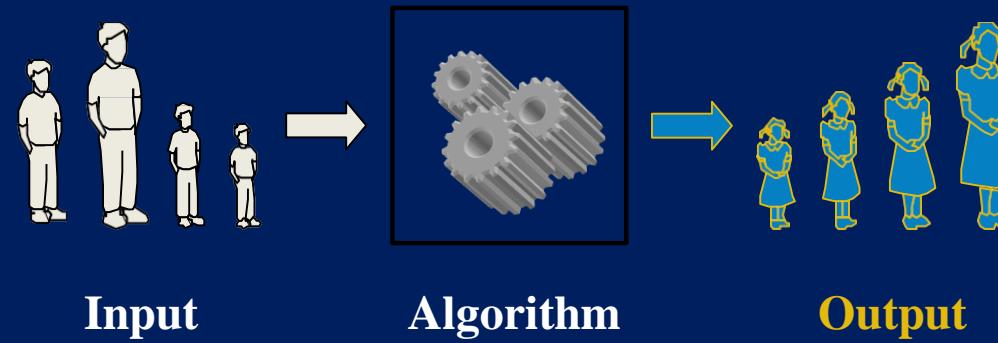
**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 5: Divide and Conquer

## Heap Sort

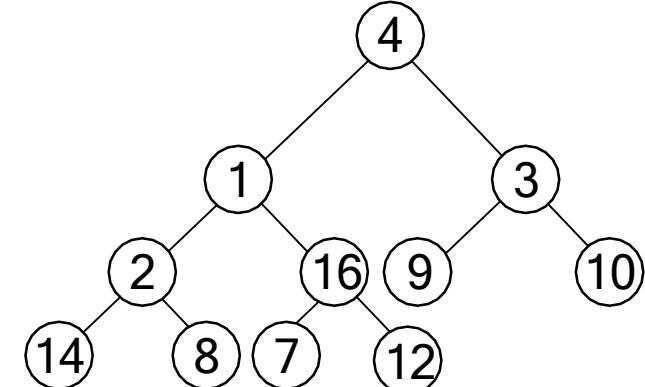


Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

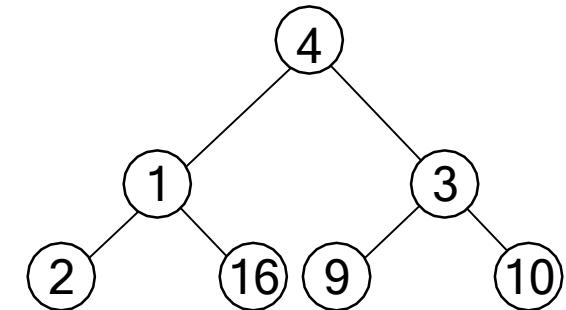
# Special Types of Trees

- **Def:** Full binary tree = a binary tree in which each node is either a leaf or has degree exactly 2.



Full binary tree

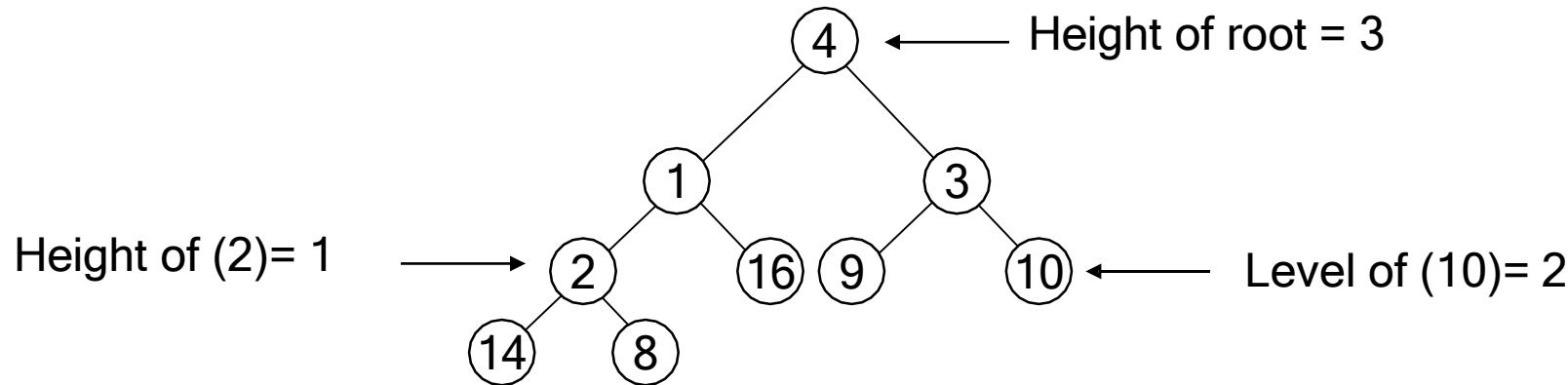
- **Def:** Complete binary tree = a binary tree in which all leaves are on the same level and all internal nodes have degree 2.



Complete binary tree

# Special Types of Trees

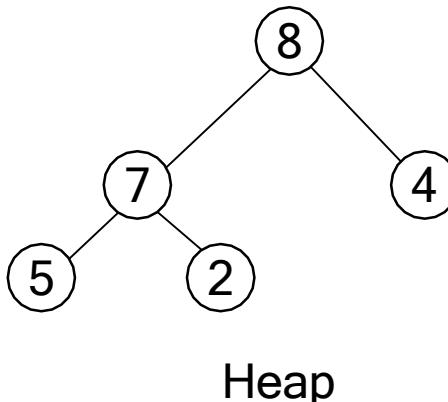
- **Height** of a node = the number of edges on the longest simple path from the node down to a leaf
- **Level** of a node = the length of a path from the root to the node
- **Height** of tree = height of root node



# The Heap Data Structure

- *Def:* A **heap** is a nearly complete binary tree with the following two properties:
  - **Structural property (shape property)** : A Heap tree is essentially complete (or simply complete), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.
  - **Order (heap) property:** for any node  $x$ ,

$$\text{Parent}(x) \geq x \quad (\text{MaxHeap})$$



From the heap property, it follows that:  
**“The root is the maximum element of the heap!”**

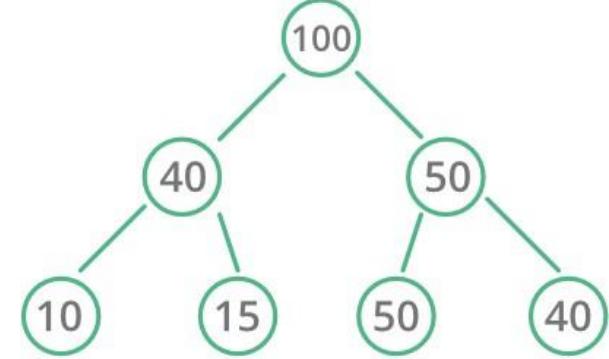
A heap is a binary tree that is filled in order

# Heap Types

- **Max-heaps** (largest element at root), have the *max-heap property*:

- for all nodes  $i$ , excluding the root:

$$A[\text{PARENT}(i)] \geq A[i]$$

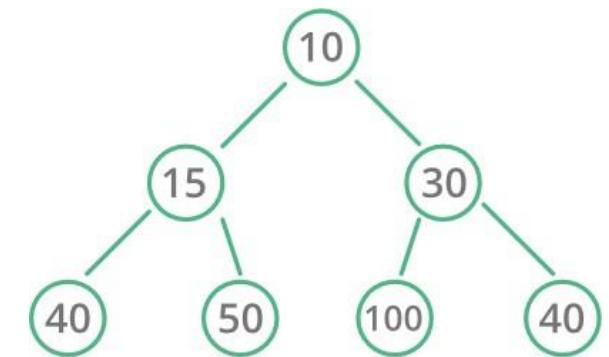


Max Heap

- **Min-heaps** (smallest element at root), have the *min-heap property*:

- for all nodes  $i$ , excluding the root:

$$A[\text{PARENT}(i)] \leq A[i]$$



Min Heap



# Heap Tree

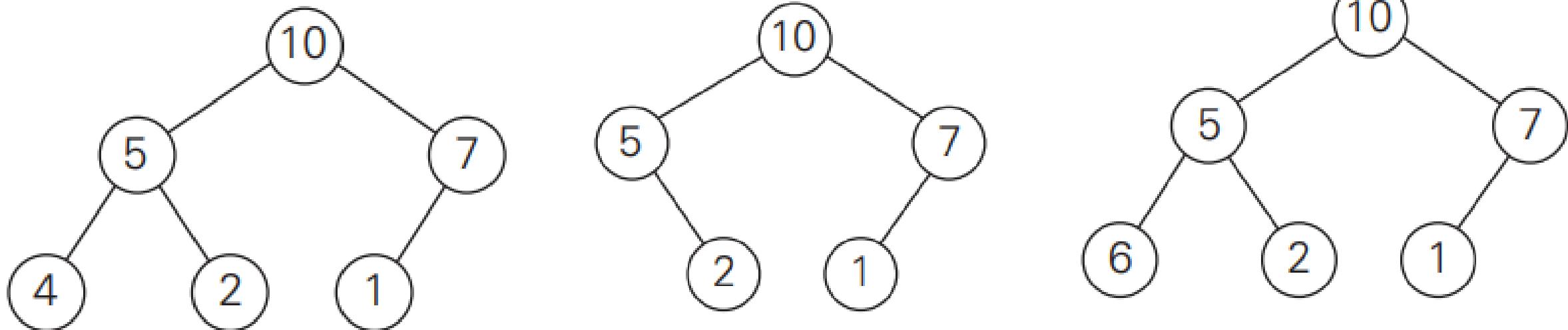
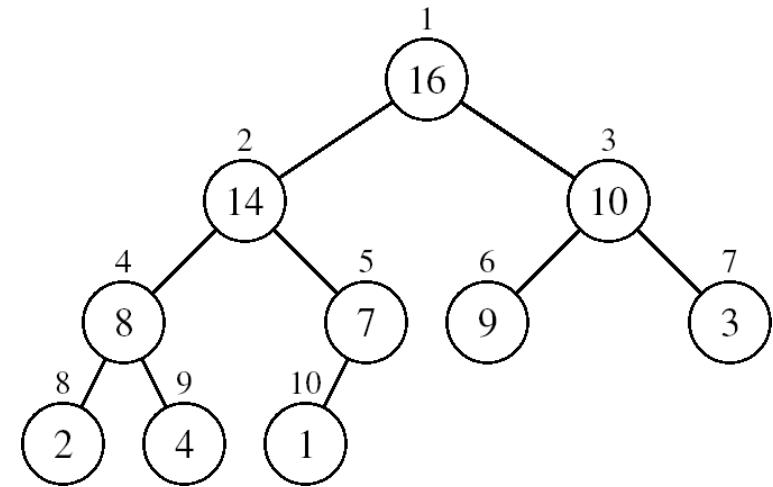
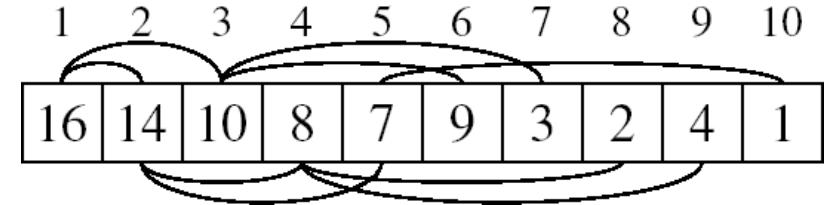


Illustration of the definition of heap

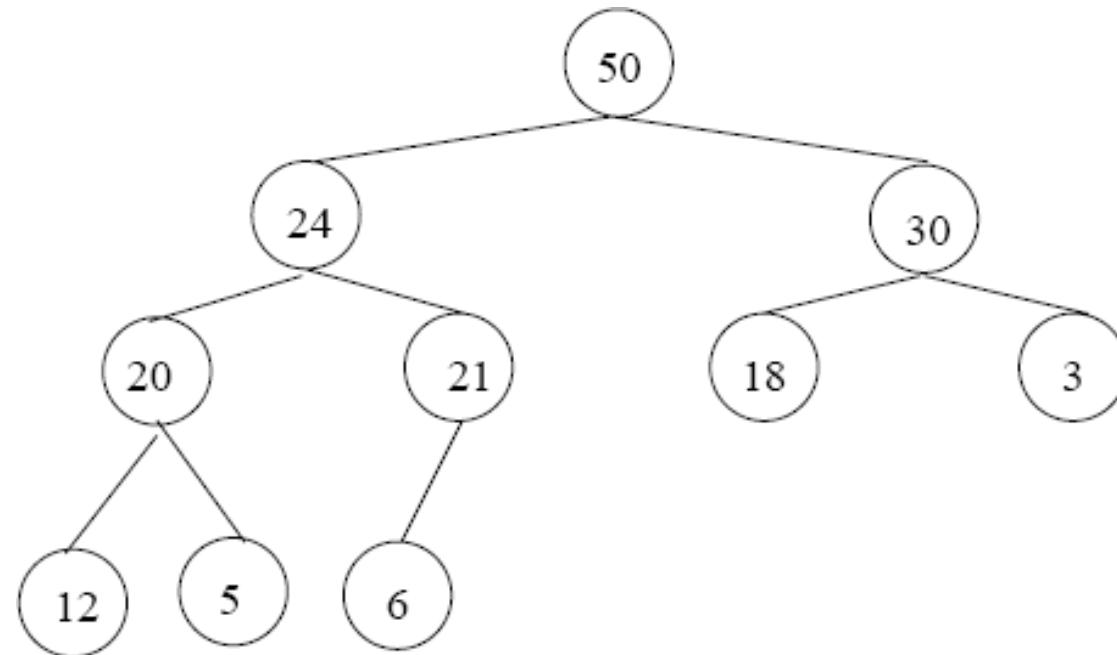
# Array Representation of Heaps

- A heap can be stored as an array  $A$ .
  - Root of tree is  $A[1]$
  - Left child of  $A[i] = A[2i]$
  - Right child of  $A[i] = A[2i + 1]$
  - Parent of  $A[i] = A[\lfloor i/2 \rfloor]$
  - Heapsize[A]  $\leq$  length[A]
- The elements in the subarray  $A[\lfloor n/2 \rfloor + 1] .. n]$  are leaves



# Adding/Deleting Nodes

- New nodes are always inserted at the bottom level (left to right)
- Nodes are removed from the bottom level (right to left)

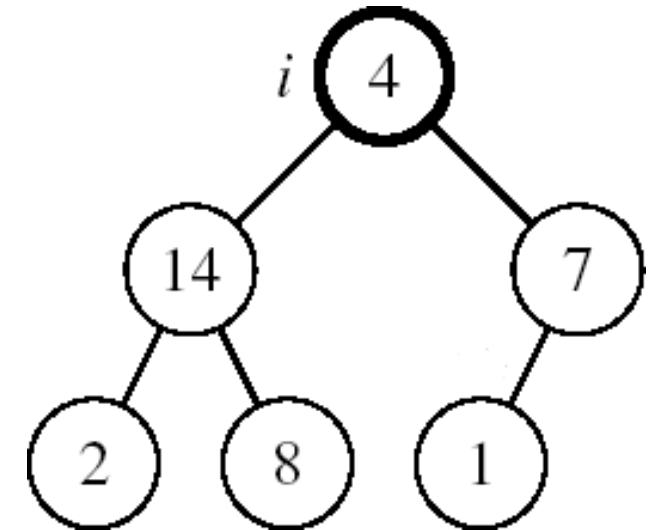


# Operations on Heaps

- Maintain/Restore the max-heap property
  - MAX-HEAPIFY
- Create a max-heap from an unordered array
  - BUILD-MAX-HEAP
- Sort an array in place
  - HEAPSORT
- Priority queues

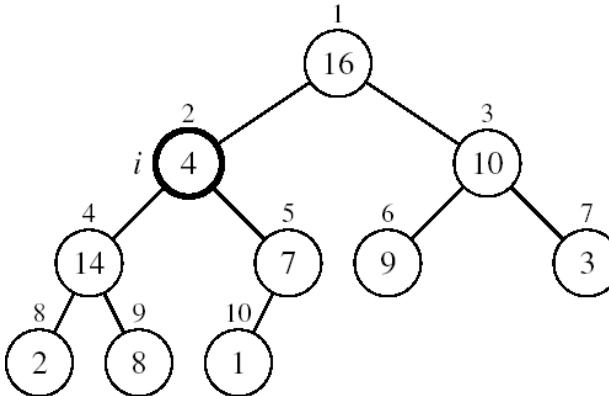
# Maintaining the Heap Property

- **Suppose a node is smaller than a child**
  - Left and Right subtrees of  $i$  are max-heaps
- **To eliminate the violation:**
  - Exchange with larger child
  - Move down the tree
  - Continue until node is not smaller than children

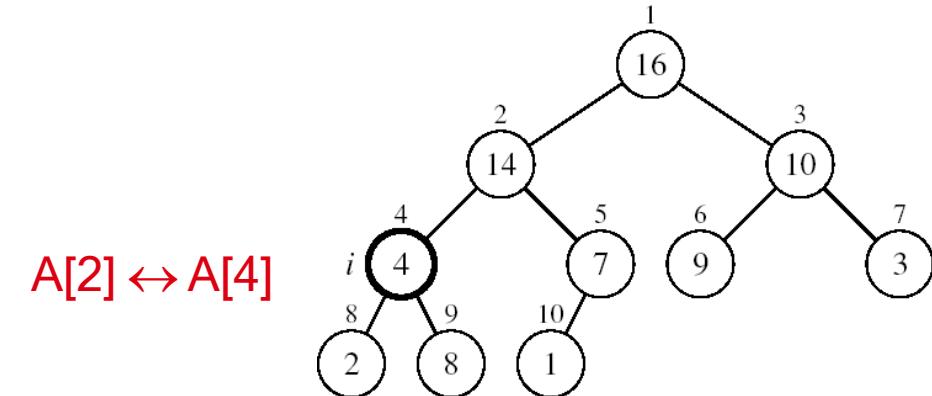


# Example

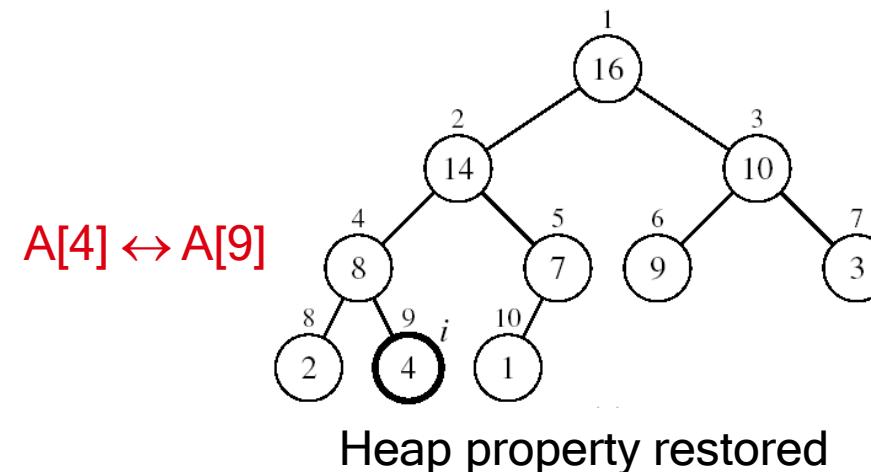
## MAX-HEAPIFY(A, 2, 10)



A[2] violates the heap property



A[4] violates the heap property

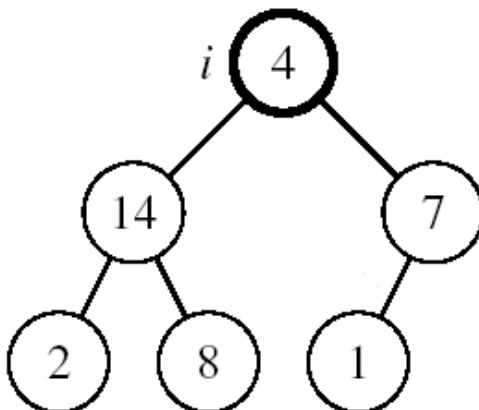


Heap property restored

A[4]  $\leftrightarrow$  A[9]

# Maintaining the Heap Property

- Assumptions:
  - Left and Right subtrees of  $i$  are max-heaps
  - $A[i]$  may be smaller than its children



*Alg:* MAX-HEAPIFY( $A, i, n$ )

1.  $l \leftarrow \text{LEFT}(i)$
2.  $r \leftarrow \text{RIGHT}(i)$
3. if  $l \leq n$  and  $A[l] > A[i]$
4.   then  $\text{largest} \leftarrow l$
5. Else
6.    $\text{largest} \leftarrow i;$
7. if  $r \leq n$  and  $A[r] > A[\text{largest}]$
8.   then  $\text{largest} \leftarrow r$
9. if  $\text{largest} \neq i$
10.   then exchange  $A[i] \leftrightarrow A[\text{largest}]$
11.           MAX-HEAPIFY( $A, \text{largest}, n$ )



# MAX-HEAPIFY Running Time

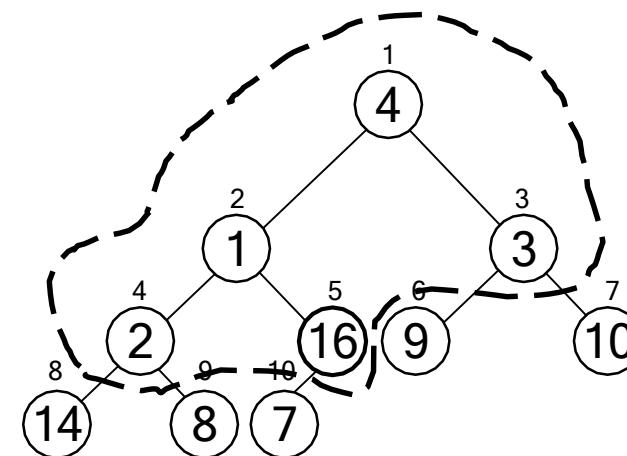
- Intuitively:
  - It traces a path from the root to a leaf (longest path length:  $h$ )
  - At each level, it makes exactly 2 comparisons
  - Total number of comparisons is  $2h$
  - Running time is  $O(h)$  or  $O(lgn)$
- Running time of MAX-HEAPIFY is  $O(lgn)$
- Can be written in terms of the height of the heap, as being  $O(h)$ 
  - Since the height of the heap is  $\lfloor lgn \rfloor$

# Building a Heap

- Convert an array  $A[1 \dots n]$  into a max-heap ( $n = \text{length}[A]$ )
- The elements in the subarray  $A[\lfloor n/2 \rfloor + 1 \dots n]$  are leaves
- Apply MAX-HEAPIFY on elements between 1 and  $\lfloor n/2 \rfloor$

*Alg:* BUILD-MAX-HEAP( $A$ )

1.  $n = \text{length}[A]$
2. **for**  $i \leftarrow \lfloor n/2 \rfloor$  **downto** 1
3.   **do** MAX-HEAPIFY( $A, i, n$ )



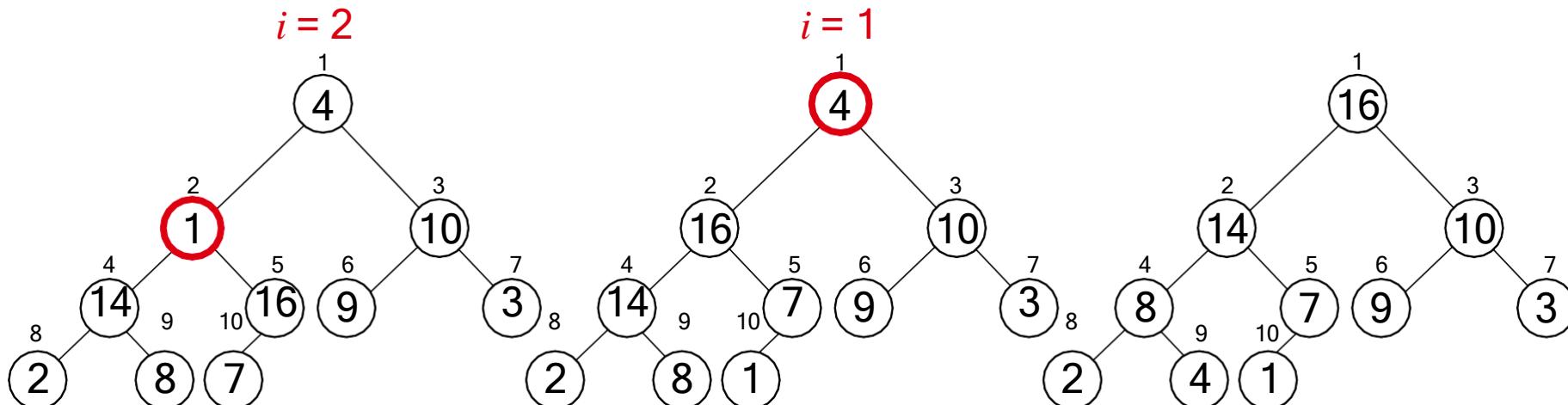
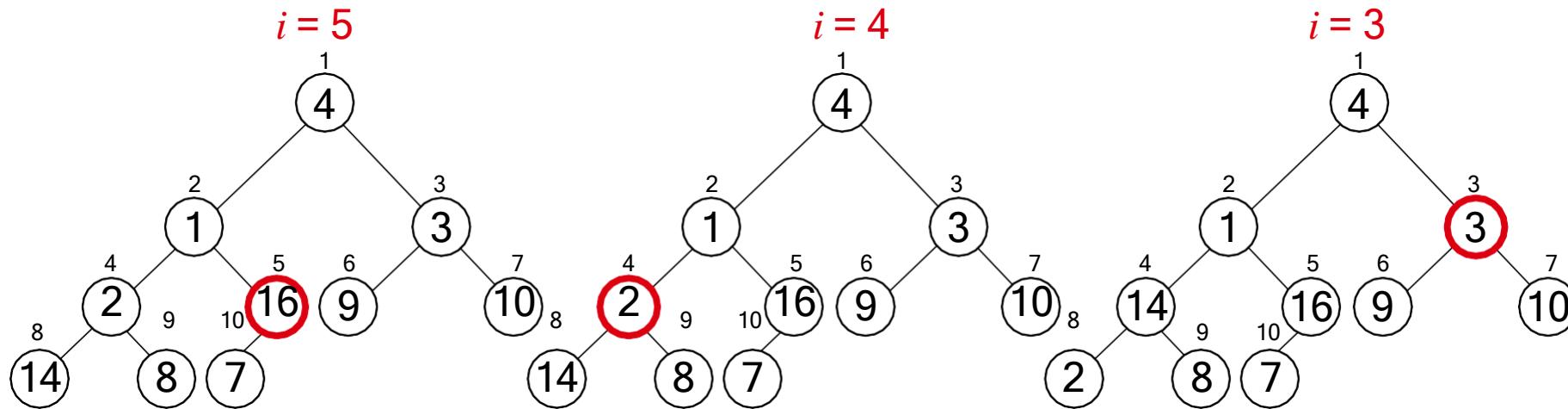
A: 

4	1	3	2	16	9	10	14	8	7
---	---	---	---	----	---	----	----	---	---

# Example:

A

4	1	3	2	16	9	10	14	8	7
---	---	---	---	----	---	----	----	---	---



# Running Time of BUILD MAX HEAP

## **Alg** BUILD-MAX-HEAP(*A*)

1.  $n = \text{length}[A]$
  2. **for**  $i \leftarrow \lfloor n/2 \rfloor$  **downto** 1
  3.     **do** MAX-HEAPIFY(*A*, *i*, *n*)
- $O(\lg n)$  }  $O(n)$

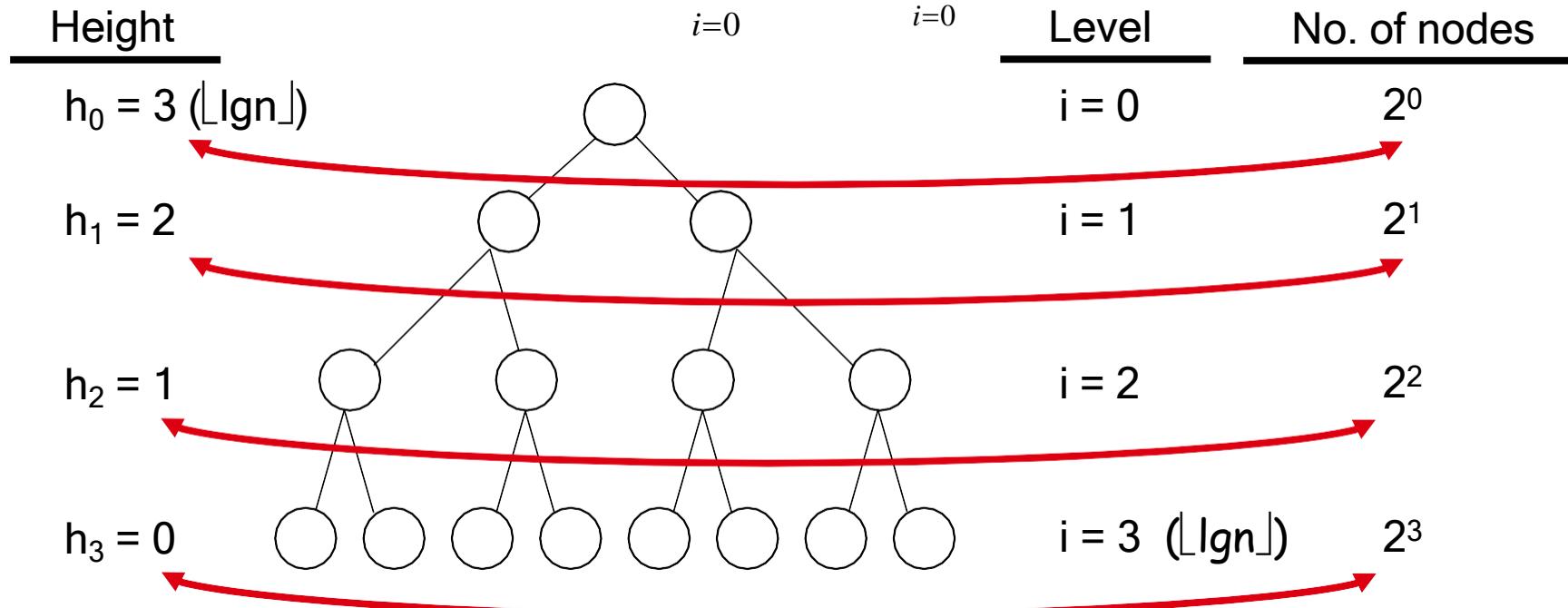
⇒ Running time:  $O(n \lg n)$

- This is not an asymptotically tight upper bound

# Running Time of BUILD MAX HEAP

- HEAPIFY takes  $O(h)$   $\Rightarrow$  the cost of HEAPIFY on a node  $i$  is proportional to the height of the node  $i$  in the tree

$$\Rightarrow T(n) = \sum_{i=0}^h n_i h_i = \sum_{i=0}^h 2^i (h-i) = O(n)$$



$h_i = h - i$  height of the heap rooted at level  $i$

$n_i = 2^i$  number of nodes at level  $i$

# Running Time of BUILD MAX HEAP

$$\begin{aligned}
 T(n) &= \sum_{i=0}^h n_i h_i && \text{Cost of HEAPIFY at level } i * \text{ number of nodes at that level} \\
 &= \sum_{i=0}^h 2^i (h-i) && \text{Replace the values of } n_i \text{ and } h_i \text{ computed before} \\
 &= \sum_{i=0}^h \frac{h-i}{2^{h-i}} 2^h && \text{Multiply by } 2^h \text{ both at the nominator and denominator and} \\
 &&& \text{write } 2^i \text{ as } \frac{1}{2^{-i}} \\
 &= 2^h \sum_{k=0}^h \frac{k}{2^k} && \text{Change variables: } k = h - i \\
 &\leq n \sum_{k=0}^{\infty} \frac{k}{2^k} && \text{The sum above is smaller than the sum of all elements to } \infty \\
 &= O(n) && \text{The sum above is smaller than } 2
 \end{aligned}$$

Running time of BUILD-MAX-HEAP:  $T(n) = O(n)$

# Heapsort

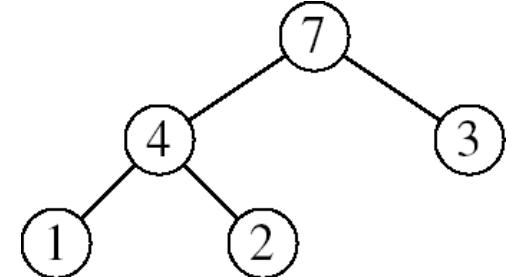
Heapsort—an interesting sorting algorithm discovered by J. W. J. Williams. This is a two-stage algorithm that works as follows.

**Stage 1 (heap construction):** Construct a heap for a given array.

**Stage 2 (maximum deletions):** Apply the root-deletion operation  $n - 1$  times to the remaining heap.

# Heapsort

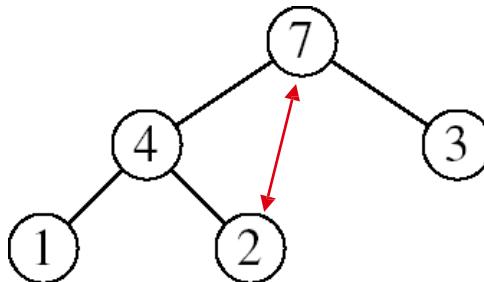
- Goal:
  - Sort an array using heap representations
- Idea:
  - Build a **max-heap** from the array
  - Swap the root (the maximum element) with the last element in the array
  - “Discard” this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root
  - Repeat this process until only one node remains



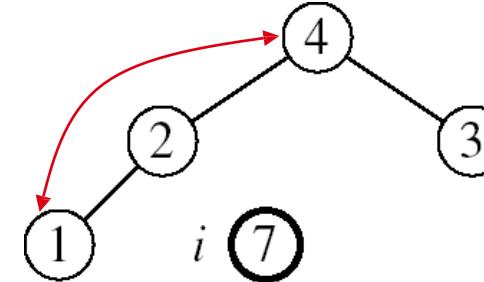
# Heapsort ...

Example:

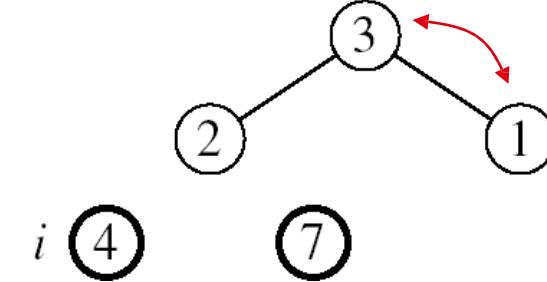
$A=[7, 4, 3, 1, 2]$



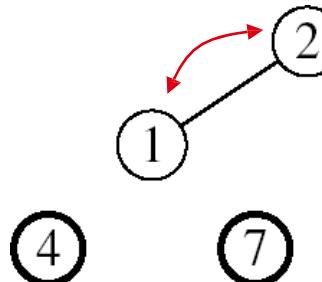
MAX-HEAPIFY( $A, 1, 4$ )



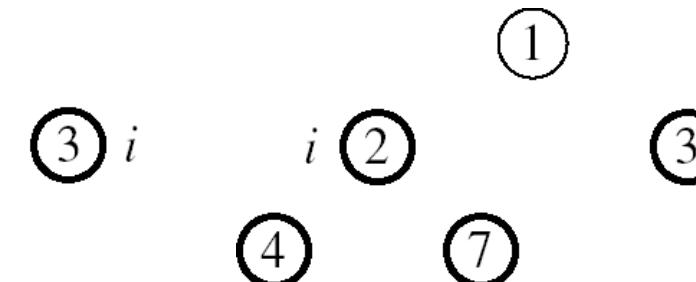
MAX-HEAPIFY( $A, 1, 3$ )



MAX-HEAPIFY( $A, 1, 2$ )



MAX-HEAPIFY( $A, 1, 1$ )



$A$ 

1	2	3	4	7
---	---	---	---	---

# Heapsort ...

1. **BUILD-MAX-HEAP( $A$ )**
2. **for**  $i \leftarrow \text{length}[A]$  **downto** 2
3.     **do exchange**  $A[1] \leftrightarrow A[i]$
4.     **MAX-HEAPIFY( $A, 1, i - 1$ )**

$O(n)$

$O(\lg n)$

}  $n-1$  times

- Running time:  $O(n \lg n)$  --- Can be shown to be  $\Theta(n \lg n)$



# Heapsort : HEAP-EXTRACT-MAX

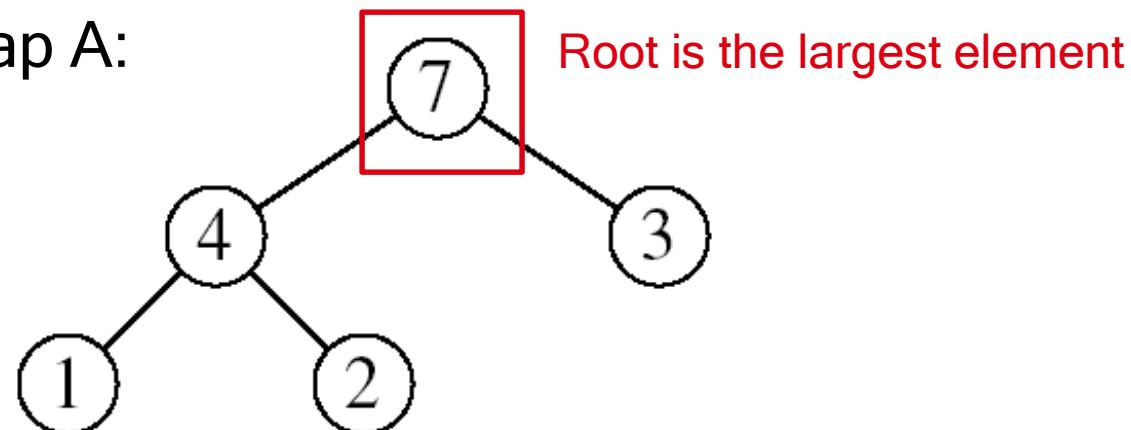
## Goal:

- Extract the largest element of the heap (i.e., return the max value and also remove that element from the heap)

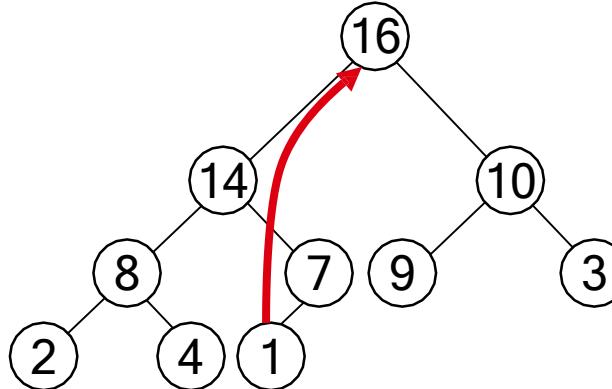
## Idea:

- Exchange the root element with the last
- Decrease the size of the heap by 1 element
- Call MAX-HEAPIFY on the new root, on a heap of size n-1

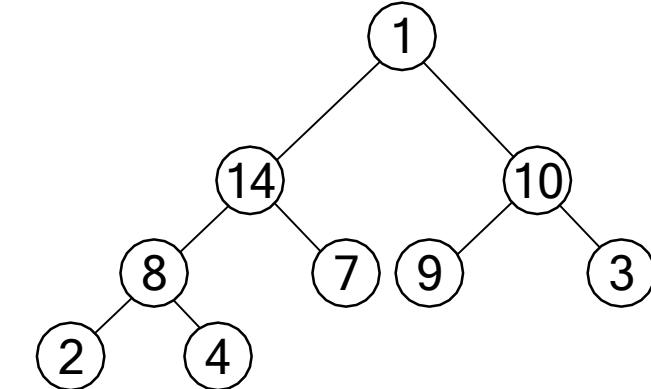
Heap A:



# Heapsort : HEAP-EXTRACT-MAX ...

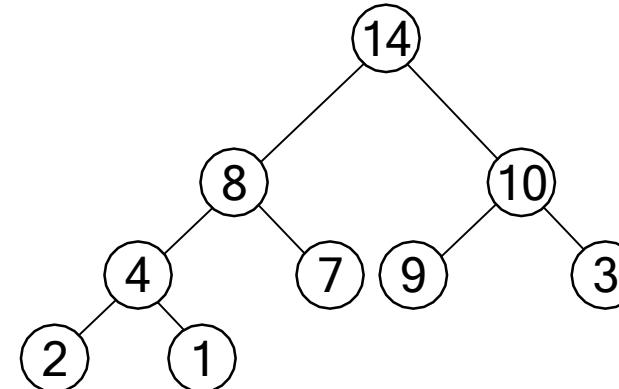


max = 16



Heap size decreased with 1

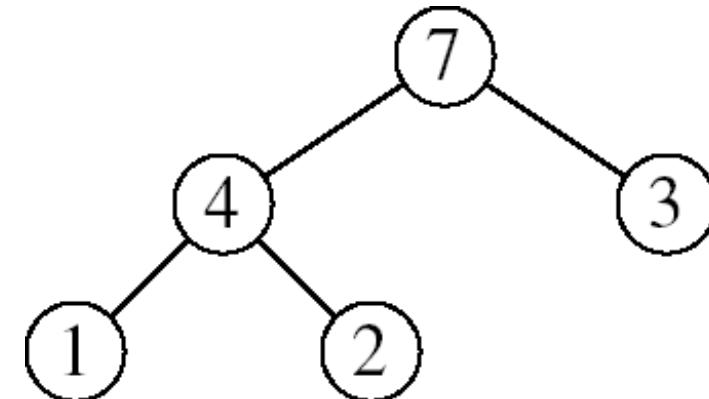
Call MAX-HEAPIFY( $A, 1, n-1$ )



# Heapsort : HEAP-EXTRACT-MAX ...

*Alg:* HEAP-EXTRACT-MAX( $A, n$ )

1. if  $n < 1$
2. then error “heap underflow”
3.  $\max \leftarrow A[1]$
4.  $A[1] \leftarrow A[n]$
5. MAX-HEAPIFY( $A, 1, n-1$ ) ▷ remakes heap
6. return  $\max$



Running time:  $O(\lg n)$

# Heapsort

- We can perform the following operations on heaps:
  - MAX-HEAPIFY  $O(lgn)$
  - BUILD-MAX-HEAP  $O(n)$
  - HEAP-SORT  $O(nlgn)$
  - MAX-HEAP-INSERT  $O(lgn)$
  - HEAP-EXTRACT-MAX  $O(lgn)$
  - HEAP-INCREASE-KEY  $O(lgn)$
  - HEAP-MAXIMUM  $O(1)$

Average  
 $O(lgn)$

“Thank you”

*Any Questions ?*

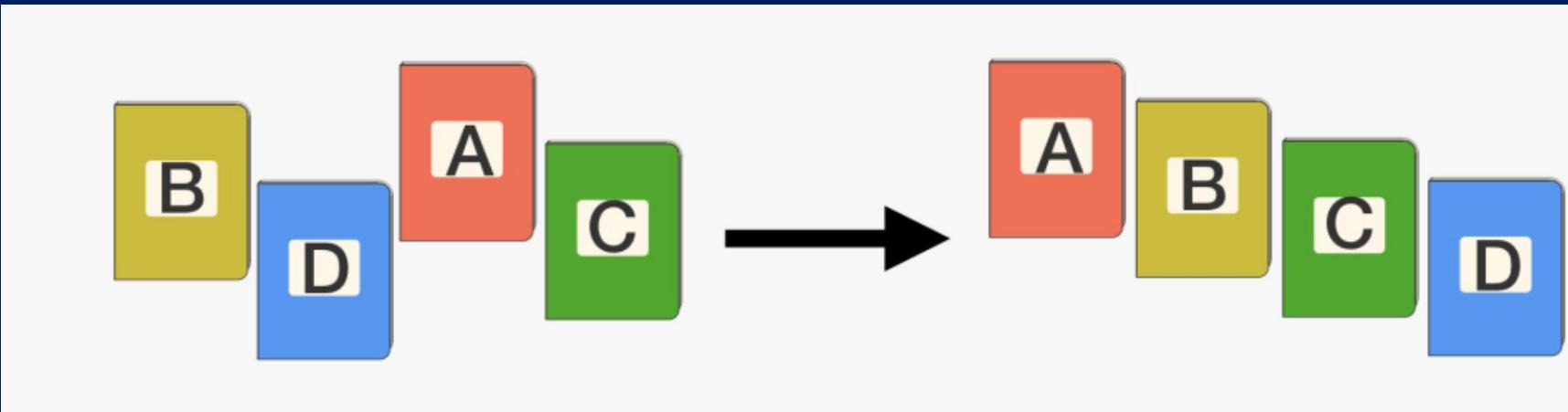


**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

## Chapter 6: Non-comparison Sorting



Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

# How Fast Can We Sort?

	Best	Average	Worst
<b>Selection Sort</b>	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
<b>Bubble Sort</b>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
<b>Insertion Sort</b>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
<b>Merge Sort</b>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$
<b>Quick Sort</b>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$
<b>Heap Sort</b>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$

- What is common to all these algorithms?
  - Make **comparisons** between input elements

$a_i < a_j$ ,    $a_i \leq a_j$ ,    $a_i = a_j$ ,    $a_i \geq a_j$ ,   or    $a_i > a_j$

To sort  $n$  elements, comparison sorts **must** make  $\Omega(n \lg n)$  comparisons in the worst case.

# Can we do better?

- Linear sorting algorithms
  - Counting Sort
  - Radix Sort
  - Bucket sort
- Make certain assumptions about the data
- Linear sorts are NOT “**comparison sorts**”

# COUNTING SORT ALGORITHM



# Counting Sort

for  $i \leftarrow 1$  to  $k$

do  $C[i] \leftarrow 0;$

for  $j \leftarrow 1$  to  $\text{length}[A]$

do  $C[A[j]] \leftarrow C[A[j]] + 1;$

for  $i \leftarrow 2$  to  $k$

do  $C[i] \leftarrow C[i] + C[i - 1];$

for  $j \leftarrow \text{length}[A]$  downto 1

do begin

$B[C[A[j]]] \leftarrow A[j];$

$C[A[j]] \leftarrow C[A[j]] - 1;$

end - for

$A: 3, 6, 4, 1, 3, 4, 1, 4$

$C: 2, 0, 2, 3, 0, 1$

$C: 2, 2, 4, 7, 7, 8$

$A: 3, 6, 4, 1, 3, 4, 1, \hat{4}$

$B: , , , , , , 4,$

$C: 2, 2, 4, 6, 7, 8$

# Counting sort ...

$A: 3, 6, 4, 1, 3, 4, \hat{1}, 4$

$A: 3, 6, 4, 1, 3, \hat{4}, 1, 4$

$B: \ , 1, \ , \ , \ , \ , 4,$

$B: \ , 1, \ , \ , \ , 4, 4,$

$C: 1, 2, 4, 6, 7, 8$

$C: 1, 2, 4, 5, 7, 8$

$A: 3, 6, 4, 1, \hat{3}, 4, 1, 4$

$A: 3, 6, 4, \hat{1}, 3, 4, 1, 4$

$B: \ , 1, \ , 3, \ , 4, 4,$

$B: 1, 1, \ , 3, \ , 4, 4,$

$C: 1, 2, 3, 5, 7, 8$

$C: 0, 2, 3, 5, 7, 8$

$A: 3, 6, \hat{4}, 1, 3, 4, 1, 4$

$A: 3, \hat{6}, 4, 1, 3, 4, 1, 4$

$B: 1, 1, \ , 3, 4, 4, 4,$

$B: 1, 1, \ , 3, 4, 4, 4, 6$

$C: 0, 2, 3, 4, 7, 8$

$C: 0, 2, 3, 4, 7, 7$

$A: \hat{3}, 6, 4, 1, 3, 4, 1, 4$

$B: 1, 1, 3, 3, 4, 4, 4, 6$

$C: 0, 2, 2, 4, 7, 7$

# Counting sort ...

```

for  $i \leftarrow 1$  to  $k$                                  $O(k)$ 
    do  $C[i] \leftarrow 0;$ 

for  $j \leftarrow 1$  to  $\text{length}[A]$                    $O(n)$ 
    do  $C[A[j]] \leftarrow C[A[j]] + 1;$ 

for  $i \leftarrow 2$  to  $k$                                  $O(k)$ 
    do  $C[i] \leftarrow C[i] + C[i - 1];$ 

for  $j \leftarrow \text{length}[A]$  downto 1                 $O(n)$ 
    do begin
         $B[C[A[j]]] \leftarrow A[j];$ 
         $C[A[j]] \leftarrow C[A[j]] - 1;$ 
    end - for

```

$O(n+k)$

**Counting sort is **stable****

**That is, the same value appear in the output array in the same order as they do in the input array.**

# **RADIX SORT ALGORITHM**



# Radix Sort

- Represents keys as  $d$ -digit numbers in some base- $k$

$\text{key} = x_1x_2\dots x_d$       where  $0 \leq x_i \leq k-1$

- Example:  $\text{key}=15$

$\text{key}_{10} = 15, d=2, k=10$     where  $0 \leq x_i \leq 9$

$\text{key}_2 = 1111, d=4, k=2$     where  $0 \leq x_i \leq 1$

# Radix Sort ...

- Assumptions

$d=O(1)$  and  $k=O(n)$

- Sorting looks at one column at a time
  - For a  $d$  digit number, sort the least significant digit first
  - Continue sorting on the next least significant digit, until all digits have been sorted
  - Requires only  $d$  passes through the list



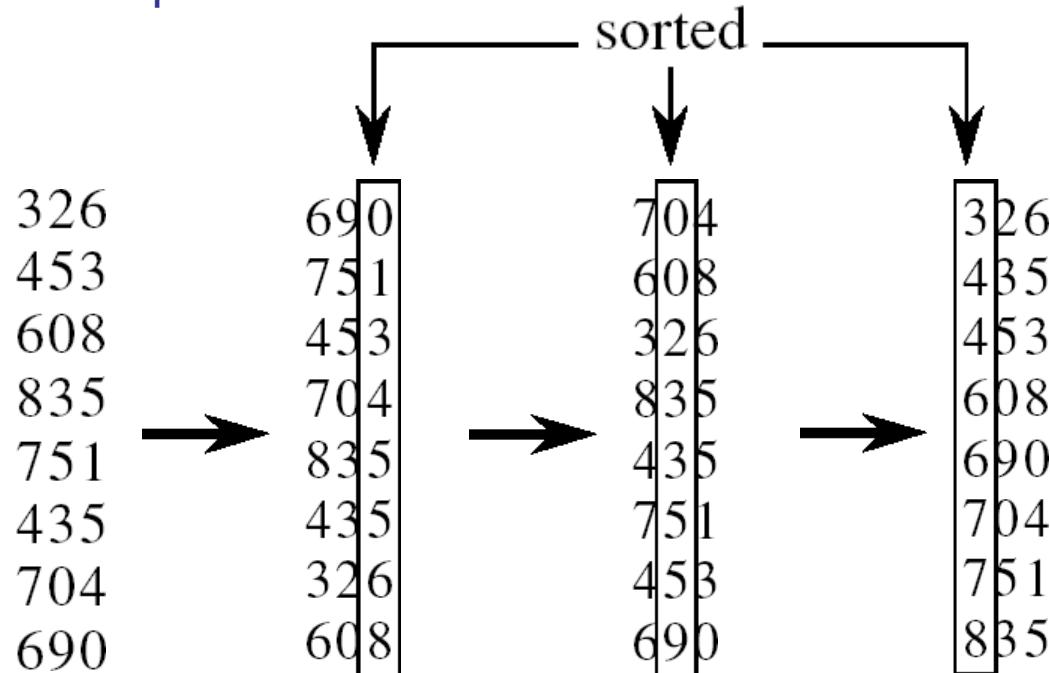
# Radix Sort ...

Alg.: RADIX-SORT( $A, d$ )

for  $i \leftarrow 1$  to  $d$

do use a **stable** sort to sort array  $A$  on digit  $i$

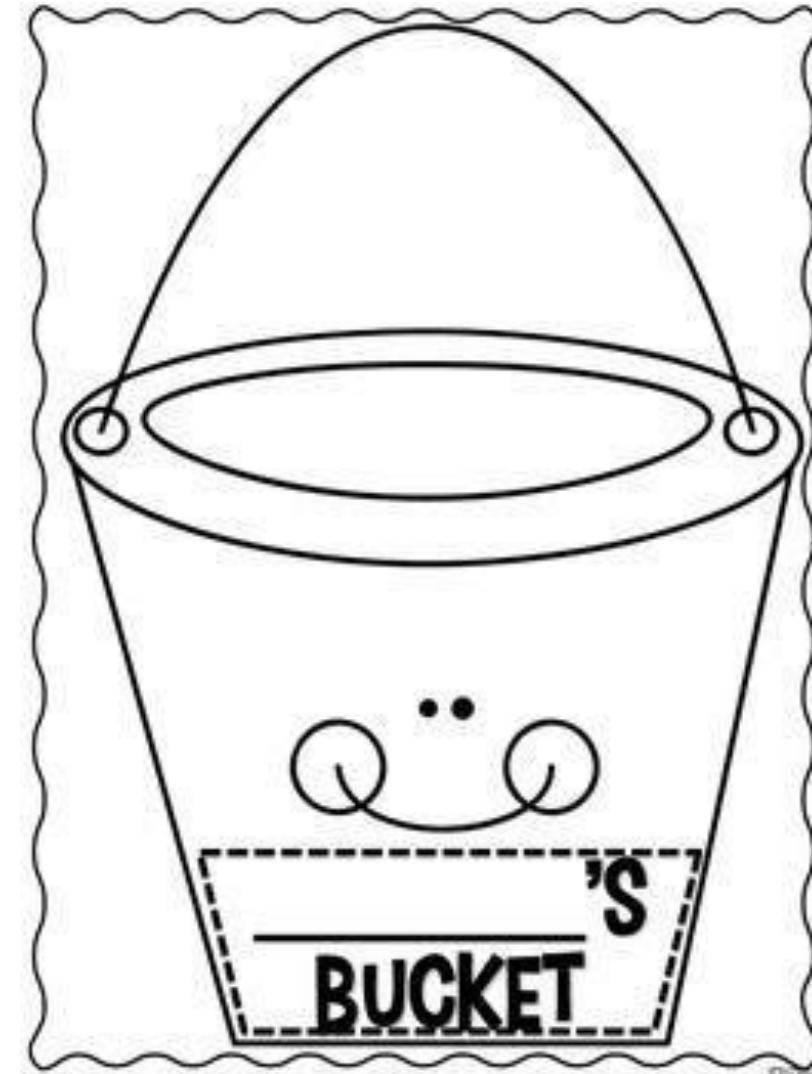
(stable sort: preserves order of identical elements)



# Analysis of Radix Sort

- Given  $n$  numbers of  $d$  digits each, where each digit may take up to  $k$  possible values, RADIX-SORT correctly sorts the numbers in  $O(d(n+k))$ 
  - One pass of sorting per digit takes  $O(n+k)$  assuming that we use **counting sort**
  - There are  $d$  passes (for each digit)
  - Assuming  $d=O(1)$  and  $k=O(n)$ , running time is  $O(n)$

# Bucket Sort

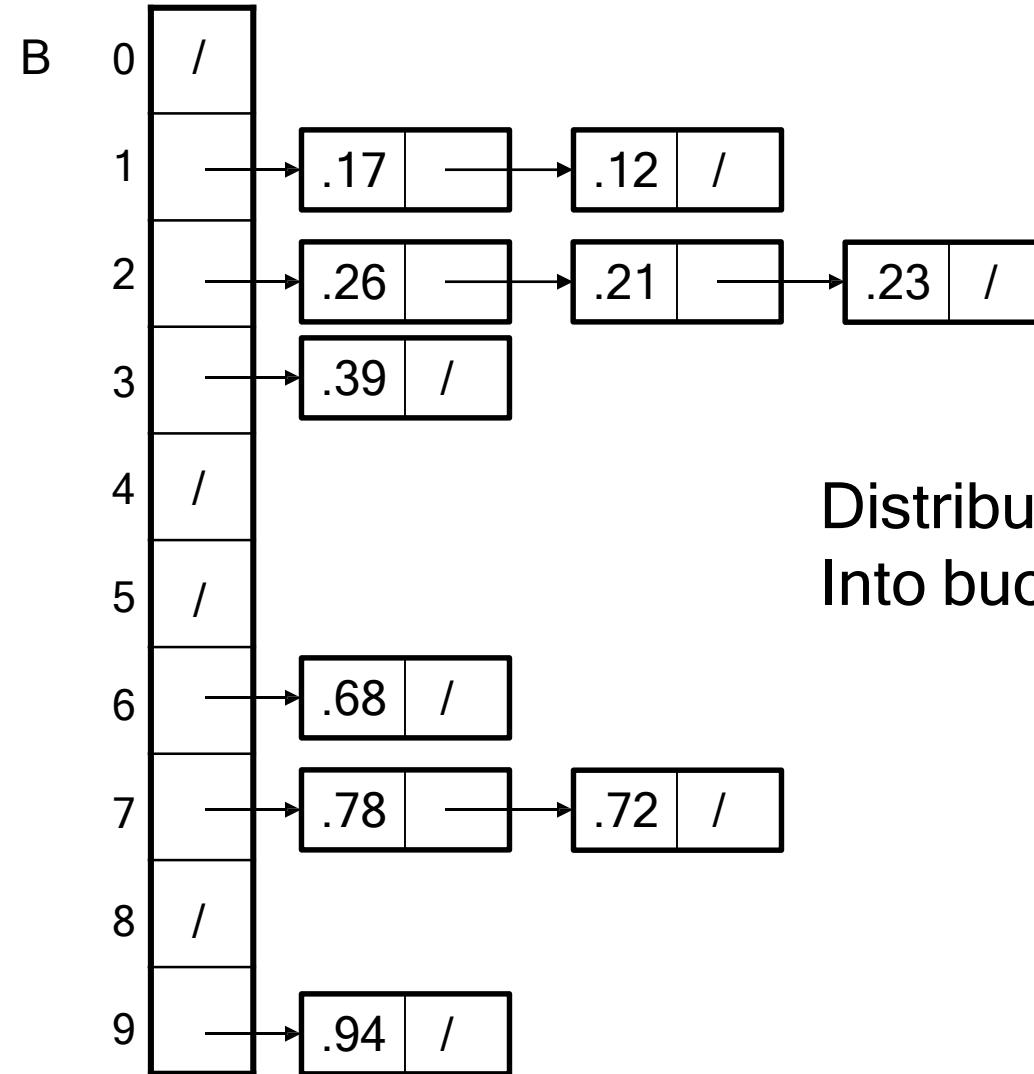


# Bucket Sort

- Assumption:
  - the input is generated by a random process that distributes elements uniformly over  $[0, 1)$
- Idea:
  - Divide  $[0, 1)$  into **k** equal-sized buckets ( $k=\Theta(n)$ )
  - Distribute the **n** input values into the buckets
  - Sort each bucket (e.g., using quicksort)
  - Go through the buckets in order, listing elements in each one
- **Input:**  $A[1 \dots n]$ , where  $0 \leq A[i] < 1$  for all  $i$
- **Output:** elements  $A[i]$  sorted

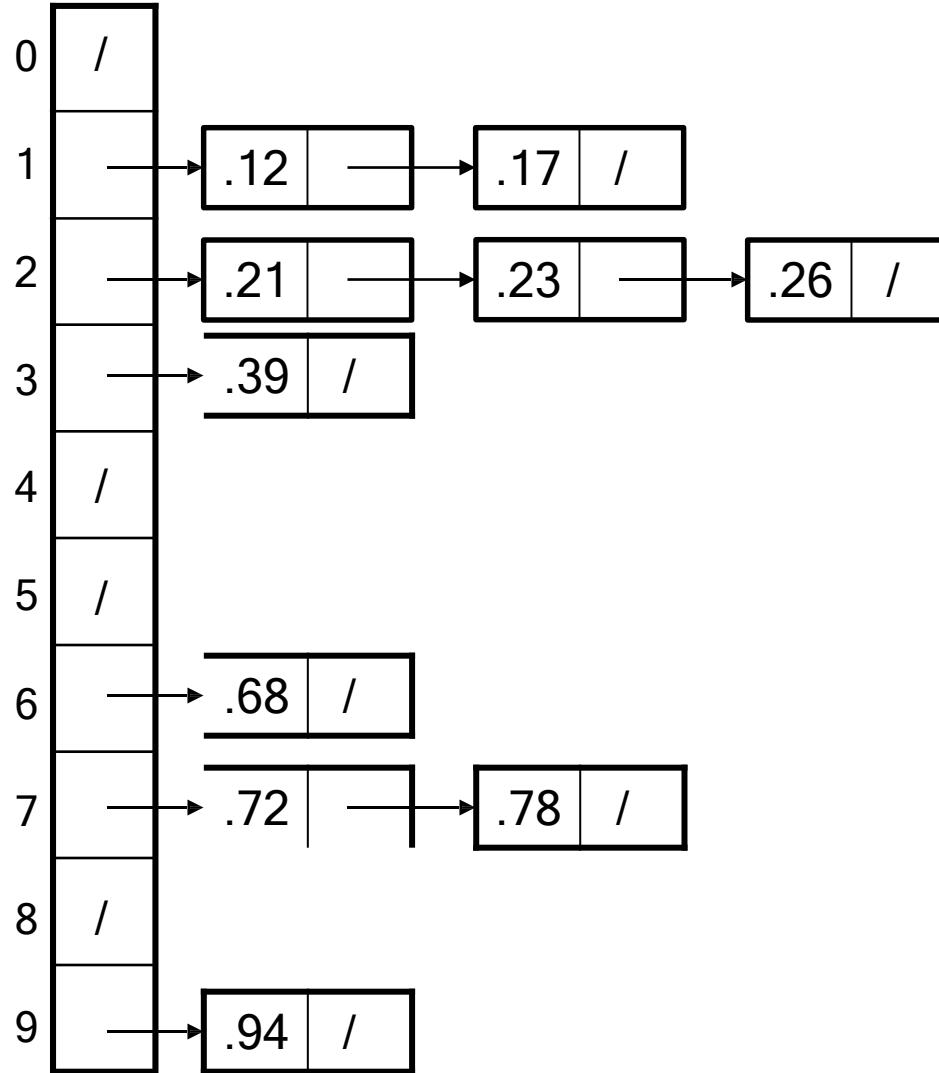
# Bucket Sort ..

