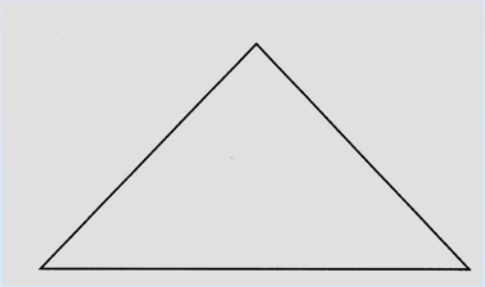
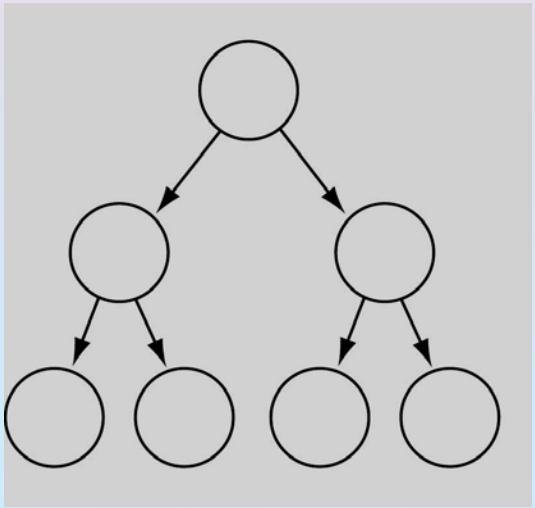
Heaps

Full Binary Tree

- Every non-leaf node has two children
- All the leaves are on the same level

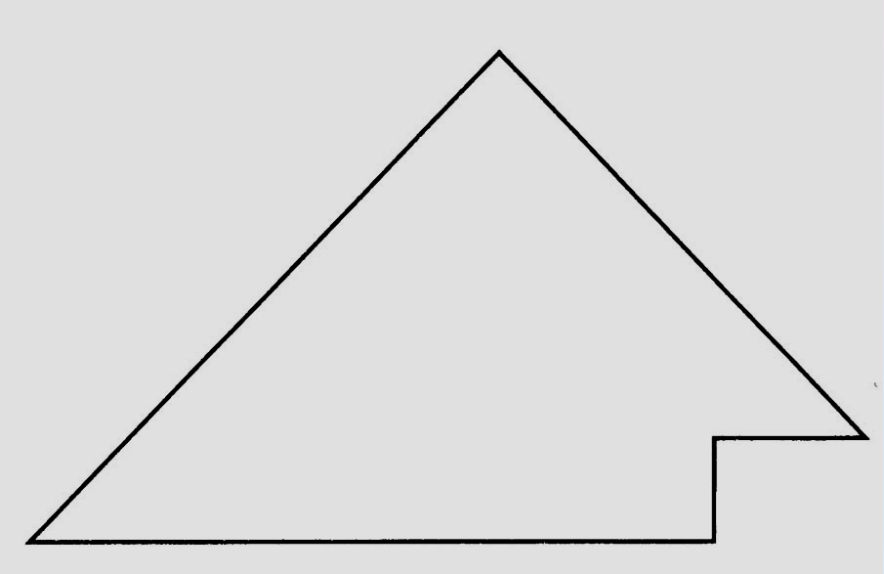




Complete Binary Tree

- A binary tree that is either full or full through the next-to-last level
- The last level is full **from left to right** (i.e., leaves are as far to the left as possible)

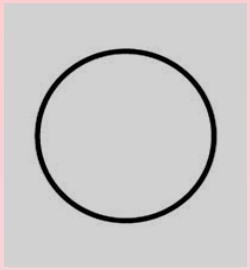
Full ? ✗



Complete ? ✓

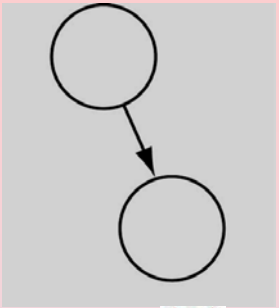
Complete Binary Tree

Complete Binary Tree



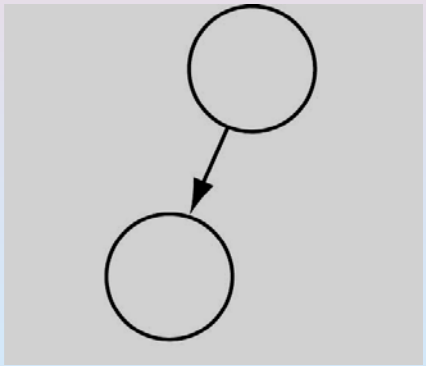
Full ? ✓

Complete ? ✓



Full ? ✗

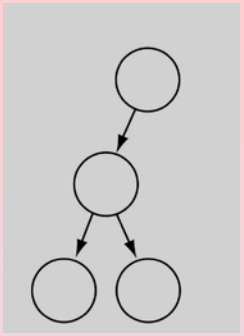
Complete ? ✗



Full ? ✗

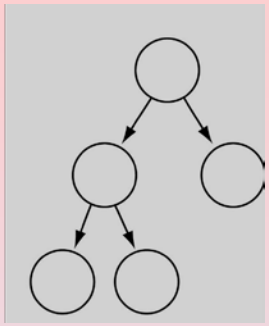
Complete ? ✓

Complete Binary Tree



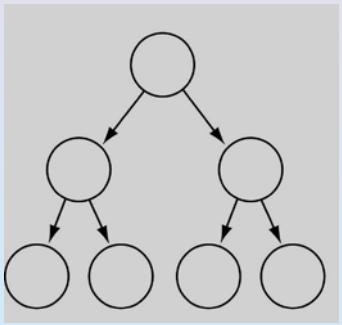
Full ? ✗

Complete ? ✗



Full ? ✗

Complete ? ✓



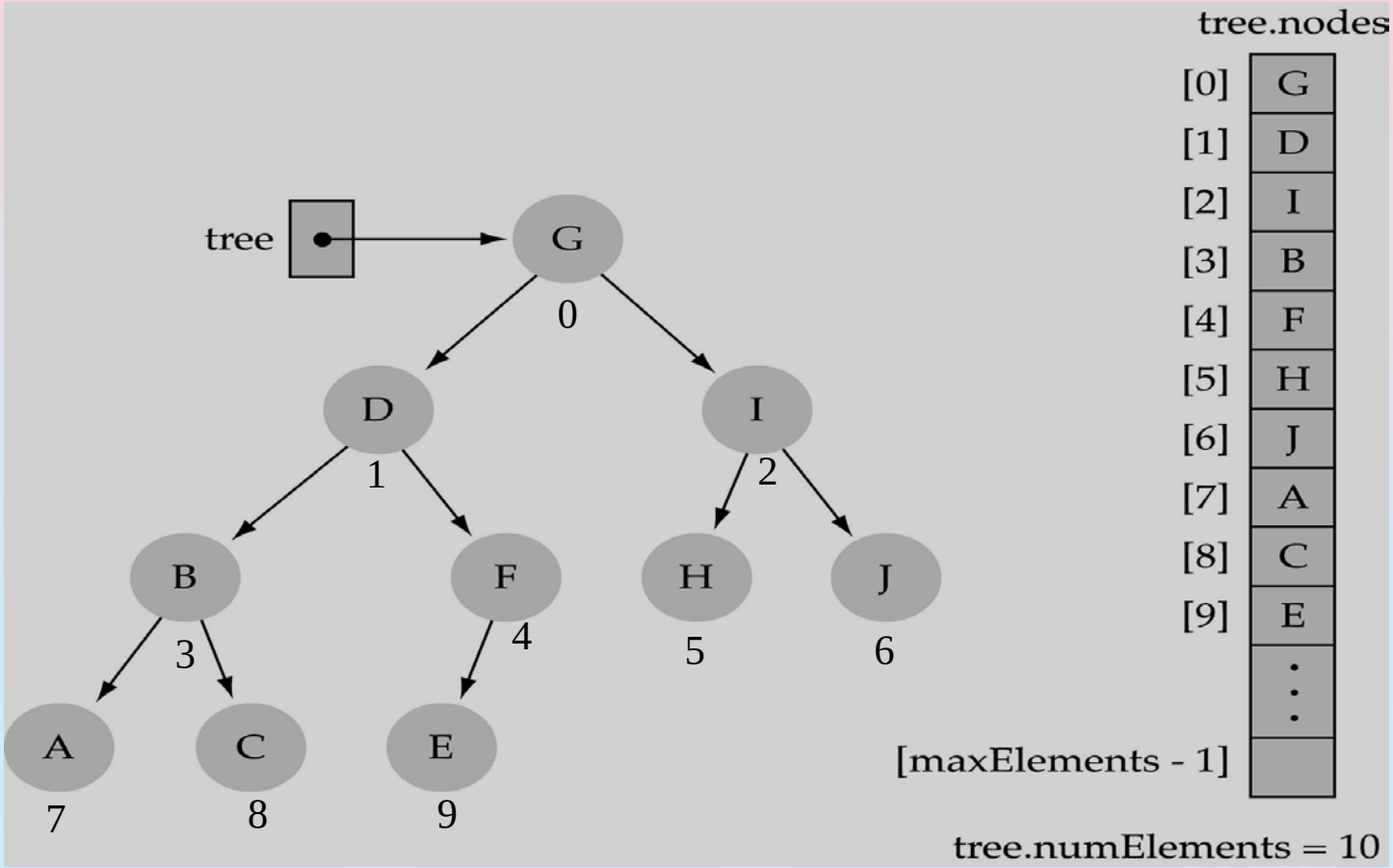
Full ? ✓

Complete ? ✓

Array-based representation of Binary trees

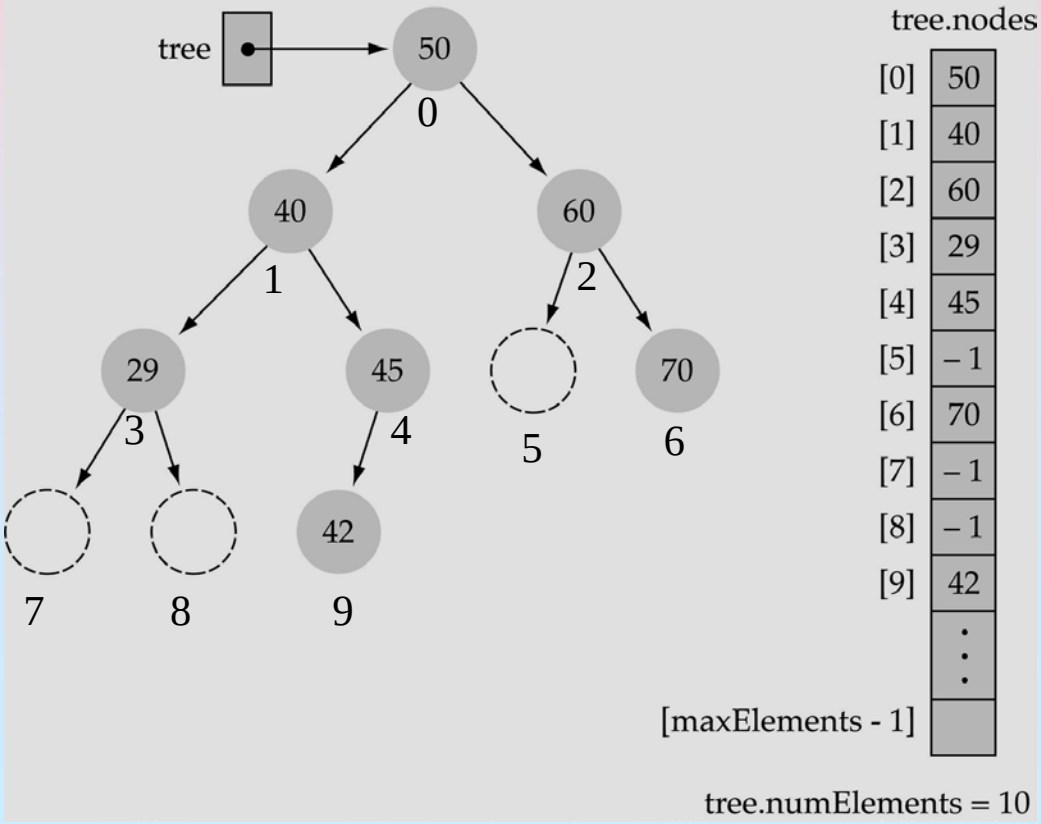
- **Memory space is saved (no pointers are required)**
- **Preserve parent-child relationships by storing the tree elements in the array**

  ***(i)* level by level, and *(ii)* left to right**



Array-based representation of binary trees (cont.)

- Full or complete trees can be implemented easily using an array-based representation (elements occupy contiguous array slots)
- "Dummy nodes" are required for trees which are not full or complete



What is a Heap?

- A **Heap** is a kind of **complete binary tree**.
- It is a **Binary tree** with the following properties:
  - *Property 1:* it is a **complete binary tree**
  - *Property 2:* the value stored at a **node** is **greater** or **equal** to the values stored at the **children** (**Heap property**)

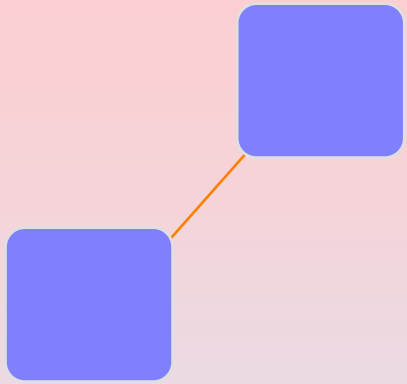
Heaps

Root

When a complete binary tree is built, its first node must be the root.

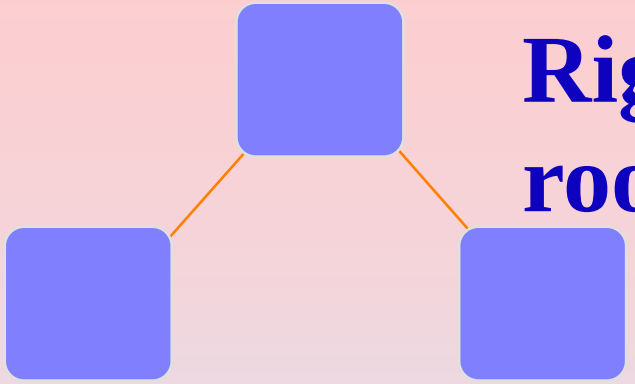
Heaps

Left child of the root



The second node is always the left child of the root.

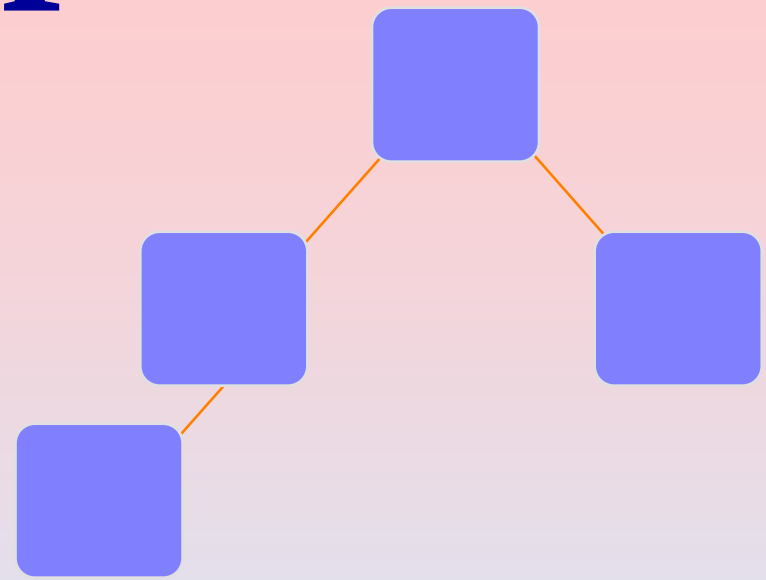
Heaps



Right child of the root

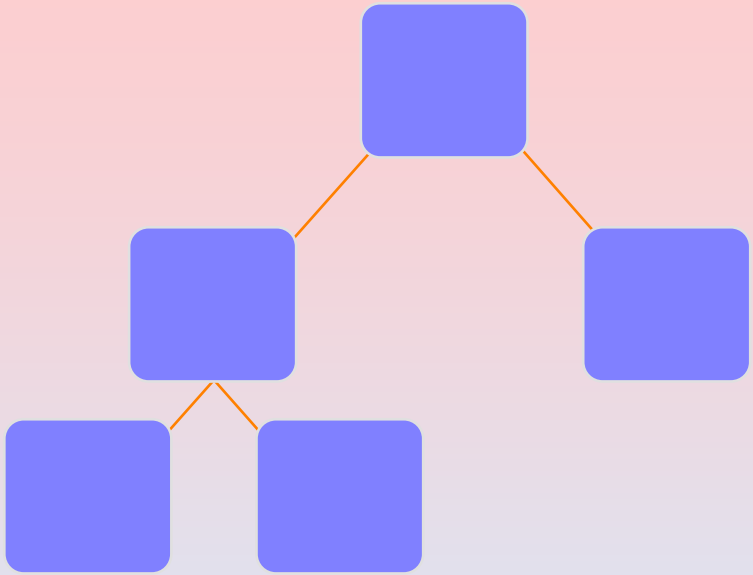
The third node is always the right child of the root.

Heaps



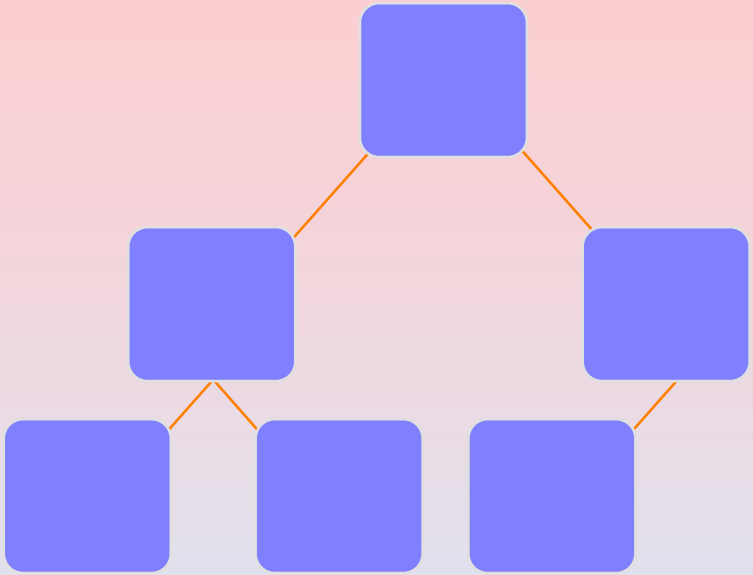
The next nodes always fill the next level from left-to-right.

Heaps



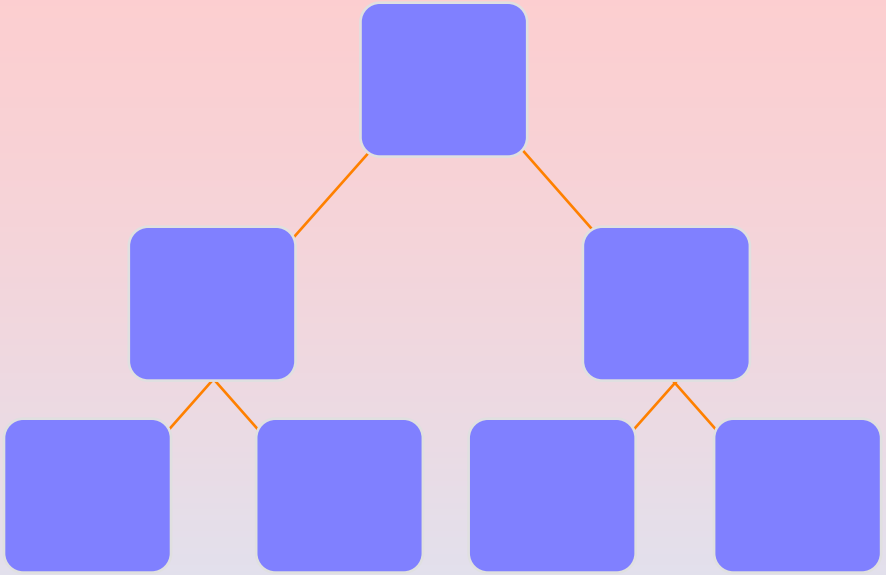
The next nodes always fill the next level from left-to-right.

Heaps



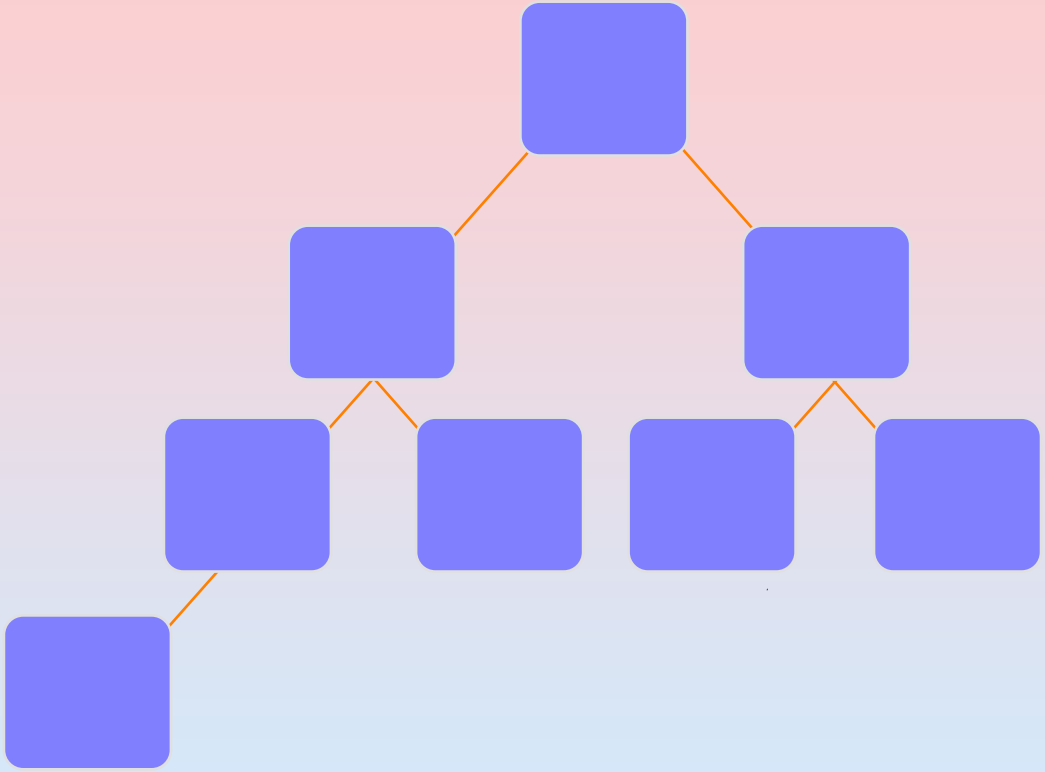
The next nodes always fill the next level from left-to-right.

Heaps

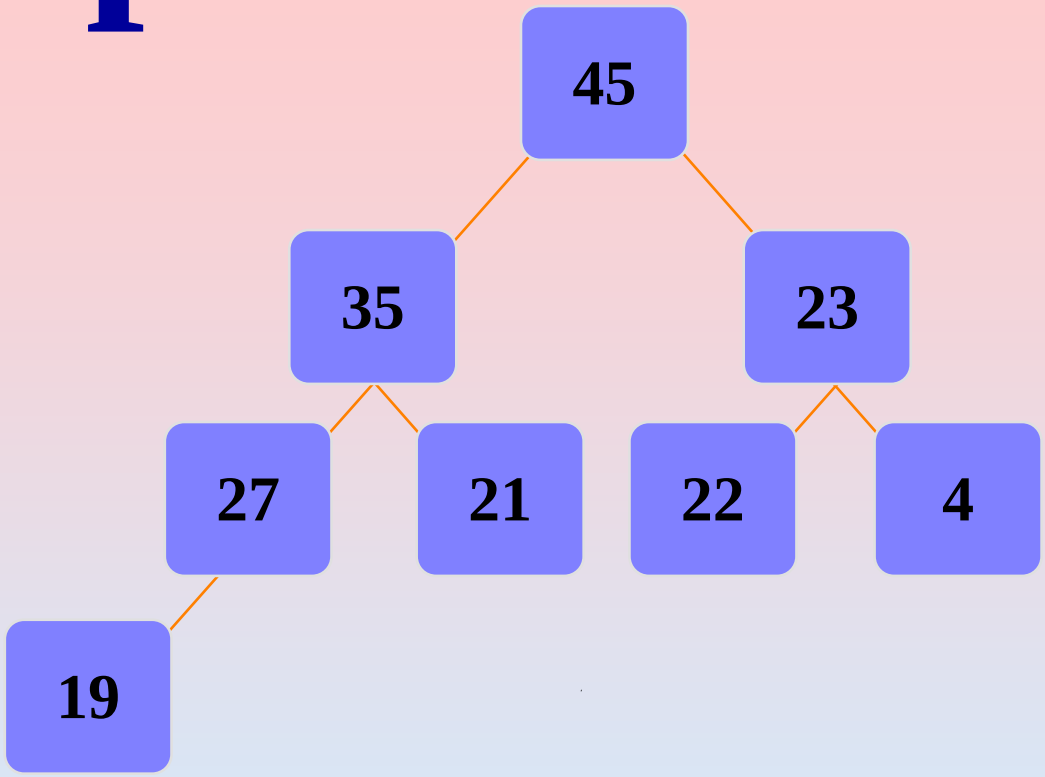


The next nodes always fill the next level from left-to-right.

Heaps

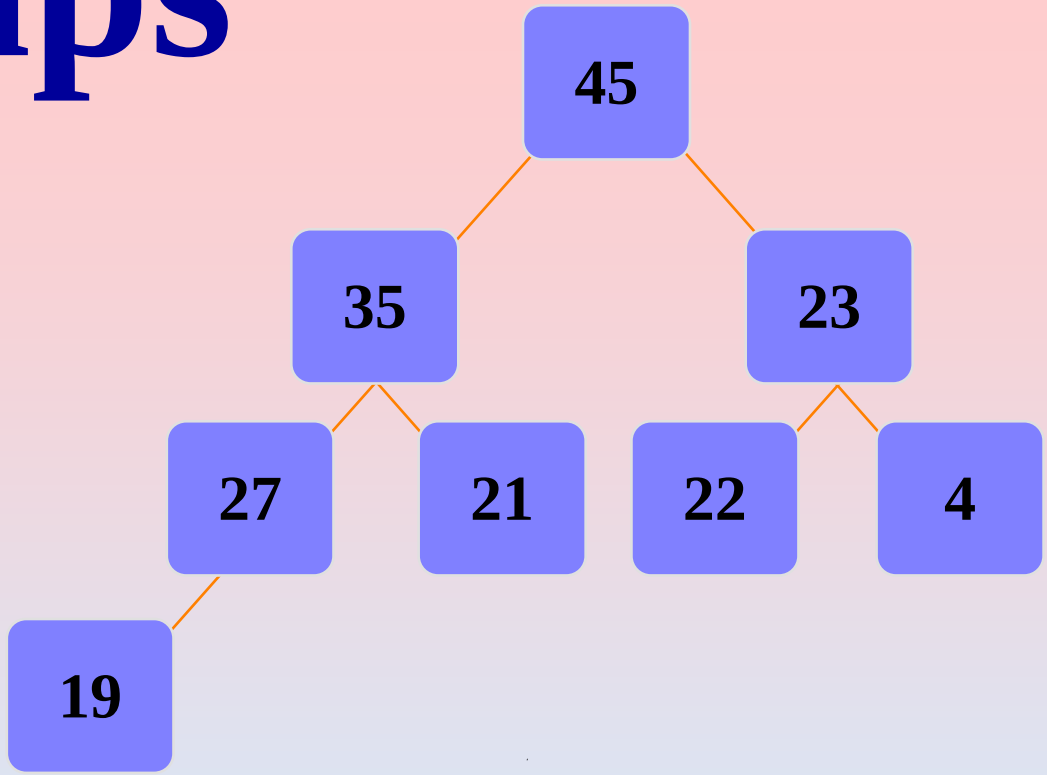


Heaps

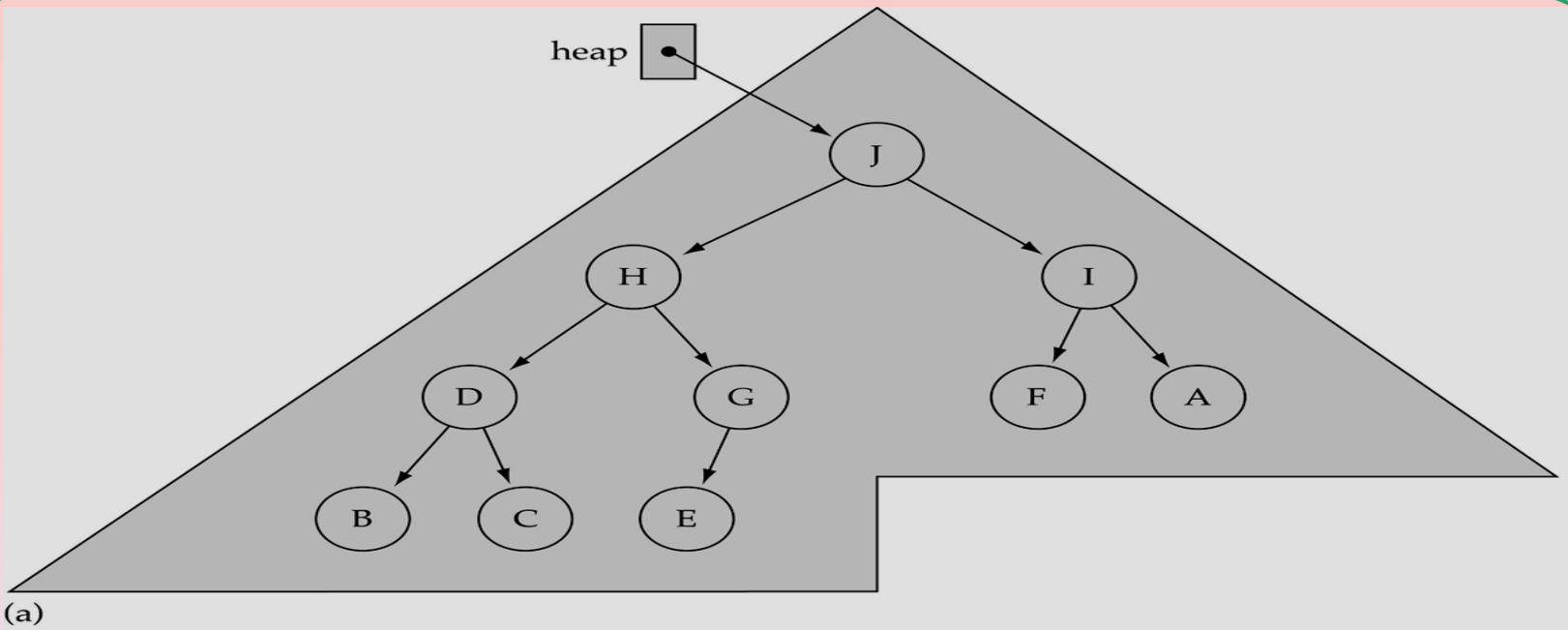


Each node in a heap contains a key that can be compared to other nodes' keys.

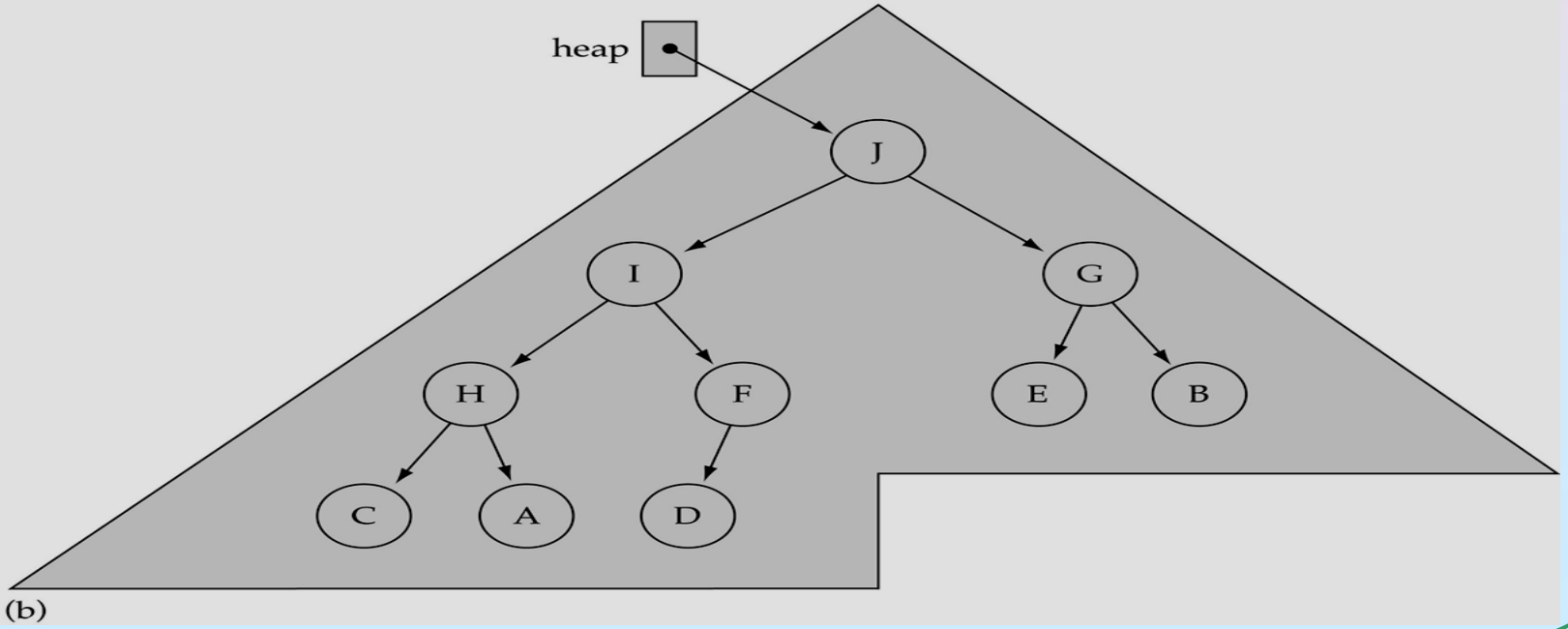
Heaps



The "heap property“ requires that each node's key is >= the keys of its children



(a)



(b)

Two Special Heaps

- **Max-Heap**
- **Min-Heap**

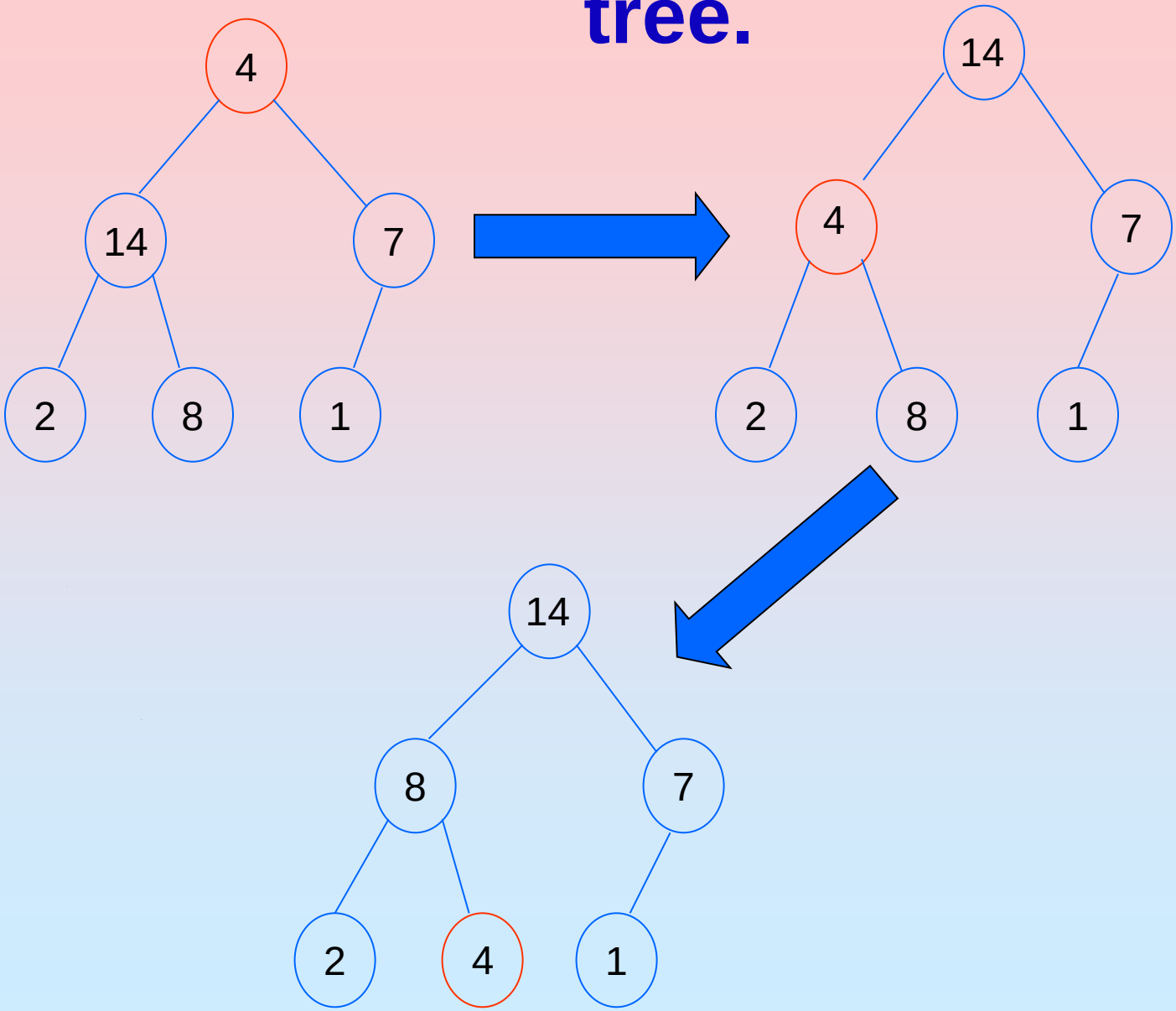
In a max - heap, every node i other than the root satisfies the following property :

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

In a min - heap, every node i other than the root satisfies the following property :

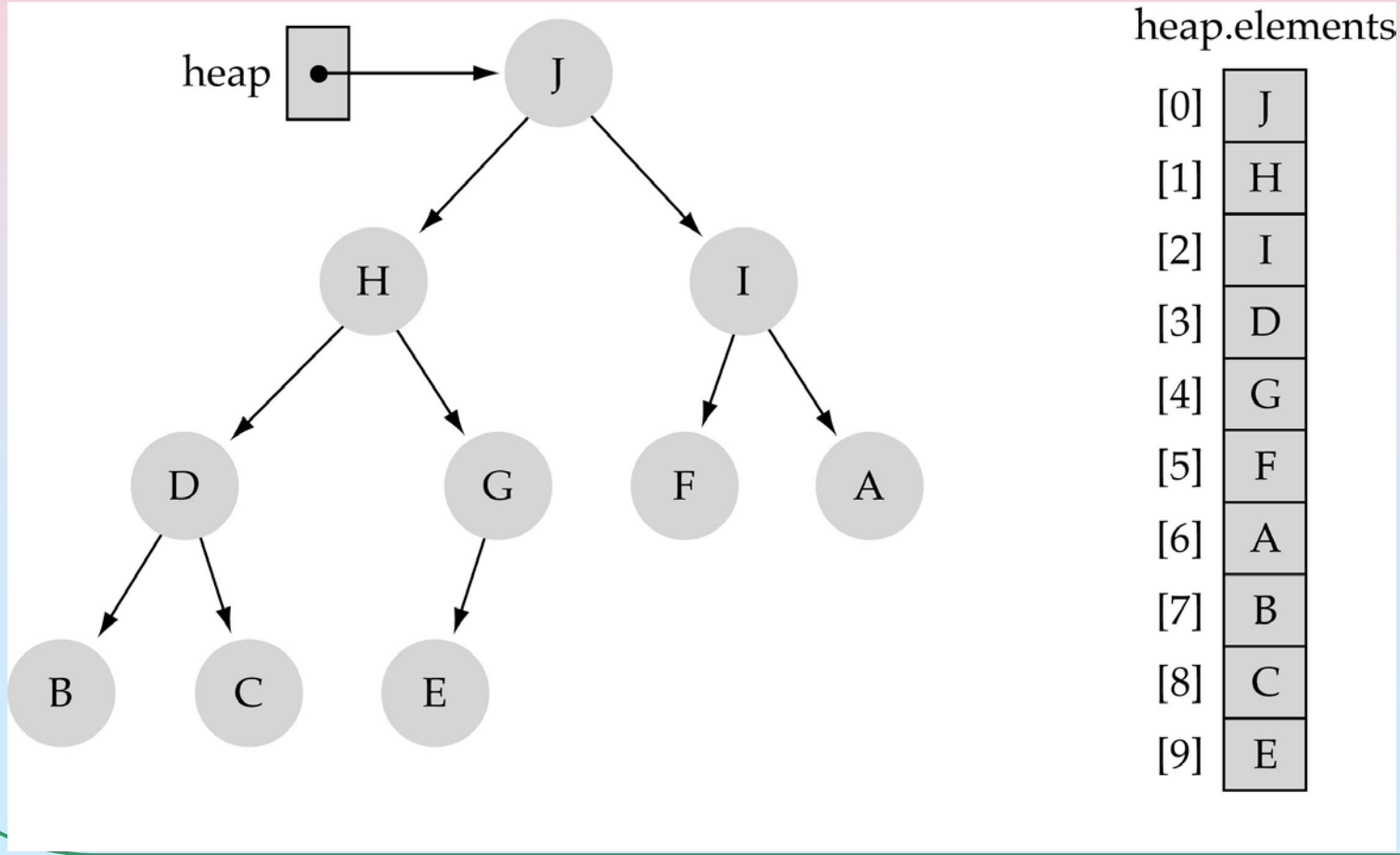
$$A[\text{Parent}(i)] \leq A[i].$$

Heapify - creating a heap from a binary tree.



