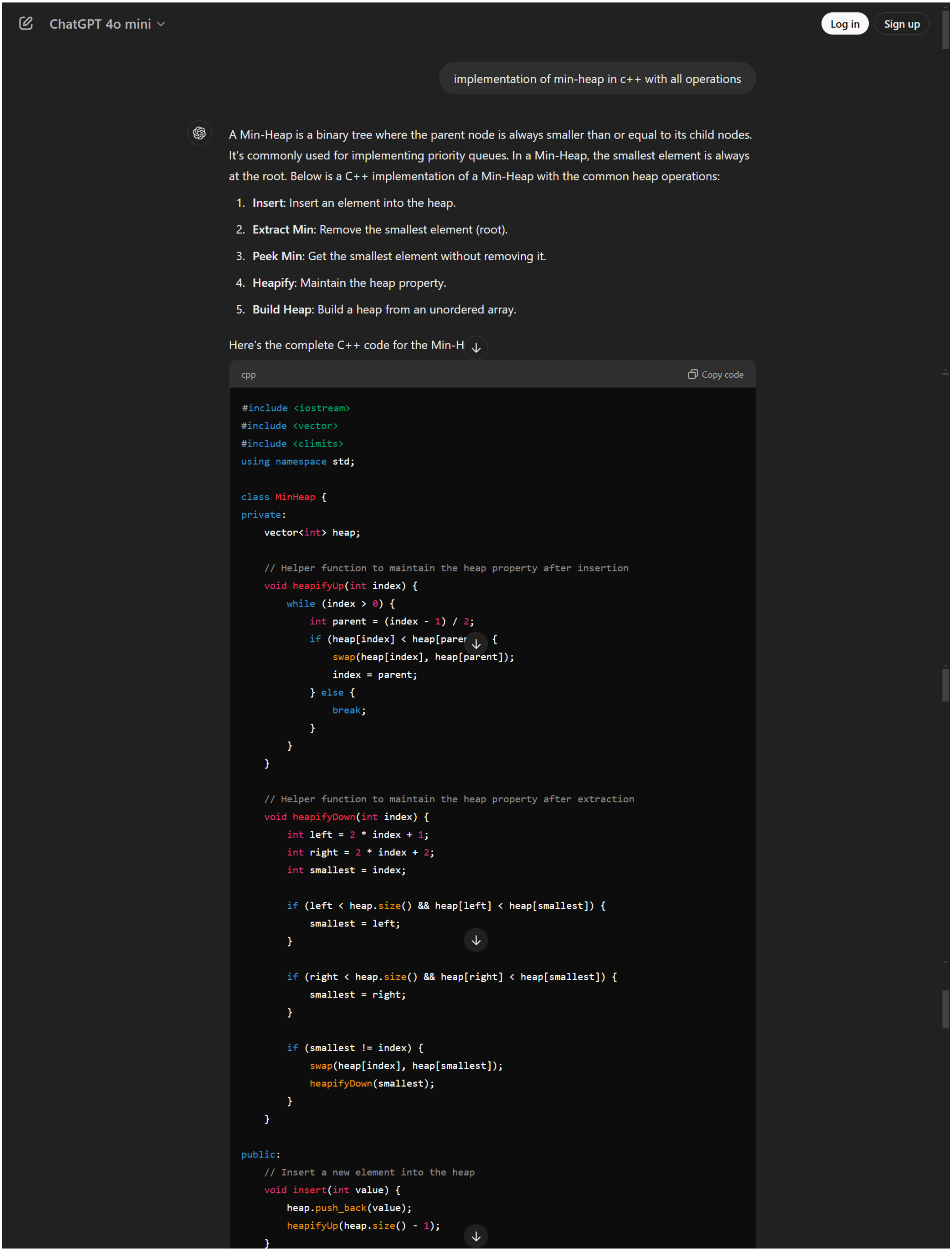


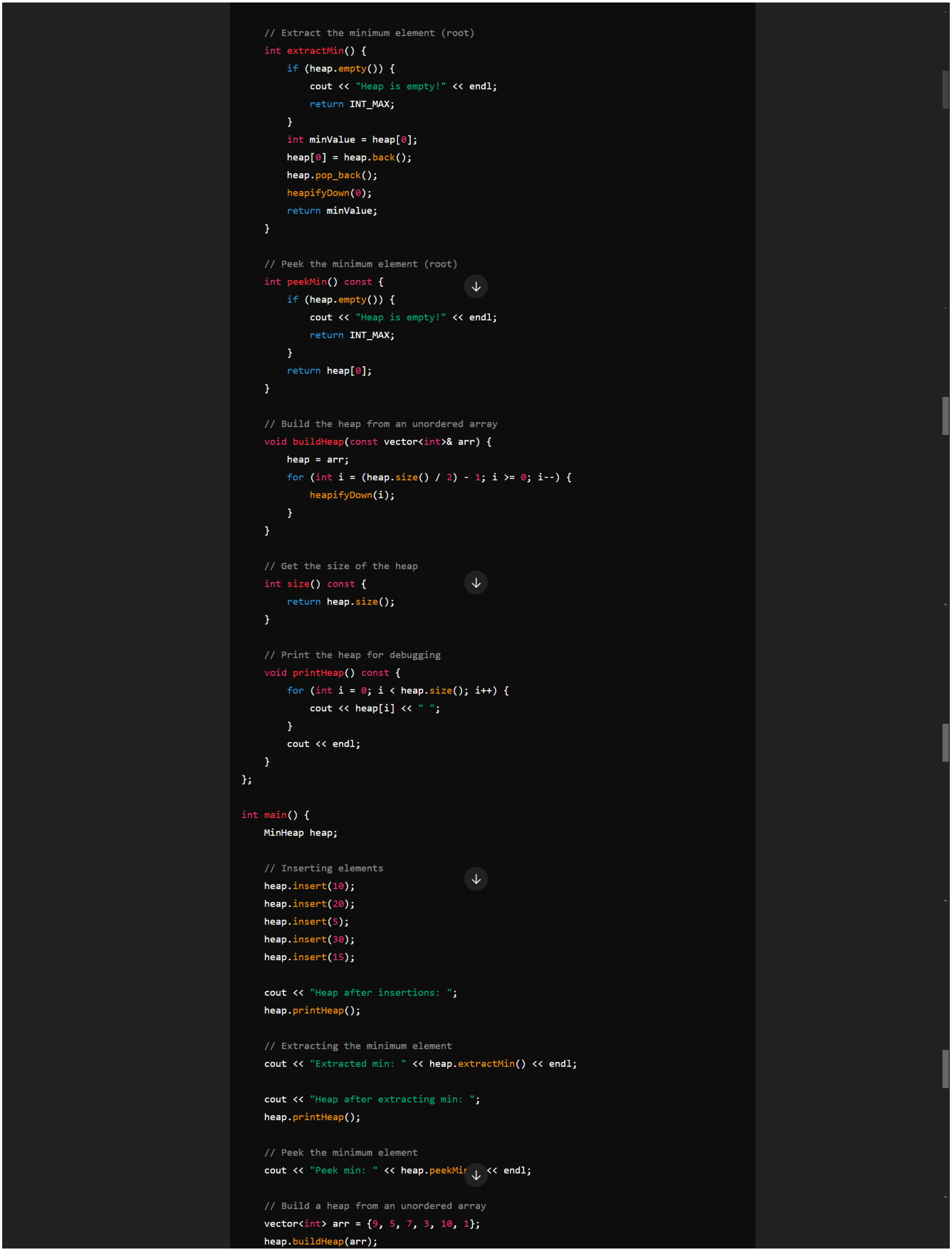
implementation of min-heap in c++ with all operations

A Min-Heap is a binary tree where the parent node is always smaller than or equal to its child nodes. It's commonly used for implementing priority queues. In a Min-Heap, the smallest element is always at the root. Below is a C++ implementation of a Min-Heap with the common heap operations:

