Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

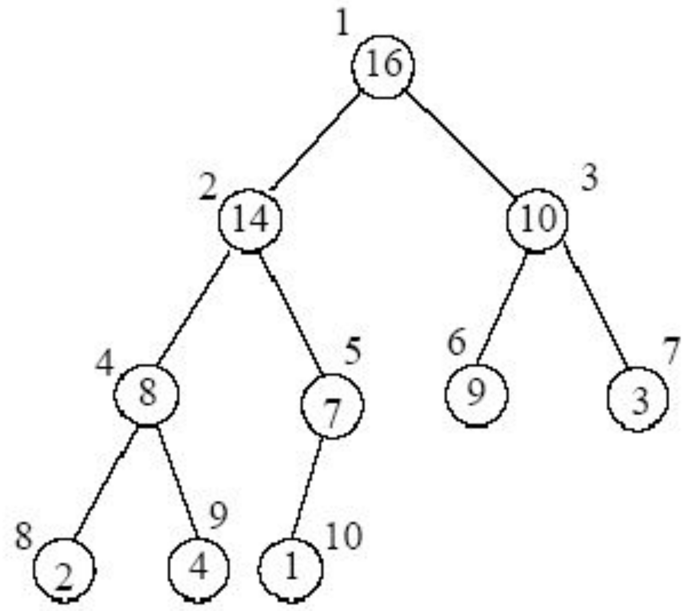
1. Heap Data Structure
2. Heap Sort

Heap Data Structure

- **Introduction**
  - Heap is coined in the context of **Heap Sort**
  - It is also used to maintain **Priority Queue**
  - Heap structure is used as "**Garbage-collected Storage**" in Java and Lisp programming language
  - Heap data structure is also used to maintain "**Dynamic Memory Allocation**" in C language (whereas for static memory allocation it uses **Stack**)
  - Heap (or binary heap) data structure can be viewed as a complete binary tree that maintains some constraints

Heap Data Structure

- **Array Representation**



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

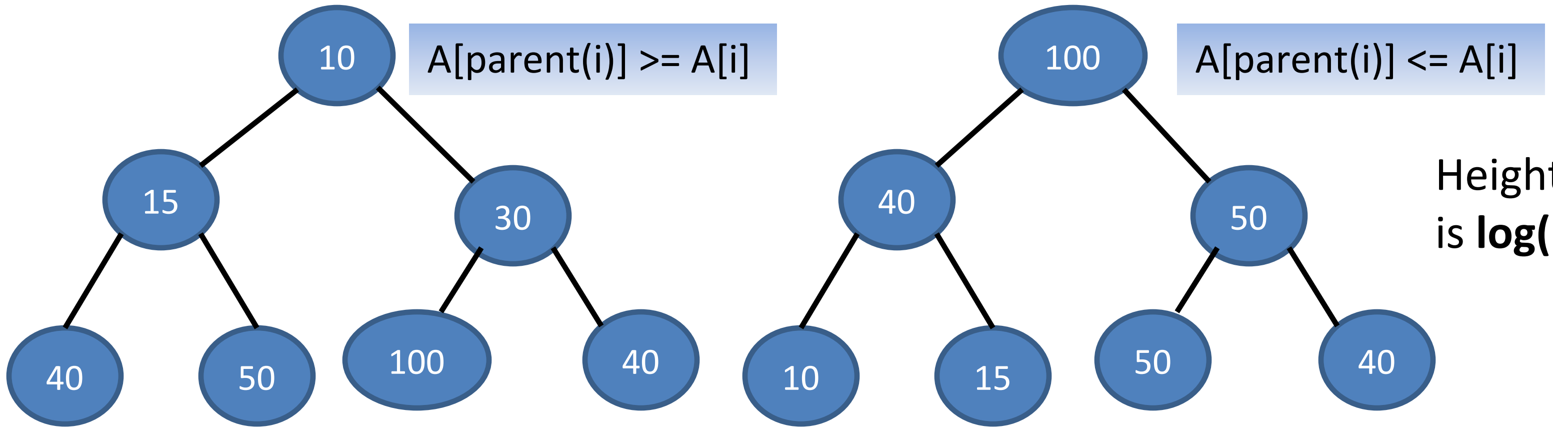
```
Parent(i)
   return  floor(i/2)

Left(i)
   return 2i

Right(i)
   return 2i+1
```

Heap Data Structure

- **Types of Heap**
  - Max heap (used in Heap Sort)
    - ✔ *The value of a node is at most the value of its parent*
  - Min heap (used in priority queue)
    - ✔ *The smallest element in a min-heap is at the root*