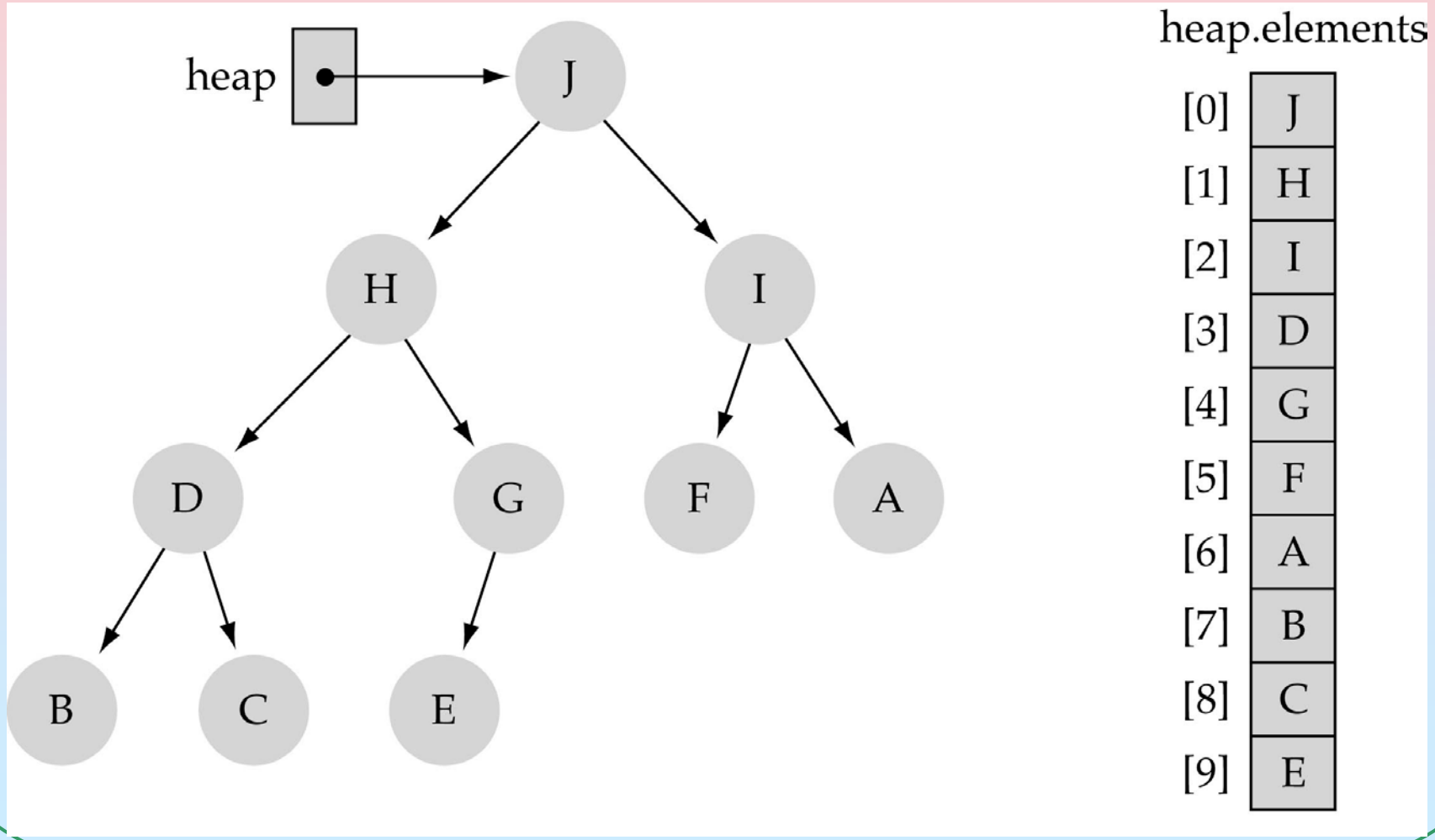
Largest heap element

- From *Property 2*, the largest value of the heap is always stored at the root



Heap implementation using array representation

A heap is a complete binary tree, so it is easy to implement heap using an array representation.



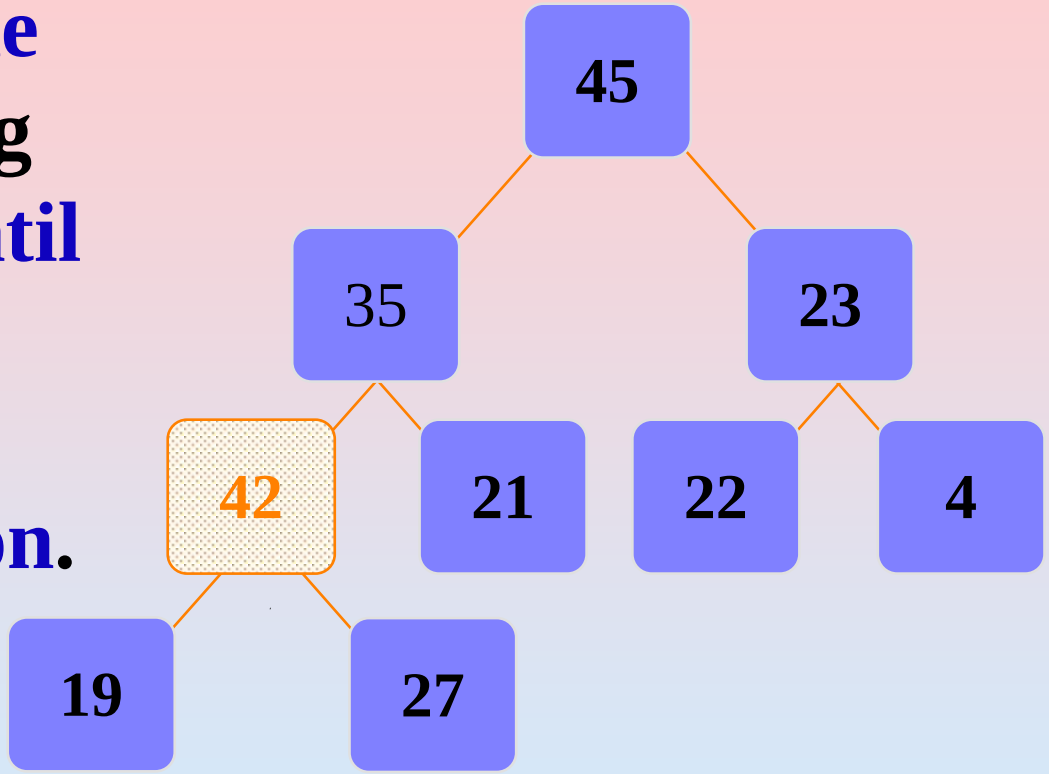
Adding a Node to a Heap

1. Put the new node in the next available spot.
2. Push the new node upward, swapping with its parent until the new node reaches an acceptable location.

Any Idea ?

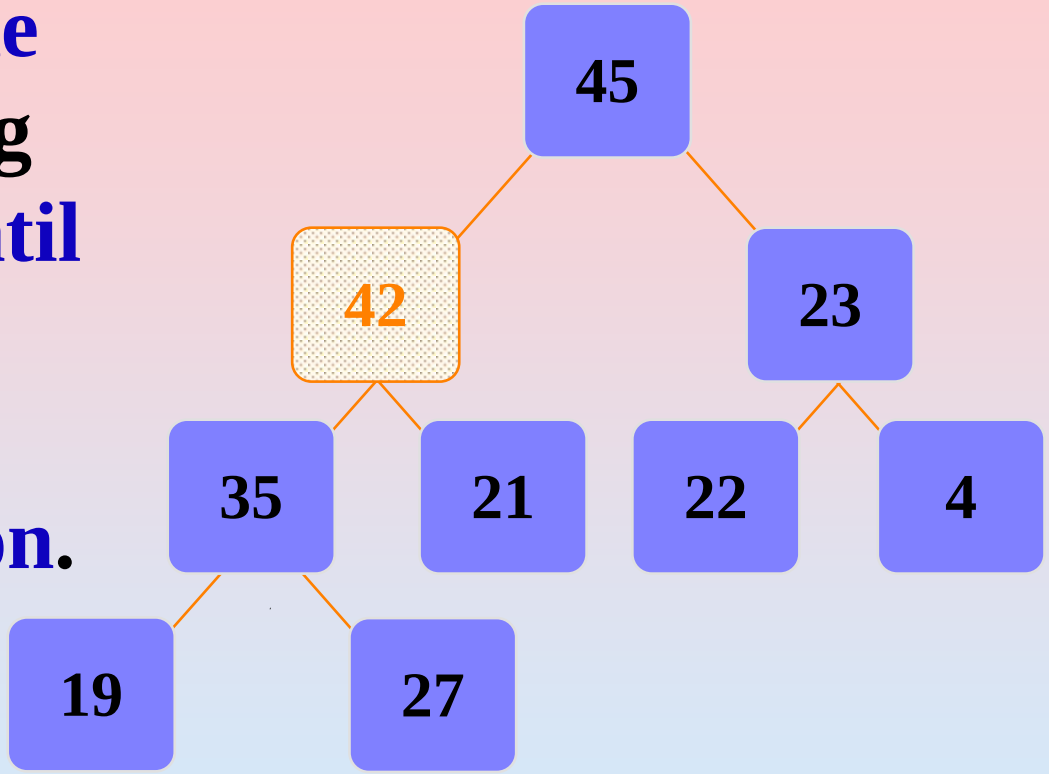
Adding a Node to a Heap

2. **Push the new node upward, swapping with its parent until the new node reaches an acceptable location.**



Adding a Node to a Heap

2. **Push the new node upward, swapping with its parent until the new node reaches an acceptable location.**

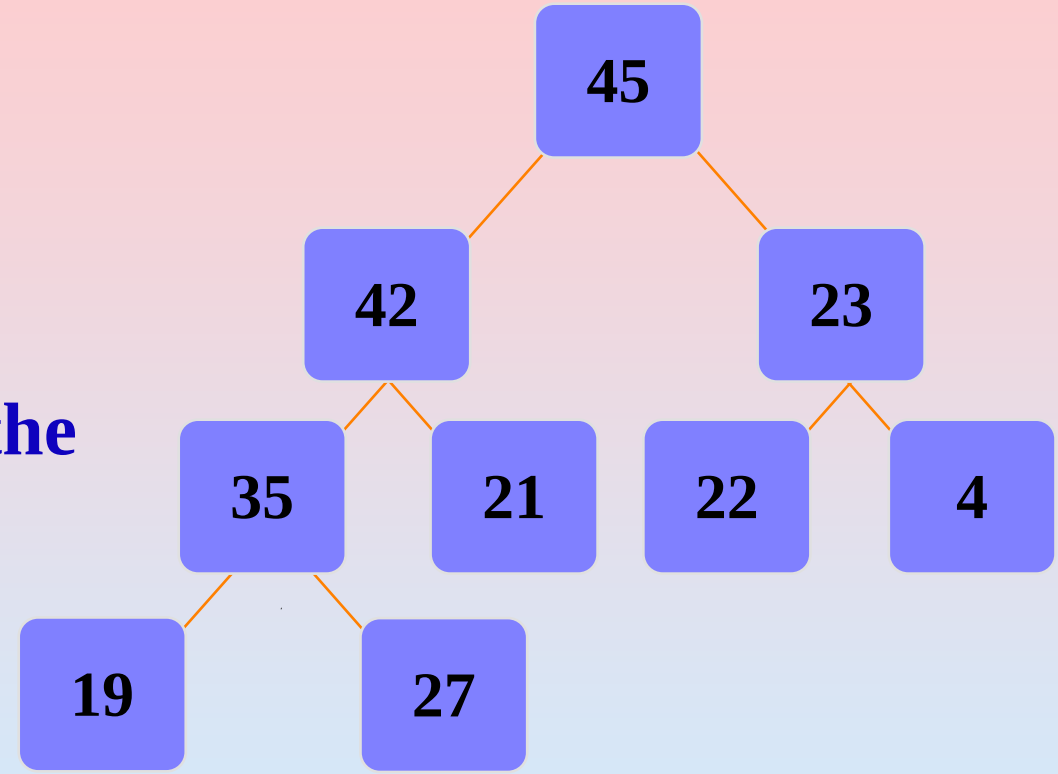


Adding a Node to a Heap

- The parent has a key that is >= new node,

  OR

- The node reaches the root.



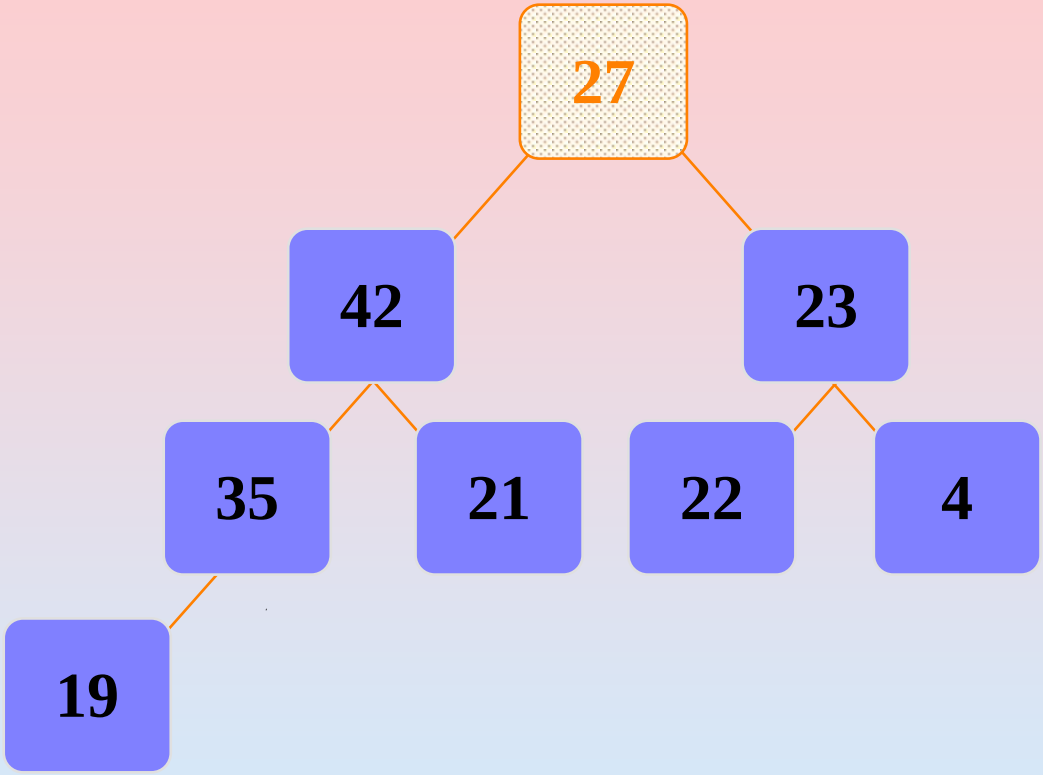
The process of pushing the new node upward is called Reheapification Upward.

Removing the Top of a Heap

Any Idea ?

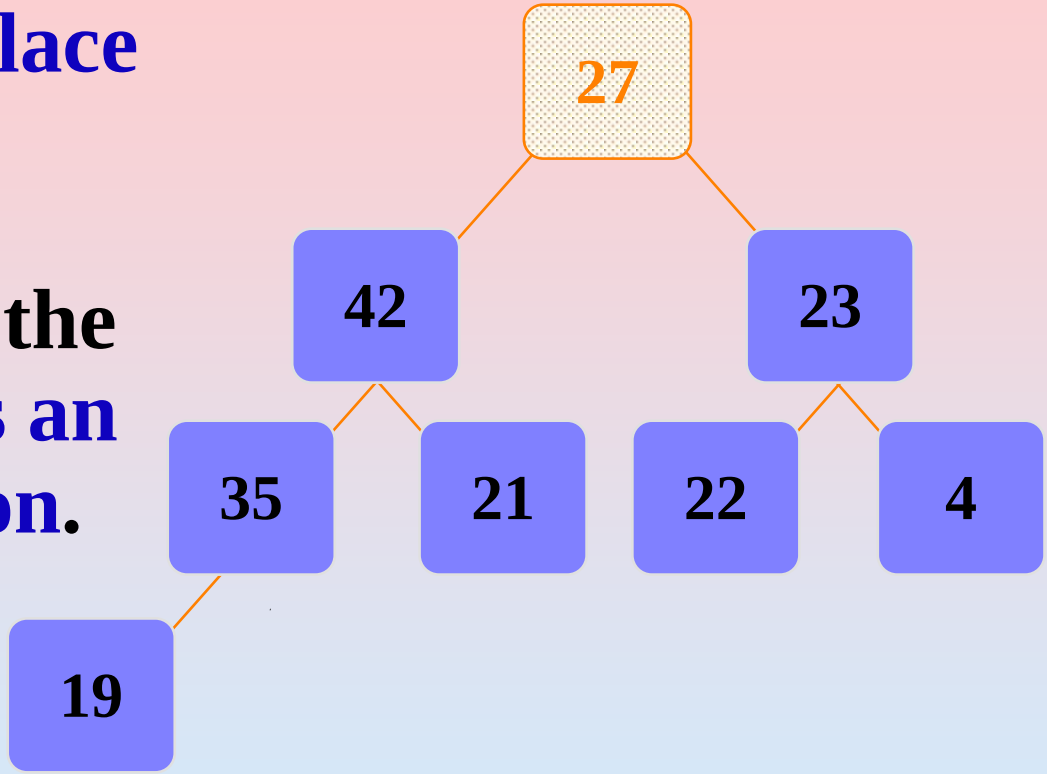
1. Move the last node onto the root.

Removing the Top of a Heap



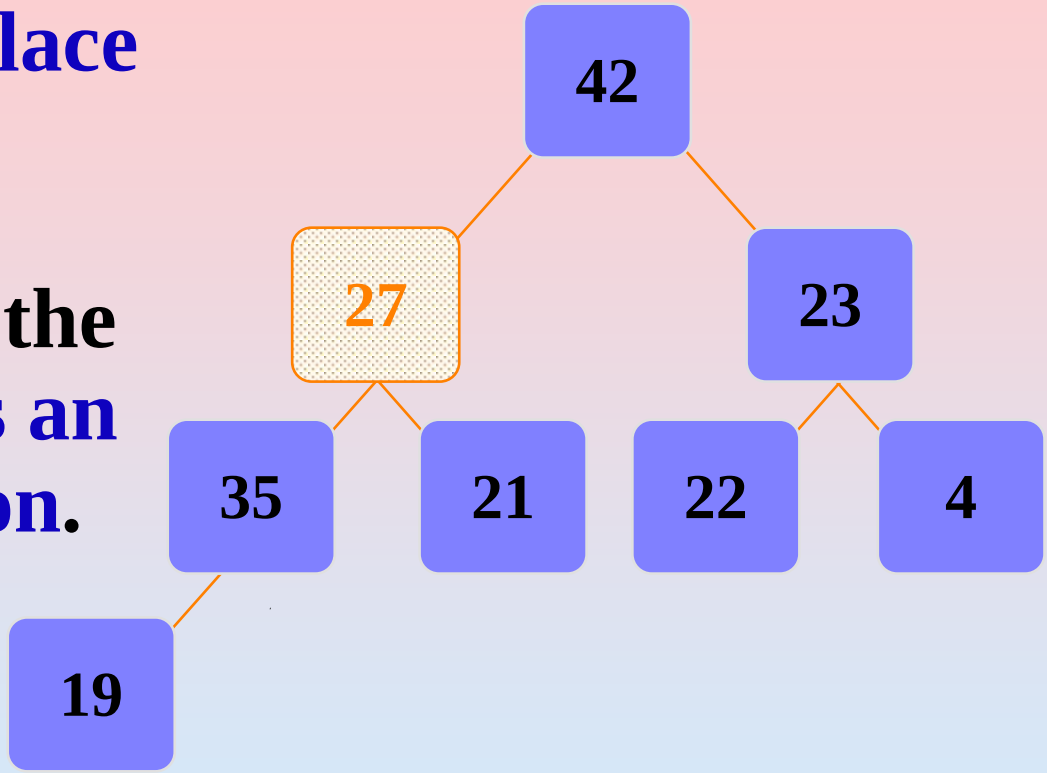
Removing the Top of a Heap

2. Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.



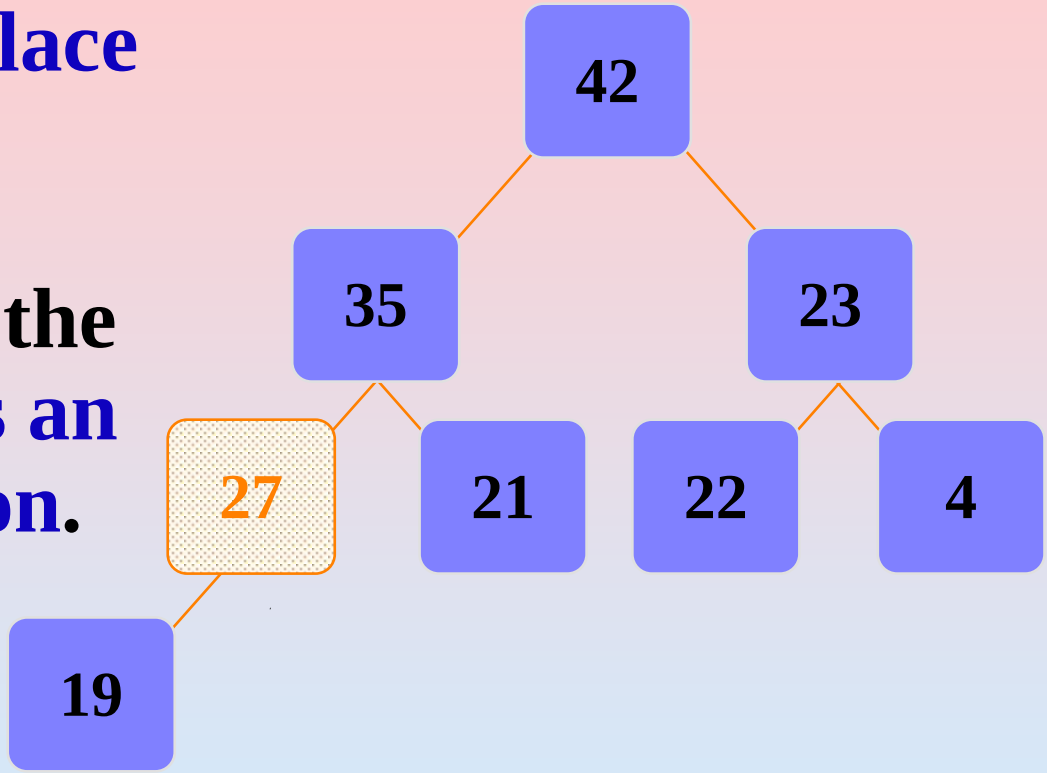
Removing the Top of a Heap

2. Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.



Removing the Top of a Heap

2. Push the out-of-place node downward, swapping with its larger child until the new node reaches an acceptable location.

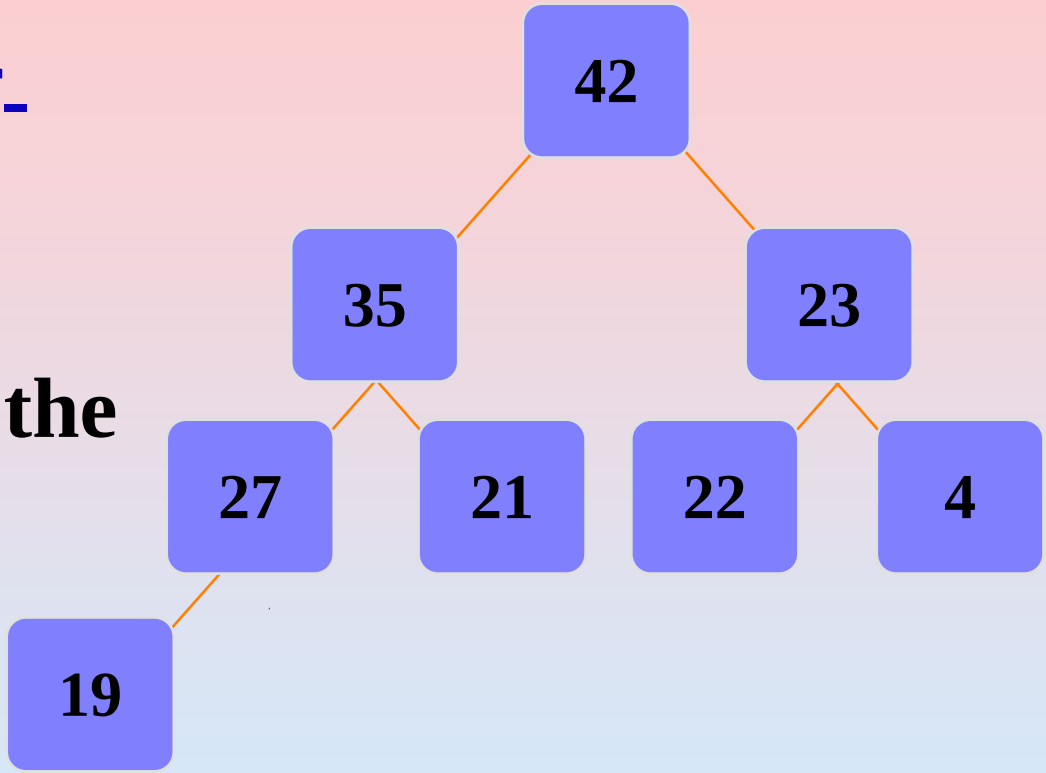


Removing the Top of a Heap

- All children have keys <= the out-of-place node,

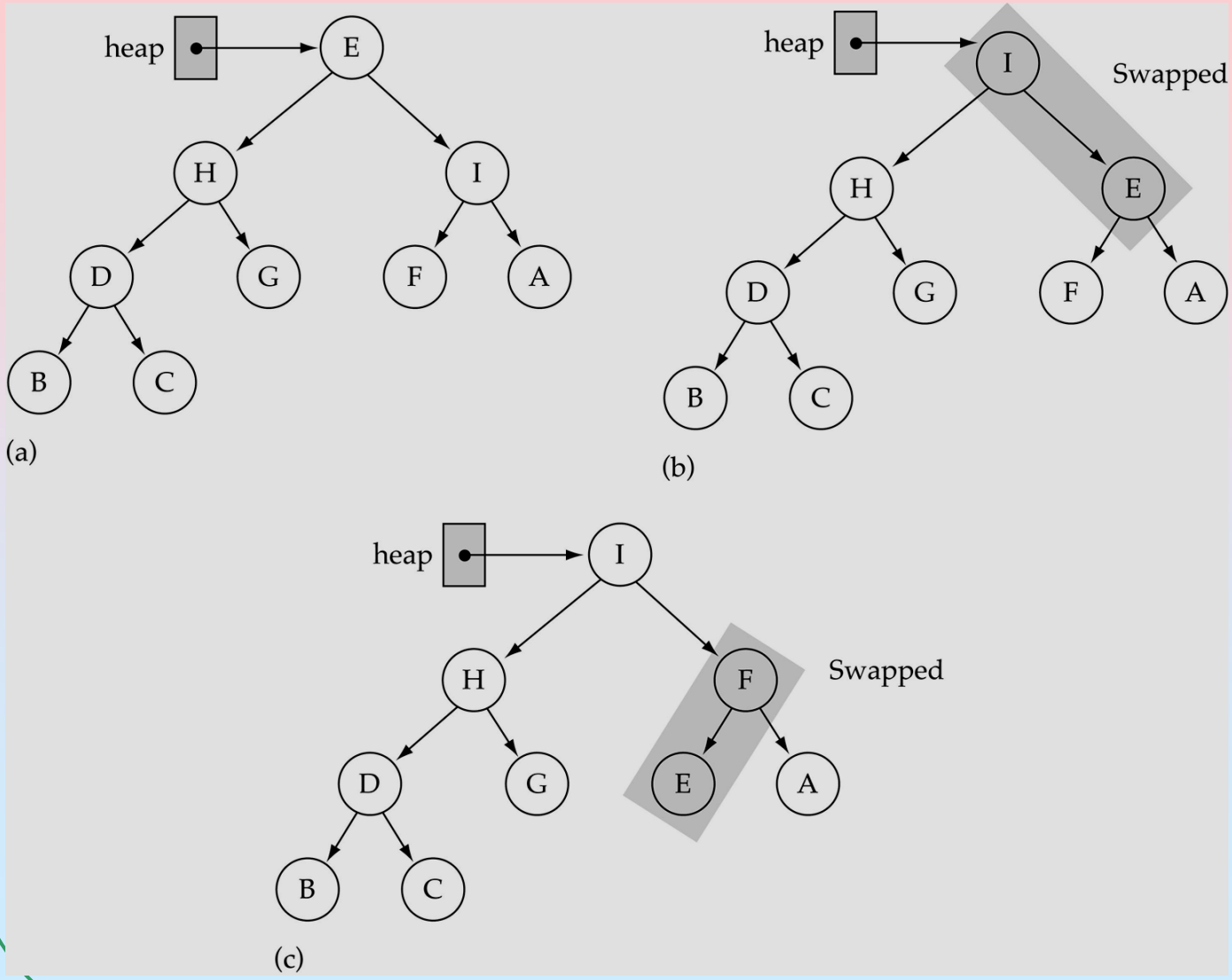
  OR

- The node reaches the leaf.



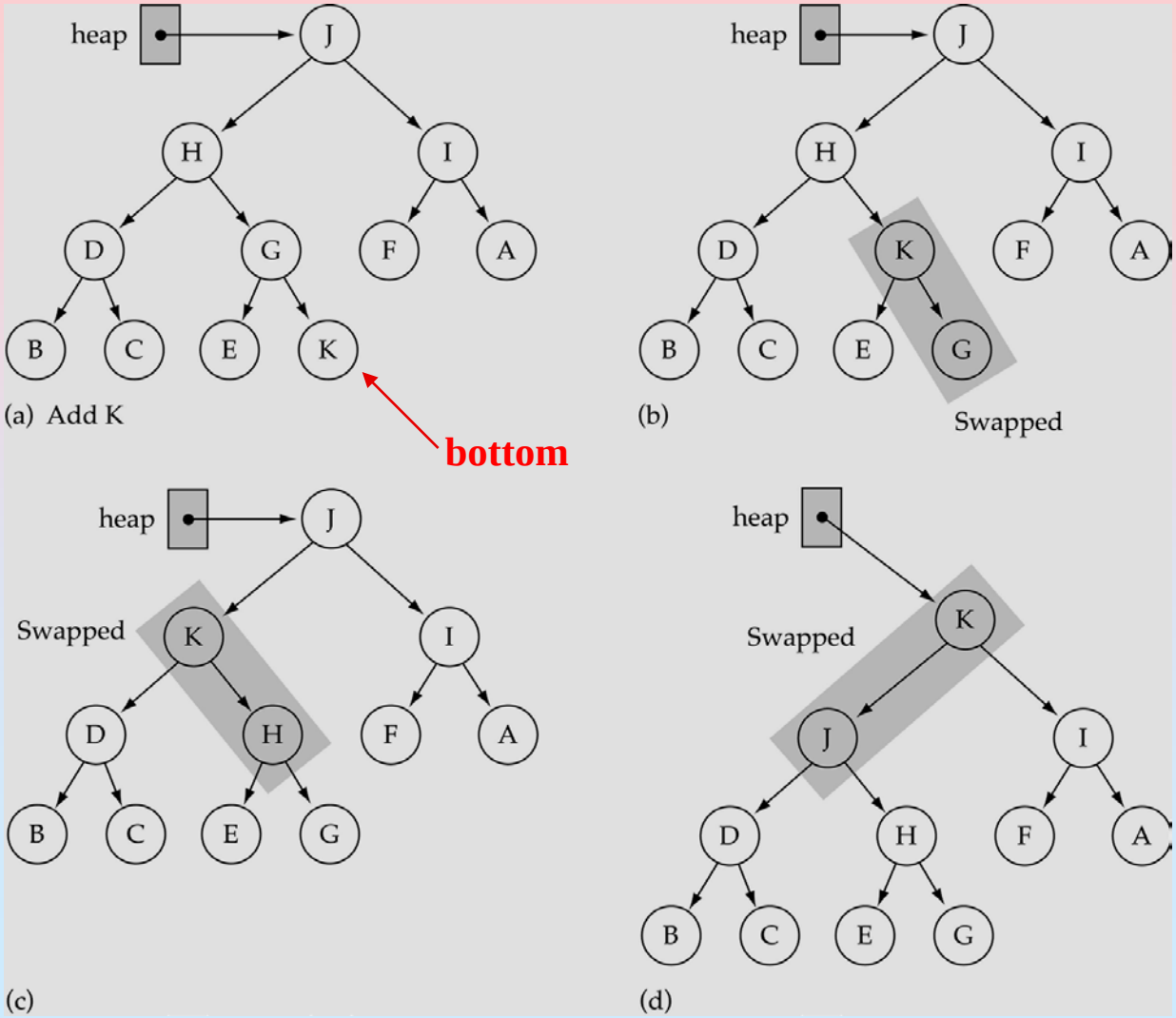
The process of pushing the new node downward is called Reheapification Downward.