A	1	.78
	2	.17
	3	.39
	4	.26
	5	.72
	6	.94
	7	.21
	8	.12
	9	.23
	10	.68



Distribute  
Into buckets

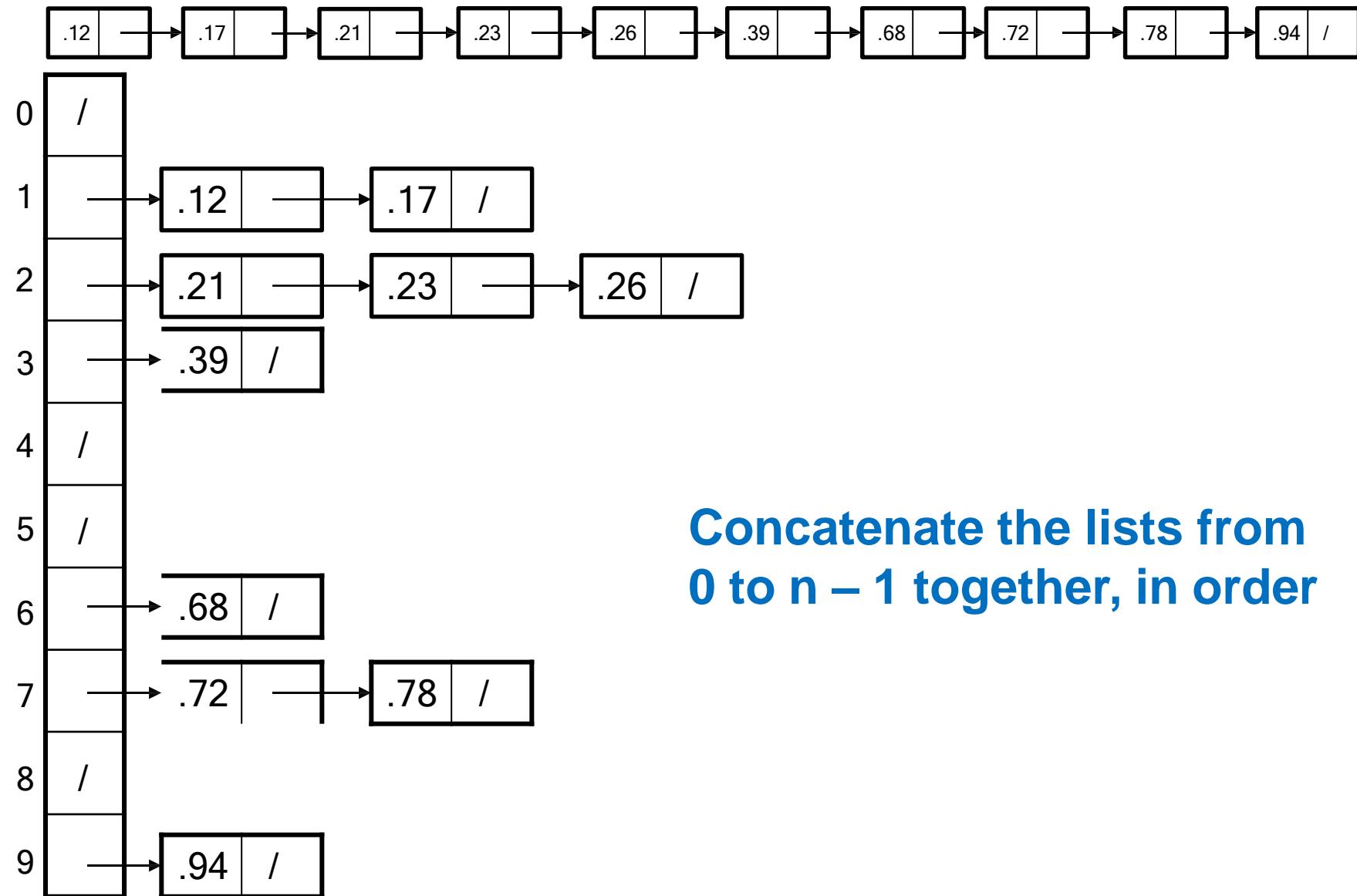
# Bucket Sort ..



**Sort within each bucket**

# Bucket Sort ..

Accredited with **A** Grade by NAAC



# Analysis of Bucket Sort

*Alg.:* BUCKET-SORT( $A, n$ )

```
for i ← 1 to n
    do insert  $A[i]$  into list  $B[\lfloor nA[i] \rfloor]$ 
for i ← 0 to k - 1
    do sort list  $B[i]$  with quicksort sort
concatenate lists  $B[0], B[1], \dots, B[n - 1]$ 
together in order
return the concatenated lists
```

$\left. \begin{array}{l} O(n) \\ k O(n/k \log(n/k)) \\ = O(n \log(n/k)) \\ O(k) \end{array} \right\}$

---

$O(n)$  (if  $k = \Theta(n)$ )

“Thank you”

*Any Questions ?*

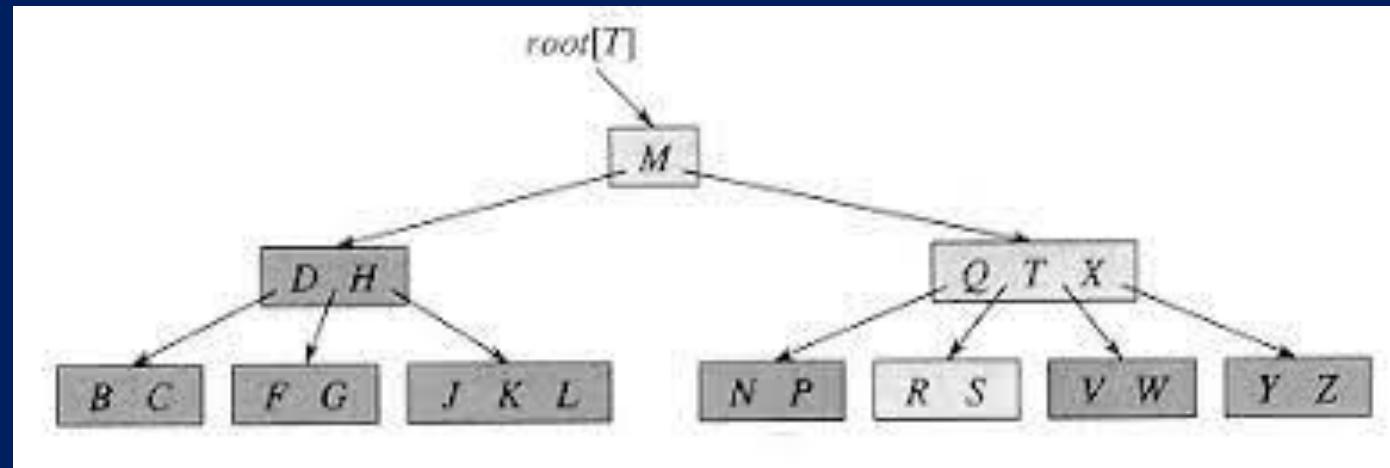


**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

## Chapter 7: B-Trees



Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

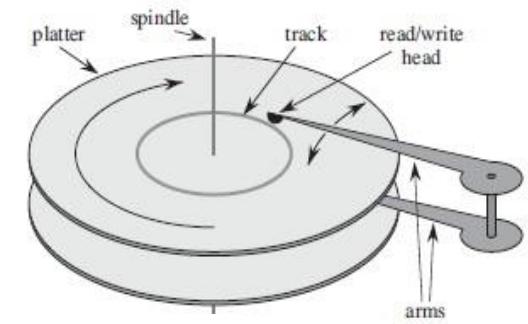
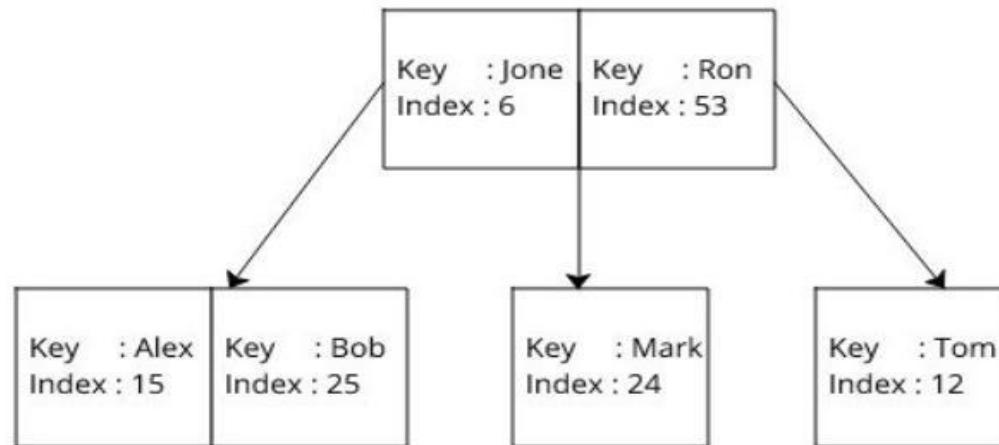
# B-Trees

**B-trees are balanced search trees designed to work well on disks or other direct access secondary storage devices.**

**B-Trees are better at minimizing disks I/O operations.**

**Database systems use B-Trees and its variants for indexing.**

Name	Mark	Age
Jone	5	28
Alex	32	45
Tom	37	23
Ron	87	13
Mark	20	48
Bob	89	32



# B-Trees

- B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like AVL and Red-Black Trees), it is assumed that everything is in the main memory.
- To understand the use of B-Trees, we must think of the huge amount of data that cannot fit in the main memory. When the number of keys is high, the data is read from the disk in the form of blocks. Disk access time is very high compared to the main memory access time. The main idea of using B-Trees is to reduce the number of disk accesses.
- Most of the tree operations (search, insert, delete, max, min, ..etc ) require  $O(h)$  disk accesses where  $h$  is the height of the tree. B-tree is a fat tree. The height of B-Trees is kept low by putting the maximum possible keys in a B-Tree node.
- Generally, the B-Tree node size is kept equal to the disk block size. Since the height of the B-tree is low so total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Tree, Red-Black Tree, etc.

# B-Tree

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

**1. Every node x has the following attributes:**

- a.  $x.n$ , the number of keys currently stored in node x,
- b. the  $x.n$  keys themselves,  $x.key_1, x.key_2, \dots, x.key_n$ , stored in non-decreasing, so that  $x.key_1 \leq x.key_2 \leq \dots \leq x.key_n$ ,
- c.  $x.leaf$ , a Boolean value that is true if x is a leaf and false if x is an internal node.



**n=3, keys 5,1,3**



# B-Trees

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

**1. Every node x has the following attributes:**

- a.  $x.n$ , the number of keys currently stored in node x,
- b. the  $x.n$  keys themselves,  $x.key_1, x.key_2, \dots, x.key_n$ , stored  **$n=3$ , keys 5,1,3** in non-decreasing, so that  $x.key_1 \leq x.key_2 \leq \dots \leq x.key_n$ ,
- c.  $x.leaf$ , a Boolean value that is true if x is a leaf and false if x is an internal node.



# B-Trees

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

**1. Every node x has the following attributes:**

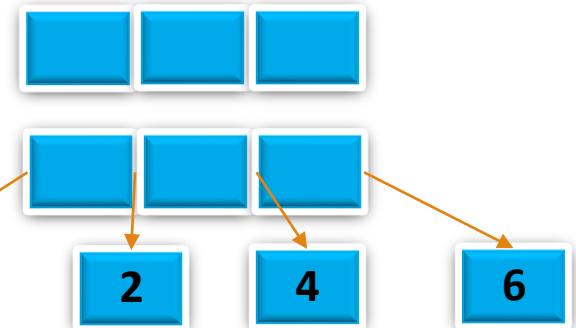
- a.  $x.n$ , the number of keys currently stored in node x,
  - b. the  $x.n$  keys themselves,  $x.key_1, x.key_2, \dots, x.key_n$ , stored in non-decreasing, so that  $x.key_1 \leq x.key_2 \leq \dots \leq x.key_n$ ,
  - c.  $x.leaf$ , a Boolean value that is true if x is a leaf and false if x is an internal node.
- 1    2    ...    n**
- n=3, keys 5,1,3**
- 1    3    5**

# B-Trees

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

**1. Every node x has the following attributes:**

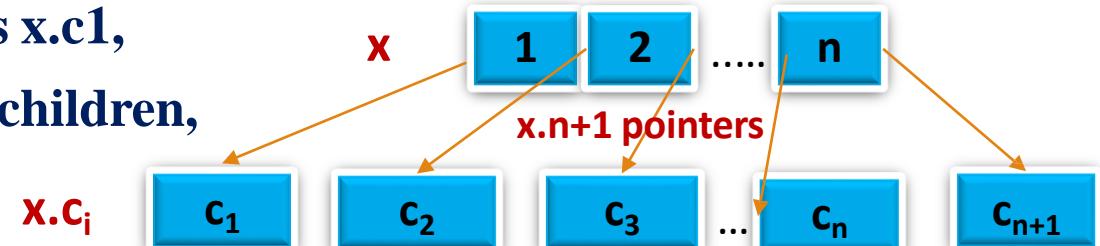
- a.  $x.n$ , the number of keys currently stored in node x,
- b. the  $x.n$  keys themselves,  $x.key_1, x.key_2, \dots, x.key_n$ , stored in non-decreasing, so that  $x.key_1 \leq x.key_2 \leq \dots \leq x.key_n$ ,
- c.  $x.leaf$ , a Boolean value that is true if x is a leaf and false if x is an internal node.



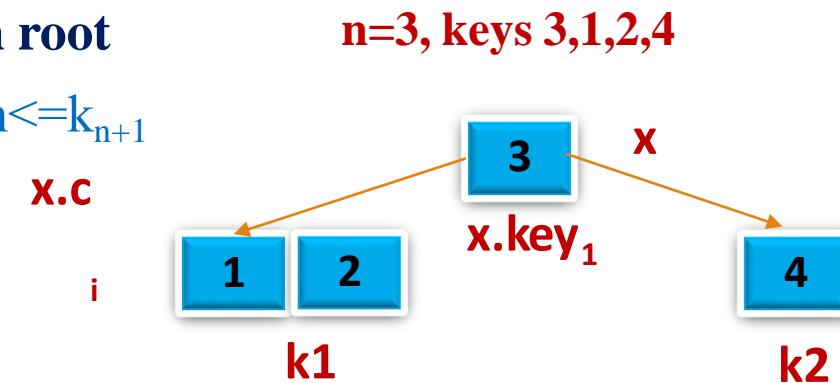
# B-Trees

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

2. Each internal node  $x$  also contains  $x.n+1$  pointers  $x.c_1, x.c_2, \dots, x.c_{n+1}$  to its children. Leaf nodes have no children, and so their  $c_i$  attributes are undefined.



3. The key  $x.key_i$  separate the ranges of keys stored in each subtree, if  $k_i$  is any key stored in the subtree with root  $x.c_i$  then,  $k_1 <= x.key_1 <= k_2 <= x.key_2 <= \dots <= x.key_n <= k_{n+1}$



# B-Trees

**A B-tree T is a rooted tree (whose root is T.root) having following properties:**

**4. All leaves have the same depth, which is the tree's height**

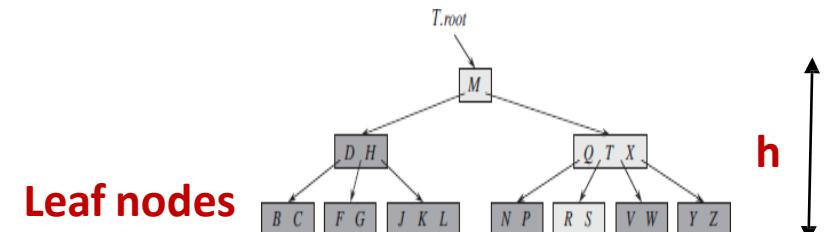
**h.**

**5. Nodes have lower and upper bounds, on the number of keys, they can contain.**

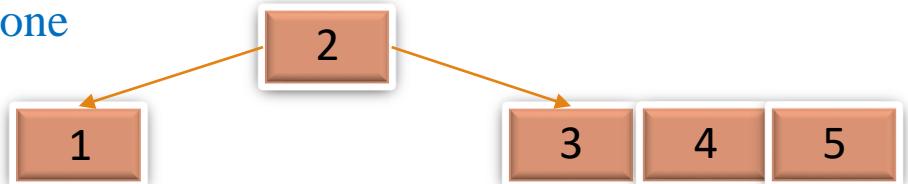
These bounds are fixed integer  $t \geq 2$  is called minimum degree of B-tree:

a. Every node other than the root, must have at least  $t-1$  keys.

Every internal node other than the root thus has at least  $t$  children. If the tree is non-empty, the root must have at least one key.



Keys-1,2,3,4,5  
Min. degree  $t=2$   
min. keys= $t-1=1$   
max. keys= $2t-1=3$



# B-Trees

**A B-tree T is a rooted tree having following properties:**

5. b. Every node must contain at most  $2*t-1$  keys.

Therefore an internal node may have atmost  $2*t$  children.

A node is full if it has  $2*t-1$  keys.

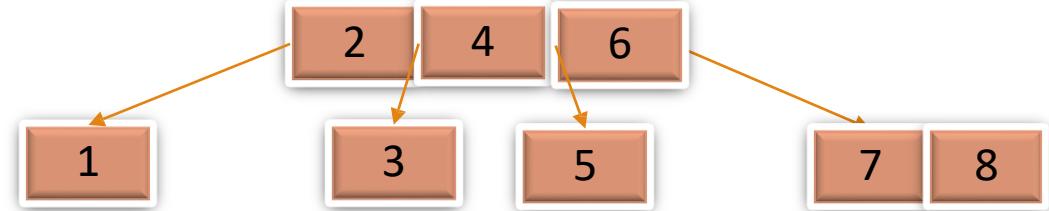
**Degree represents the lower bound on the number of children a node in the B Tree can have (except for the root). i.e the minimum number of children possible.**

Keys-1,2,3,4,5,6,7,8

Min. degree t=2

min. keys=t-1=1

max. keys=2t-1=3



**Order represents the upper bound on the number of children. ie. the maximum number possible.**

# B-Trees-Creation

**1,2,3,4,5,6,7,8,9,10,11,12,13,14,15**

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



Accredited with **A** Grade by **NAAC**

# B-Trees-Creation...

1,**2**,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



# B-Trees-Creation...

1,2,**3**,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



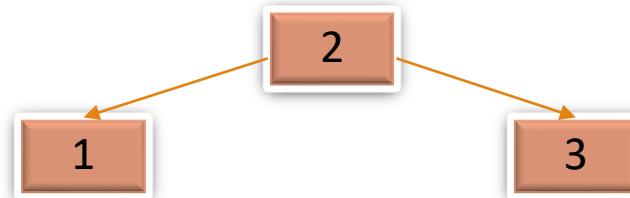
# B-Trees-Creation...

1,2,3,**4**,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



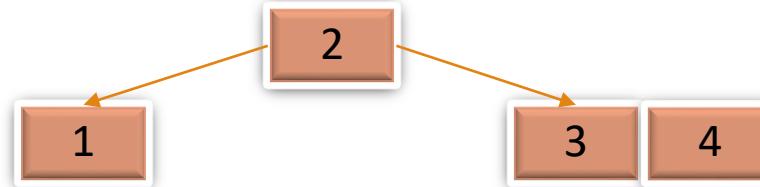
# B-Trees-Creation...

1,2,3,**4**,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



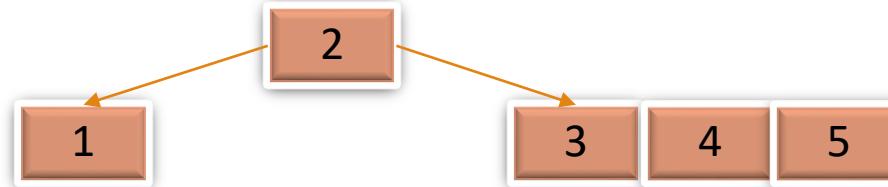
# B-Trees-Creation...

1,2,3,4,**5**,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



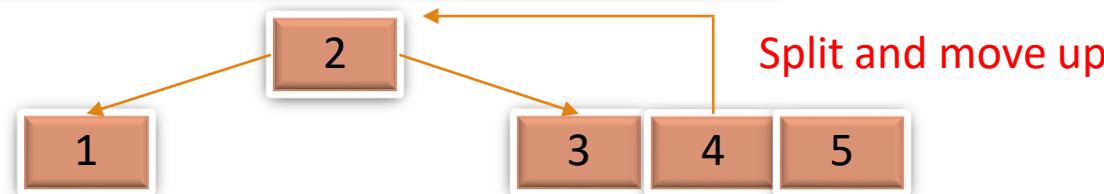
# B-Trees-Creation...

1,2,3,4,5,**6**,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



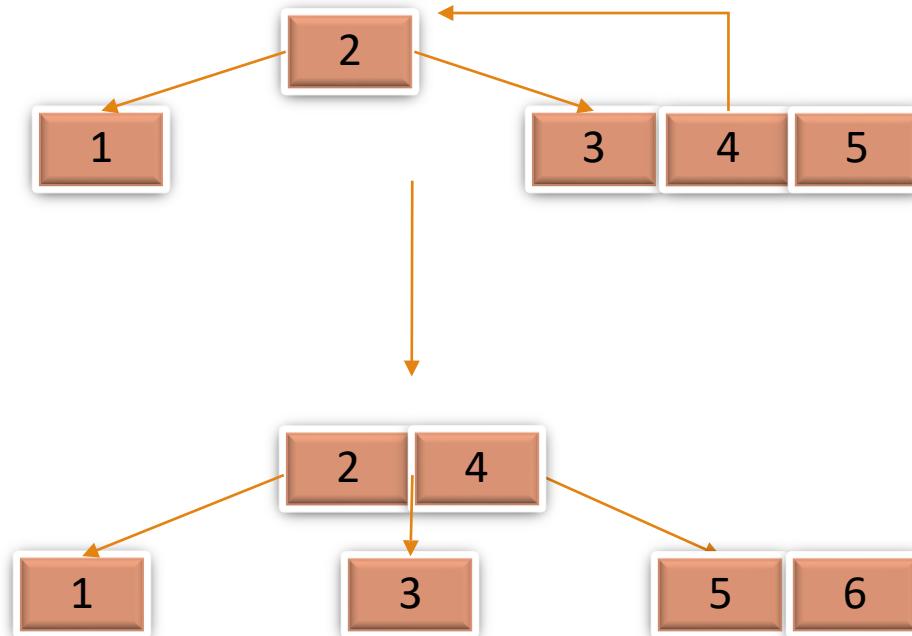
# B-Trees-Creation...

1,2,3,4,5,**6**,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



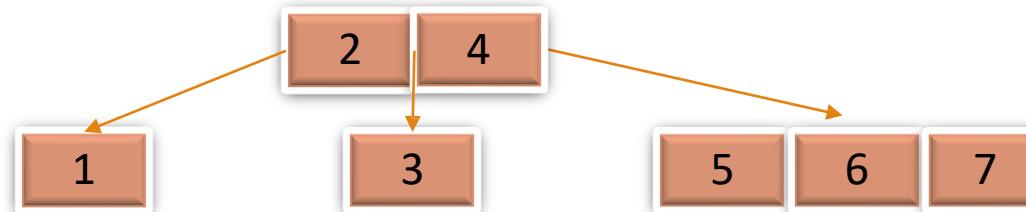
# B-Trees-Creation...

1,2,3,4,5,6,**7**,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



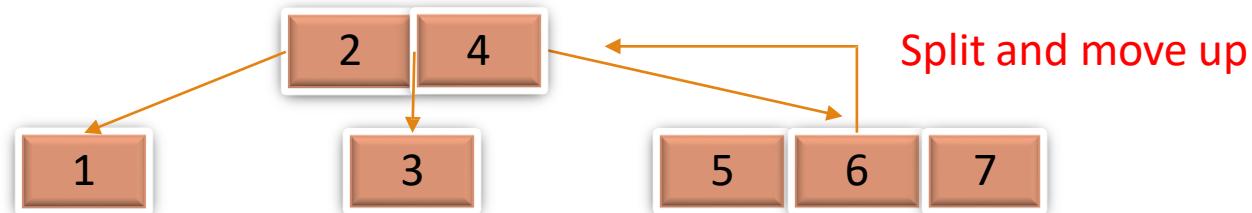
# B-Trees-Creation...

1,2,3,4,5,6,7,**8**,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



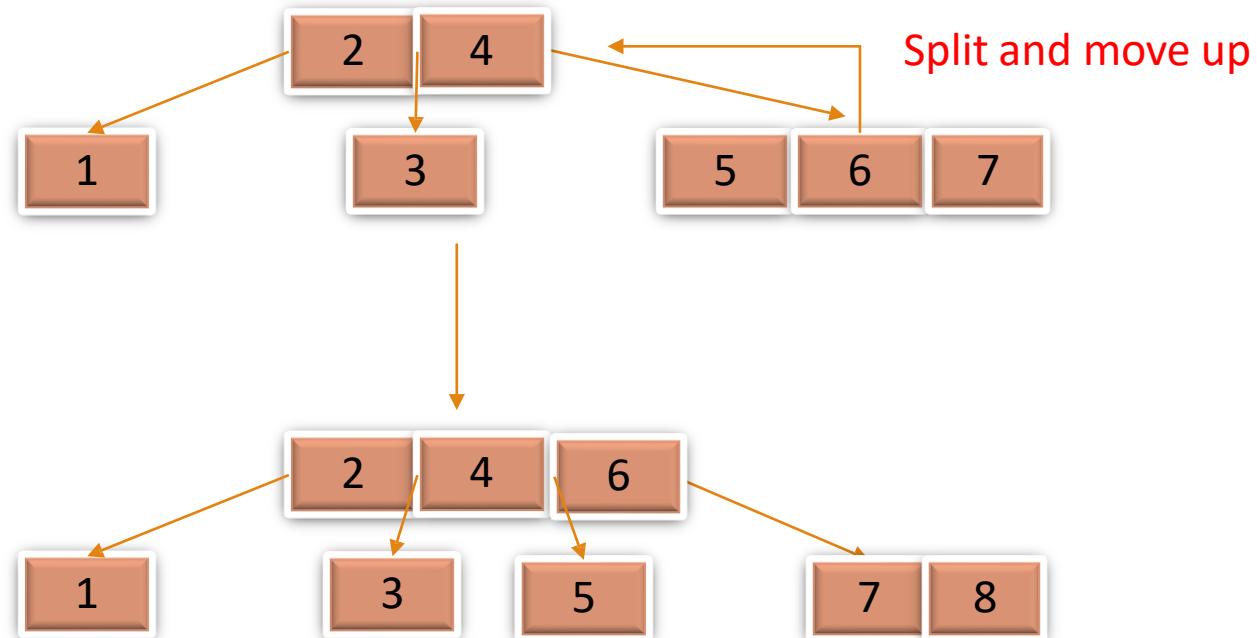
# B-Trees-Creation...

1,2,3,4,5,6,7,8,**9**,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



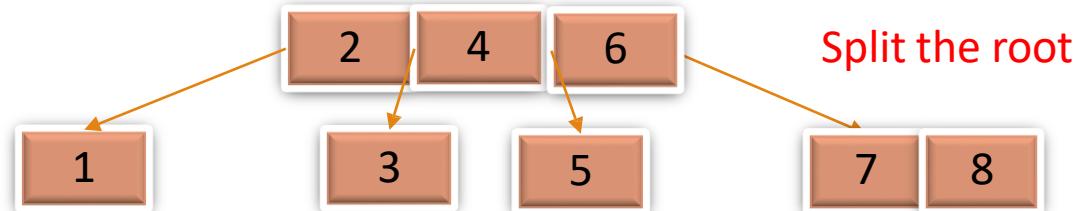
# B-Trees-Creation...

1,2,3,4,5,6,7,8,**9**,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



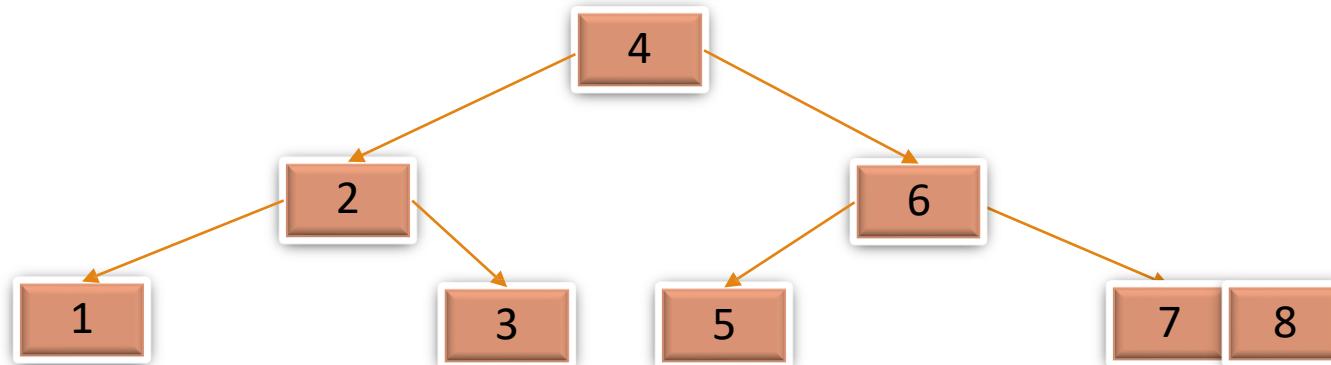
# B-Trees-Creation...

1,2,3,4,5,6,7,8,**9**,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



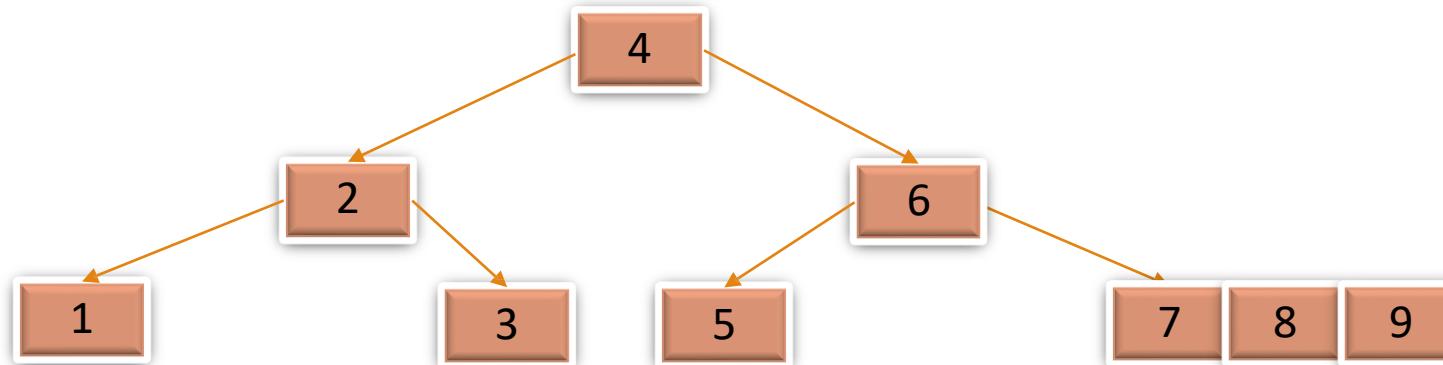
# B-Trees-Creation...

1,2,3,4,5,6,7,8,**9**,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



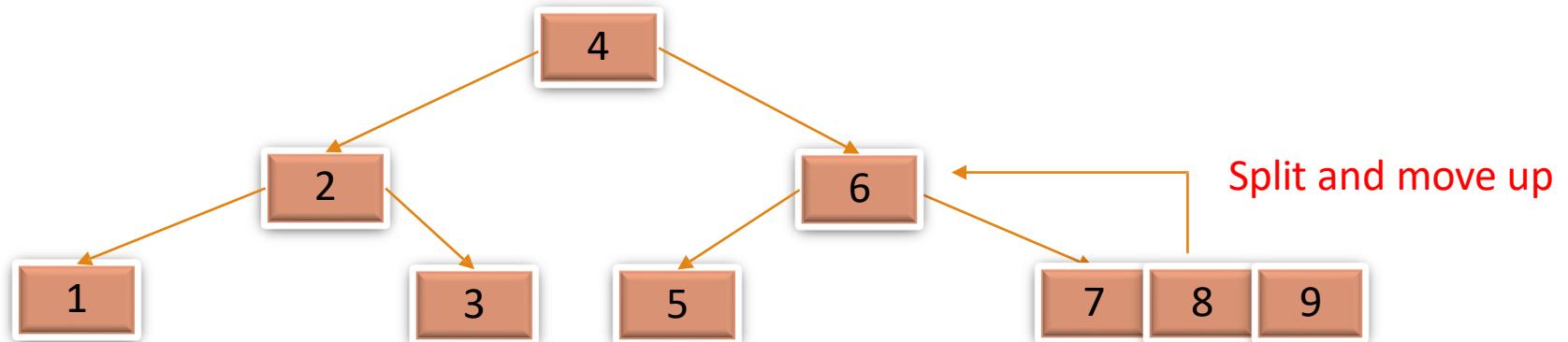
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,**10**,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



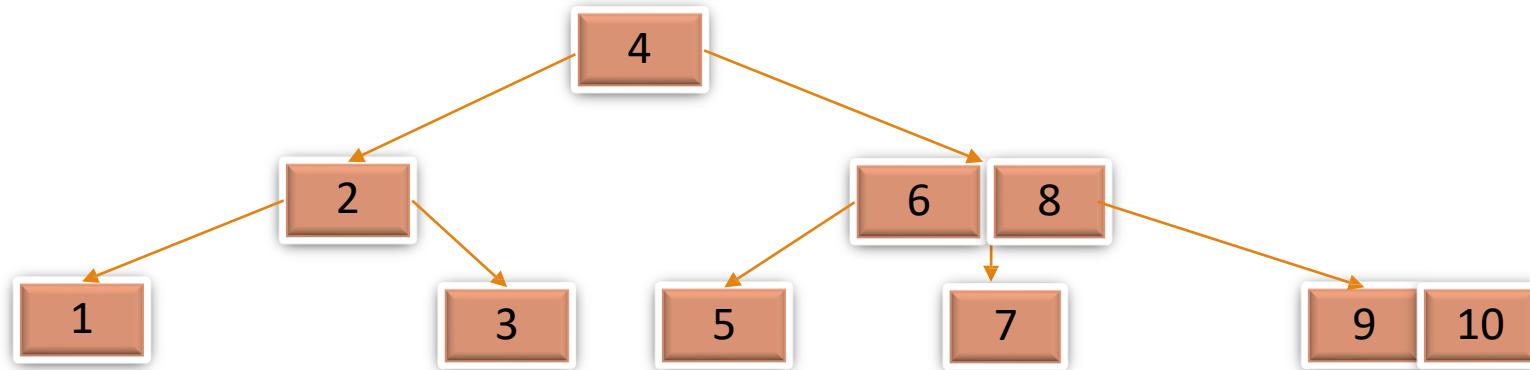
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,**10**,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



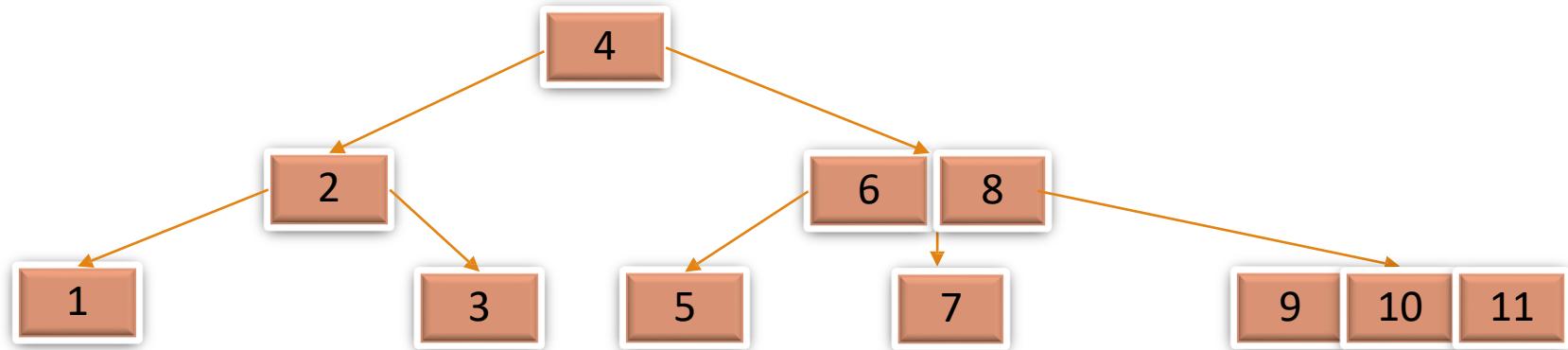
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,**11**,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



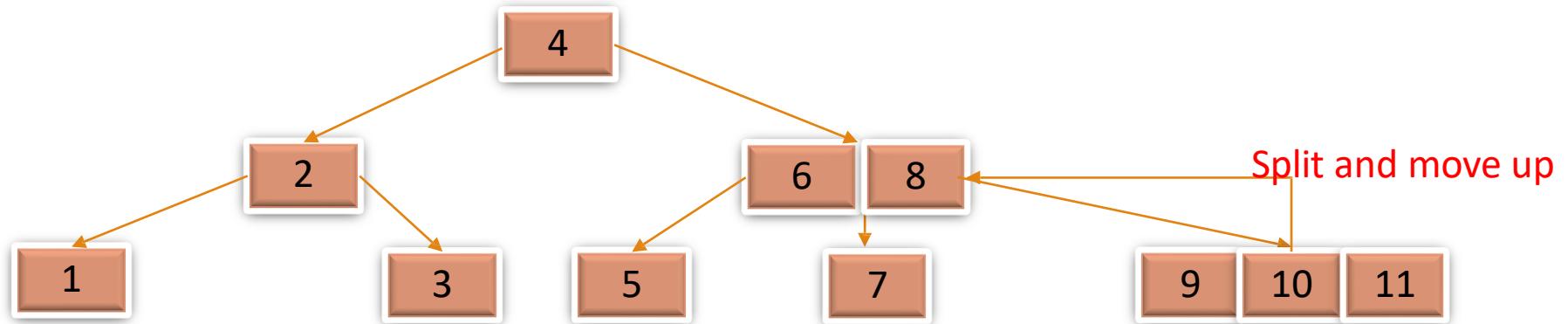
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,**12**,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



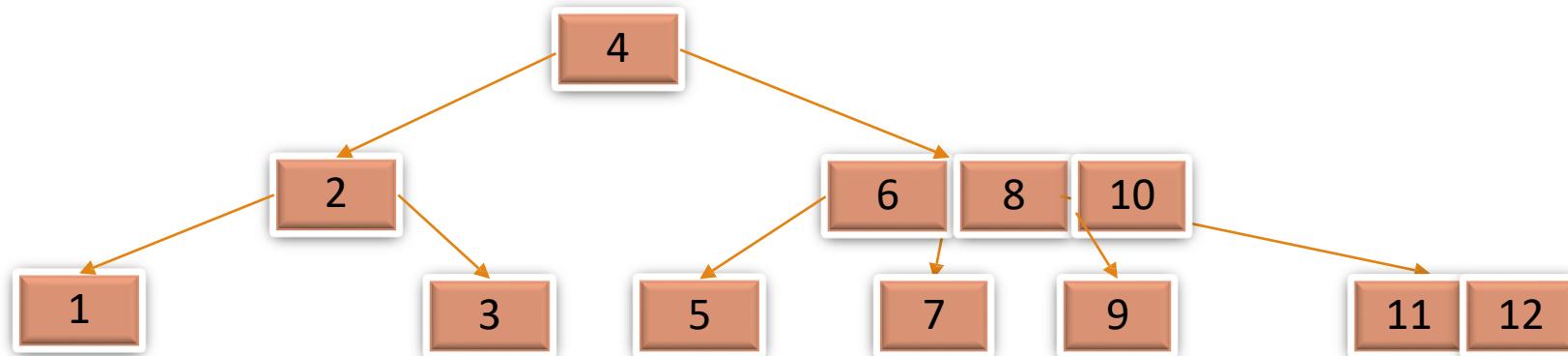
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,**12**,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



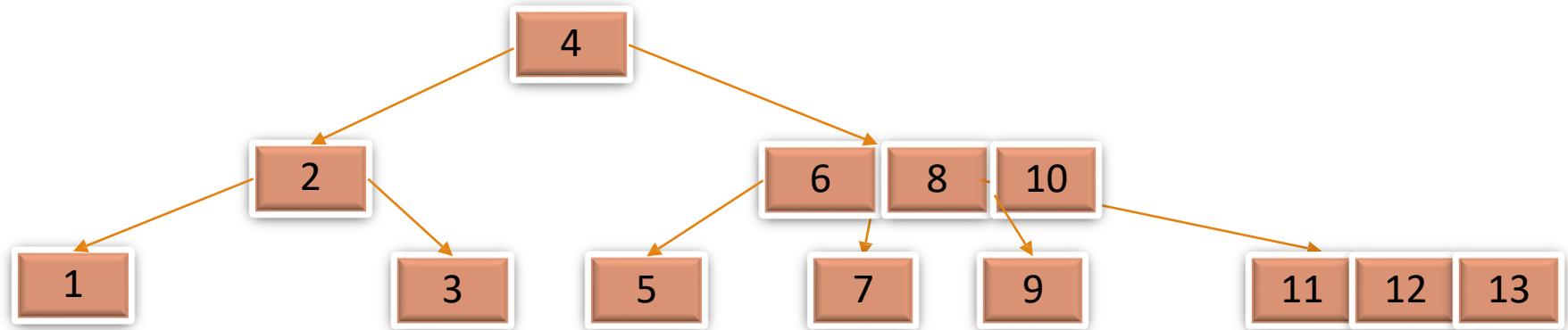
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



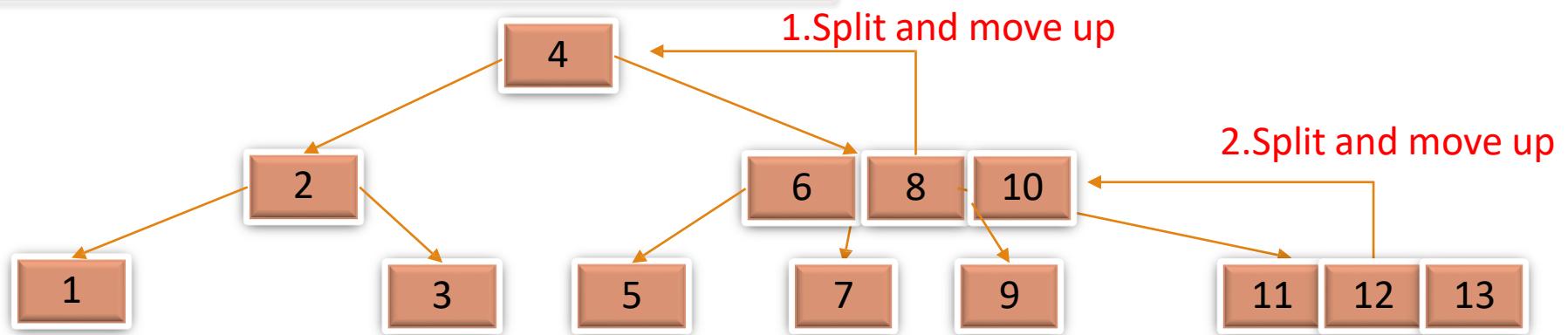
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



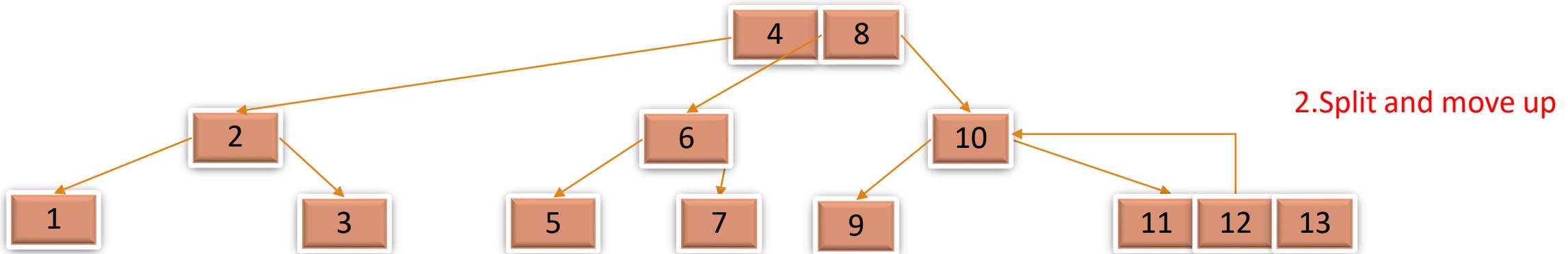
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



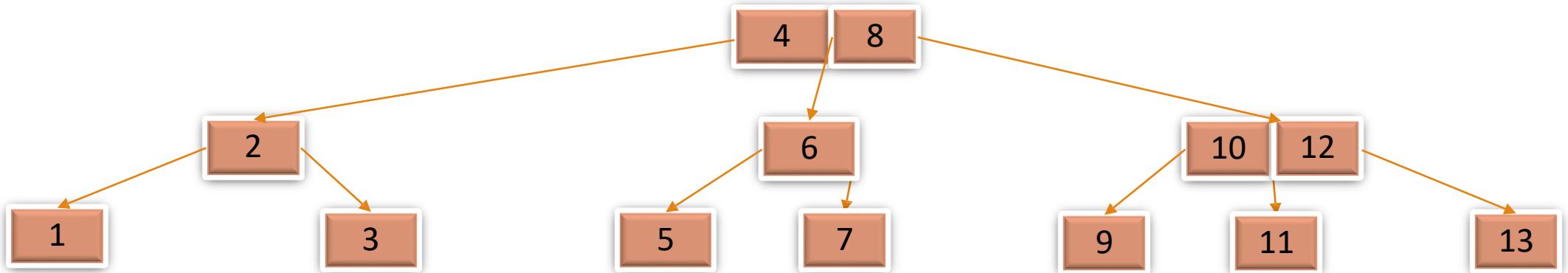
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



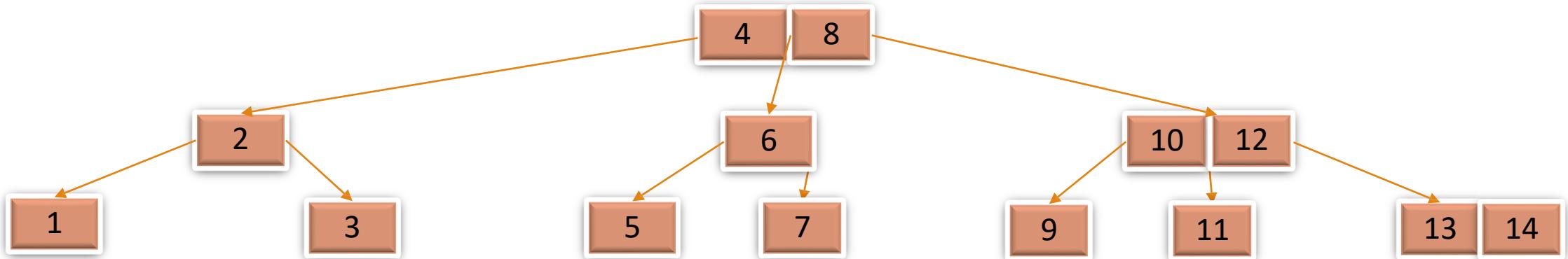
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



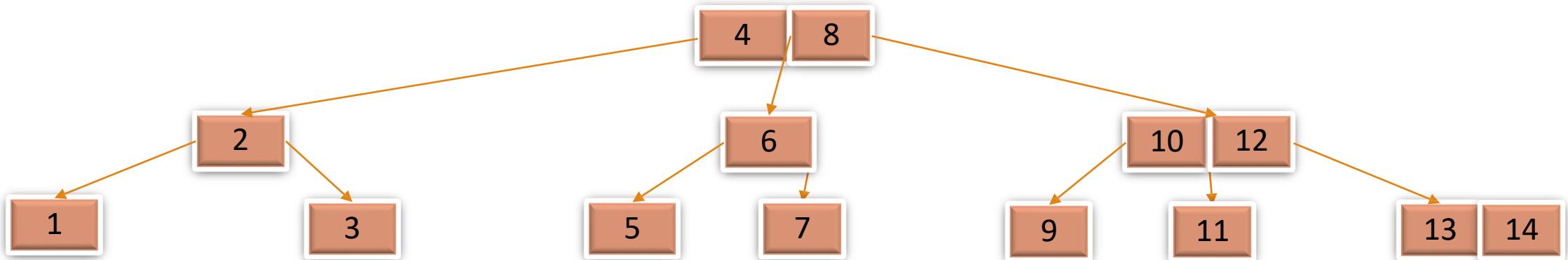
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,**15**

Degree t=2

min. keys=t-1=1

max. keys=2t-1=3



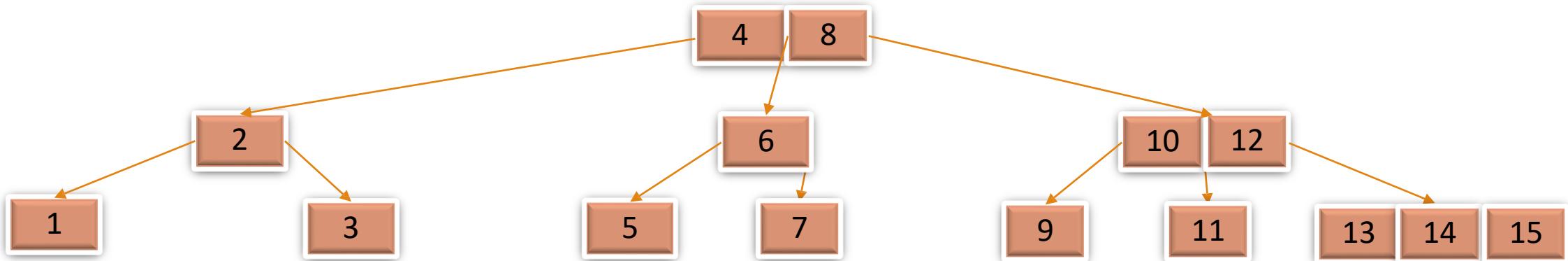
# B-Trees-Creation...

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

Degree t=2

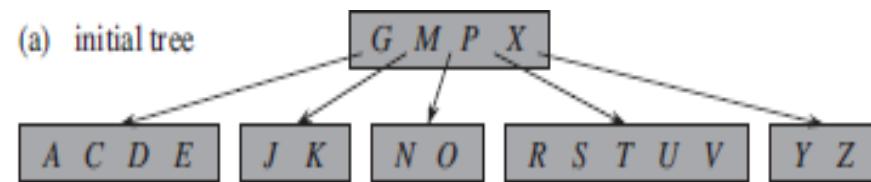
min. keys=t-1=1

max. keys=2t-1=3



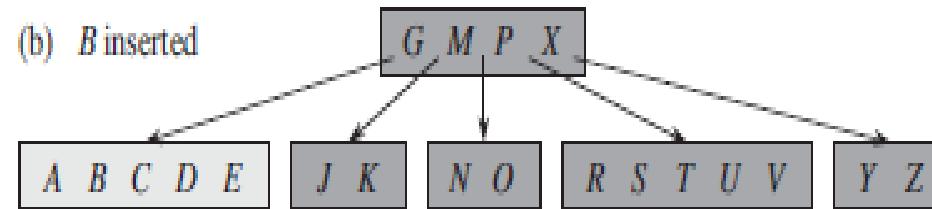
# B-Trees-Insertion

**Minimum degree is t=3.**



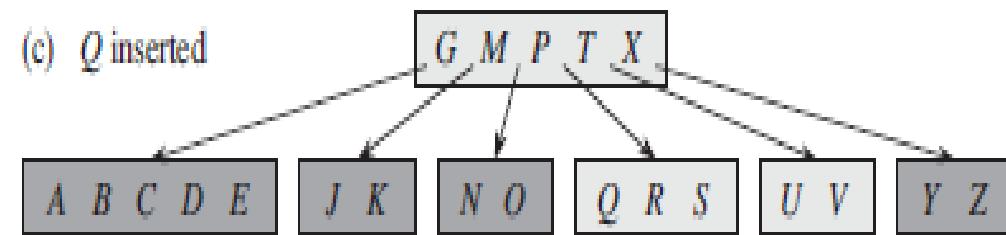
# B-Trees-Insertion...

Minimum degree is  $t=3$ .



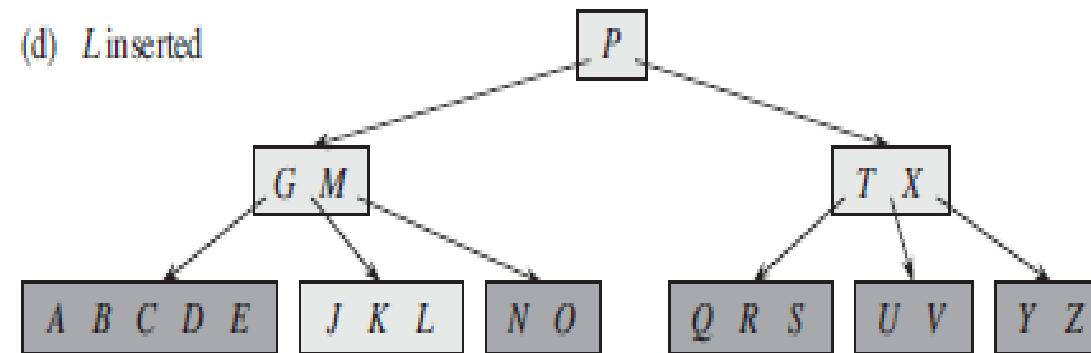
# B-Trees-Insertion...

**Minimum degree is  $t=3$ .**



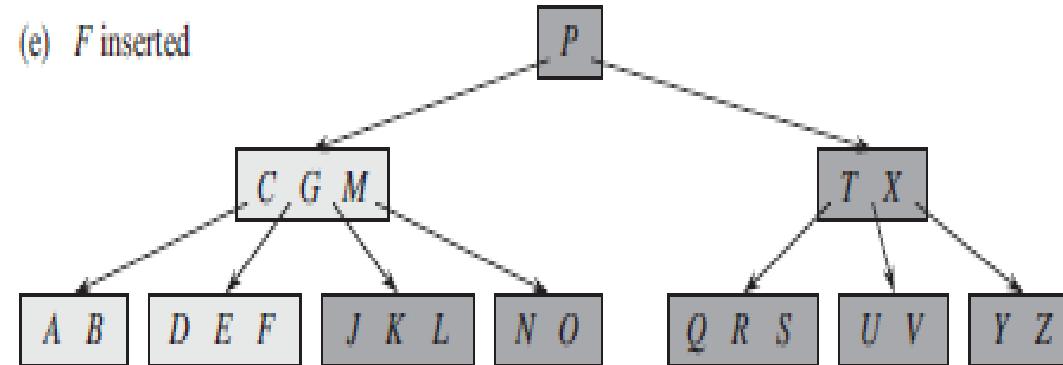
# B-Trees-Insertion...