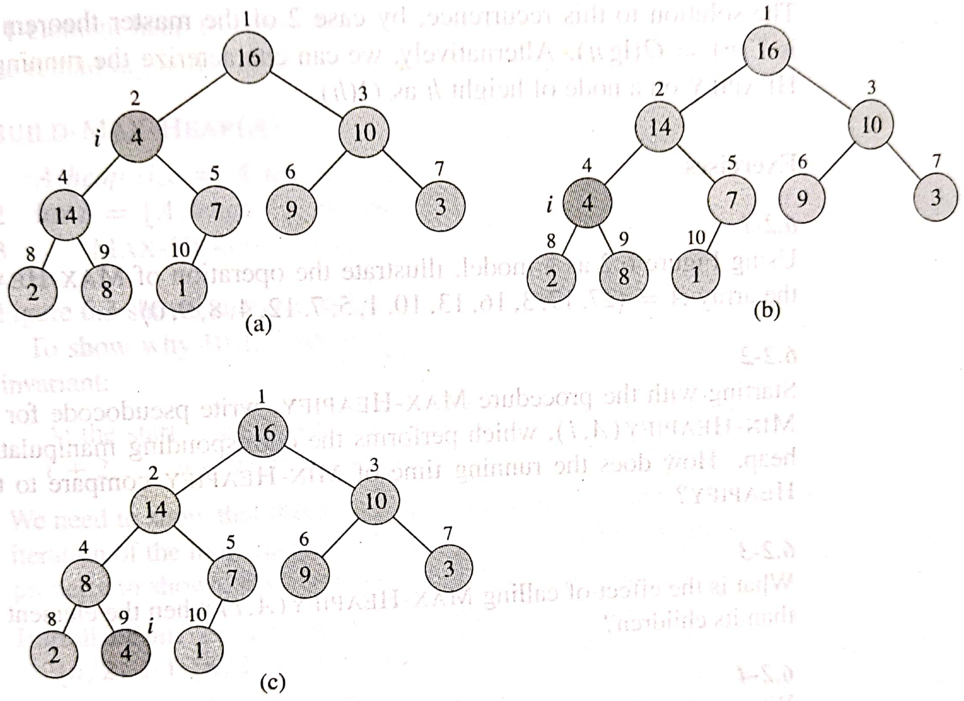
Min Heap

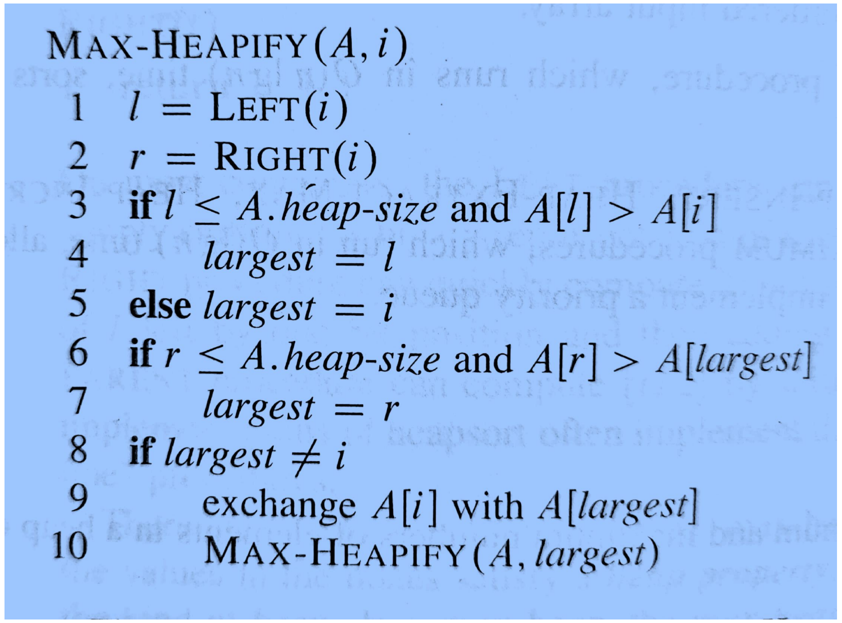
Max Heap

Height of Heap is **log(n)**

Heap Data Structure

- **Max Heapify**



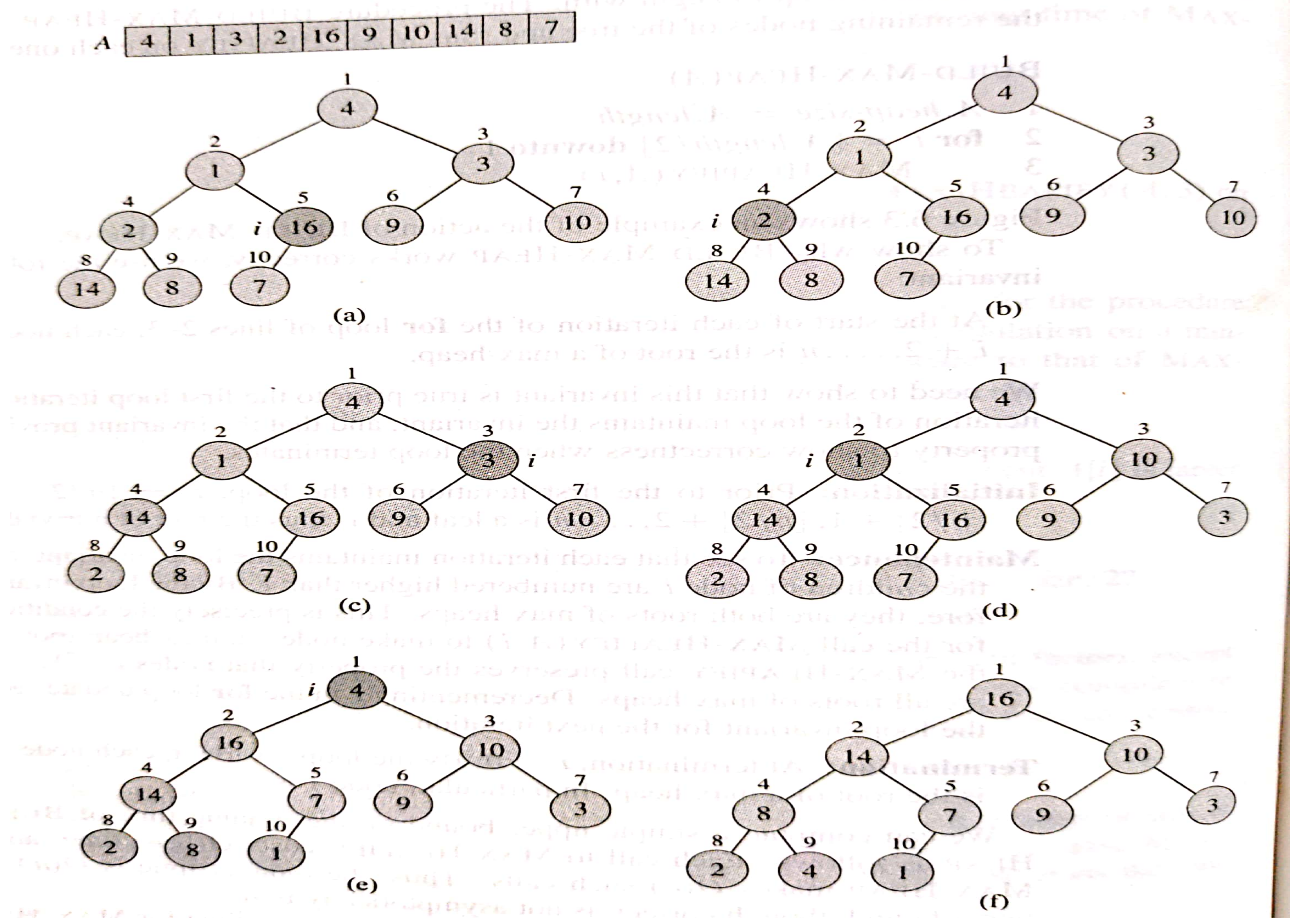


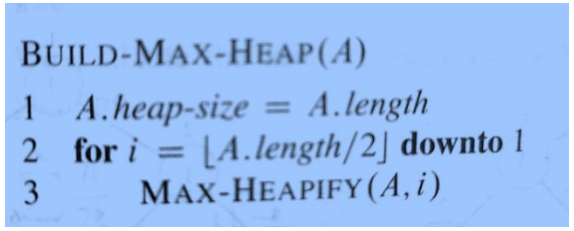
```
MAX-HEAPIFY(A, i)
1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4      largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7      largest = r
8  if largest ≠ i
9      exchange A[i] with A[largest]
10     MAX-HEAPIFY(A, largest)
```