The ReheapDown function (used by deleteItem)



Assumption: heap property is violated at the root of the tree

The ReheapUp function (used by insertItem)



Assumption: heap property is violated at the rightmost node at the last level of the tree

Removing the largest element from the Heap

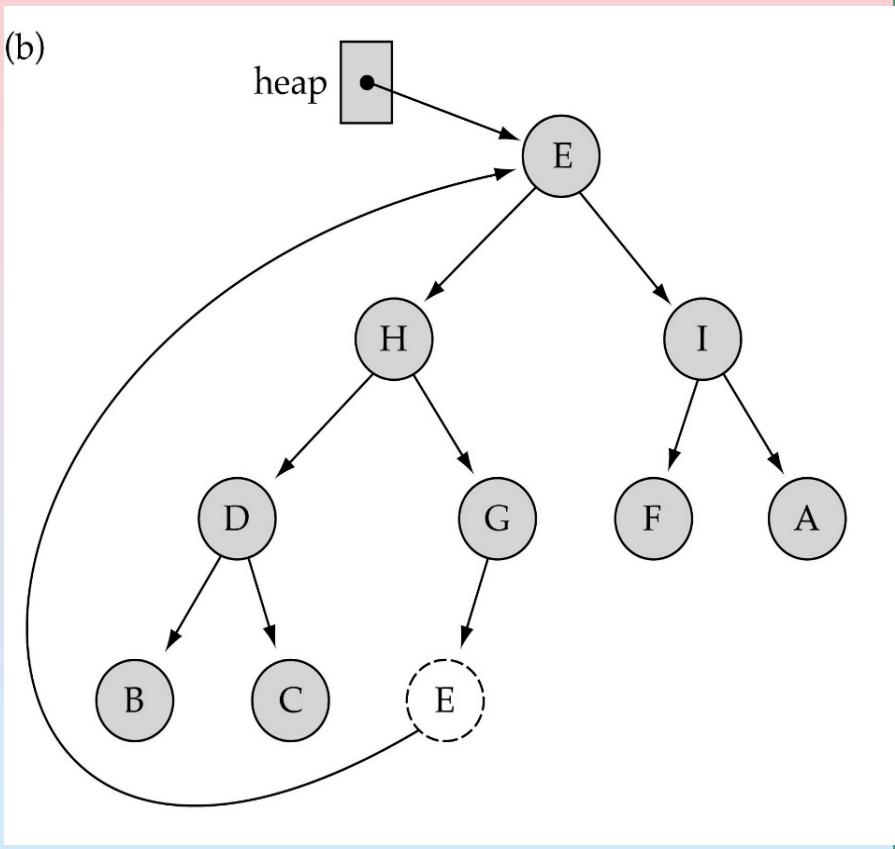
Any Idea ?

(1) Copy the bottom rightmost element to the root

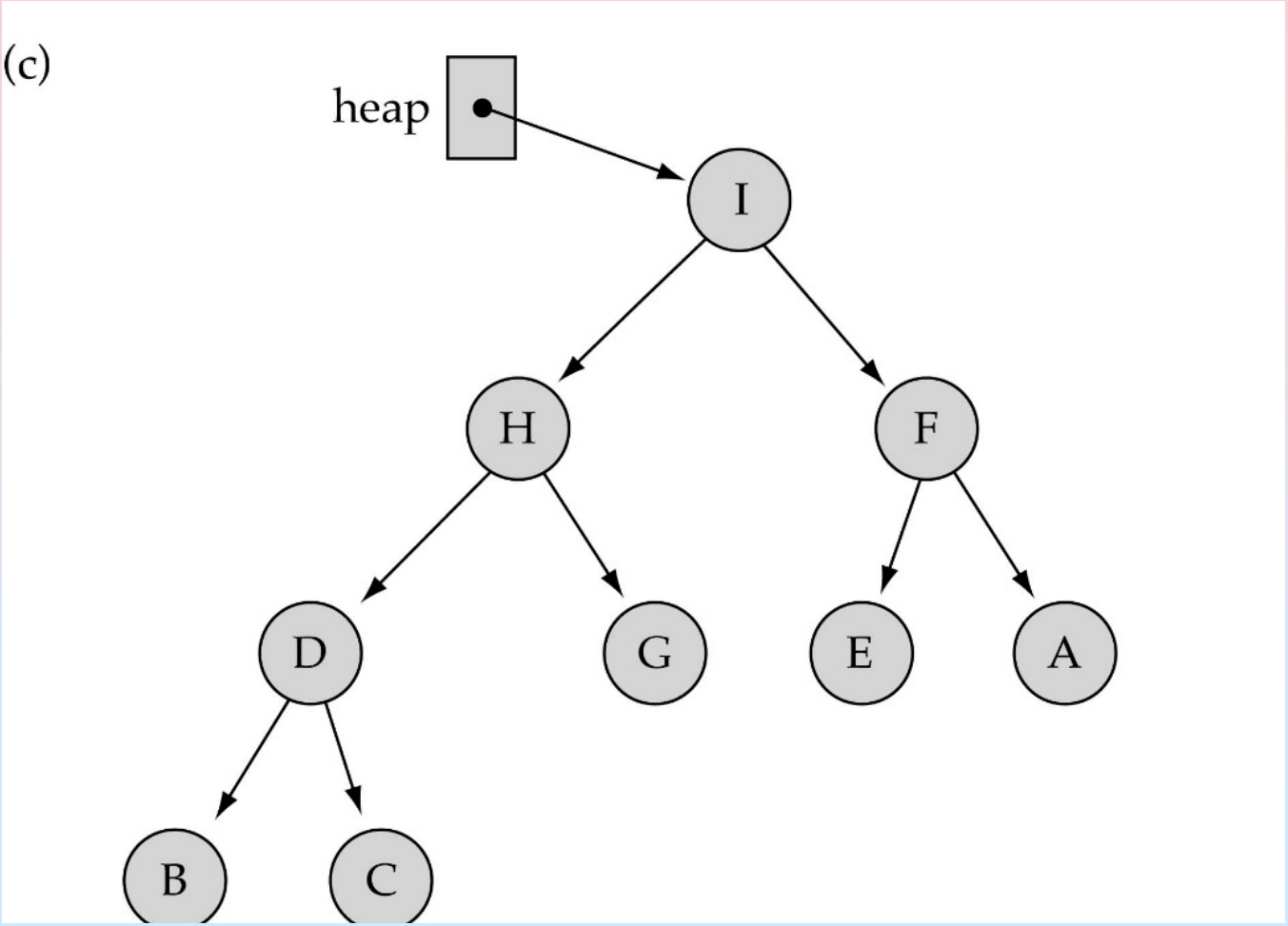
(2) Delete the bottom rightmost node

(3) Fix the heap property by calling *ReheapDown*

Removing the largest element from the heap (cont.)



Removing the largest element from the heap (cont.)



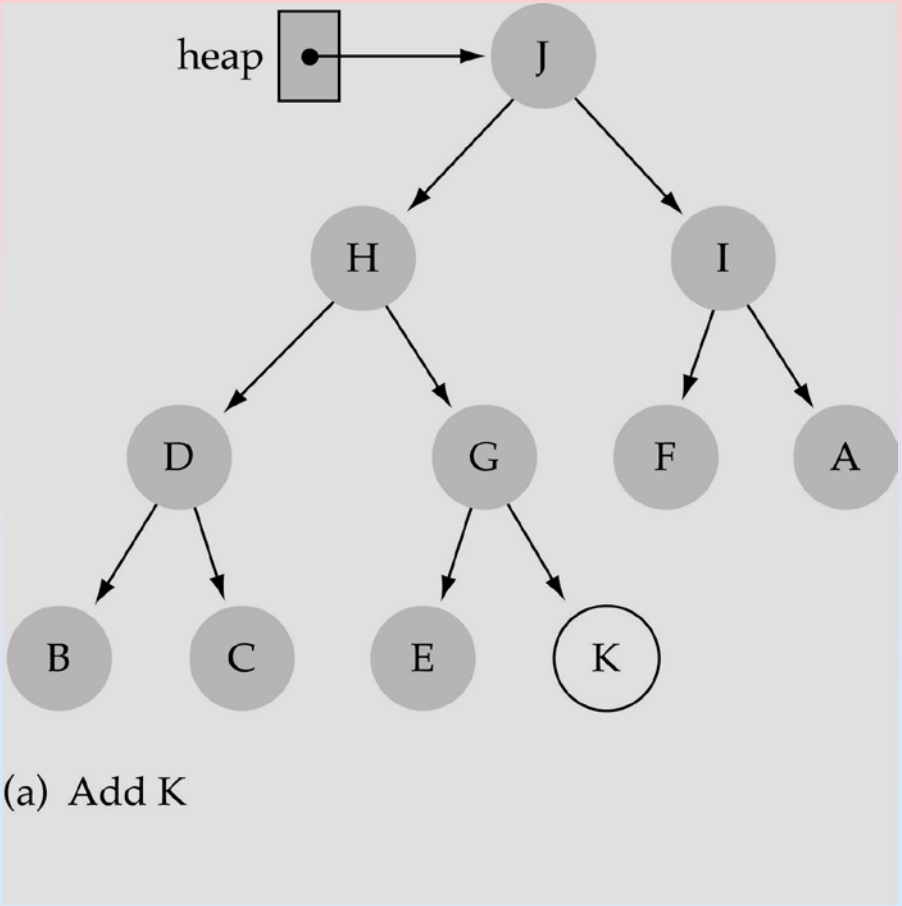
Inserting a new element into the heap

Any Idea ?

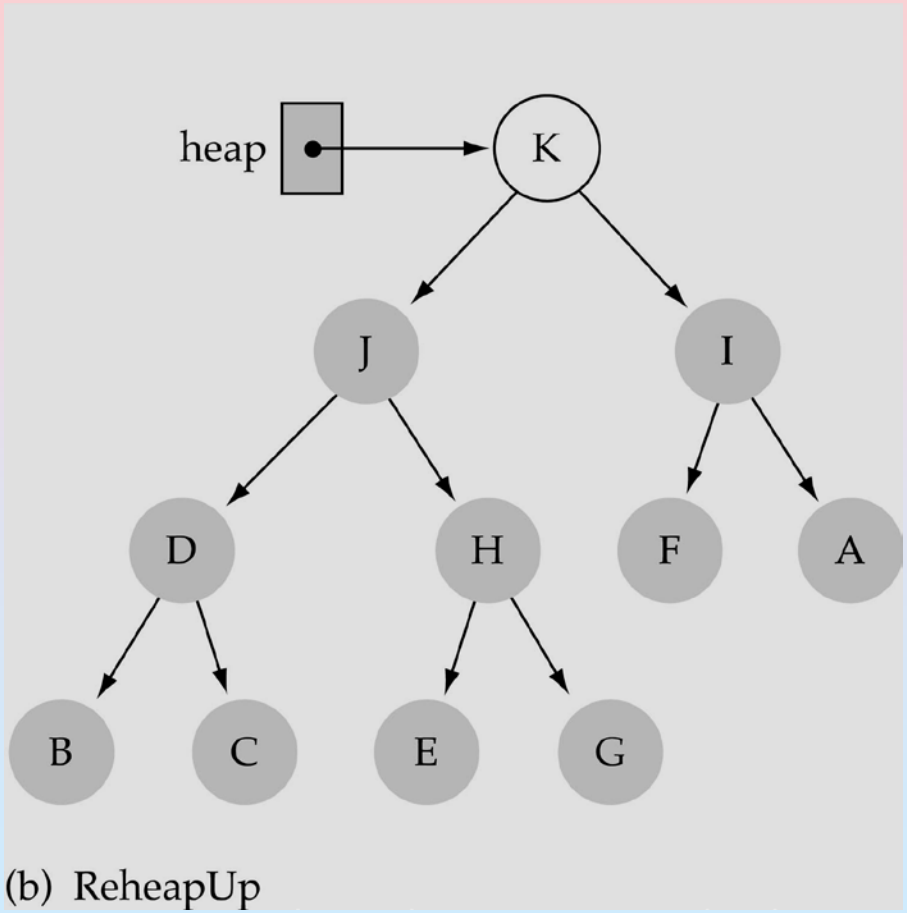
(1) Insert the new element in the next bottom **leftmost** place

(2) Fix the heap property by calling *ReheapUp*

Inserting a new element into the heap (cont.)



(a) Add K



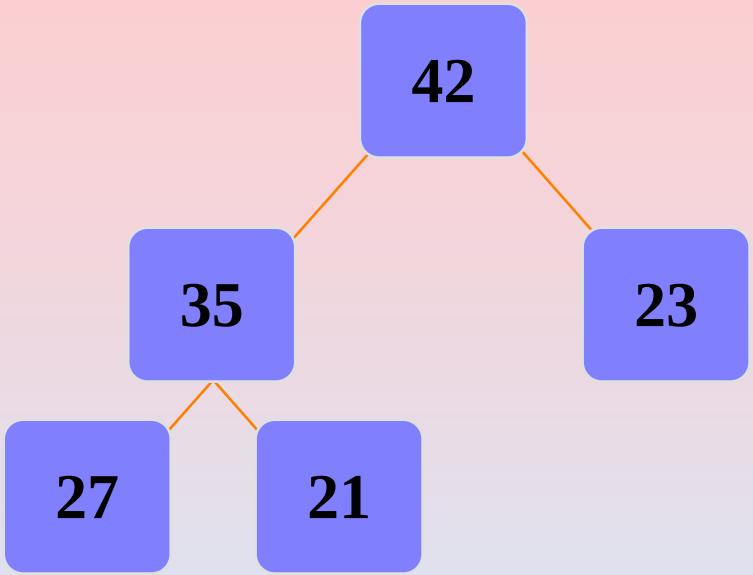
(b) ReheapUp

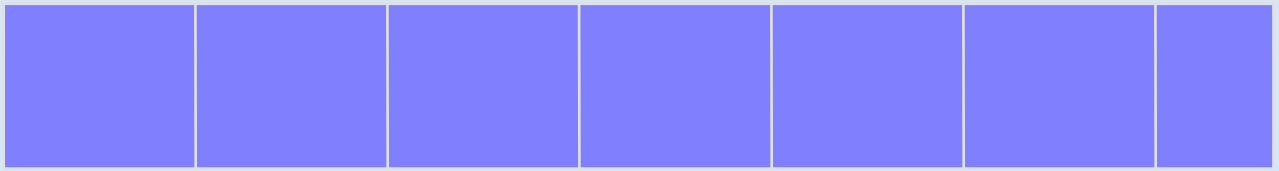
Does it sastisfy Hap property?

Fix the heap property.

Implementing a Heap

- **Store the data from the nodes in a partially-filled array.**

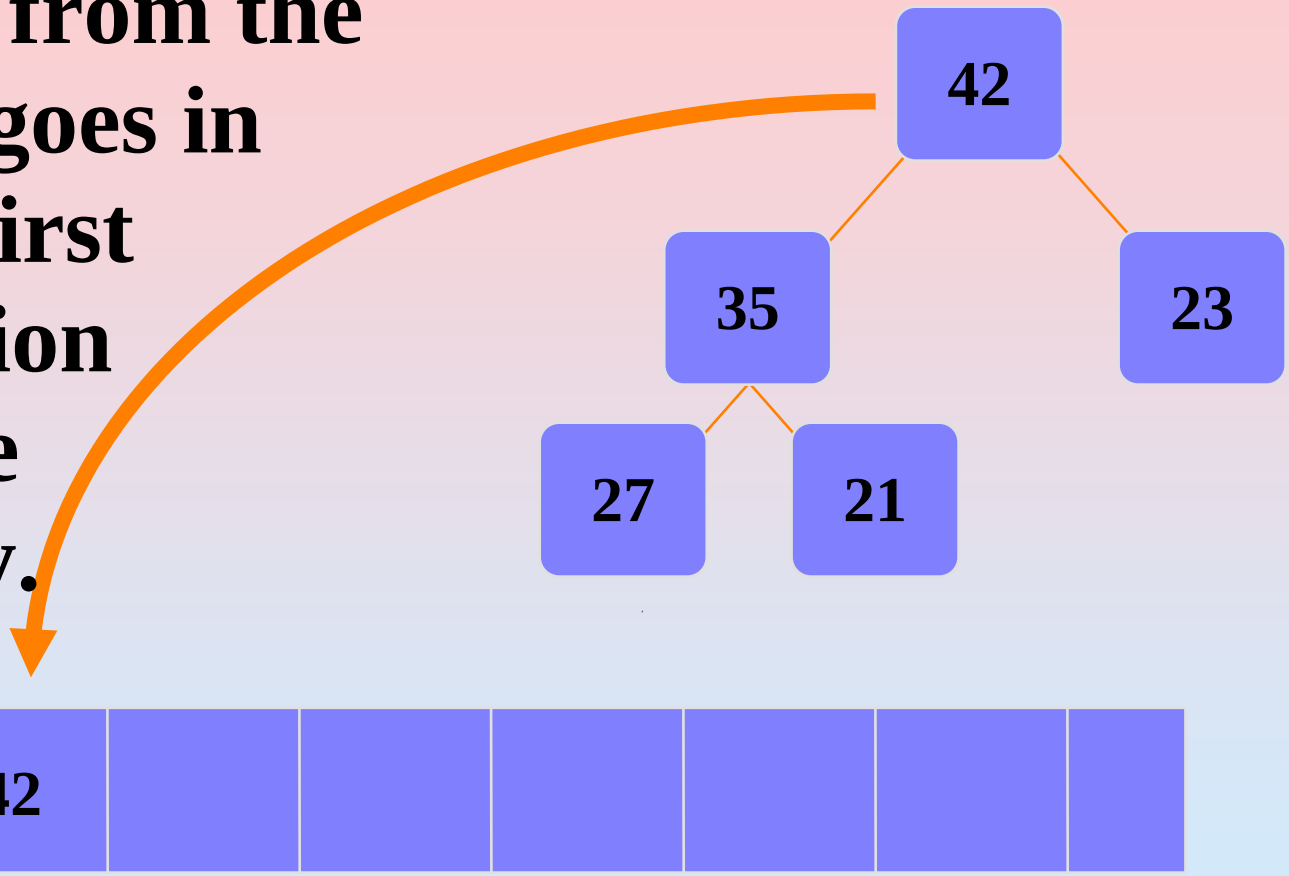




An array of data

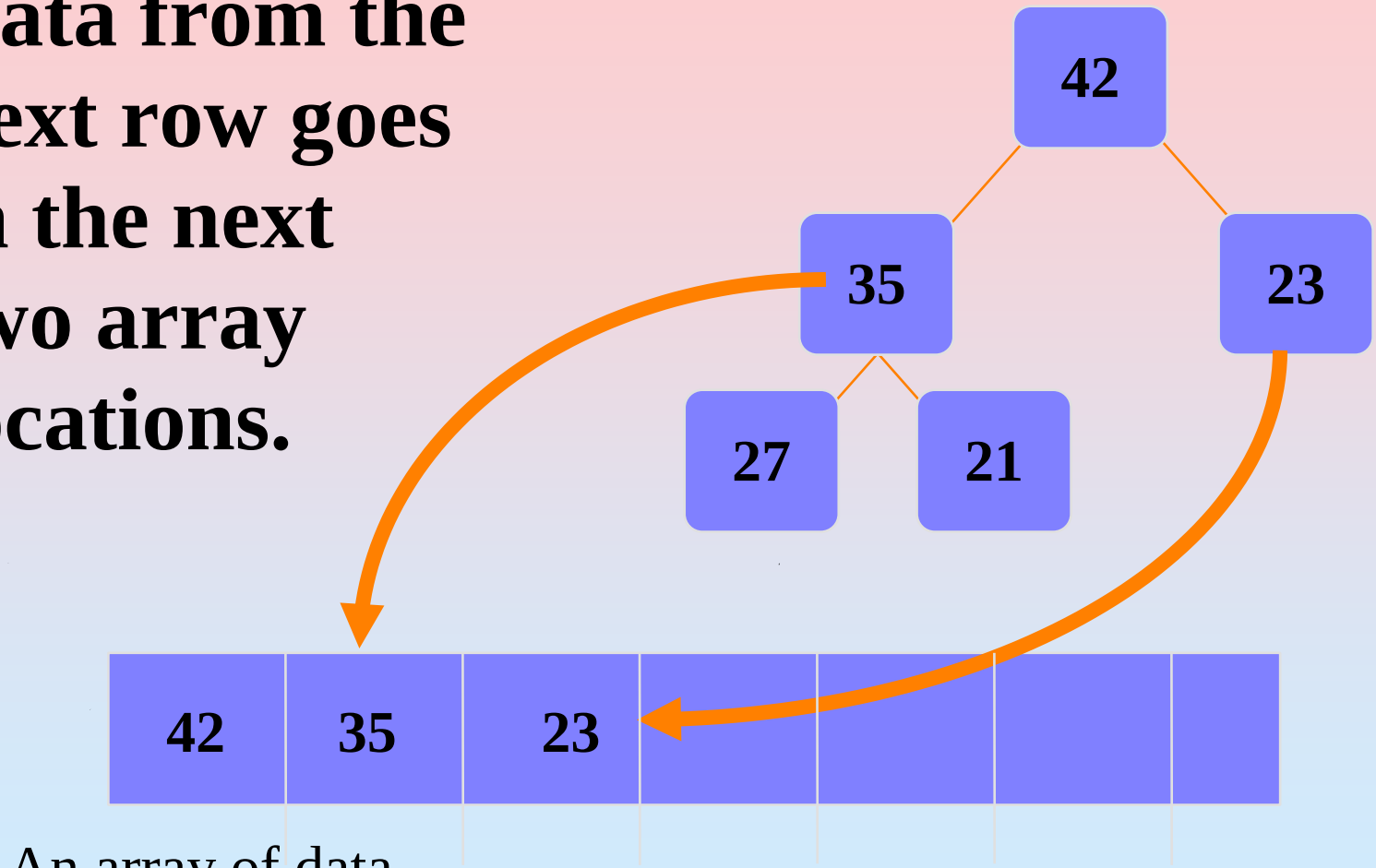
Implementing a Heap

- **Data from the root goes in the first location of the array.**



Implementing a Heap

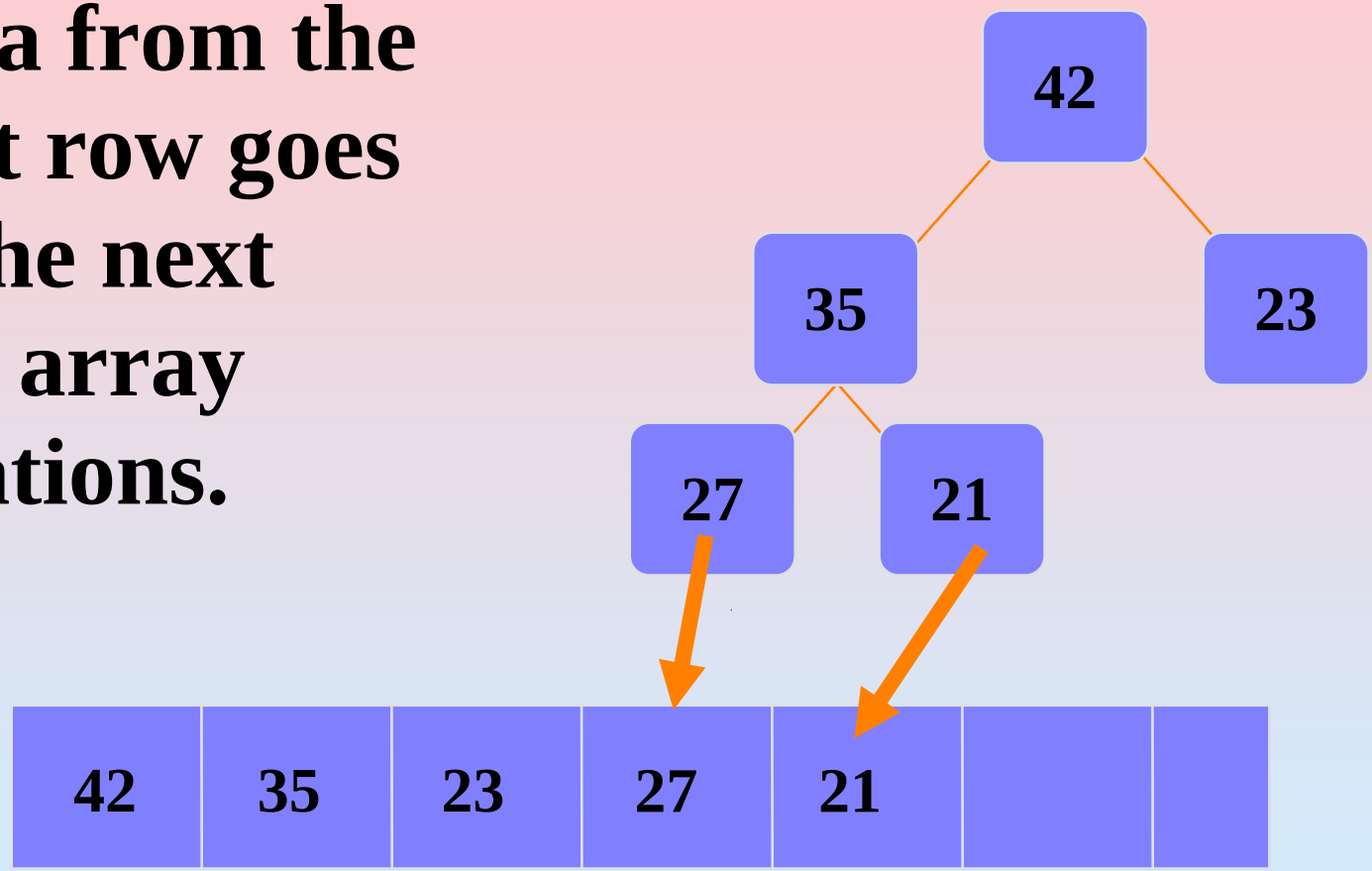
- **Data from the next row goes in the next two array locations.**



An array of data

Implementing a Heap

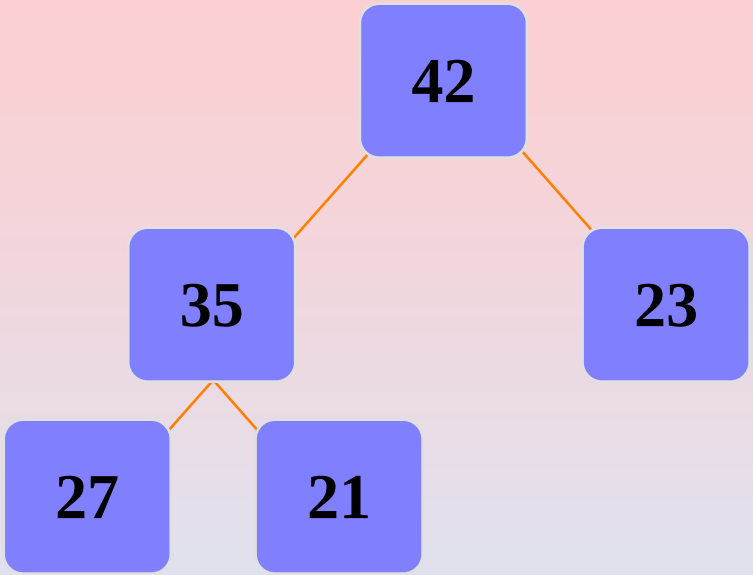
- **Data from the next row goes in the next two array locations.**



An array of data

Implementing a Heap

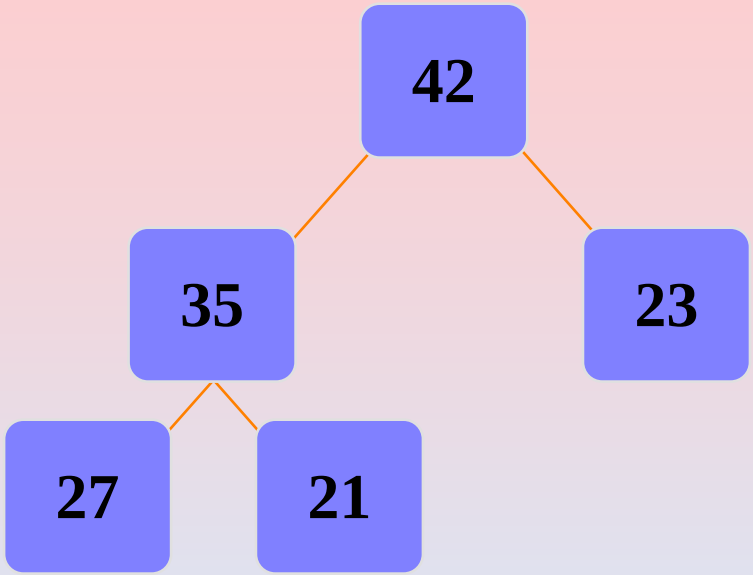
- **Data from the next row goes in the next two array locations.**



We don't care what's in this part of the array.

Important Points about the Implementation

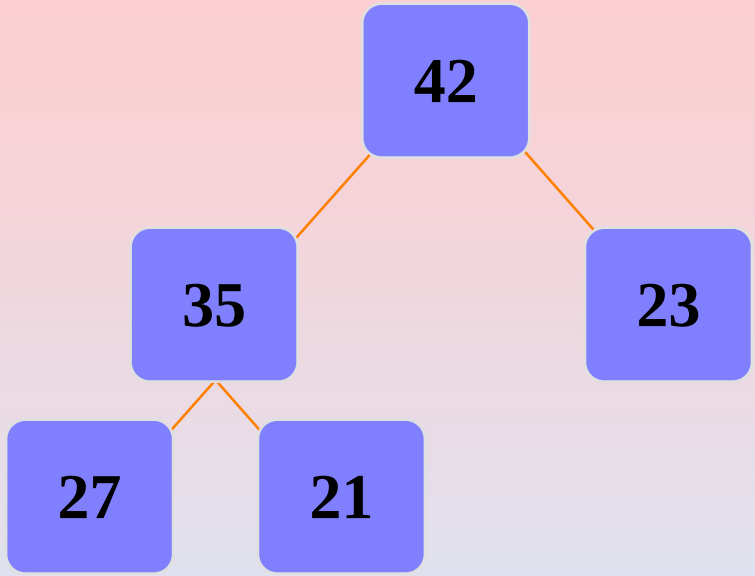
- The links between the tree's nodes are not actually stored as pointers, or in any other way.
- The only way we "know" that "the array is a tree" is from the way we manipulate the data.



42	35	23	27	21		

Important Points about the Implementation

- If you know the index of a node, then it is easy to figure out the indexes of that node's parent and children.



42	35	23	27	21		

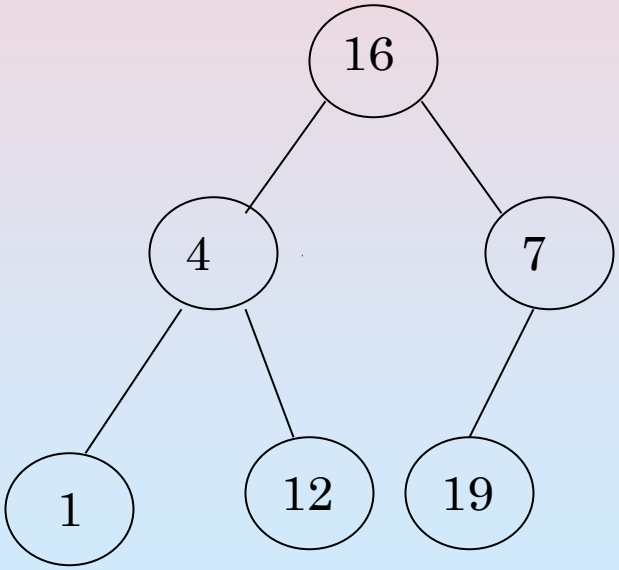
Summary

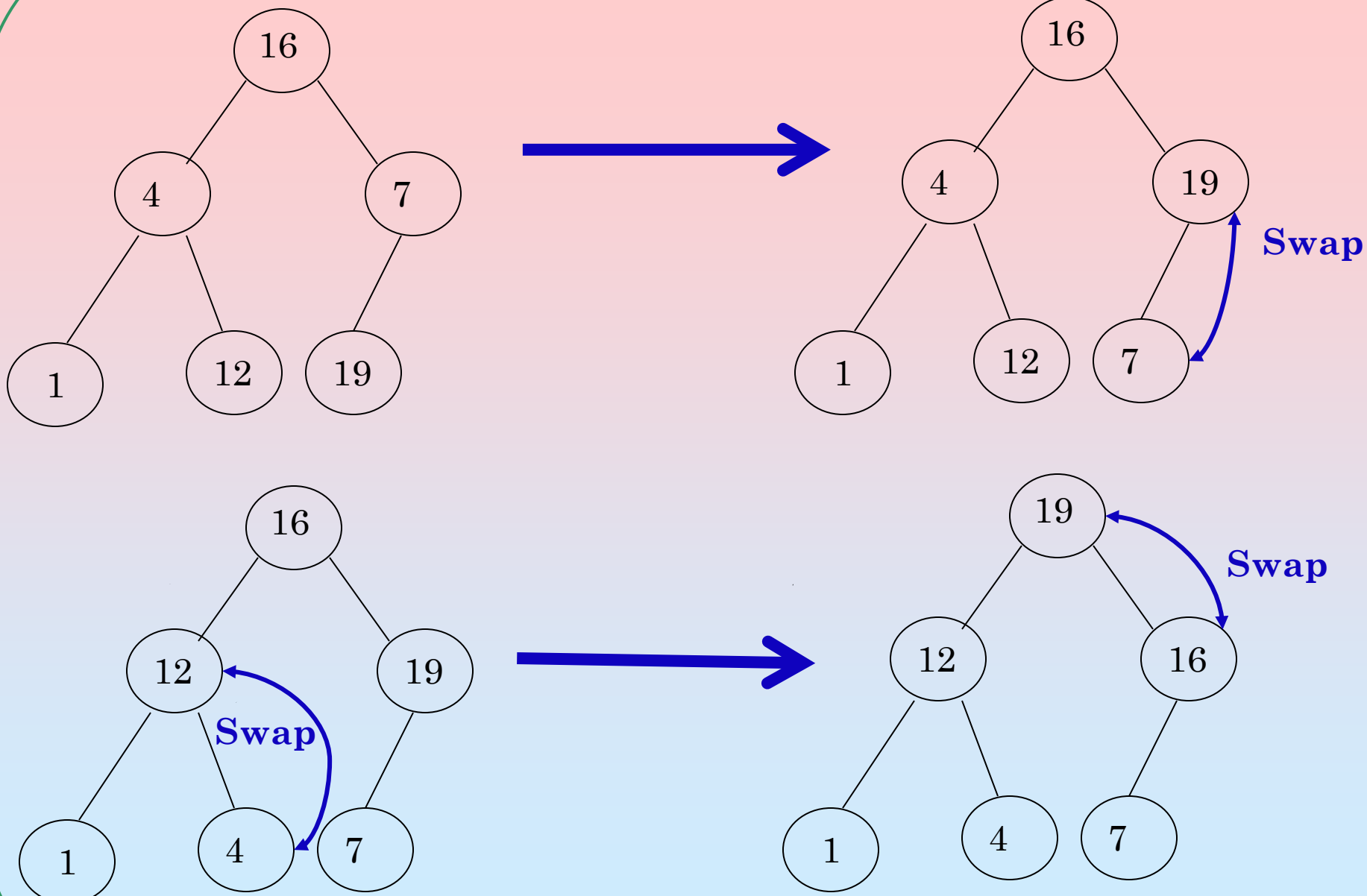
- A heap is a complete binary tree, where the entry at each node is greater than or equal to the entries in its children.
- To add an entry to a heap, place the new entry at the next available spot, and perform a reheapification upward.
- To remove the biggest entry, move the last node onto the root, and perform a reheapification downward.