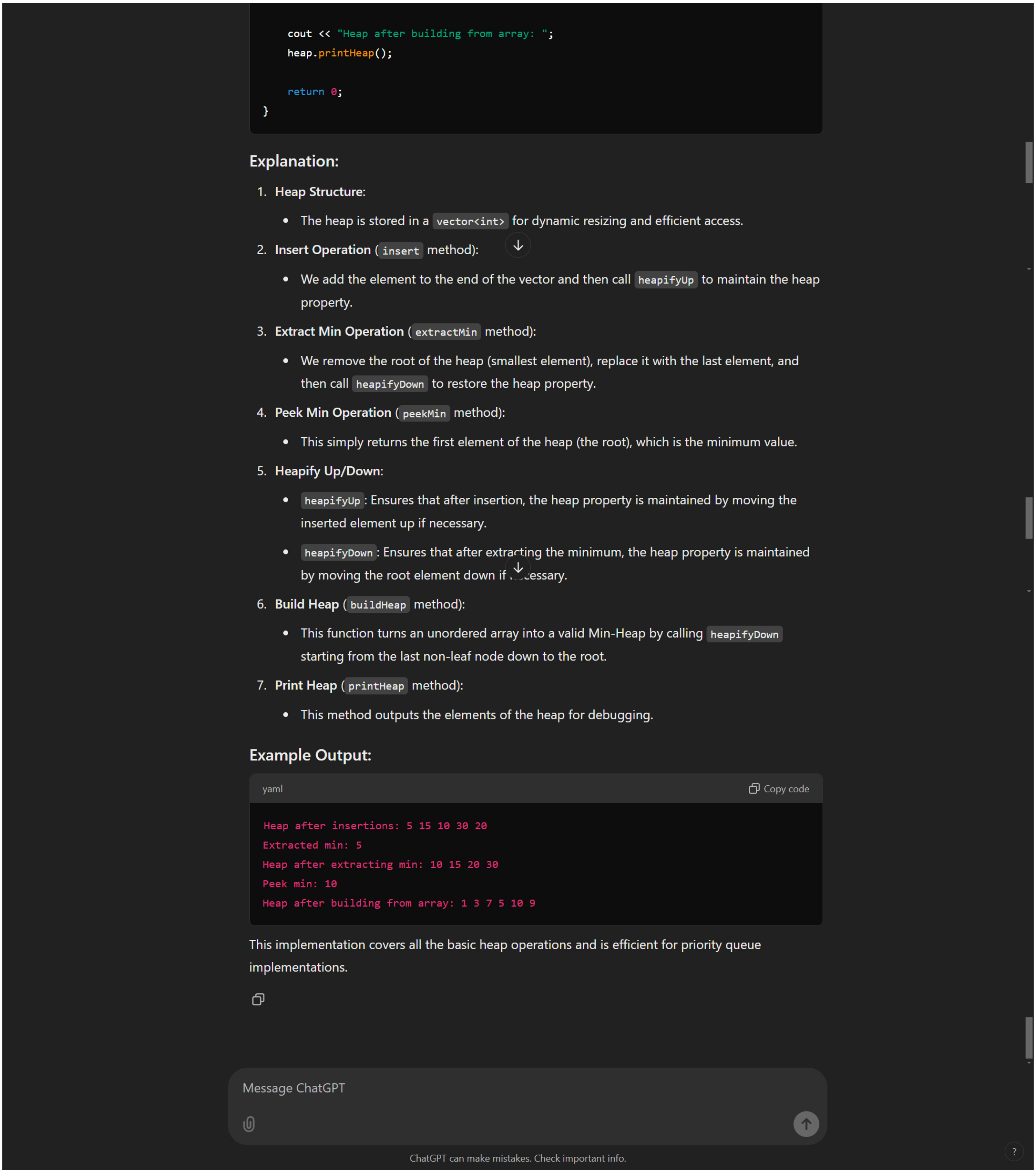
1. **Insert**: Insert an element into the heap.
2. **Extract Min**: Remove the smallest element (root).
3. **Peek Min**: Get the smallest element without removing it.
4. **Heapify**: Maintain the heap property.
5. **Build Heap**: Build a heap from an unordered array.

