Agenda:

- Introduction to Heaps and Binary Heaps
- Representing Binary Heaps using arrays
- Heapify operation
- Deletion
- Insertion
- K-th largest element
    - Sorting-based
    - Heap-based
- Merge K sorted arrays

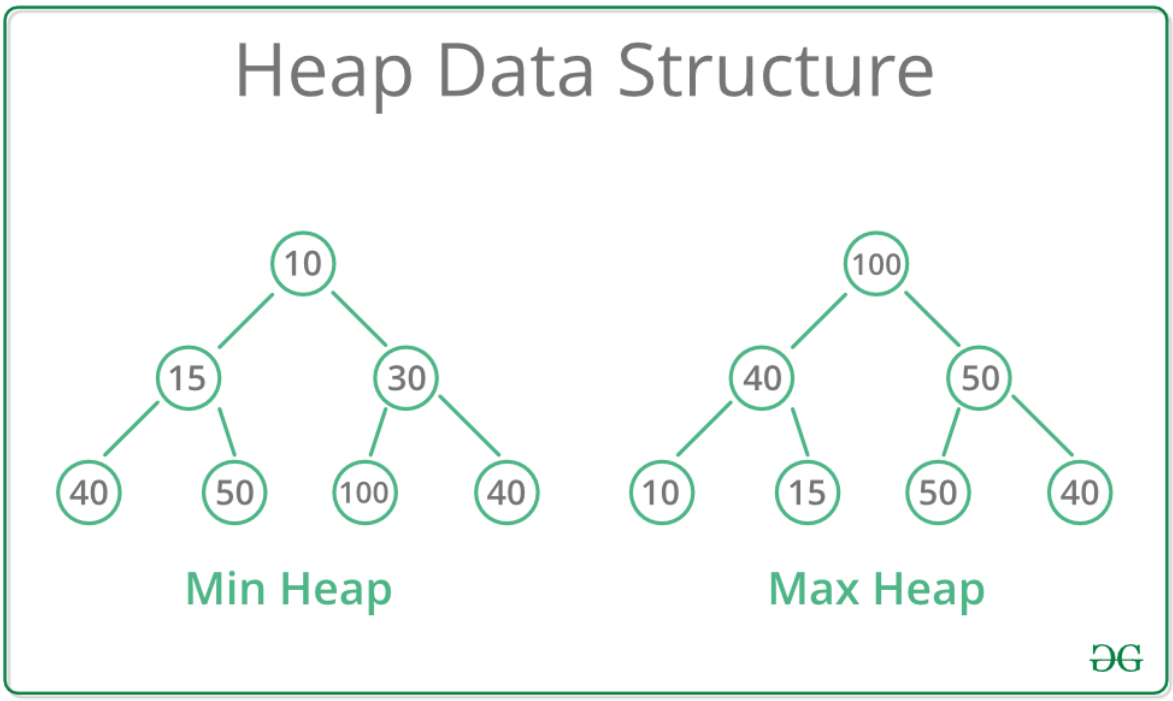
Introduction to Heaps and Binary Heaps

A Heap is a Tree-based data structure, which satisfies the below properties:

1. A Heap is a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible).

2. A Heap is either Min Heap or Max Heap. In a Min-Heap, the key at root must be minimum among all keys present in the Binary Heap. The same property must be recursively true for all nodes in the Tree. Max Heap is similar to MinHeap.

Binary Heap: A Binary Heap is a heap where each node can have at most two children. In other words, a Binary Heap is a complete Binary Tree satisfying the above-mentioned properties.