**Minimum degree is  $t=3$ .**



# B-Trees-Insertion...

Minimum degree is  $t=3$ .





---

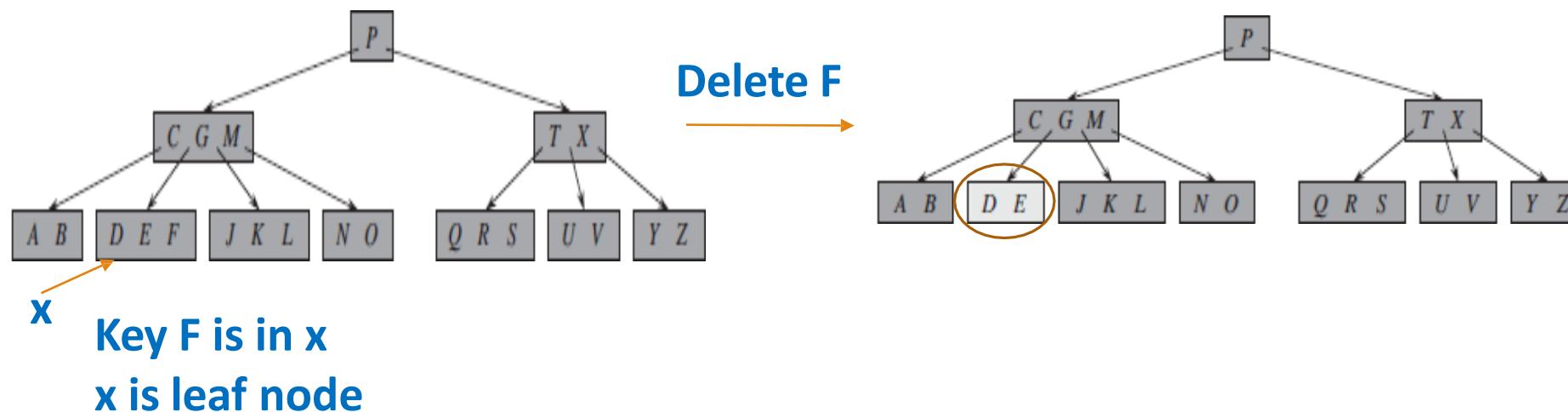
## Design and Analysis of Algorithms

# B-Trees Deletion

# B-Trees-Deletion

**Minimum degree is  $t=3$  Min. keys= $t-1=2$  Max. keys= $2t-1=5$**

**Case 1: If the key is in node x and x is leaf, delete key k from x.**

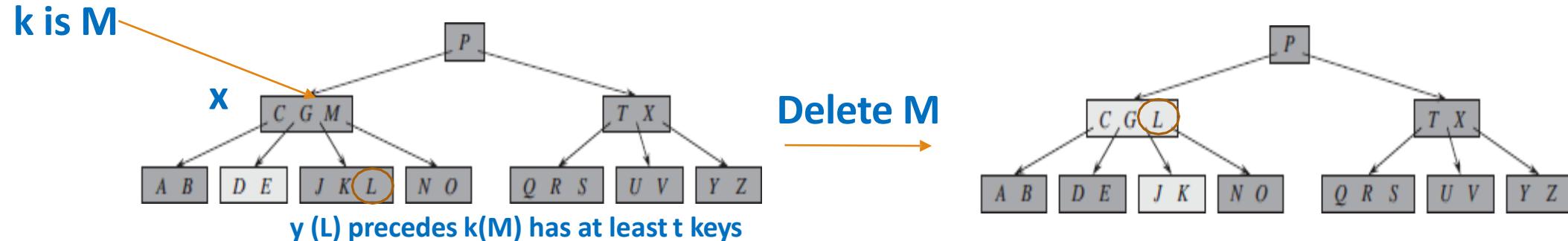


# B-Trees-Deletion

**Minimum degree is  $t=3$ .**

**Case 2:** If the key  $k$  is in node  $x$  and  $x$  is an internal node, do the following:

- a. if the child  $y$  that precede  $k$  in node  $x$  has at least  $t$  keys, then find the predecessor  $k'$  of  $k$  in the subtree rooted at  $y$ . Recursively delete  $k'$  and replace  $k$  by  $k'$  in  $x$ .

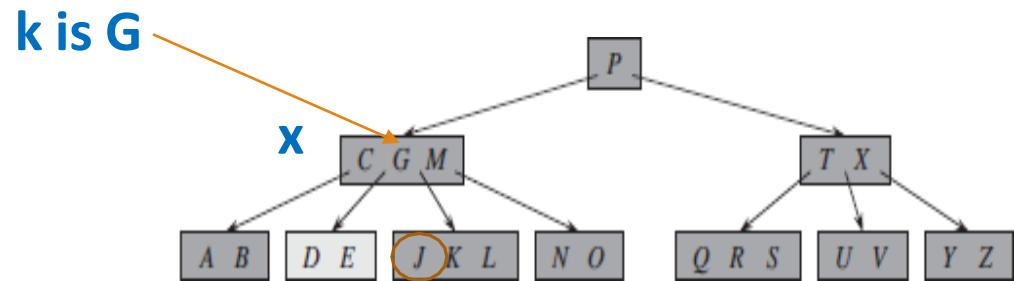


# B-Trees-Deletion

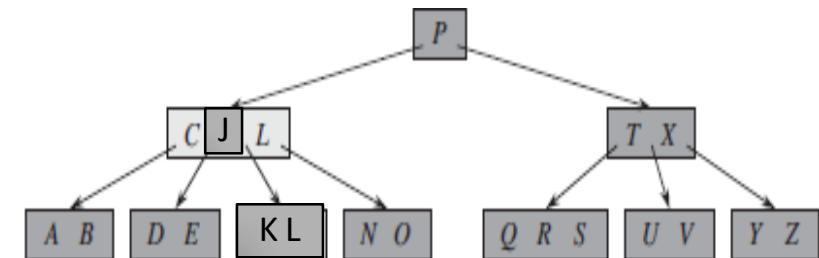
**Minimum degree is  $t=3$ .**

**Case 2:** If the key  $k$  is in node  $x$  and  $x$  is an internal node, do the following:

- b. if  $y$  has fewer than  $t$  keys, then symmetrically, examine the child  $z$  that follows  $k$  in node  $x$ . If  $z$  has at least  $t$  keys, then find the successor  $k'$  of  $k$  in the subtree rooted at  $z$ .
- z. Recursively delete  $k'$  and replace  $k$  by  $k'$  in  $x$ .



**Delete G**



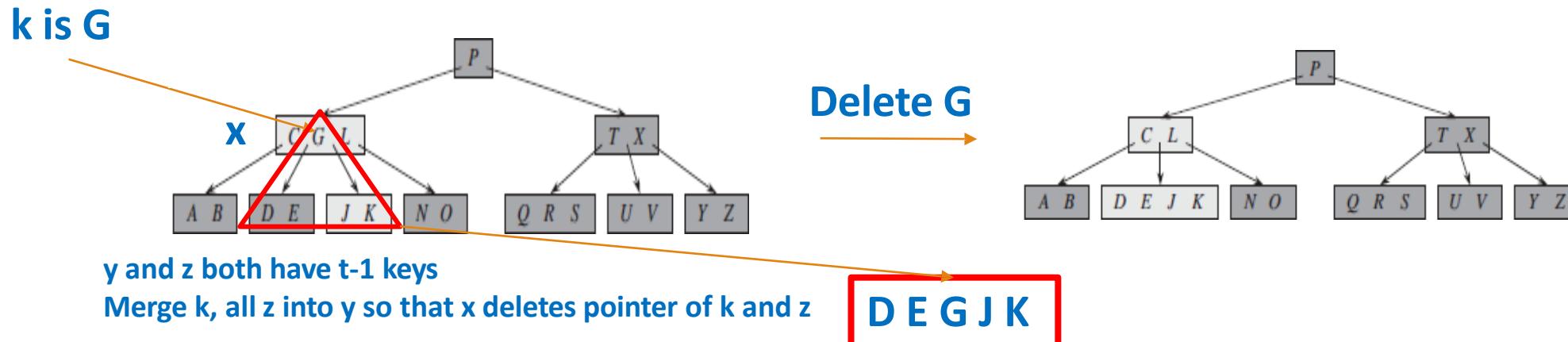
**y (E) precedes k(G) has fewer than  $t$  keys**  
**z (J) follows k(G) in x and z has at least  $t$  keys**  
**k' (J) is successor of k(G)**

# B-Trees-Deletion

**Minimum degree is  $t=3$ .**

**Case 2:** If the key  $k$  is in node  $x$  and  $x$  is an internal node, do the following:

- c. Otherwise, if both the  $y$  and  $z$  have only  $t-1$  keys, merge  $k$  and all of  $z$  into  $y$ , so that  $x$  loses both  $k$  and pointer to  $z$ , and  $y$  now contains  $2t-1$  keys, Then free  $z$  and recursively delete  $k$  from  $y$ .

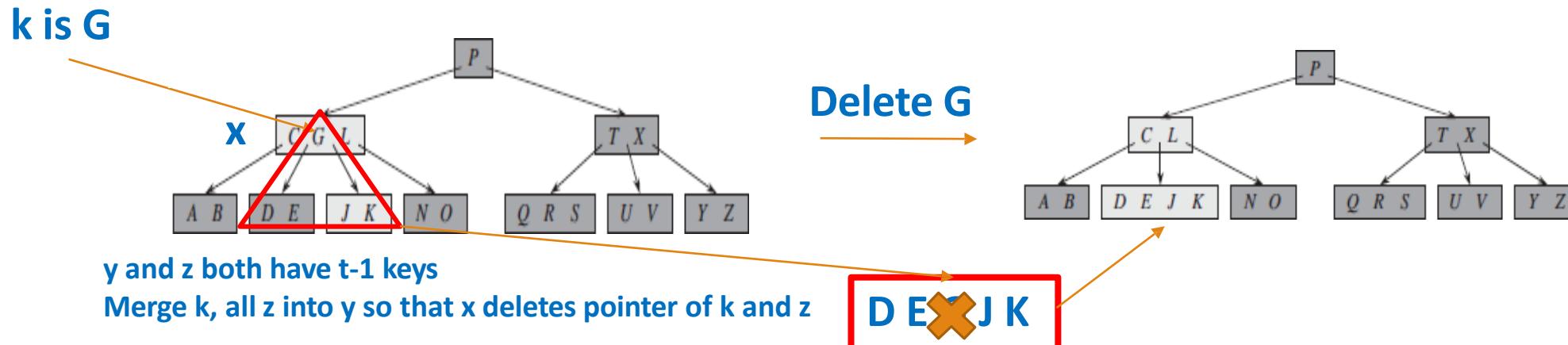


# B-Trees-Deletion

**Minimum degree is  $t=3$ .**

**Case 2:** If the key  $k$  is in node  $x$  and  $x$  is an internal node, do the following:

- c. Otherwise, if both the  $y$  and  $z$  have only  $t-1$  keys, merge  $k$  and all of  $z$  into  $y$ , so that  $x$  loses both  $k$  and pointer to  $z$ , and  $y$  now contains  $2t-1$  keys, Then free  $z$  and recursively delete  $k$  from  $y$ .



# B-Trees-Deletion

**Minimum degree is t=3.**

**Case 3:** If the key k is not present in internal node x, determine the root  $x.c_i$  of the appropriate subtree that must contain k, if k is in the tree at all. If  $x.c_i$  has only  $t-1$  keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least  $t$  keys. Then finish by recursing on the appropriate child of x.

**b. if  $x.c_i$  and both of  $x.c_i$ 's immediate siblings have  $t-1$  keys, merge  $x.c_i$  with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.**

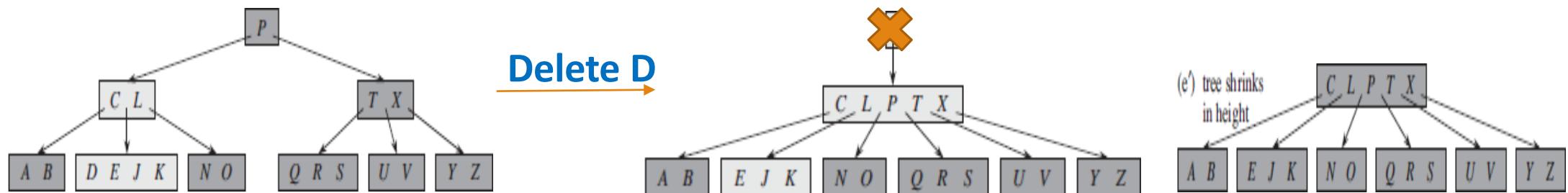


# B-Trees-Deletion

**Minimum degree is t=3.**

**Case 3:** If the key k is not present in internal node x, determine the root  $x.c_i$  of the appropriate subtree that must contain k, if k is in the tree at all. If  $x.c_i$  has only  $t-1$  keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least  $t$  keys. Then finish by recursing on the appropriate child of x.

**b. if  $x.c_i$  and both of  $x.c_i$ 's immediate siblings have  $t-1$  keys, merge  $x.c_i$  with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.**



# B-Trees-Deletion

**Minimum degree is  $t=3$ .**

**Case 3:** If the key  $k$  is not present in internal node  $x$ , determine the root  $x.c_i$  of the appropriate subtree that must contain  $k$ , if  $k$  is in the tree at all. If  $x.c_i$  has only  $t-1$  keys, execute step 3a or 3b as necessary to guarantee that we descend to a node containing at least  $t$  keys. Then finish by recursing on the appropriate child of  $x$ .

- a. if  $x.c_i$  has only  $t-1$  keys but has an immediate sibling with at least  $t$  keys, give  $x.c_i$  an extra key by moving a key from  $x$  down into  $x.c_i$ , moving a key from  $x.c_i$ 's immediate left or right sibling up into  $x$ , and moving appropriate child pointer from the sibling into  $x.c_i$ .



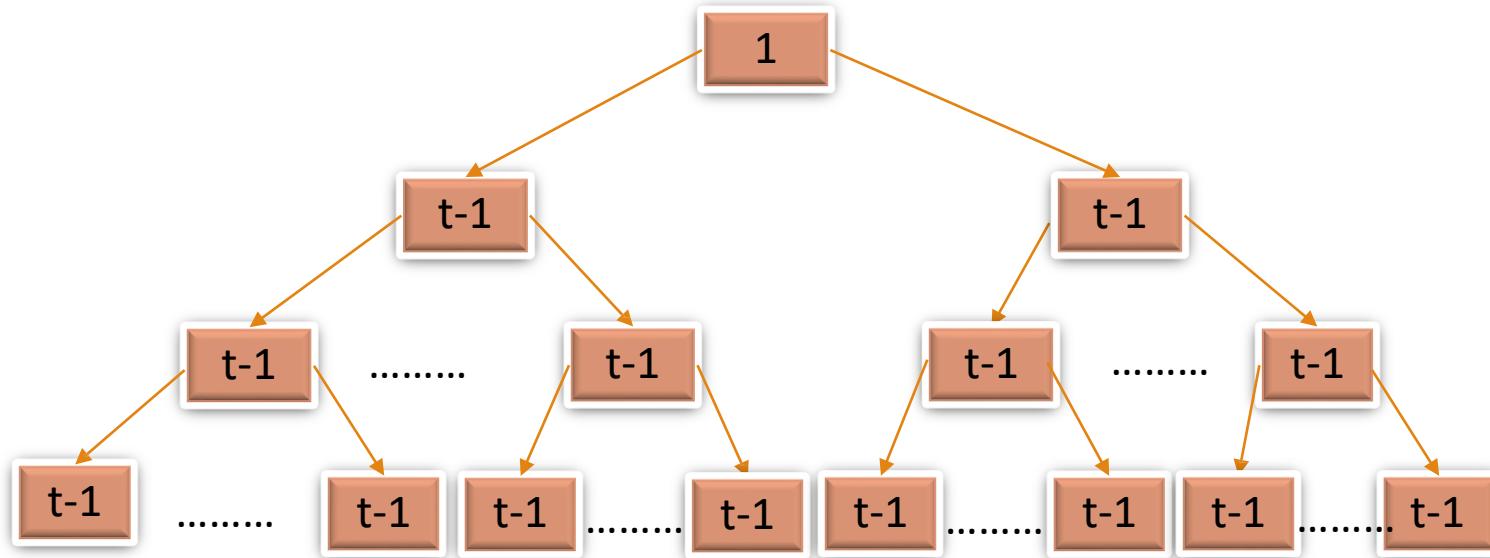


---

## Design and Analysis of Algorithms

# B-Trees-Height

# B-Trees-height...

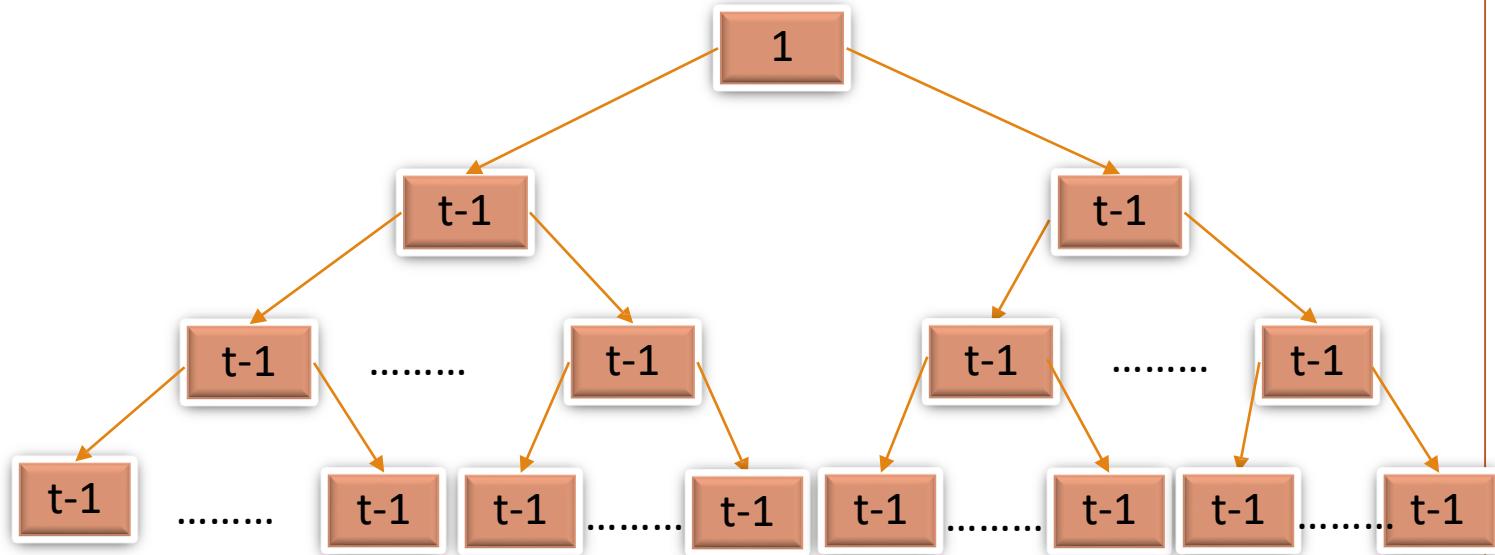


Degree=  $t$  Min. Keys= $t-1$  Max. Keys= $2t-1$

Depth      Number of nodes

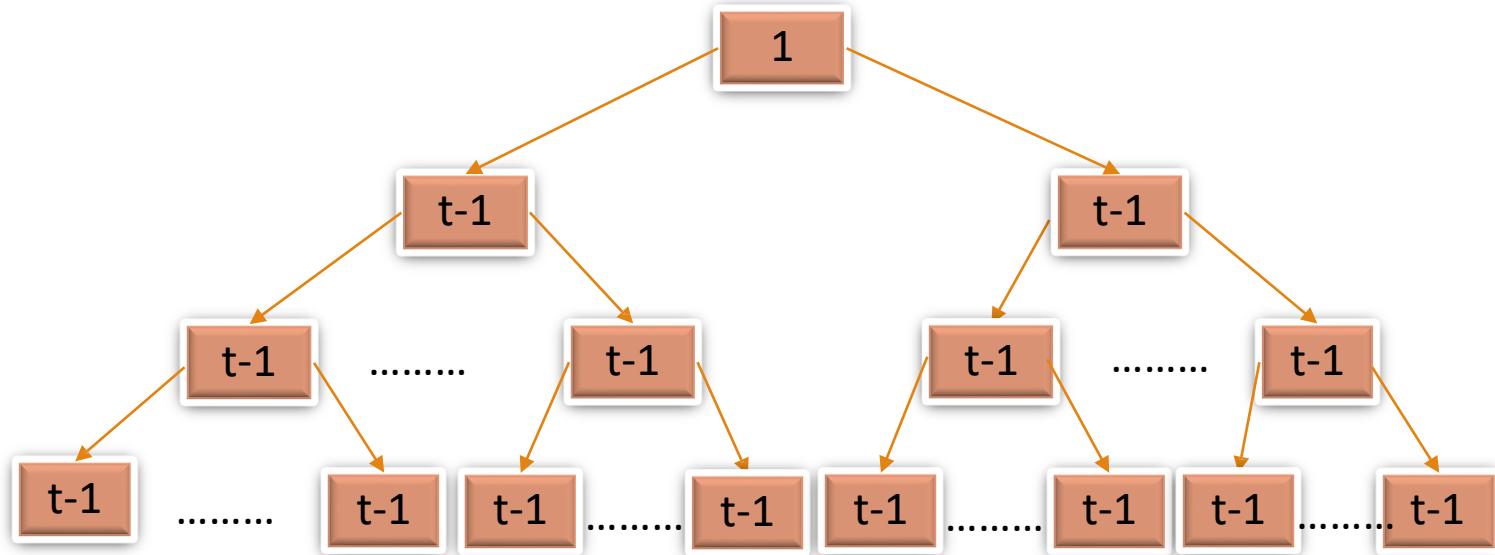
Depth	Number of nodes
0	1
1	2
2	$2t$
3	$2t^2$

# B-Trees-height...



Depth	Number of nodes
0	1
1	$2 \cdot t^0$
2	$2 \cdot t^1$
3	$2 \cdot t^2$
.	.
.	.
.	.
$h$	$2 \cdot t^{h-1}$

# B-Trees-height...



$$\text{Total nodes} = 1 + (2 \cdot t^0 + 2 \cdot t^1 + 2 \cdot t^2 + \dots + 2 \cdot t^{h-1})$$

If a B-tree has height  $h$ , the number of its nodes is minimized when the root contains one key and all other nodes contain  $t-1$  keys. In this case, there are 2 nodes at depth 1,  $2t$  nodes at depth 2,  $2t^2$  at depth 3 and so on, until at depth  $h$  there are  $2t^{h-1}$  nodes.

# B-Trees-height...

$$n \geq 1 + (t-1) (2.t^0 + 2.t^1 + 2.t^2 + \dots + 2.t^{h-1})$$

$$n \geq 1 + (t-1).2.(t^0 + t^1 + t^2 + \dots + t^{h-1})$$

$$n \geq 1 + (t-1).2.\sigma_i^h = t^{i-1}$$

$$n \geq 1 + 2.(t-1).\frac{t^h - 1}{t - 1}$$

$$n \geq 1 + 2.(t^h - 1)$$

$$n \geq 1 + 2.t^h - 2$$

$$n \geq 2.t^h - 1$$

$$n+1 \geq 2.t^h$$

$t^h \leq \frac{n+1}{2}$  Take log on both sides:

$$\log(t^h) \leq \log\left(\frac{n+1}{2}\right)$$

$$h \log t \leq \log\left(\frac{n+1}{2}\right)$$

Since every node except the root have  $(t-1)$  keys. So we multiply the expression by  $(t-1)$

A geometric series with common ratio  $r$  and first term  $a$

**nth term =  $a \times r^{n-1}$**

**The sum to n terms**

$$S_n = \frac{a(r^n - 1)}{r - 1}$$

# B-Trees-height...

$$h \log t \leq \log\left(\frac{n+1}{2}\right)$$

$$h \leq \frac{\log\left(\frac{n+1}{2}\right)}{\log t}$$

$$h \leq \log_t \frac{n+1}{2}$$

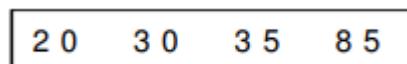
# Example 1

Create B-Tree of order 5 from the following list of data items:

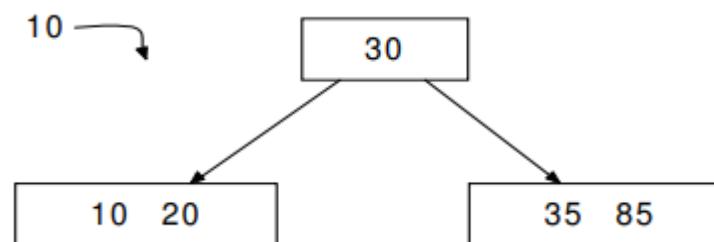
20, 30, 35, 85, 10, 55, 60,

For the given list of elements as 20 30 35 85 10 55 60 25. The B-Tree of order 5 will be created as follows

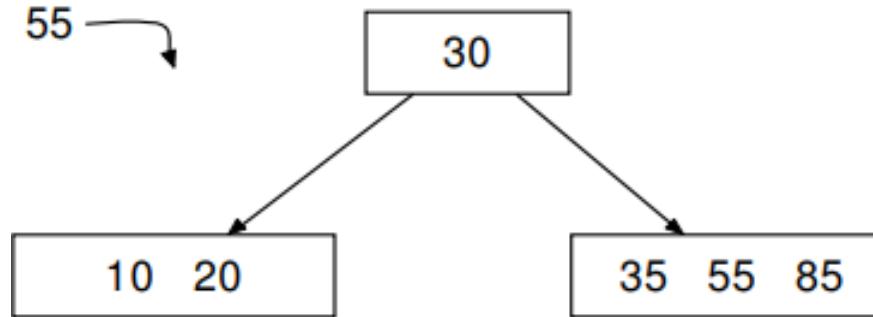
**Step 1: Insert 20, 30, 35 and 85**



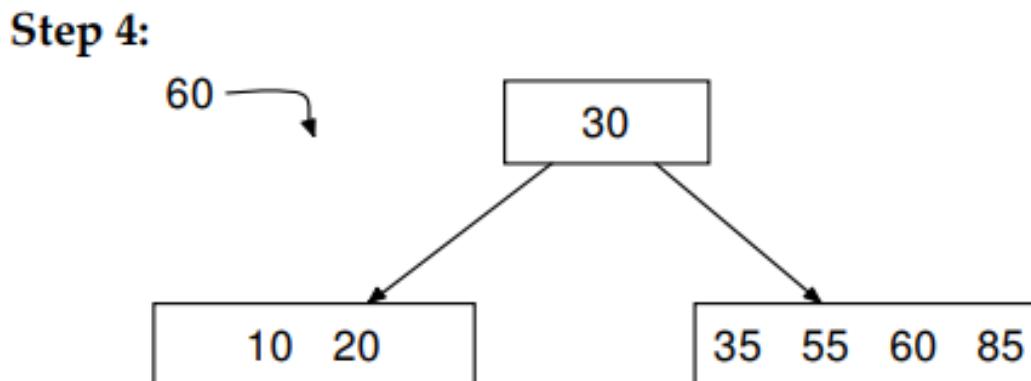
**Step 2: Insert 10**



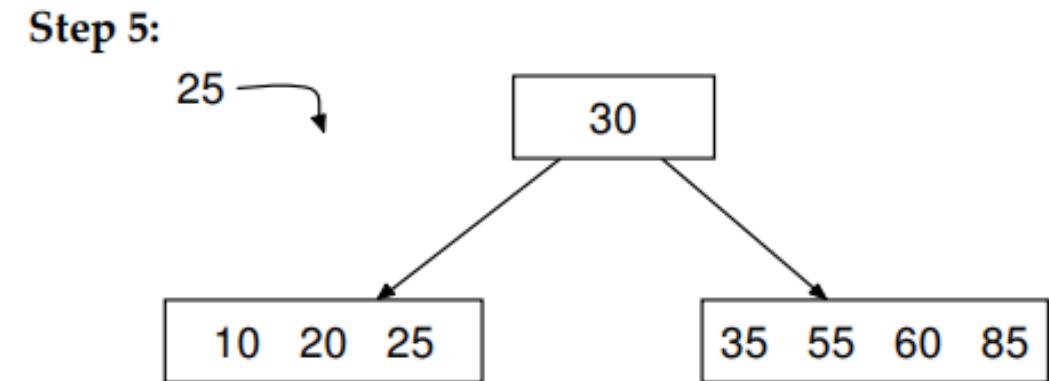
### Step 3: Insert 55



### Step 4: Insert 60



### Step 5: Insert 25



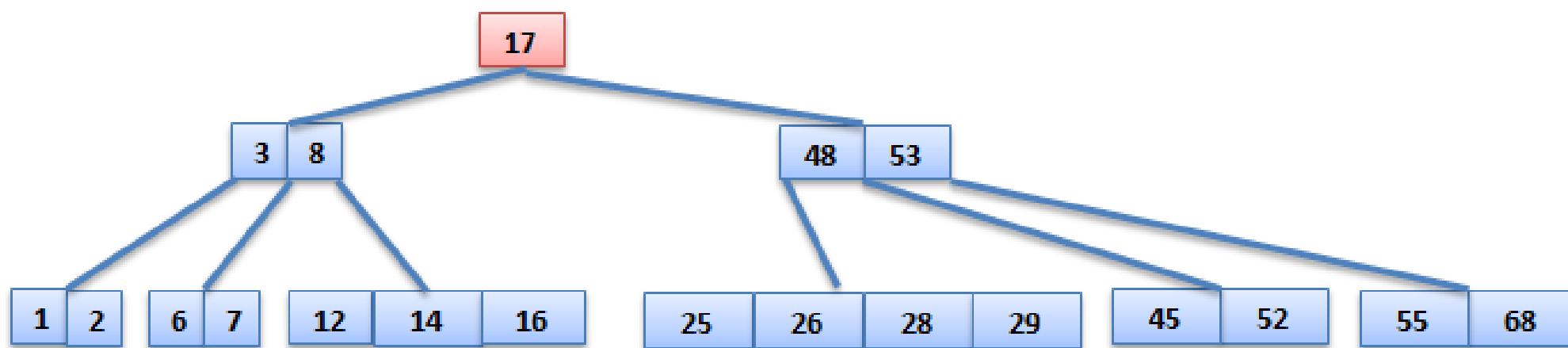
**Final B-tree created after inserting all the given elements.**

# Example 2

## Example 2

key : 1,12,8,2,25,6,14,28,17,7,52,16,48,68,3,26,29,53,55,45,67.

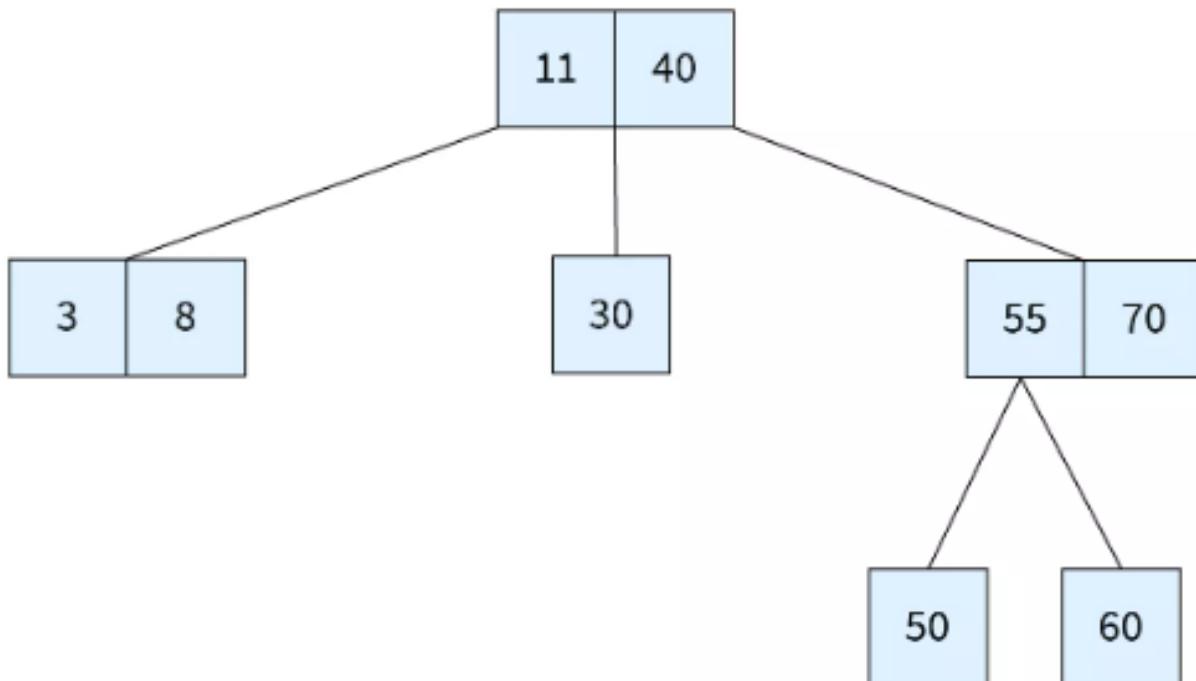
Order = 5



Final B-tree created after inserting all the given elements.

# Example 3

Q. Is the tree given in the following image a B Tree?



**Ans:** No. Since the leaf nodes are not at the same level, we can say that this is not a B Tree.

**Question:** Consider a B+-tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

- (A) 1
- (B) 2
- (C) 3
- (D) 4

**Ans: B**

**Question:** If a node has K children in a B tree, then the node contains exactly \_\_\_\_\_ keys

- a)  $K^2$
- B)  $K-1$
- C)  $K+1$
- D)  $K^{1/2}$

**Ans: B**

**Question.** Which of the following is the most widely used external memory data structure?

- a) AVL tree
- b) B-tree
- c) Red-black tree
- d) Both AVL tree and Red-black tree

**Answer: b**

Explanation: In external memory, the data is transferred in form of blocks. These blocks have data valued and pointers. And B-tree can hold both the data values and pointers. So B-tree is used as an external memory data structure.

Question: B-tree of order  $n$  is a order- $n$  multiway tree in which each non-root node contains \_\_\_\_\_

- a) at most  $(n - 1)/2$  keys
- b) exact  $(n - 1)/2$  keys
- c) at least  $2n$  keys
- d) at least  $(n - 1)/2$  keys

**Answer: d**

Explanation: A non-root node in a B-tree of order  $n$  contains at least  $(n - 1)/2$  keys. And contains a maximum of  $(n - 1)$  keys and  $n$  sons.

Question: A B-tree of order 4 and of height 3 will have a maximum of \_\_\_\_\_ keys.

- a) 255
- b) 63
- c) 127
- d) 188

**Answer: a**

Explanation: A B-tree of order  $m$  of height  $h$  will have the maximum number of keys when all nodes are completely filled. So, the B-tree will have  $n = (m^{h+1} - 1)$  keys in this situation. So, required number of maximum keys =  $4^{3+1} - 1 = 256 - 1 = 255$ .

“Thank you”

*Any Questions ?*

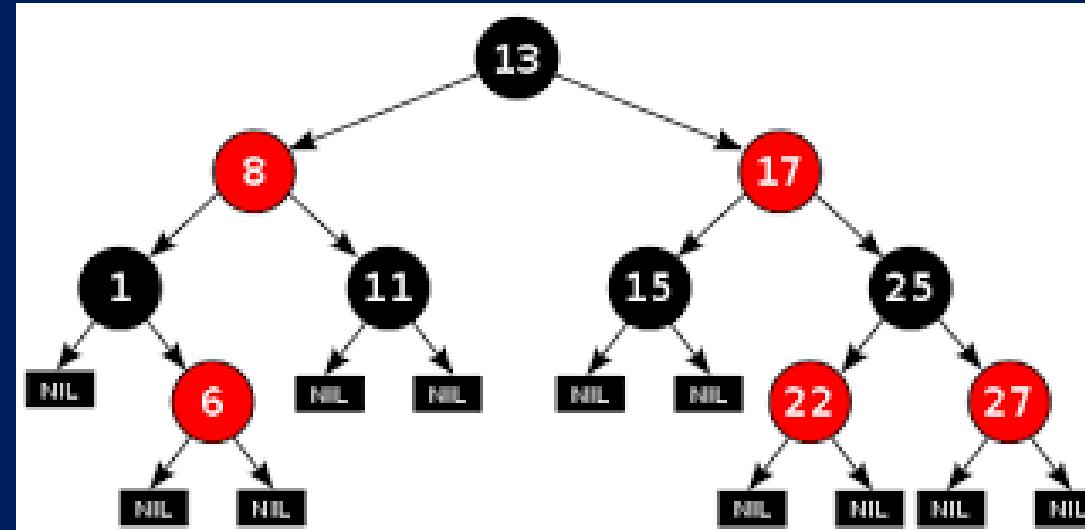


**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

# Chapter 7: RED BLACK TREE

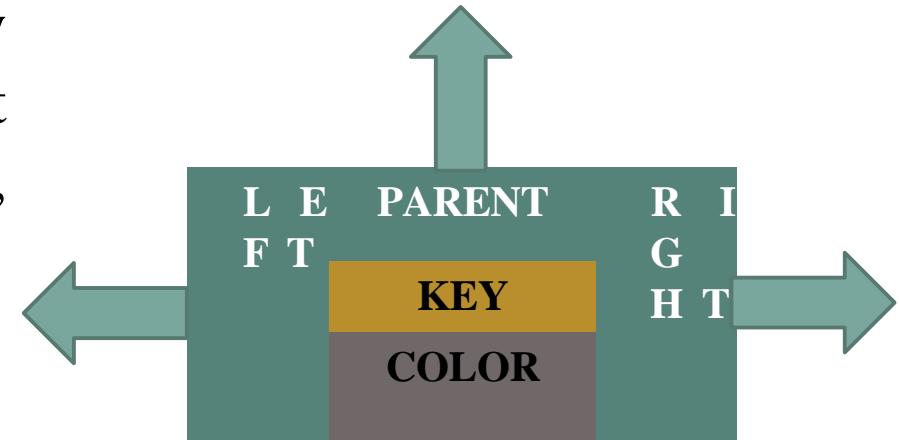


Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

# RED-BLACK TREE

- A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either **RED** or **BLACK**.
- By constraining the way nodes can be colored on any path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.
- Each node of the tree now contains the fields color, key, left, right, and p.
- If a child or the parent of a node does not exist, the corresponding pointer field of the node contains the value NIL.
- Each RBT node contains fields: left, right, parent, color and key.



**RBT NODE STRUCTURE**

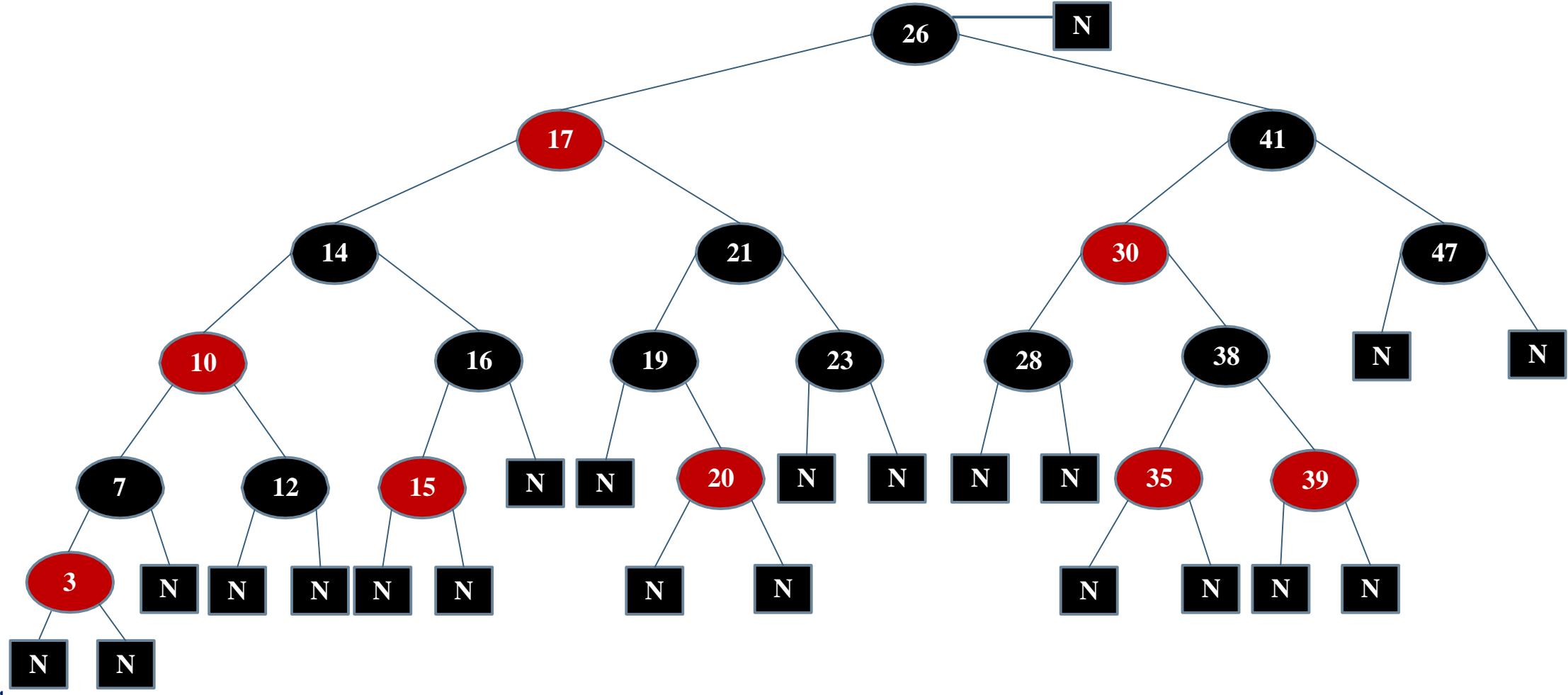


# Properties of Red Black Tree

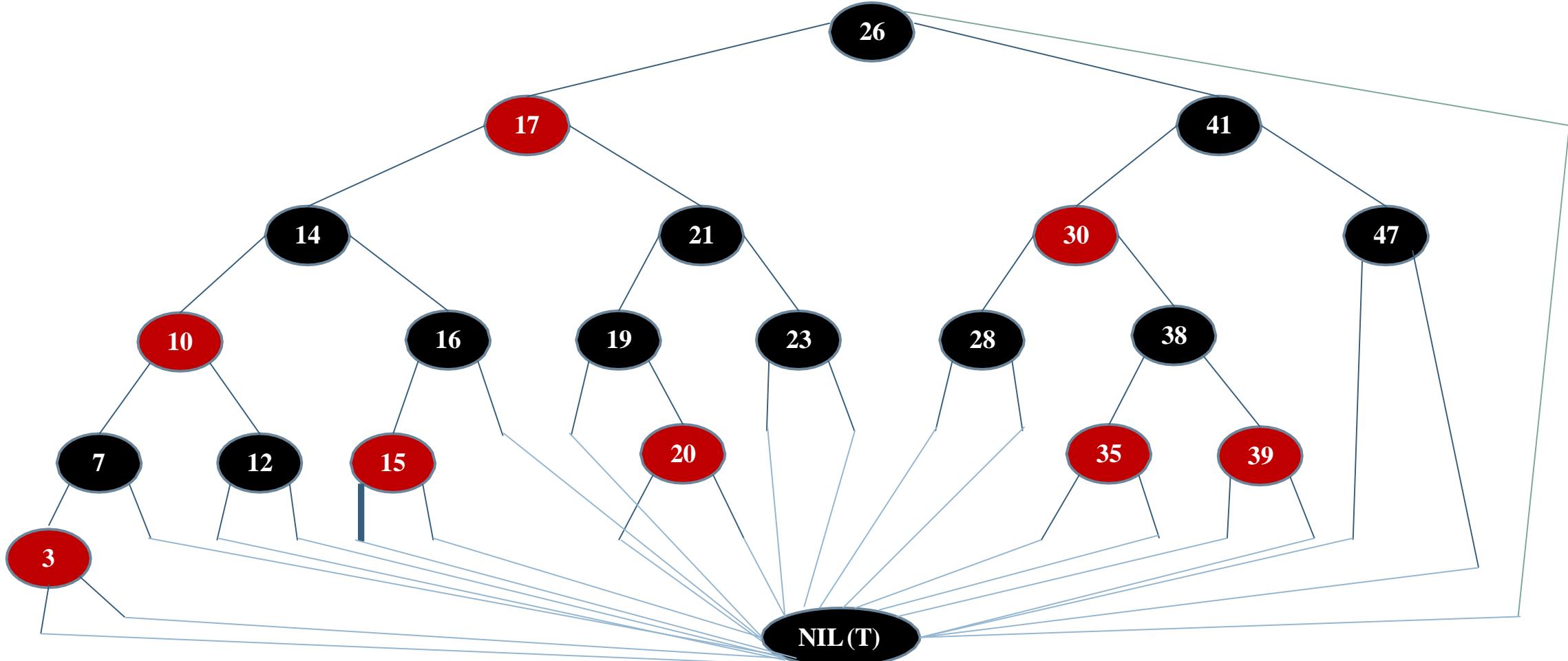
(\*\*Satisfy the binary search tree property\*\*)

1. Every node is either **red** or **black**
2. The root is **black**
3. Every leaf (NIL) is **black**
4. If a node is **red**, then both its children are **black**
  - No two consecutive red nodes on a simple path from the root to a leaf
5. For each node, all paths from that node to descendant leaves contain the same number of black nodes

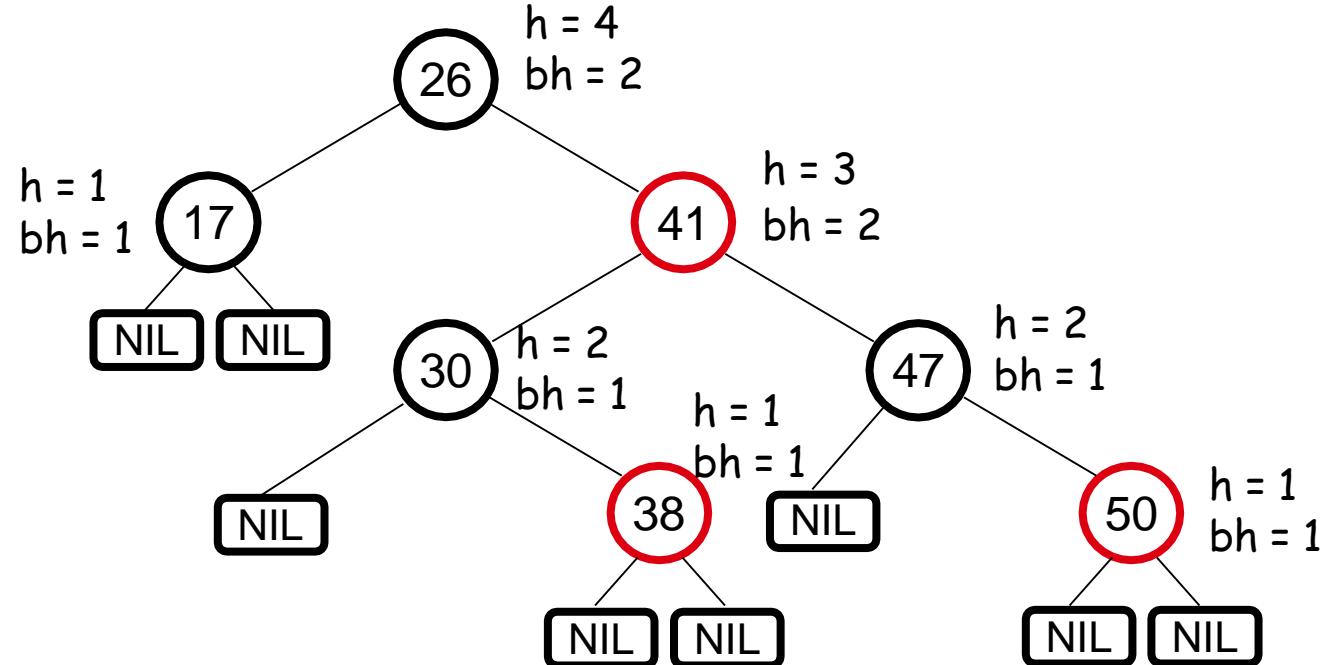
# RED-BLACK TREE



# RED-BLACK TREE



# RED-BLACK TREE



- **Height of a node:** the number of edges in the **longest** path to a leaf
- **Black-height** of a node  $x$ :  $bh(x)$  is the number of black nodes (including NIL) on the path from  $x$  to a leaf,  
not counting  $x$

“Thank you”

*Any Questions ?*

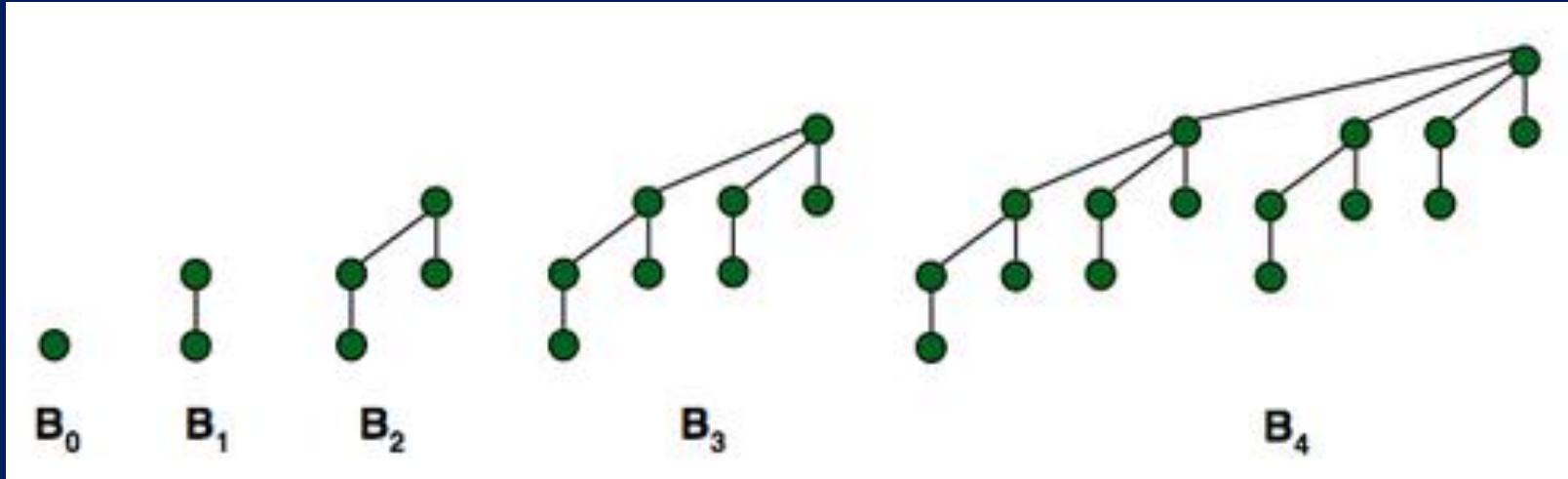


**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**

# DESIGN & ANALYSIS OF ALGORITHM

## (BCSC0012)

## Chapter 8: Binomial Heap



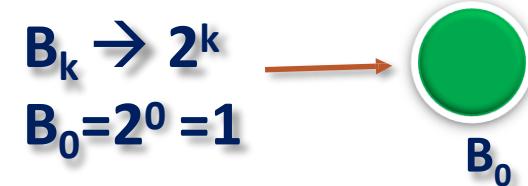
Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

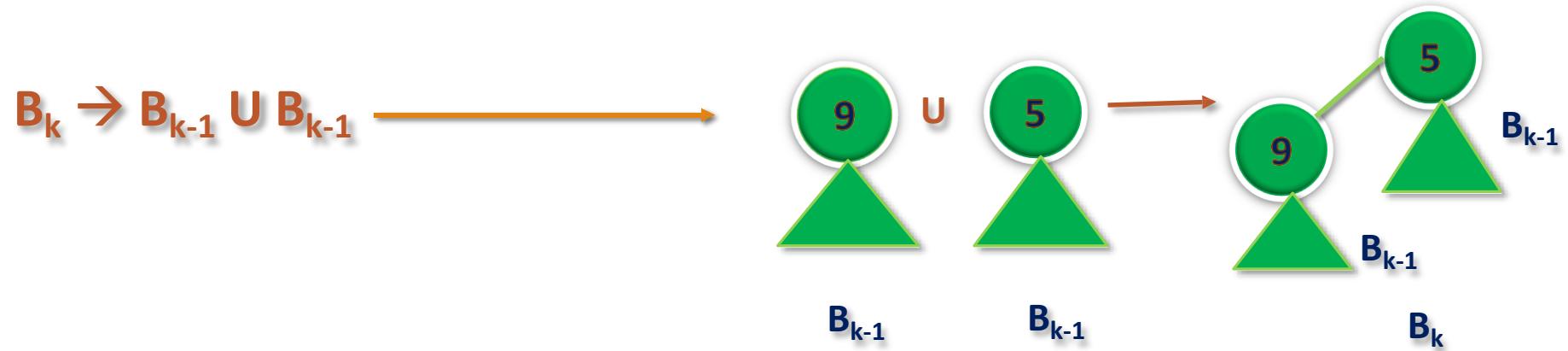
# Binomial Trees...

The binomial tree  $B_k$  is an ordered tree.

A Binomial Tree of order 0 has 1 node.



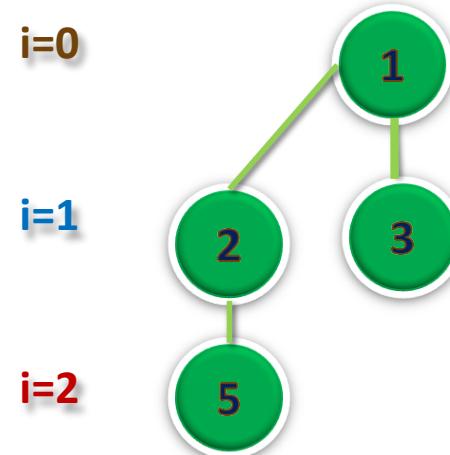
A Binomial Tree of order  $k$  can be constructed by taking two binomial trees of order  $k-1$  and making one as leftmost child or other.



# Binomial Trees...

**A Binomial Tree of order k has following properties:**

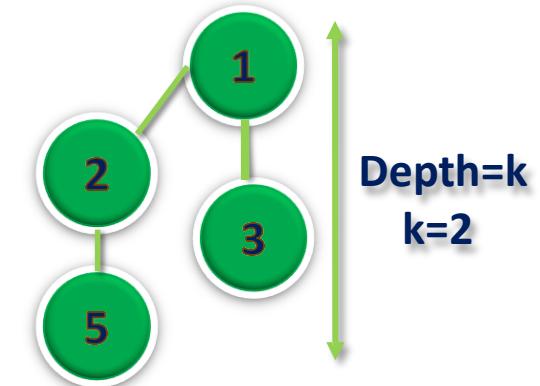
1. It has exactly  $2^k$  nodes.
2. It has depth as k.
3. There are exactly  ${}^k C_i$  nodes at depth i for  $i = 0, 1, \dots, k$ .



i=0                      depth i=0, k=2 Nodes= ${}^k C_i = {}^2 C_0 = 2! / 0!. (2-0)!$   
                             depth=2/1.2=2/2=1 Node

i=1                      depth i=1, k=2 Nodes= ${}^k C_i = {}^2 C_1 = 2! / 1!. (2-1)!$   
                             depth=2/1.1=2/1=2 Nodes

i=2                      depth i=2, k=2 Nodes= ${}^k C_i = {}^2 C_2 = 2! / 2!. (2-2)!$   
                             depth=2/2.1=2/2=1 Node



**Binomial Tree ( $B_k$ ) of order k**  
 **$k=2$ ,  $B_2$  has  $2^2 = 4$  nodes**

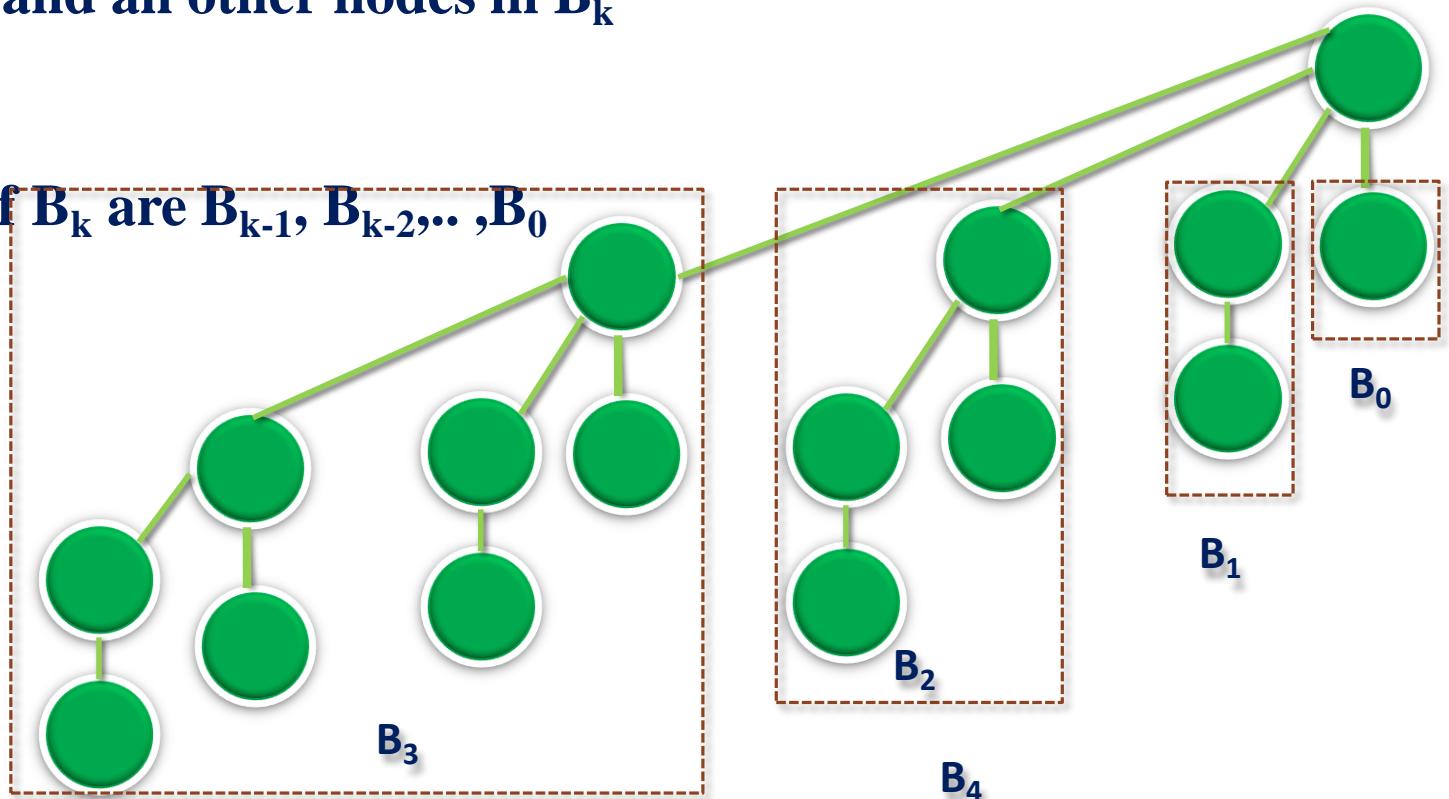
$${}^k C_i = \frac{k!}{i!. (k-i)!}$$

# Binomial Trees...

**A Binomial Tree of order k has following properties:**

**4. The root of  $B_k$  has degree k and all other nodes in  $B_k$  has degree smaller than k.**

**5. If  $k \geq 1$ , then the children of  $B_k$  are  $B_{k-1}, B_{k-2}, \dots, B_0$  from left to right.**



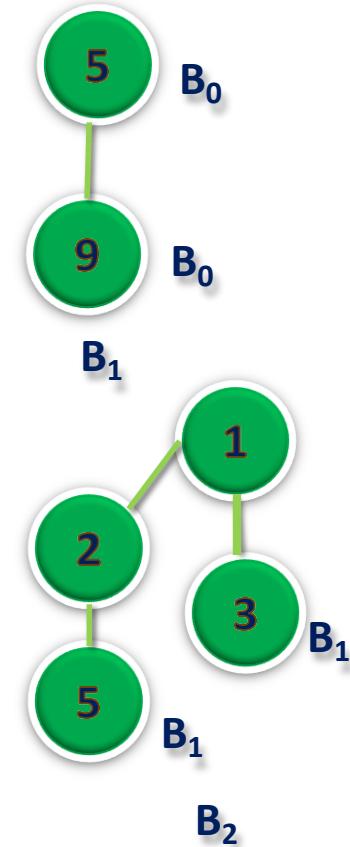
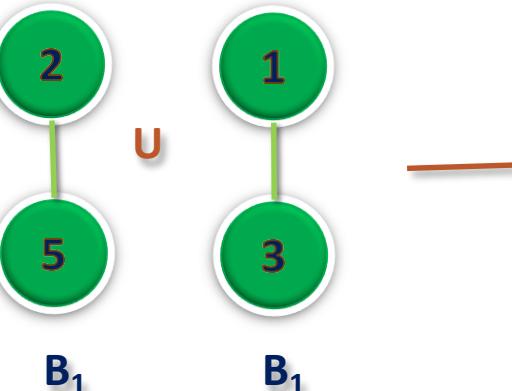
# Binomial Trees...