Heapify Costs => **O(log n)**

Heap Data Structure

- **Building a Max-Heap**





```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Build Heap => **O(n)**

Can be proved mathematically

Heap Sort

- **Algorithm**

```
Heap_sort (A)   // A is the input array
{
  Build-Max-Heap (A)
  for i = A.length downto 2
          exchange A(1) with  A(i)      // A(1) signifies the root element
          A.heap_size := A.heap_size – 1
          Max-Heapify(A, 1)
}
```

- **Complexity**

1. Build-Max-Heap takes time O(n)
2. Each Max-Heapify takes time O(log n) from the loop
3. Entire Heap Sort takes time O(n log n)

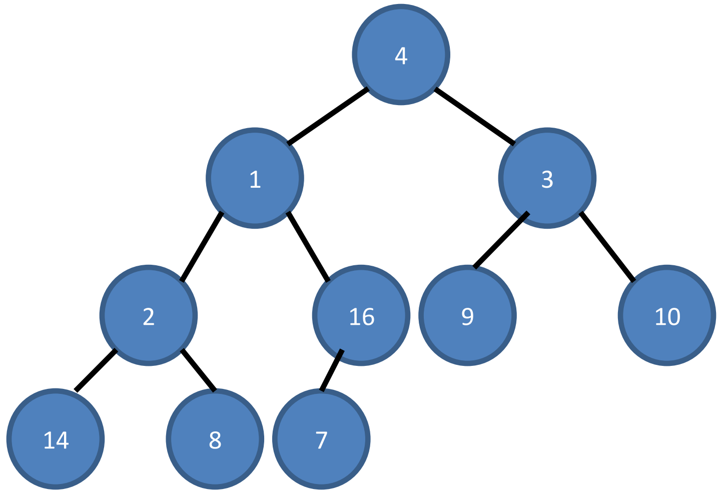
TECHNO INDIA UNIVERSITY
