Heap in C++

```cpp
#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:

```
               10
            /       \
        20             15
       /  \          /    \
    40     50     100    25
   /
45
```

- 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	