**A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1 and making one as leftmost child or other.**

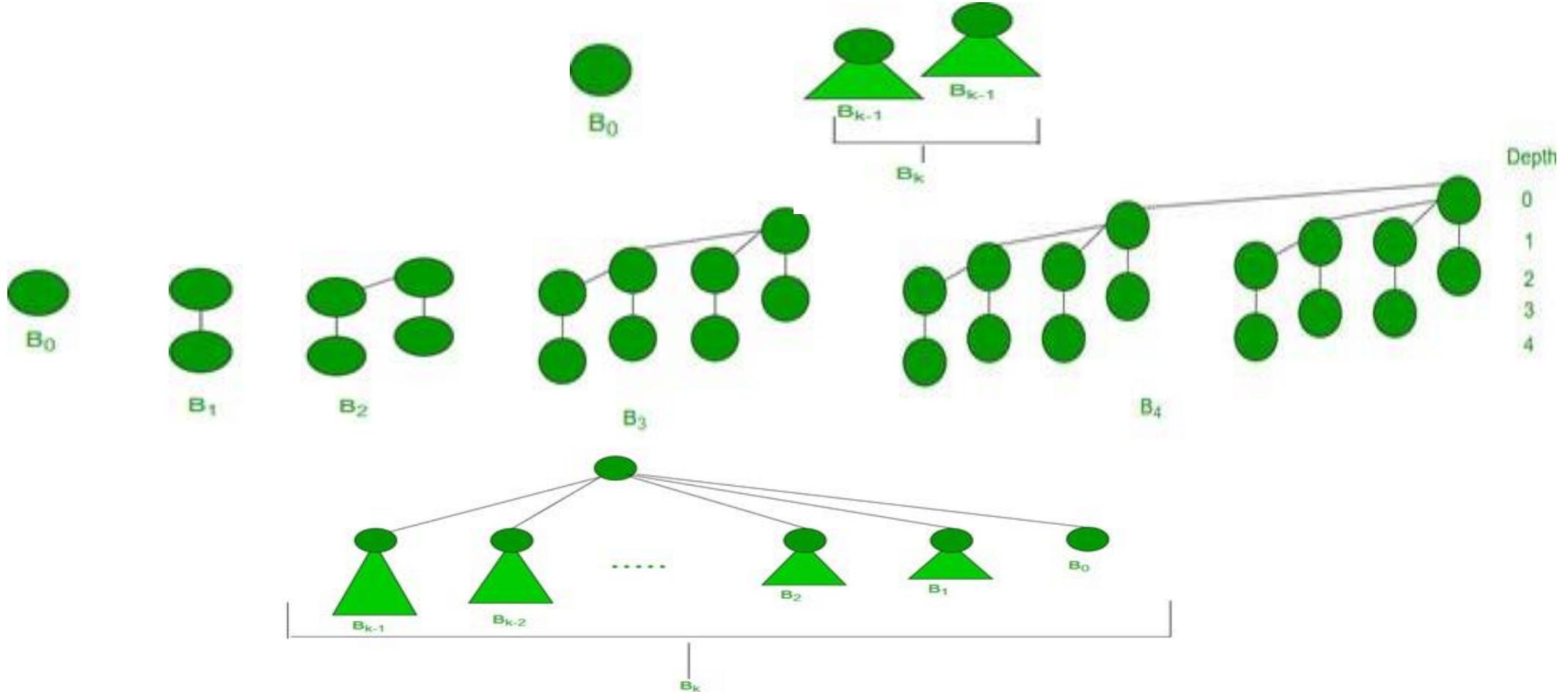
$$B_1 \rightarrow B_0 \cup B_0$$



$$B_2 \rightarrow B_1 \cup B_1$$



# Binomial Trees...





## Design and Analysis of Algorithms

# Binomial Heap - Merge Operation

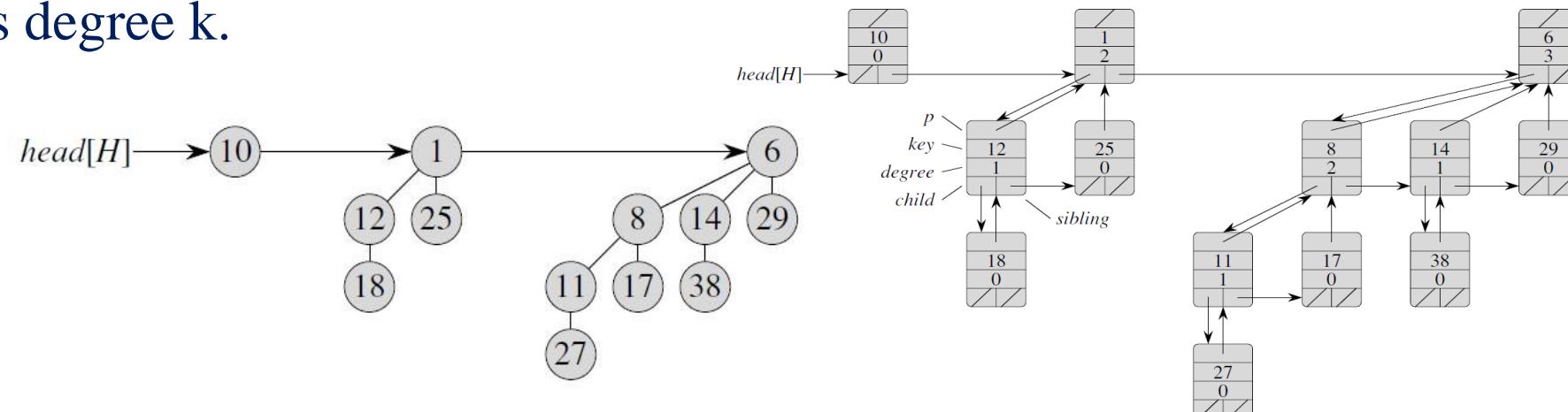
---

# Binomial Heap-characteristics

**A Binomial Heap is a collection of Binomial Trees.**

1. Each node has a key.
2. Each binomial tree in  $H$  obeys the min-heap property.
3. For any non-negative integer  $k$ , there is at-most one binomial tree in  $H$  whose root has degree  $k$ .

Parent
Key
Degree
Left Child
Sibling



# Operations on Binomial-Heap

1. **MAKE-HEAP()** creates and returns a new heap containing no elements.
2. **INSERT( $H, x$ )** inserts node  $x$ , whose *key* field has already been filled in, into heap  $H$ .
3. **MINIMUM( $H$ )** returns a pointer to the node in heap  $H$  whose key is minimum.
4. **EXTRACT-MIN( $H$ )** deletes the node from heap  $H$  whose key is minimum, returning a pointer to the node.
5. **UNION( $H_1, H_2$ )** creates and returns a new heap that contains all the nodes of heaps  $H_1$  and  $H_2$ . Heaps  $H_1$  and  $H_2$  are "destroyed" by this operation. [Takes  $O(\log n)$  ]
6. **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.
7. **DELETE( $H, x$ )** deletes node  $x$  from heap  $H$ .

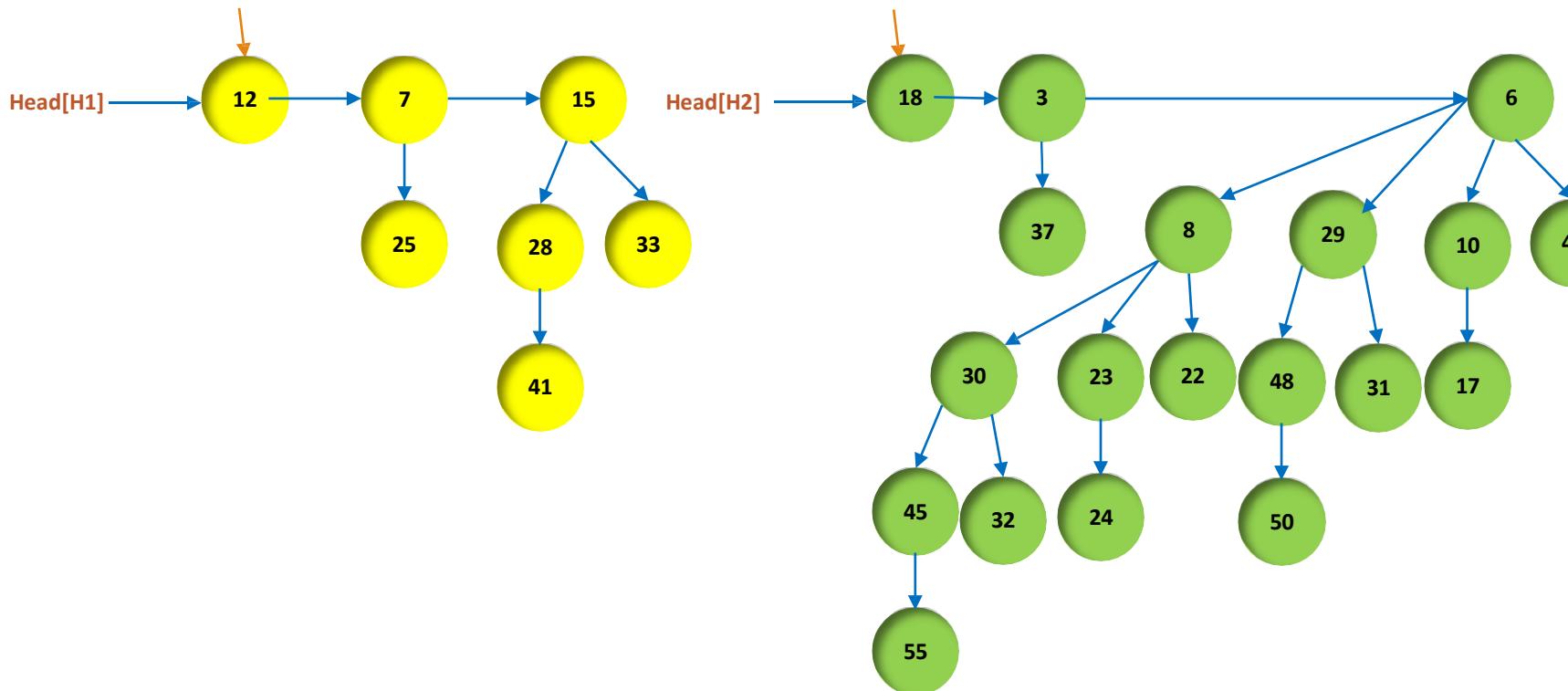
# Binomial Heap...

Application of Binary Heap is as implement priority queue.

Binomial Heap is an extension of Binary Heap that provides faster union or merge operation together with other operations provided by Binary Heap.

Procedure	Binary Heap(worst case)	Binomial Heap(worst case)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\log n)$	$\Theta(\log n)$
FIND MINIMUM	$\Theta(1)$	$\Theta(\log n)$
EXTRACT-MIN	$\Theta(\log n)$	$\Theta(\log n)$
UNION	$\Theta(n)$	$\Theta(\log n)$
DECREASE-KEY	$\Theta(\log n)$	$\Theta(\log n)$
DELETE	$\Theta(\log n)$	$\Theta(\log n)$

# Binomial Merge Operation...



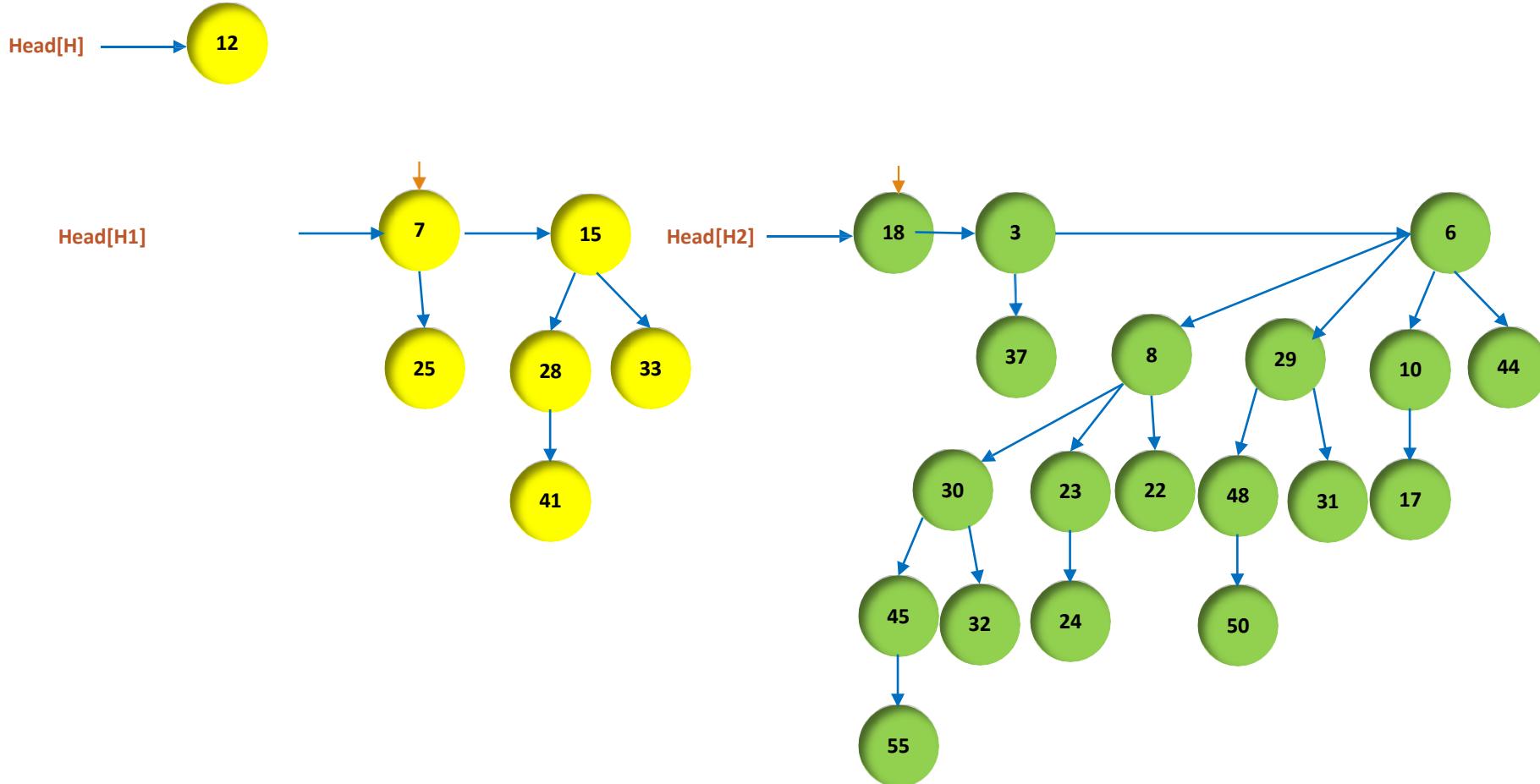
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
            then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
                   $\text{BINOMIAL-LINK}(\text{next-}x, x)$           ▷ Case 3
            else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
                  then  $\text{head}[H] \leftarrow \text{next-}x$           ▷ Case 4
            else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
                   $\text{BINOMIAL-LINK}(x, \text{next-}x)$           ▷ Case 4
                   $x \leftarrow \text{next-}x$                       ▷ Case 4
    21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

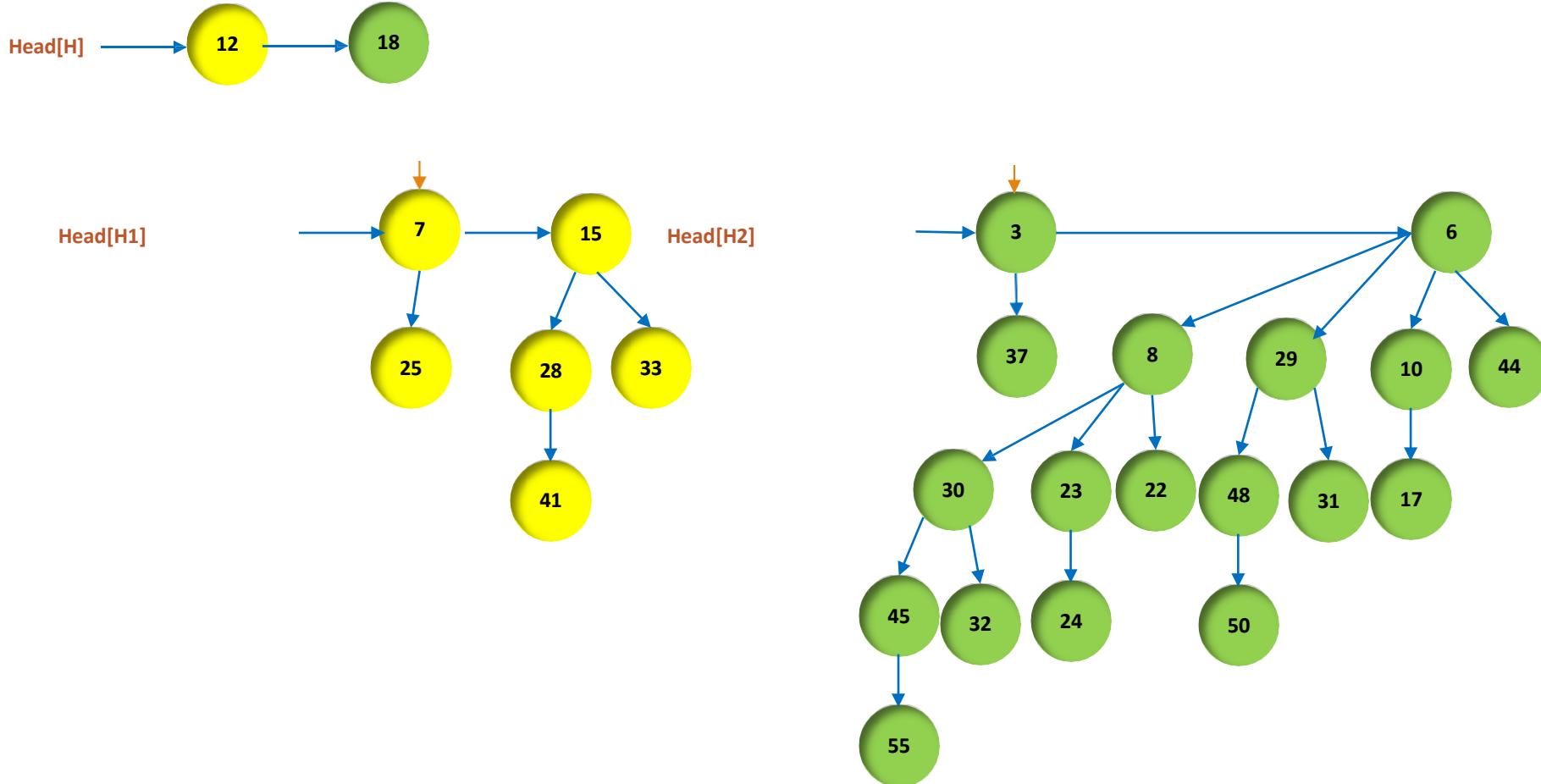
# Binomial Merge Operation...



```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 
    
```

# Binomial Merge Operation...



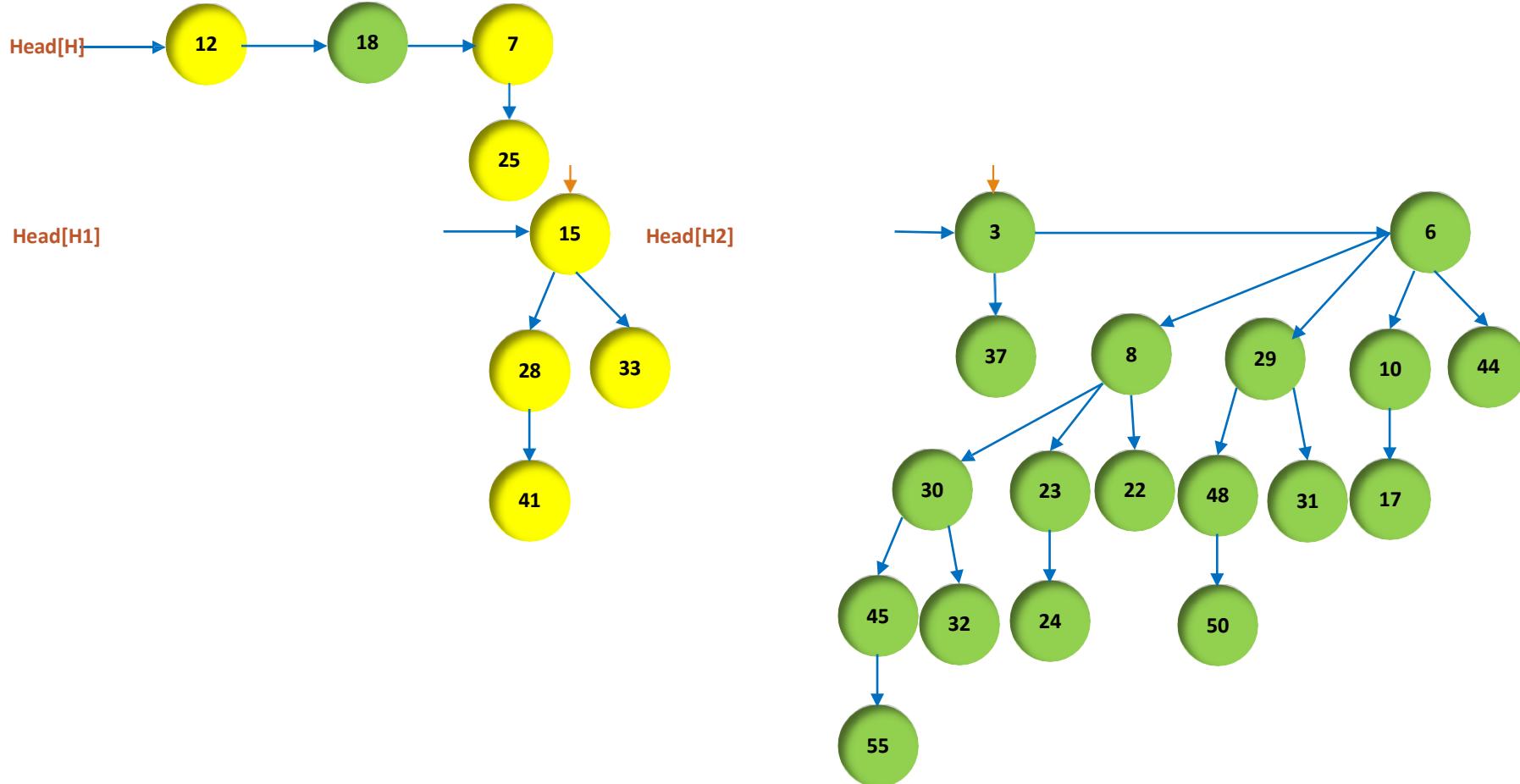
**BINOMIAL-HEAP-UNION( $H_1, H_2$ )**

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12    $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14   then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15      $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16   else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17     then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18   else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19      $\text{BINOMIAL-LINK}(x, \text{next-}x)$             ▷ Case 4
20    $x \leftarrow \text{next-}x$                           ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Merge Operation...

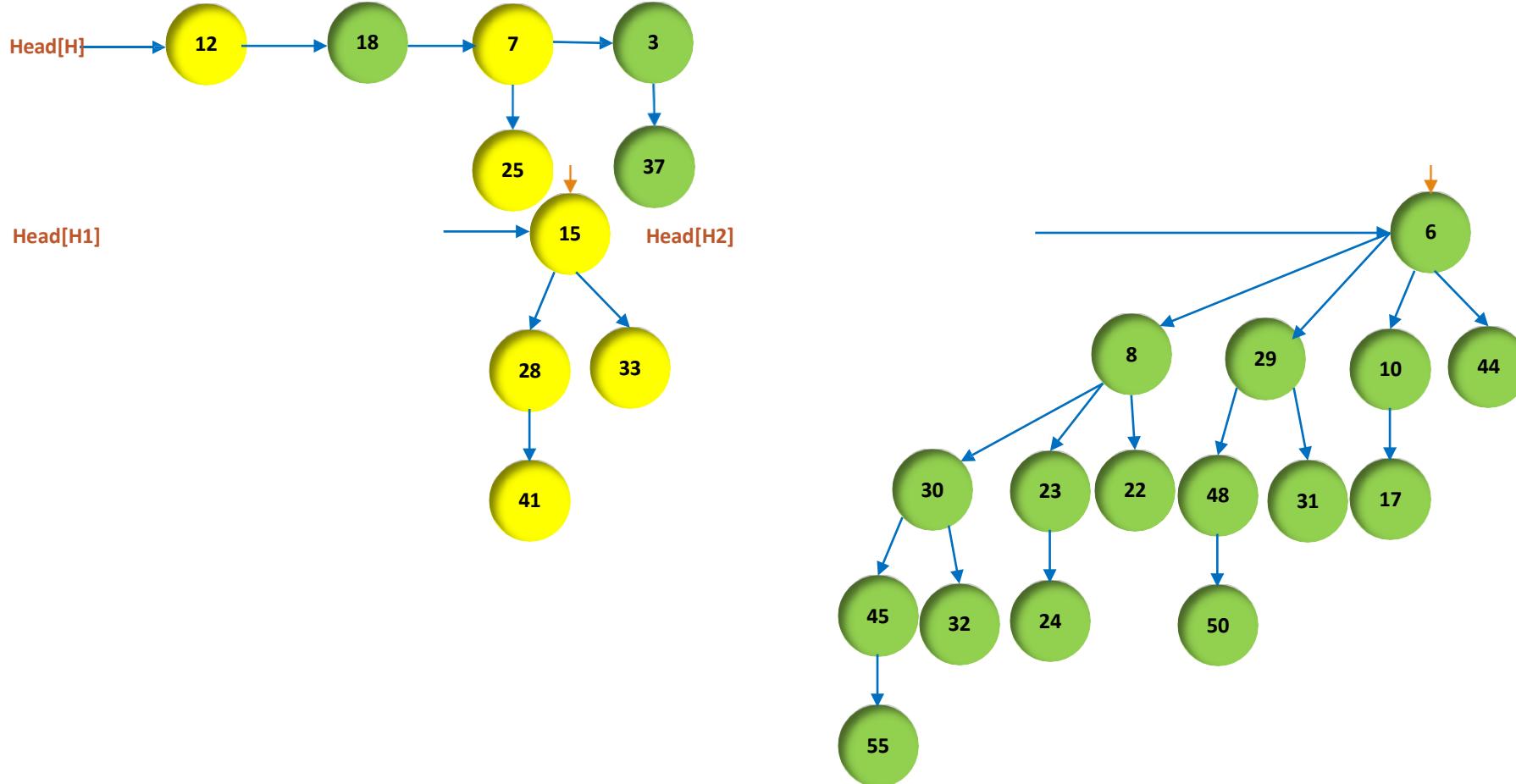


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12    $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14   then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15     BINOMIAL-LINK( $\text{next-}x, x$ )                  ▷ Case 3
16   else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17     then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18   else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$  ▷ Case 4
19     BINOMIAL-LINK( $x, \text{next-}x$ )                ▷ Case 4
20    $x \leftarrow \text{next-}x$                           ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Merge Operation...

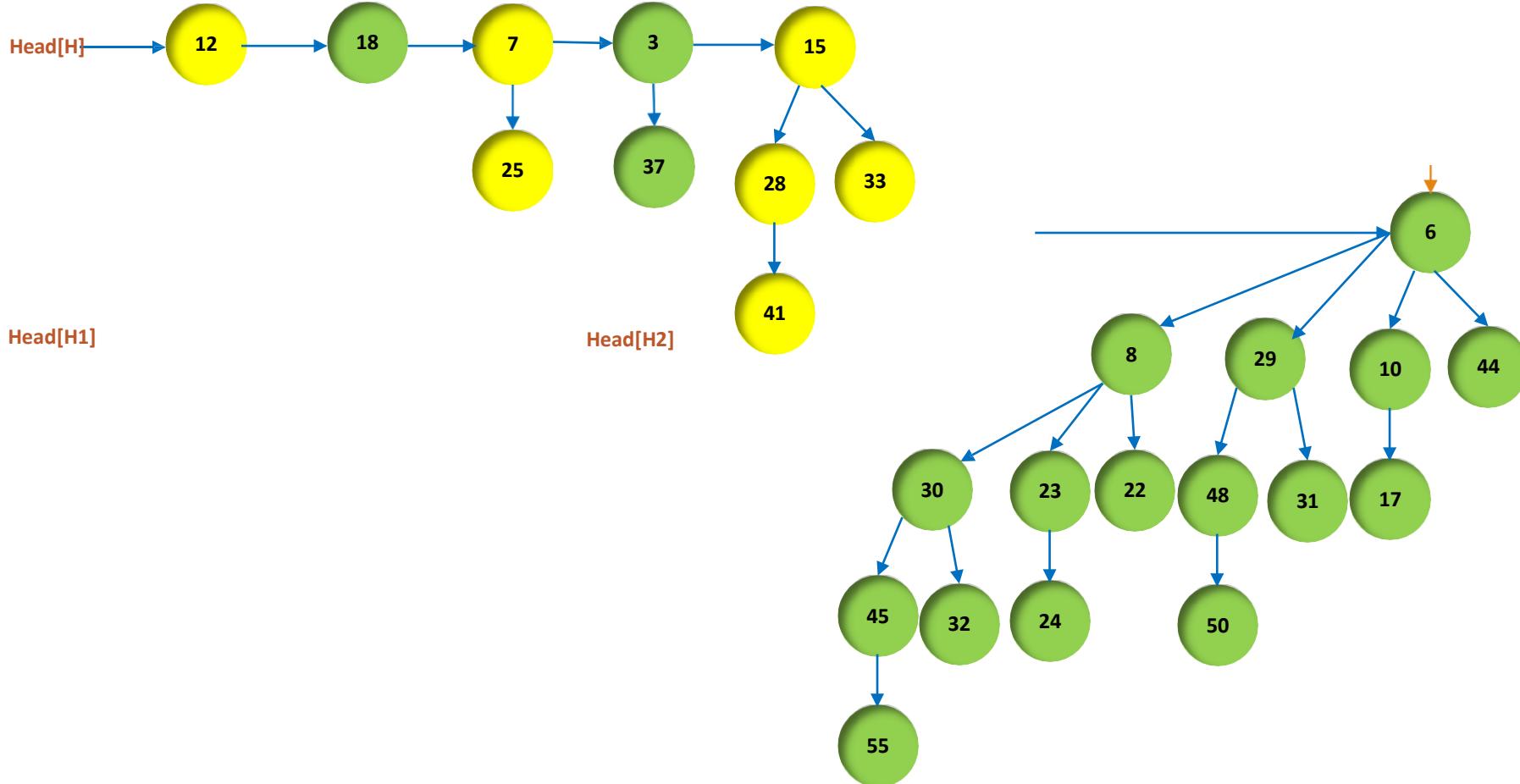


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Merge Operation...

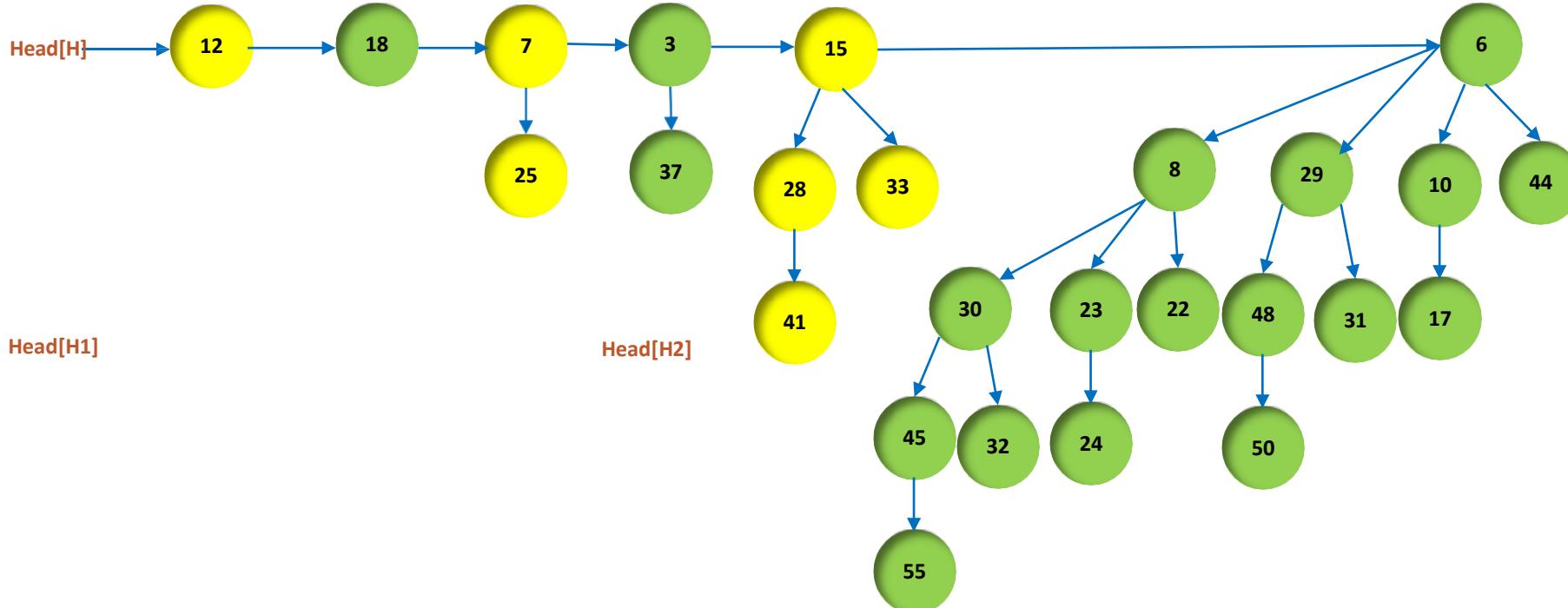


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Merge Operation...

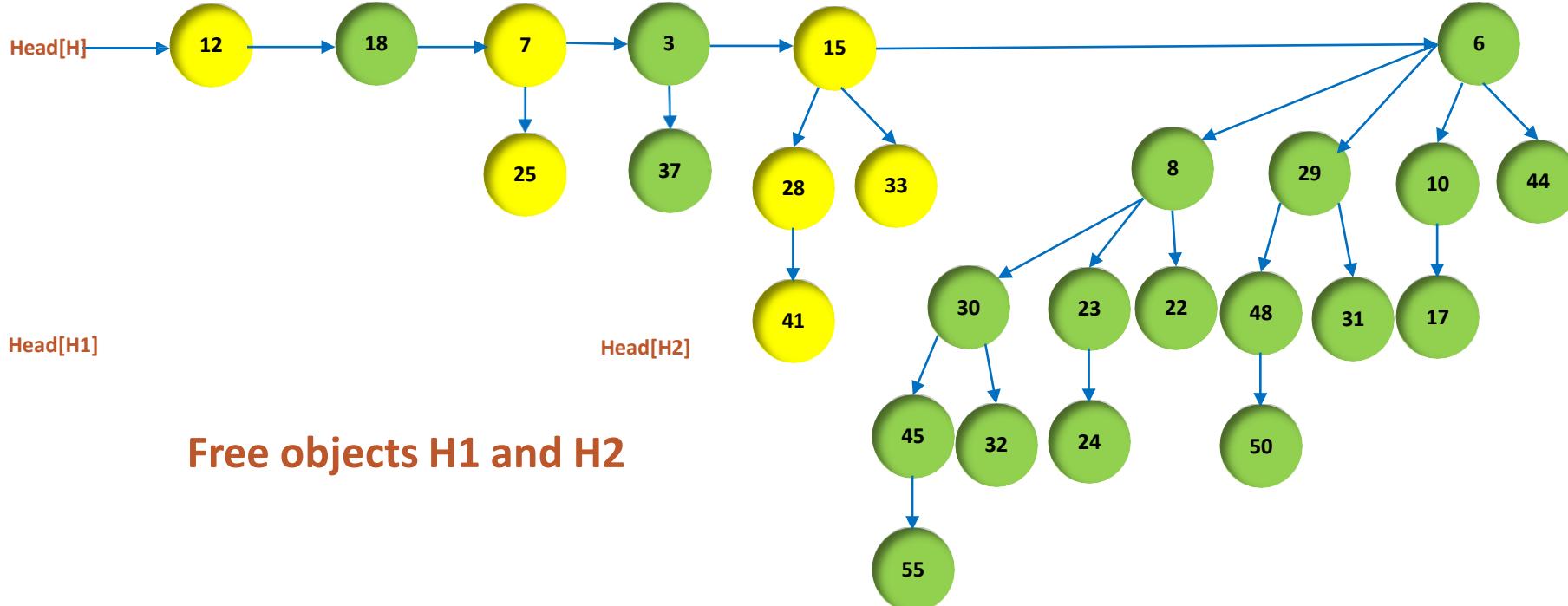


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Merge Operation...