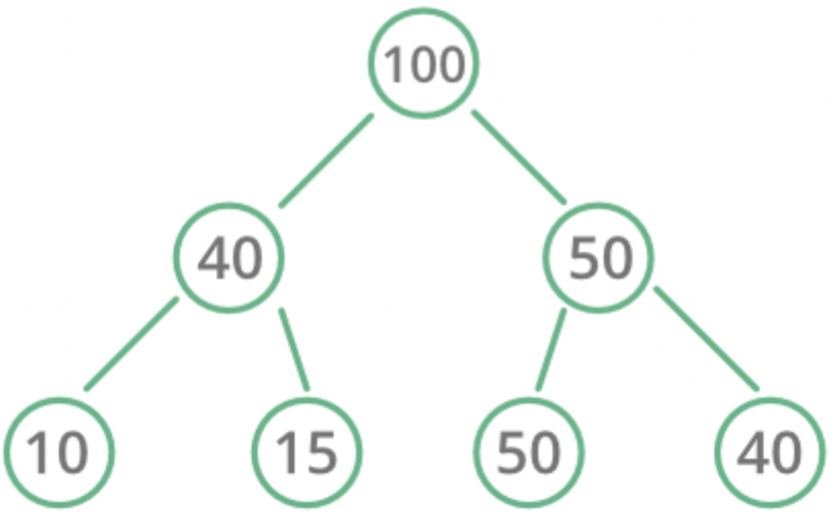


Representing Binary Heaps using Arrays

Since a Binary Heap is a complete Binary Tree, it can be easily represented using Arrays.

- The root element will be at Arr[0].
- Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node



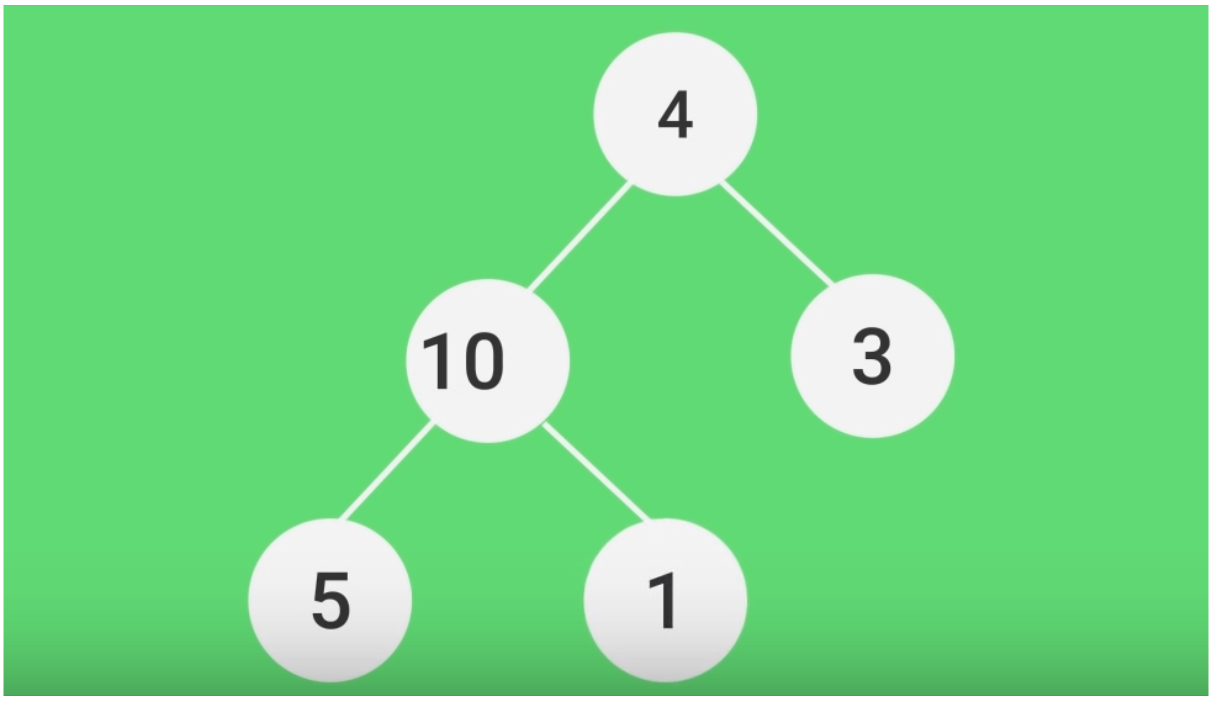
	0	**1**	**2**	**3**	**4**	**5**	**6**
Array:	[100	40	50	10	15	50	40]