Example: Convert the following array to a heap

16	4	7	1	12	19

Picture **the array as a complete binary tree:**



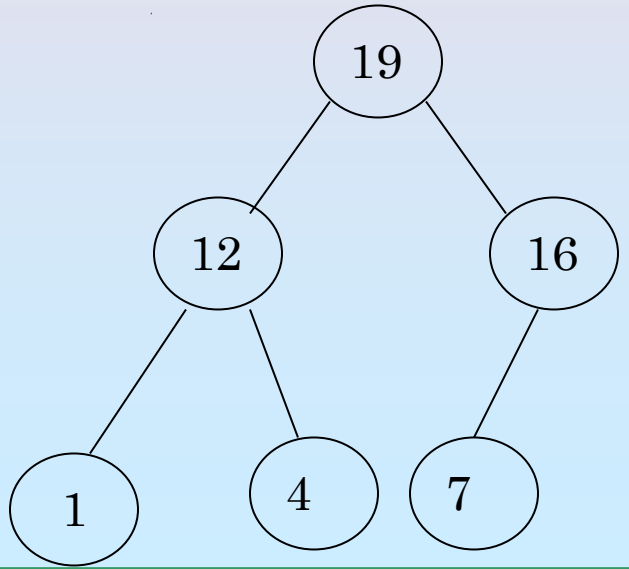

16
4
7
1
12
19
16
4
19
Swap
1
12
7
16
12
19
Swap
1
4
7
19
Swap
12
16
1
4
7

Heap Sort

- The heapsort algorithm consists of two phases:
  - build a heap from an arbitrary array
  - use the heap to sort the data

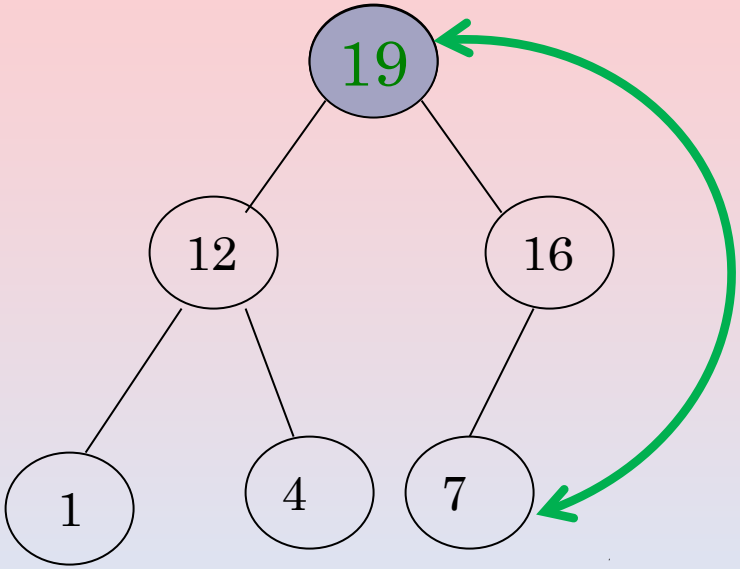
To sort the elements in the decreasing order, use a min heap

- To sort the elements in the increasing order, use a max heap



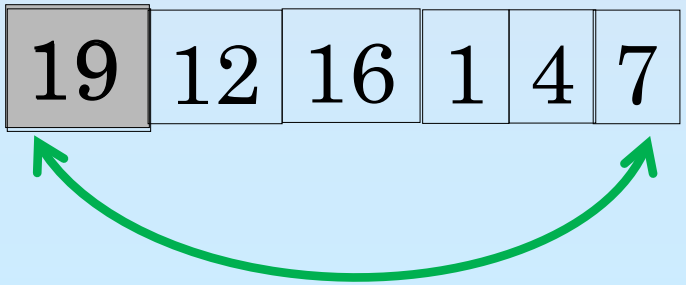
Example of Heap Sort

Place the largest element at right index



Swap the last element with the root

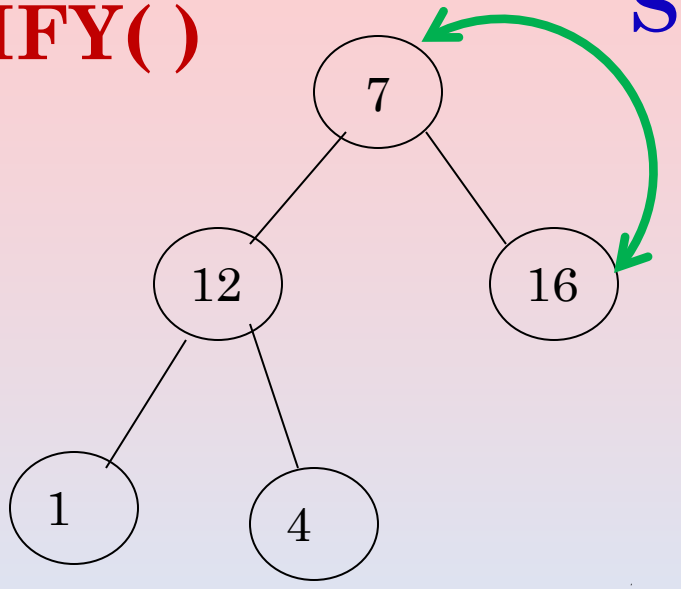
Array representation is



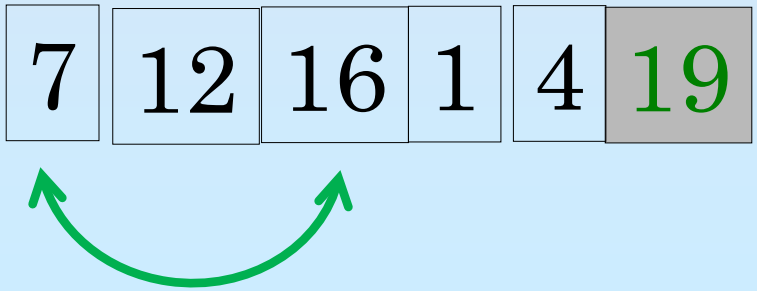
Example of Heap Sort

HEAPIFY( )

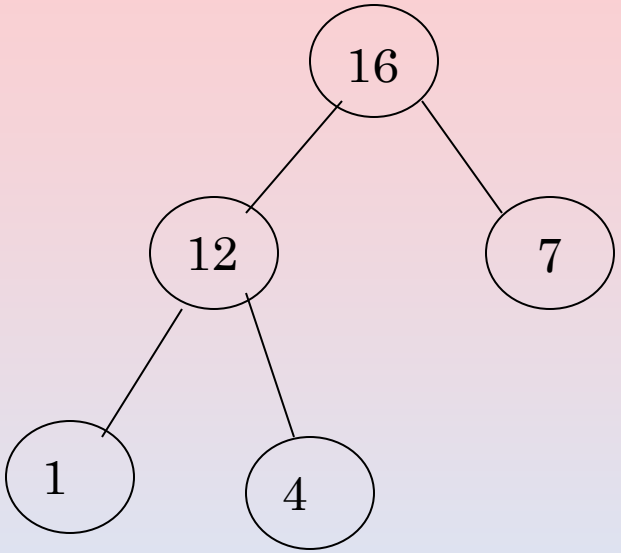
Swap



Unsorted elements **Sorted elements**



Example of Heap Sort

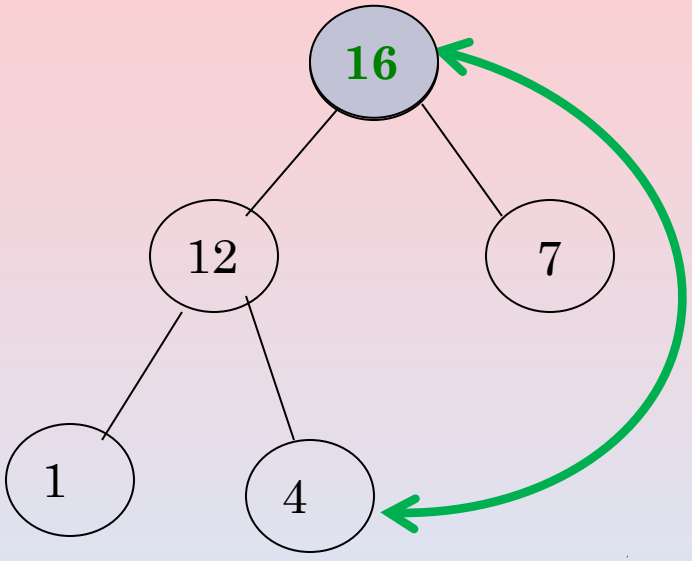


Unsorted elements **Sorted elements**

16	12	7	1	4	19

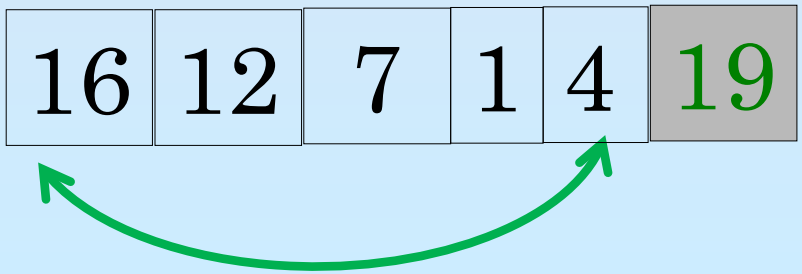
Example of Heap Sort

Place the largest element at right index

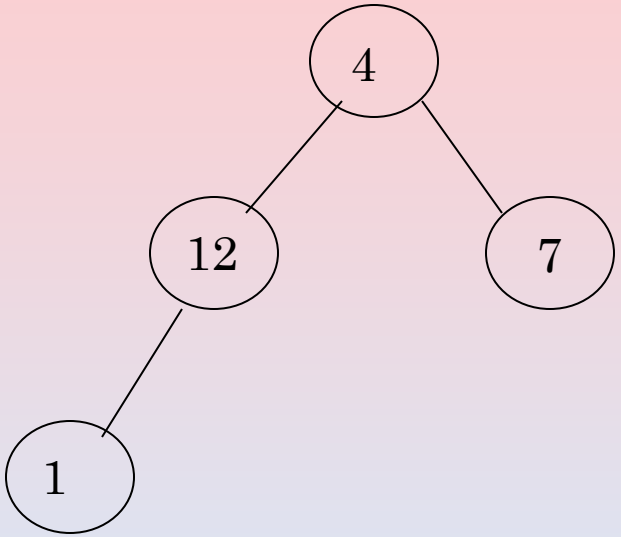


Swap the last element with the root

Unsorted elements | **Sorted elements**



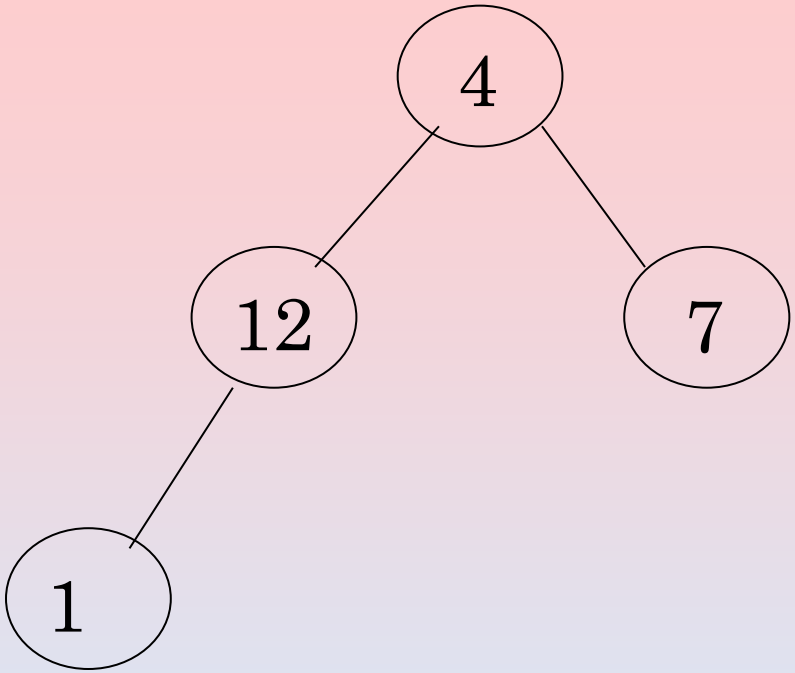
Example of Heap Sort



Unsorted elements **Sorted elements**

4	12	7	1	16	19

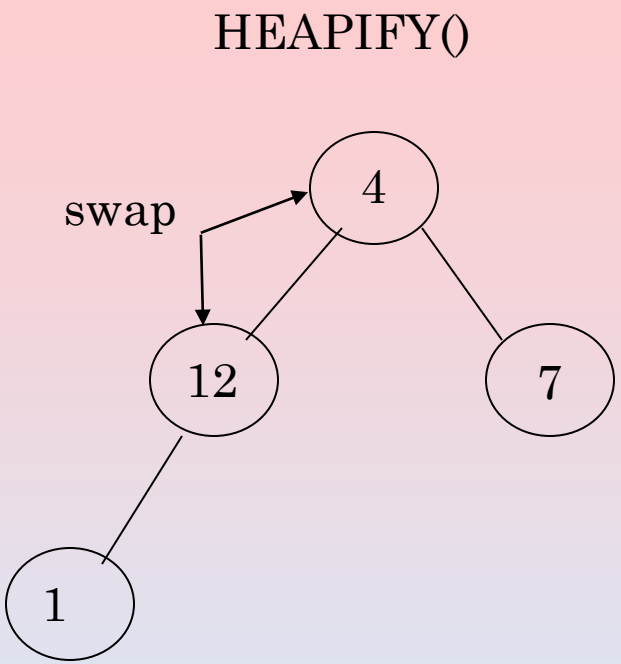
Example of Heap Sort



Unsorted | Sorted:

4	12	7	1	16	19

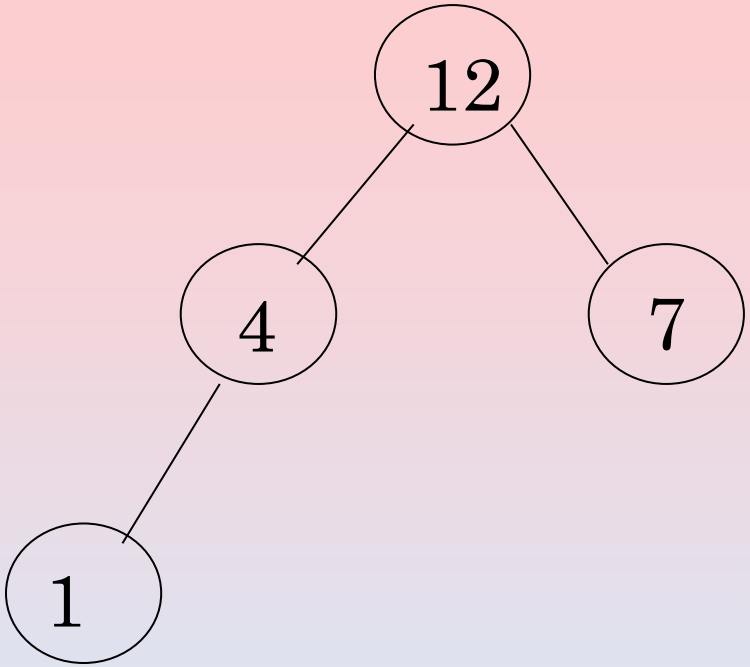
Example of Heap Sort



Unsorted Sorted:

4	12	7	1	16	19

Example of Heap Sort

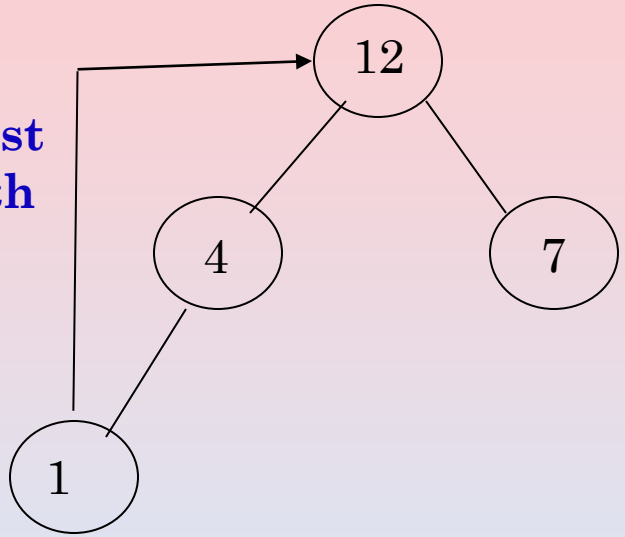


Unsorted Sorted:

12	4	7	1	16	19

Example of Heap Sort

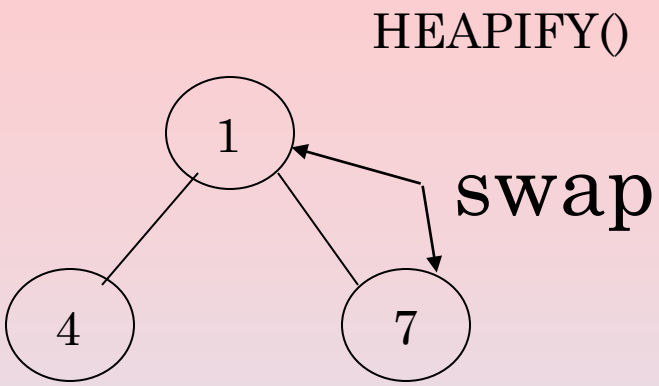
Swap the last element with the root



Unsorted | Sorted:

1	4	7	12	16	19

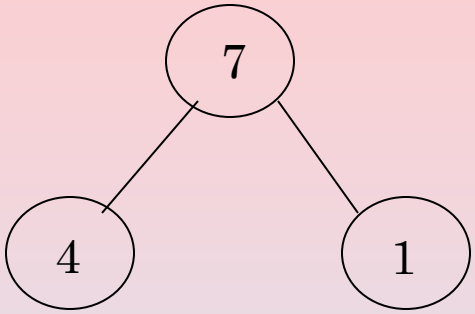
Example of Heap Sort



Unsorted Sorted:

1	4	7	12	16	19

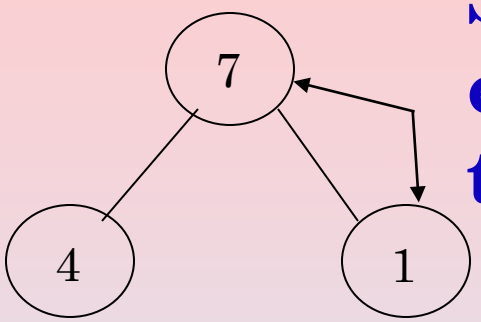
Example of Heap Sort



Unsorted | Sorted:

7	4	1	12	16	19

Example of Heap Sort



Swap the last element with the root

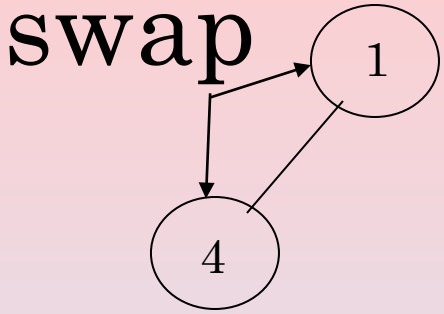
Unsorted | Sorted

1	4	7	12	16	19

Example of Heap Sort

HEAPIFY()

swap

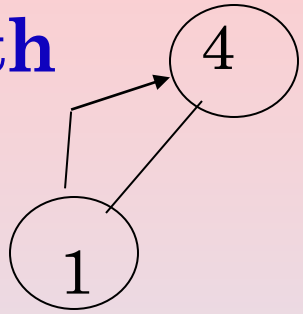


Unsorted | Sorted

4	1	7	12	16	19

Example of Heap Sort

Swap the last element with the root



Unsorted | Sorted

1	4	7	12	16	19

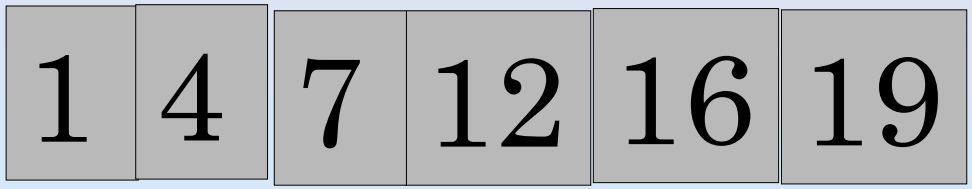
Example of Heap Sort

Place the largest element at right index

1

Unsorted

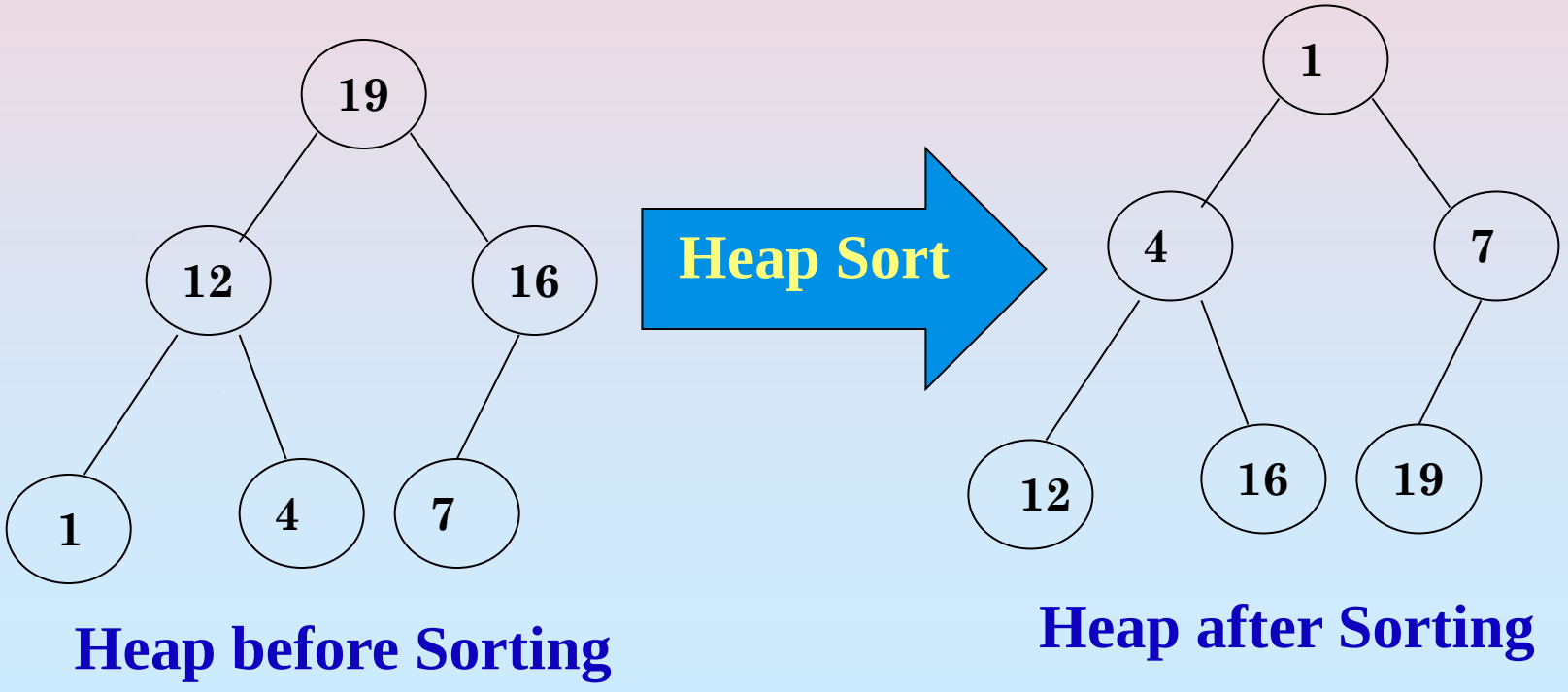
Sorted



Example of Heap Sort

Sorted:

1	4	7	12	16	19



Heap before Sorting

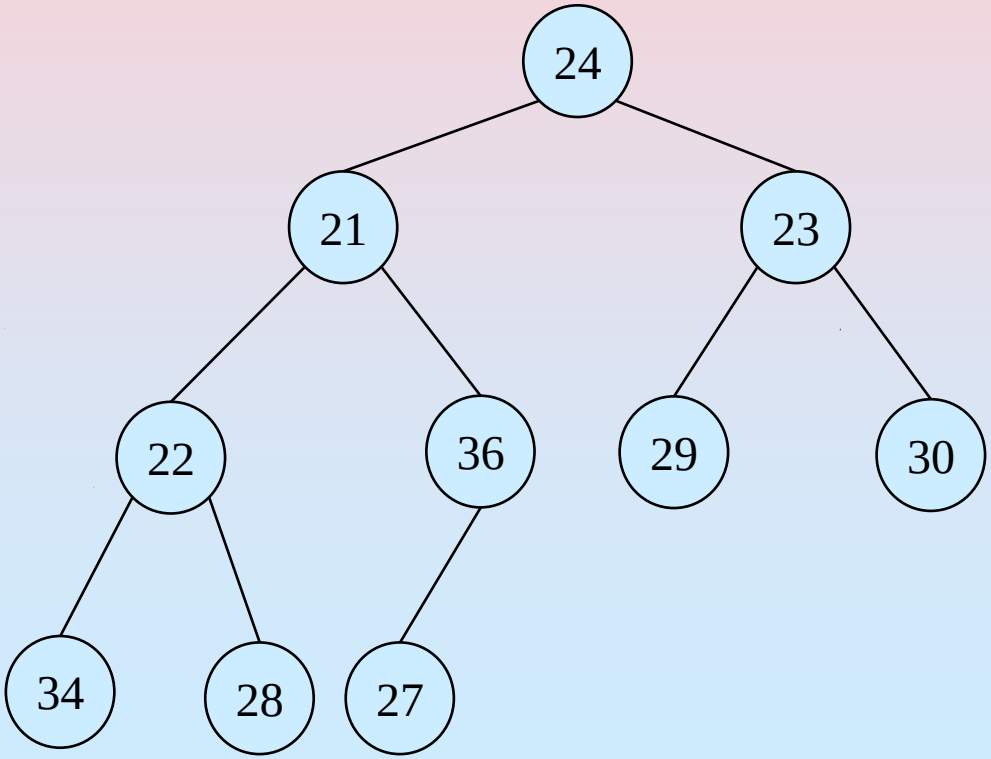
Heap after Sorting

BuildHeap – Example

Input Array:

24	21	23	22	36	29	30	34	28	27

Build the Initial Heap:(<u>not</u> max-heap)



BuildMaxHeap – Example

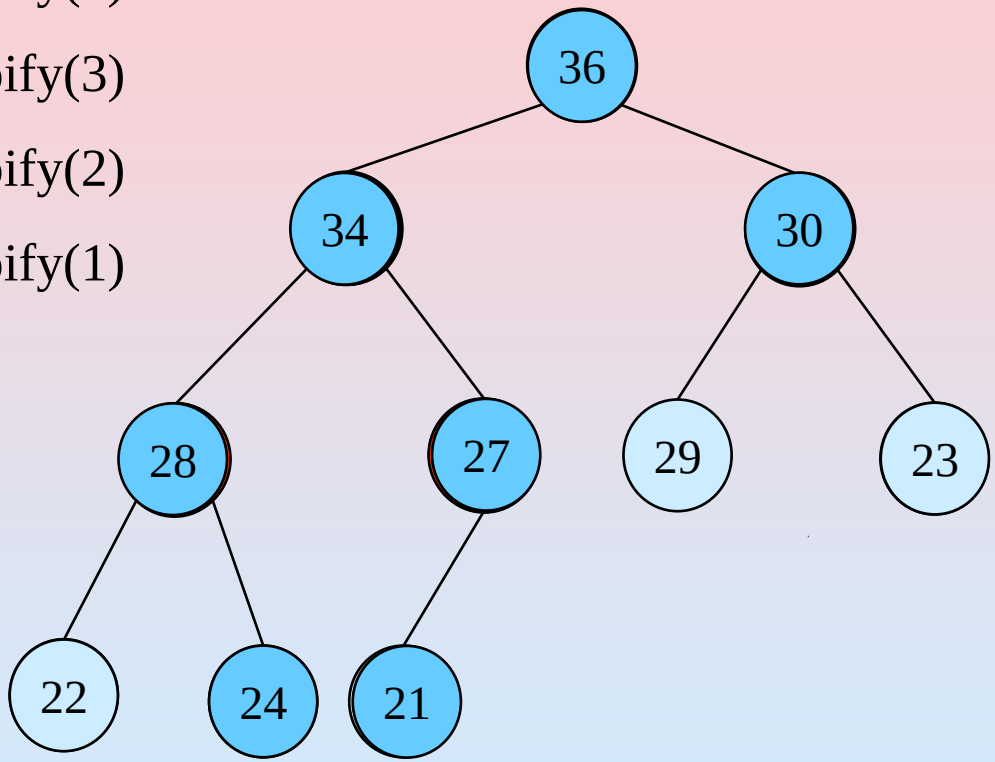
MaxHeapify(⌊10/2⌋ = 5)

MaxHeapify(4)

MaxHeapify(3)

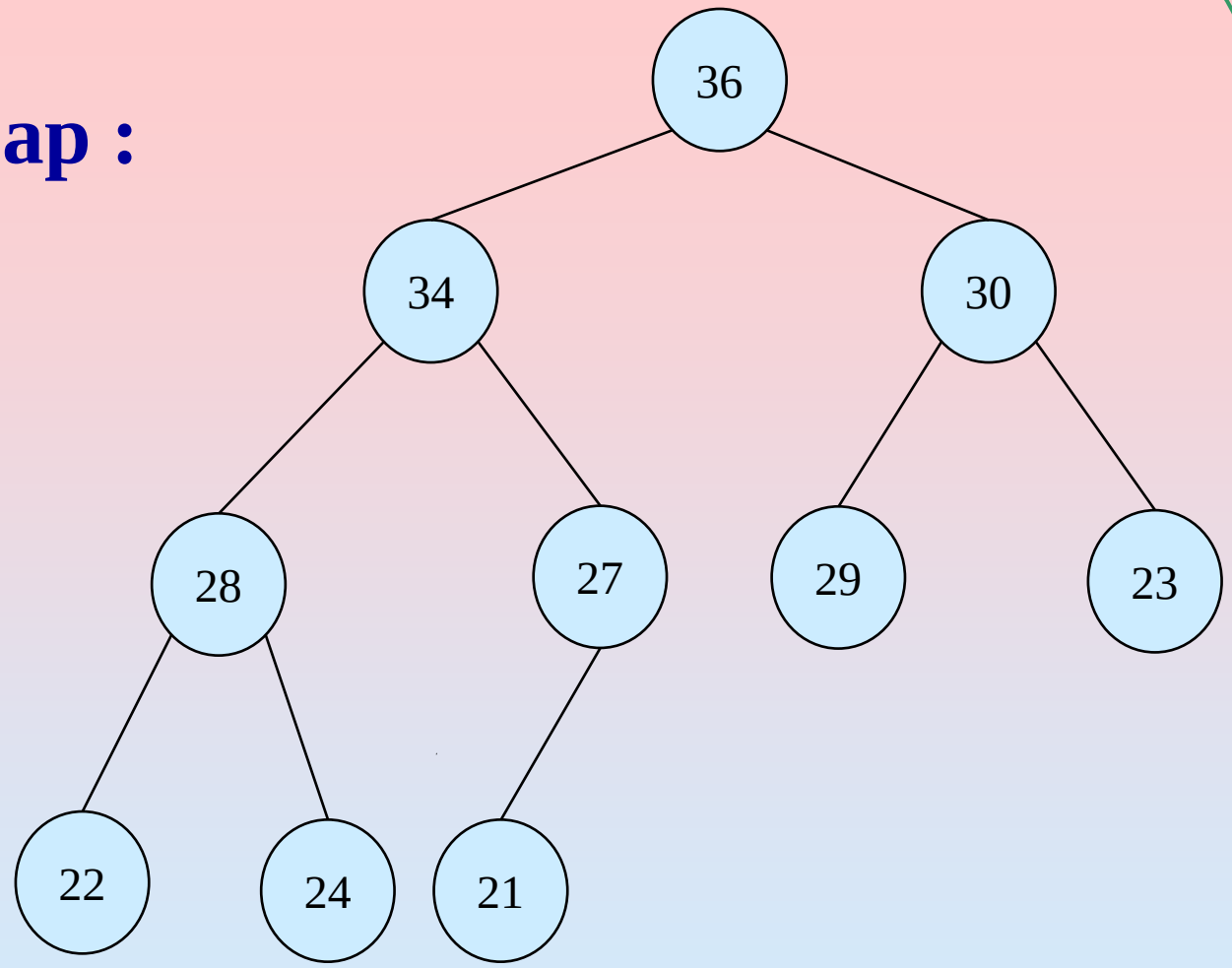
MaxHeapify(2)

MaxHeapify(1)



HeapSort – Example

MaxHeap :

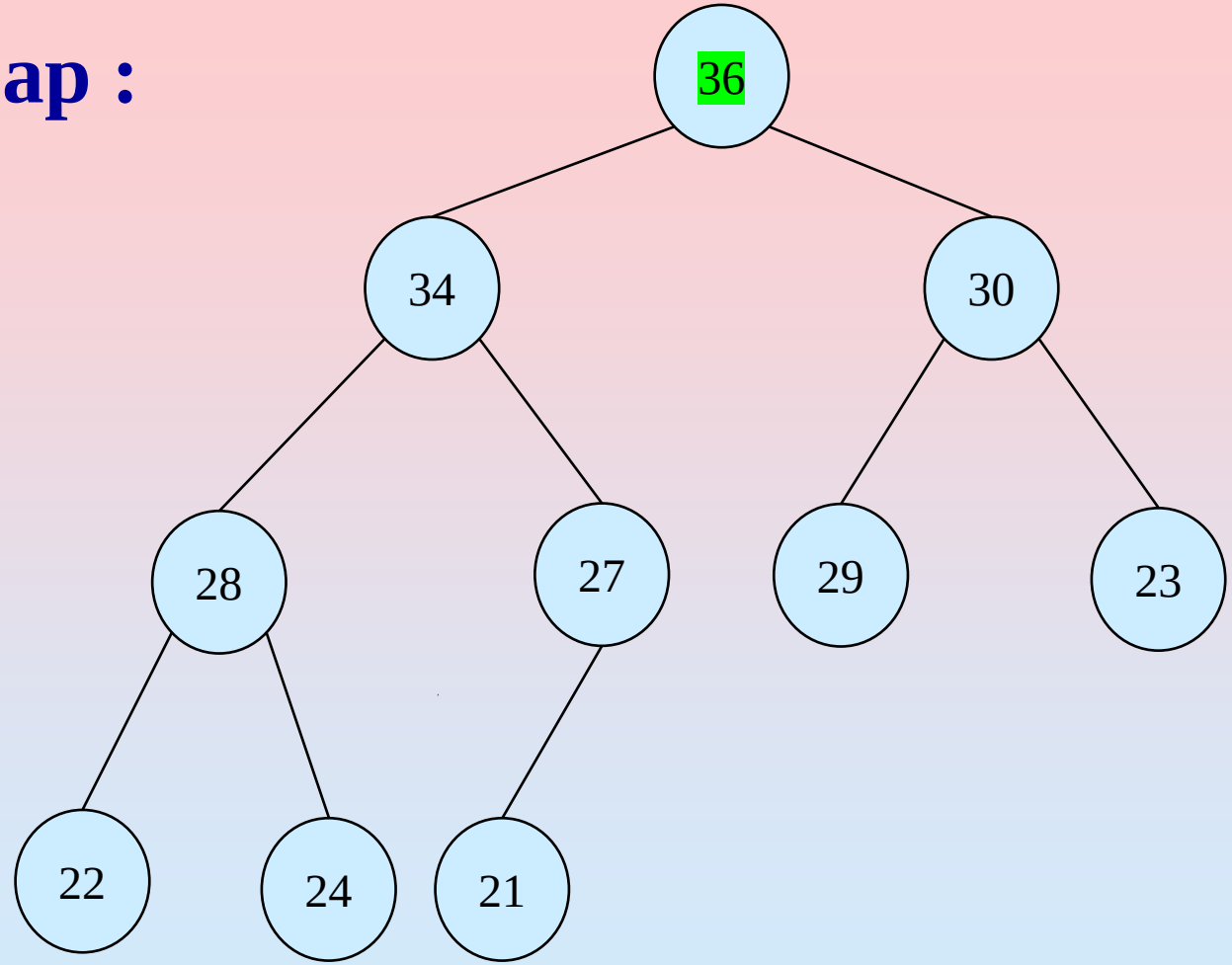


Array representation :

36	34	30	28	27	29	23	22	24	21

Show the contents of the array during HeapSort

MaxHeap :

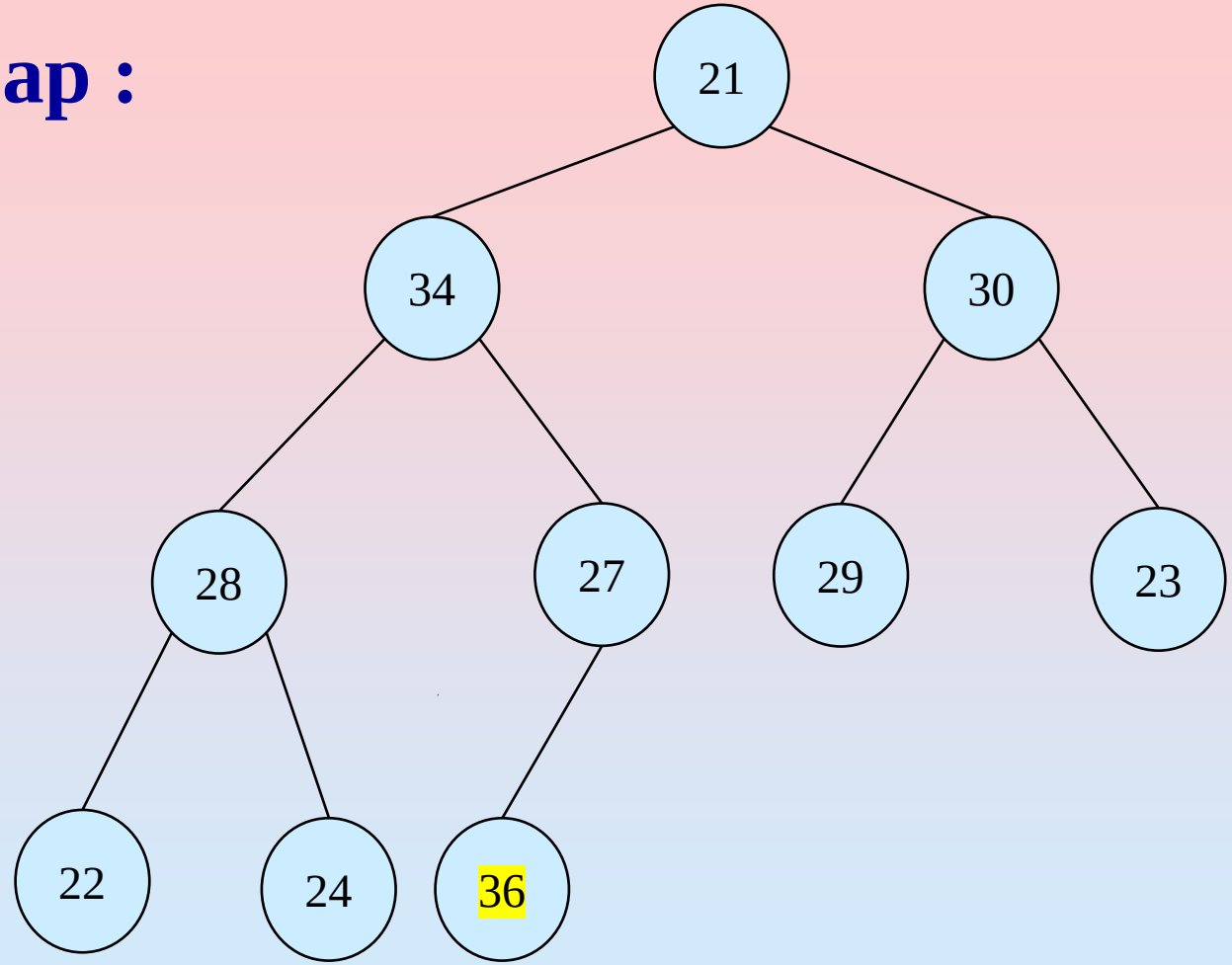


Array representation :