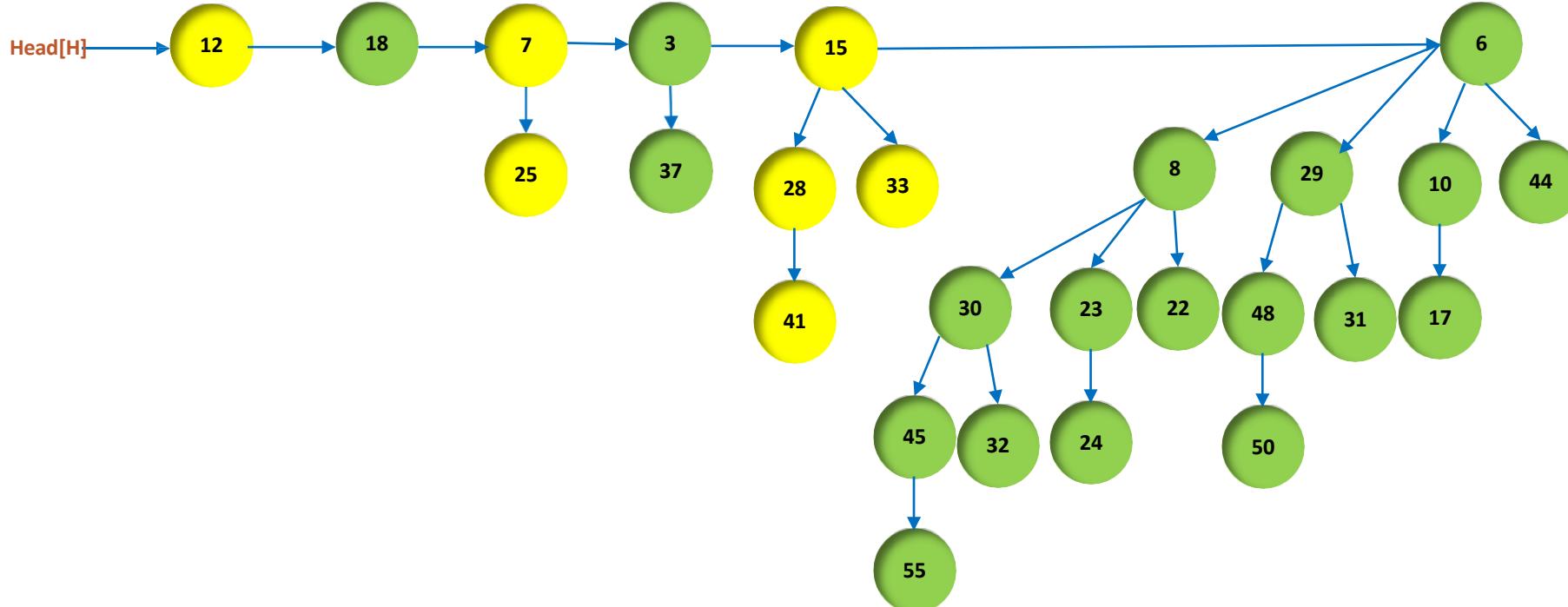


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
       $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
12 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
13 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
       $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
14 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
15 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
16 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
17  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
18  $x \leftarrow \text{next-}x$                             ▷ Case 4
19  $\text{next-}x \leftarrow \text{sibling}[x]$                     ▷ Case 4
20
21
22 return  $H$ 

```

# Binomial Merge Operation...



BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12    $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14   then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15     BINOMIAL-LINK( $\text{next-}x, x$ )                  ▷ Case 3
16   else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17     then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18   else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$  ▷ Case 4
19     BINOMIAL-LINK( $x, \text{next-}x$ )                ▷ Case 4
20    $x \leftarrow \text{next-}x$                           ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```



Design and Analysis of  
Algorithms

# Binomial Heap - Union Operation

---

# Binomial Union Operation



There can be following 4 cases when we traverse the list of roots.

**Case 1:** Orders of x and next-x are not same, we simply move ahead.

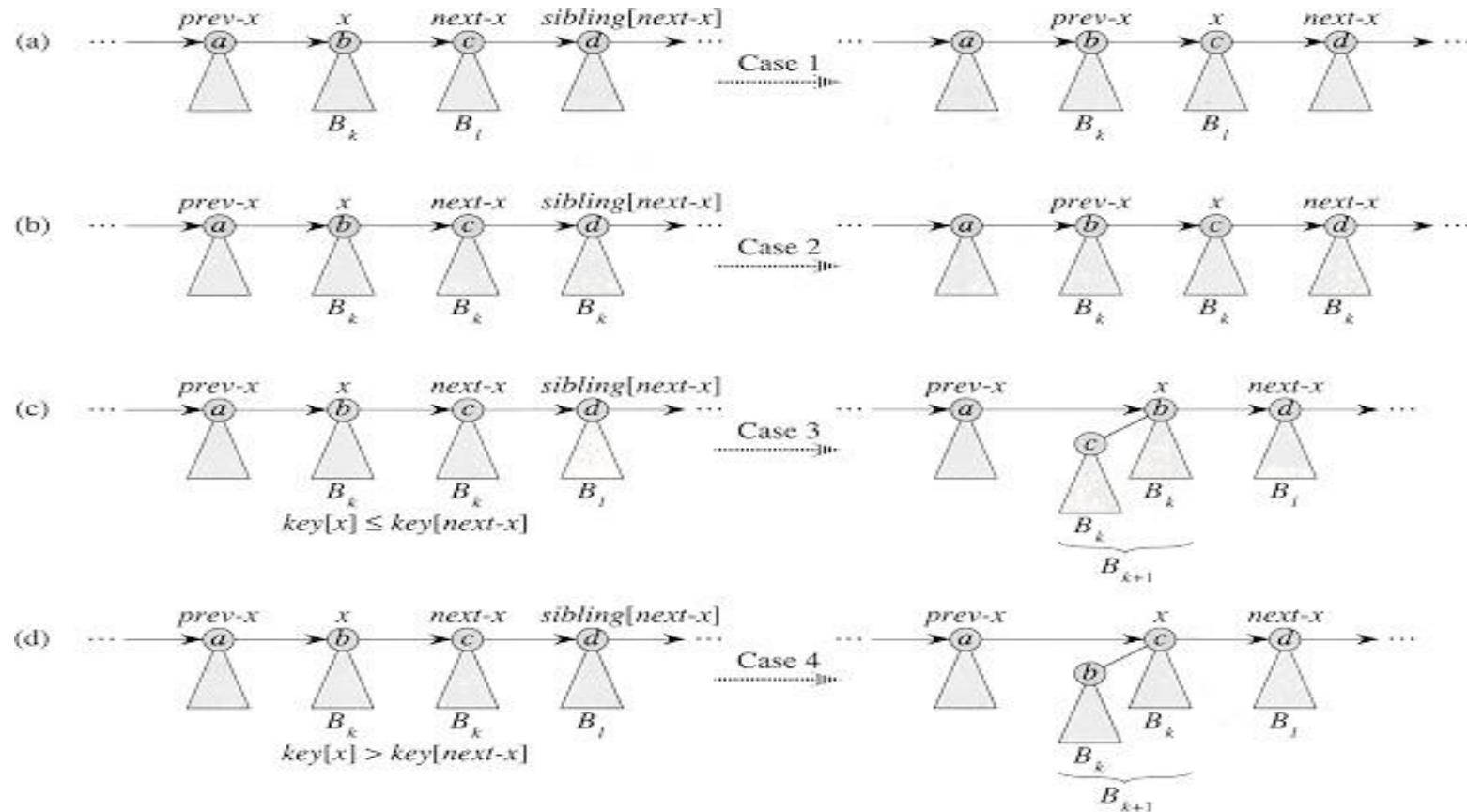
**In following 3 cases orders of x and next-x are same:**

**Case 2:** If the order of next-next-x is also same, move ahead.

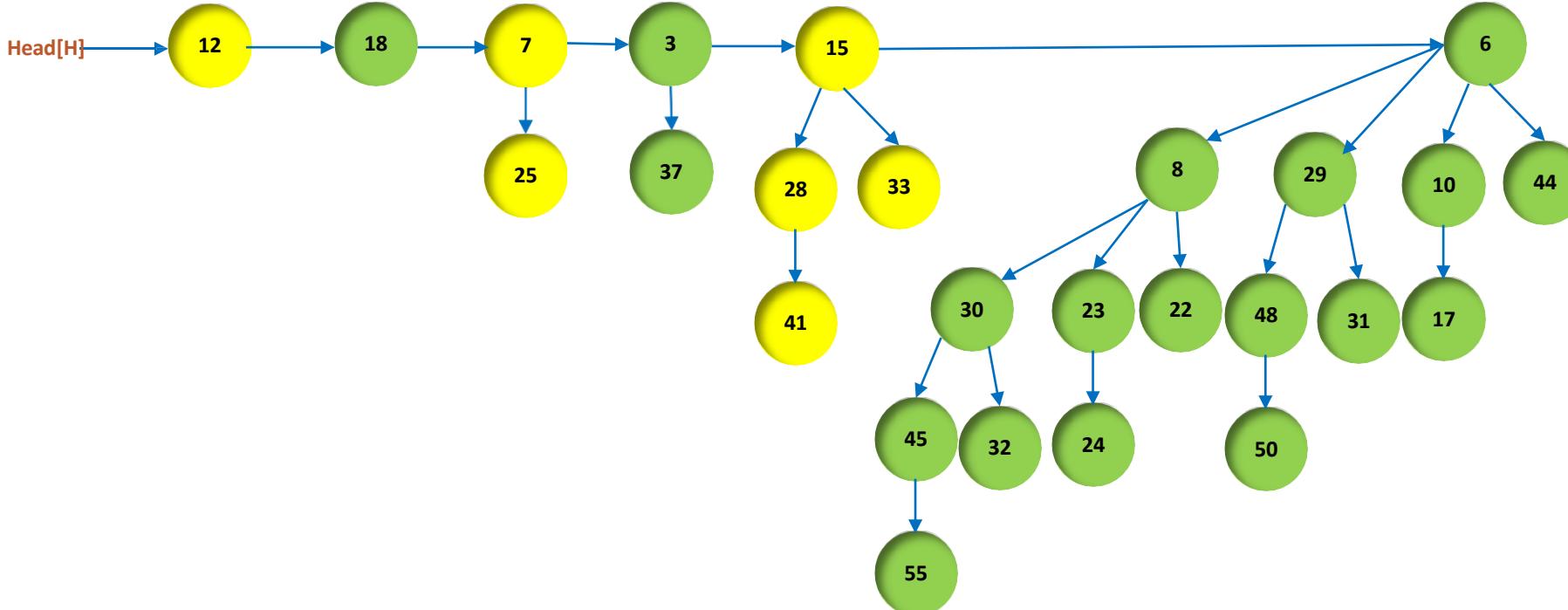
**Case 3:** If the key of x is smaller than or equal to the key of next-x, then make next-x as a child of x by linking it with x.

**Case 4:** If the key of x is greater, then make x as the child of next.

# Binomial Union Operation



# Binomial Union Operation...



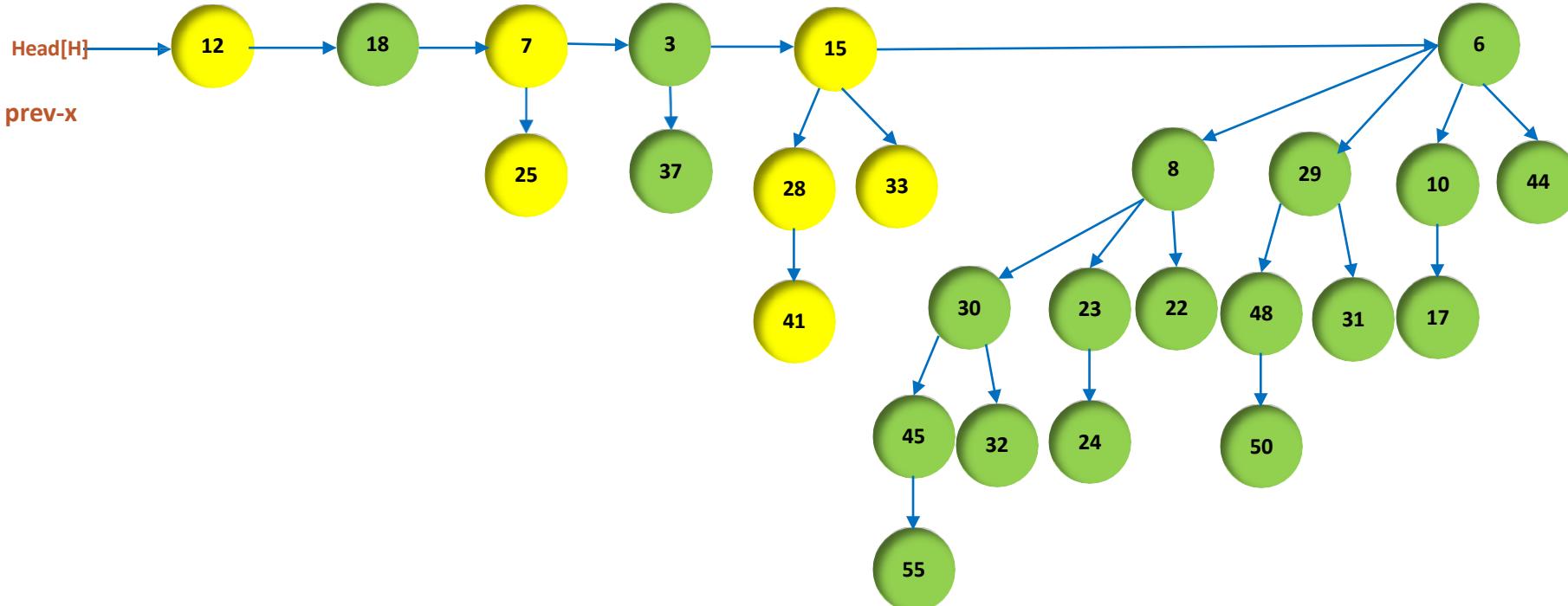
**BINOMIAL-HEAP-UNION( $H_1, H_2$ )**

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3  $\text{free the objects } H_1 \text{ and } H_2 \text{ but not the lists they point to}$ 
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

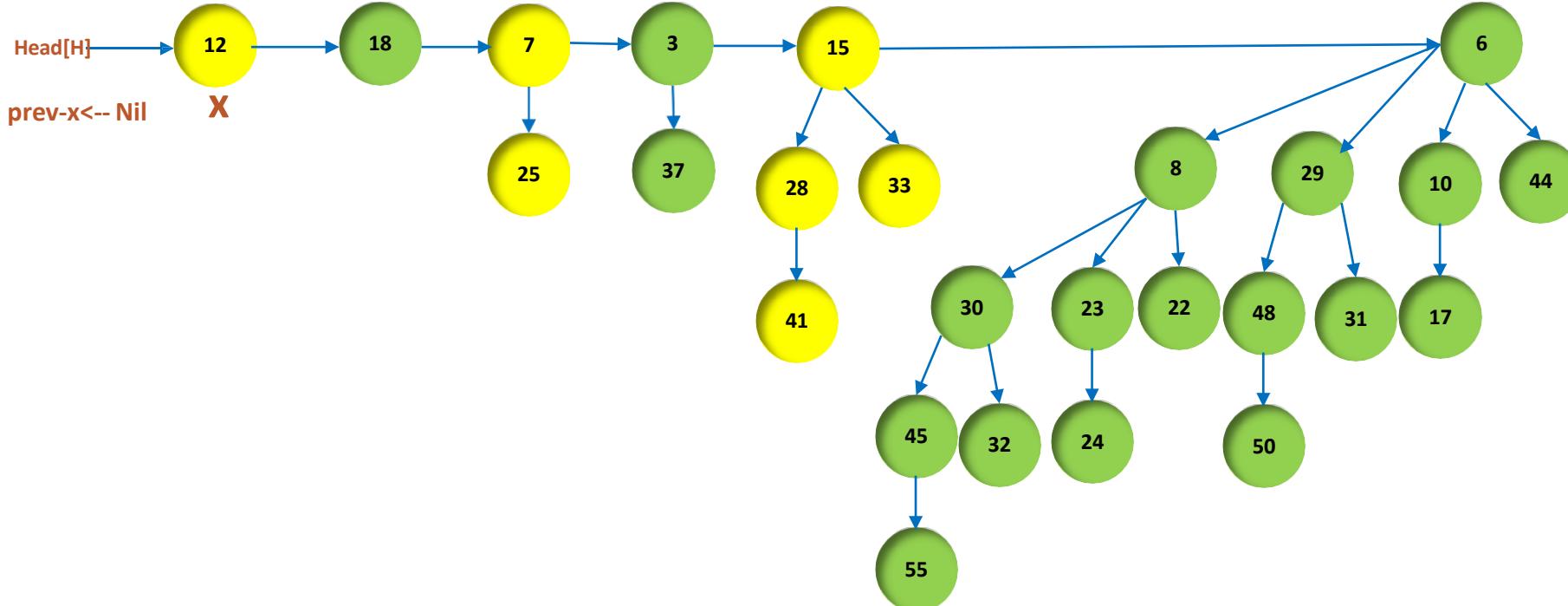


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12    $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14   then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15      $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16   else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17     then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18   else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19      $\text{BINOMIAL-LINK}(x, \text{next-}x)$             ▷ Case 4
20    $x \leftarrow \text{next-}x$                           ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



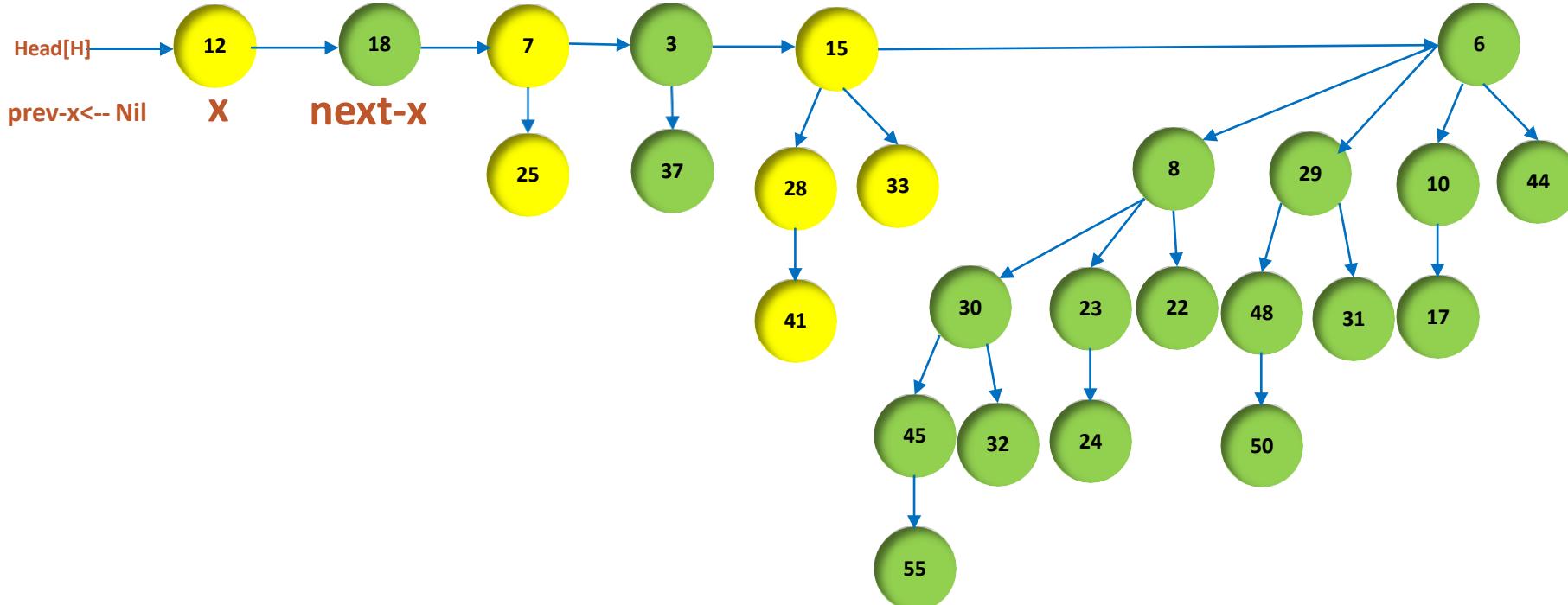
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{ sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{ sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                  $\triangleright$  Cases 1 and 2
12  $x \leftarrow \text{next-}x$                              $\triangleright$  Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{ sibling}[x] \leftarrow \text{ sibling}[\text{next-}x]$        $\triangleright$  Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                    $\triangleright$  Case 3
16 else if  $\text{prev-}x = \text{NIL}$                           $\triangleright$  Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                  $\triangleright$  Case 4
18 else  $\text{ sibling}[\text{prev-}x] \leftarrow \text{next-}x$      $\triangleright$  Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                    $\triangleright$  Case 4
20  $x \leftarrow \text{next-}x$                              $\triangleright$  Case 4
21  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

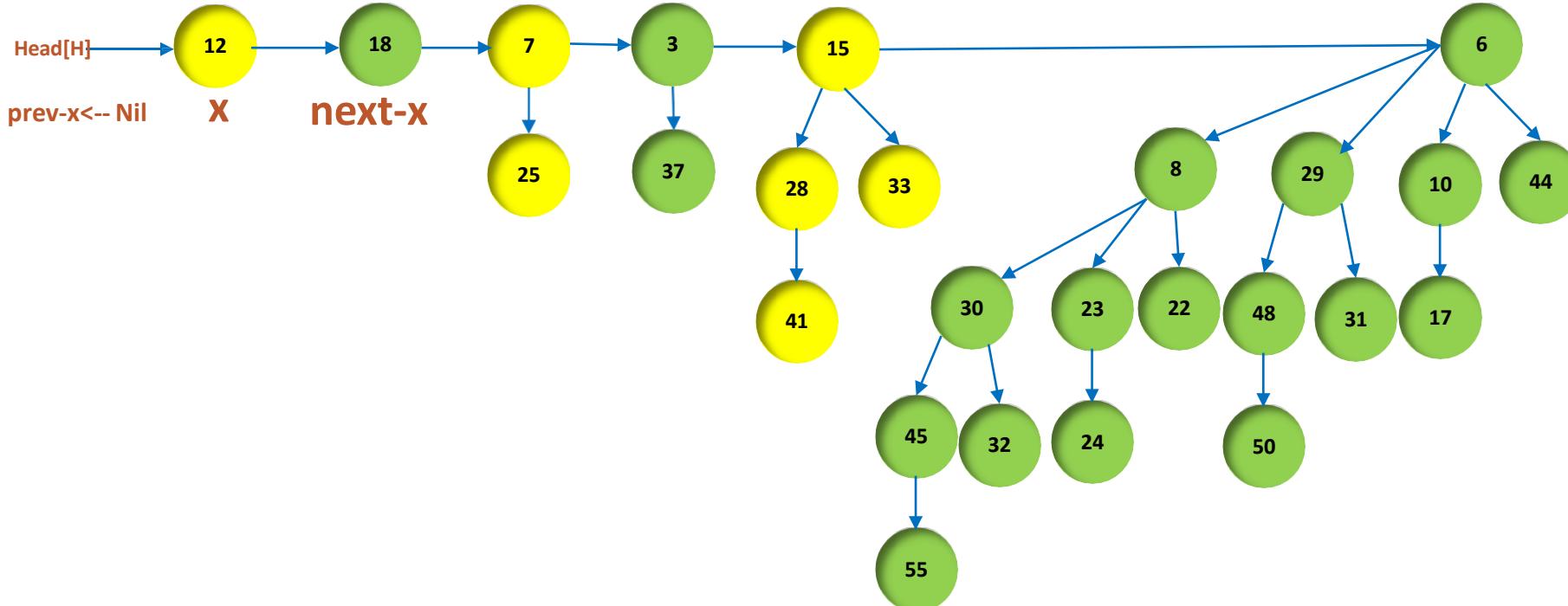


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                 ▷ Case 4
20  $x \leftarrow \text{next-}x$                           ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



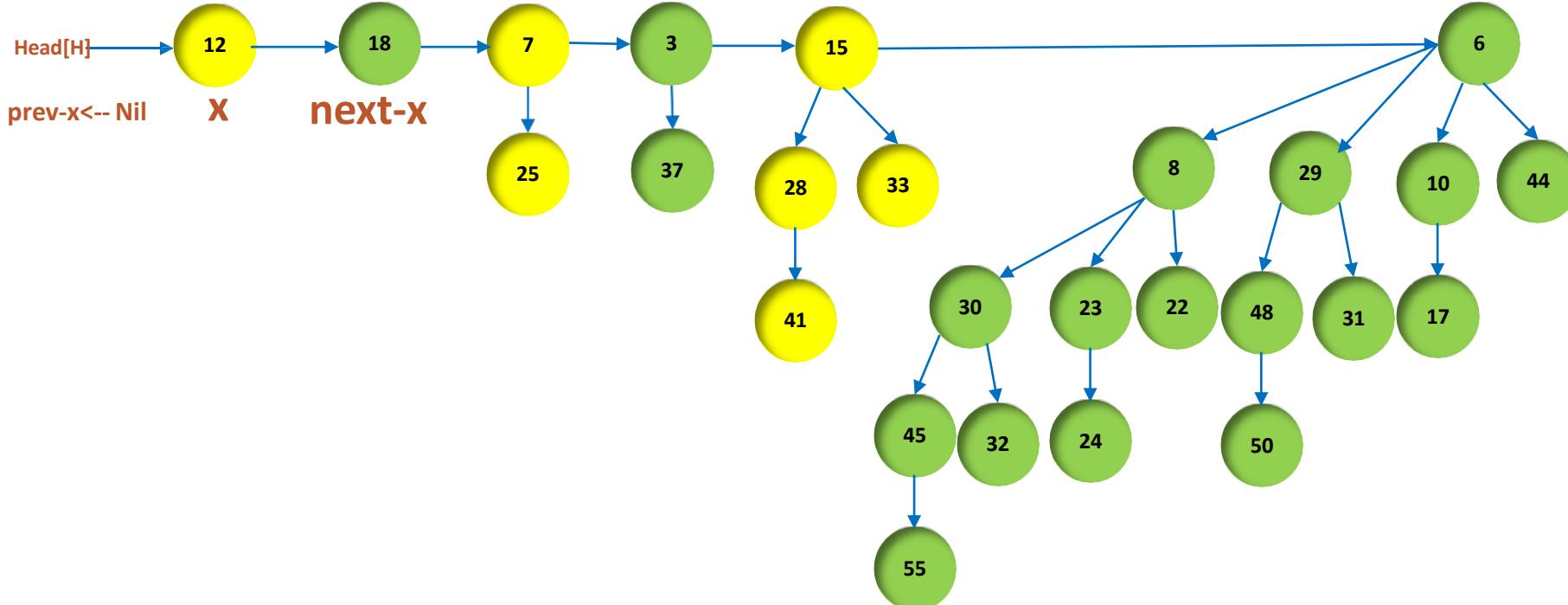
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev}-x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next}-x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next}-x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next}-x]$ ) or
     ( $\text{sibling}[\text{next}-x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next}-x]] = \text{degree}[x]$ )
11 then  $\text{prev}-x \leftarrow x$                                  $\triangleright$  Cases 1 and 2
12  $x \leftarrow \text{next}-x$                              $\triangleright$  Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next}-x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next}-x]$            $\triangleright$  Case 3
15  $\text{BINOMIAL-LINK}(\text{next}-x, x)$                    $\triangleright$  Case 3
16 else if  $\text{prev}-x = \text{NIL}$                           $\triangleright$  Case 4
17 then  $\text{head}[H] \leftarrow \text{next}-x$                  $\triangleright$  Case 4
18 else  $\text{sibling}[\text{prev}-x] \leftarrow \text{next}-x$         $\triangleright$  Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next}-x)$                    $\triangleright$  Case 4
20  $x \leftarrow \text{next}-x$                              $\triangleright$  Case 4
21  $\text{next}-x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

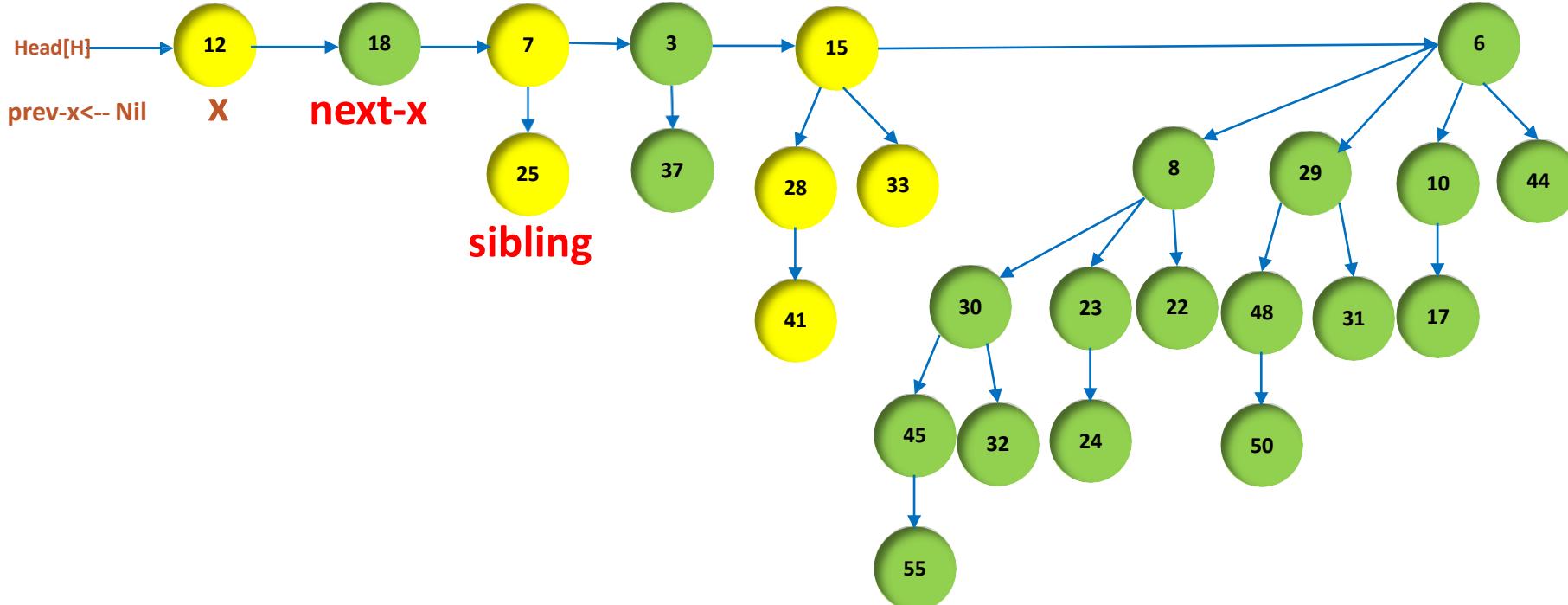


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-x} \neq \text{NIL}$ 
10 do if  $(\text{degree}[x] \neq \text{degree}[\text{next-x}])$  or
        $(\text{sibling}[\text{next-x}] \neq \text{NIL}$ 
          and  $\text{degree}[\text{sibling}[\text{next-x}]] = \text{degree}[x]$ )
    then  $\text{prev-x} \leftarrow x$                                 ▷ Cases 1 and 2
     $x \leftarrow \text{next-x}$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-x}]$ 
    then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-x}]$       ▷ Case 3
     $\text{BINOMIAL-LINK}(\text{next-x}, x)$                   ▷ Case 3
16 else if  $\text{prev-x} = \text{NIL}$                          ▷ Case 4
    then  $\text{head}[H] \leftarrow \text{next-x}$                 ▷ Case 4
    else  $\text{sibling}[\text{prev-x}] \leftarrow \text{next-x}$       ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-x})$                   ▷ Case 4
20  $x \leftarrow \text{next-x}$                                 ▷ Case 4
21  $\text{next-x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

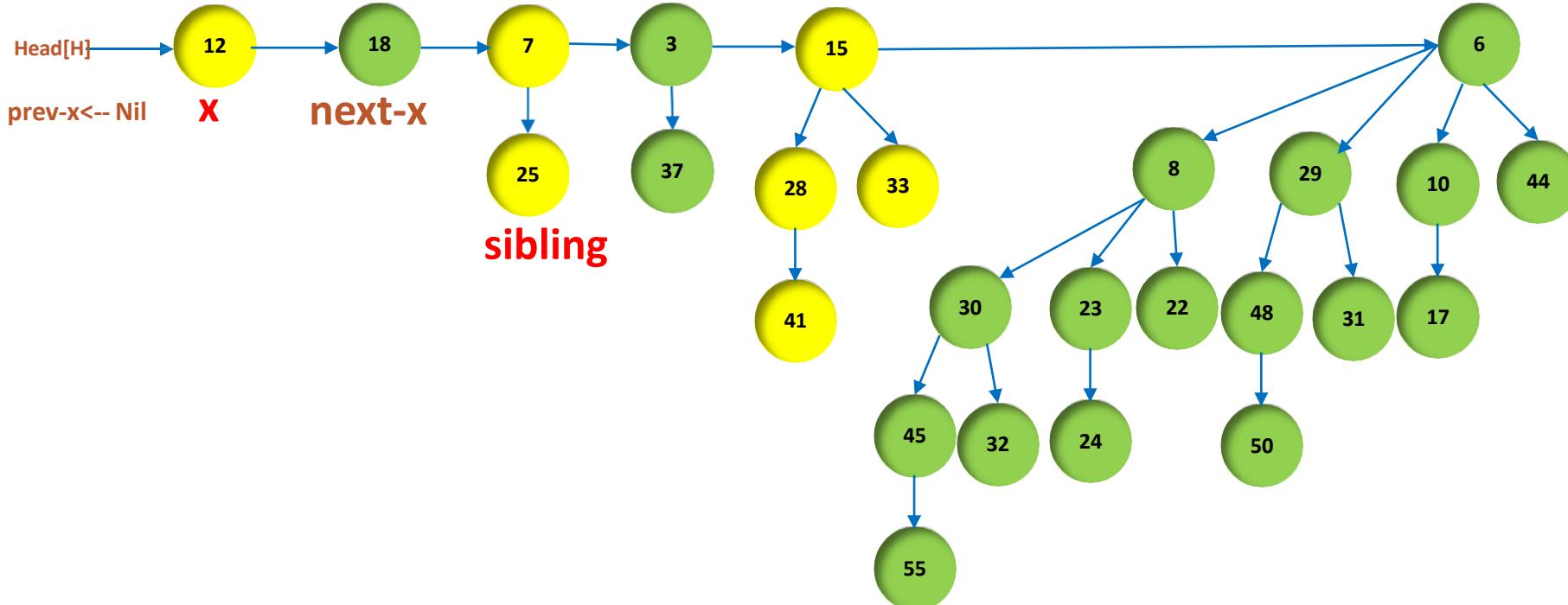


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

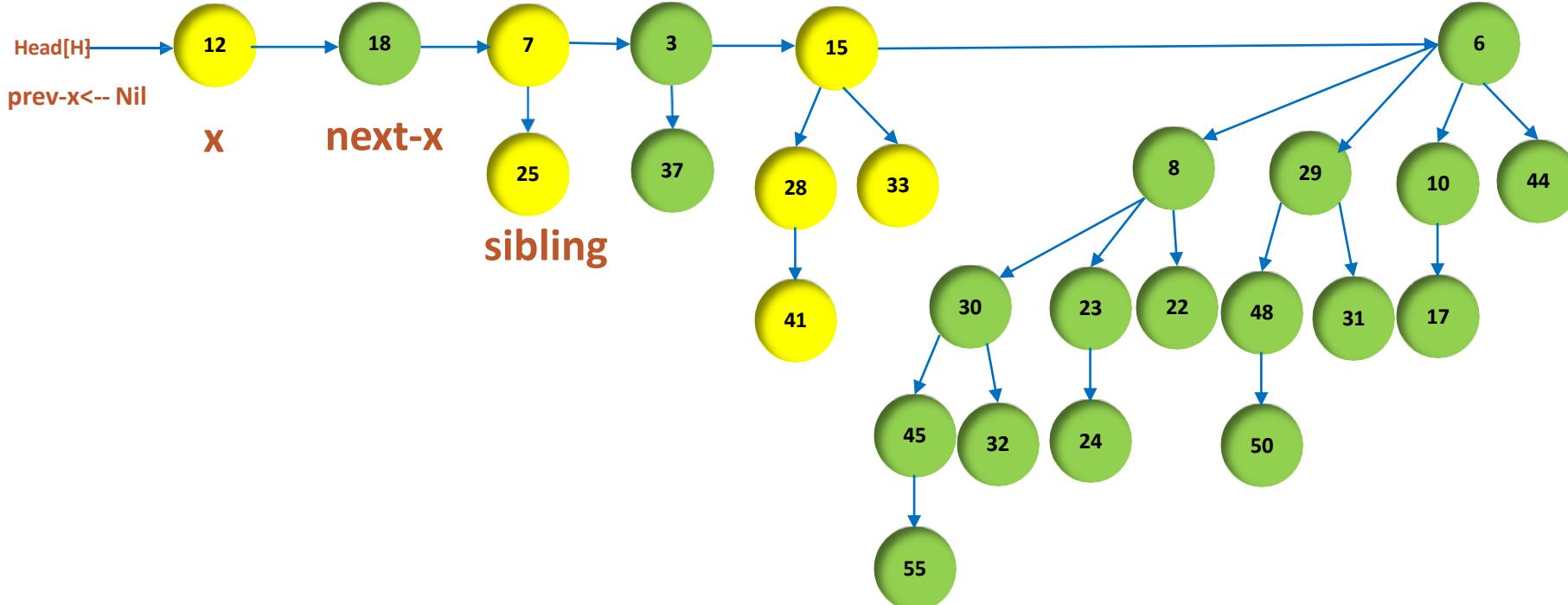


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev}-x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next}-x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next}-x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next}-x]$ ) or
       ( $\text{sibling}[\text{next}-x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next}-x]] = \text{degree}[x]$ )
           then  $\text{prev}-x \leftarrow x$                                 ▷ Cases 1 and 2
                 $x \leftarrow \text{next}-x$                             ▷ Cases 1 and 2
           else if  $\text{key}[x] \leq \text{key}[\text{next}-x]$ 
                then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next}-x]$       ▷ Case 3
                       $\text{BINOMIAL-LINK}(\text{next}-x, x)$           ▷ Case 3
                else if  $\text{prev}-x = \text{NIL}$                          ▷ Case 4
                      then  $\text{head}[H] \leftarrow \text{next}-x$           ▷ Case 4
                else  $\text{sibling}[\text{prev}-x] \leftarrow \text{next}-x$     ▷ Case 4
                       $\text{BINOMIAL-LINK}(x, \text{next}-x)$           ▷ Case 4
                 $x \leftarrow \text{next}-x$                             ▷ Case 4
            $\text{next}-x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

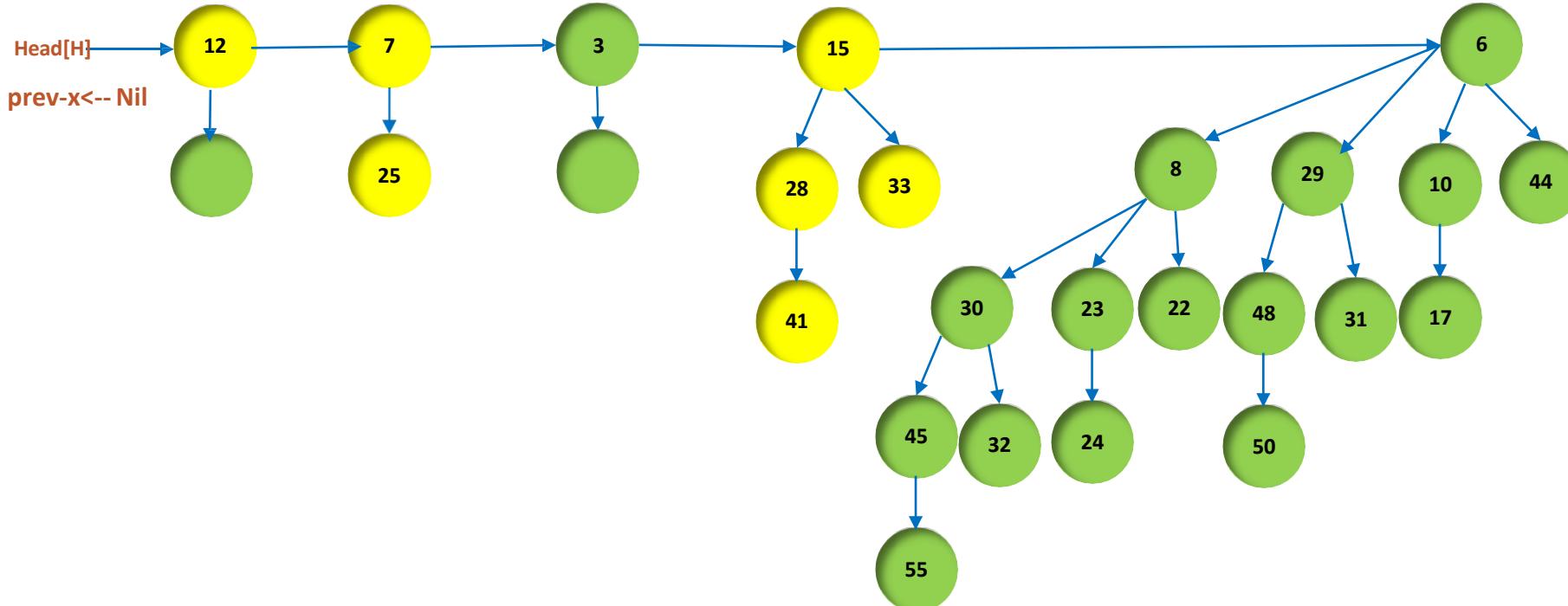


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-x} \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-x}]$ ) or
     ( $\text{sibling}[\text{next-x}] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-x}]] = \text{degree}[x]$ )
11 then  $\text{prev-x} \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-x}$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-x}]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-x}]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-x}, x)$                       ▷ Case 3
16 else if  $\text{prev-x} = \text{NIL}$                             ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-x}$                     ▷ Case 4
18 else  $\text{sibling}[\text{prev-x}] \leftarrow \text{next-x}$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-x})$                       ▷ Case 4
20  $x \leftarrow \text{next-x}$                                 ▷ Case 4
21  $\text{next-x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

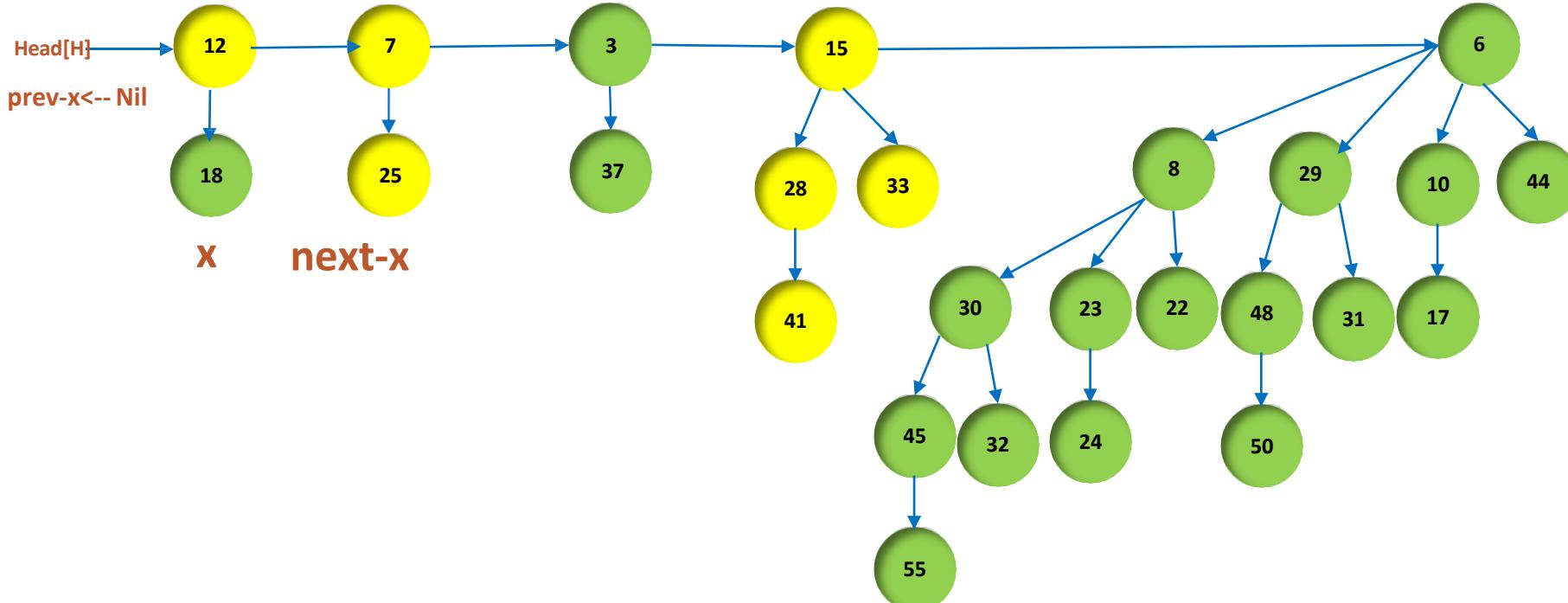


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] < \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

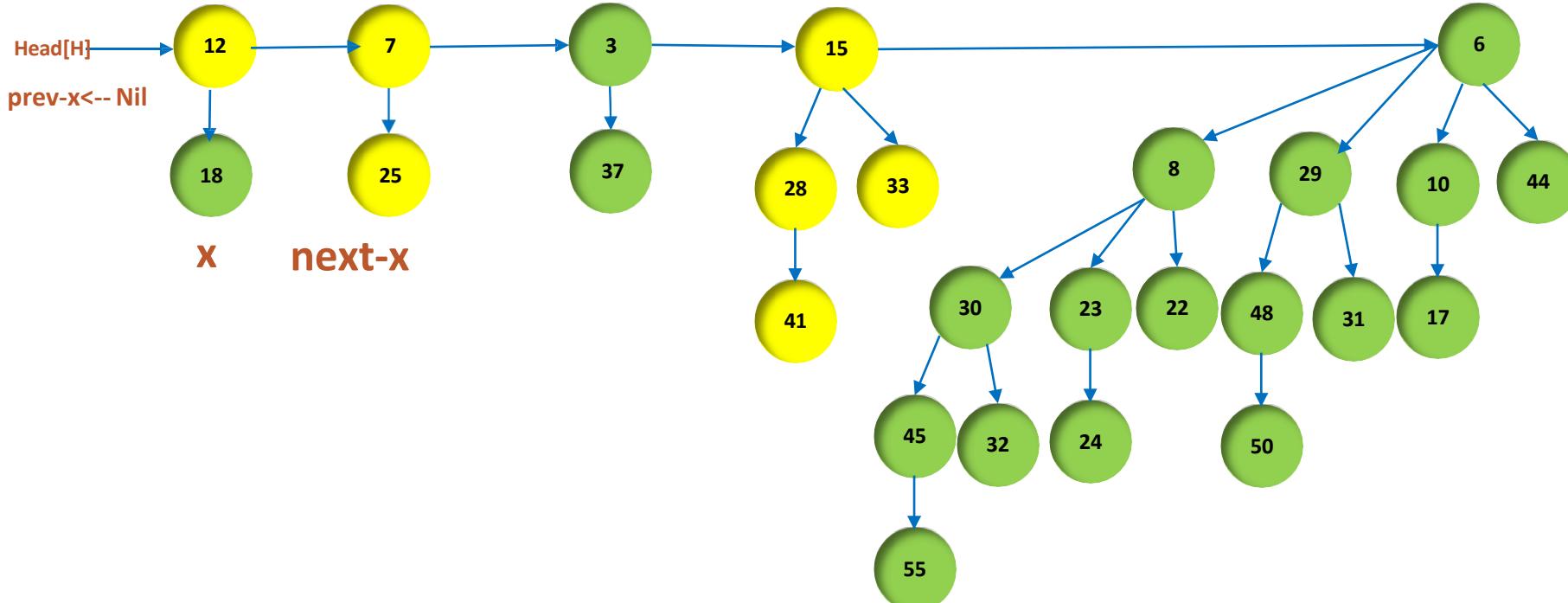


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-x} \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-x}]$ ) or
       ( $\text{sibling}[\text{next-x}] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-x}]] = \text{degree}[x]$ )
    then  $\text{prev-x} \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-x}$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next-x}]$ 
        then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-x}]$       ▷ Case 3
               $\text{BINOMIAL-LINK}(\text{next-x}, x)$            ▷ Case 3
    else if  $\text{prev-x} = \text{NIL}$                          ▷ Case 4
        then  $\text{head}[H] \leftarrow \text{next-x}$                 ▷ Case 4
    else  $\text{sibling}[\text{prev-x}] \leftarrow \text{next-x}$         ▷ Case 4
           $\text{BINOMIAL-LINK}(x, \text{next-x})$              ▷ Case 4
           $x \leftarrow \text{next-x}$                           ▷ Case 4
21  $\text{next-x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

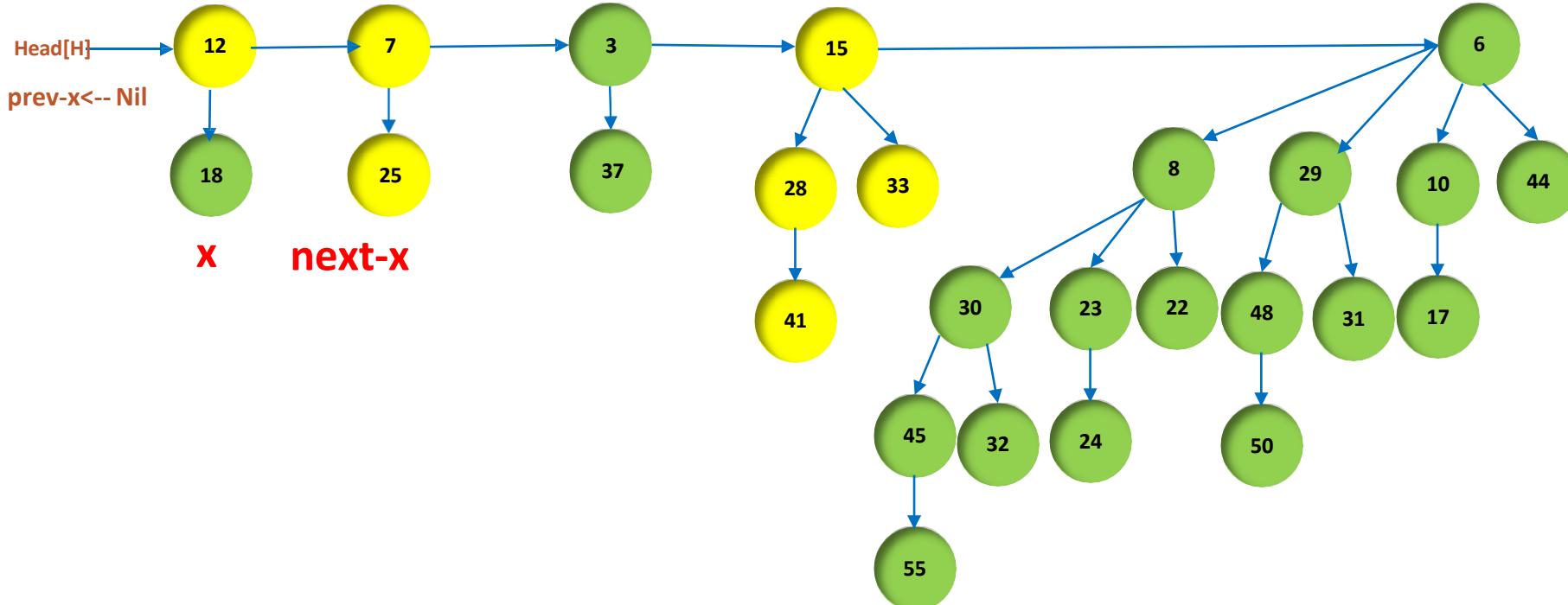


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-x} \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-x}]$ ) or
     ( $\text{sibling}[\text{next-x}] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-x}]] = \text{degree}[x]$ )
11 then  $\text{prev-x} \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-x}$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-x}]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-x}]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-x}, x)$                    ▷ Case 3
16 else if  $\text{prev-x} = \text{NIL}$                           ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-x}$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-x}] \leftarrow \text{next-x}$        ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-x})$                    ▷ Case 4
20  $x \leftarrow \text{next-x}$                                 ▷ Case 4
21  $\text{next-x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

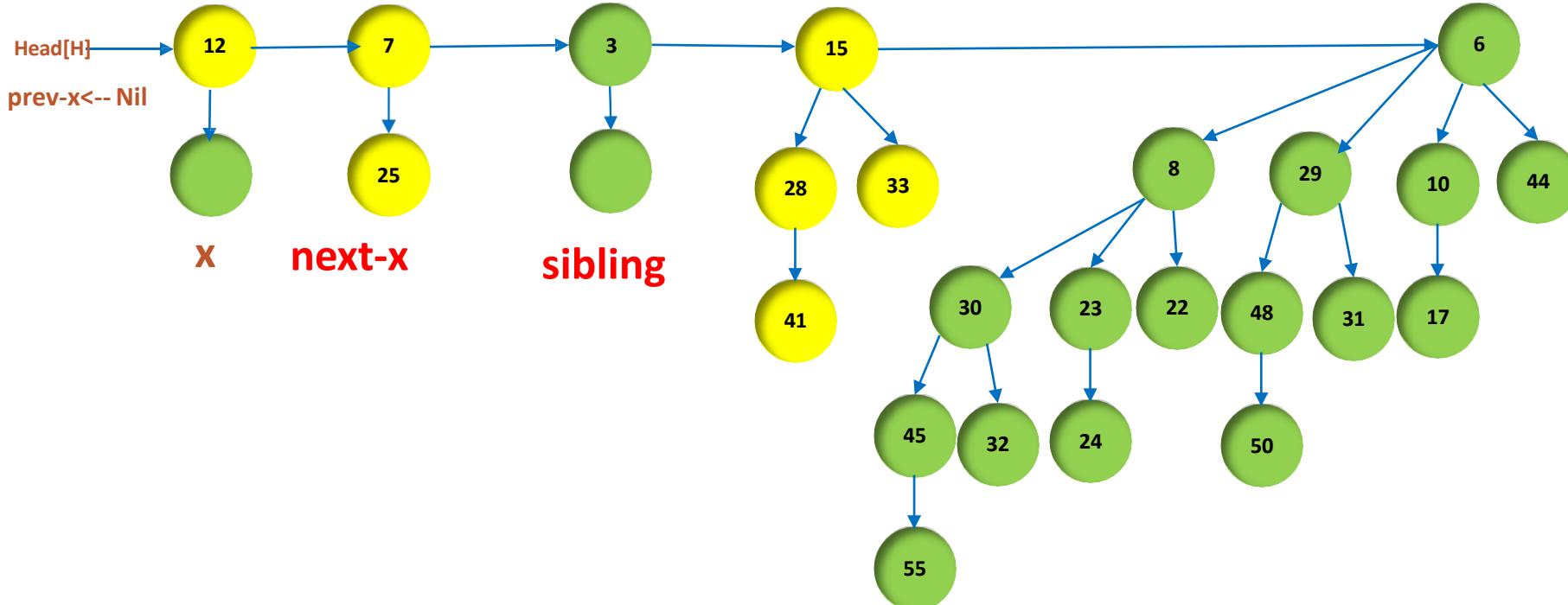


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
     $x \leftarrow \text{next-}x$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
    then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
     $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
    then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

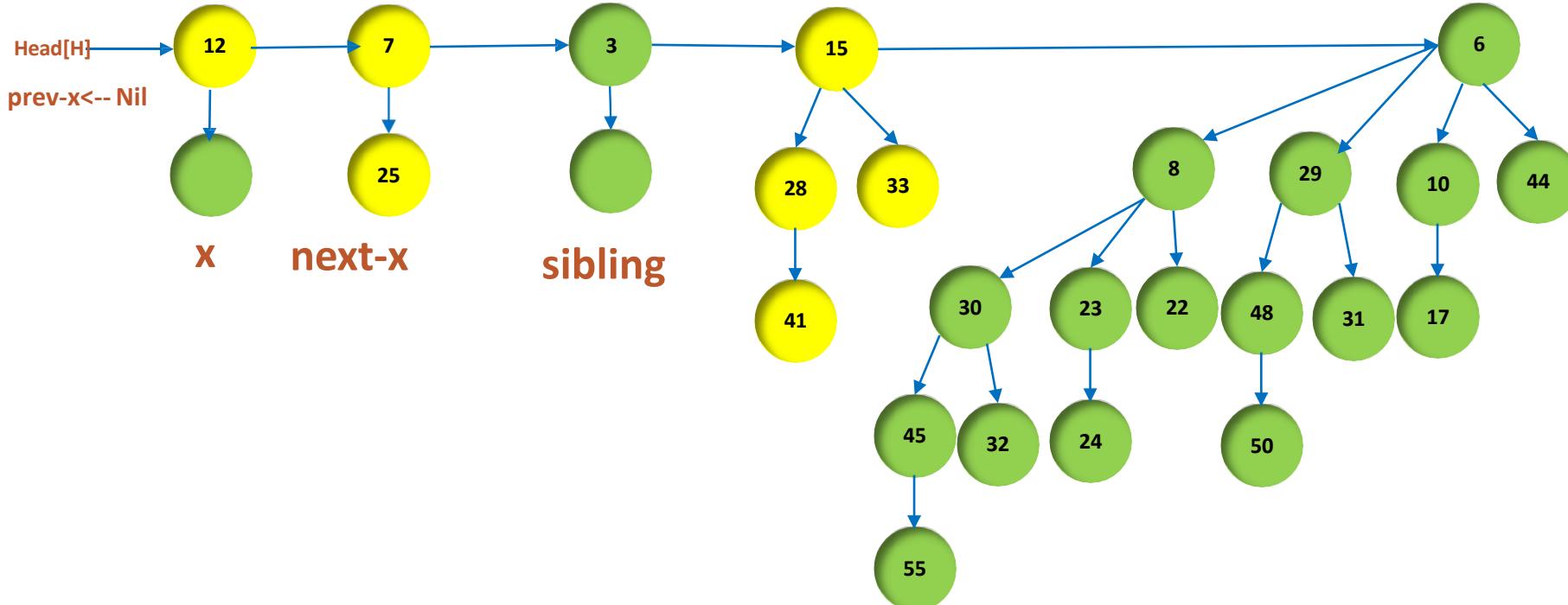
# Binomial Union Operation...



```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $head[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $head[H] = \text{NIL}$ 
5 then return  $H$ 
6  $prev-x \leftarrow \text{NIL}$ 
7  $x \leftarrow head[H]$ 
8  $next-x \leftarrow \text{sibling}[x]$ 
9 while  $next-x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[next-x]$ ) or
     ( $\text{sibling}[next-x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[next-x]] = \text{degree}[x]$ )
     then  $prev-x \leftarrow x$                                 ▷ Cases 1 and 2
           $x \leftarrow next-x$                             ▷ Cases 1 and 2
     else if  $\text{key}[x] \leq \text{key}[next-x]$ 
            then  $\text{sibling}[x] \leftarrow \text{sibling}[next-x]$     ▷ Case 3
                   $\text{BINOMIAL-LINK}(next-x, x)$            ▷ Case 3
            else if  $prev-x = \text{NIL}$                       ▷ Case 4
                  then  $head[H] \leftarrow next-x$              ▷ Case 4
            else  $\text{sibling}[prev-x] \leftarrow next-x$        ▷ Case 4
                   $\text{BINOMIAL-LINK}(x, next-x)$            ▷ Case 4
                   $x \leftarrow next-x$                      ▷ Case 4
             $next-x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 
  
```

# Binomial Union Operation...

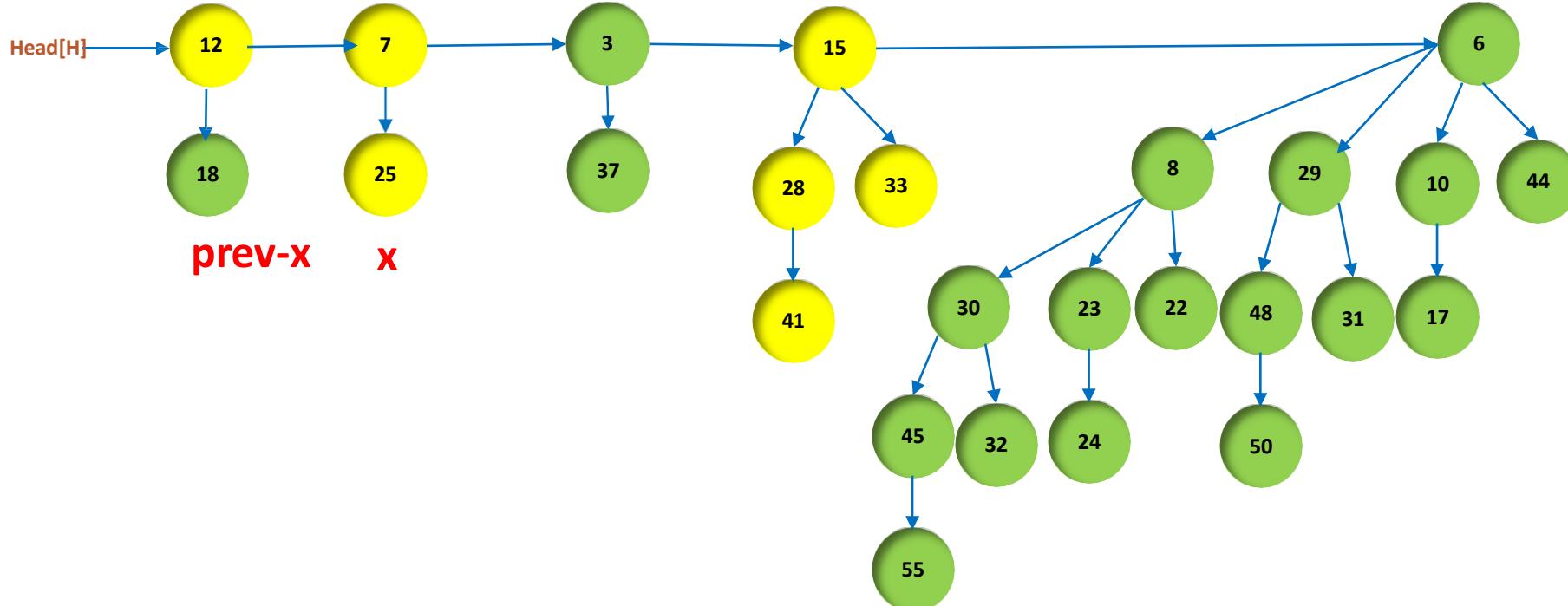


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-x} \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-x}]$ ) or
       ( $\text{sibling}[\text{next-x}] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-x}]] = \text{degree}[x]$ )
           then  $\text{prev-x} \leftarrow x$                                 ▷ Cases 1 and 2
                 $x \leftarrow \text{next-x}$                             ▷ Cases 1 and 2
           else if  $\text{key}[x] \leq \text{key}[\text{next-x}]$ 
                    then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-x}]$     ▷ Case 3
                            $\text{BINOMIAL-LINK}(\text{next-x}, x)$           ▷ Case 3
           else if  $\text{prev-x} = \text{NIL}$ 
                    then  $\text{head}[H] \leftarrow \text{next-x}$           ▷ Case 4
           else  $\text{sibling}[\text{prev-x}] \leftarrow \text{next-x}$       ▷ Case 4
                            $\text{BINOMIAL-LINK}(x, \text{next-x})$           ▷ Case 4
            $x \leftarrow \text{next-x}$                             ▷ Case 4
            $\text{next-x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

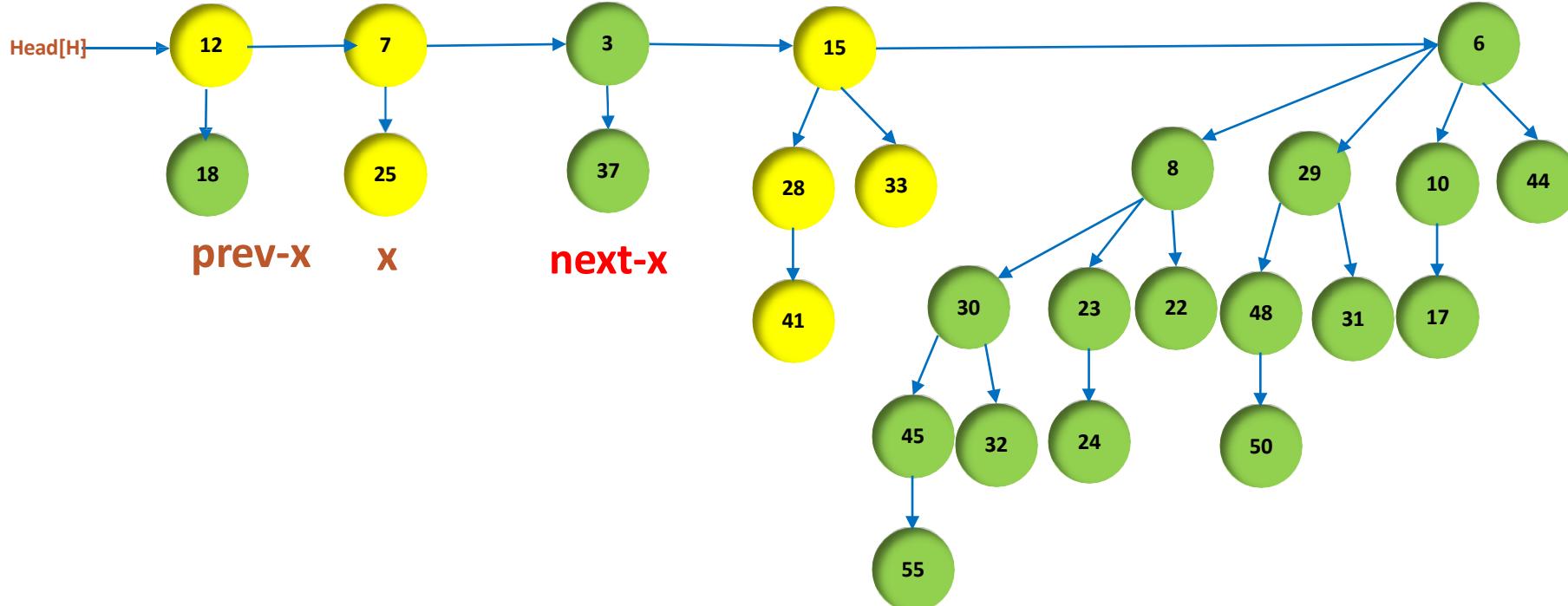


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{ sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{ sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{ sibling}[x] \leftarrow \text{ sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{ sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



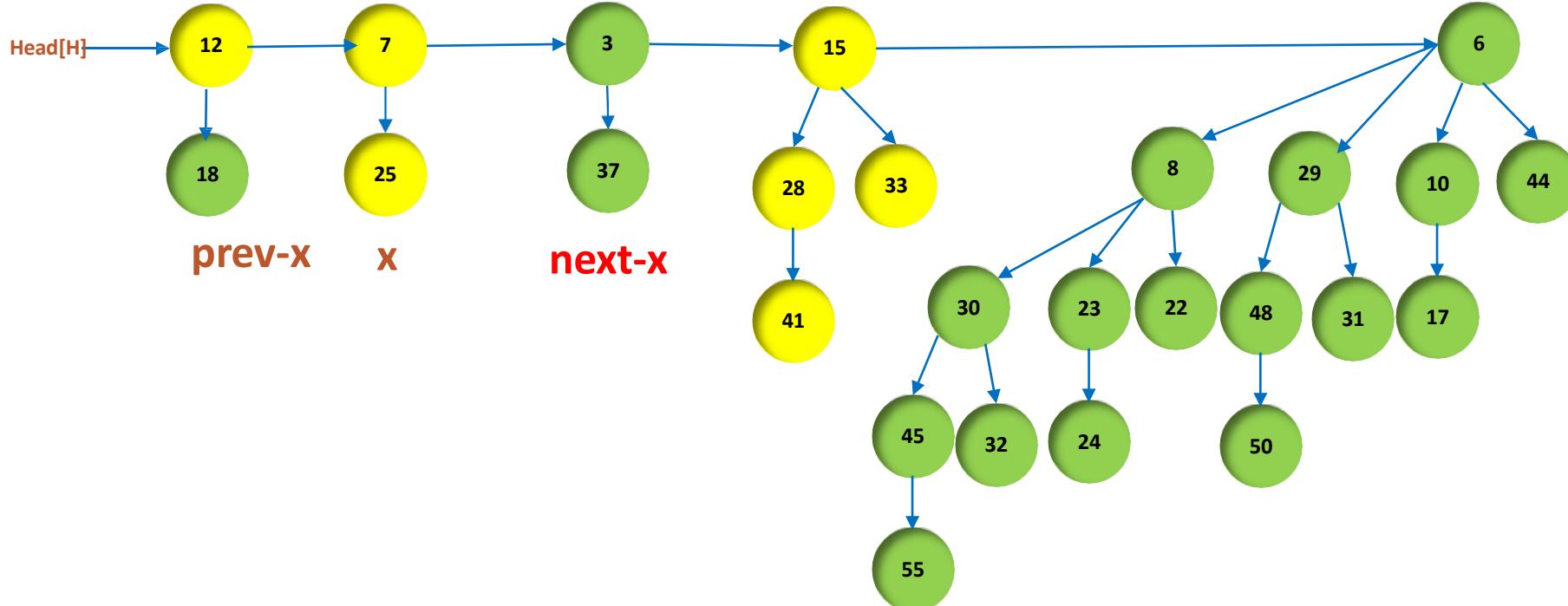
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{ sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{ sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{ sibling}[x] \leftarrow \text{ sibling}[\text{next-}x]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{ sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                                 ▷ Case 4
21  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

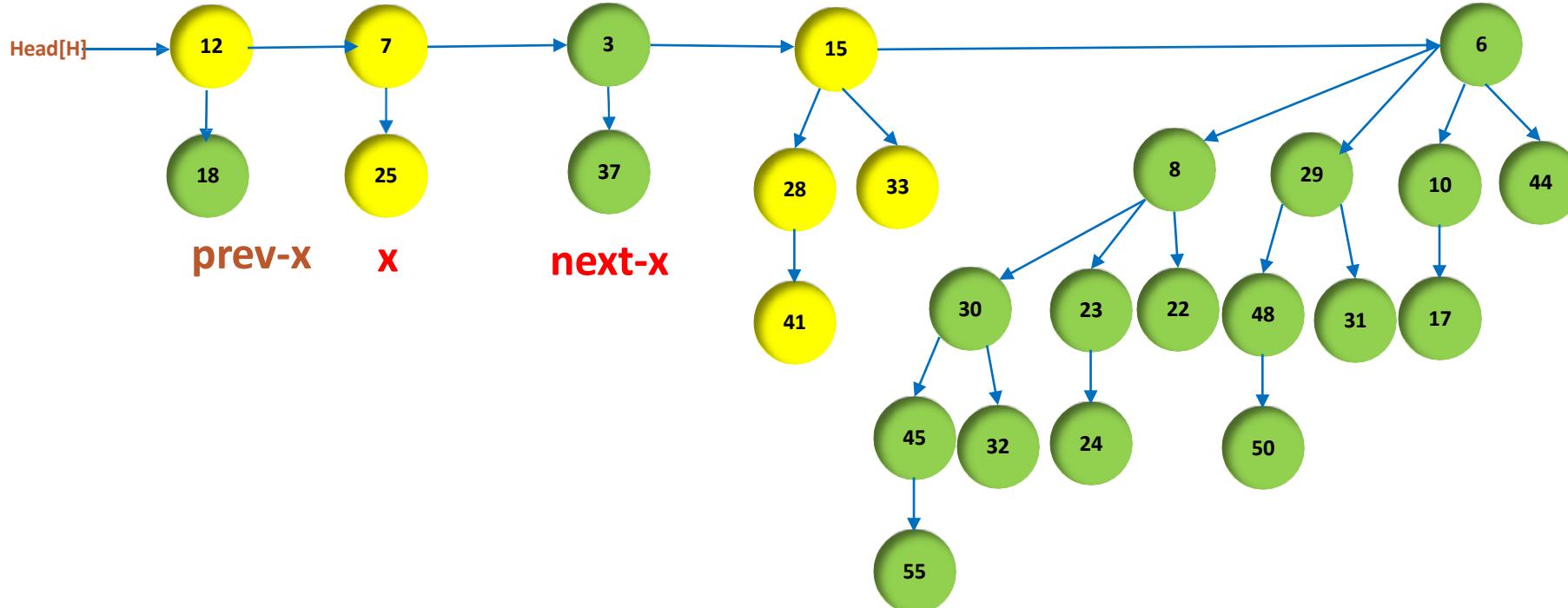


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$  or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
       and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ ))
      then  $\text{prev-}x \leftarrow x$                    ▷ Cases 1 and 2
            $x \leftarrow \text{next-}x$                    ▷ Cases 1 and 2
      else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
             then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$            ▷ Case 3
                    $\text{BINOMIAL-LINK}(\text{next-}x, x)$                    ▷ Case 3
             else if  $\text{prev-}x = \text{NIL}$ 
                   then  $\text{head}[H] \leftarrow \text{next-}x$                    ▷ Case 4
             else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$                    ▷ Case 4
                    $\text{BINOMIAL-LINK}(x, \text{next-}x)$                    ▷ Case 4
              $x \leftarrow \text{next-}x$                                    ▷ Case 4
              $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

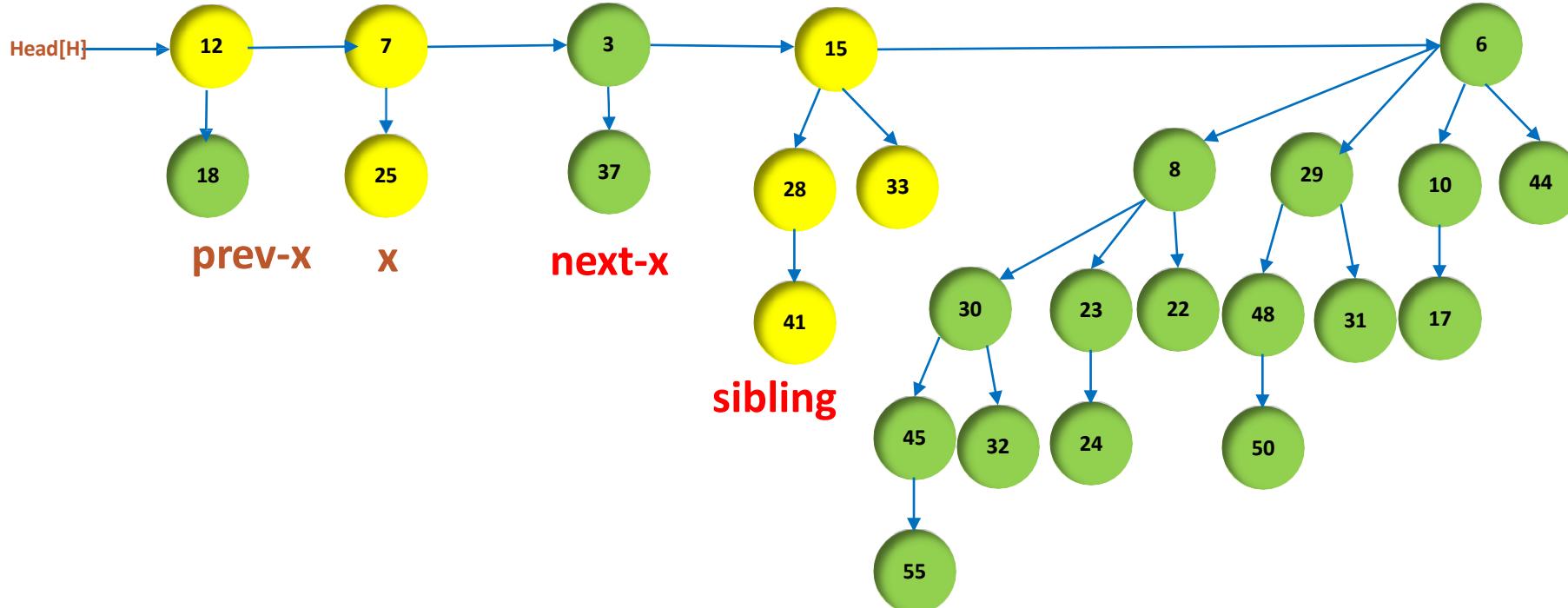


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
        then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
               $\text{BINOMIAL-LINK}(\text{next-}x, x)$            ▷ Case 3
        else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
            then  $\text{head}[H] \leftarrow \text{next-}x$           ▷ Case 4
        else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
               $\text{BINOMIAL-LINK}(x, \text{next-}x)$           ▷ Case 4
         $x \leftarrow \text{next-}x$                           ▷ Case 4
     $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

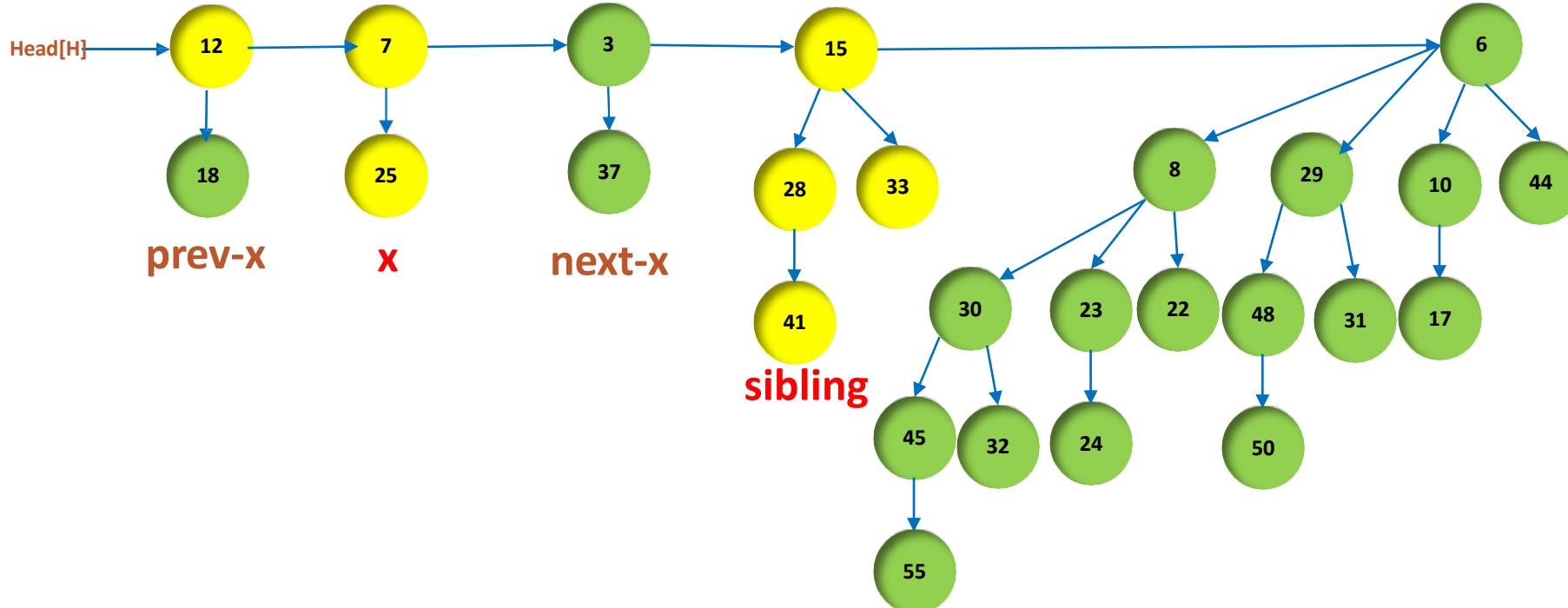


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
     $x \leftarrow \text{next-}x$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
    then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
     $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
    then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                                 ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

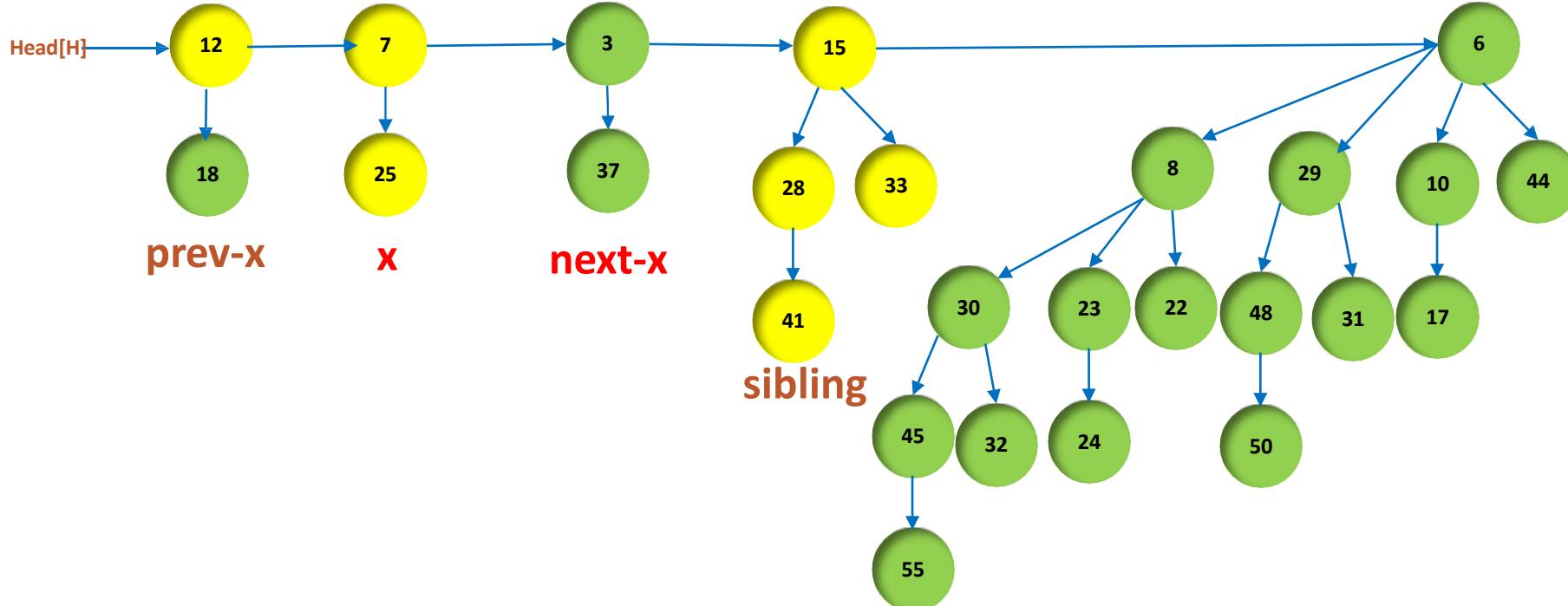


BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                                 ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 
    
```