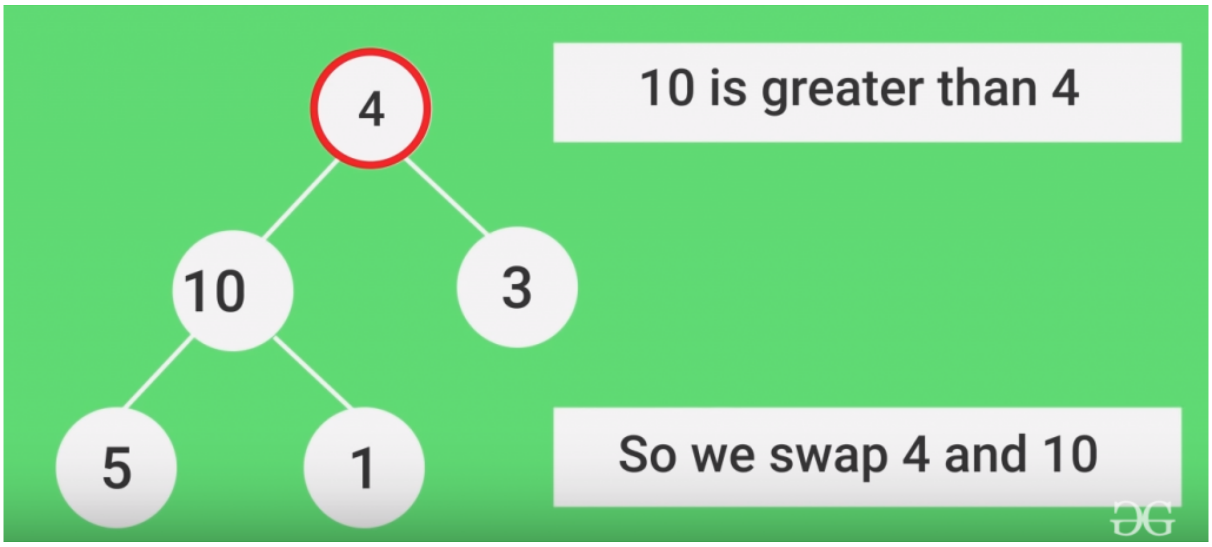
Index of Parent of 50(index 2):	(2 - 1) / 2 = 0
Index of left child of 50(index 2):	2*2 + 1 = 5
Index of right child of 50(index 2):	2*2 + 2 = 6

Heapify operation

Generally, on inserting a new element onto a Heap, it does not satisfy the property of Heap as stated above on its own. The process of placing the element at the correct location so that it satisfies the Heap property is known as Heapify.

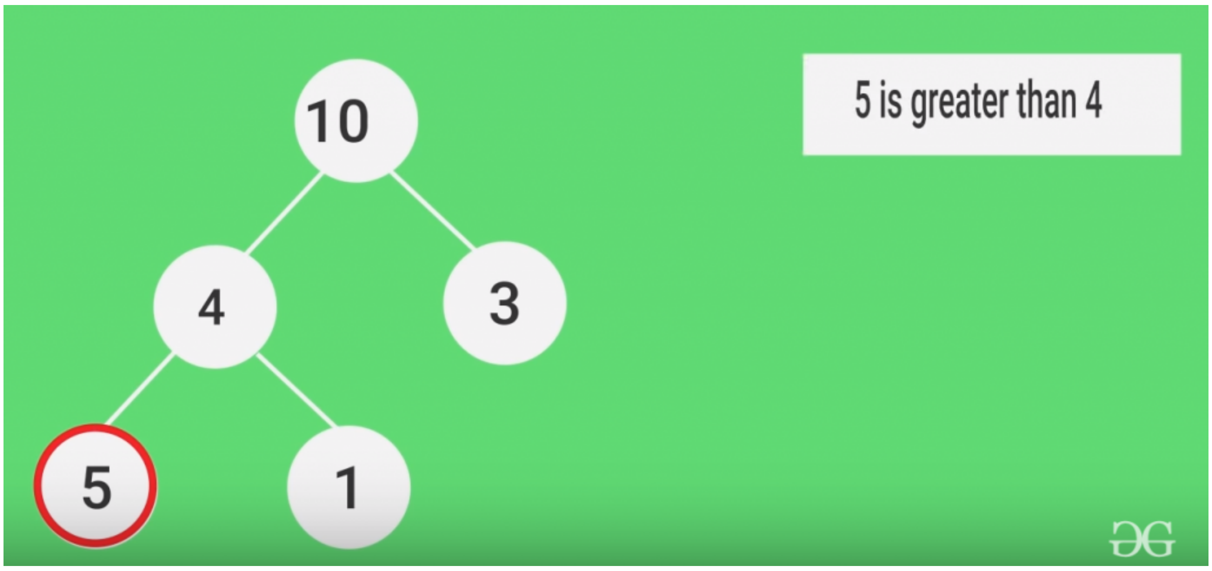
Heapifying in a Max Heap: The property of Max Heap says that every node's value must be greater than the values of its children nodes. So, to **heapify** a particular node swap the value of the node with the maximum value of its children nodes and continue the process until all of the nodes below it satisfies the Heap property.