36	34	30	28	27	29	23	22	24	21

Show the contents of the array during HeapSort

MaxHeap :

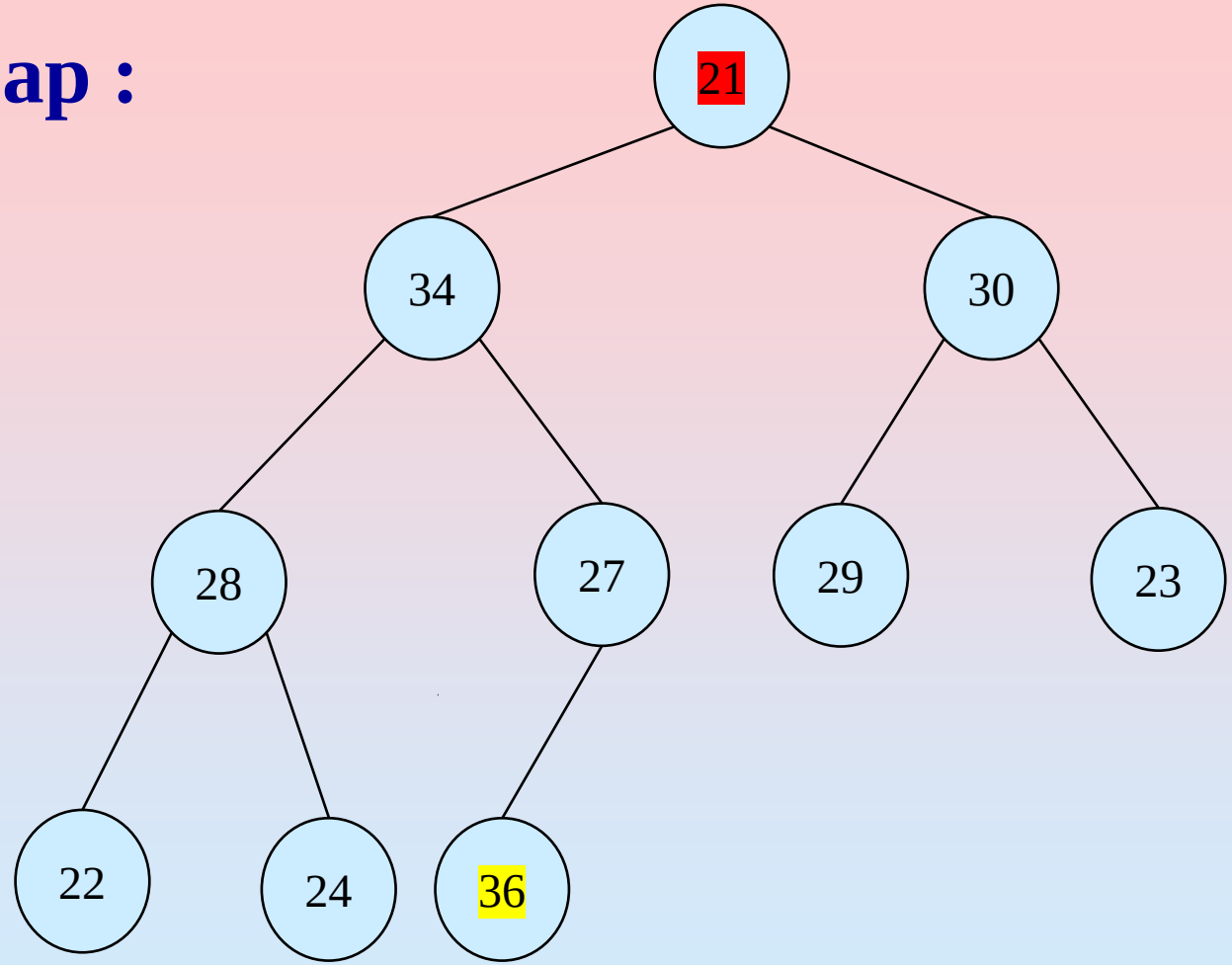


Array representation :

21	34	30	28	27	29	23	22	24	36

Show the contents of the array during HeapSort

MaxHeap :

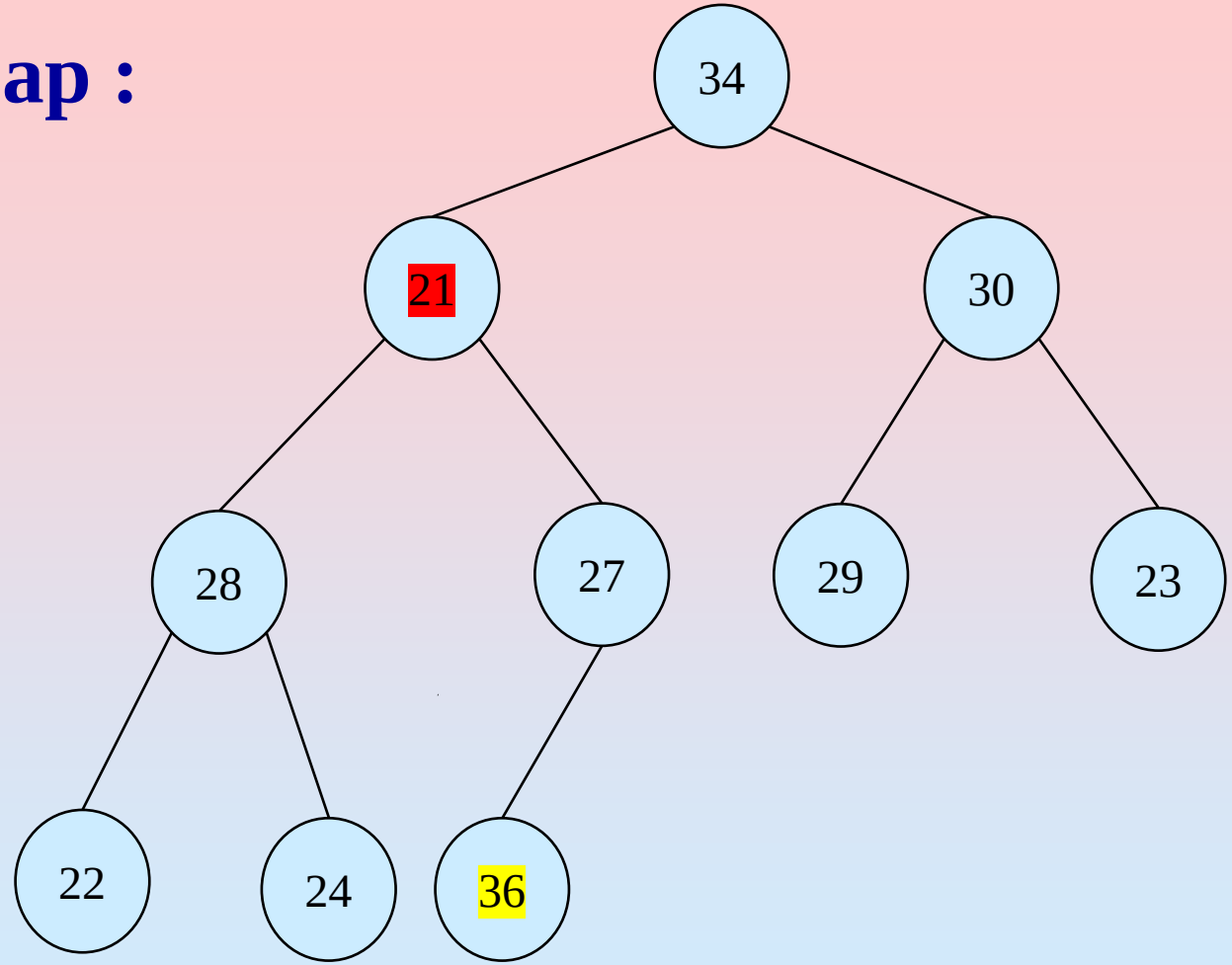


Array representation :

21	34	30	28	27	29	23	22	24	36

Show the contents of the array during HeapSort

MaxHeap :

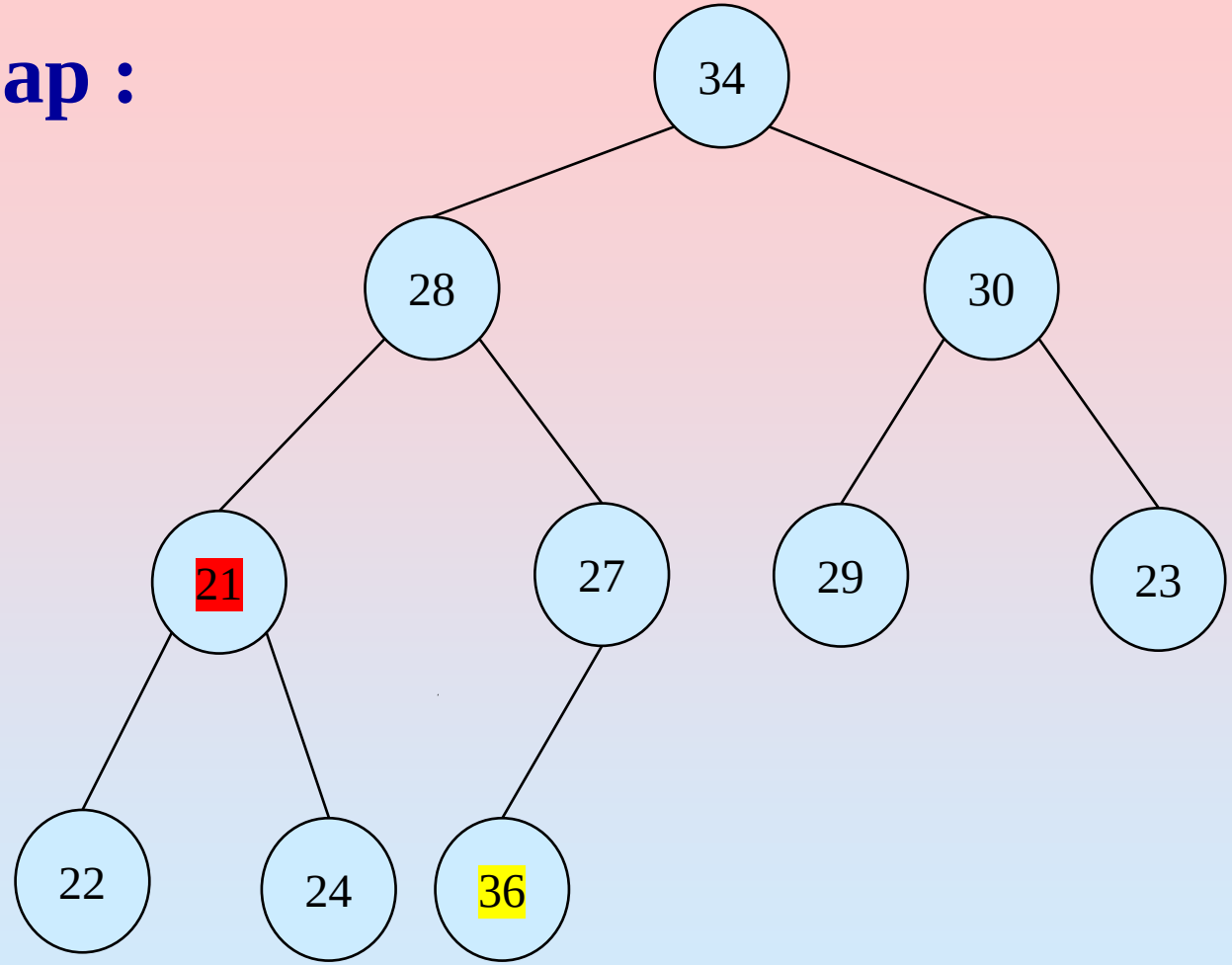


Array representation :

34	21	30	28	27	29	23	22	24	36

Show the contents of the array during HeapSort

MaxHeap :

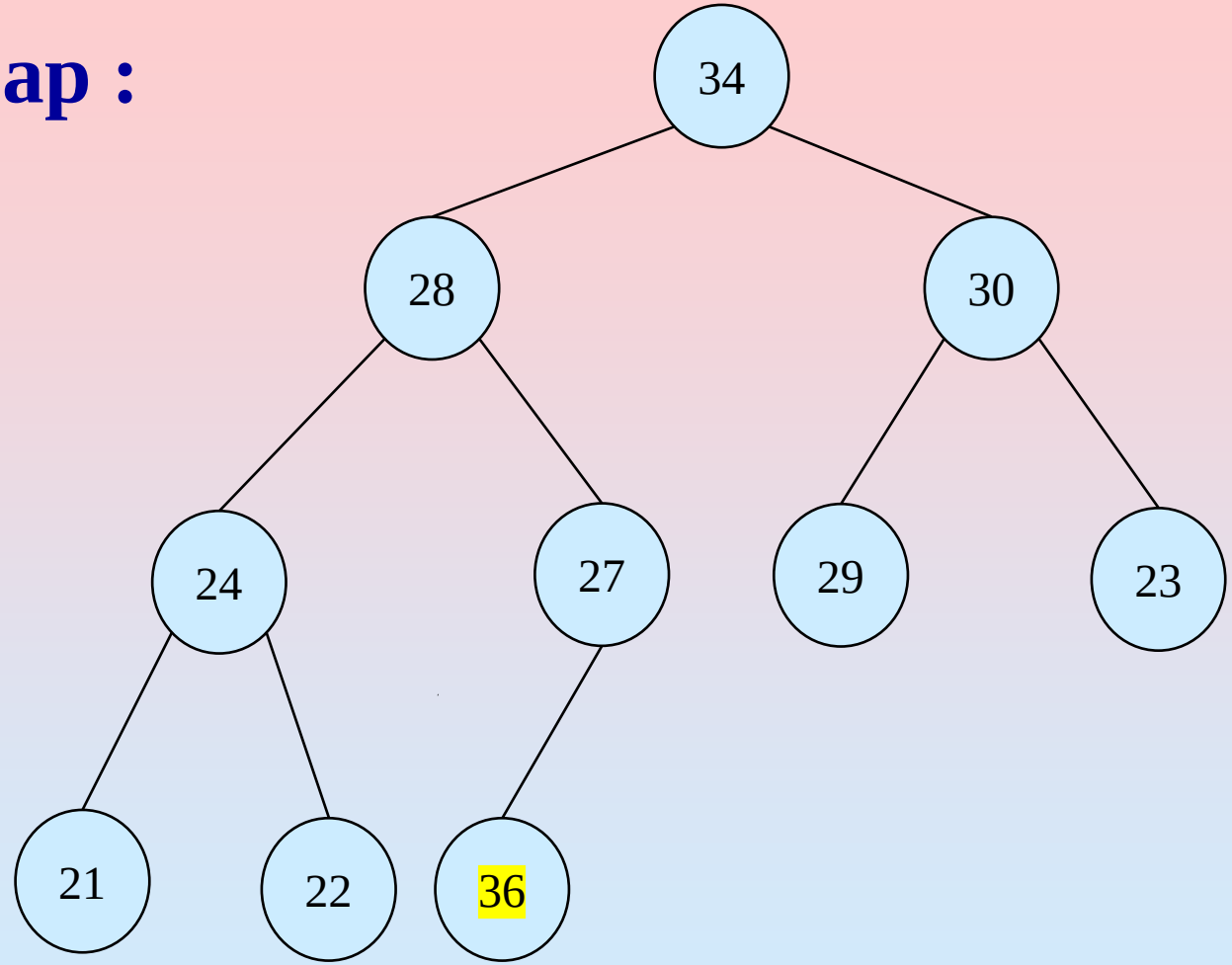


Array representation :

34	28	30	21	27	29	23	22	24	36

Show the contents of the array during HeapSort

MaxHeap :

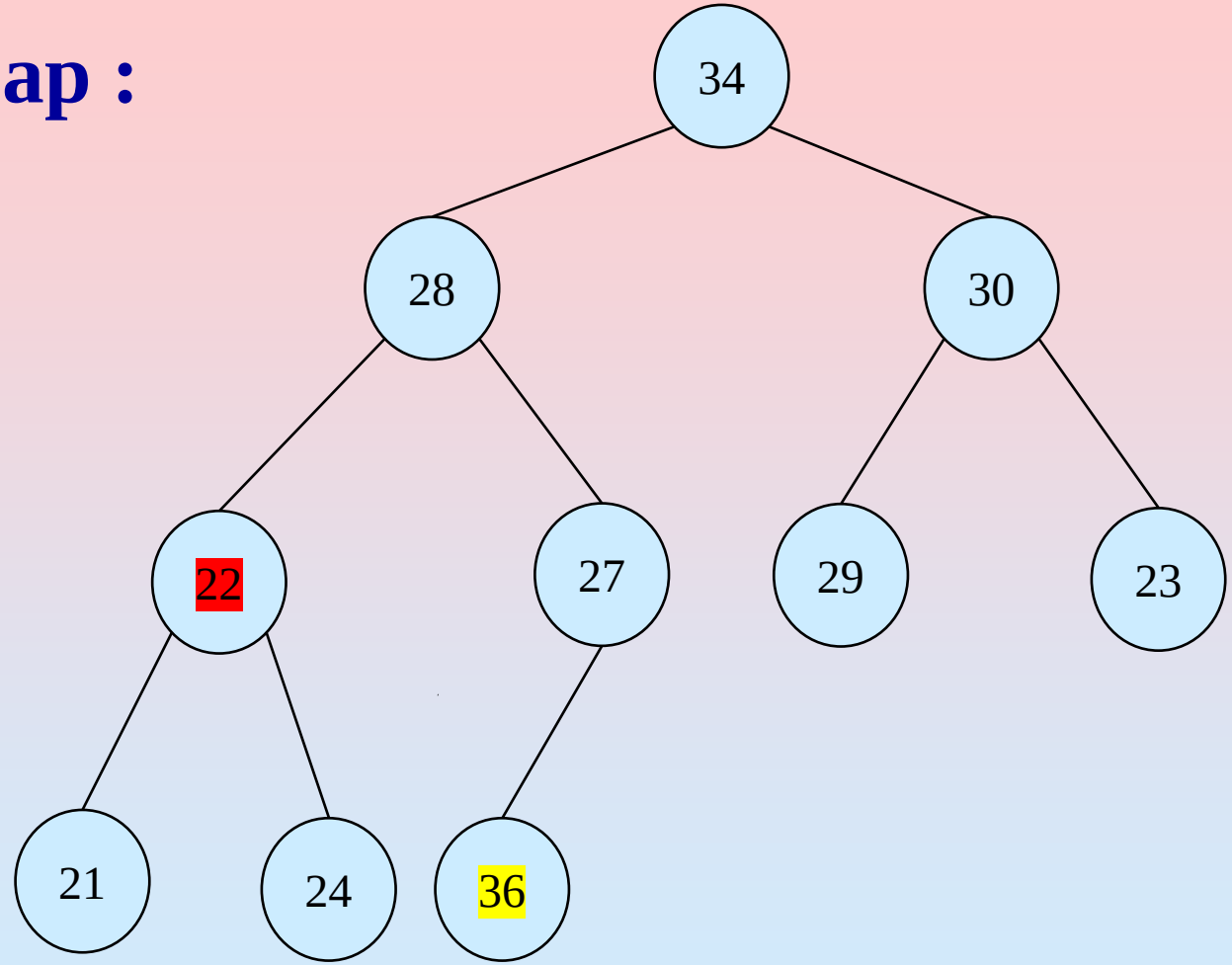


Array representation :

34	28	30	22	27	29	23	21	24	36

Show the contents of the array during HeapSort

MaxHeap :

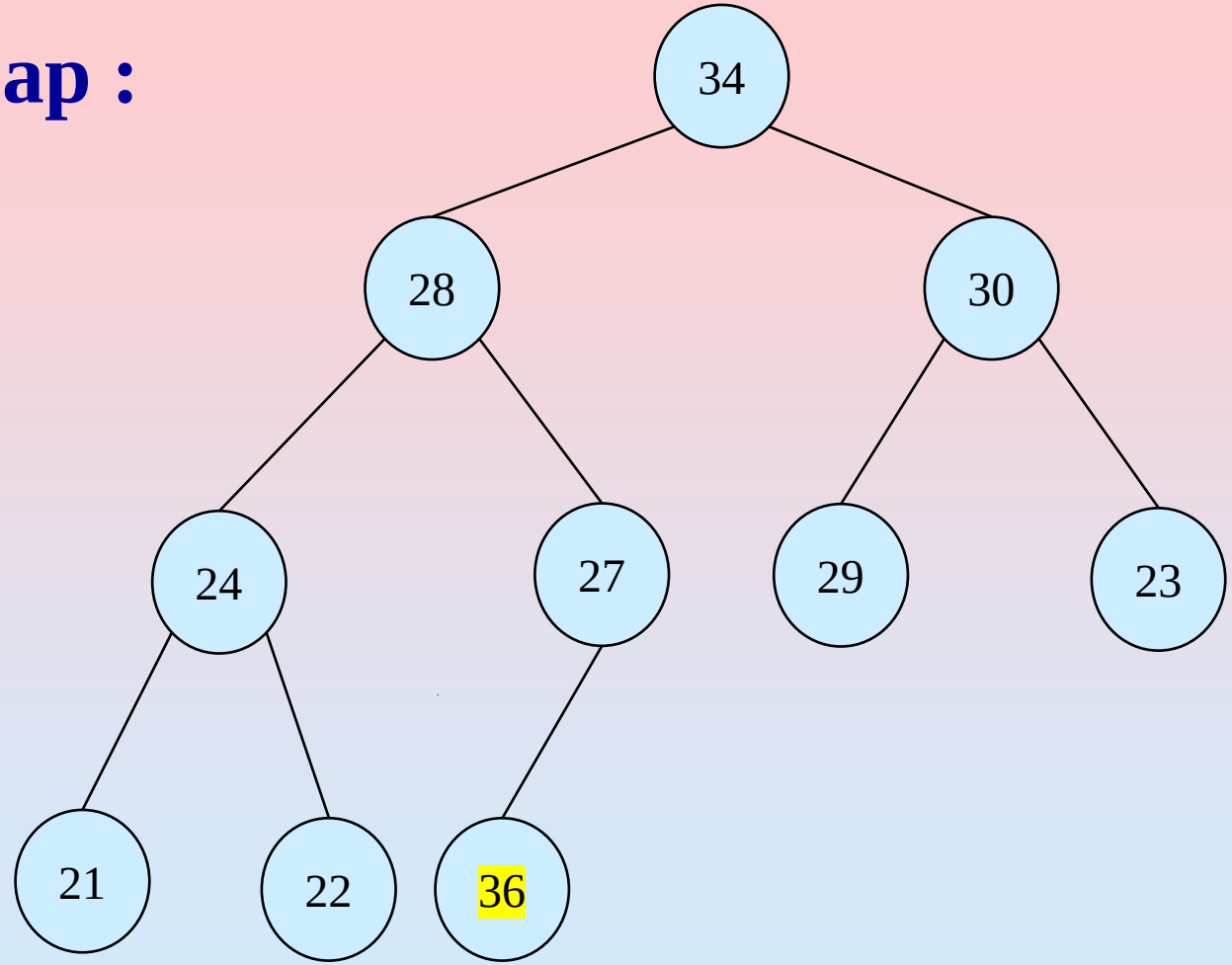


Array representation :

34	28	30	22	27	29	23	21	24	36

Show the contents of the array during HeapSort

MaxHeap :

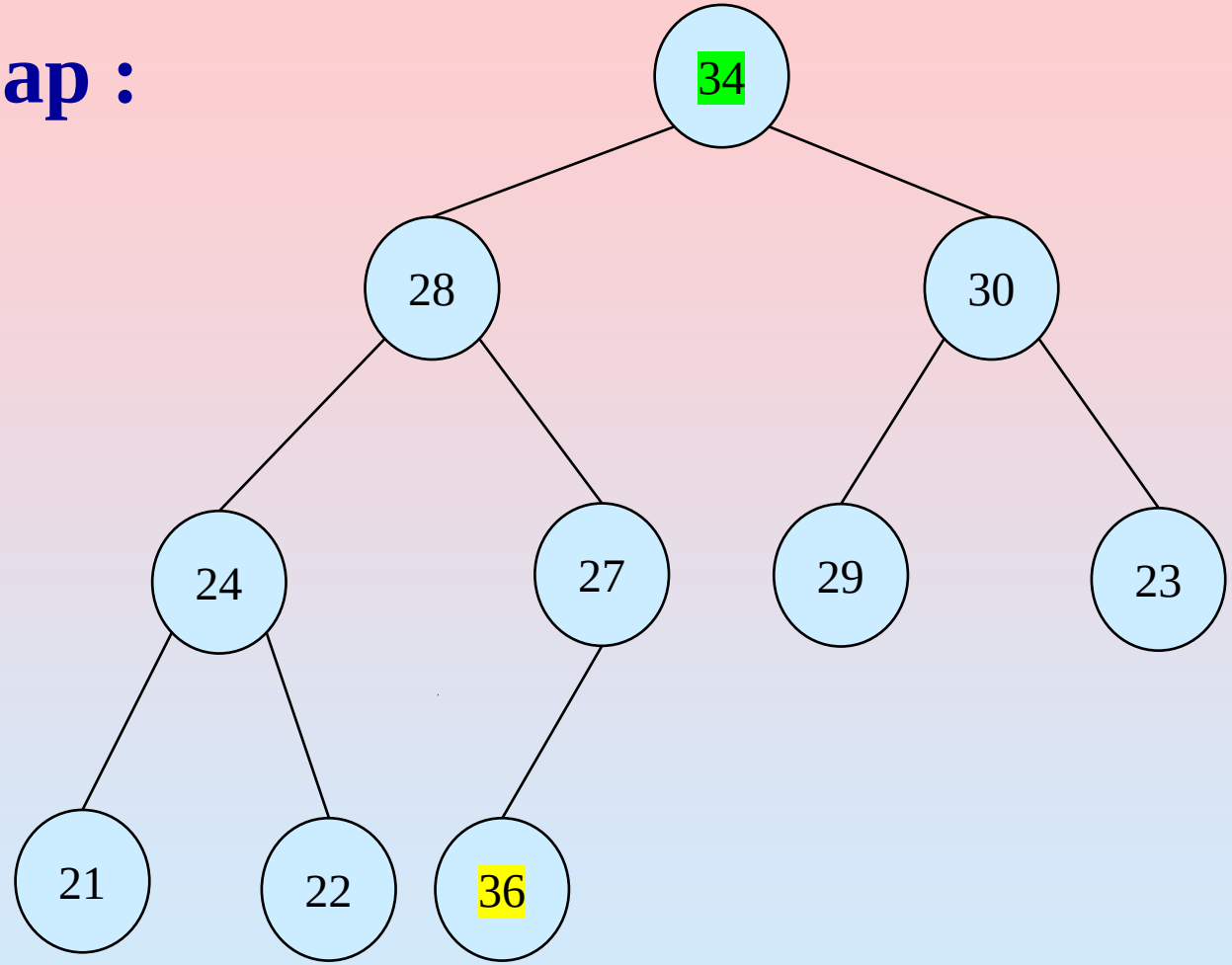


Array representation :

34	28	30	22	27	29	23	21	24	36

Show the contents of the array during HeapSort

MaxHeap :

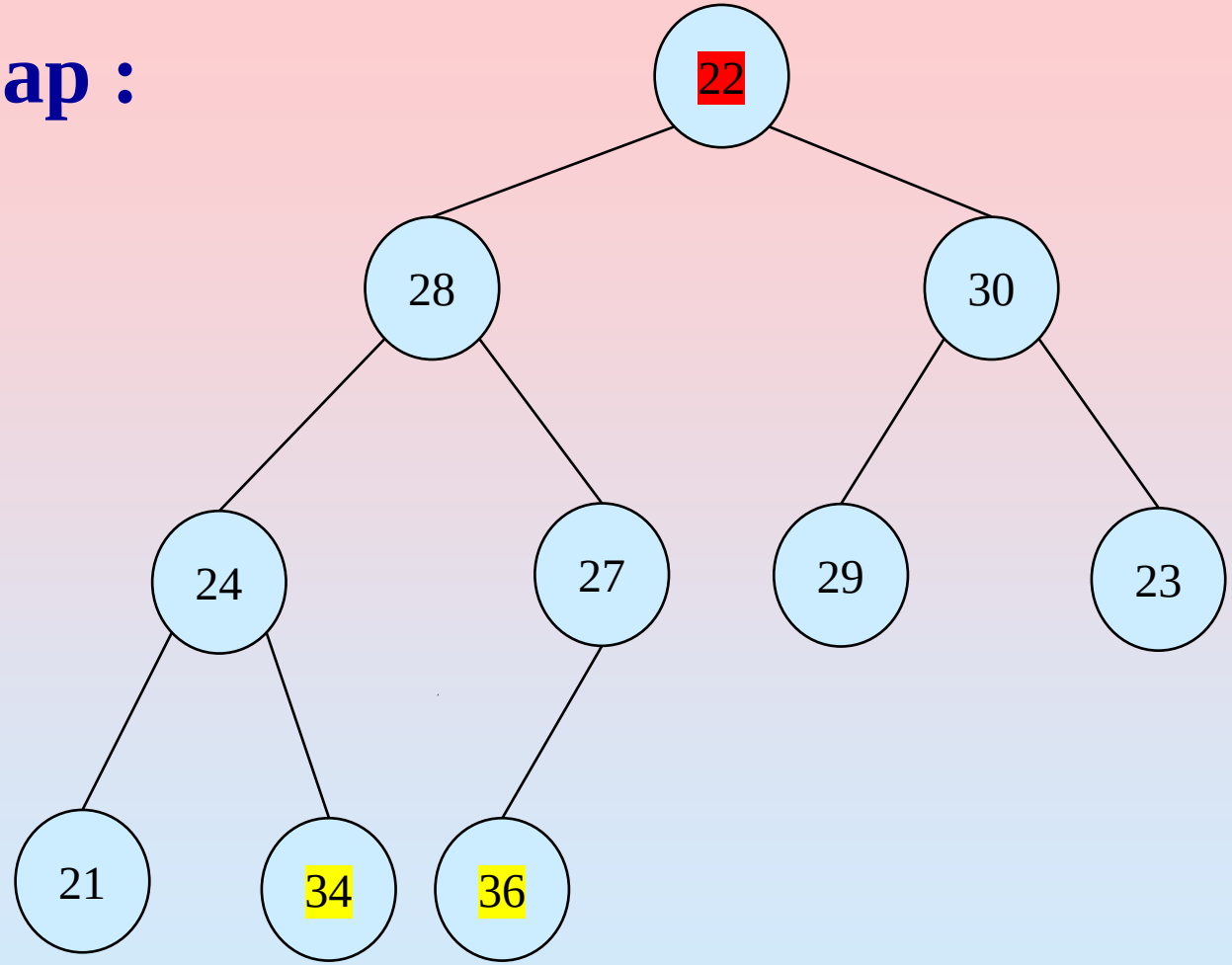


Array representation :

34	28	30	24	27	29	23	21	22	36

Show the contents of the array during HeapSort

MaxHeap :

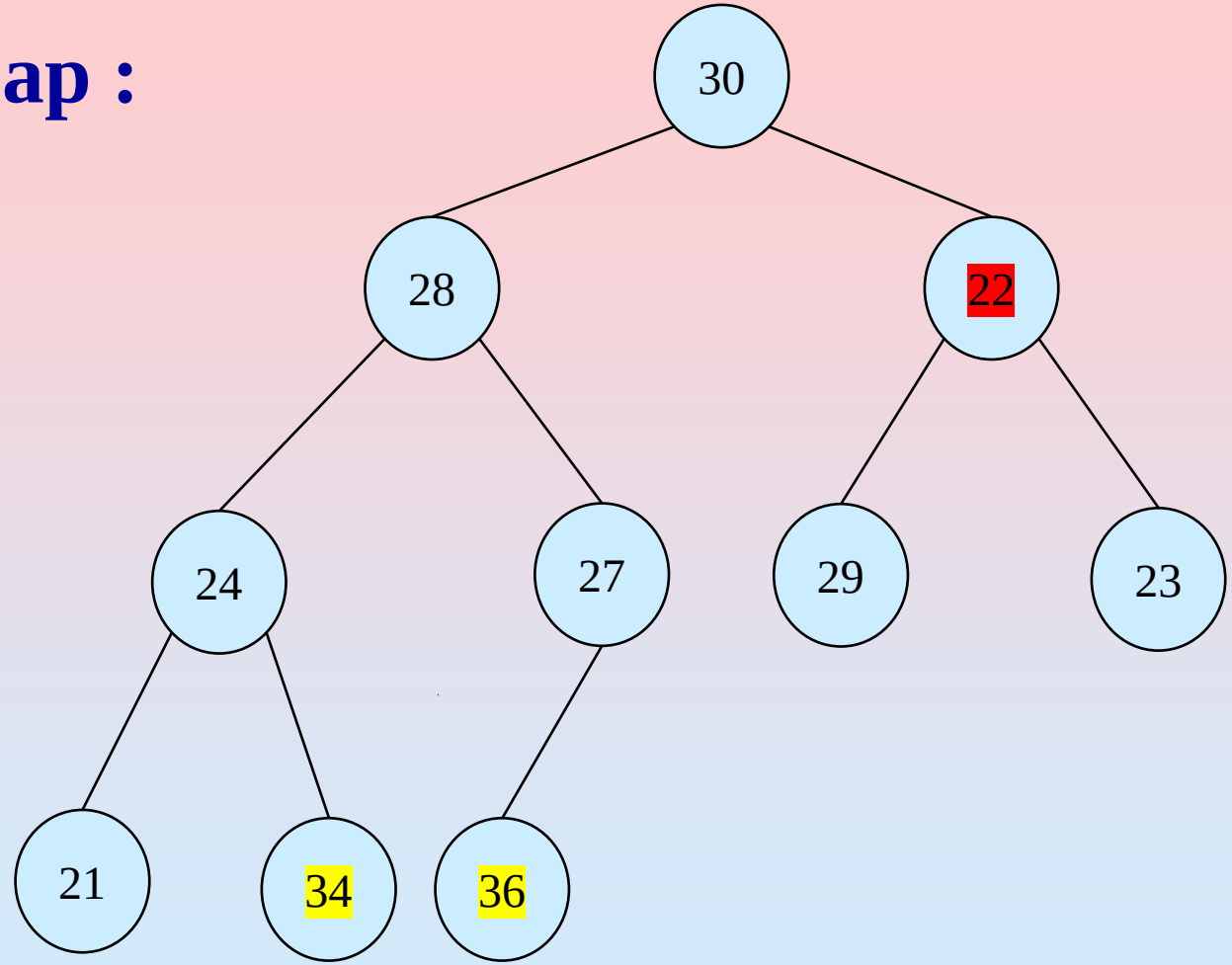


Array representation :

22	28	30	22	27	29	23	21	34	36

Show the contents of the array during HeapSort

MaxHeap :

