Heap in C++

```
#include <iostream>
#include <vector>

using namespace std;

class MinHeap {
    vector<int> arr;
    int size;
    int capacity;

public:
    MinHeap(int c) {
        size = 0;
        capacity = c;
        arr.resize(c);
    }

    int left(int i) {
        return 2 * i + 1;
    }

    int right(int i) {
        return 2 * i + 2;
    }

    int parent(int i) {
        return (i - 1) / 2;
    }

    void show() {
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    void insert(int x) {
        if (size == capacity) {
            return;
        }
        size++;
        arr[size - 1] = x;
        int i = size - 1;
        while (i != 0 && arr[parent(i)] > arr[i]) {
            swap(arr[i], arr[parent(i)]);
            i = parent(i);
        }
    }
};

int main() {
    MinHeap h(9);
    h.insert(10);
    h.insert(20);
    h.insert(15);
    h.insert(40);
    h.insert(50);
    h.insert(100);
    h.insert(25);
    h.insert(45);
```

Initial Setup

- We create a `MinHeap` of capacity `9`.
- Insert sequence: `10, 20, 15, 40, 50, 100, 25, 45`

🧠 What is a MinHeap?

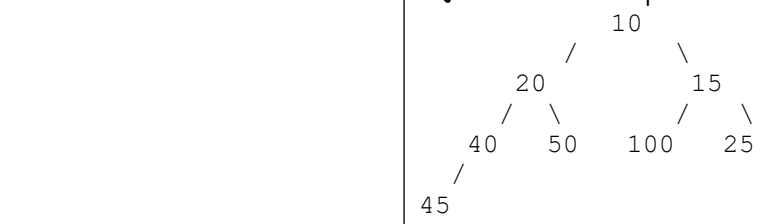
A MinHeap is a **complete binary tree** where:

- Every parent node is **less than or equal to** its children.
- Insertions maintain this structure by "bubbling up" the inserted value if it violates the heap property.

▼ Step-by-Step Dry Run Table

Step	Inserted Value	Heap Before Insert	Heap After Insert + Bubble Up
1	10	`[]`	`[10]`
2	20	`[10]`	`[10, 20]`
3	15	`[10, 20]`	`[10, 20, 15]`
4	40	`[10, 20, 15]`	`[10, 20, 15, 40]`
5	50	`[10, 20, 15, 40]`	`[10, 20, 15, 40, 50]`
6	100	`[10, 20, 15, 40, 50]`	`[10, 20, 15, 40, 50, 100]`
7	25	`[10, 20, 15, 40, 50, 100]`	`[10, 20, 15, 40, 50, 100, 25]`
8	45	`[10, 20, 15, 40, 50, 100, 25]`	`[10, 20, 15, 40, 50, 100, 25, 45]`

🔍 Final MinHeap Tree Representation:



- The heap property is maintained at each

h.show();
 	
return 0;
}
	step.
	• No bubbling up required beyond one level in most cases.
	
	✓ Output of `h.show();`
	`10 20 15 40 50 100 25 45`

10 20 15 40 50 100 25 45	

K largest elements in C++

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

void klargest(vector<int>& arr, int k) {
    priority_queue<int, vector<int>, greater<int>> pq;

    // Insert the first k elements into the min heap
    for (int i = 0; i < k; i++) {
        pq.push(arr[i]);
    }

    // For each element from k to end of array, check if
it's larger than the smallest in the heap
    for (int i = k; i < arr.size(); i++) {
        if (pq.top() < arr[i]) {
            pq.pop();
            pq.push(arr[i]);
        }
    }

    // Print the k largest elements
    cout << "K largest elements: ";
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
    cout << endl;
}

int main() {
    // Hardcoded input array
    vector<int> arr = {5, 15, 10, 20, 8, 25, 18};
    int k = 3;

    // Call the klargest function to find and print the k
largest elements
    klargest(arr, k);

    return 0;
}
```

Step-by-Step Dry Run

Step	i	Element	Min Heap Before	Action	Min Heap After
Init	-	-	[]	Start inserting first k=3	
1	0	5	[]	Push 5	[5]
2	1	15	[5]	Push 15	[5, 15]
3	2	10	[5, 15]	Push 10	[5, 15, 10]
4	3	20	[5, 15, 10]	20 > 5 → pop 5, push 20	[10, 15, 20]
5	4	8	[10, 15, 20]	8 < 10 → do nothing	[10, 15, 20]
6	5	25	[10, 15, 20]	25 > 10 → pop 10, push 25	[15, 20, 25]
7	6	18	[15, 20, 25]	18 > 15 → pop 15, push 18	[18, 25, 20]

✅ **Final Heap Contents: [18, 25, 20]**

This heap now contains the **top 3 largest elements**: **18, 25, 20**

🖨️ Output:
K largest elements: 18 20 25

K largest elements: 18 20 25

K sorted array in C++

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

void sortKSortedArray(vector<int>& arr, int k) {
    priority_queue<int, vector<int>, greater<int>> pq; // Min heap

    // Push the first k+1 elements into the priority queue
    for (int i = 0; i <= k; ++i) {
        pq.push(arr[i]);
    }

    int index = 0;

    // Process the remaining elements
    for (int i = k + 1; i < arr.size(); ++i) {
        arr[index++] = pq.top(); // Get the smallest element from the heap
        pq.pop(); // Remove the smallest element from the heap
        pq.push(arr[i]); // Push the current element into the heap
    }

    // Extract all remaining elements from the heap
    while (!pq.empty()) {
        arr[index++] = pq.top();
        pq.pop();
    }

    // Print sorted array
    for (int i = 0; i < arr.size(); ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    // Hardcoded input array
    vector<int> arr = {7, 8, 9, 19, 18};
    int k = 3;

    // Sort the k-sorted array
    sortKSortedArray(arr, k);

    return 0;
}
```

Input:

```
arr = {7, 8, 9, 19, 18}
k = 3
```

We will walk through it step-by-step in a table format showing the **min heap**, **index**, and how the array is being modified.

🟦 Initial Step – Insert first `k+1 = 4` elements into min-heap:

Step	Action	Min Heap	arr[]	index
0	Insert first 4 elements (0–3)	[7, 8, 9, 19]	[7, 8, 9, 19, 18]	—

🔁 Main Loop (from i = k+1 to end):

Step	i	Action	Min Heap Before	Element Pushed	Popped → arr[index]	Min Heap After	arr[]	index
1	4	Push 18, Pop & insert 7	[7, 8, 9, 19]	18	7	[8, 18, 9, 19]	[7, 8, 9, 19, 18]	0
2	—	Pop & insert 8	[8, 18, 9, 19]	—	8	[9, 18, 19]	[7, 8, 9, 19, 18]	1
3	—	Pop & insert 9	[9, 18, 19]	—	9	[18, 19]	[7, 8, 9, 19, 18]	2
4	—	Pop & insert 18	[18, 19]	—	18	[19]	[7, 8, 9, 18, 18]	3
5	—	Pop & insert 19	[19]	—	19	[]	[7, 8, 9, 18, 19]	4

	✅ Final Output: 7 8 9 18 19
7 8 9 18 19	