10 is greater than 4

So we swap 4 and 10



5 is greater than 4

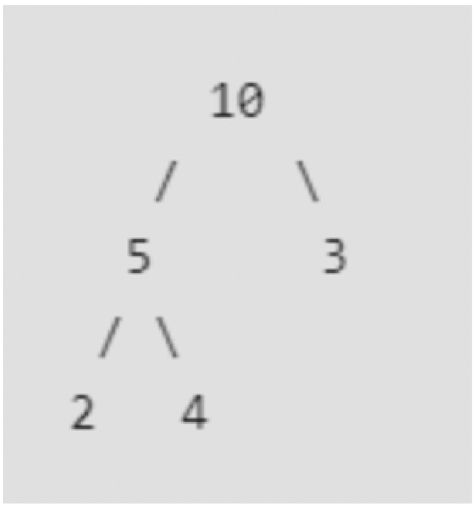
Deletion Operation:

Deletion is always done at the root node of a Heap. For example, deletion in a Max-heap means to delete the maximum value from the heap whereas deletion in a Min-heap means to delete the minimum value.

Steps:

1. Replace the value of root with the last node present in the array and delete the last element.
2. Heapify root.

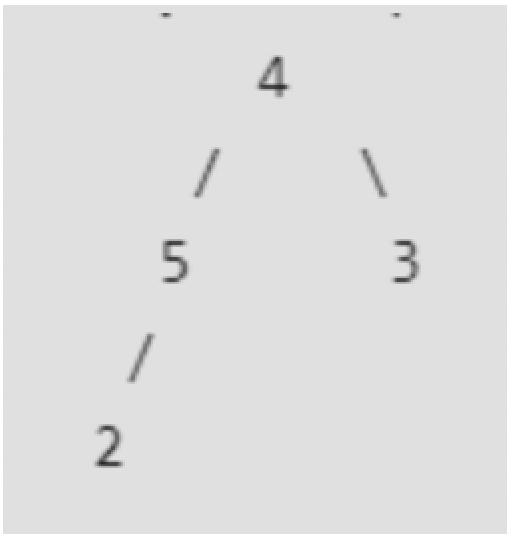
For example:



```
        10
      /    \
     5       3
    / \
   2   4
```

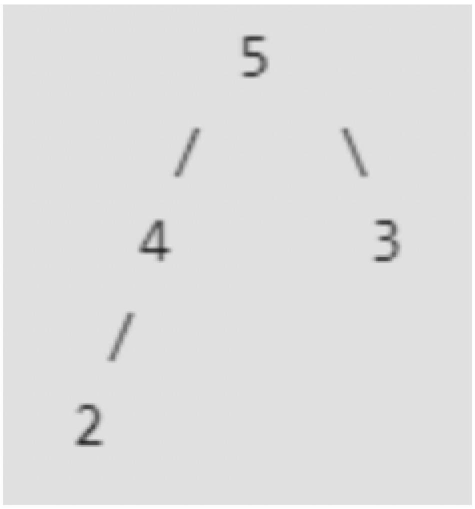
(We need to delete 10)

Step-1: Replace the last element with root, and delete it.



```
        4
      /   \
     5     3
    /
   2
```