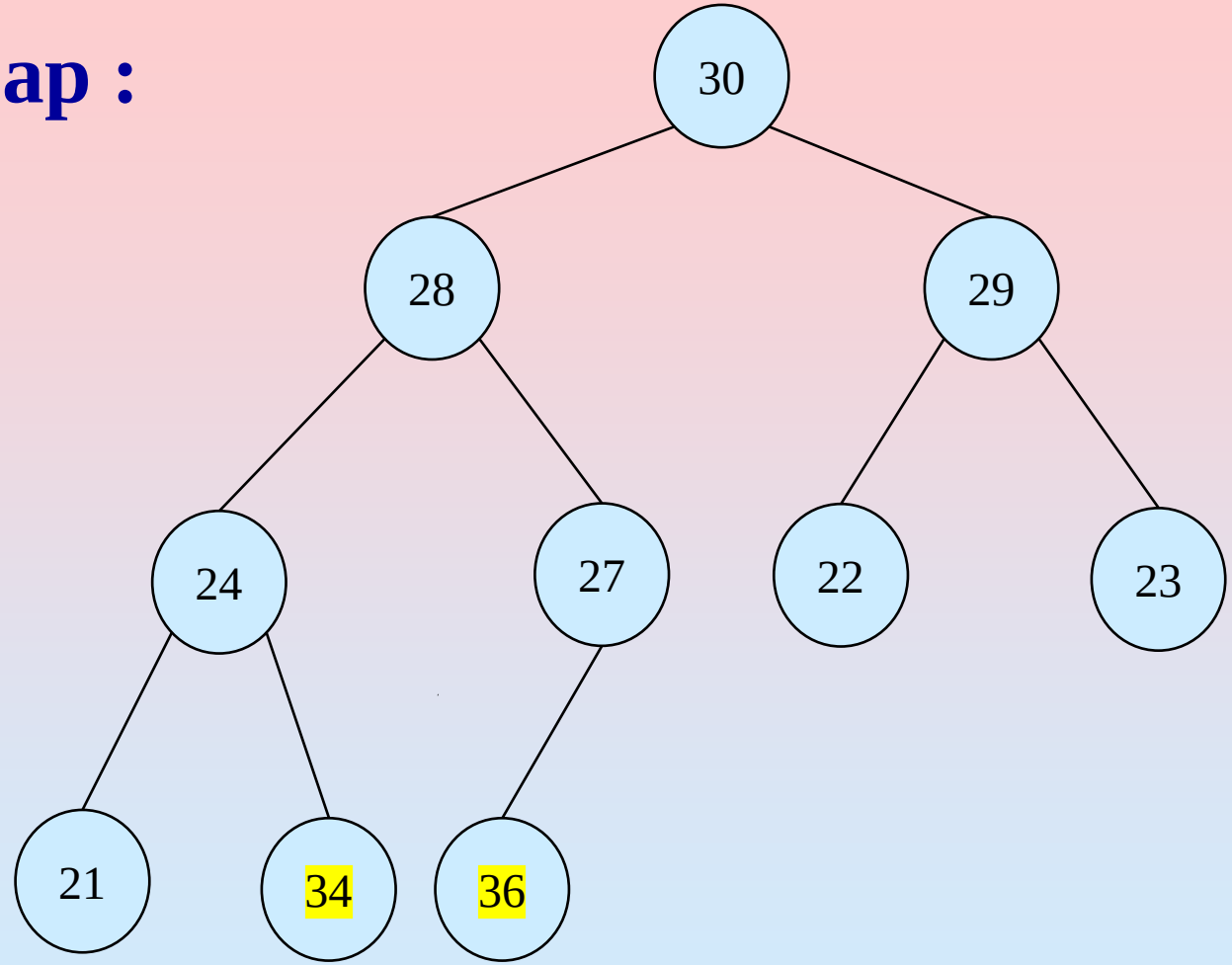


Array representation :

30	28	22	24	27	29	23	21	34	36

Show the contents of the array during HeapSort

MaxHeap :

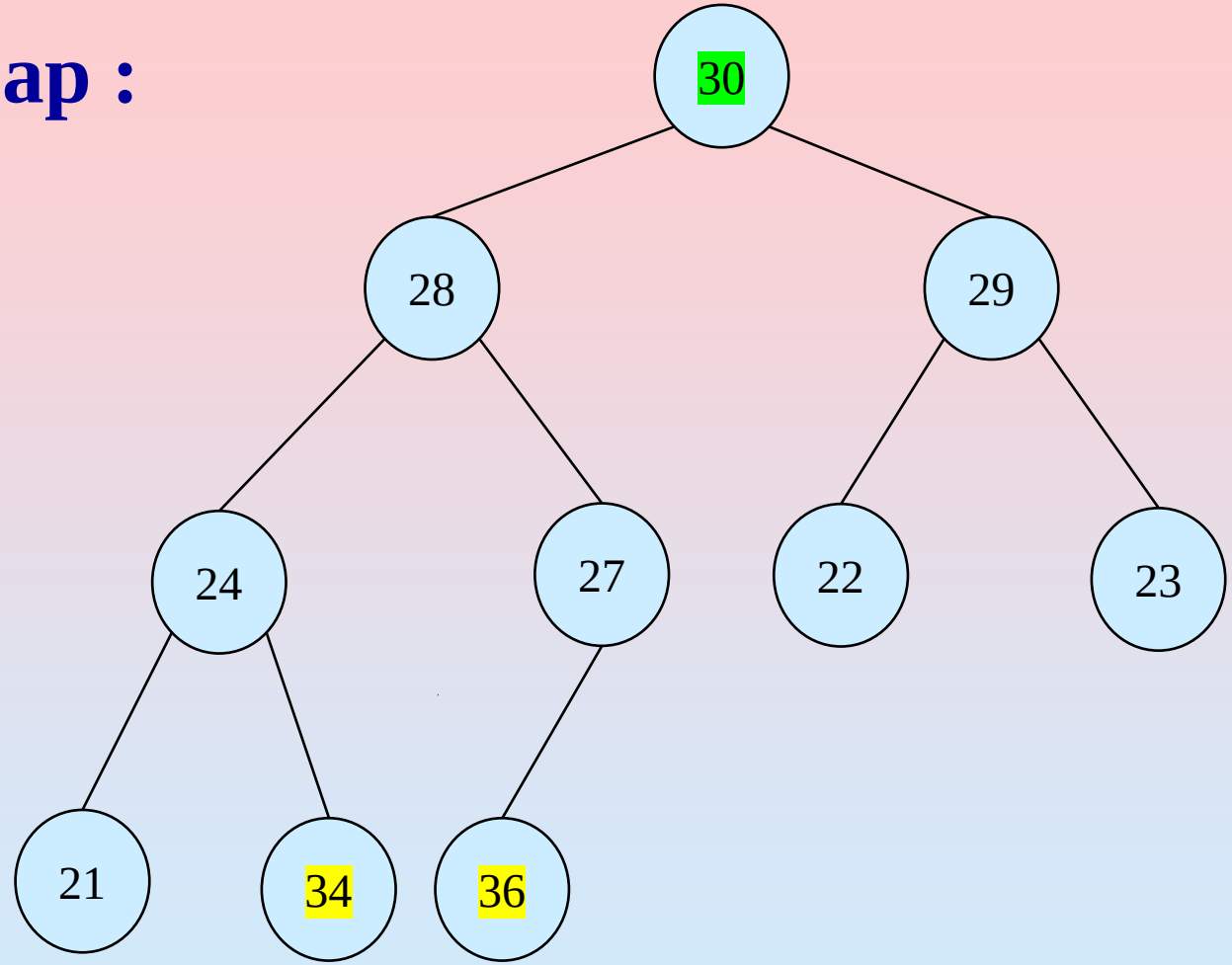


Array representation :

30	28	29	24	27	22	23	21	34	36

Show the contents of the array during HeapSort

MaxHeap :

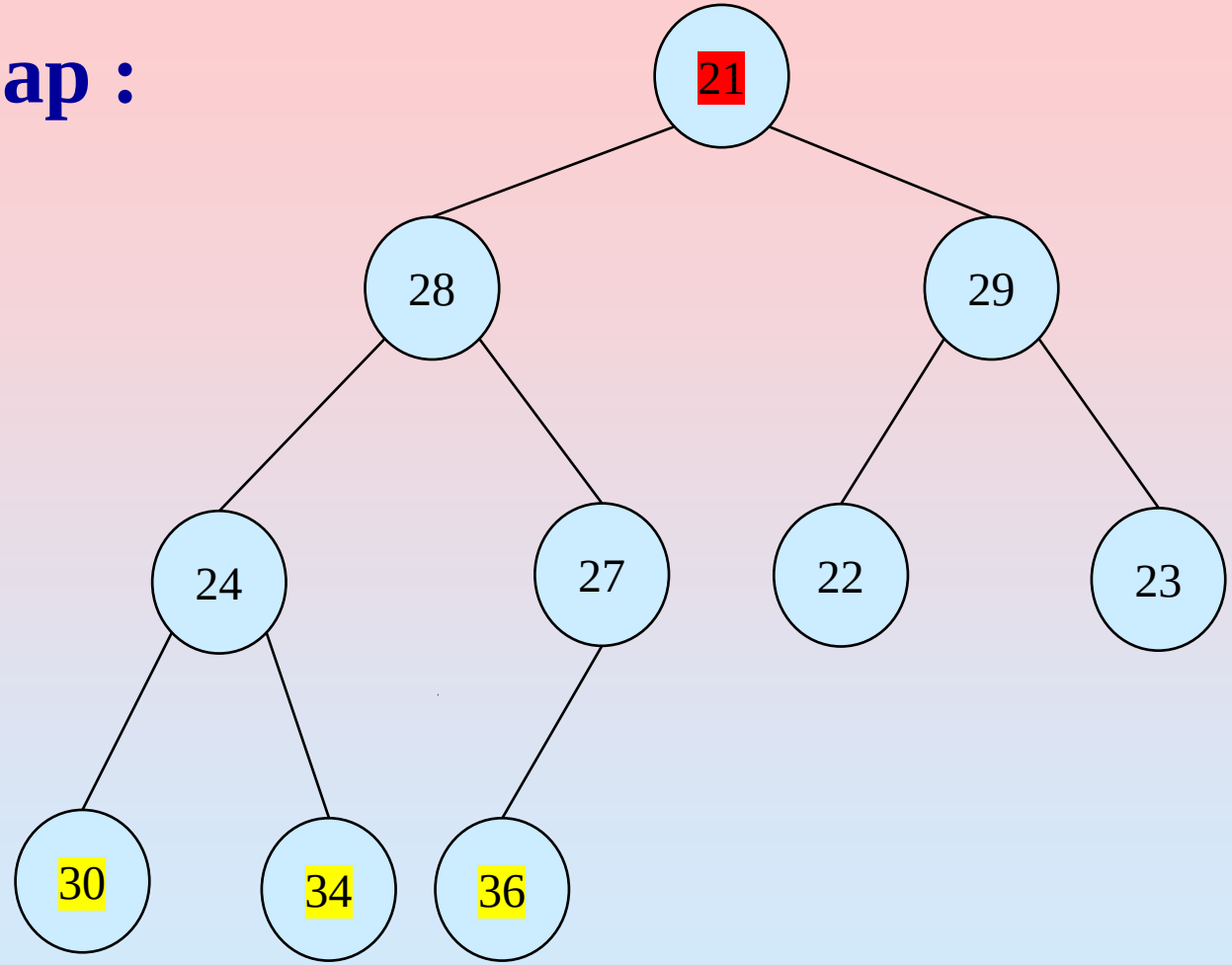


Array representation :

30	28	29	24	27	22	23	21	34	36

Show the contents of the array during HeapSort

MaxHeap :

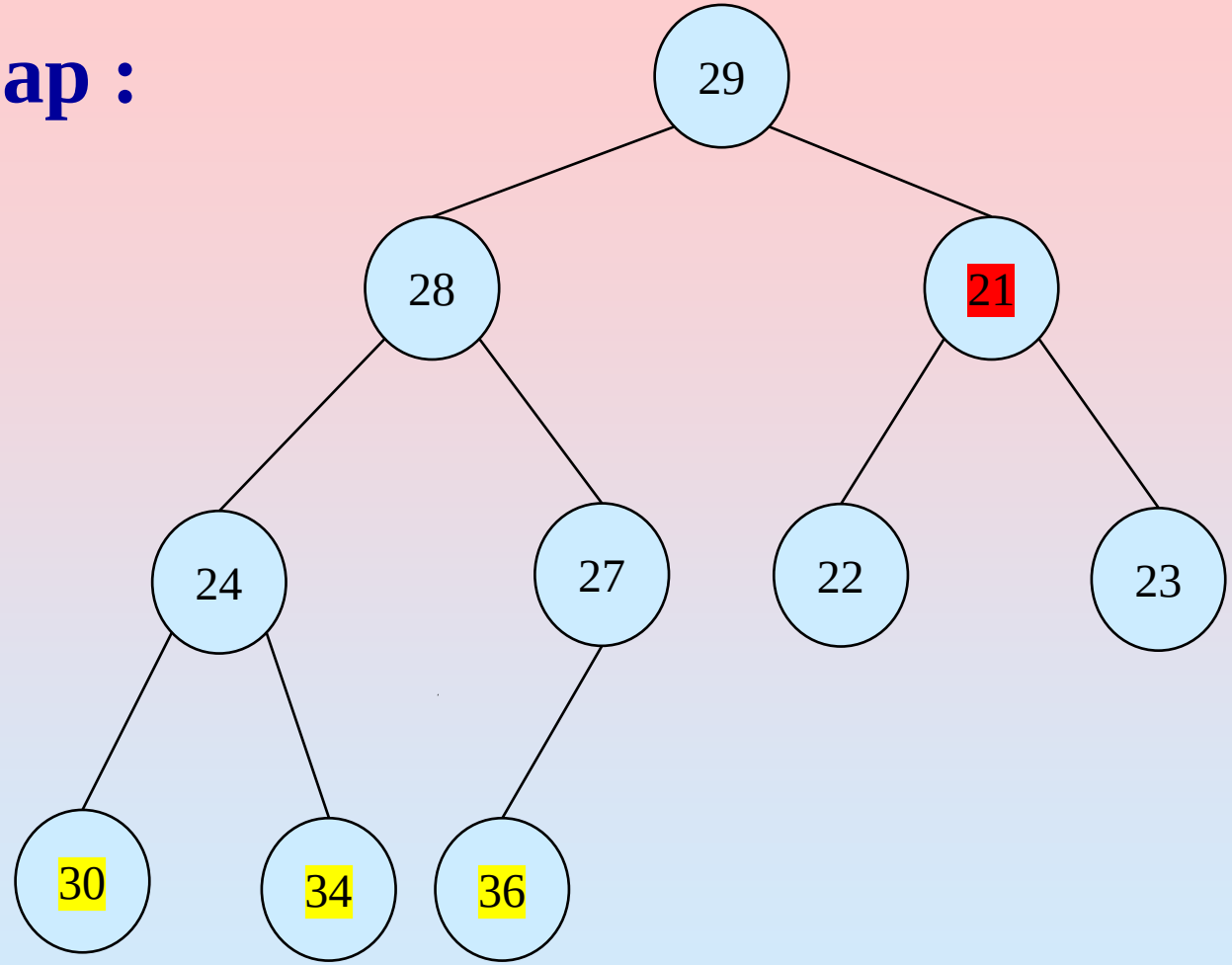


Array representation :

21	28	29	24	27	22	23	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

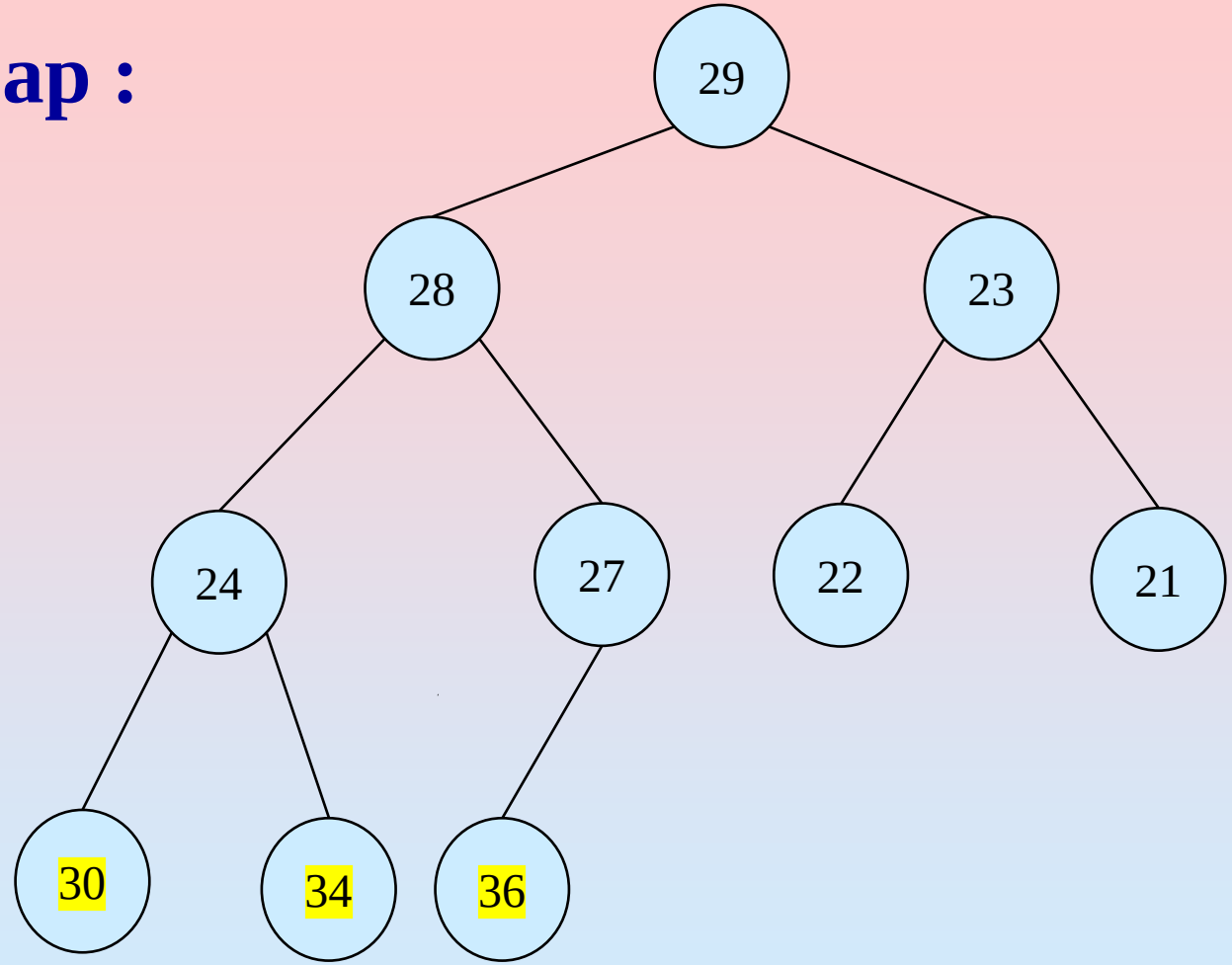


Array representation :

29	28	21	24	27	22	23	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

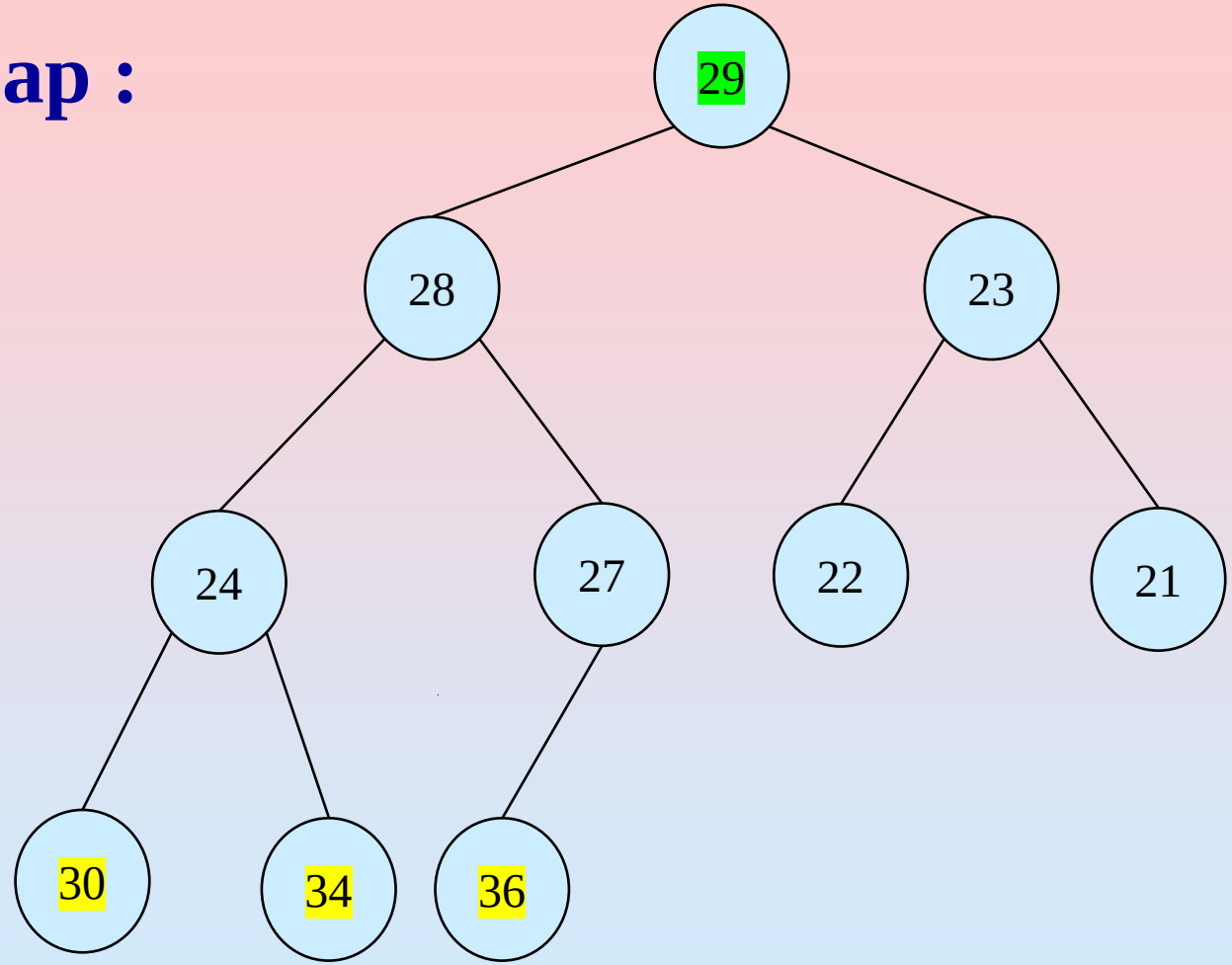


Array representation :

29	28	23	24	27	22	21	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

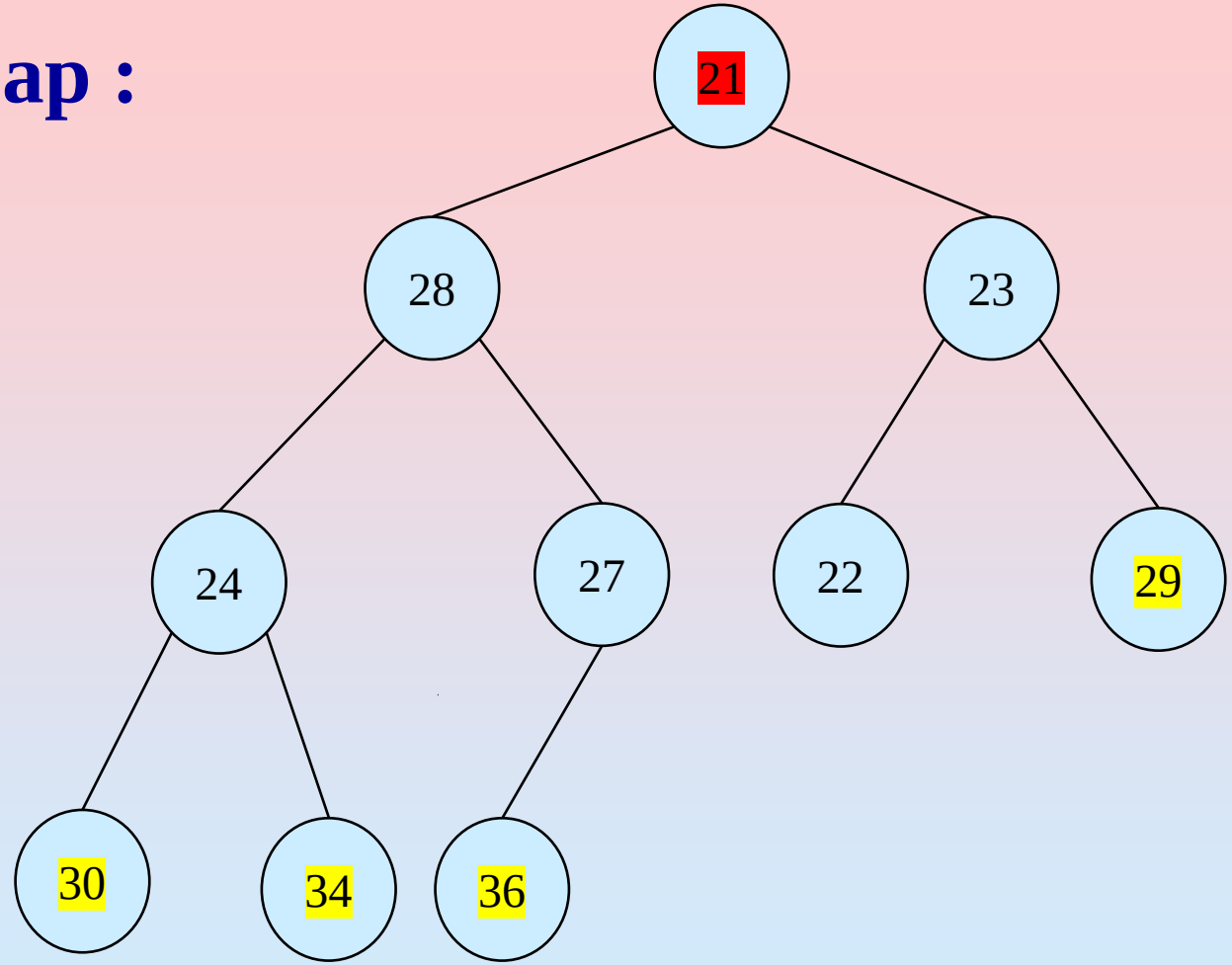


Array representation :

29	28	23	24	27	22	21	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

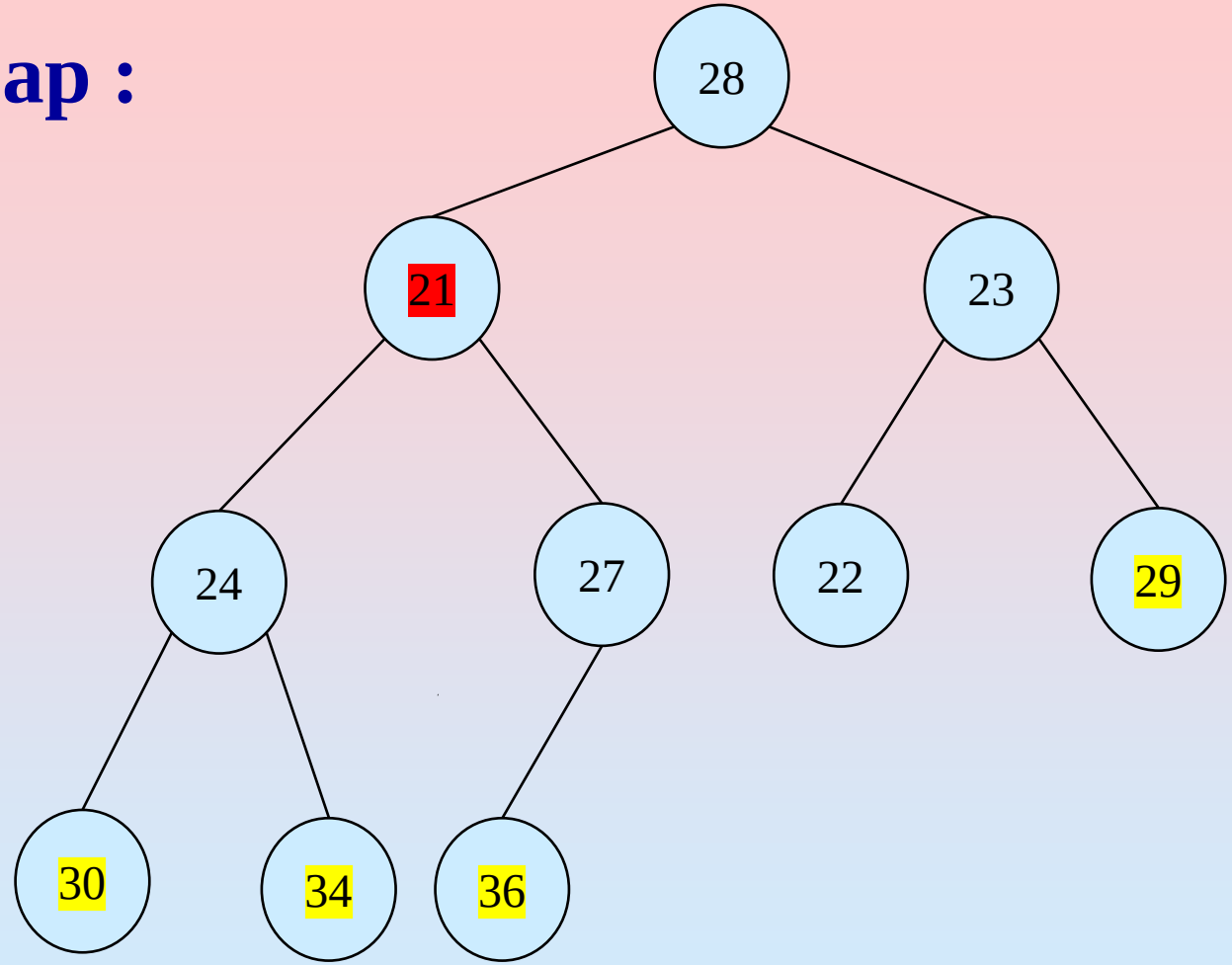


Array representation :

21	28	23	24	27	22	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

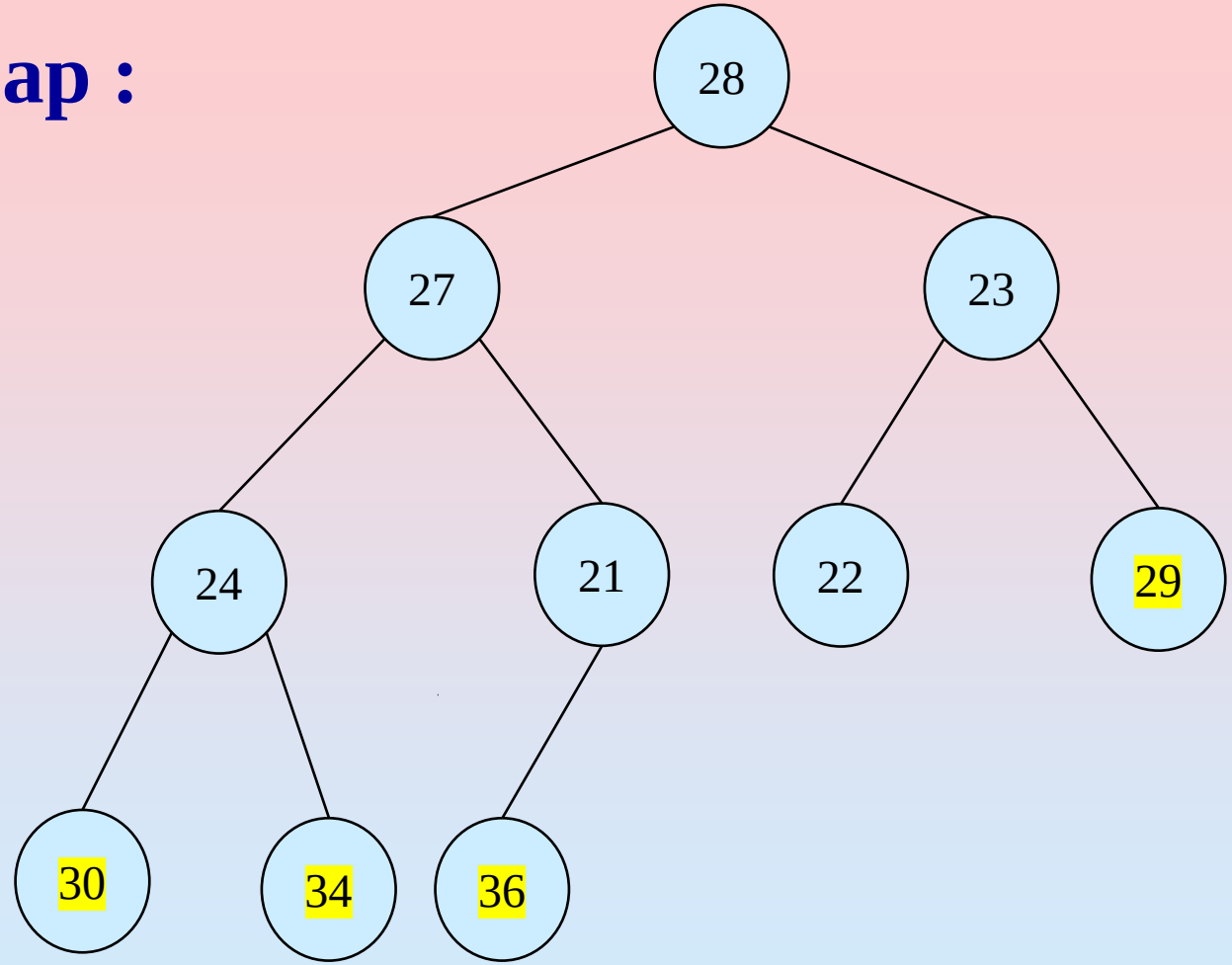


Array representation :

28	27	23	24	21	22	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

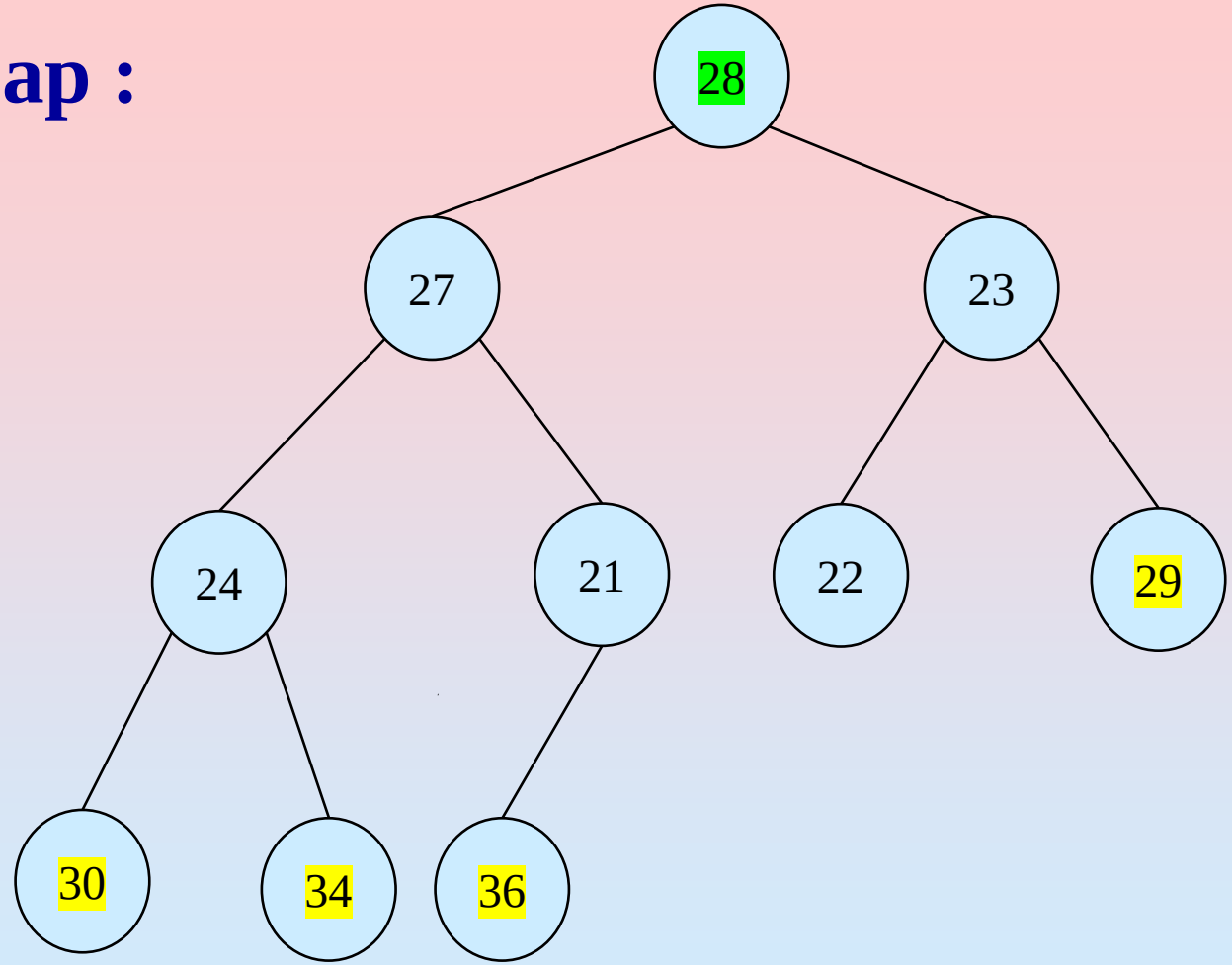


Array representation :