# Binomial Union Operation...

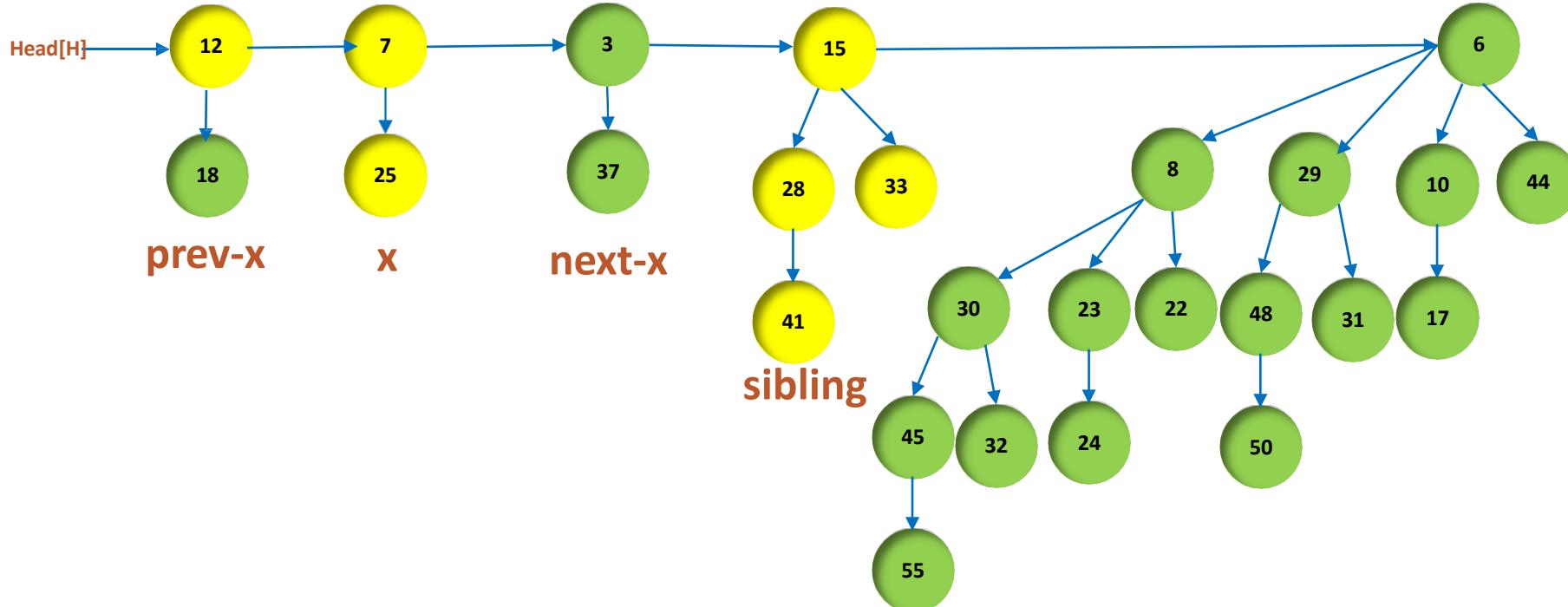


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
   ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
    and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

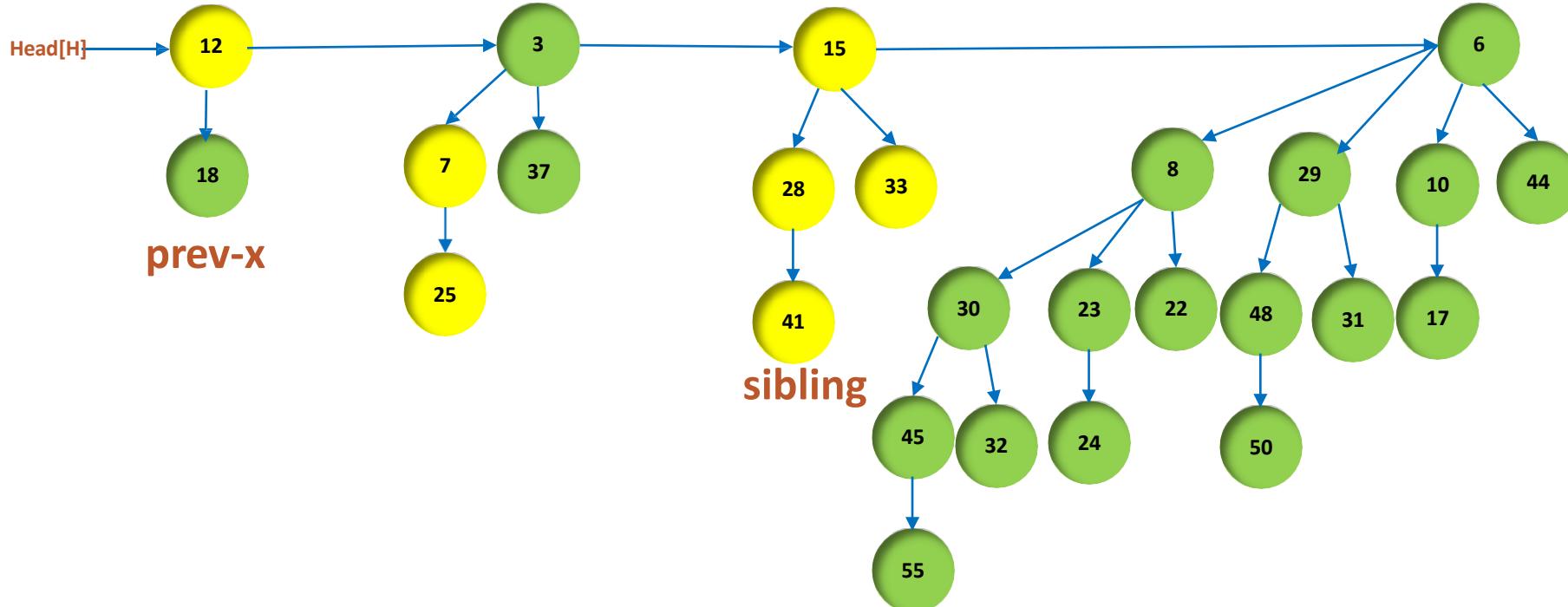


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$             ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$  ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                 ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

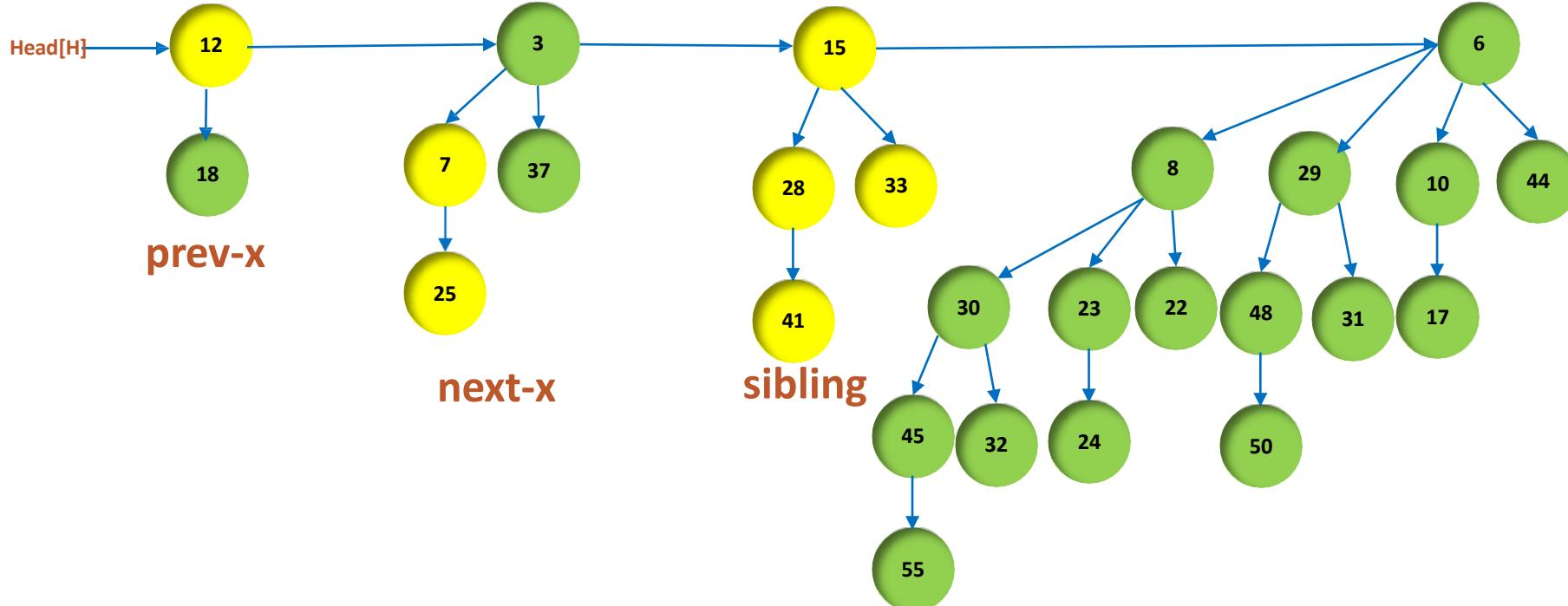


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                 ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

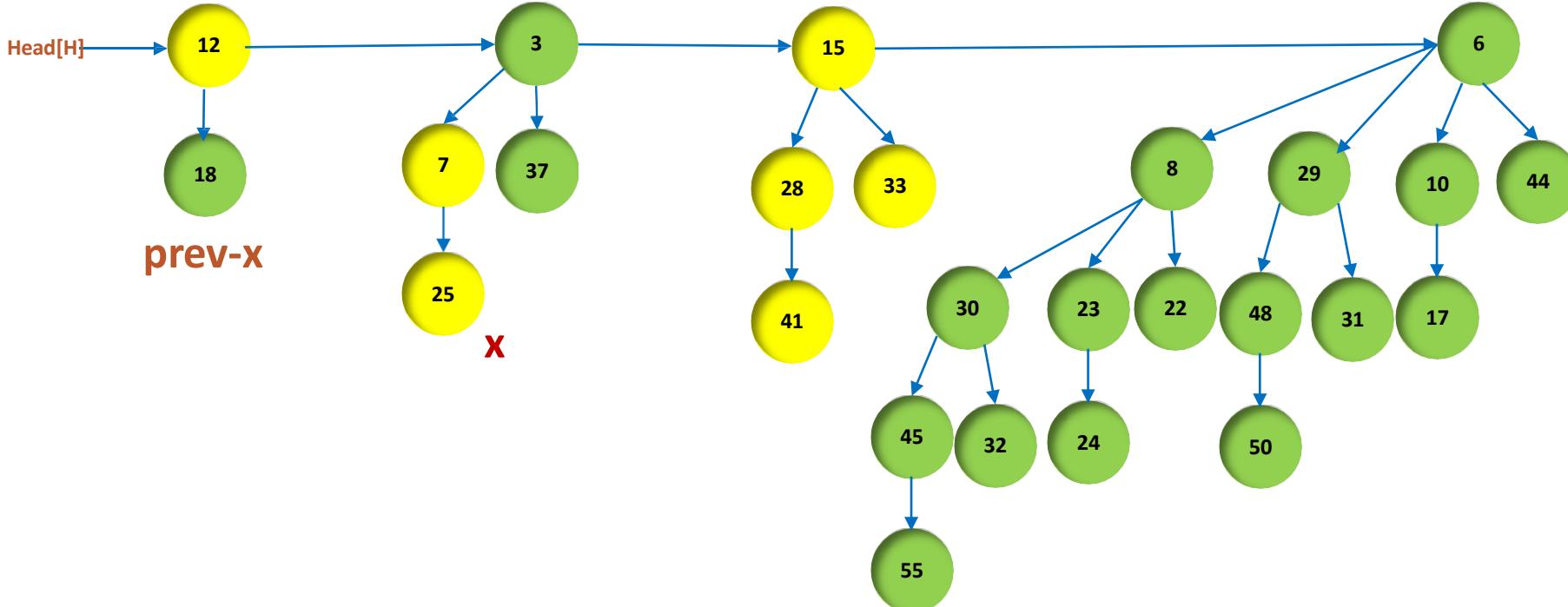


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
     ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                 ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

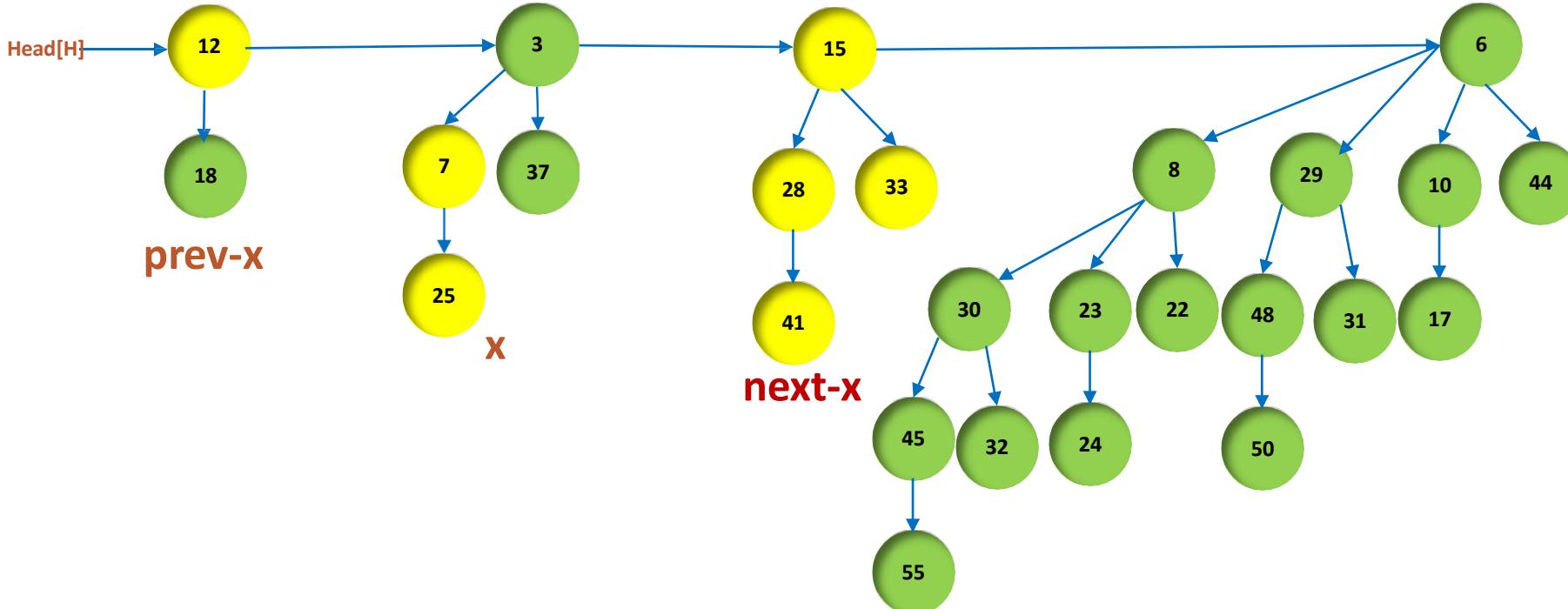


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{ sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{ sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
        then  $\text{ sibling}[x] \leftarrow \text{ sibling}[\text{next-}x]$       ▷ Case 3
               $\text{BINOMIAL-LINK}(\text{next-}x, x)$             ▷ Case 3
    else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
        then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
    else  $\text{ sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
           $\text{BINOMIAL-LINK}(x, \text{next-}x)$             ▷ Case 4
    20  $x \leftarrow \text{next-}x$                             ▷ Case 4
    21  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

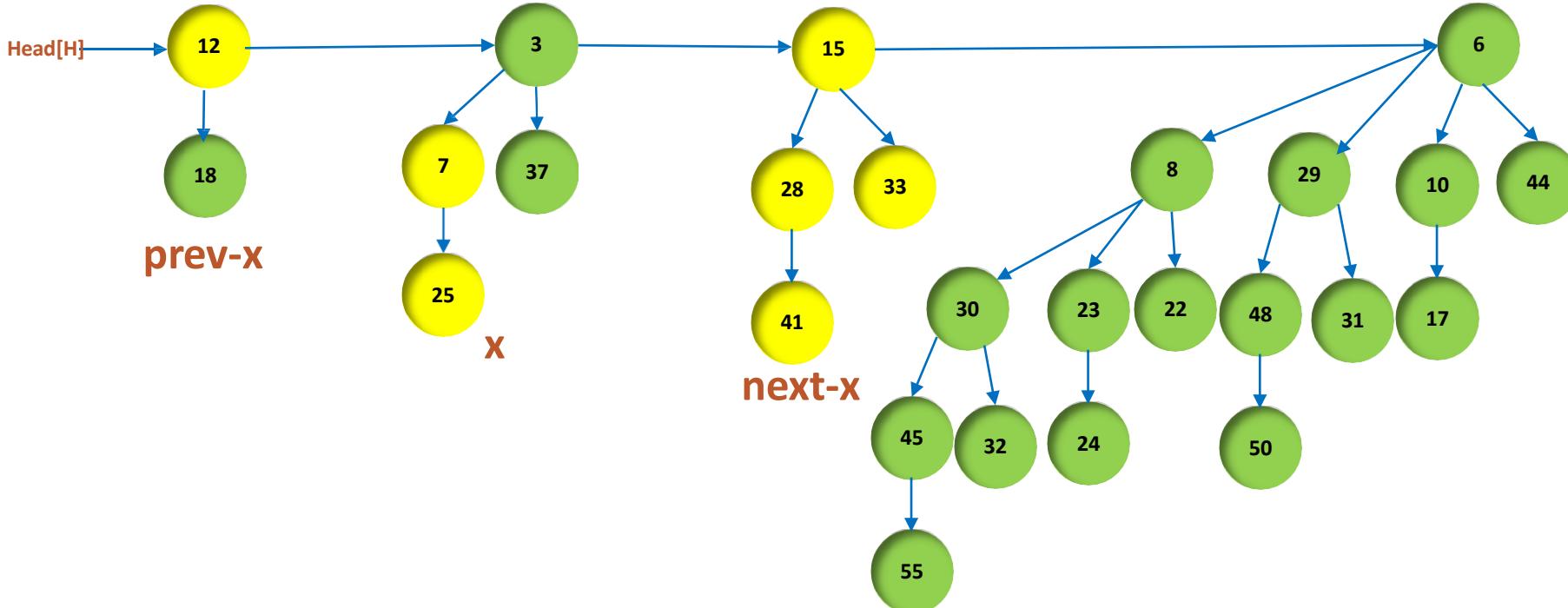


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if  $(\text{degree}[x] \neq \text{degree}[\text{next-}x])$  or
    $(\text{sibling}[\text{next-}x] \neq \text{NIL})$ 
   and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ 
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12    $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14   then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15    $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
17   then  $\text{head}[H] \leftarrow \text{next-}x$                   ▷ Case 4
18   else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$       ▷ Case 4
19    $\text{BINOMIAL-LINK}(x, \text{next-}x)$                  ▷ Case 4
20    $x \leftarrow \text{next-}x$                             ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

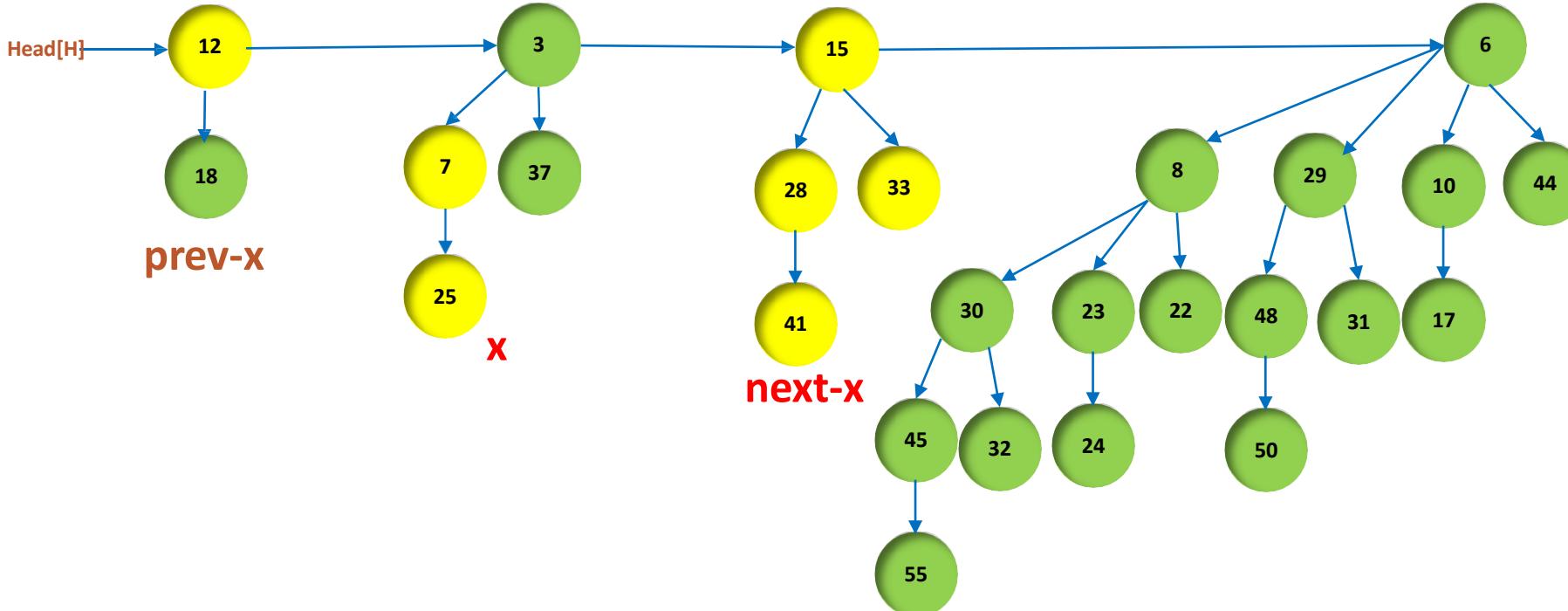


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev\_x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next\_x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next\_x} \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next\_x}]$ ) or
       ( $\text{sibling}[\text{next\_x}] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next\_x}]] = \text{degree}[x]$ )
    then  $\text{prev\_x} \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next\_x}$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next\_x}]$ 
        then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next\_x}]$       ▷ Case 3
              $\text{BINOMIAL-LINK}(\text{next\_x}, x)$             ▷ Case 3
        else if  $\text{prev\_x} = \text{NIL}$                       ▷ Case 4
            then  $\text{head}[H] \leftarrow \text{next\_x}$           ▷ Case 4
            else  $\text{sibling}[\text{prev\_x}] \leftarrow \text{next\_x}$  ▷ Case 4
                   $\text{BINOMIAL-LINK}(x, \text{next\_x})$       ▷ Case 4
                   $x \leftarrow \text{next\_x}$                   ▷ Case 4
                   $\text{next\_x} \leftarrow \text{sibling}[x]$            ▷ Case 4
22 return  $H$ 

```

# Binomial Union Operation...



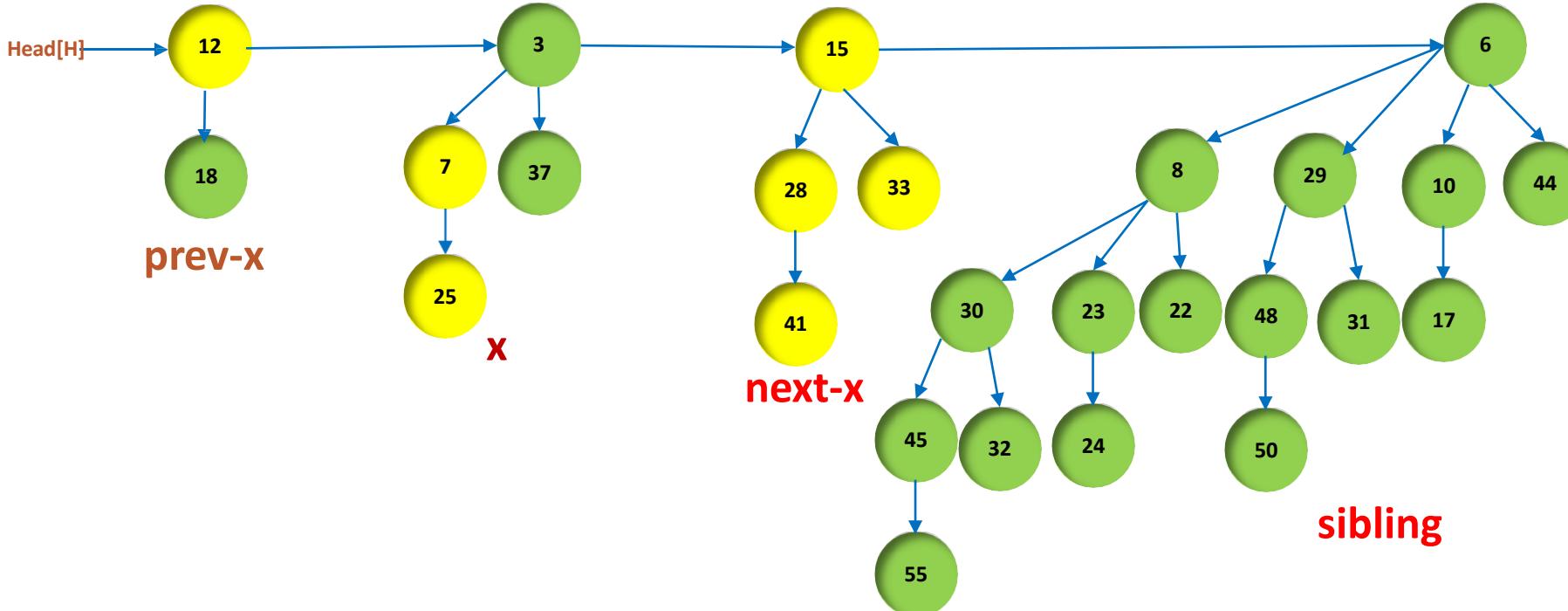
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



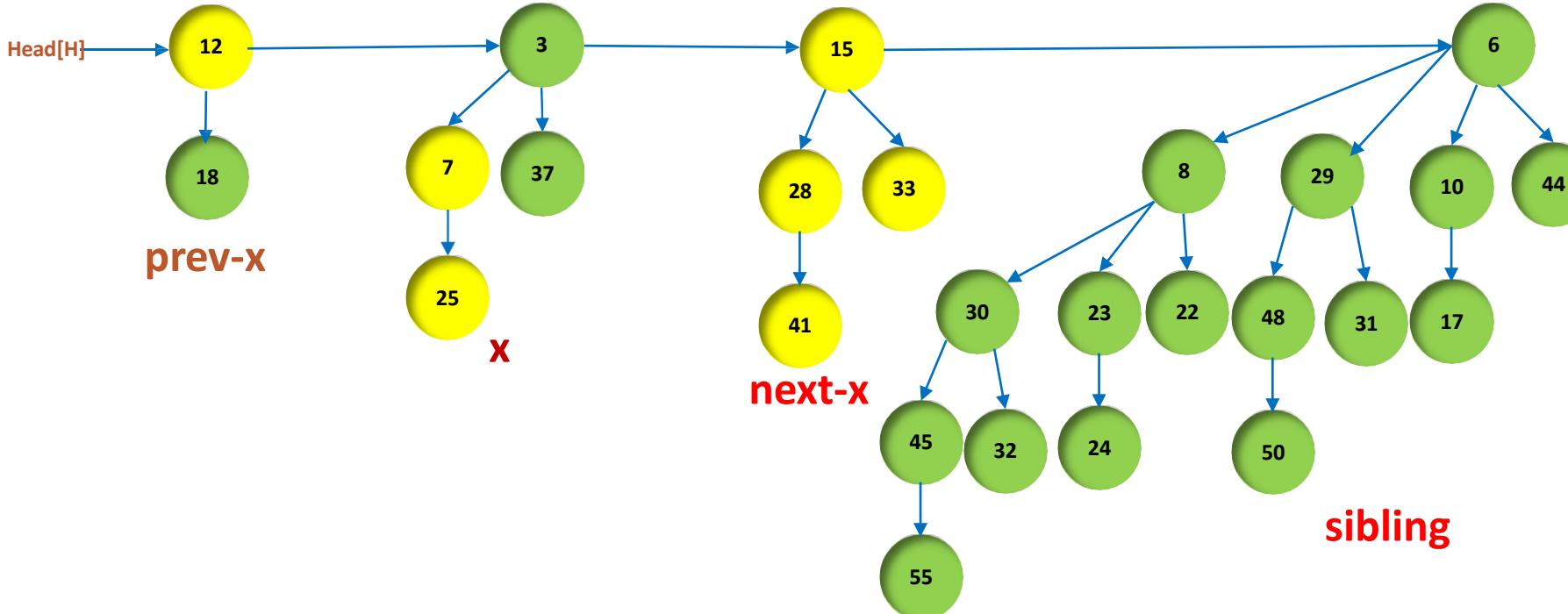
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev\_x} \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next\_x} \leftarrow \text{sibling}[x]$ 
9 while  $\text{next\_x} \neq \text{NIL}$ 
10 do if  $(\text{degree}[x] \neq \text{degree}[\text{next\_x}])$  or
     ( $\text{sibling}[\text{next\_x}] \neq \text{NIL}$ 
      and  $\text{degree}[\text{sibling}[\text{next\_x}]] = \text{degree}[x]$ )
11 then  $\text{prev\_x} \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next\_x}$                                 ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next\_x}]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next\_x}]$           ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next\_x}, x)$                   ▷ Case 3
16 else if  $\text{prev\_x} = \text{NIL}$                           ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next\_x}$                   ▷ Case 4
18 else  $\text{sibling}[\text{prev\_x}] \leftarrow \text{next\_x}$           ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next\_x})$                   ▷ Case 4
20  $x \leftarrow \text{next\_x}$                                 ▷ Case 4
21  $\text{next\_x} \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



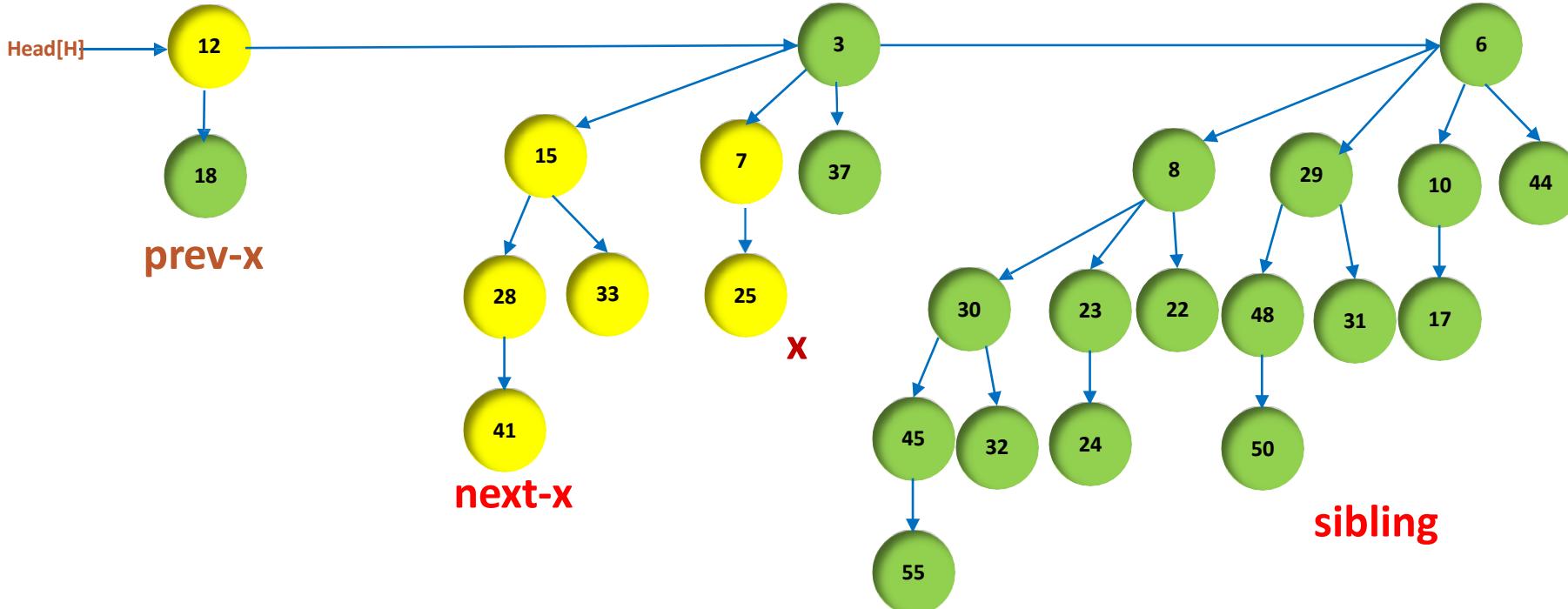
BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
   ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
    and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                  $\triangleright$  Cases 1 and 2
12  $x \leftarrow \text{next-}x$                                  $\triangleright$  Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$            $\triangleright$  Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                        $\triangleright$  Case 3
16 else if  $\text{prev-}x = \text{NIL}$                              $\triangleright$  Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                        $\triangleright$  Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$            $\triangleright$  Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                        $\triangleright$  Case 4
20  $x \leftarrow \text{next-}x$                                    $\triangleright$  Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

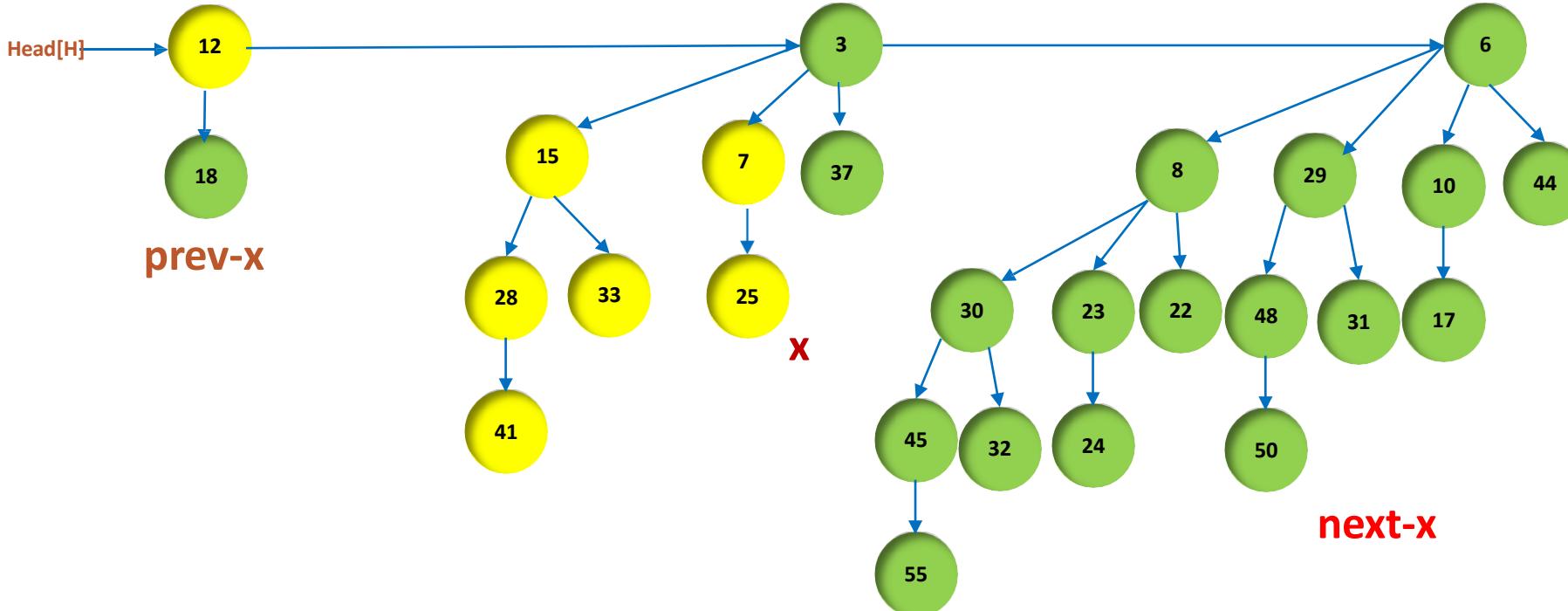


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
       and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] < \text{key}[\text{next-}x]$ 
        then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
               $\text{BINOMIAL-LINK}(\text{next-}x, x)$             ▷ Case 3
    16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
            then  $\text{head}[H] \leftarrow \text{next-}x$           ▷ Case 4
    18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
    19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
    20  $x \leftarrow \text{next-}x$                           ▷ Case 4
    21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

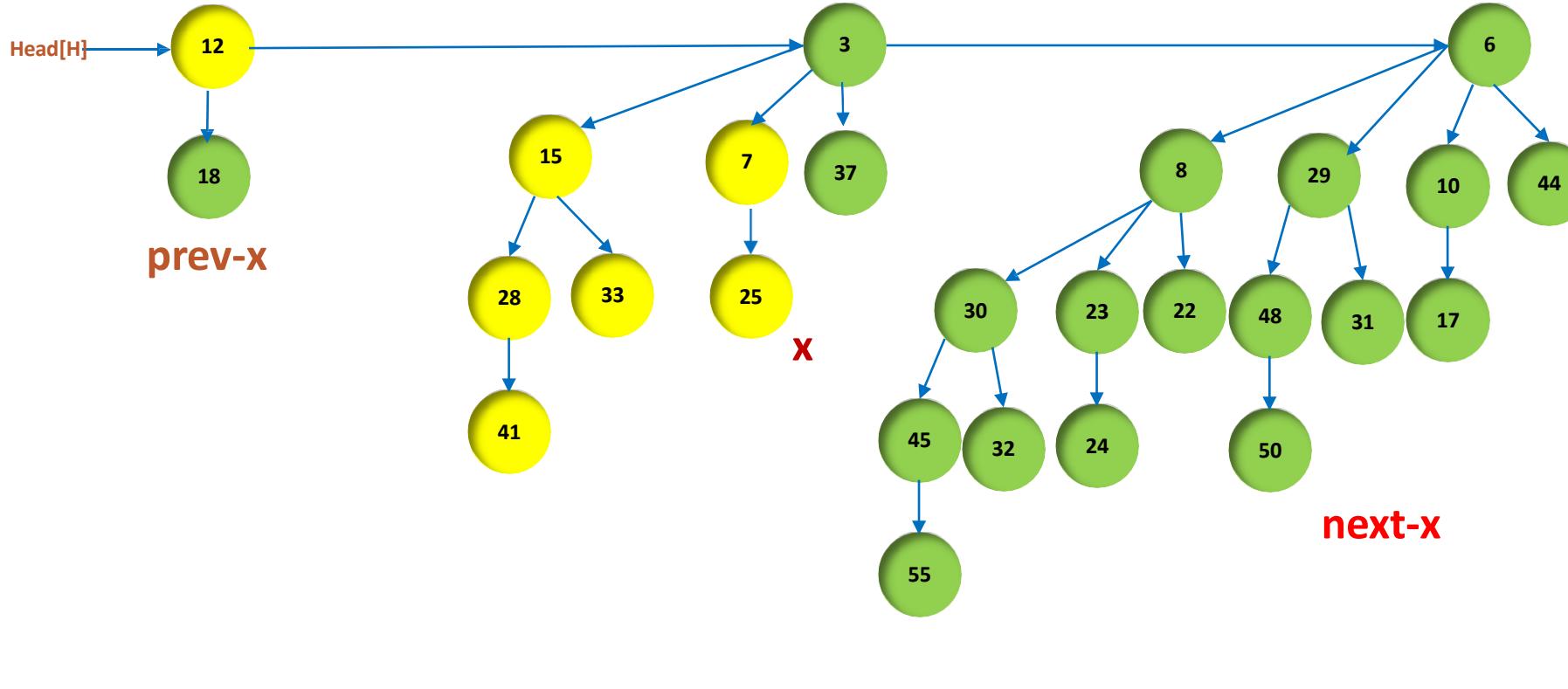


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
    then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
          $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
    else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
        then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$       ▷ Case 3
               $\text{BINOMIAL-LINK}(\text{next-}x, x)$             ▷ Case 3
        else  $\text{prev-}x = \text{NIL}$                          ▷ Case 4
              then  $\text{head}[H] \leftarrow \text{next-}x$           ▷ Case 4
              else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$  ▷ Case 4
               $\text{BINOMIAL-LINK}(x, \text{next-}x)$            ▷ Case 4
               $x \leftarrow \text{next-}x$                          ▷ Case 4
21    $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...

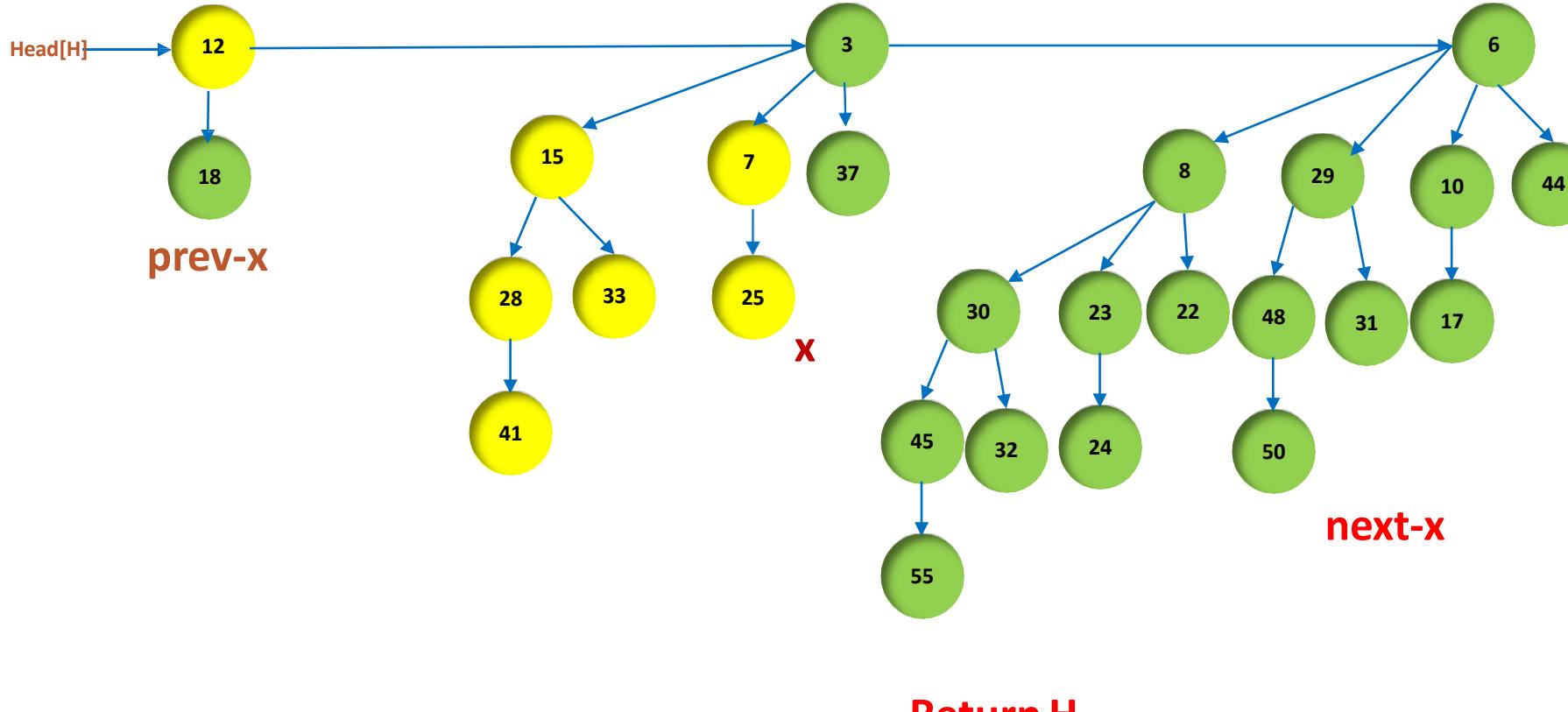


```

BINOMIAL-HEAP-UNION( $H_1, H_2$ )
1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{sibling}[\text{next-}x] \neq \text{NIL}$ 
       and  $\text{degree}[\text{sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{sibling}[x] \leftarrow \text{sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                 ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                 ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{sibling}[x]$ 
22 return  $H$ 

```

# Binomial Union Operation...



BINOMIAL-HEAP-UNION( $H_1, H_2$ )