Step-2: Heapify root



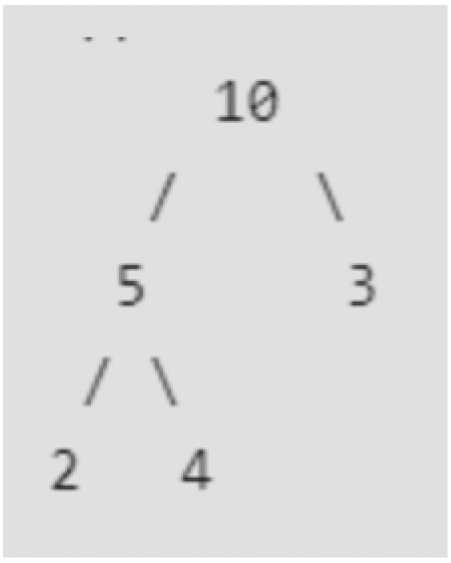
```
        5
      /   \
     4     3
    /
   2
```

Insertion Operation

Elements can be inserted to the heap following a similar approach as discussed above for deletion. The idea is to:

- First increase the heap size by 1, so that it can store the new element.
- Insert the new element at the end of the Heap.
- This newly inserted element may distort the properties of Heap for its parents. So, in order to keep the properties of Heap, heapify this newly inserted element following a bottom-up approach.

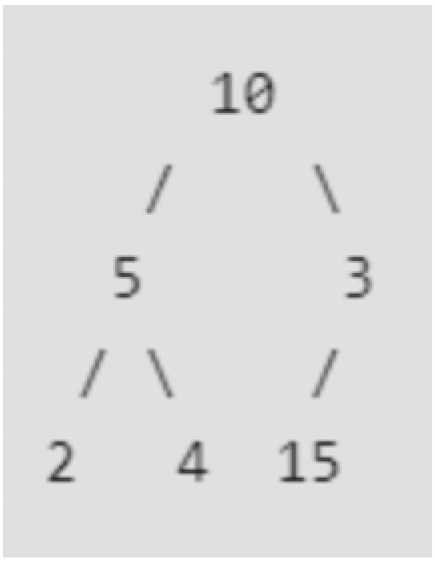
Example:



```
      10
    /    \
   5      3
  / \
 2   4
```

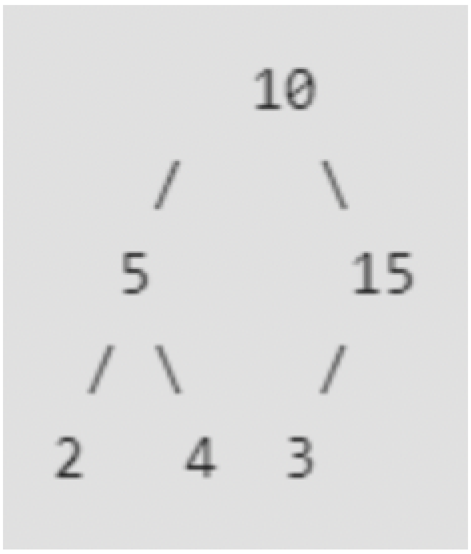
(Insert 15)

Step-1: Insert at the end

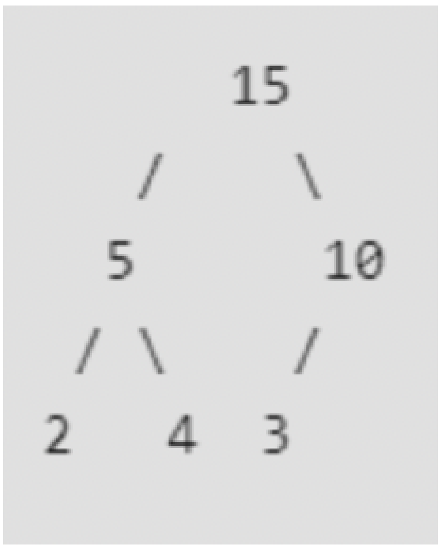


```
      10
    /    \
   5      3
  / \    /
 2   4  15
```