WESTBENGAL

Heap Sort

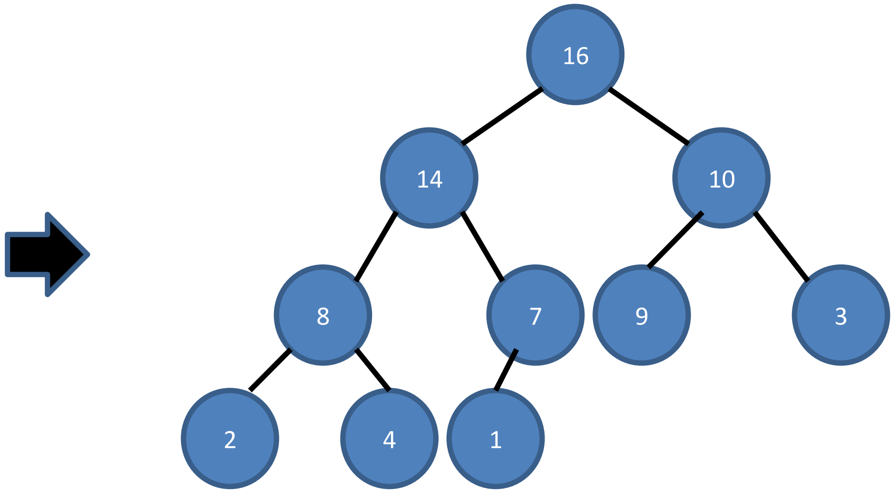
- **Mechanism**

Input array

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



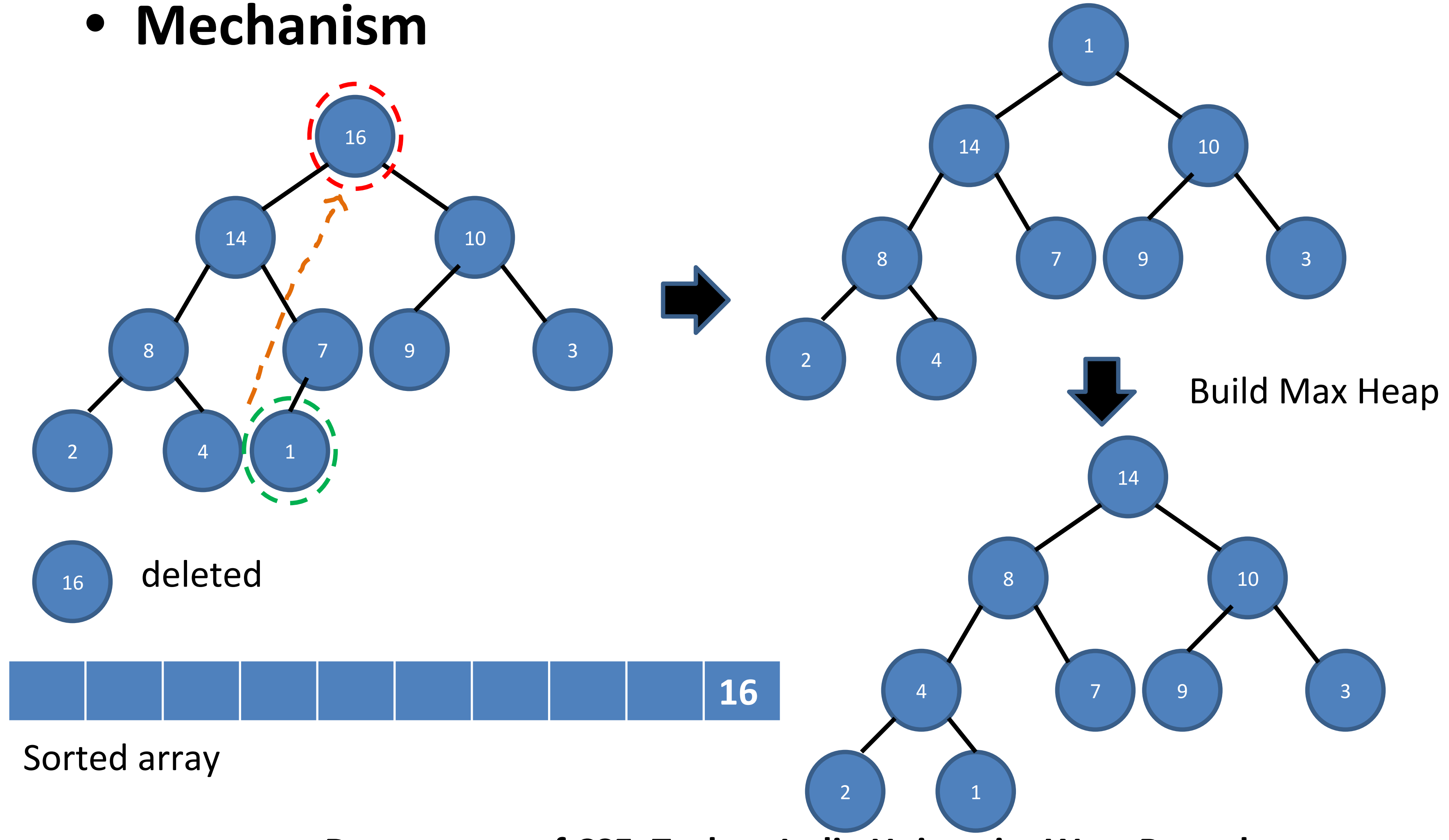
Create the Heap



Build Max Heap

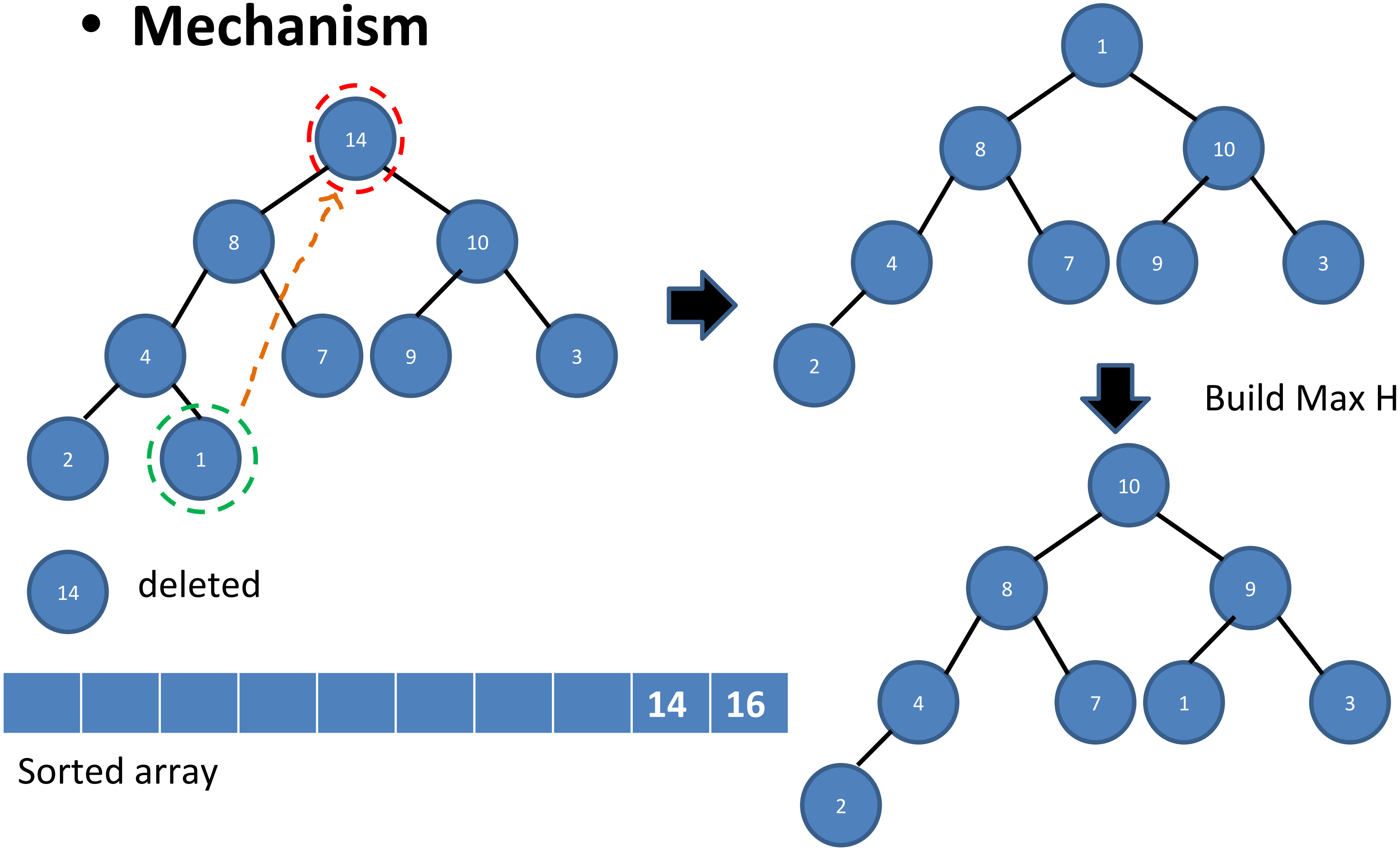
Heap Sort

- **Mechanism**



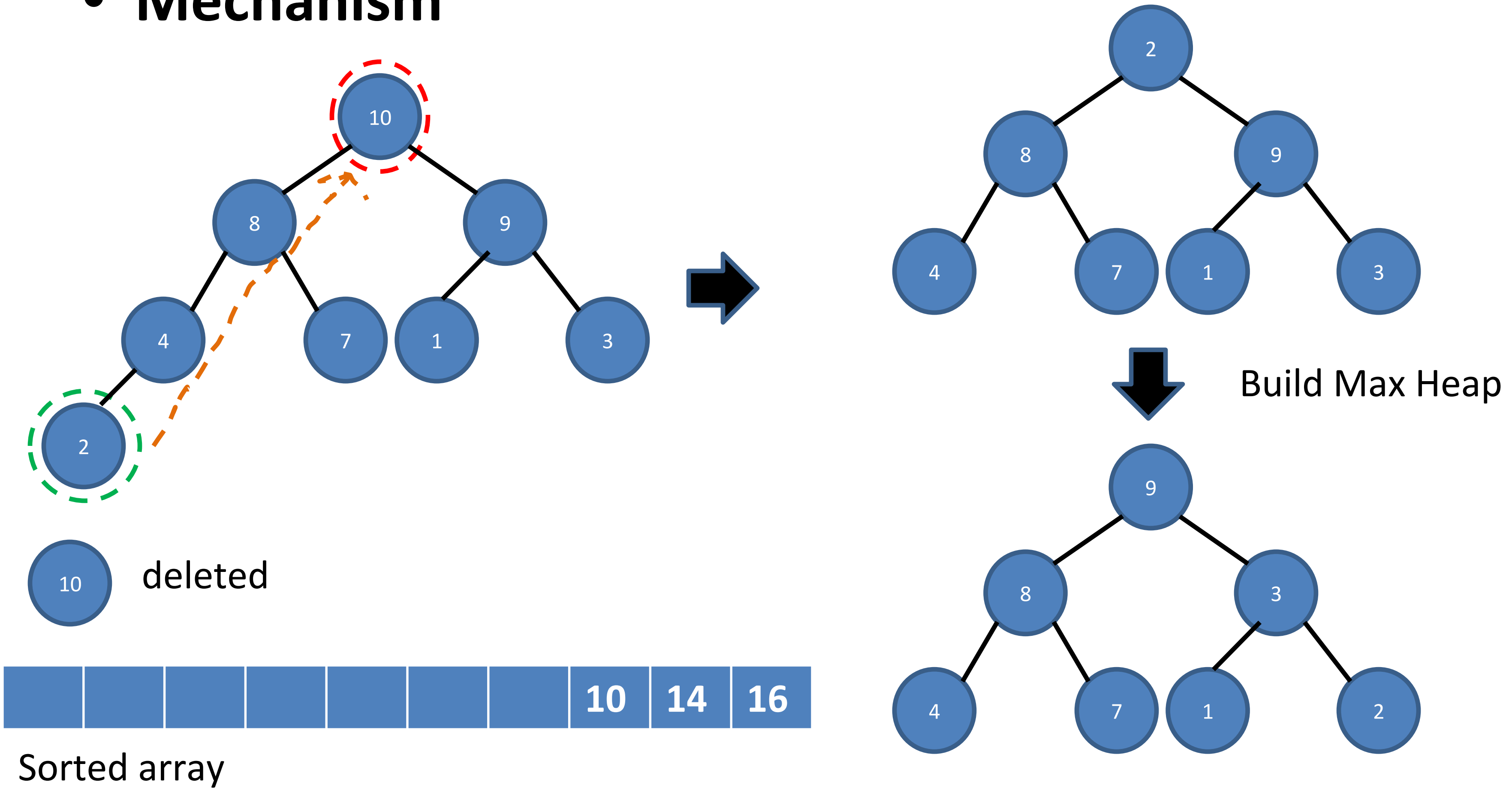