```

1  $H \leftarrow \text{MAKE-BINOMIAL-HEAP}()$ 
2  $\text{head}[H] \leftarrow \text{BINOMIAL-HEAP-MERGE}(H_1, H_2)$ 
3 free the objects  $H_1$  and  $H_2$  but not the lists they point to
4 if  $\text{head}[H] = \text{NIL}$ 
5 then return  $H$ 
6  $\text{prev-}x \leftarrow \text{NIL}$ 
7  $x \leftarrow \text{head}[H]$ 
8  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
9 while  $\text{next-}x \neq \text{NIL}$ 
10 do if ( $\text{degree}[x] \neq \text{degree}[\text{next-}x]$ ) or
       ( $\text{ sibling}[\text{next-}x] \neq \text{NIL}$ 
        and  $\text{degree}[\text{ sibling}[\text{next-}x]] = \text{degree}[x]$ )
11 then  $\text{prev-}x \leftarrow x$                                 ▷ Cases 1 and 2
12  $x \leftarrow \text{next-}x$                             ▷ Cases 1 and 2
13 else if  $\text{key}[x] \leq \text{key}[\text{next-}x]$ 
14 then  $\text{ sibling}[x] \leftarrow \text{ sibling}[\text{next-}x]$     ▷ Case 3
15  $\text{BINOMIAL-LINK}(\text{next-}x, x)$                   ▷ Case 3
16 else if  $\text{prev-}x = \text{NIL}$                       ▷ Case 4
17 then  $\text{head}[H] \leftarrow \text{next-}x$                 ▷ Case 4
18 else  $\text{ sibling}[\text{prev-}x] \leftarrow \text{next-}x$     ▷ Case 4
19  $\text{BINOMIAL-LINK}(x, \text{next-}x)$                   ▷ Case 4
20  $x \leftarrow \text{next-}x$                             ▷ Case 4
21  $\text{next-}x \leftarrow \text{ sibling}[x]$ 
22 return  $H$ 

```



## Design and Analysis of Algorithms

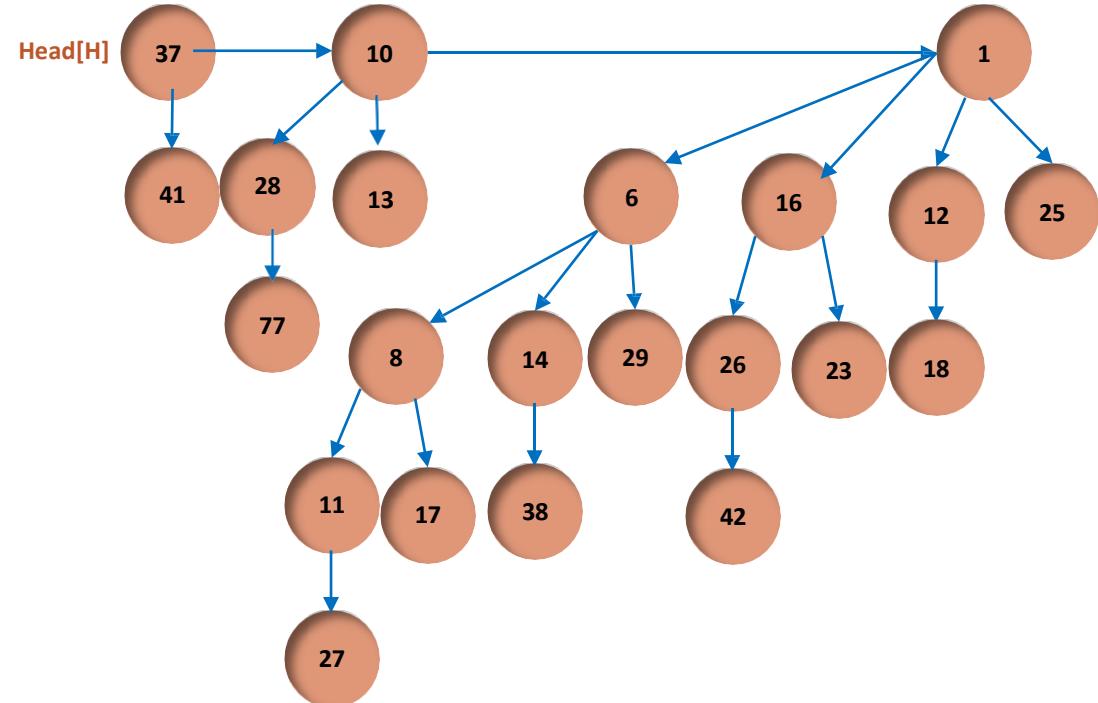
# Binomial Heap – Extracting Node of Min. Key

---

# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

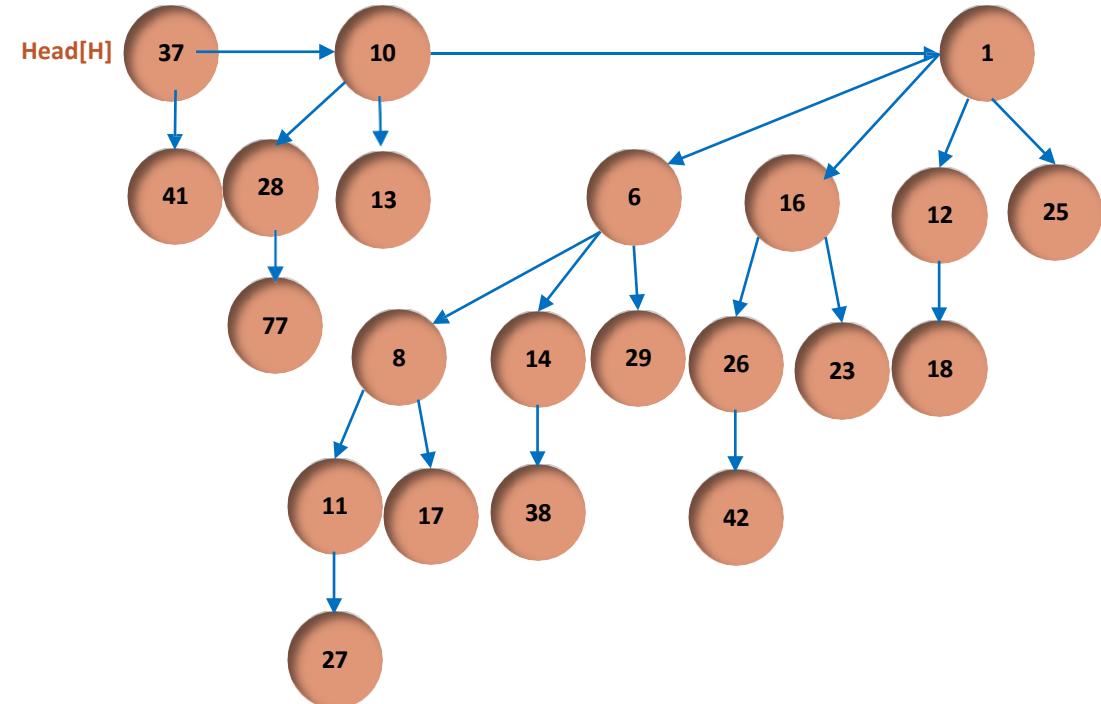
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$
2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$
3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list
4.  $H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



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

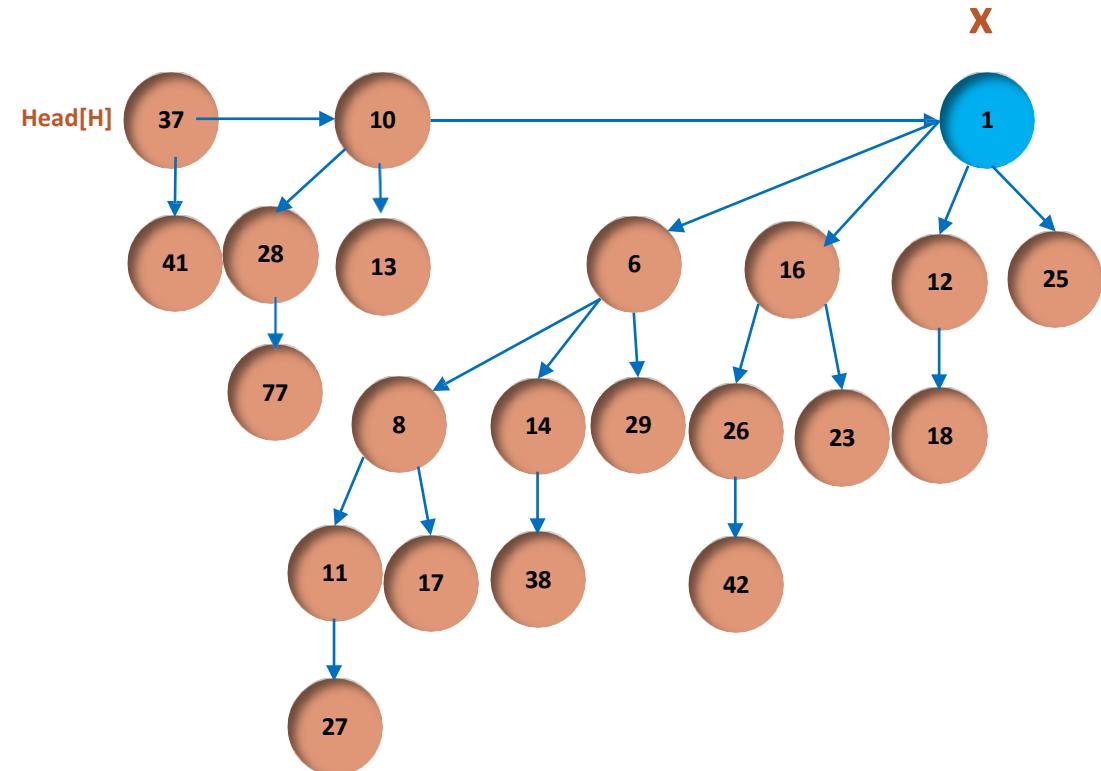
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$
2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$
3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list
4.  $H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



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

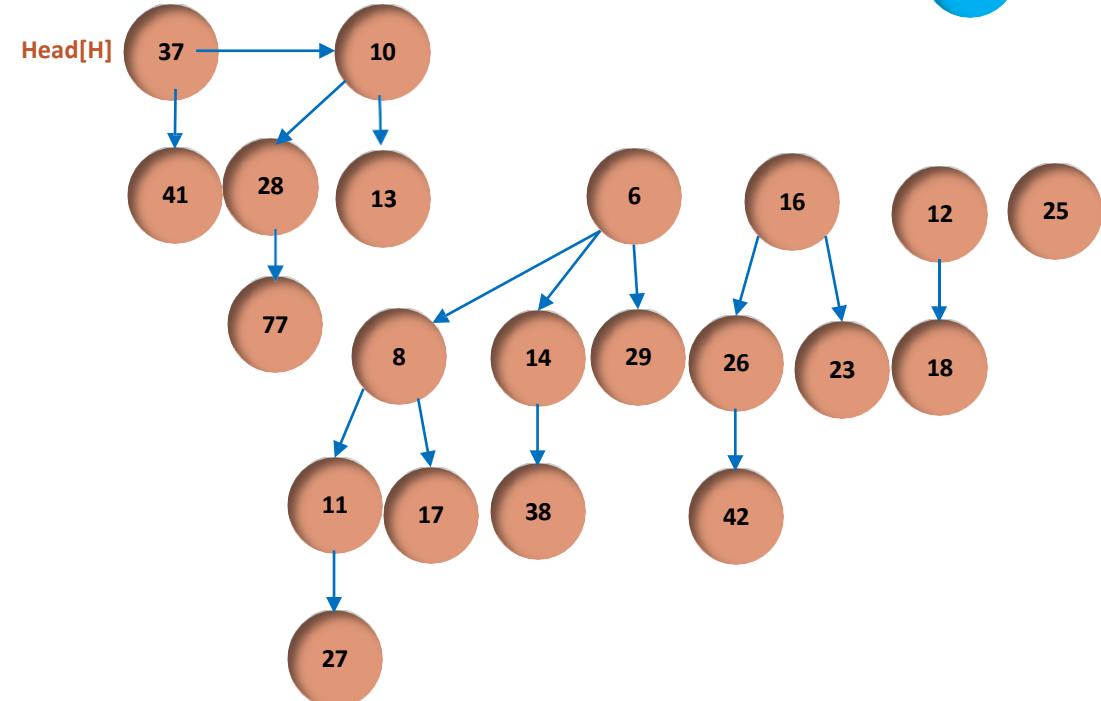
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$
2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$
3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list
4.  $H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



# Extracting Node with Min. 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.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$
3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list
4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$
5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

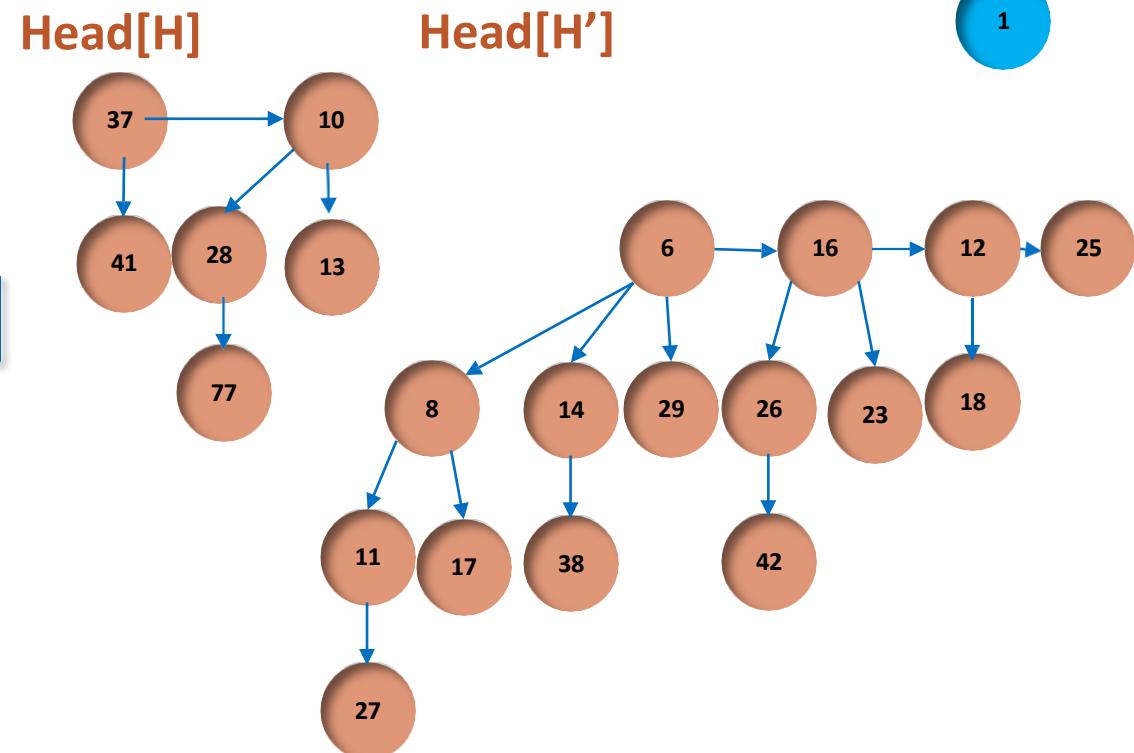
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

### 2. $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

### 4. $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

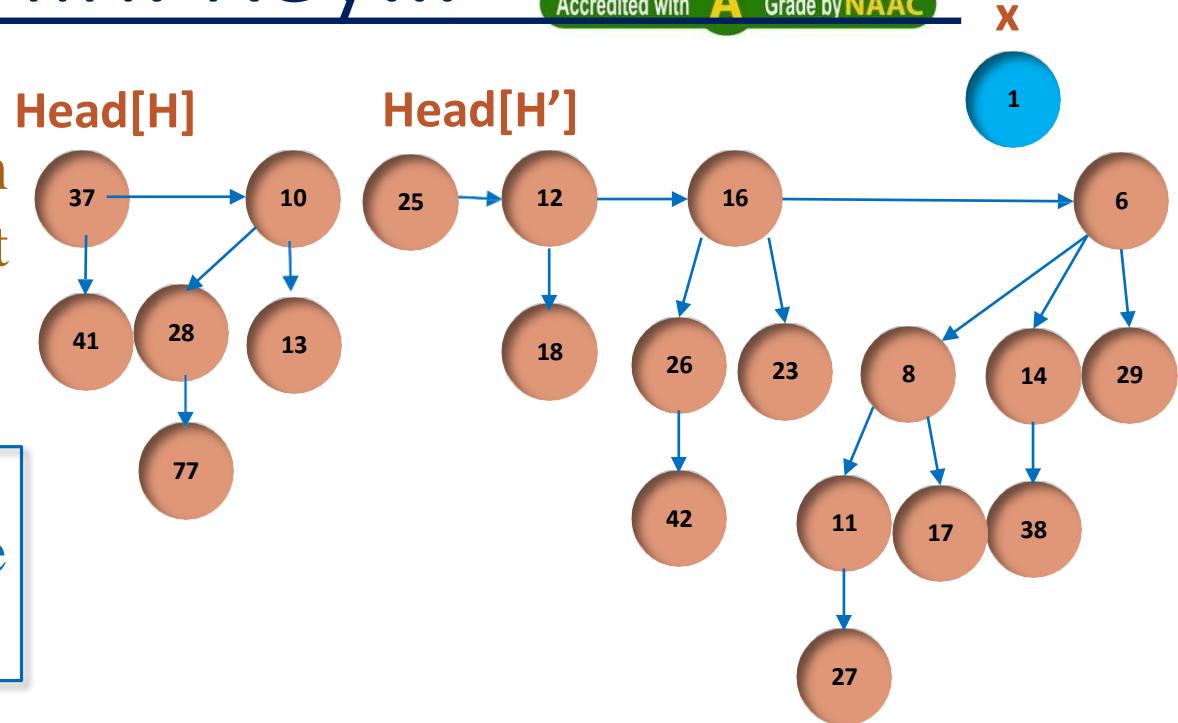
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

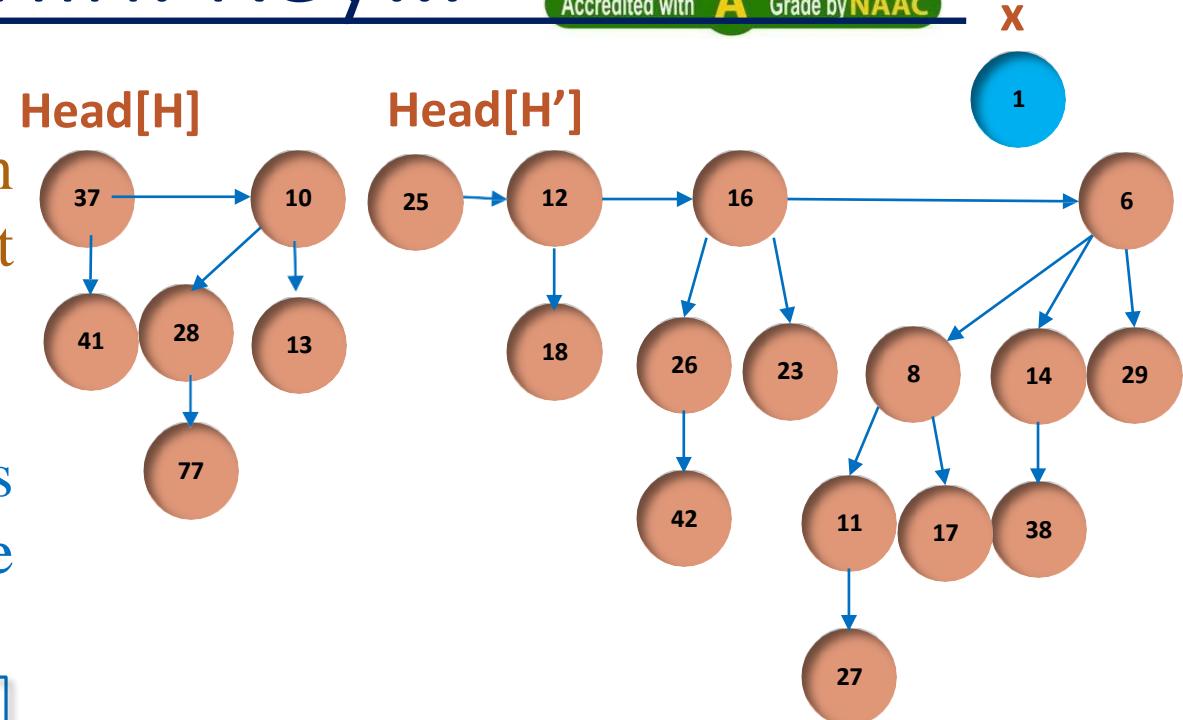
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## **BINOMIAL-HEAP-EXTRACT-MIN( $H$ )**

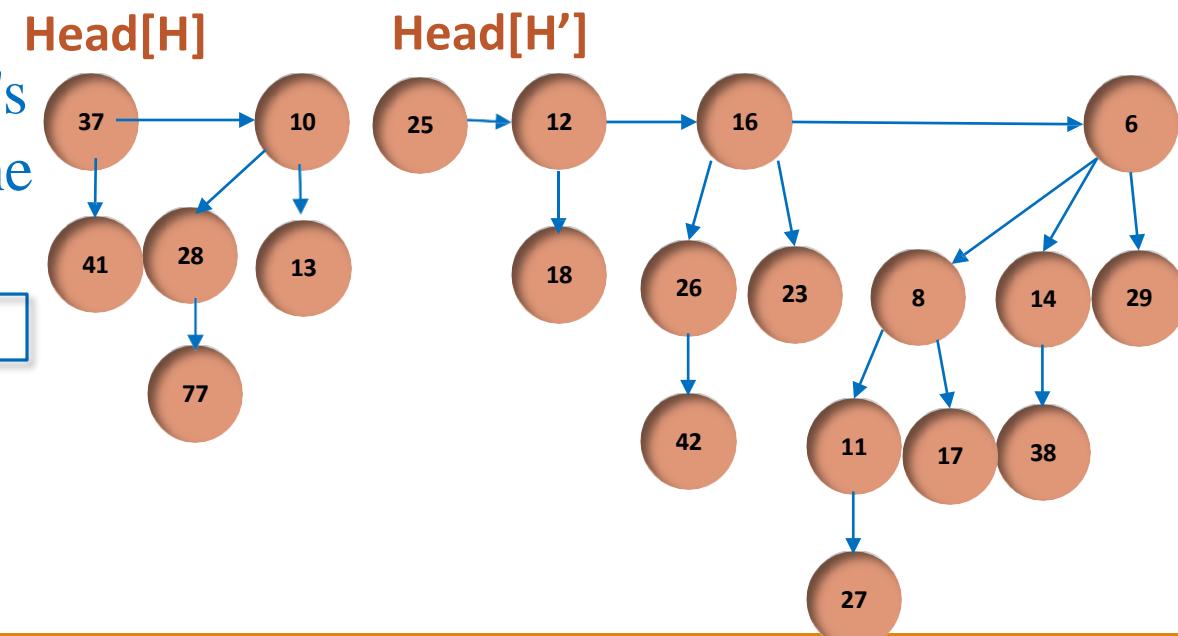
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

### **2. $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$**

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

### **4. $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$**

5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

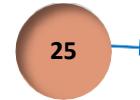
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

### 2. $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

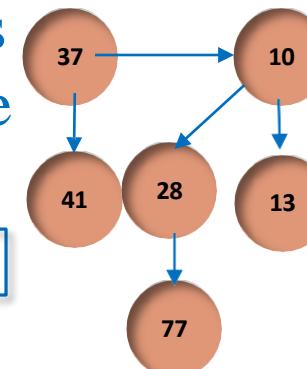
3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

### 4. $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

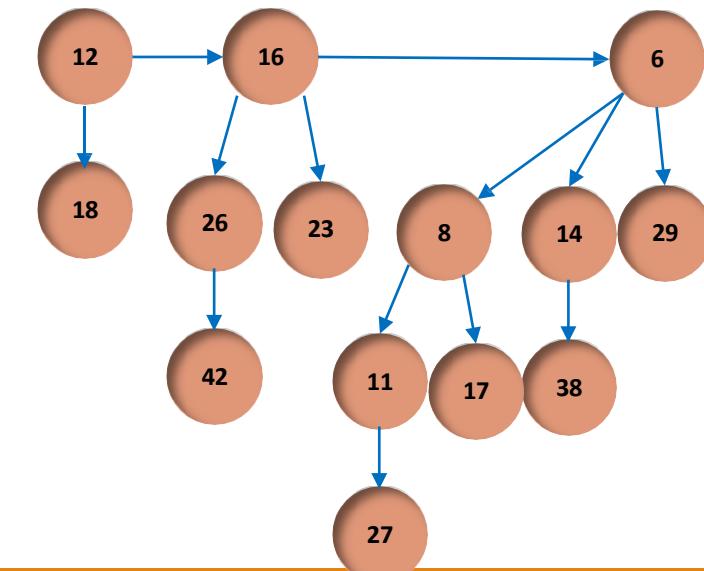
5. return  $x$



**Head[ $H$ ]**



**Head[ $H'$ ]**



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

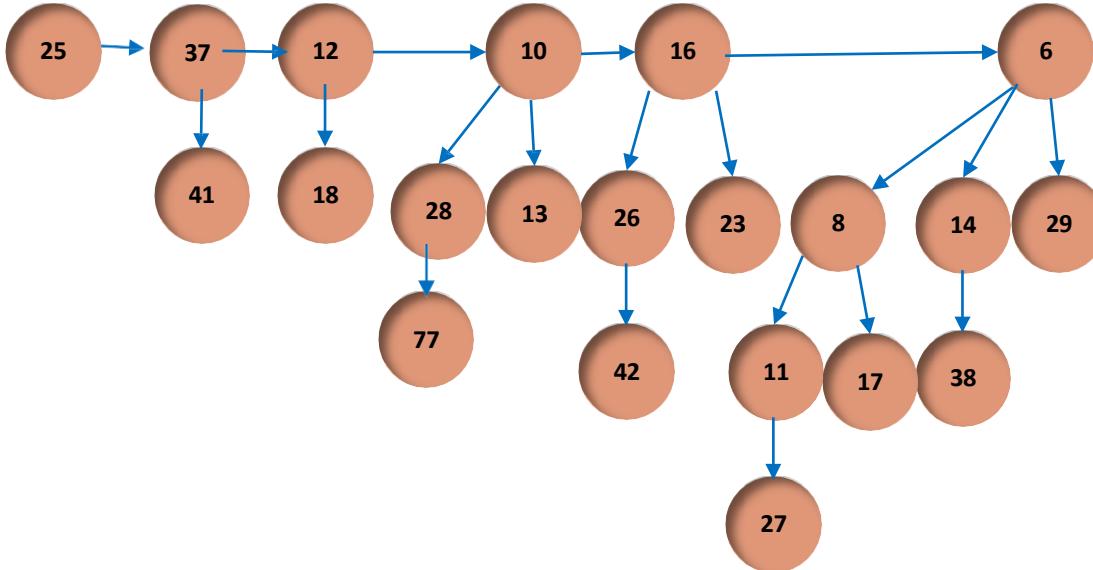
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## **BINOMIAL-HEAP-EXTRACT-MIN( $H$ )**

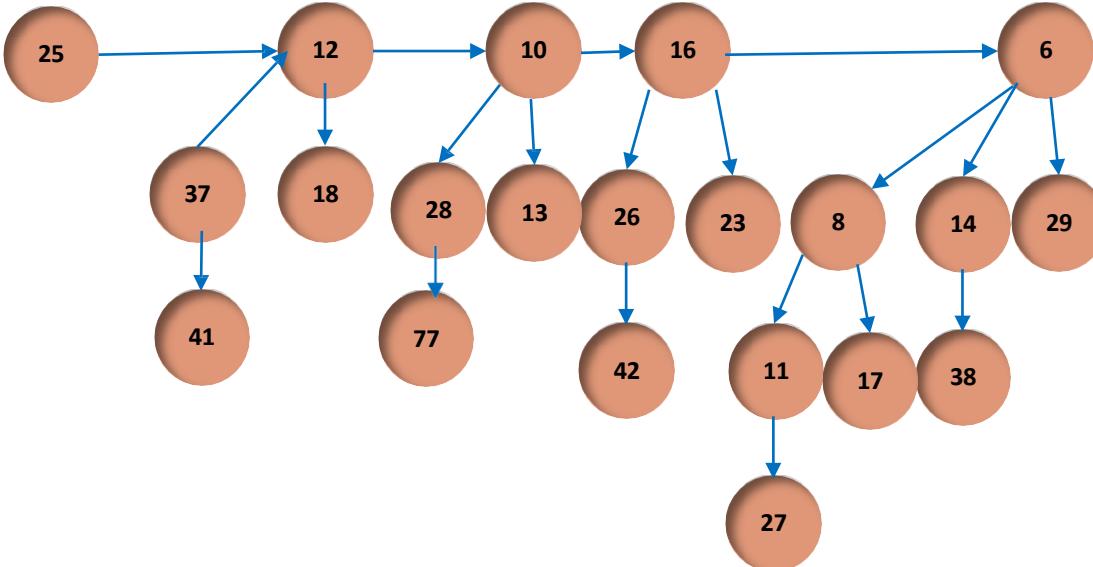
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## **BINOMIAL-HEAP-EXTRACT-MIN( $H$ )**

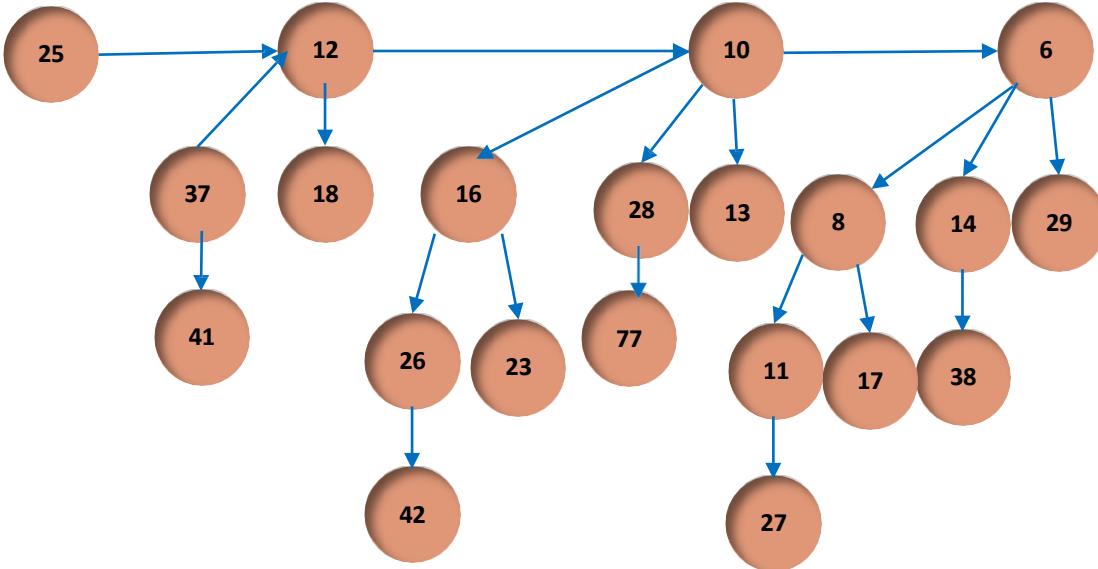
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

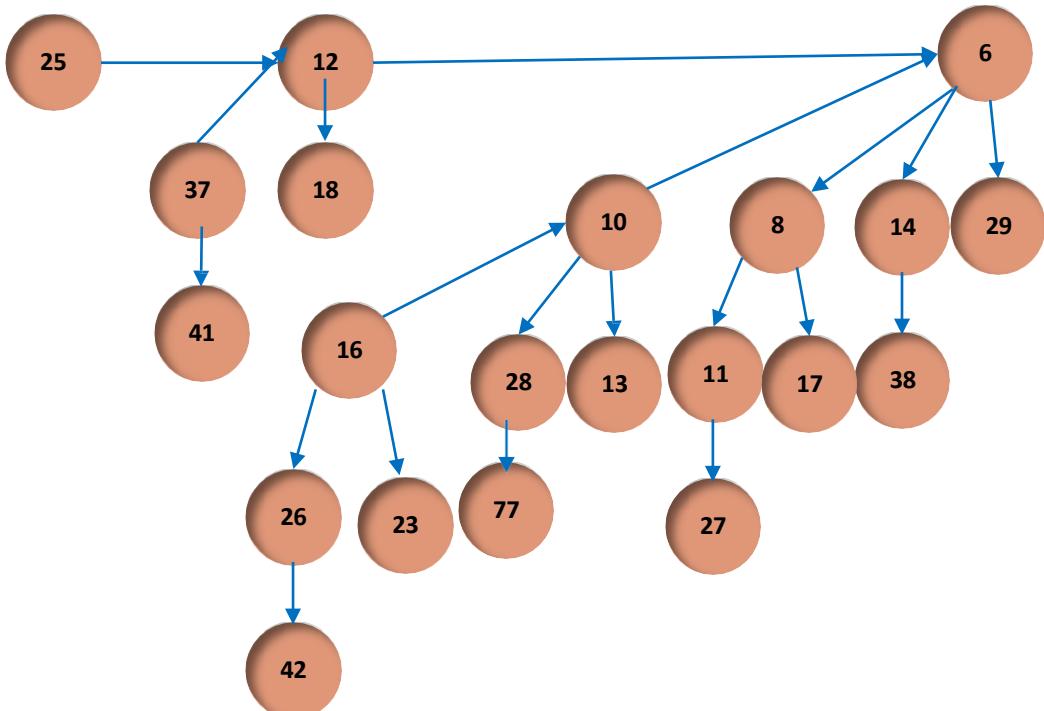
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$



# Extracting Node with Min. Key...

## BINOMIAL-HEAP-EXTRACT-MIN( $H$ )

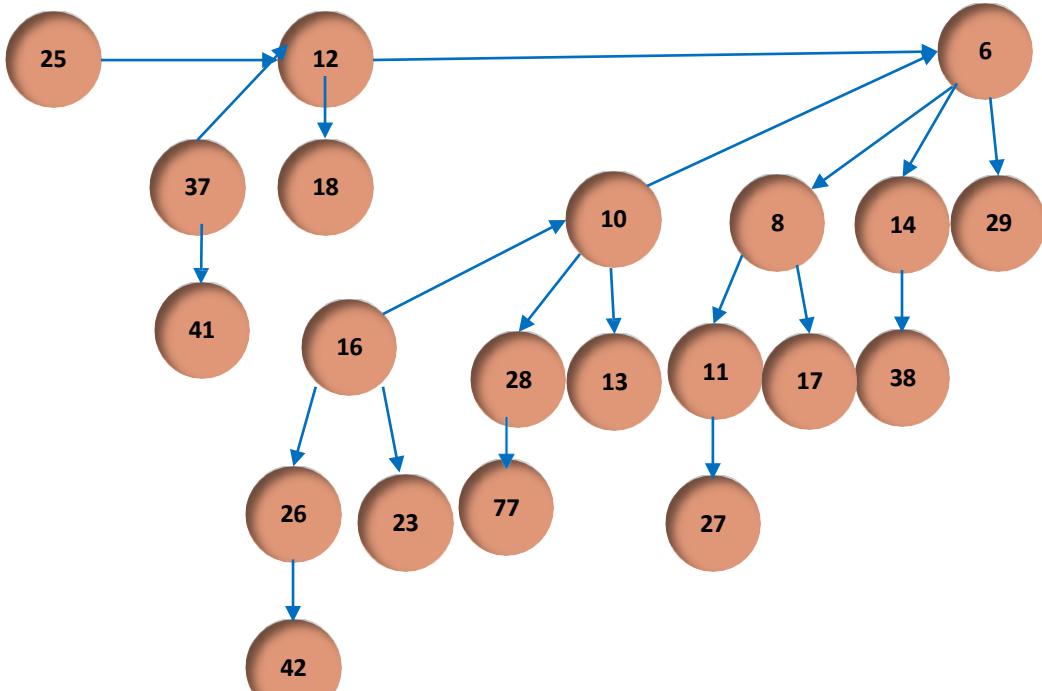
1. find the root  $x$  with the minimum key in the root list of  $H$  and remove  $x$  from the root list of  $H$

2.  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$

3. reverse the order of the linked list of  $x$ 's children, and set  $\text{head}[H']$  to point to the head of the resulting list

4.  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

5. return  $x$





## Design and Analysis of Algorithms

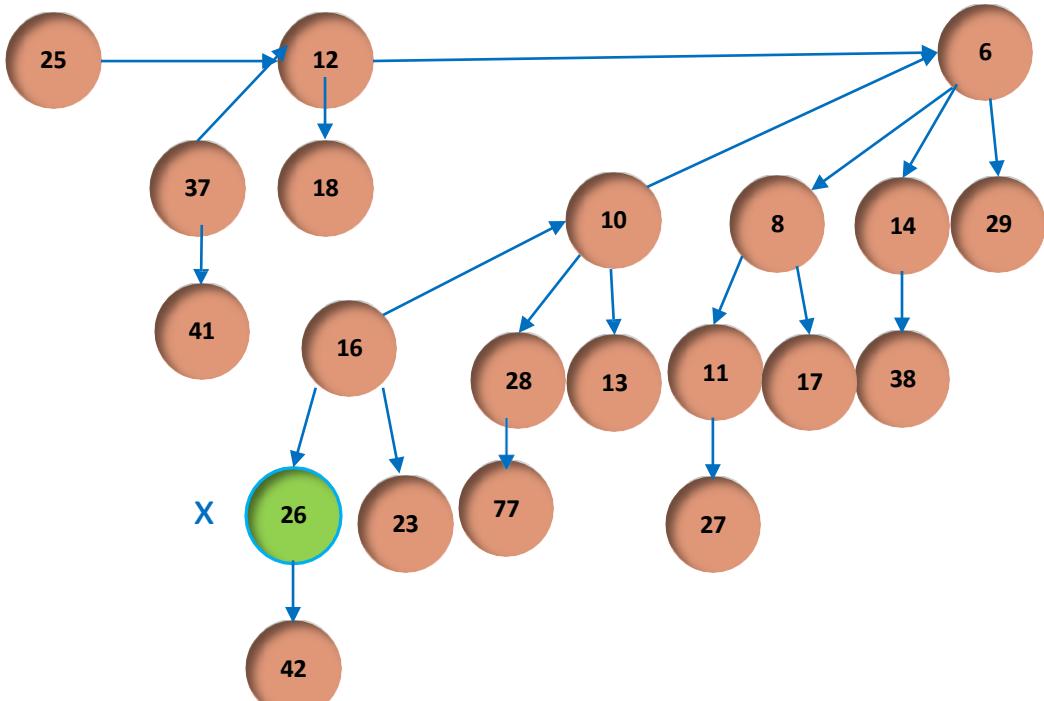
# Binomial Heap – Decrease Key

---

# Decrease Key

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

```
1 if  $k > \text{key}[x]$ 
2 then error "new key is greater than current key"
3  $\text{key}[x] \leftarrow k$ 
4  $y \leftarrow x$ 
5  $z \leftarrow p[y]$ 
6 while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7 do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8 ▷ If  $y$  and  $z$  have satellite fields, exchange them, too.
9  $y \leftarrow z$ 
10  $z \leftarrow p[y]$ 
```



Decrease x=26 with k=7

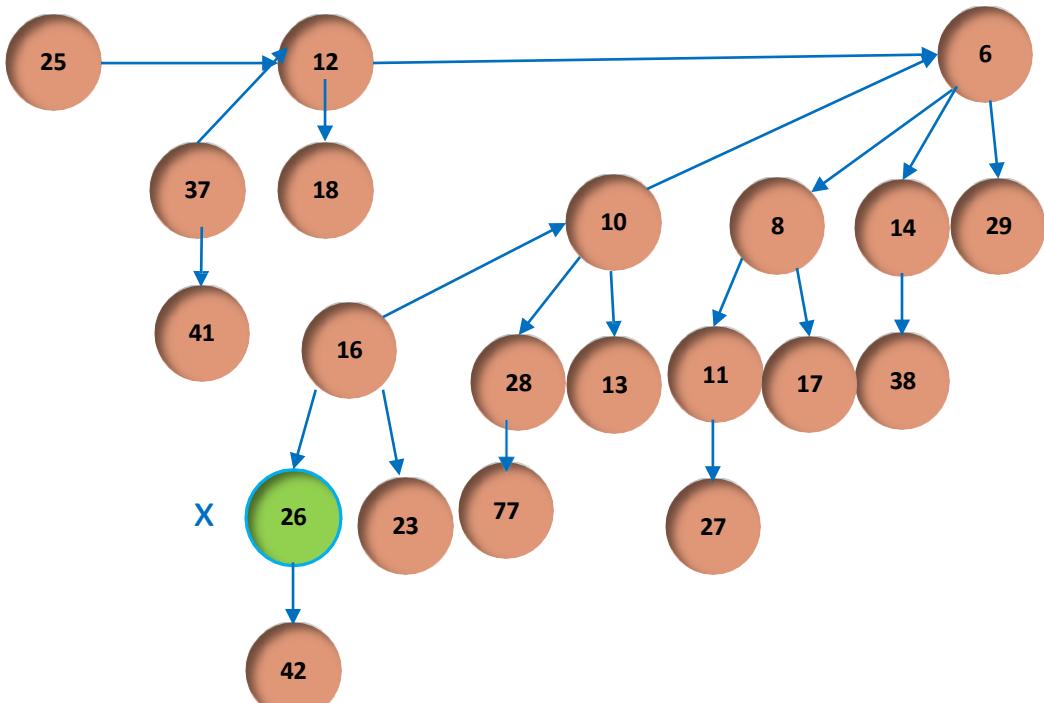
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$            k> key[x]  7>26
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

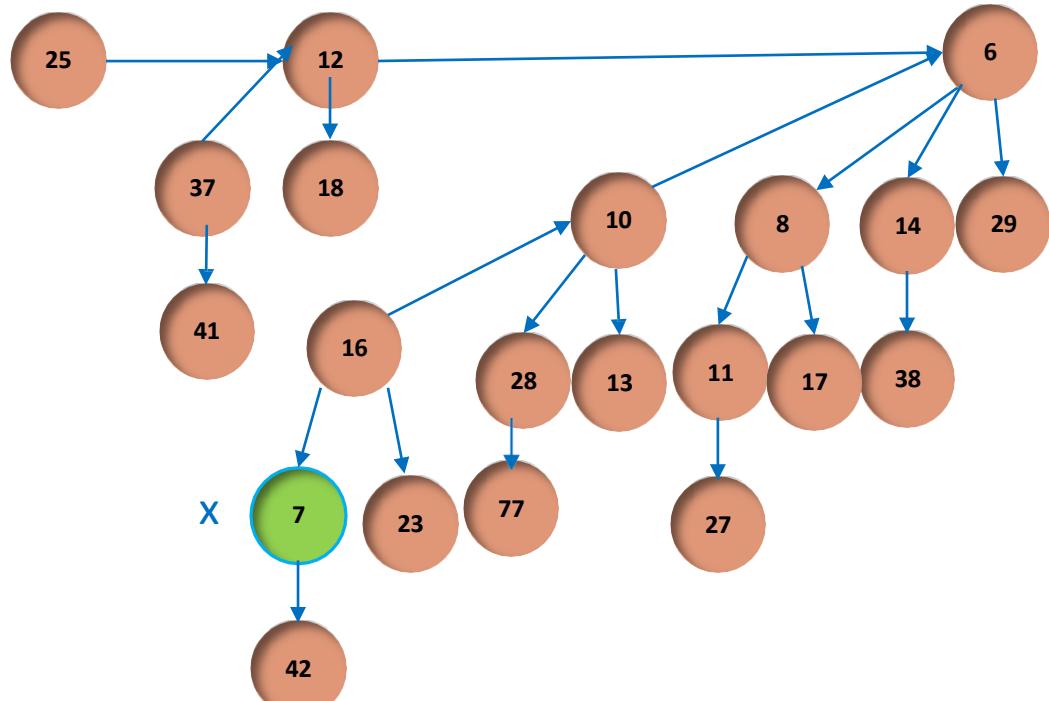
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



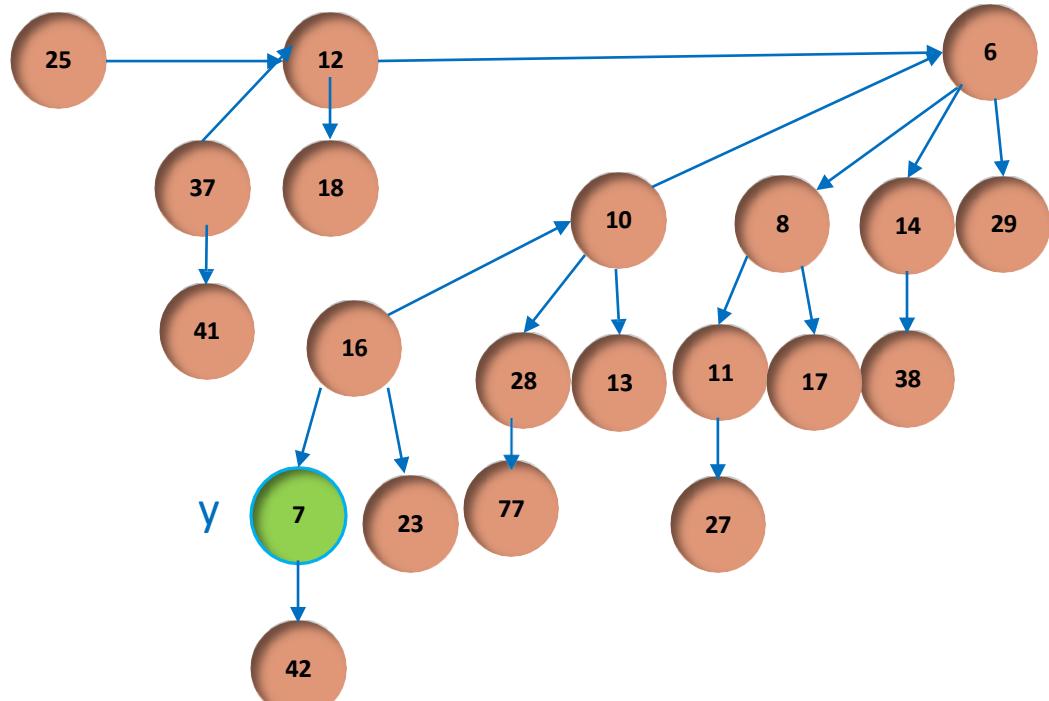
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

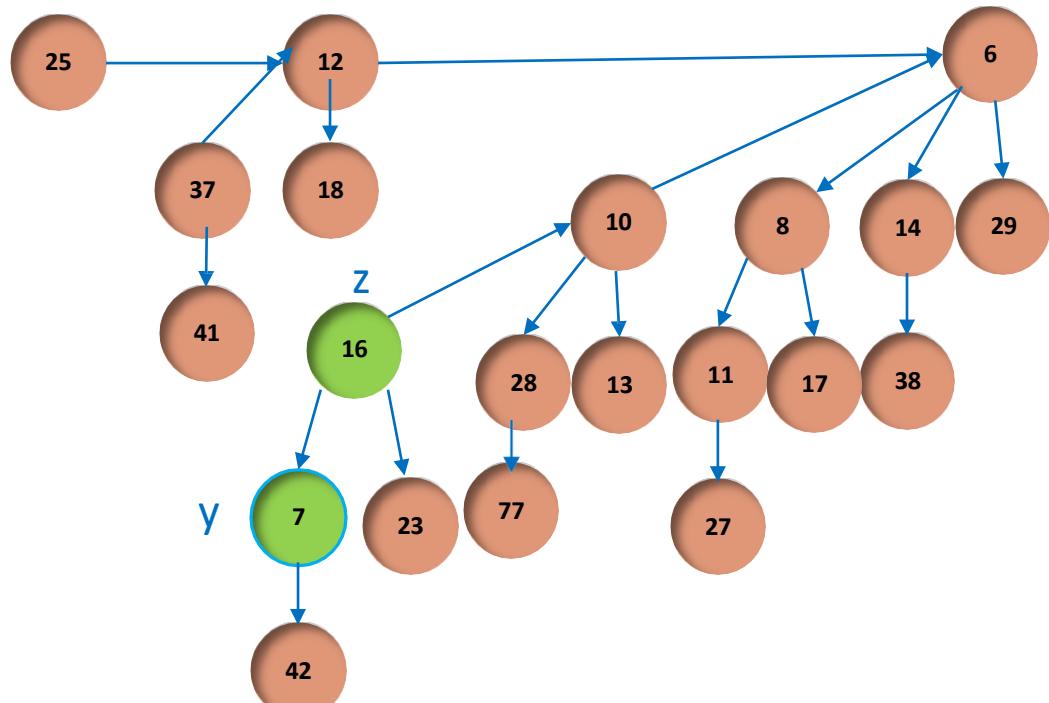
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



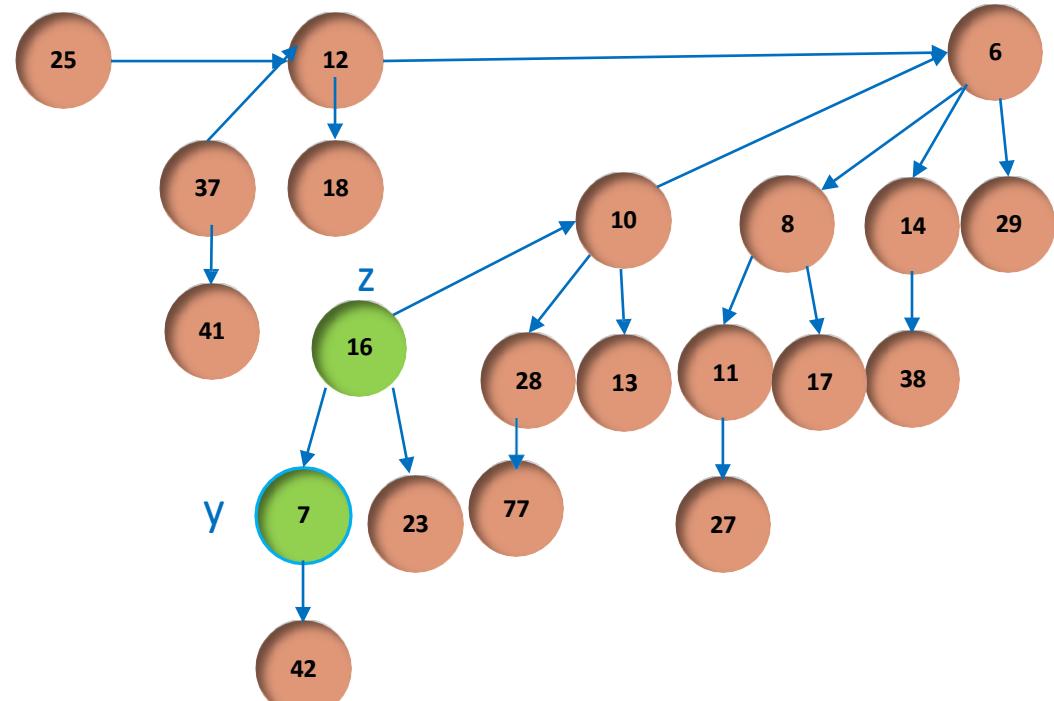
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$    k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$  True and true
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If  $y$  and  $z$  have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

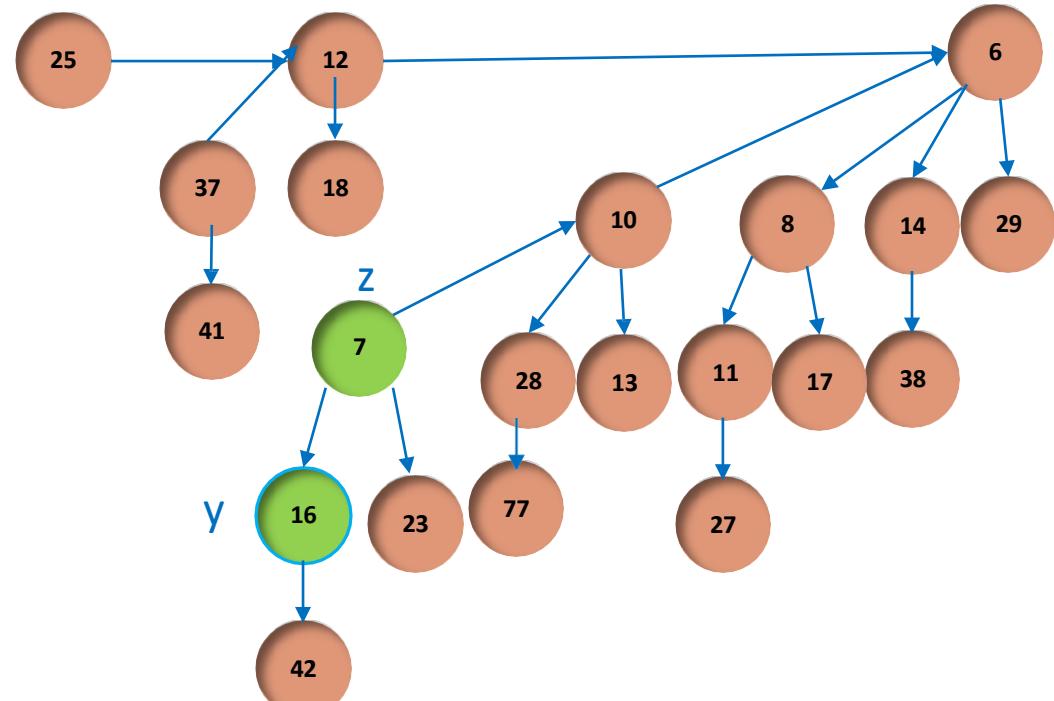
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$  Exchange 7 and 16
8      ▷ If y and z have satellite fields, exchange them, too.
9       $y \leftarrow z$ 
10      $z \leftarrow p[y]$ 

```



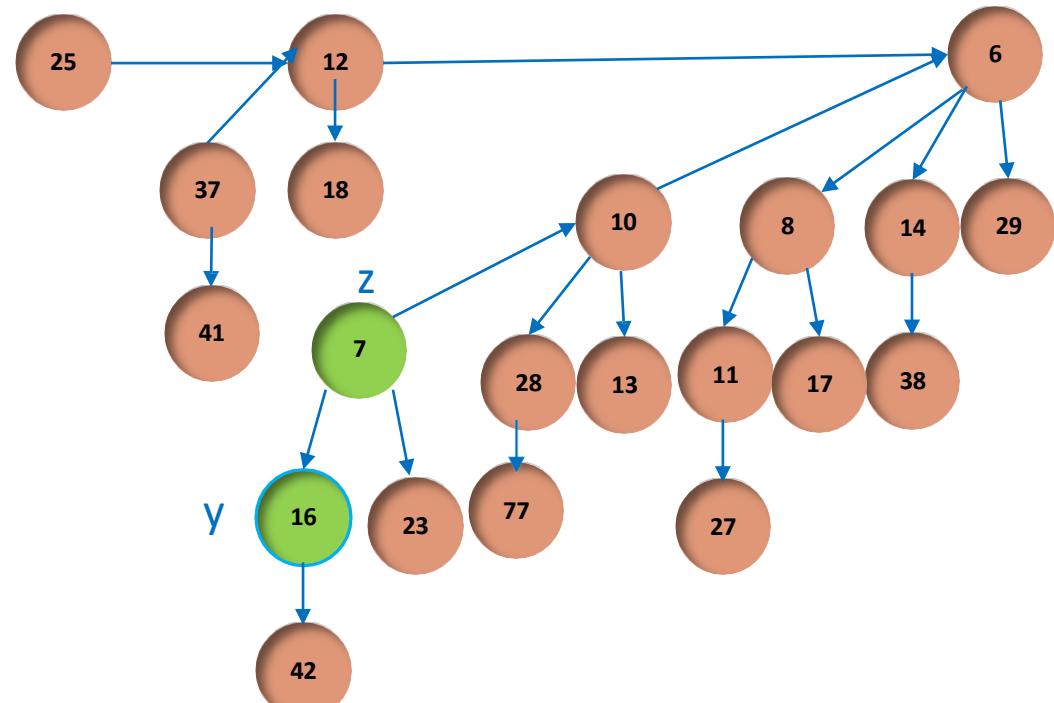
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If  $y$  and  $z$  have satellite fields, exchange them, too.
9
10    $y \leftarrow z$ 
11    $z \leftarrow p[y]$ 

```



Decrease  $x=26$  with  $k=7$

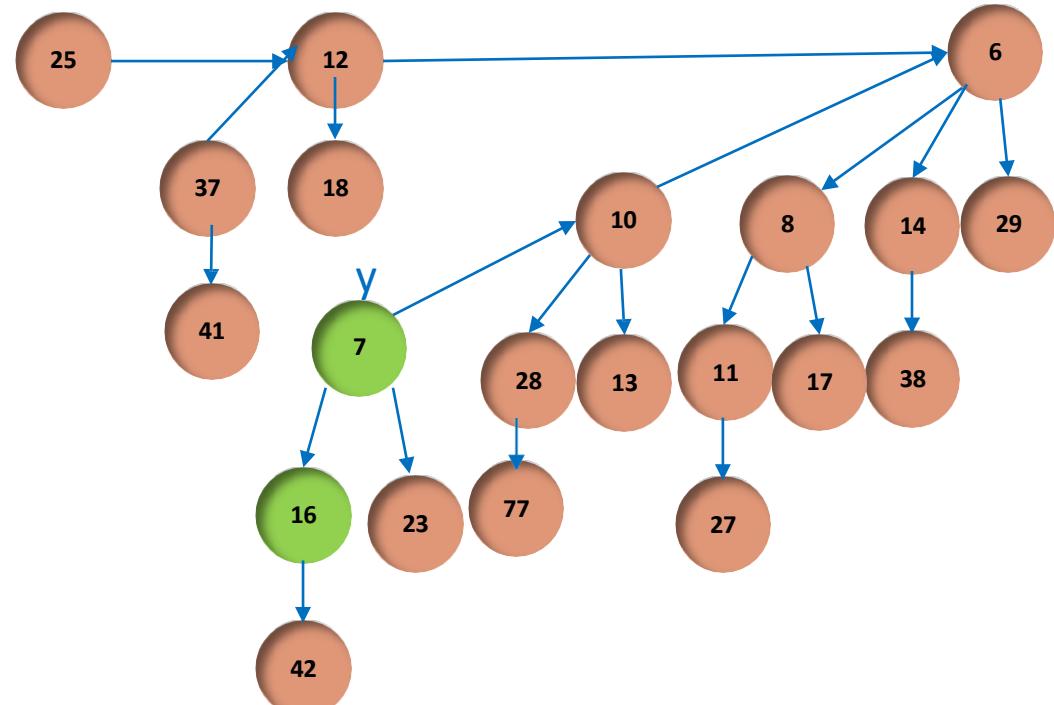
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9
10    $y \leftarrow z$ 
11    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

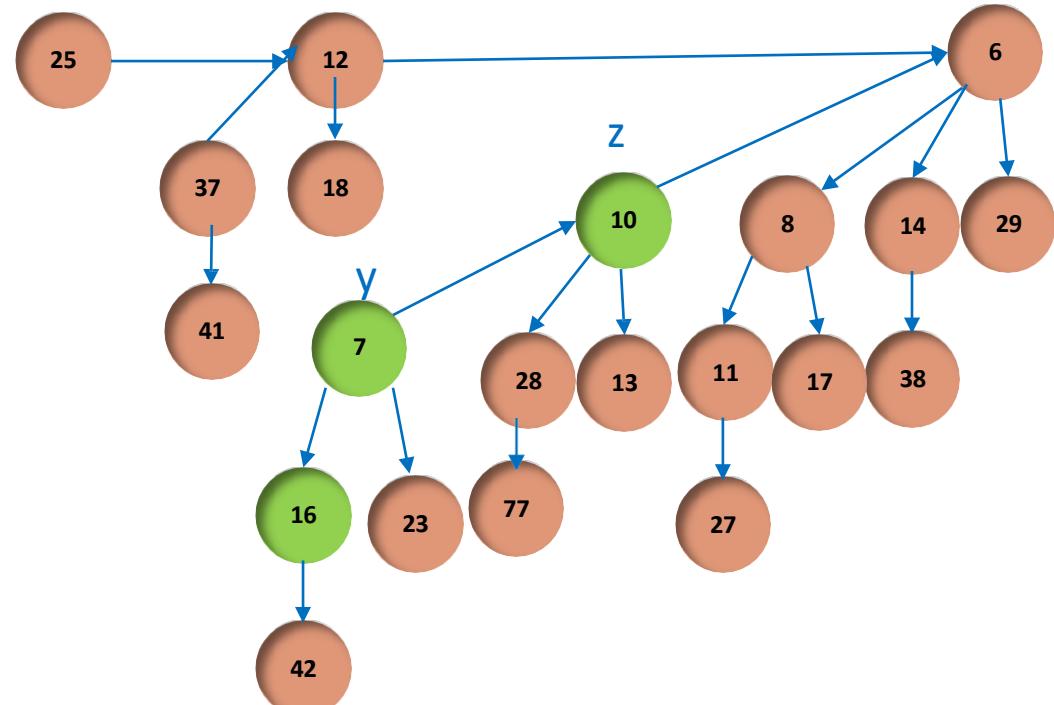
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9
10    $y \leftarrow z$ 
11
12    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

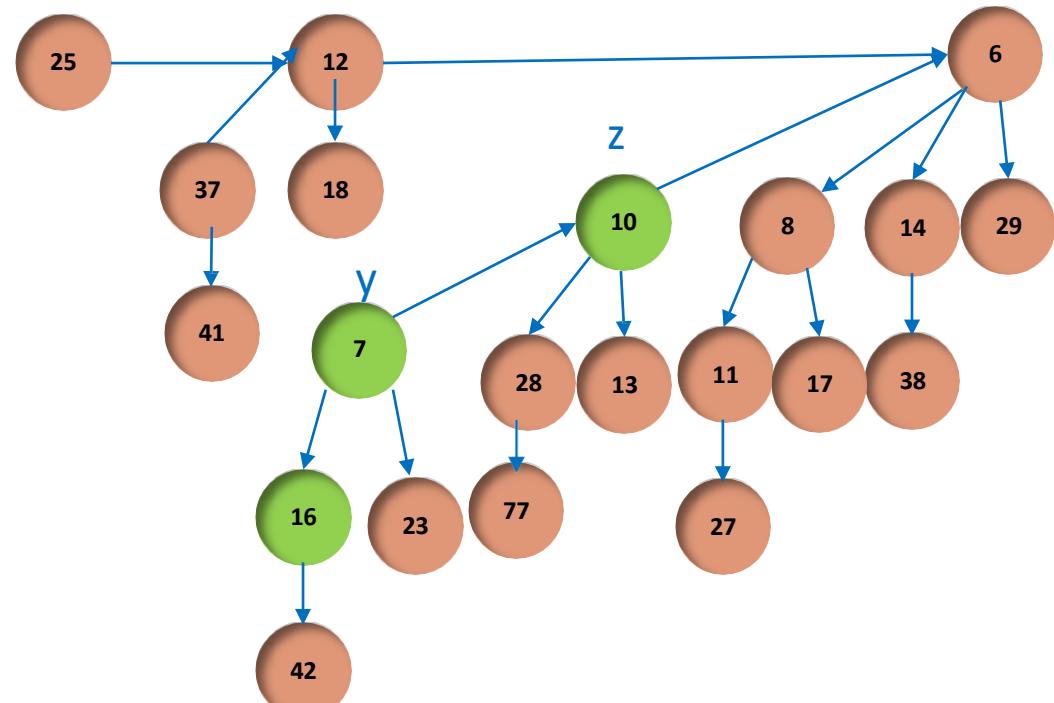
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$   True and True
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9     $y \leftarrow z$ 
10    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

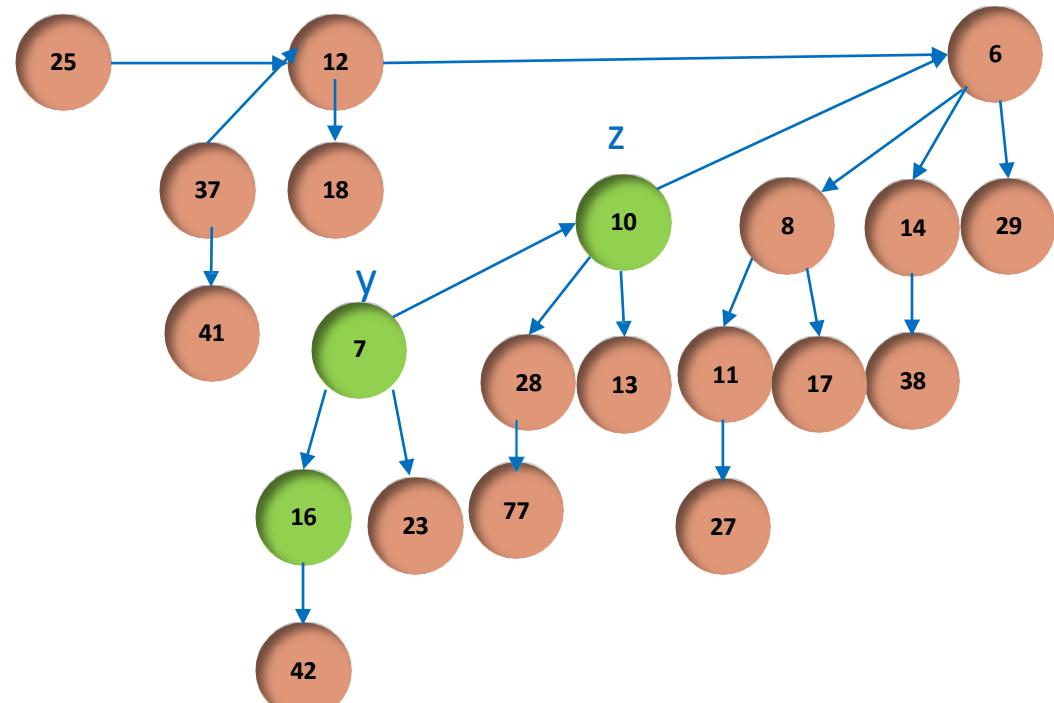
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$  Exchange 7 and 10
8      ▷ If y and z have satellite fields, exchange them, too.
9       $y \leftarrow z$ 
10      $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

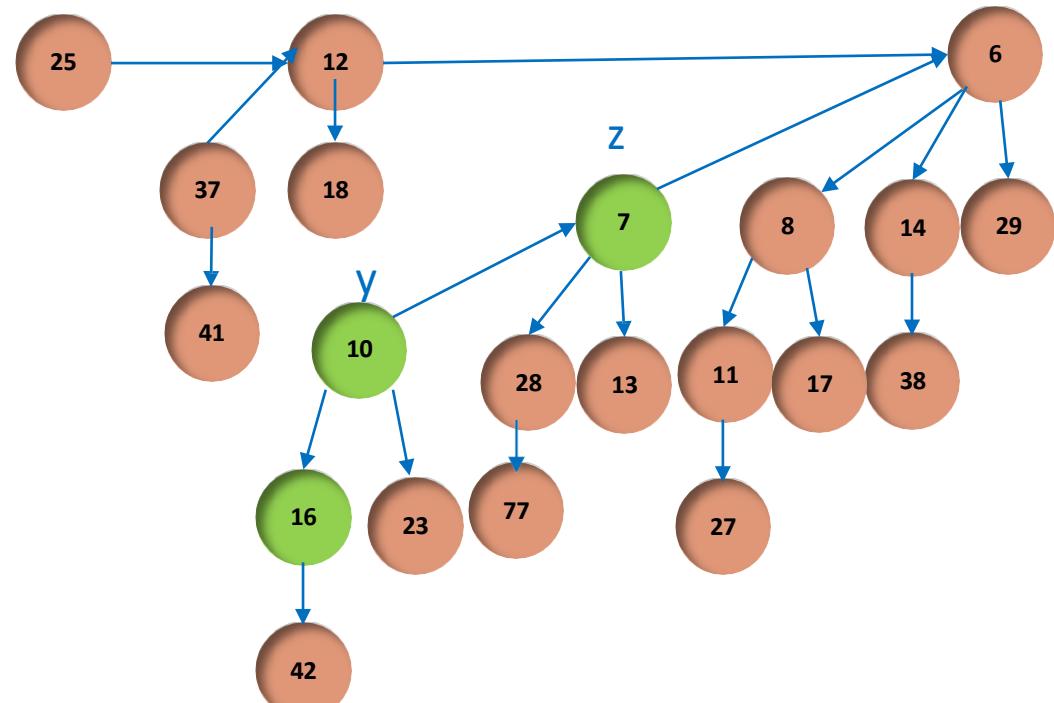
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$  Exchange 7 and 10
8      ▷ If y and z have satellite fields, exchange them, too.
9       $y \leftarrow z$ 
10      $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

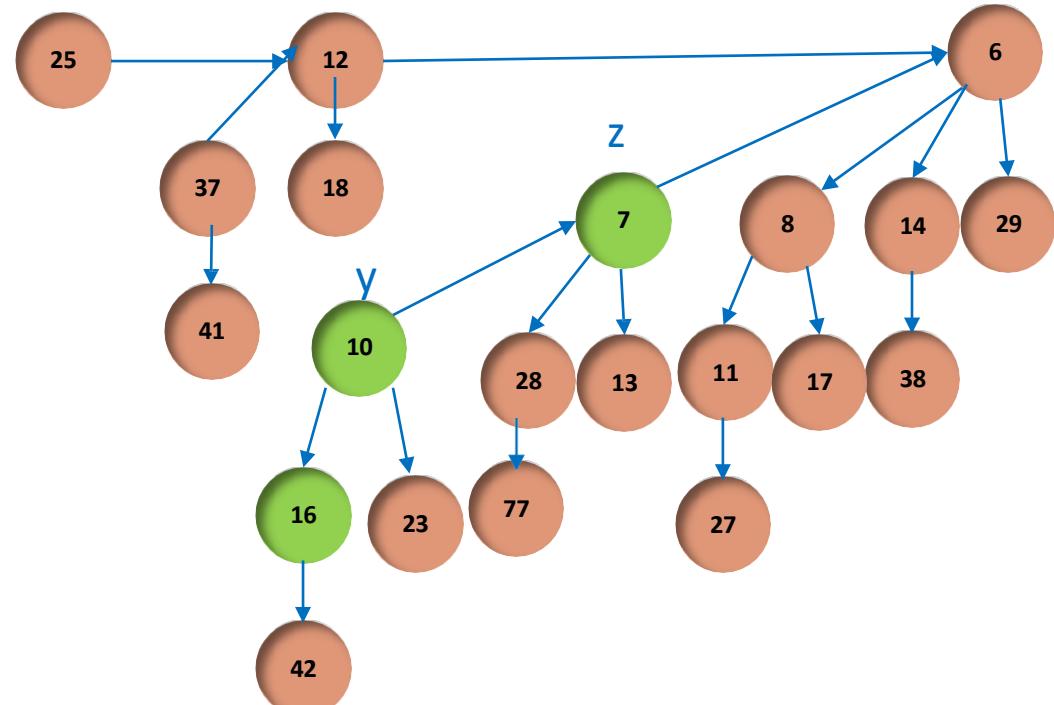
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9
10    $y \leftarrow z$ 
11    $z \leftarrow p[y]$ 

```



Decrease **x=26** with **k=7**

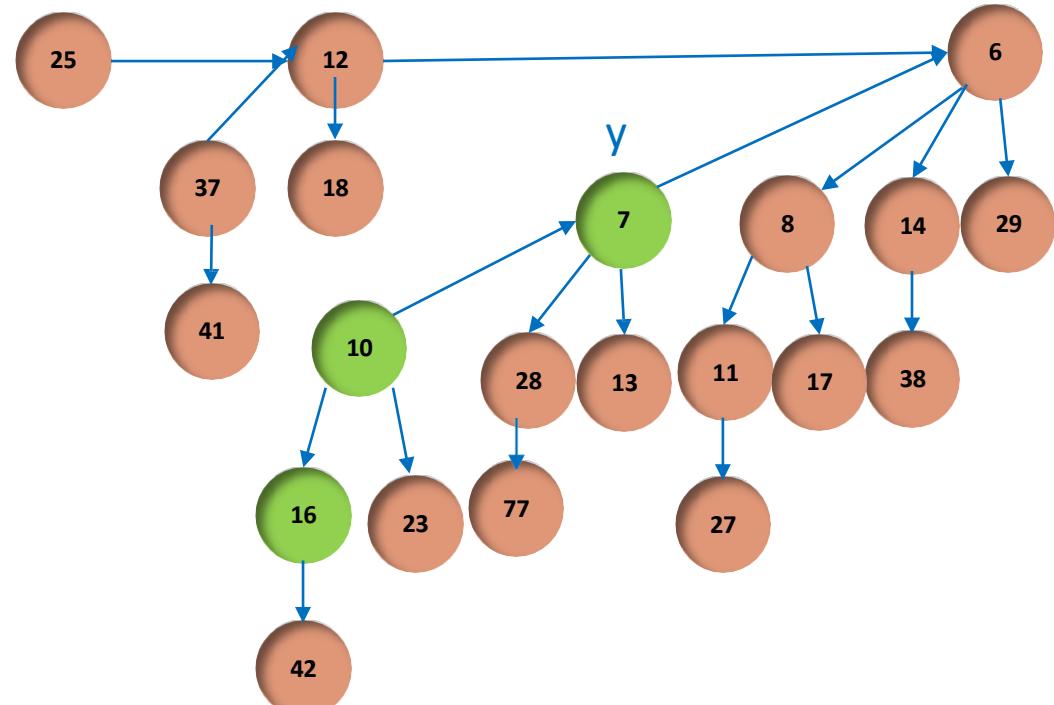
# Decrease Key...

