

# HEAP / PRIORITY QUEUE

# Heap

- **Heap** is a complete binary tree that satisfy the heap property (max or min)
- **Min heap**: root node contains the minimum value
- **Max heap**: root node contains the maximum value

Min Heap



Max Heap



# Heap in C++

## Two main ways to implement:

### 1. Using **std::make\_heap** from **<algorithm>**

```
std::make_heap(RandomIt first, RandomIt last)
```

```
std::push_heap(RandomIt first, RandomIt last)
```

```
std::pop_heap(RandomIt first, RandomIt last)
```

```
std::sort_heap(RandomIt first, RandomIt last)
```

### 2. Using **std::priority\_queue** from **<queue>** **(recommended)**

```
std::priority_queue<T, Container, Compare>
```

# Heap in C++ – `std::priority_queue` example

## Min heap

```
std::priority_queue<int, std::vector<int>, std::greater<int>>
```

## Max heap

```
std::priority_queue<int> or
```

```
std::priority_queue<int, std::vector<int> std::less<int>>
```

```
// Min heap
std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;

minHeap.push(3);
minHeap.push(6);
minHeap.push(4);
// remove top element (3)
minHeap.pop();
// root node (top) is now 4
std::cout << minHeap.top();
```

# Problem – Kth Largest Element in an Array

Medium

<https://leetcode.com/problems/kth-largest-element-in-an-array>

## Problem

Given an integer array `nums` and an integer `k`, return the  $k^{\text{th}}$  largest element in the array. Note that it is the  $k^{\text{th}}$  largest element in the sorted order, not the  $k^{\text{th}}$  distinct element.

## Example 1

Input: `nums = [3,2,1,5,6,4]`, `k = 2`

Output: 5

## Example 2

Input: `nums = [3,2,3,1,2,4,5,5,6]`, `k = 4`

Output: 4

Although this problem is classified as “medium”, in my opinion it should be classified as “easy”

# Solution 1 – Kth Largest Element in an Array

Medium

<https://leetcode.com/problems/kth-largest-element-in-an-array>

// SOLUTION 1

```
int findKthLargest(vector<int>& nums, int k) {  
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;  
    for (const auto& num : nums) {  
        if (minHeap.size() < k) {  
            minHeap.push(num);  
        } else if (num > minHeap.top()) {  
            minHeap.pop();  
            minHeap.push(num);  
        }  
    }  
    return minHeap.top();  
}
```

# Solution 2 – Kth Largest Element in an Array

Medium

<https://leetcode.com/problems/kth-largest-element-in-an-array>

```
// SOLUTION 2 - Simpler approach
```

```
int findKthLargest(vector<int>& nums, int k) {  
    // min heap: minimum values will be always at the top  
    std::priority_queue<int, std::vector<int>, std::greater<int>> minHeap;  
    for (const auto& num : nums) {  
        // push each num to the heap  
        minHeap.push(num);  
        // we need the kth largest element only, so once after pushing more than k  
        // elements, remove the smallest one (the top)  
        if (minHeap.size() > k) {  
            minHeap.pop();  
        }  
    }  
    return minHeap.top();  
}
```