Step-2: Heapify

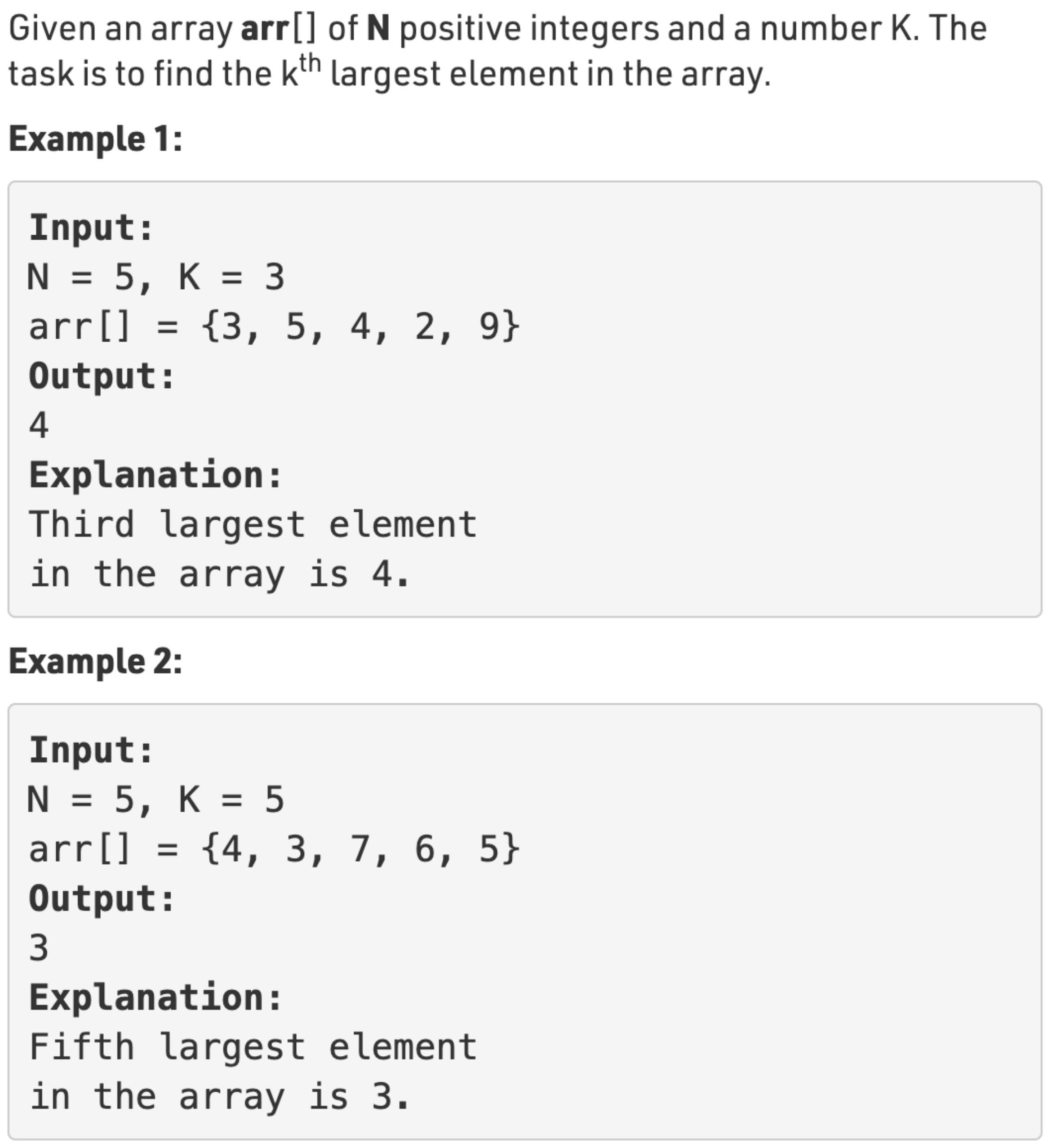


```
      10
    /    \
   5      15
  / \    /
 2   4  3
```



```
      15
    /    \
   5      10
  / \    /
 2   4  3
```

Kth Largest Element



Given an array **arr**[] of **N** positive integers and a number K. The task is to find the k^th largest element in the array.