28	27	23	24	21	22	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

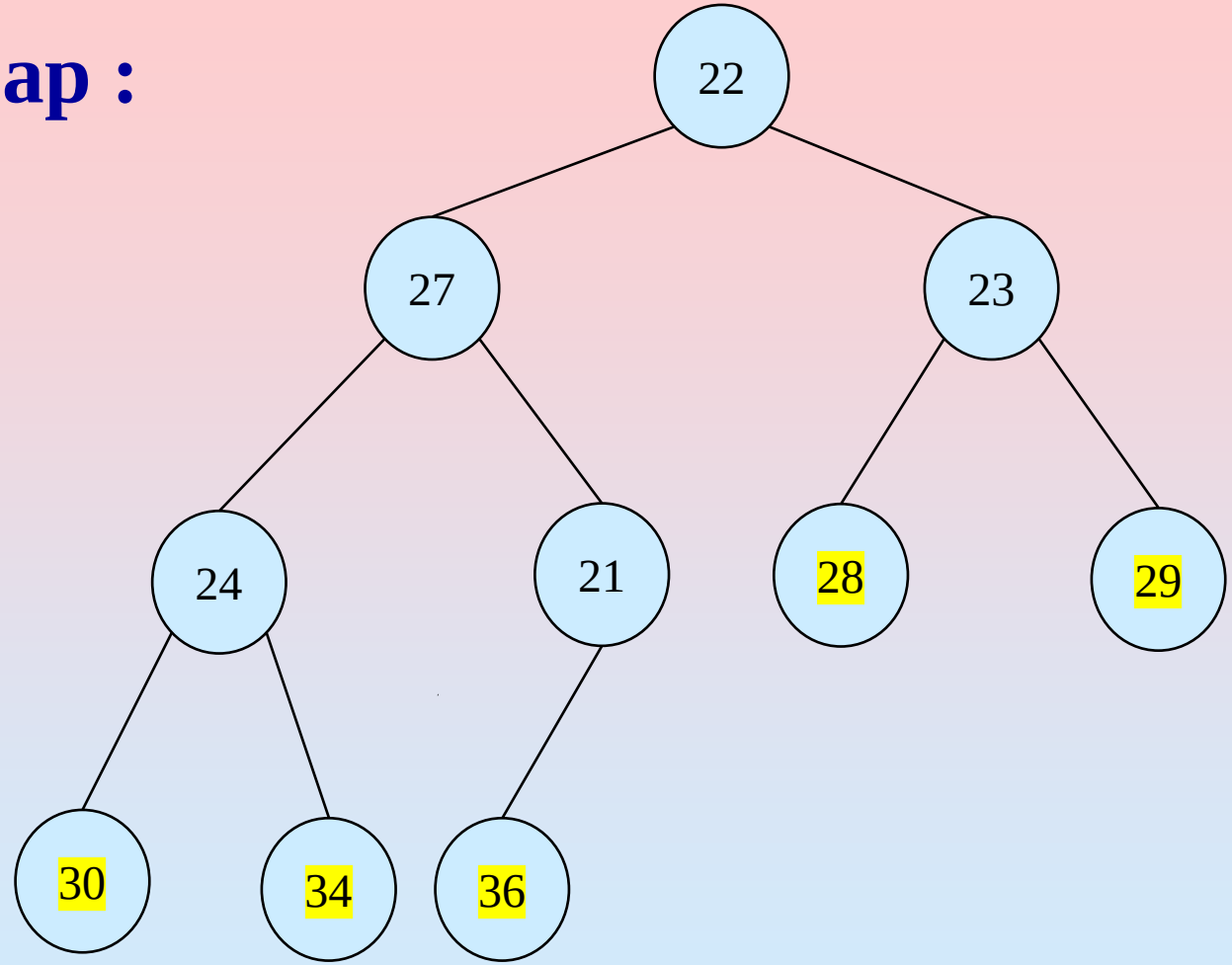


Array representation :

28	27	23	24	21	22	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

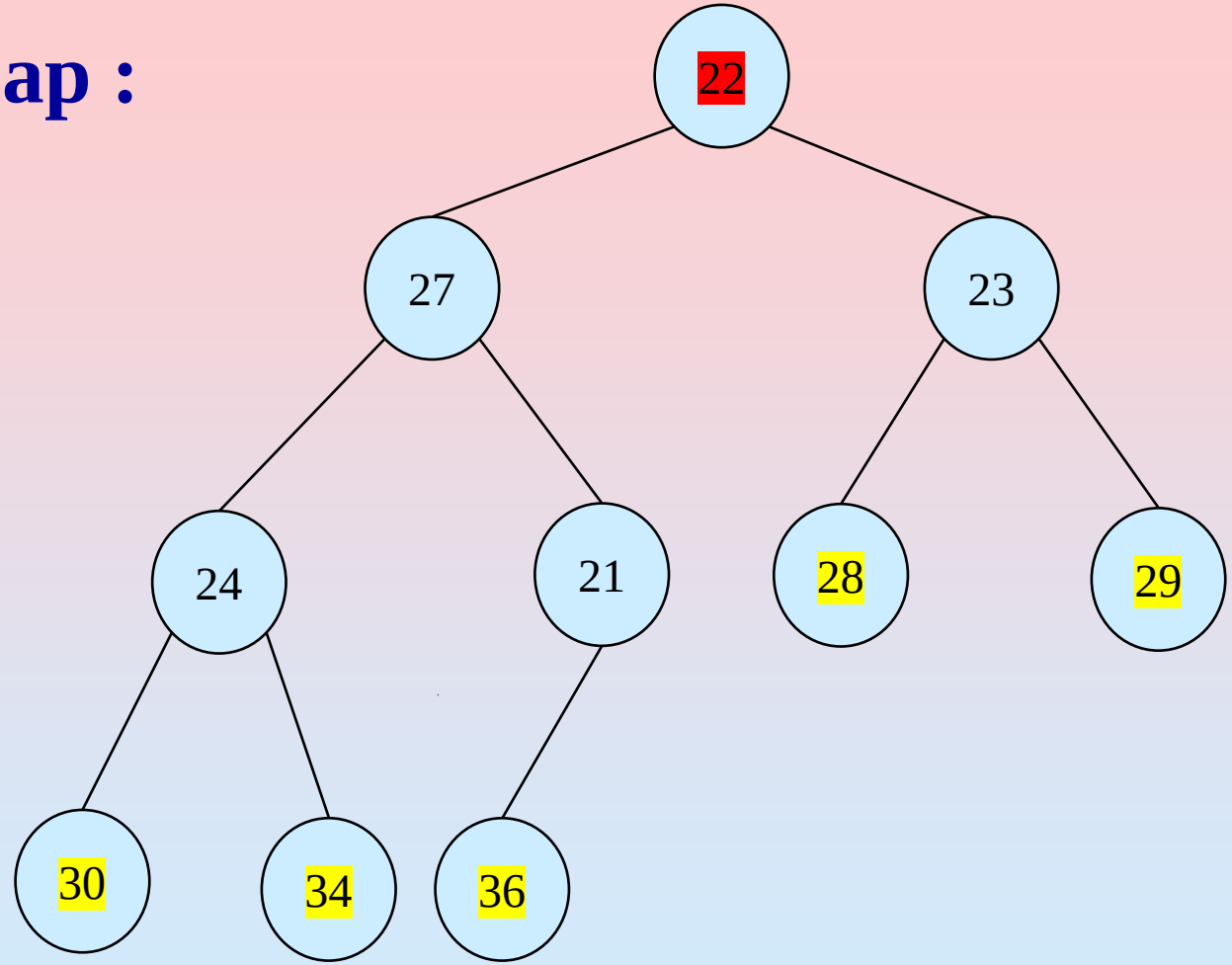


Array representation :

22	27	23	24	21	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

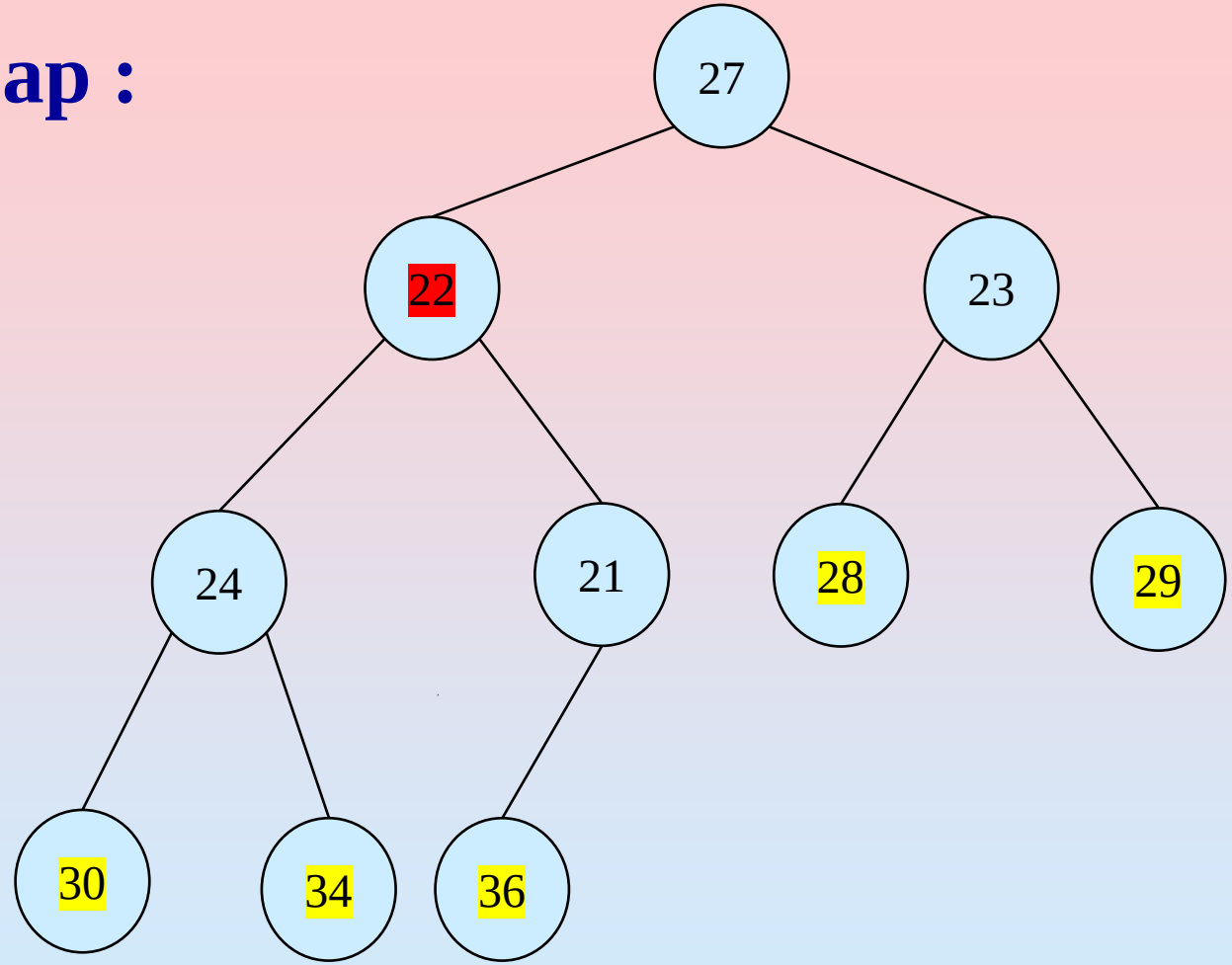


Array representation :

22	27	23	24	21	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

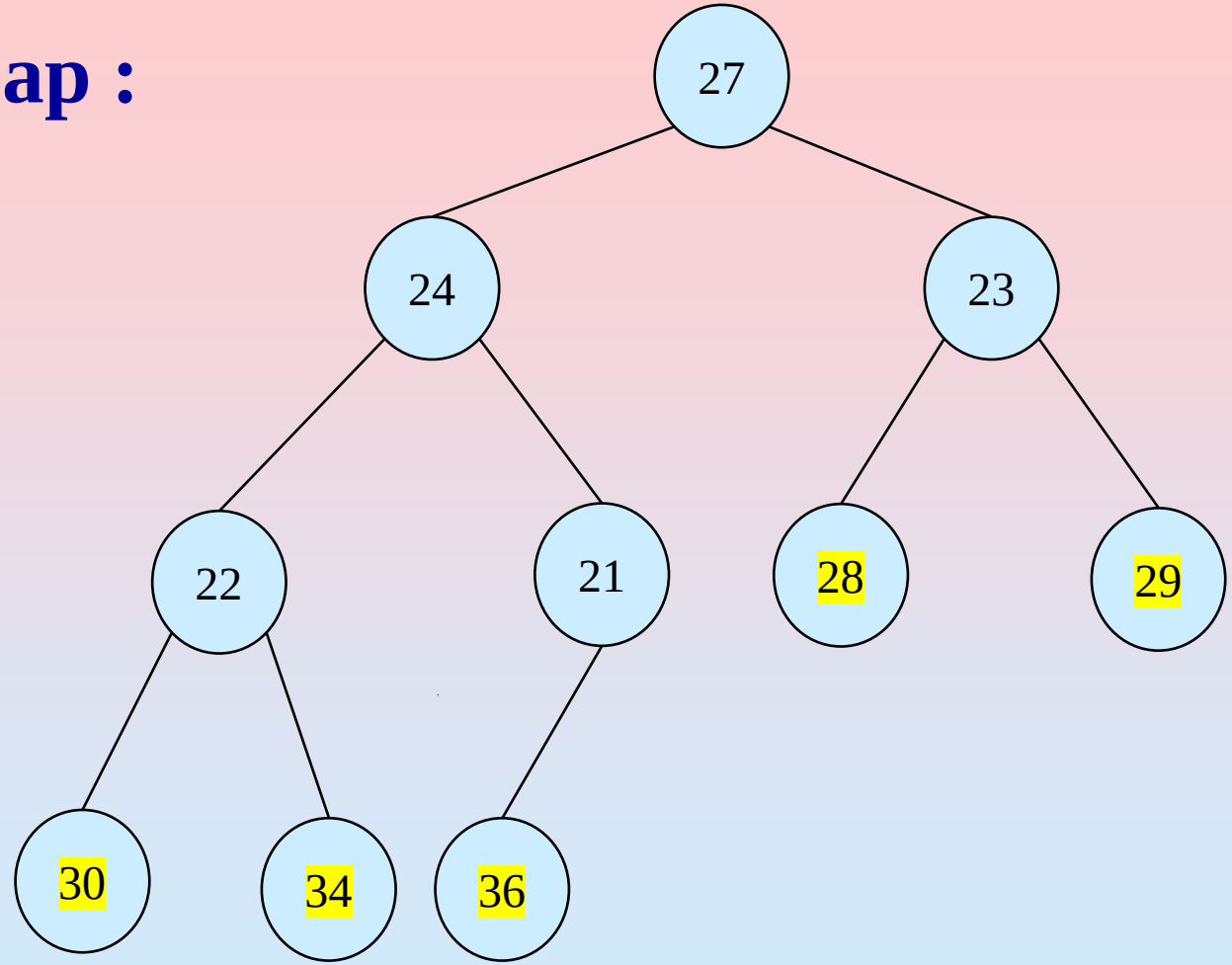


Array representation :

27	22	23	24	21	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

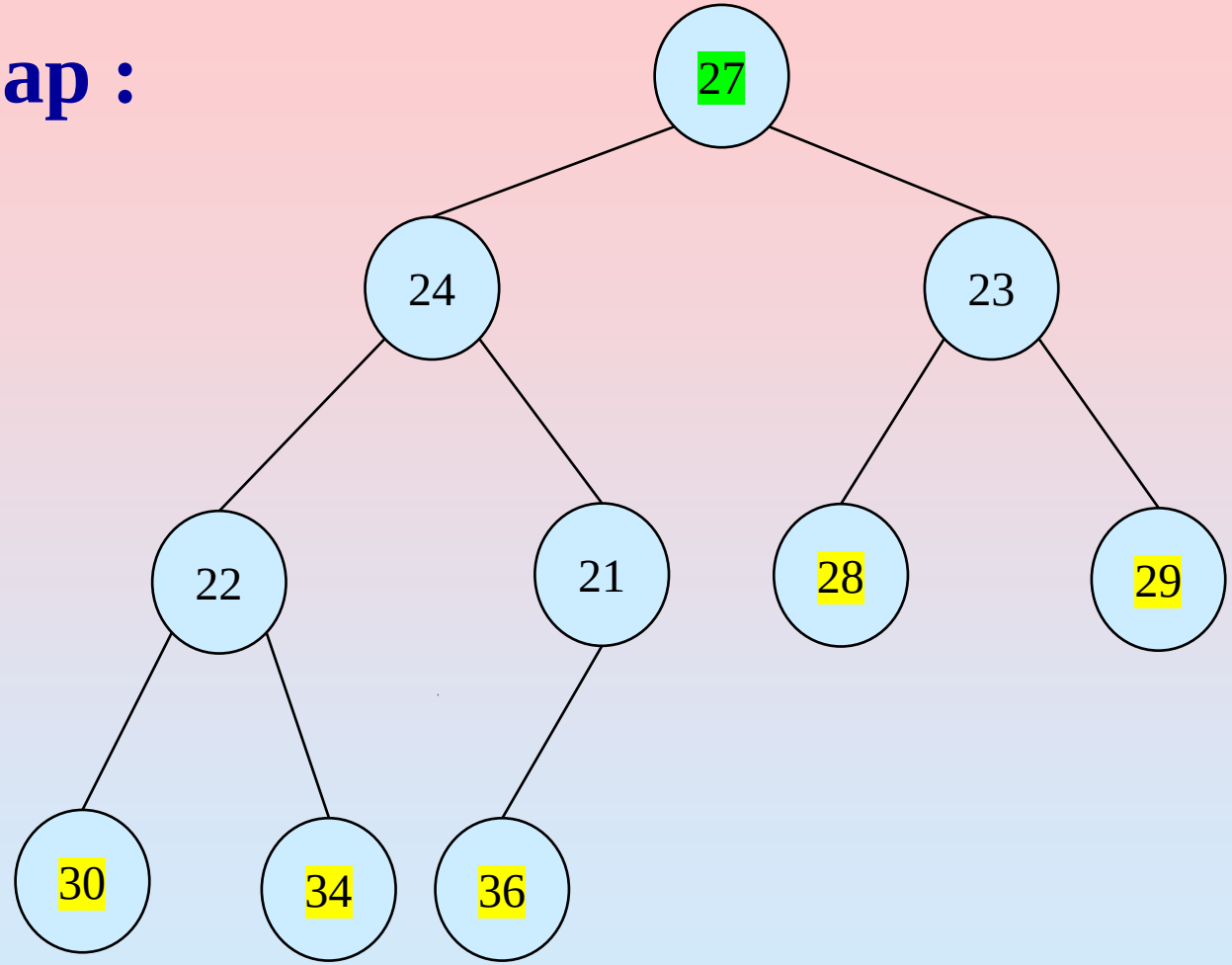


Array representation :

27	24	23	22	21	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

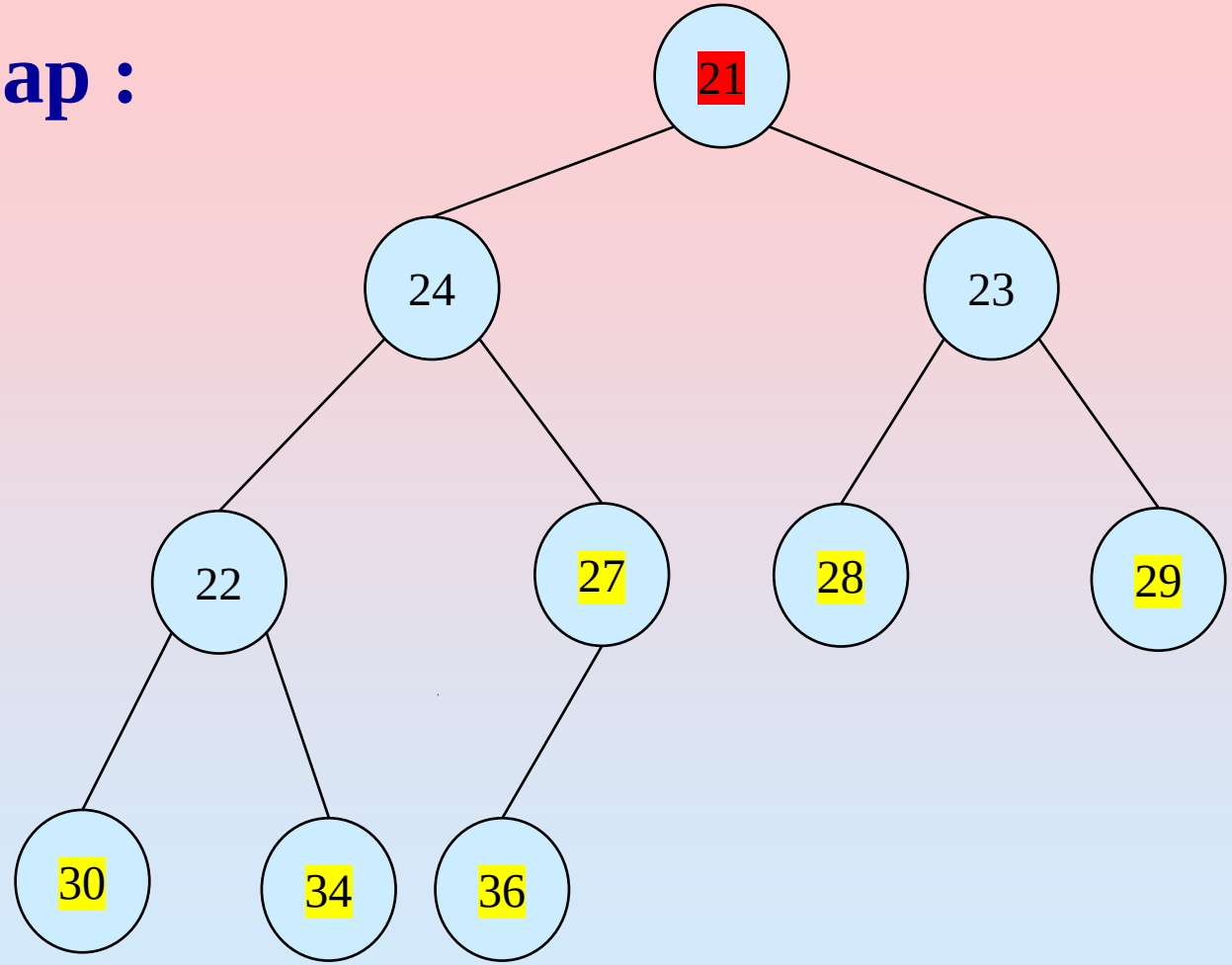


Array representation :

27	24	23	22	21	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

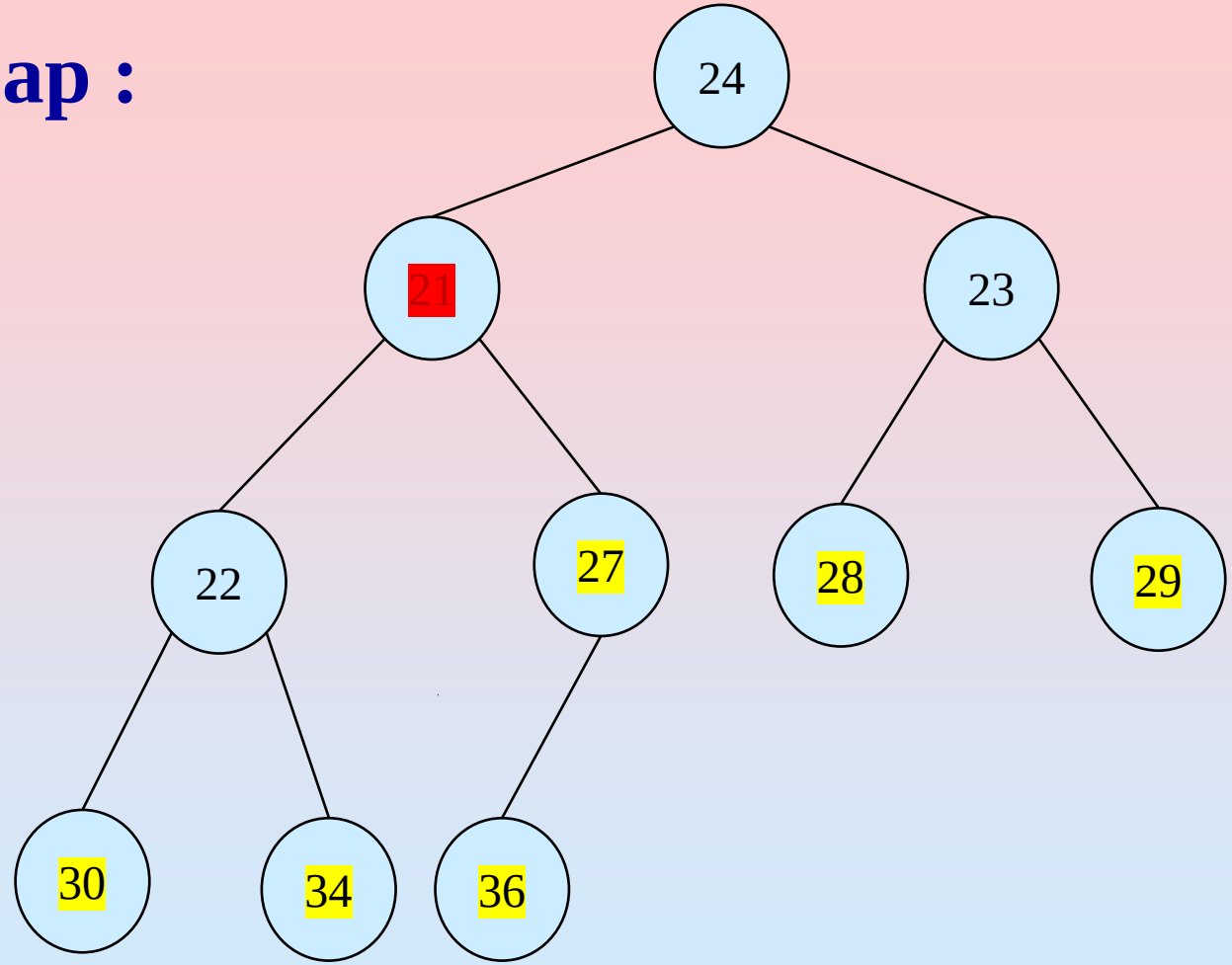


Array representation :

21	24	23	22	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

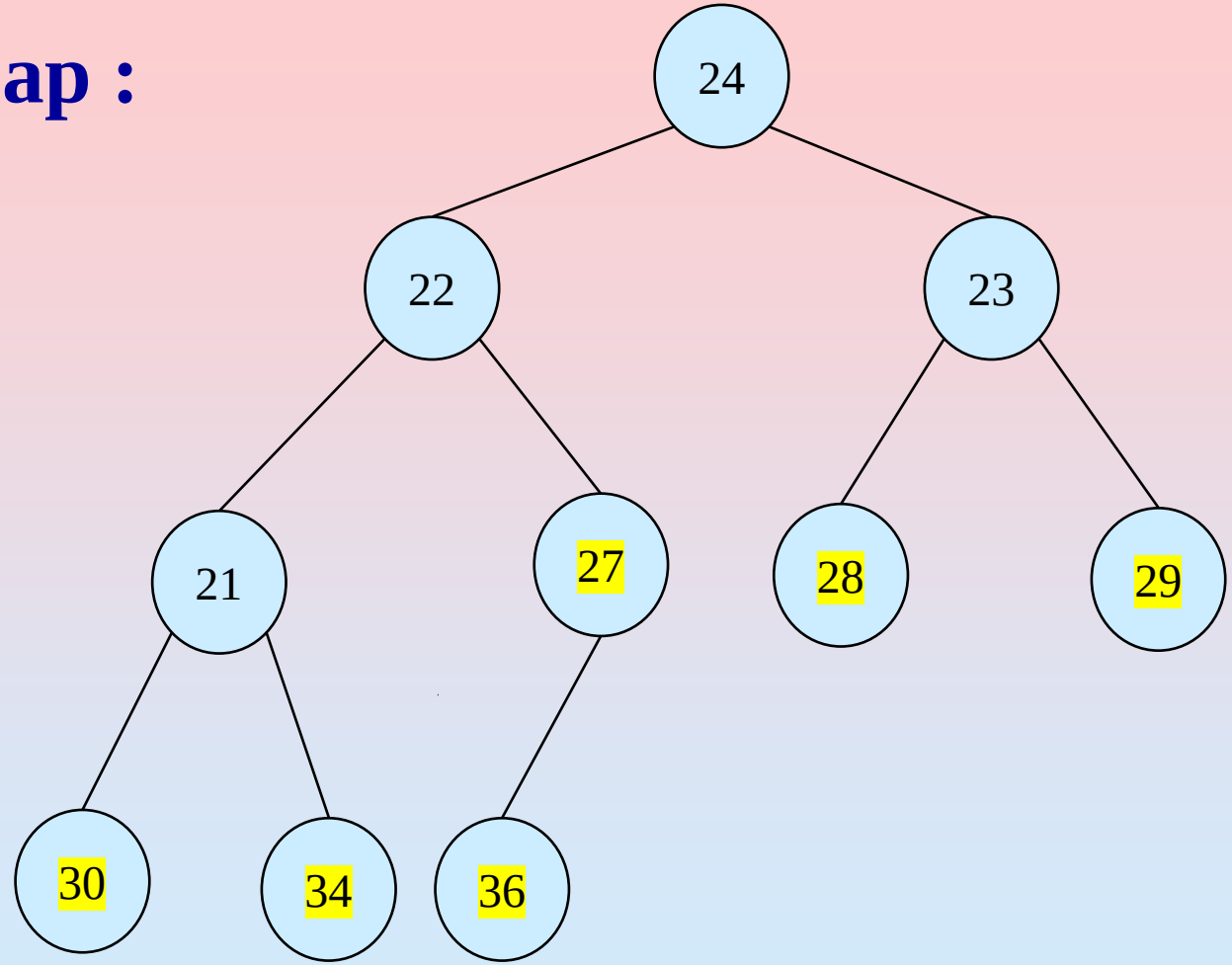


Array representation :

24	21	23	22	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

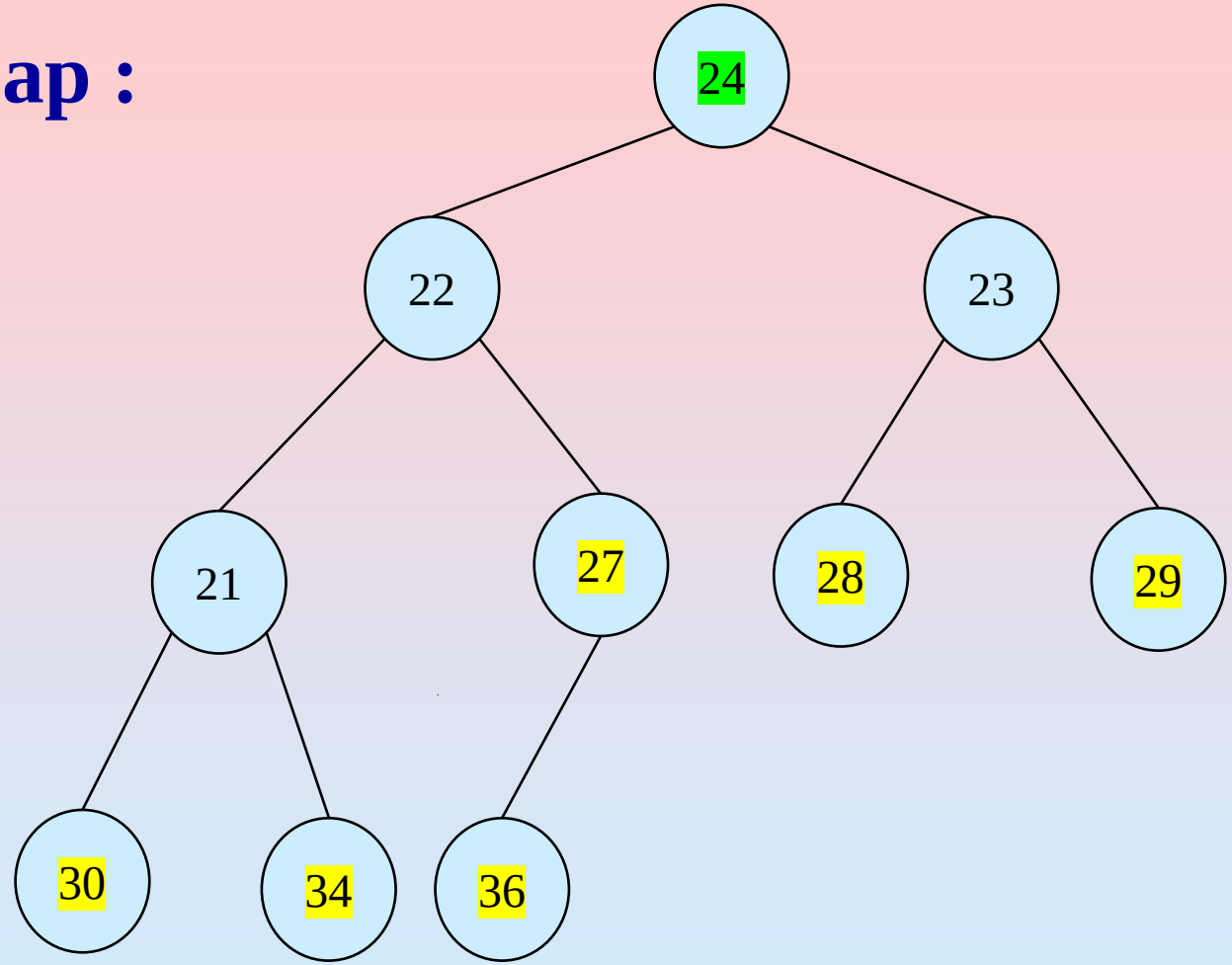


Array representation :

24	22	23	21	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

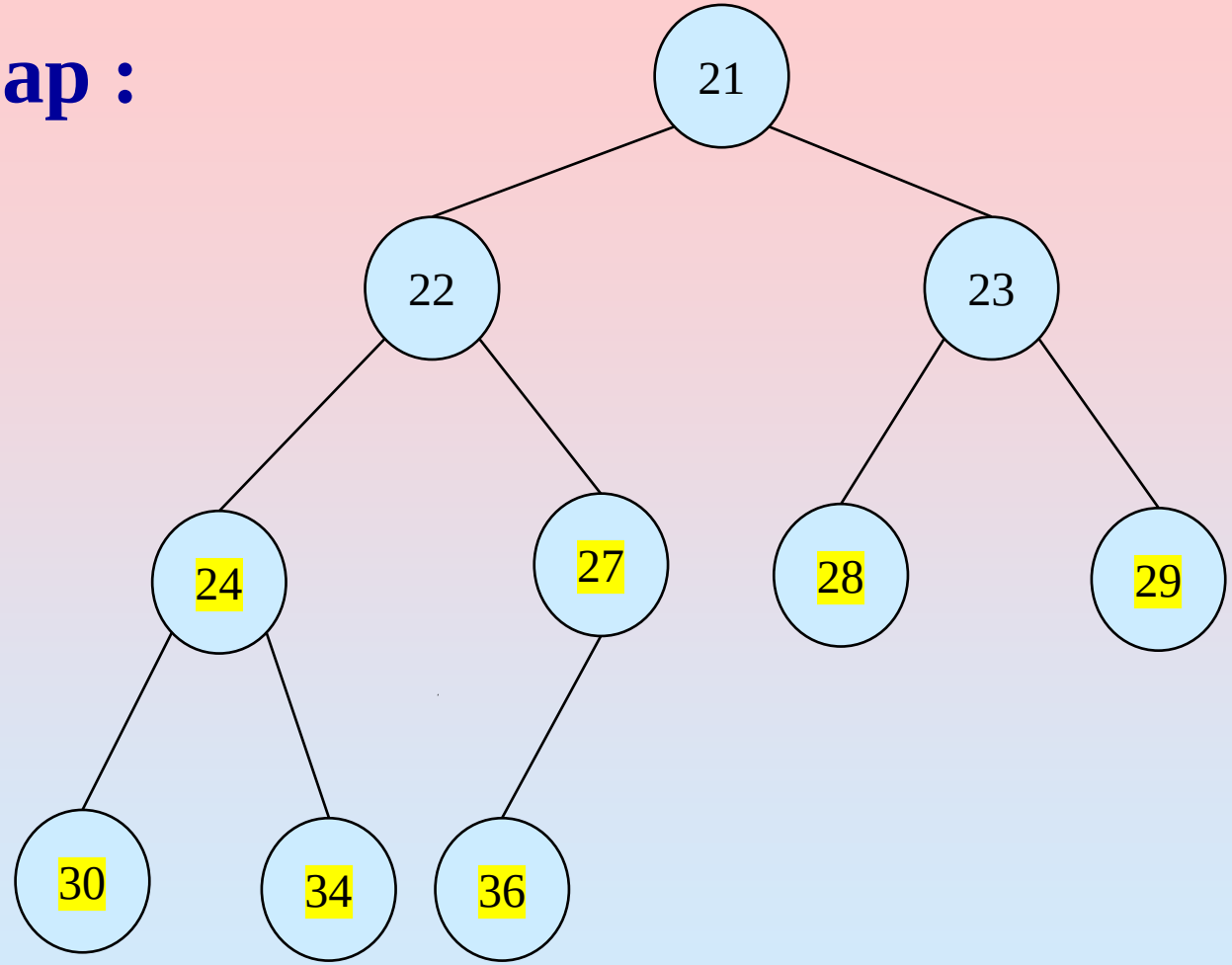


Array representation :