Here's the complete C++ code for the Min-H

```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

class MinHeap {
private:
    vector<int> heap;

    // Helper function to maintain the heap property after insertion
    void heapifyUp(int index) {
        while (index > 0) {
            int parent = (index - 1) / 2;
            if (heap[index] < heap[paren   {
                swap(heap[index], heap[parent]);
                index = parent;
            } else {
                break;
            }
        }
    }

    // Helper function to maintain the heap property after extraction
    void heapifyDown(int index) {
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int smallest = index;

        if (left < heap.size() && heap[left] < heap[smallest]) {
            smallest = left;
        }

        if (right < heap.size() && heap[right] < heap[smallest]) {
            smallest = right;
        }

        if (smallest != index) {
            swap(heap[index], heap[smallest]);
            heapifyDown(smallest);
        }
    }

public:
    // Insert a new element into the heap
    void insert(int value) {
        heap.push_back(value);
        heapifyUp(heap.size() - 1);
    }
```



```
    // Extract the minimum element (root)
    int extractMin() {
        if (heap.empty()) {
            cout << "Heap is empty!" << endl;
            return INT_MAX;
        }
        int minValue = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);
        return minValue;
    }

    // Peek the minimum element (root)
    int peekMin() const {
        if (heap.empty()) {
            cout << "Heap is empty!" << endl;
            return INT_MAX;
        }
        return heap[0];
    }

    // Build the heap from an unordered array
    void buildHeap(const vector<int>& arr) {
        heap = arr;
        for (int i = (heap.size() / 2) - 1; i >= 0; i--) {
            heapifyDown(i);
        }
    }

    // Get the size of the heap
    int size() const {
        return heap.size();
    }

    // Print the heap for debugging
    void printHeap() const {
        for (int i = 0; i < heap.size(); i++) {
            cout << heap[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    MinHeap heap;

    // Inserting elements
    heap.insert(10);
    heap.insert(20);
    heap.insert(5);
    heap.insert(30);
    heap.insert(15);

    cout << "Heap after insertions: ";
    heap.printHeap();

    // Extracting the minimum element
    cout << "Extracted min: " << heap.extractMin() << endl;

    cout << "Heap after extracting min: ";
    heap.printHeap();

    // Peek the minimum element
    cout << "Peek min: " << heap.peekMi    << endl;

    // Build a heap from an unordered array
    vector<int> arr = {9, 5, 7, 3, 10, 1};
    heap.buildHeap(arr);
```



```
    cout << "Heap after building from array: ";
    heap.printHeap();

    return 0;
}
```

Explanation:

1. **Heap Structure**:
   - The heap is stored in a `vector<int>` for dynamic resizing and efficient access.
2. **Insert Operation** (`insert` method):
   - We add the element to the end of the vector and then call `heapifyUp` to maintain the heap property.
3. **Extract Min Operation** (`extractMin` method):
   - We remove the root of the heap (smallest element), replace it with the last element, and then call `heapifyDown` to restore the heap property.
4. **Peek Min Operation** (`peekMin` method):
   - This simply returns the first element of the heap (the root), which is the minimum value.
5. **Heapify Up/Down**:
   - `heapifyUp`: Ensures that after insertion, the heap property is maintained by moving the inserted element up if necessary.
   - `heapifyDown`: Ensures that after extracting the minimum, the heap property is maintained by moving the root element down if necessary.
6. **Build Heap** (`buildHeap` method):
   - This function turns an unordered array into a valid Min-Heap by calling `heapifyDown` starting from the last non-leaf node down to the root.
7. **Print Heap** (`printHeap` method):
   - This method outputs the elements of the heap for debugging.

Example Output:

```yaml
Heap after insertions: 5 15 10 30 20
Extracted min: 5
Heap after extracting min: 10 15 20 30
Peek min: 10
Heap after building from array: 1 3 7 5 10 9
```

This implementation covers all the basic heap operations and is efficient for priority queue implementations.

ChatGPT can make mistakes. Check important info.