Heap Sort

- **Mechanism**



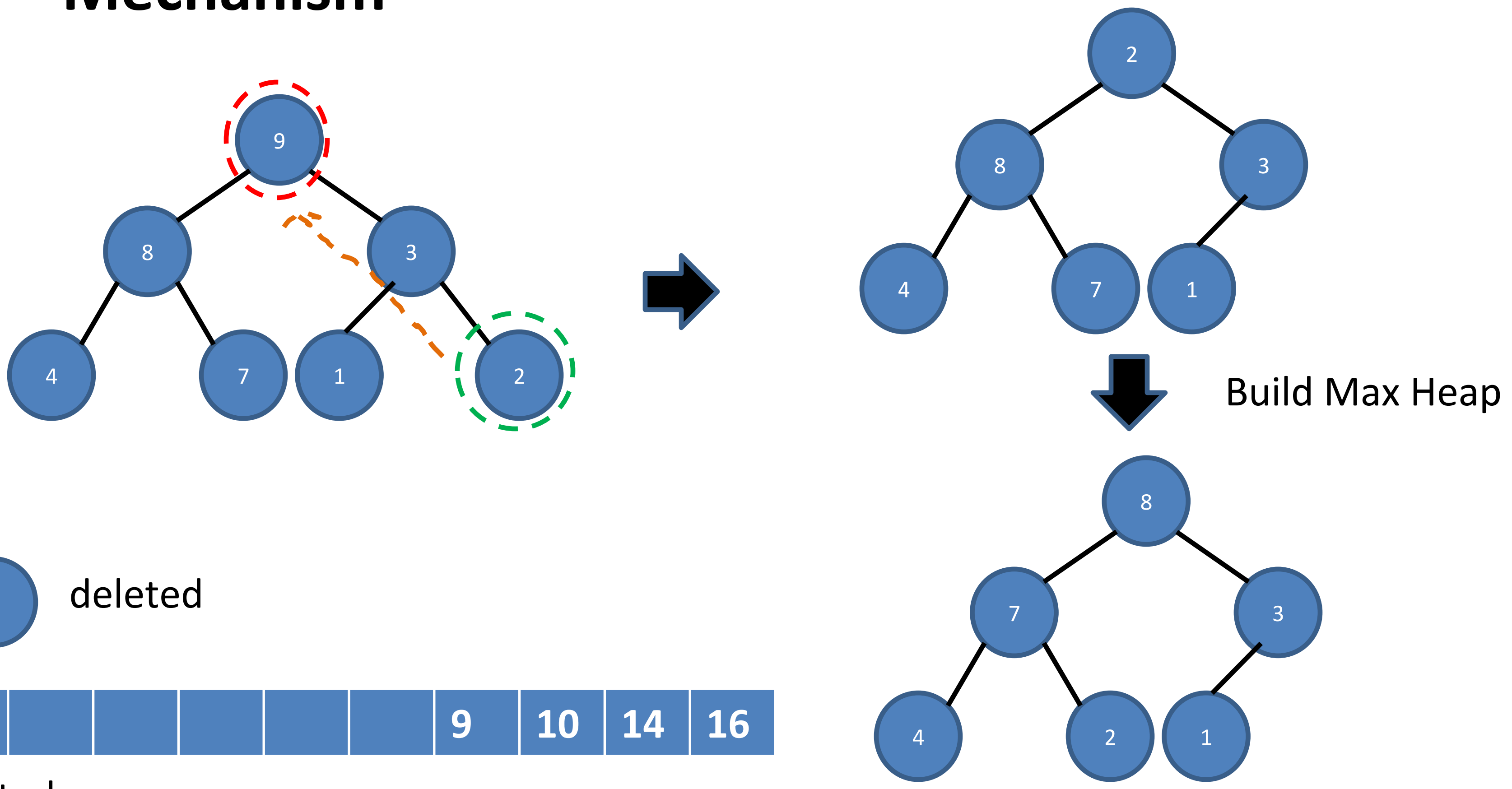
Heap Sort

- **Mechanism**

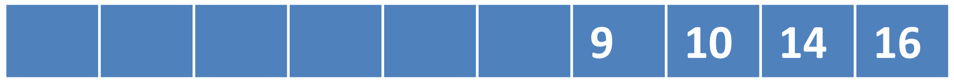


Heap Sort

- **Mechanism**



deleted

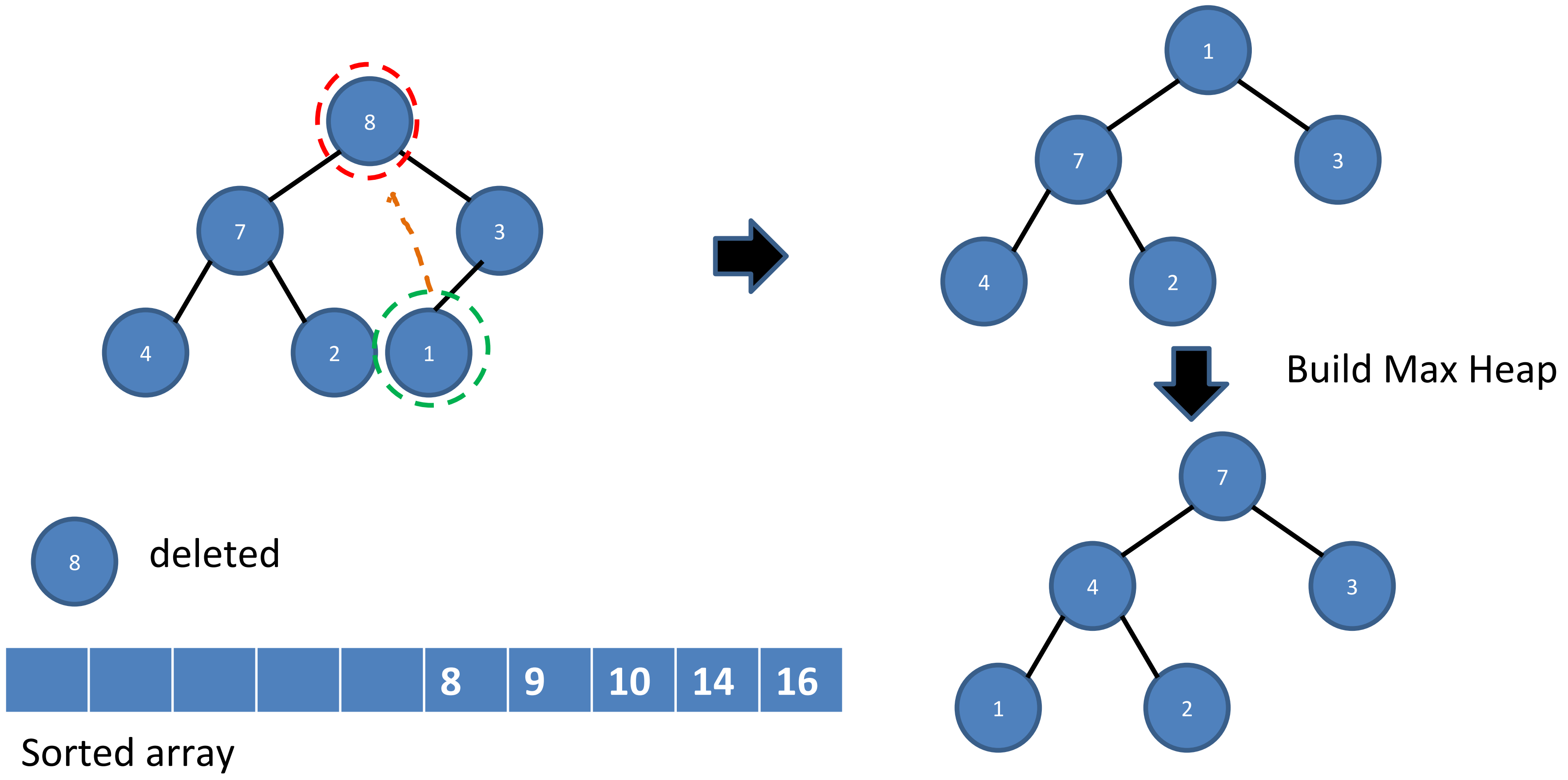


						9	**10**	**14**	**16**

Sorted array

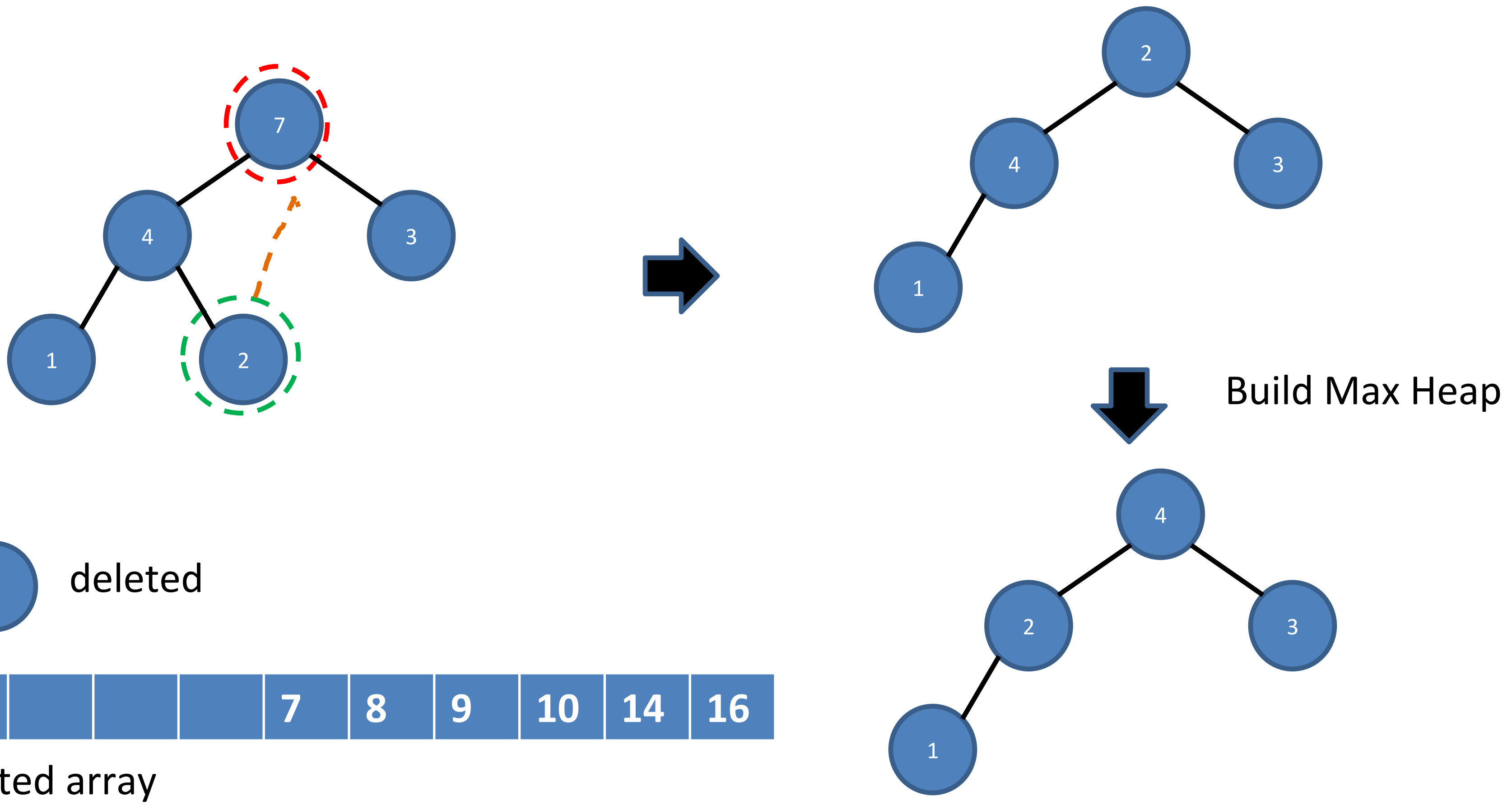
Heap Sort

- **Mechanism**