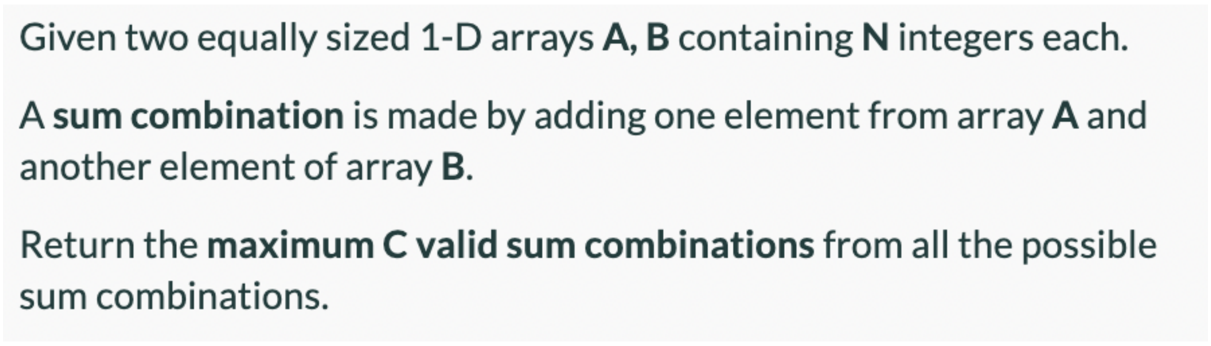
Example 1:

```
Input:
N = 5, K = 3
arr[] = {3, 5, 4, 2, 9}
Output:
4
Explanation:
Third largest element
in the array is 4.
```

Example 2:

```
Input:
N = 5, K = 5
arr[] = {4, 3, 7, 6, 5}
Output:
3
Explanation:
Fifth largest element
in the array is 3.
```

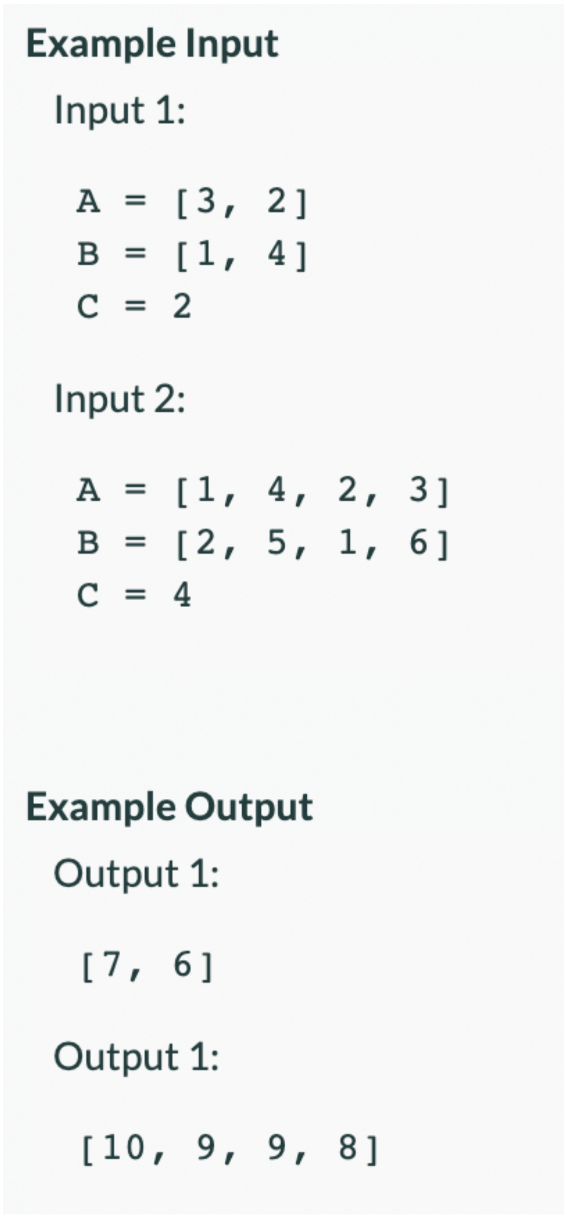
Maximum Sum Combination



Given two equally sized 1-D arrays **A, B** containing **N** integers each.

A **sum combination** is made by adding one element from array **A** and another element of array **B**.