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

```

1  if  $k > \text{key}[x]$   k > key[x]  7 > 26  False
2  then error "new key is greater than current key"
3   $\text{key}[x] \leftarrow k$ 
4   $y \leftarrow x$ 
5   $z \leftarrow p[y]$ 
6  while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7    do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8      ▷ If y and z have satellite fields, exchange them, too.
9
10    $y \leftarrow z$ 
11    $z \leftarrow p[y]$ 

```

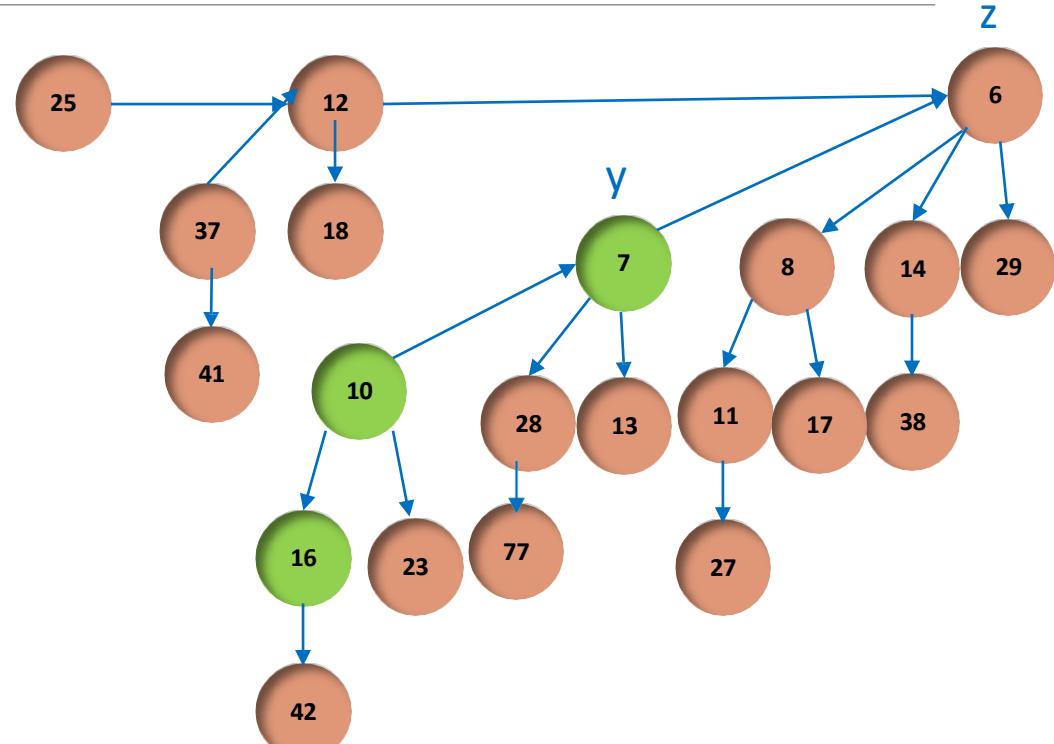


Decrease **x=26** with **k=7**

# Decrease Key...

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

```
1 if  $k > \text{key}[x]$     k > key[x]    7 > 26  False
2 then error "new key is greater than current key"
3  $\text{key}[x] \leftarrow k$ 
4  $y \leftarrow x$ 
5  $z \leftarrow p[y]$ 
6 while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$ 
7   do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8     ▷ If y and z have satellite fields, exchange them, too.
9    $y \leftarrow z$ 
10   $z \leftarrow p[y]$ 
```

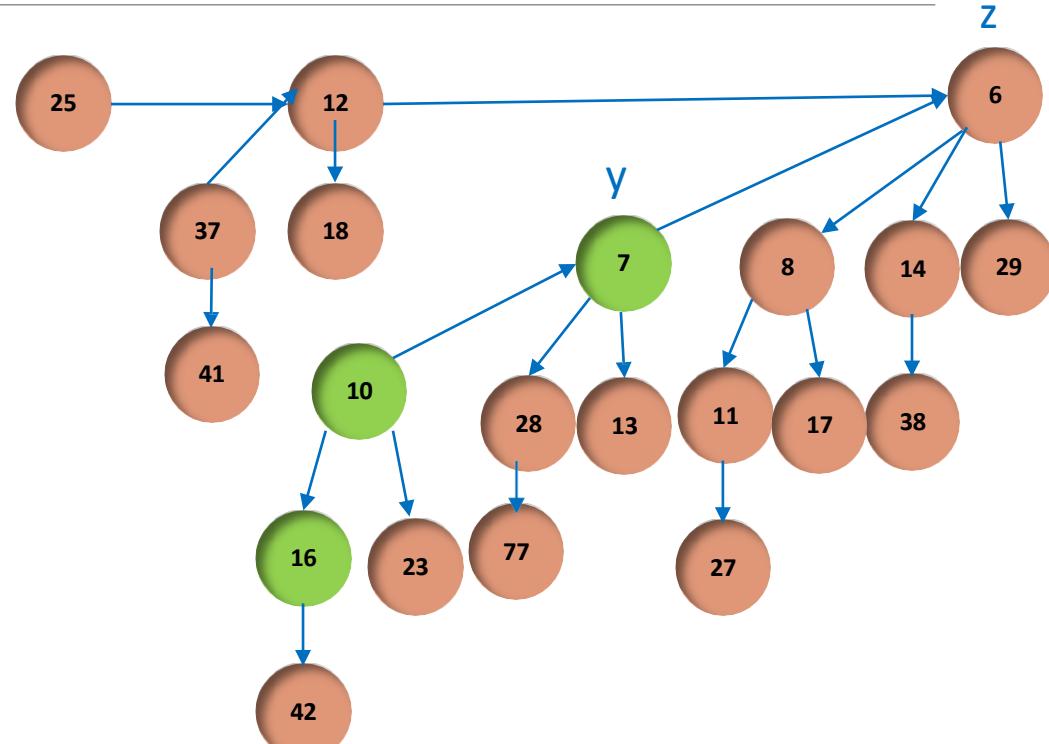


Decrease  $x=26$  with  $k=7$

# Decrease Key...

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

```
1 if  $k > \text{key}[x]$     k > key[x]  7 > 26 False
2 then error "new key is greater than current key"
3  $\text{key}[x] \leftarrow k$ 
4  $y \leftarrow x$ 
5  $z \leftarrow p[y]$ 
6 while  $z \neq \text{NIL}$  and  $\text{key}[y] < \text{key}[z]$       False
7 do exchange  $\text{key}[y] \leftrightarrow \text{key}[z]$ 
8   ▷ If y and z have satellite fields, exchange them, too.
9    $y \leftarrow z$ 
10   $z \leftarrow p[y]$ 
```



Decrease  $x=26$  with  $k=7$



## Design and Analysis of Algorithms

# Binomial Heap – Deletion of Key

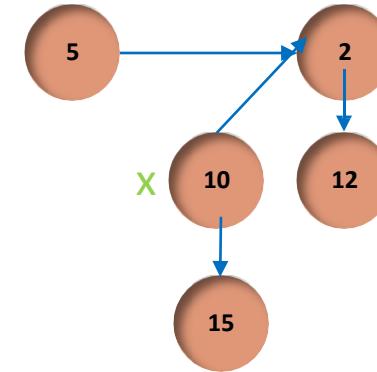
---

# Deletion of Key...

**BINOMIAL-HEAP-DELETE( $H, x$ )**

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

Head H



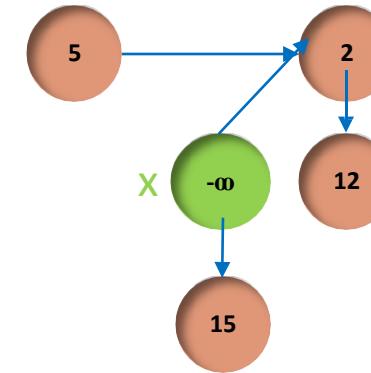
Delete 10

# Deletion of Key...

BINOMIAL-HEAP-DELETE( $H, x$ )

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

Head H



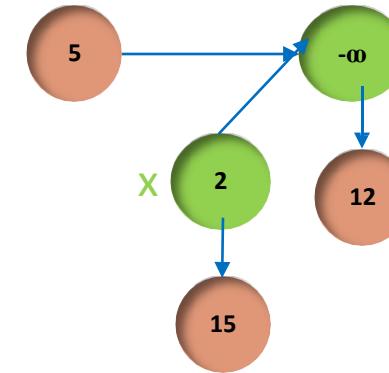
Delete 10

# Deletion of Key...

BINOMIAL-HEAP-DELETE( $H, x$ )

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

Head H

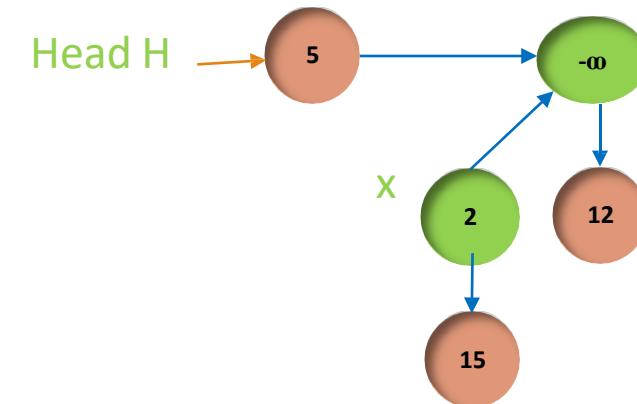


Delete 10

# Deletion of Key...

BINOMIAL-HEAP-DELETE( $H, x$ )

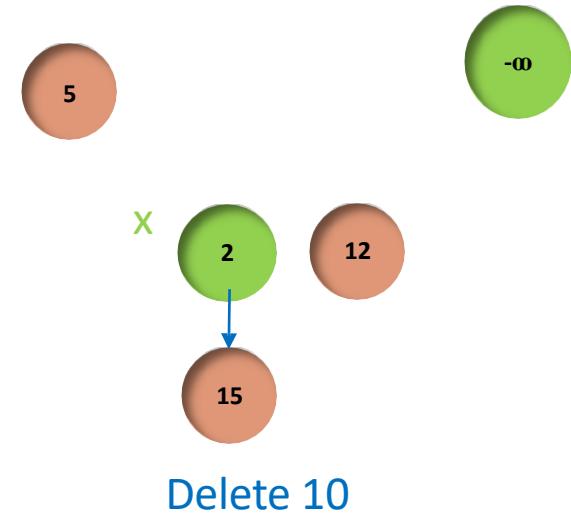
- 1 BINOMIAL-HEAP-DECREASE-KEY( $H, x, -\infty$ )
- 2 BINOMIAL-HEAP-EXTRACT-MIN( $H$ )



# Deletion of Key...

BINOMIAL-HEAP-DELETE( $H, x$ )

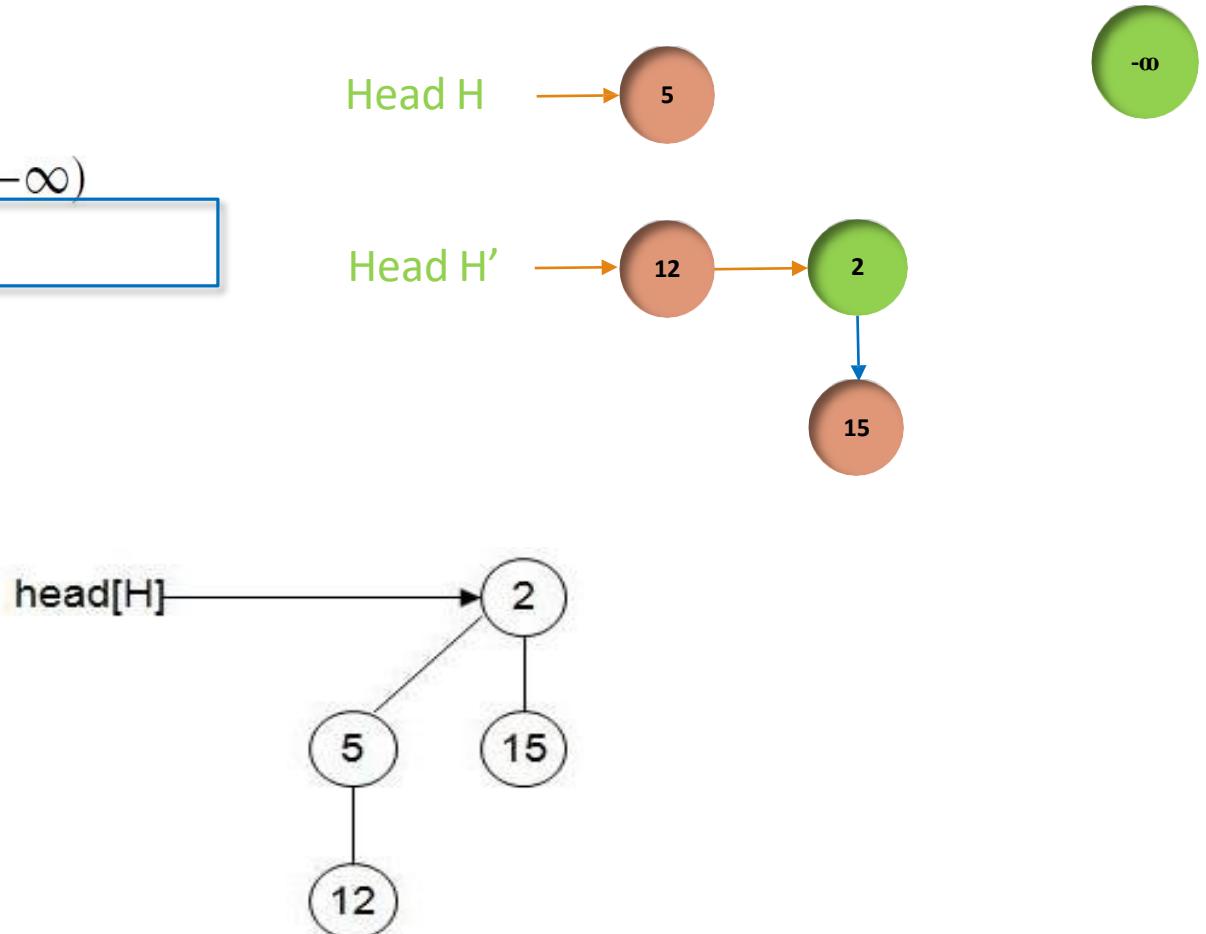
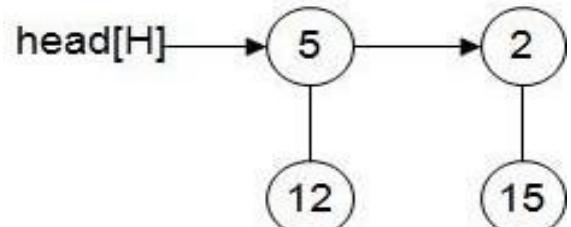
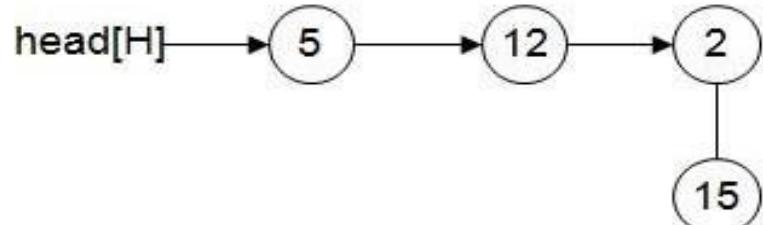
- 1 BINOMIAL-HEAP-DECREASE-KEY( $H, x, -\infty$ )
- 2 BINOMIAL-HEAP-EXTRACT-MIN( $H$ )



# Deletion of Key...

BINOMIAL-HEAP-DELETE( $H, x$ )

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



# Insertion of Key...

**BINOMIAL-HEAP-INSERT( $H, x$ )**

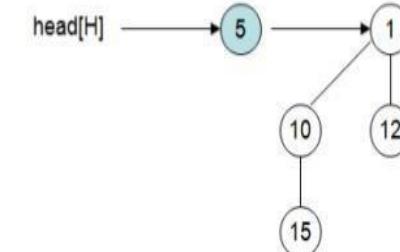
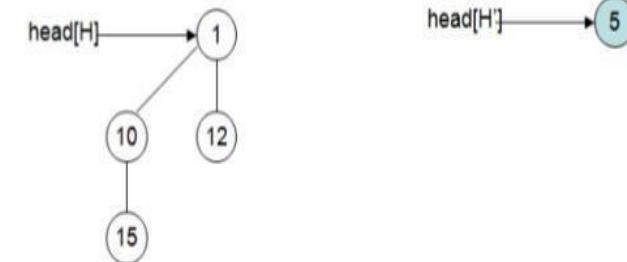
- 1  $H' \leftarrow \text{MAKE-BINOMIAL-HEAP}()$
- 2  $p[x] \leftarrow \text{NIL}$
- 3  $\text{child}[x] \leftarrow \text{NIL}$
- 4  $\text{ sibling}[x] \leftarrow \text{NIL}$
- 5  $\text{degree}[x] \leftarrow 0$
- 6  $\text{head}[H'] \leftarrow x$
- 7  $H \leftarrow \text{BINOMIAL-HEAP-UNION}(H, H')$

**Do it by yourself**

## Insert New Node

New node:

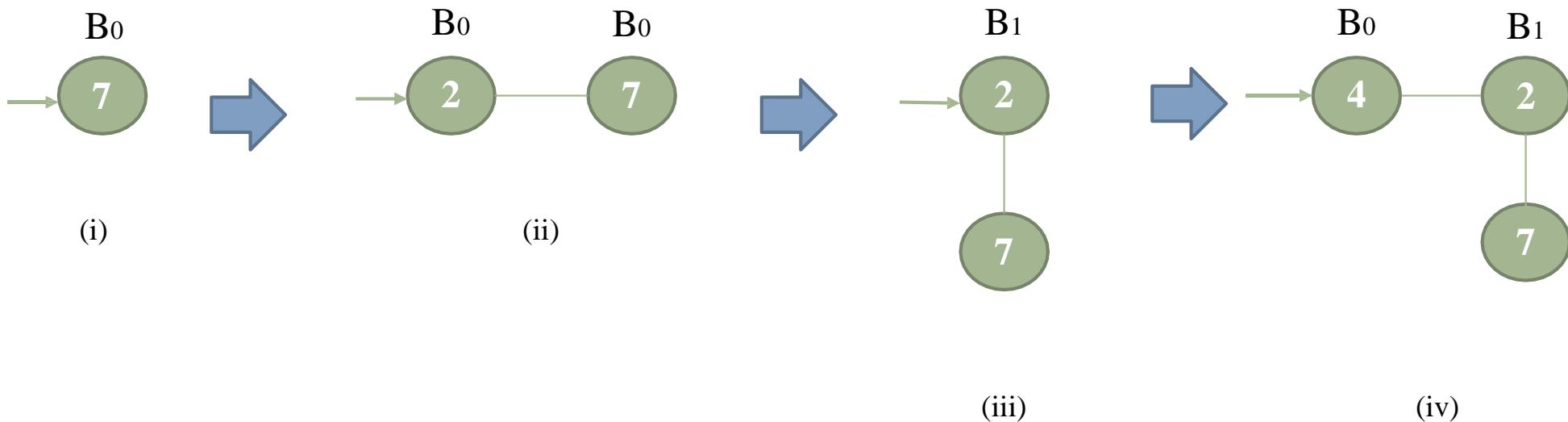
5



# Example

Let  $A = [7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20]$

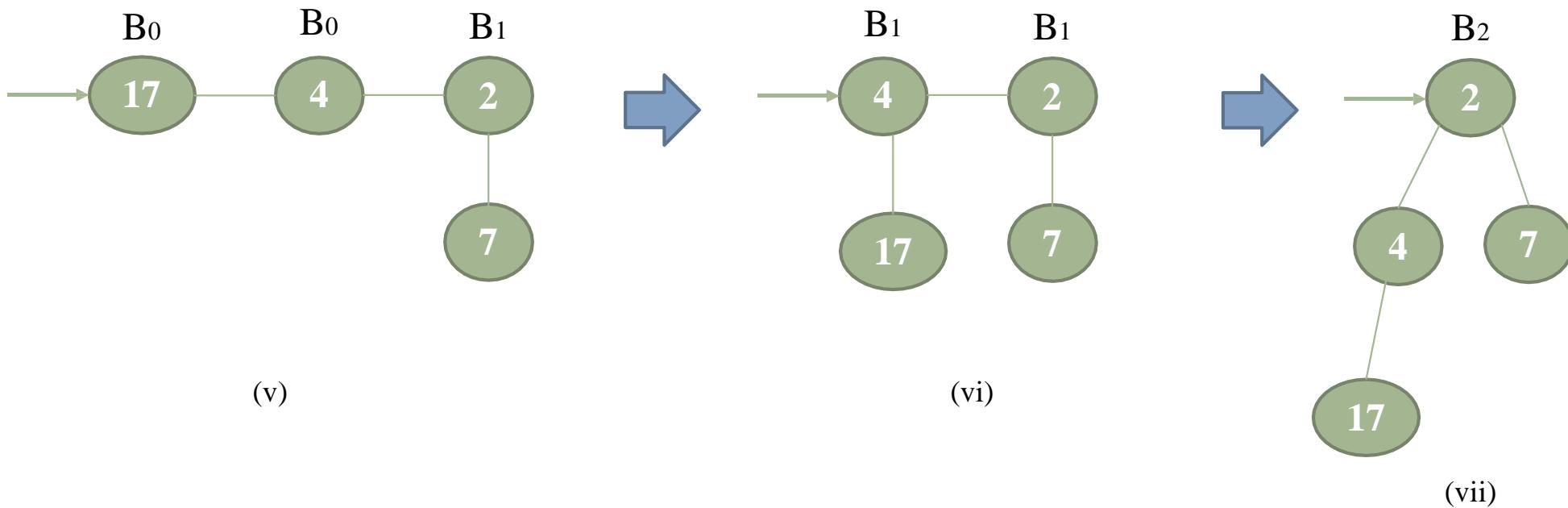
- i. Construct a Binomial Heap for the above list of elements.
- ii. Extract minimum key one time.



# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

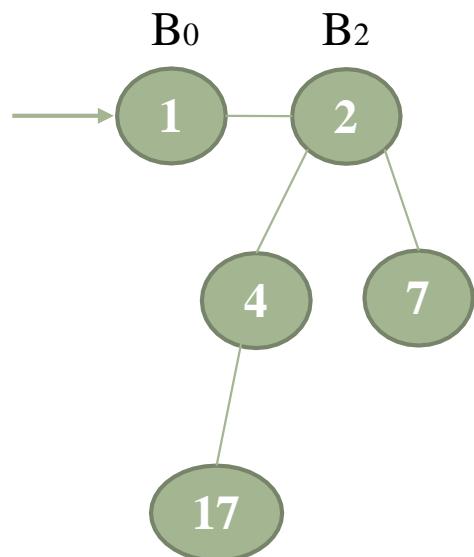
## i. Construct a Binomial Heap for the above list of elements.



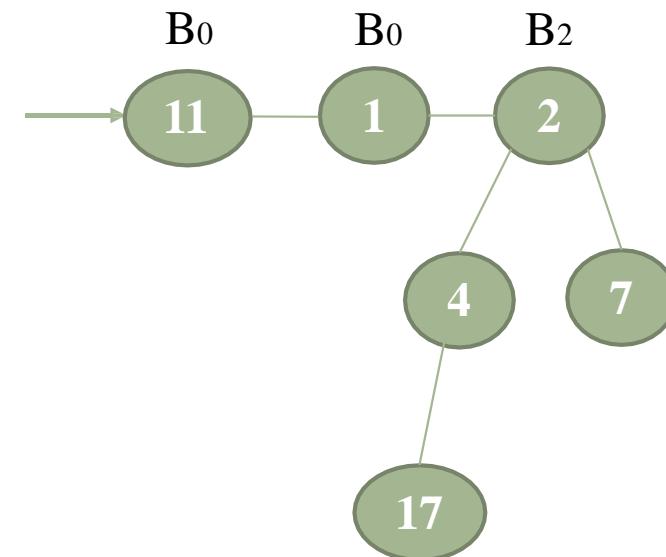
# Example ...

Let  $A = [7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20]$

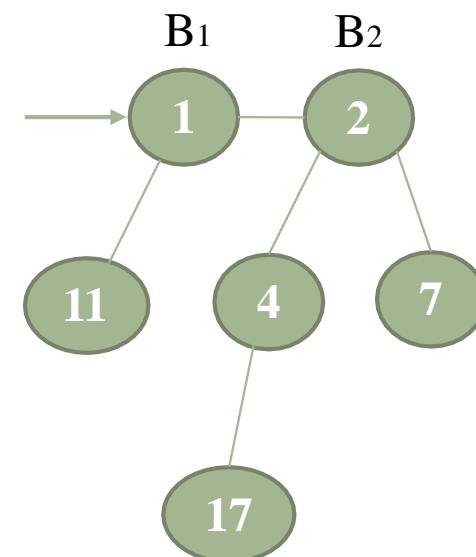
## i. Construct a Binomial Heap for the above list of elements.



(viii)



(ix)

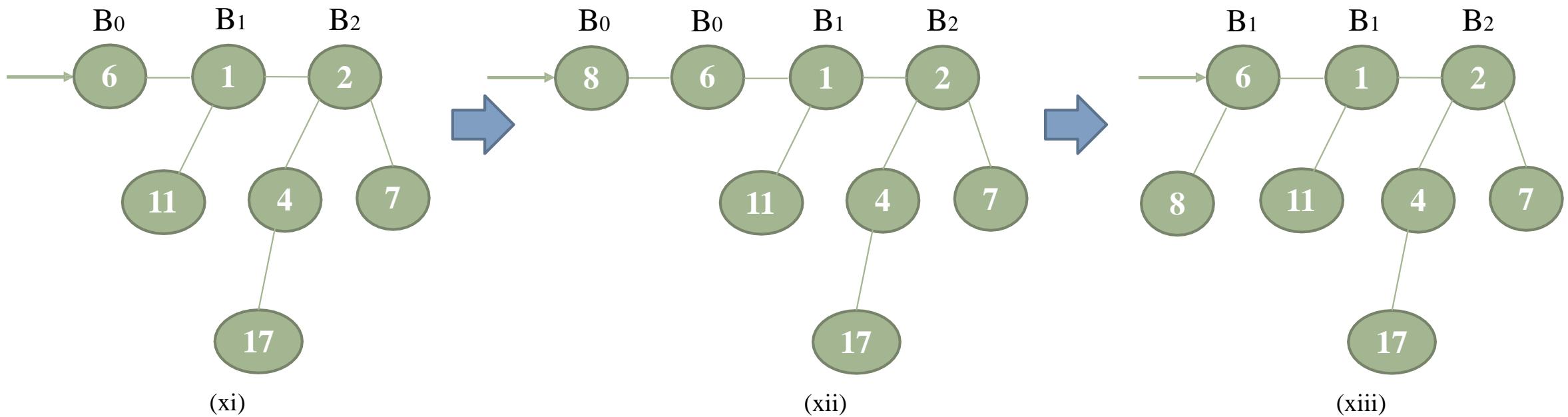


(x)

# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

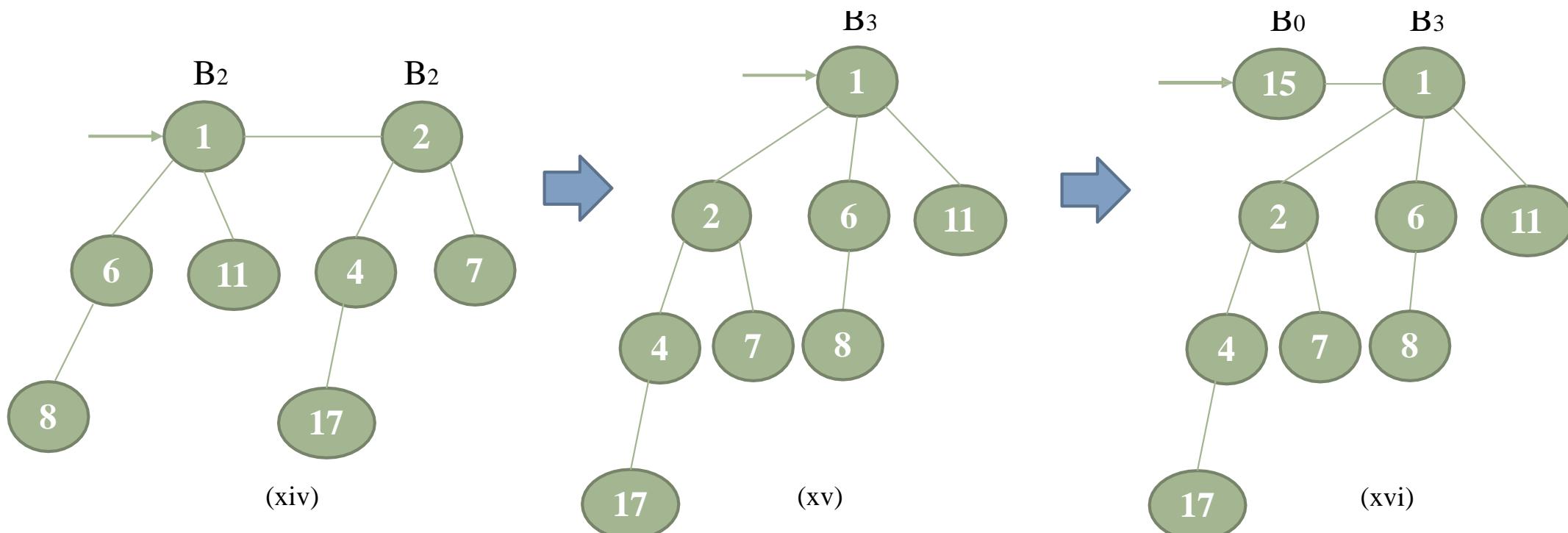
## i. Construct a Binomial Heap for the above list of elements.



# Example ...

Let  $A = [7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20]$

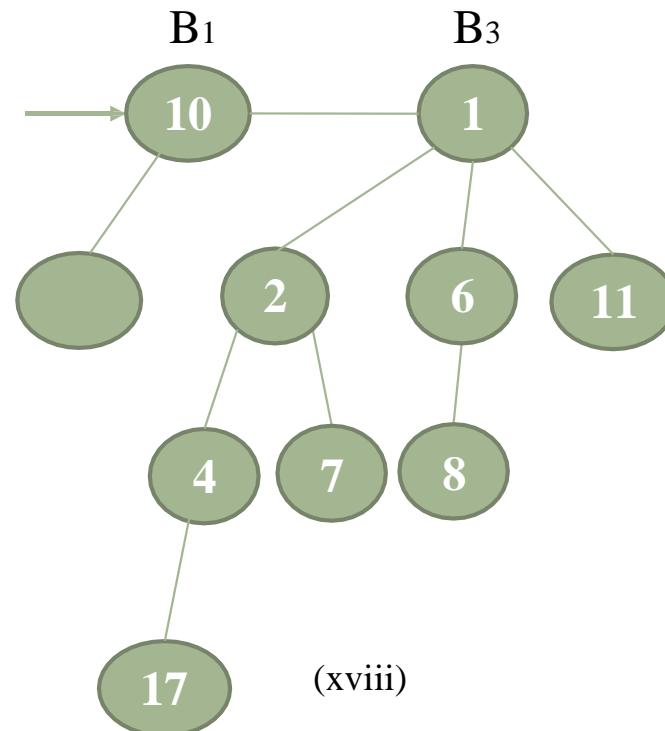
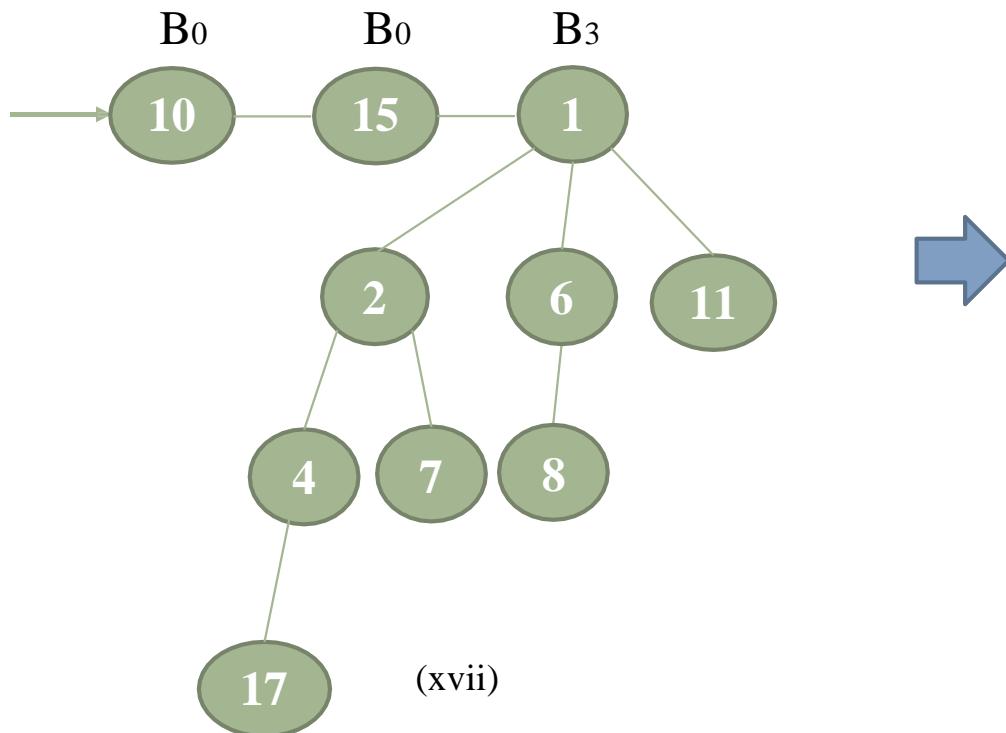
## i. Construct a Binomial Heap for the above list of elements.



# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

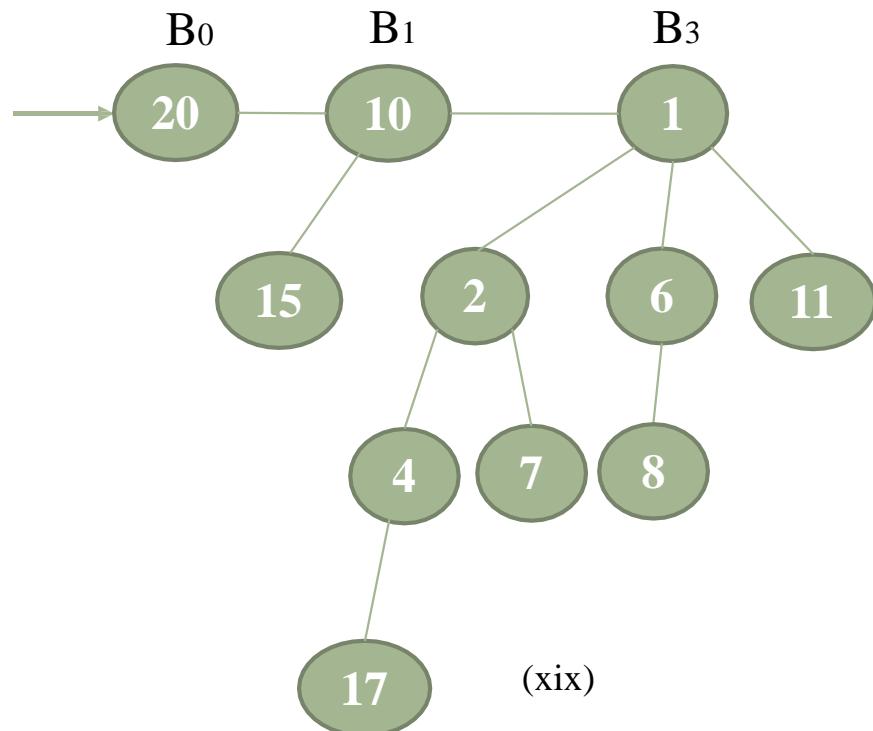
i. Construct a Binomial Heap for the above list of elements.



# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

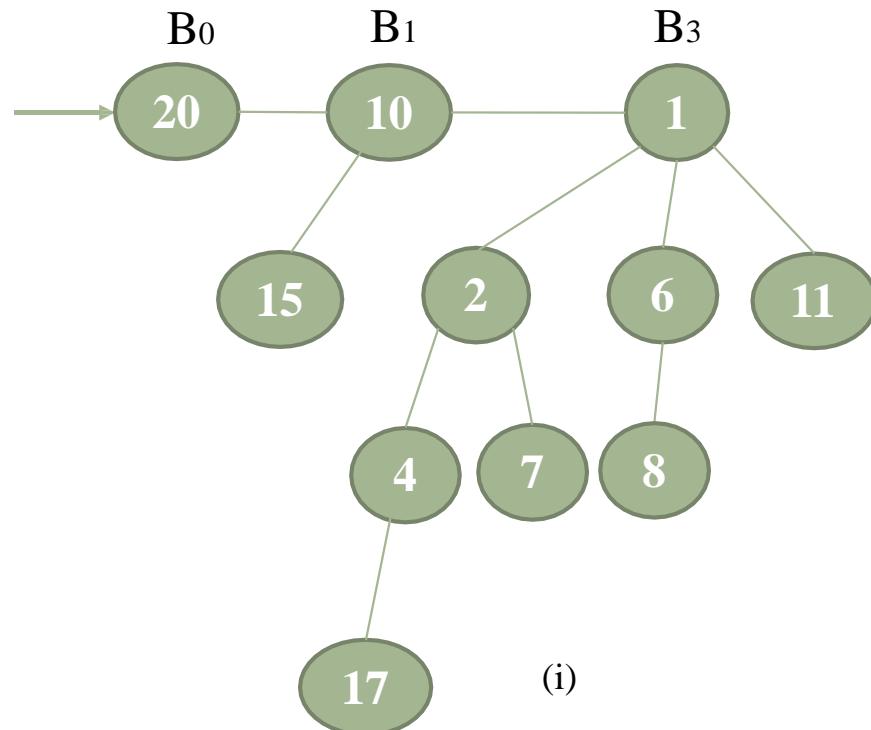
- i. Construct a Binomial Heap for the above list of elements.



# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

## i. Construct a Binomial Heap for the above list of elements.

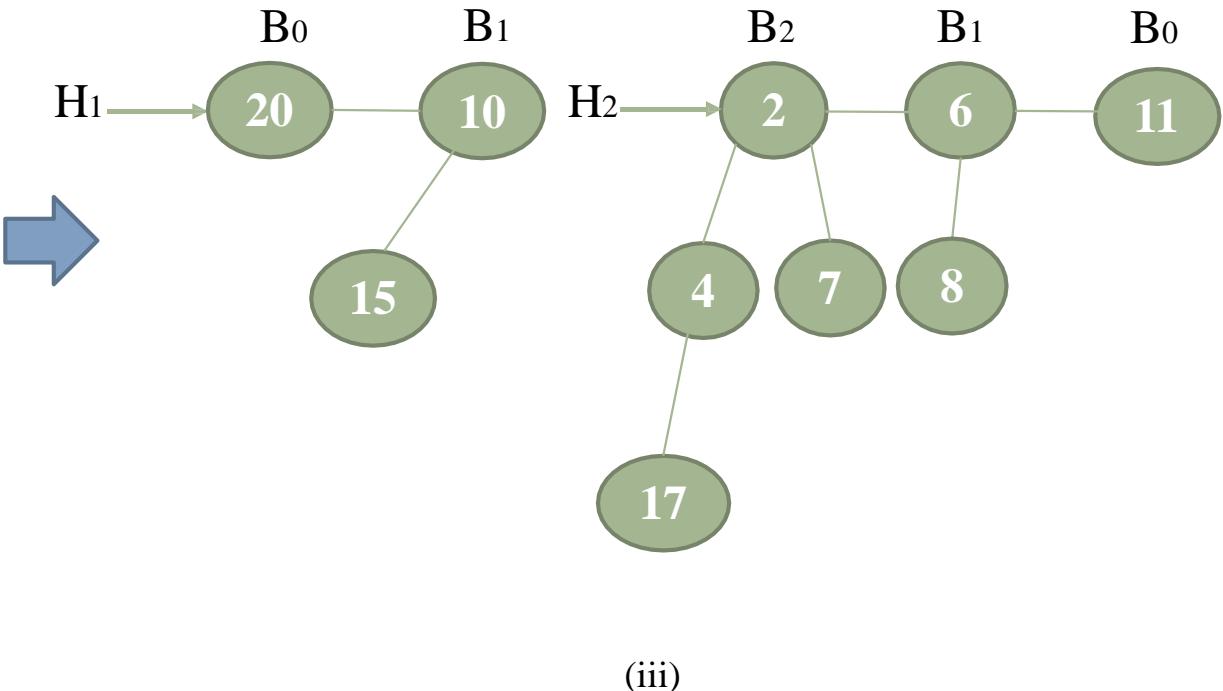
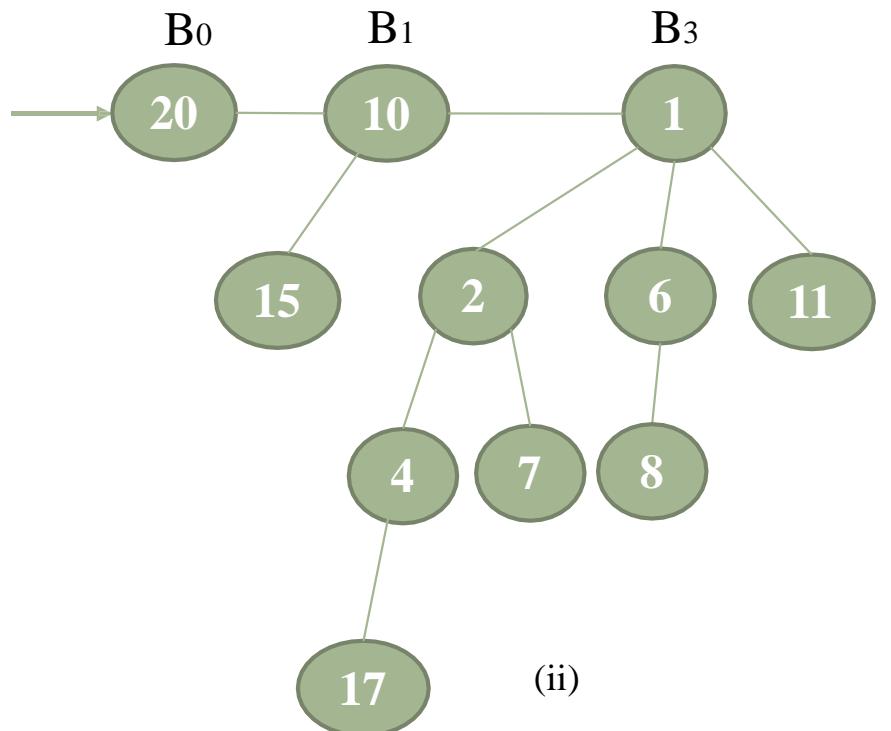


The minimum key will always be on the root as it follows the property of min-heap.  
Here, minimum key is 1.

# Example ...

Let A= [ 7,2,4,17,1,11,6,8,15,10,20]

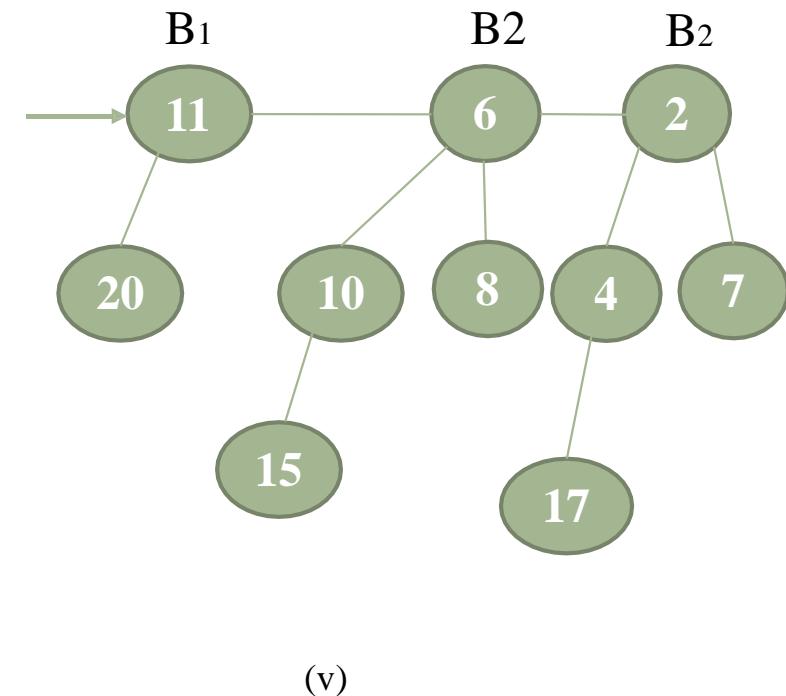
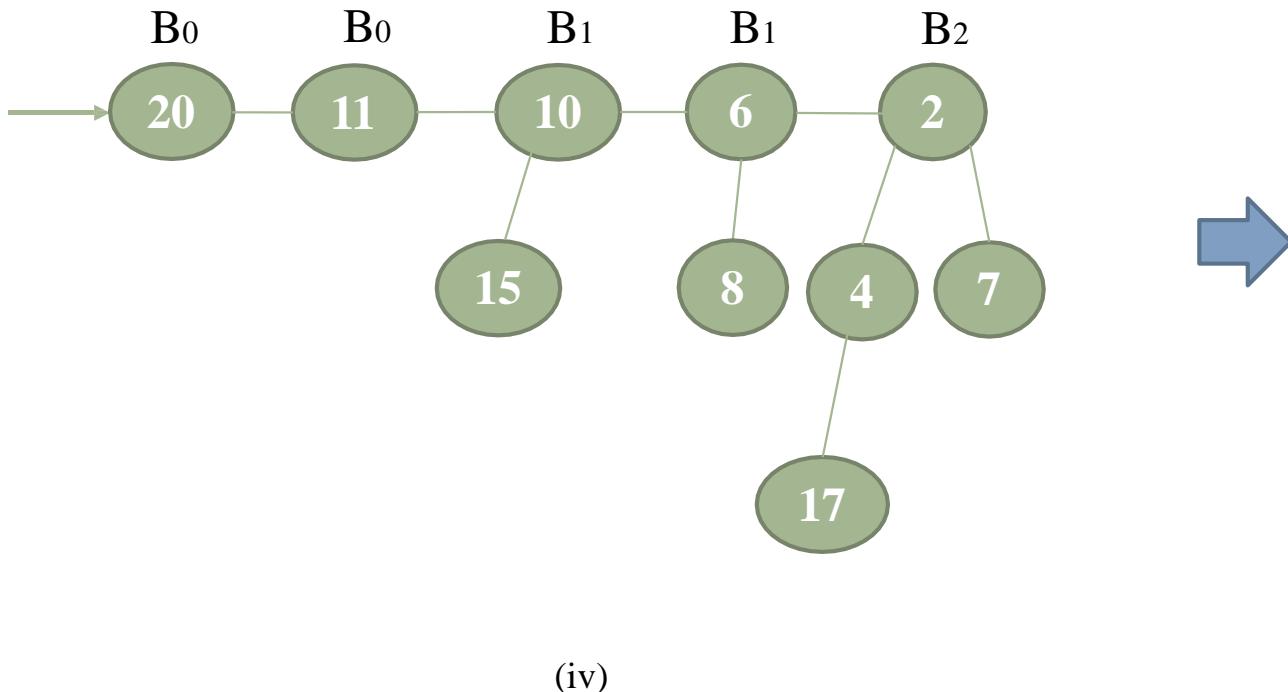
**Extract minimum key one time.**



# Example ...

Let  $A = [7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20]$

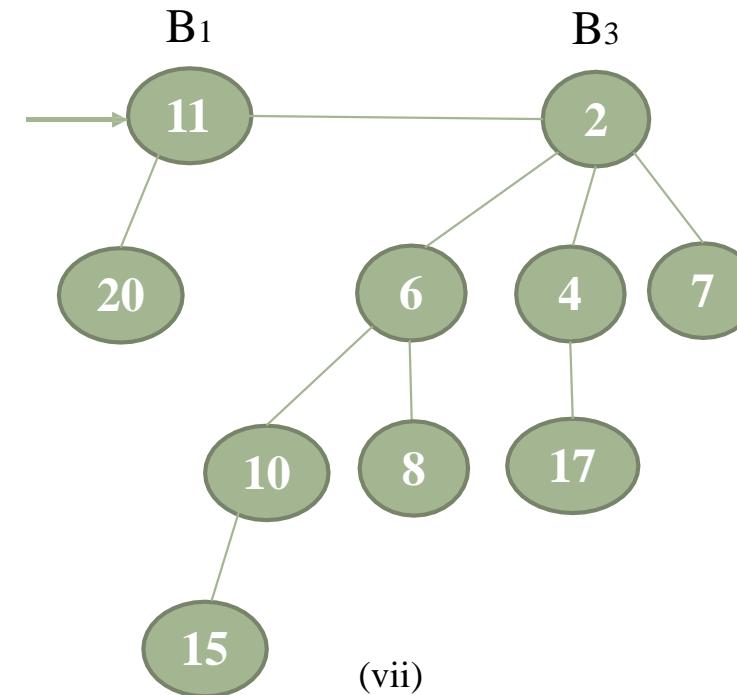
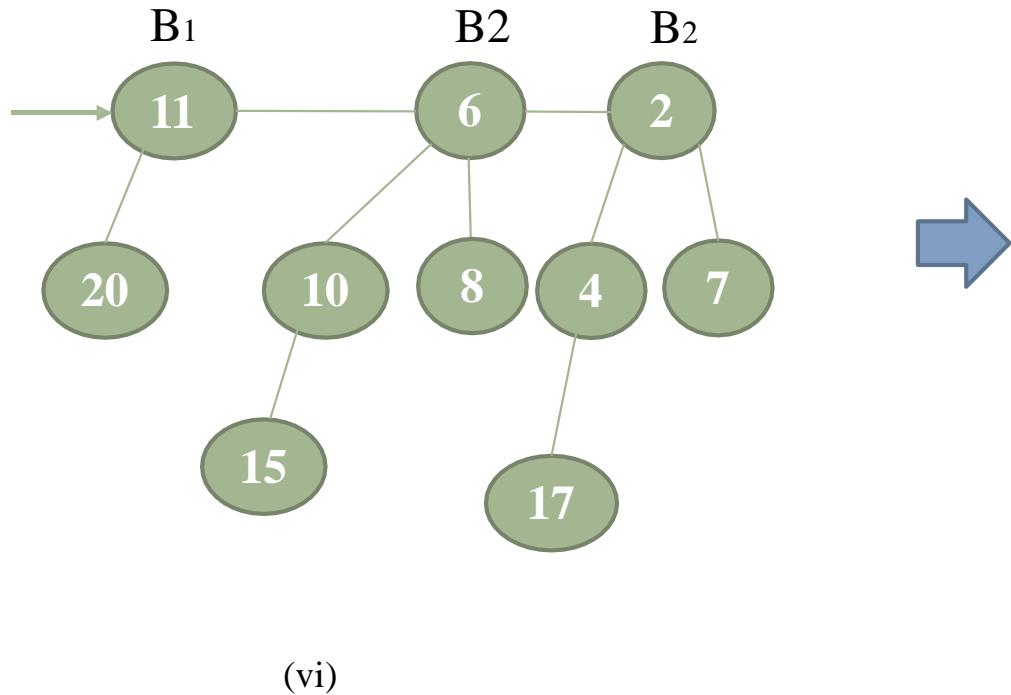
Extract minimum key one time.



# Example ...

Let  $A = [7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20]$

Extract minimum key one time.



“Thank you”

*Any Questions ?*



**Dr. Anand Singh Jalal**  
**Professor**  
**Email: [asjalal@gla.ac.in](mailto:asjalal@gla.ac.in)**