24	22	23	21	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

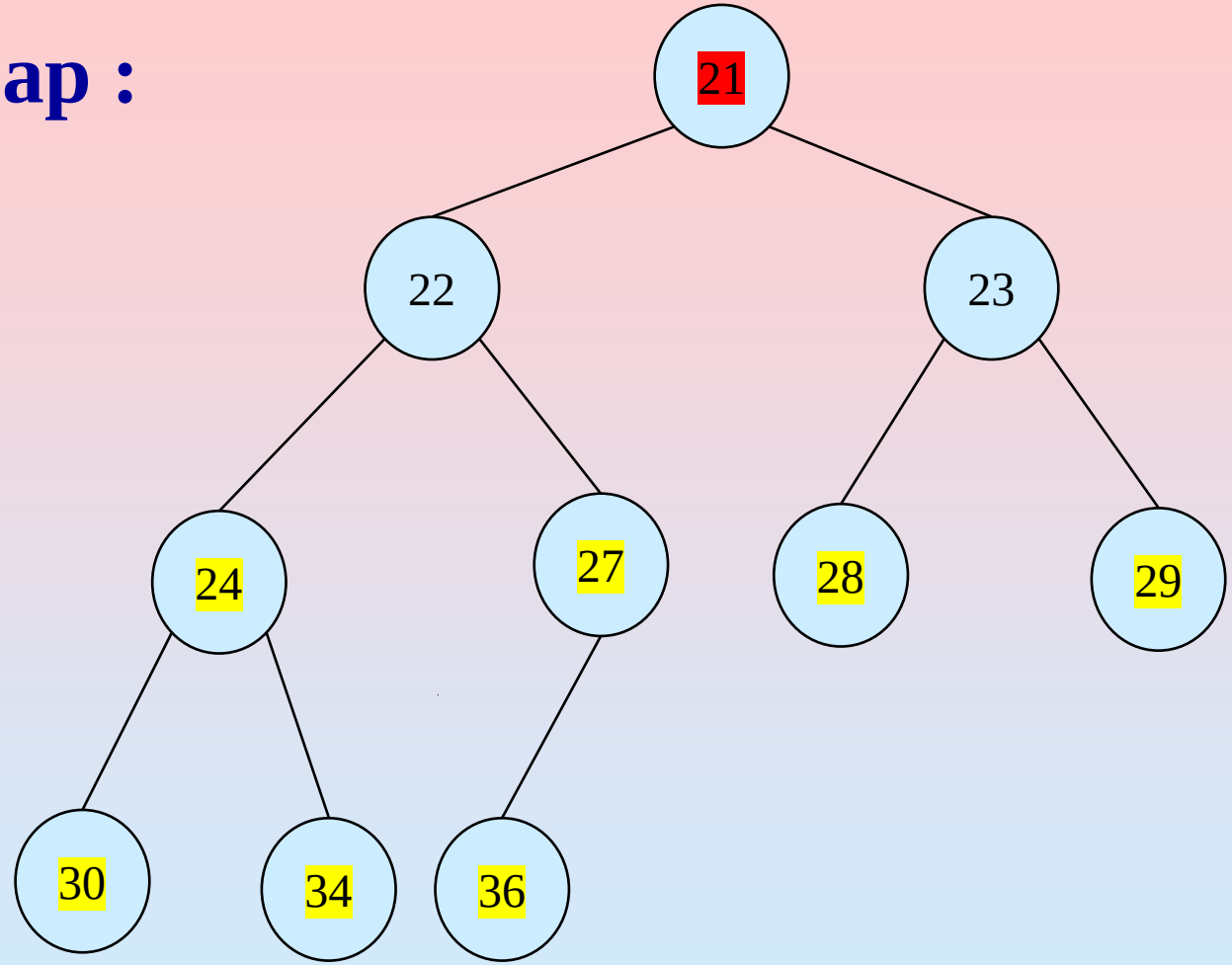


Array representation :

21	22	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

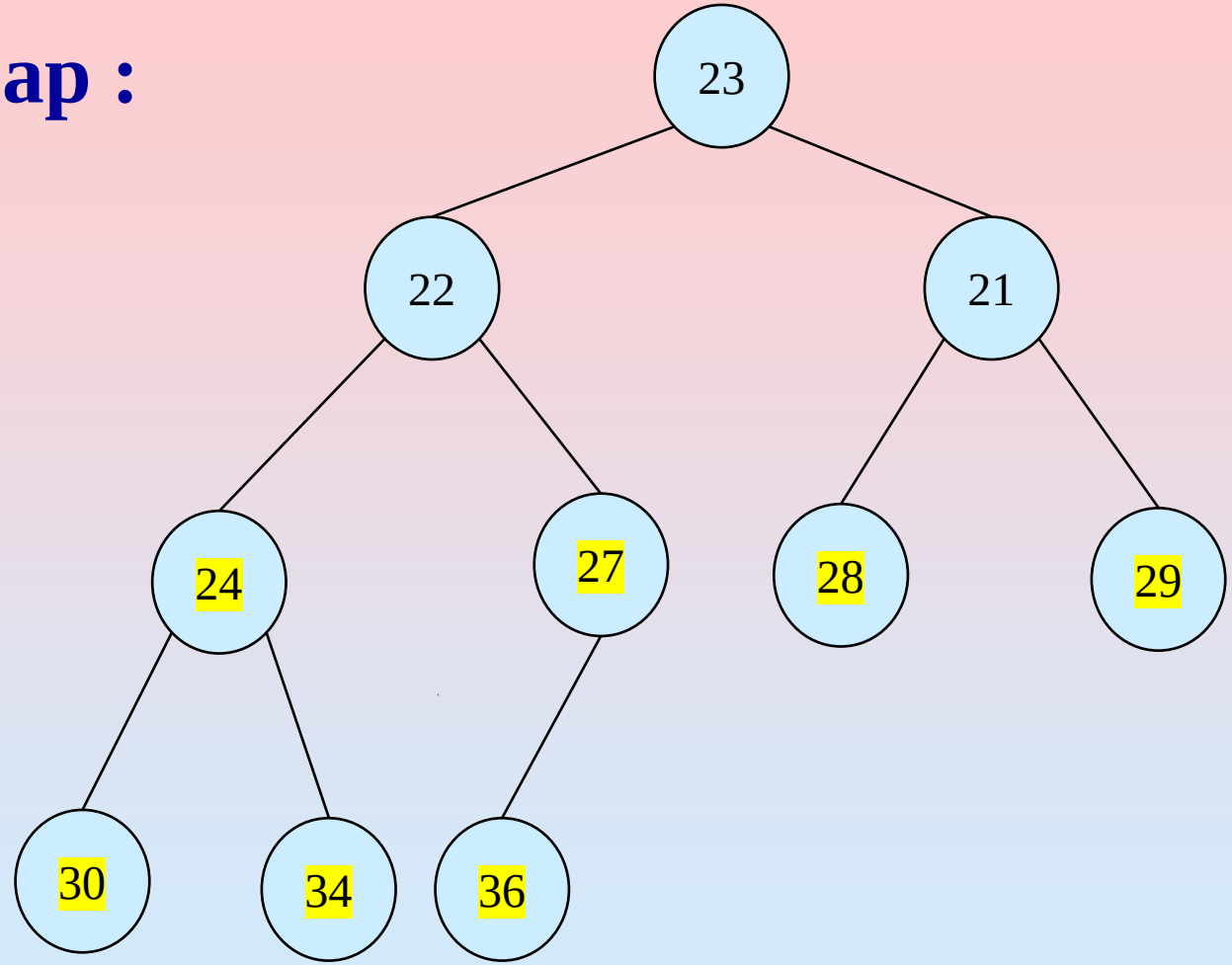


Array representation :

21	22	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

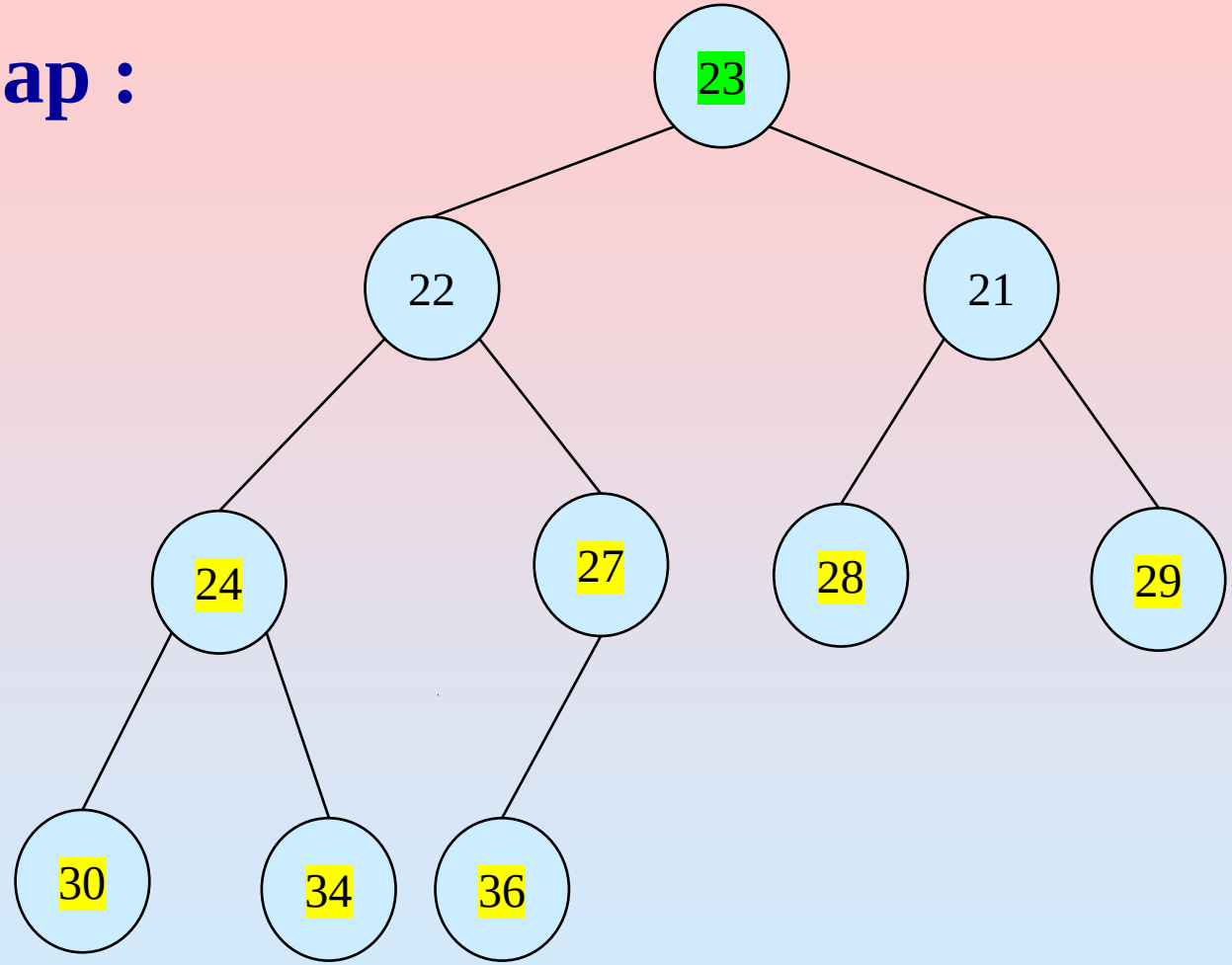


Array representation :

23	22	21	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

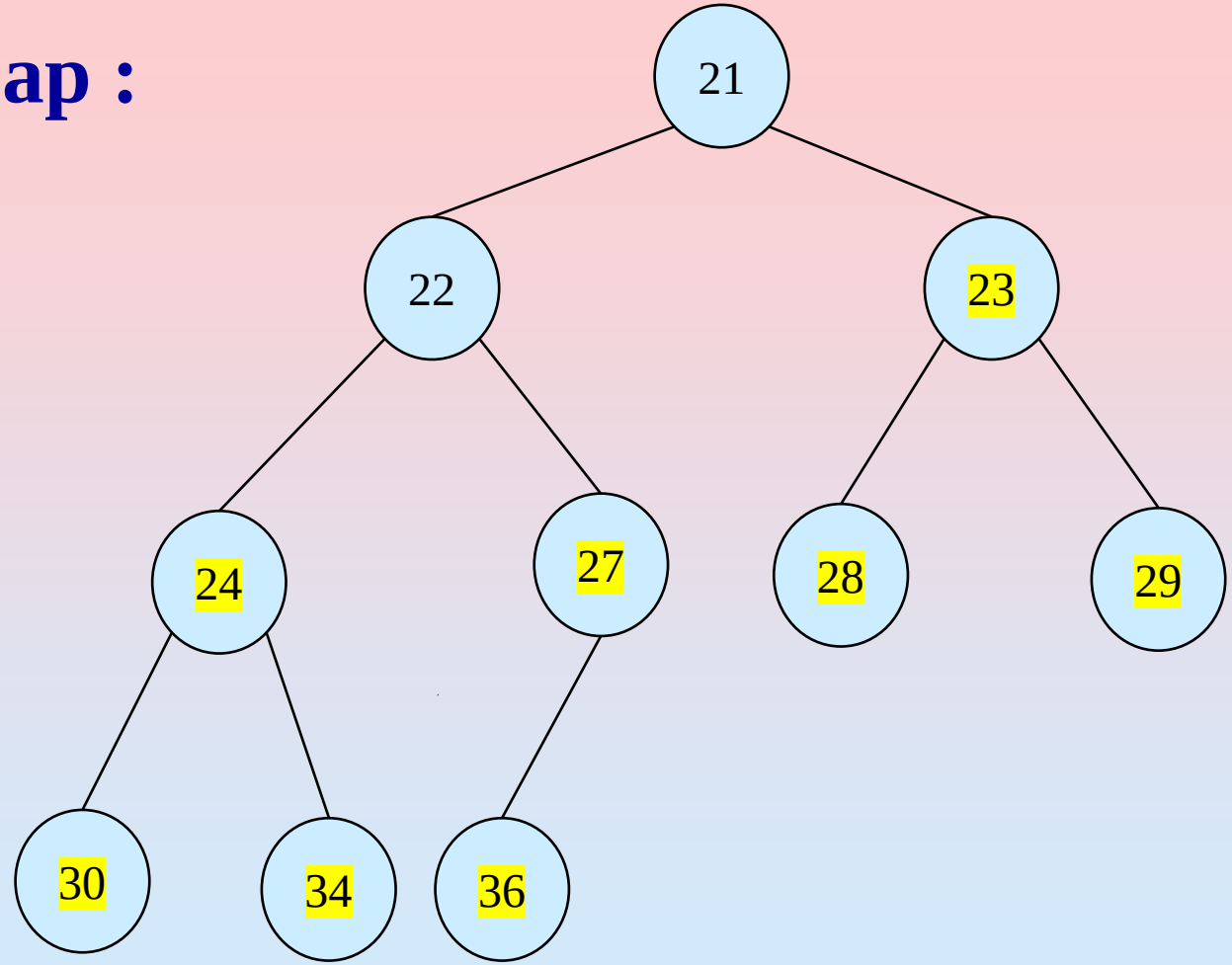


Array representation :

23	22	21	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

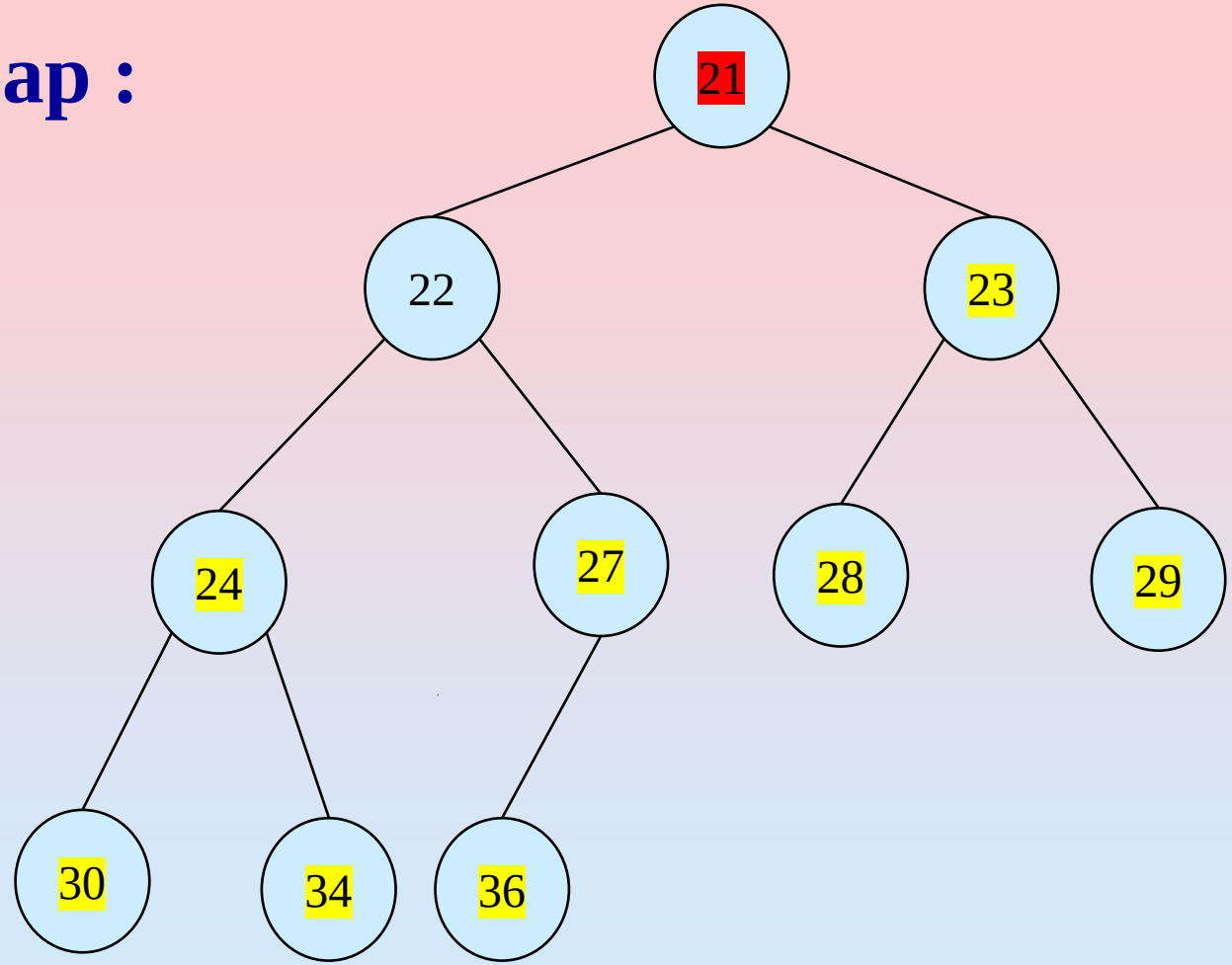


Array representation :

21	22	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

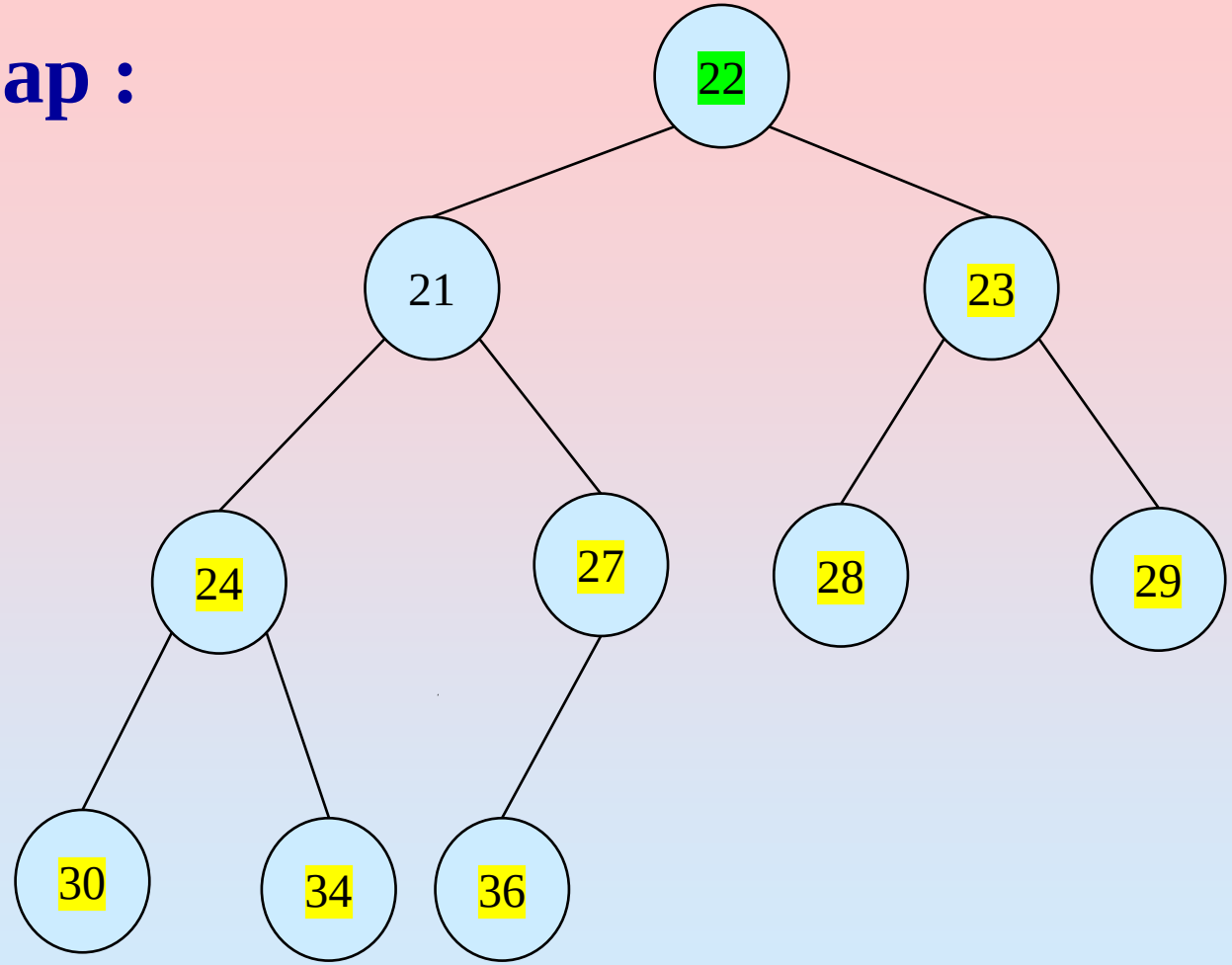


Array representation :

21	22	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

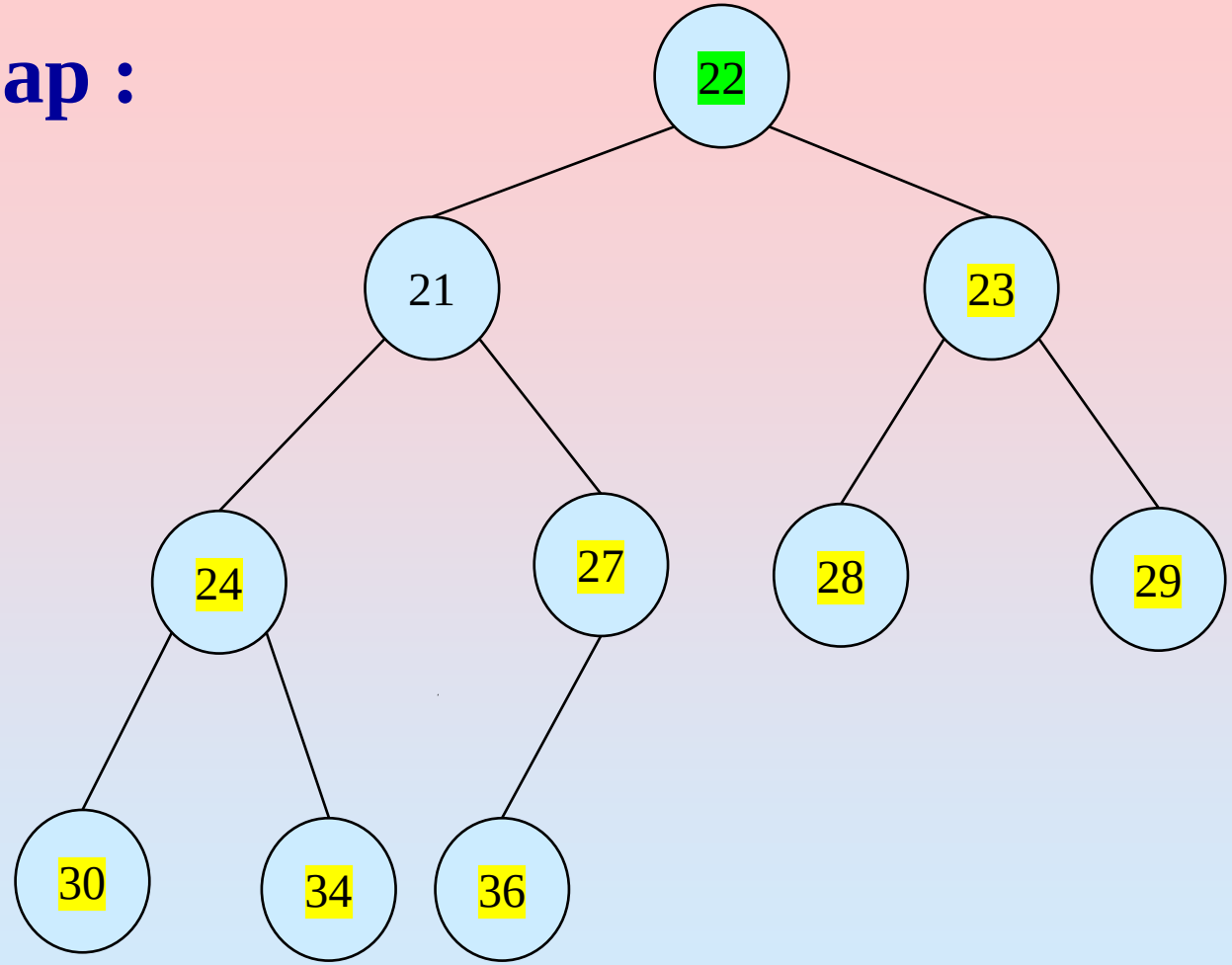


Array representation :

22	21	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

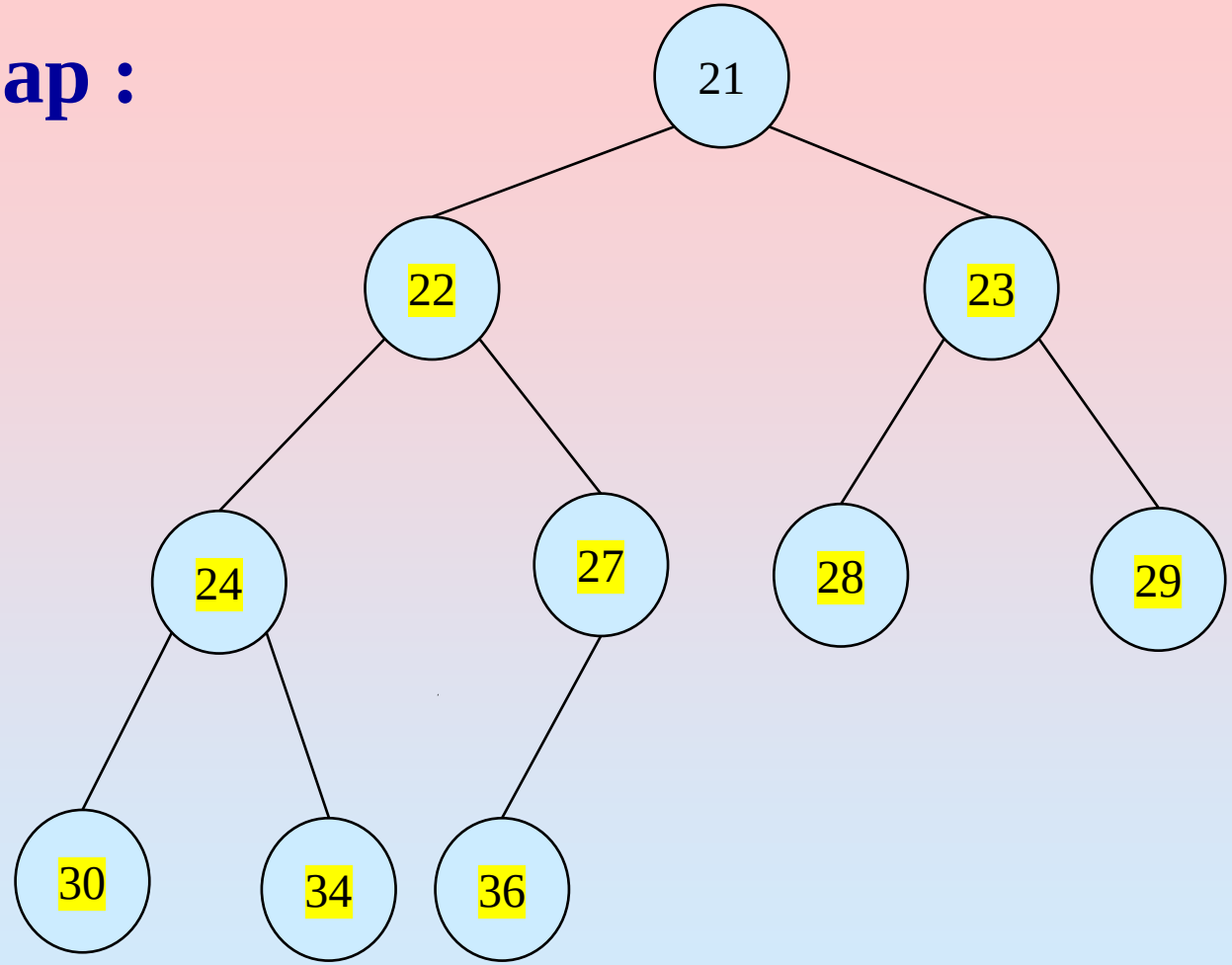


Array representation :

22	21	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

MaxHeap :

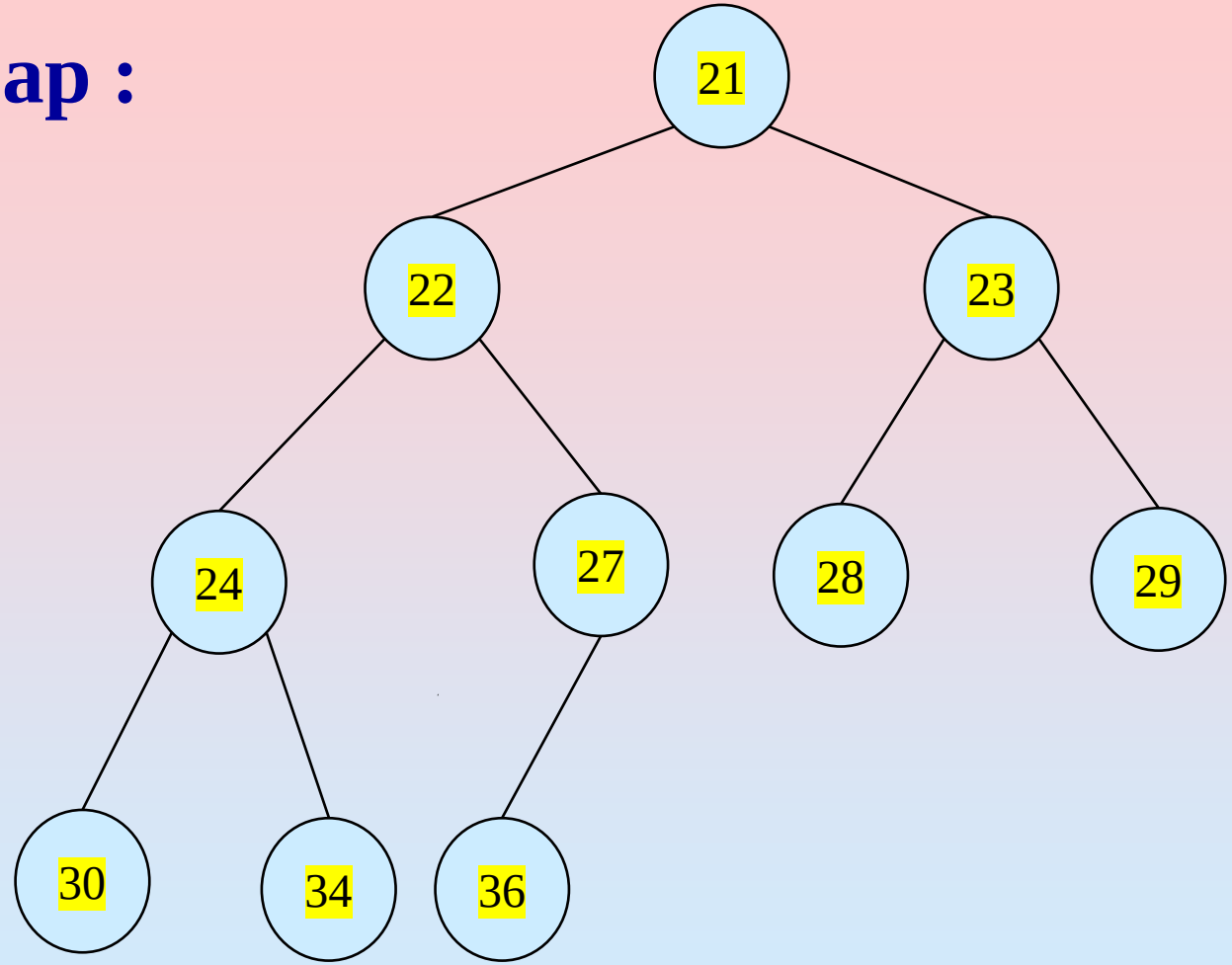


Array representation :

21	22	23	24	27	28	29	30	34	36

Show the contents of the array during HeapSort

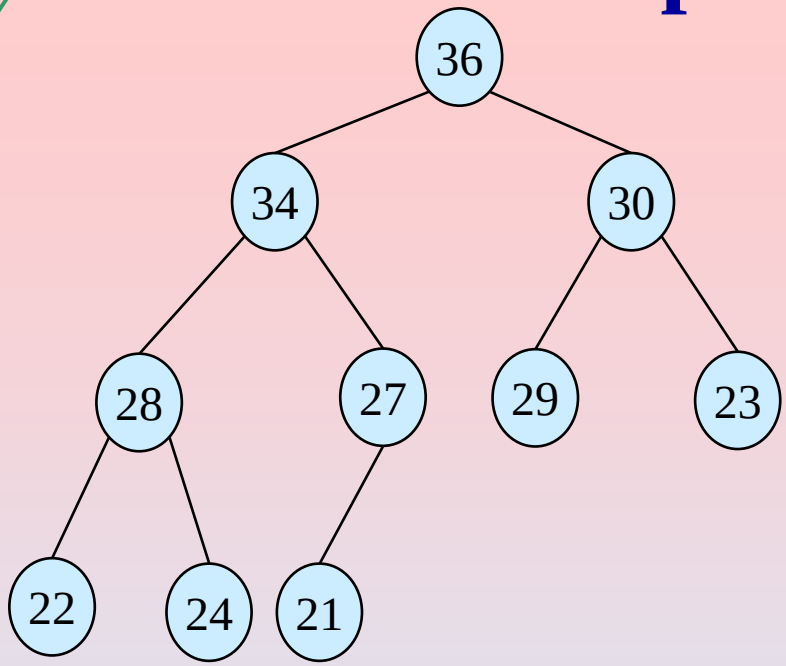
MaxHeap :

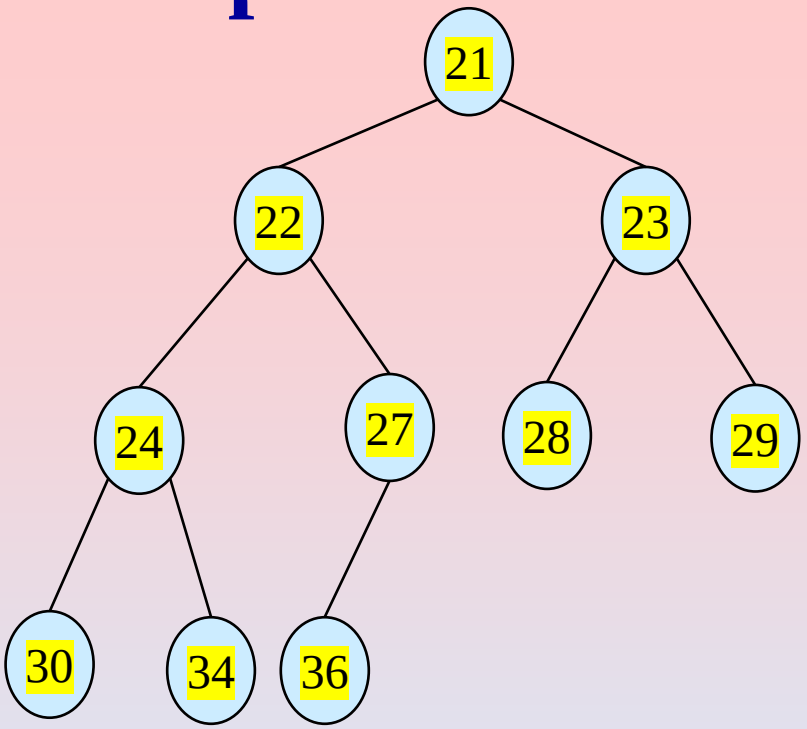


Array representation :

21	22	23	24	27	28	29	30	34	36

HeapSort – Example





Array representation before Heap Sort:

36	34	30	28	27	29	23	22	24	21

Array representation after Heap Sort:

21	22	23	24	27	28	29	30	34	36

Thank you