Return the **maximum C valid sum combinations** from all the possible sum combinations.



Example Input

Input 1:

```
 A = [3, 2]
 B = [1, 4]
 C = 2
```

Input 2:

```
 A = [1, 4, 2, 3]
 B = [2, 5, 1, 6]
 C = 4
```

Example Output

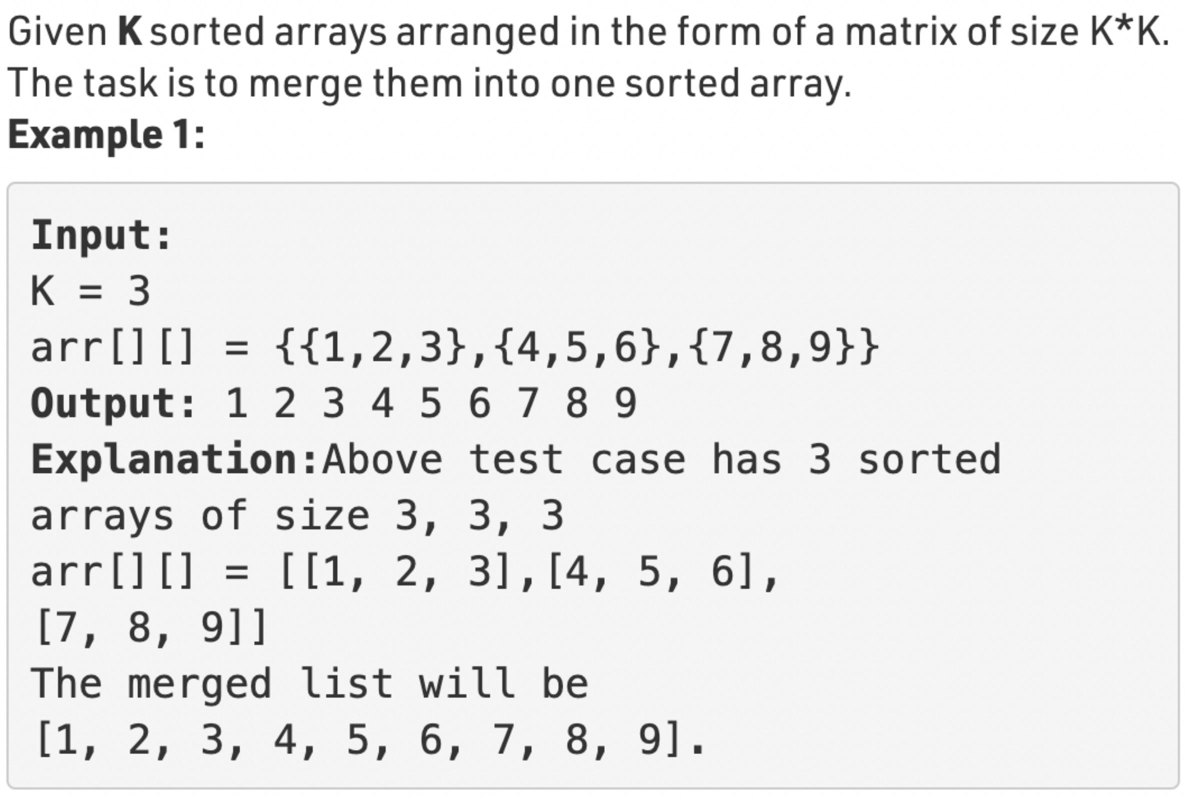
Output 1:

```
 [7, 6]
```

Output 1:

```
 [10, 9, 9, 8]
```

Merge K sorted arrays



Given **K** sorted arrays arranged in the form of a matrix of size K*K. The task is to merge them into one sorted array.

Example 1:

```
Input:
K = 3
arr[][] = {{1,2,3},{4,5,6},{7,8,9}}
Output: 1 2 3 4 5 6 7 8 9
Explanation:Above test case has 3 sorted
arrays of size 3, 3, 3
arr[][] = [[1, 2, 3],[4, 5, 6],
[7, 8, 9]]
The merged list will be
[1, 2, 3, 4, 5, 6, 7, 8, 9].
```