Heap Sort

- **Mechanism**

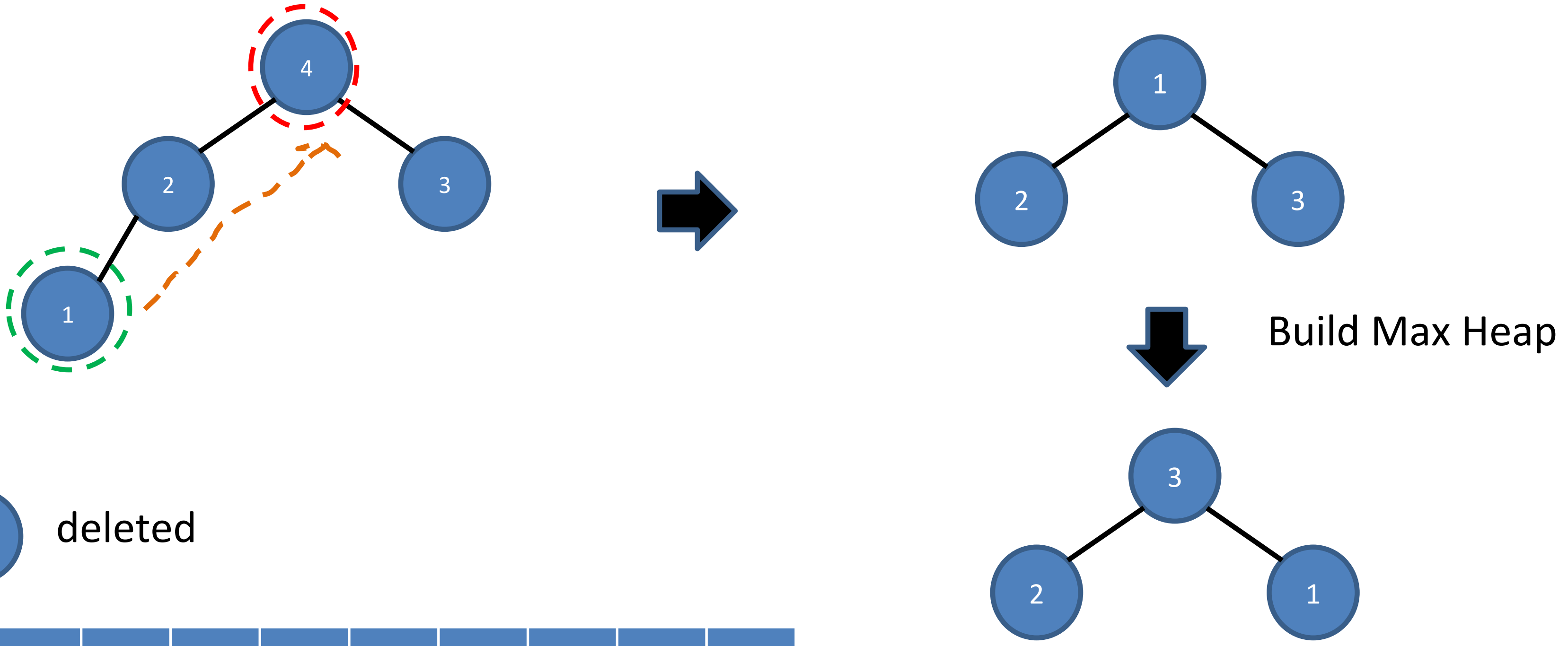


				7	8	9	10	14	16

Sorted array

Heap Sort

- **Mechanism**

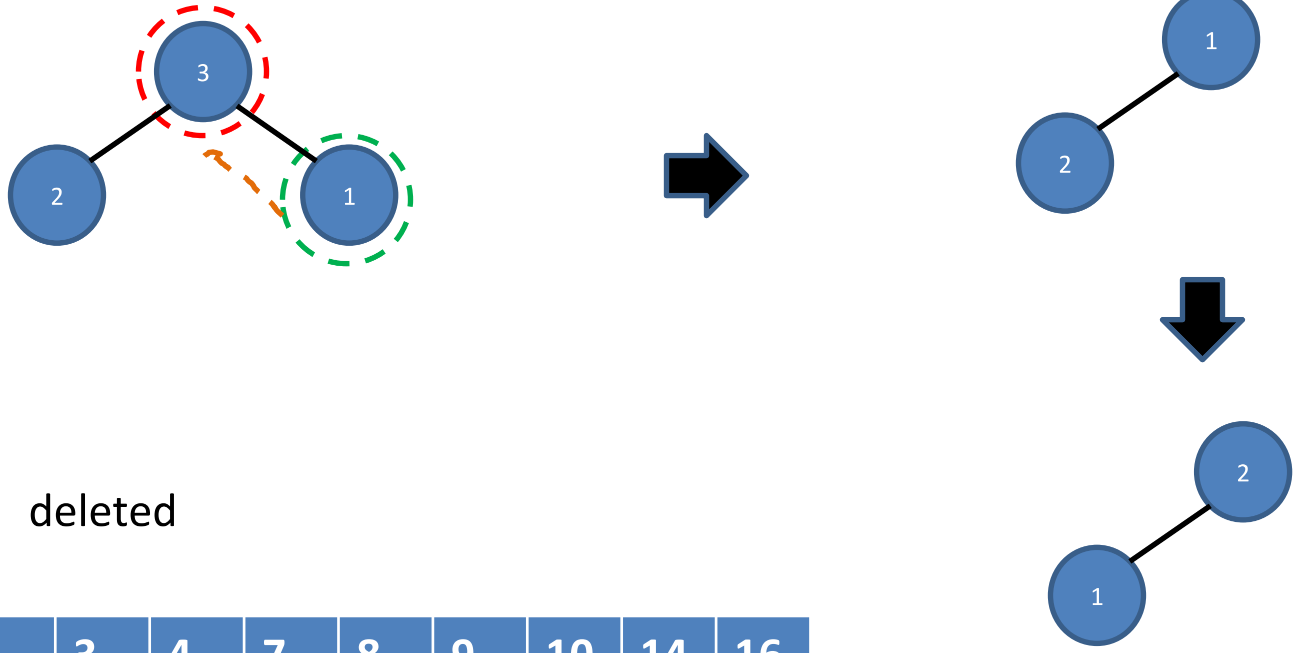


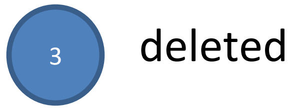
			4	**7**	**8**	**9**	**10**	**14**	**16**

Sorted array

Heap Sort

- **Mechanism**



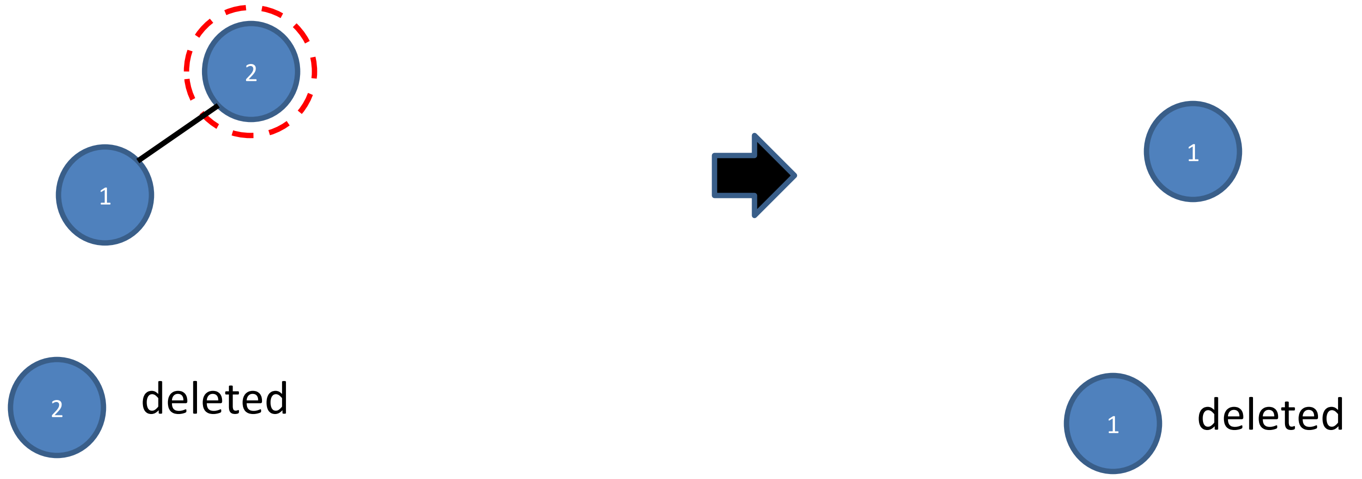


		3	4	7	8	9	10	14	16

Sorted array

Heap Sort

- **Mechanism**



1	2	3	4	7	8	9	10	14	16

Sorted array

Queries?

Practice Problem

1. Apply heap sort algorithm on the input data set provided below to arrange them in ascending order

{ 6 , 7 , 1 , 10 , 9 , 2 , 15 , 12 , 30 , 25 }

2. Build a min-heap from the above data