Priority Queue in C++

What is a Priority Queue?

A **priority queue** is a special type of queue where each element has a **priority**. Instead of using **FIFO (First-In-First-Out)** like a normal queue, a priority queue **dequeues the element with the highest priority first**.

Key Properties of Priority Queues:

1. **Elements are ordered based on priority.**
2. **Higher priority elements are dequeued before lower priority elements.**
3. **Can be implemented using different data structures:**
    - **Heap (Binary Heap, Fibonacci Heap, etc.)** → Most common.
    - **Balanced BST (Red-Black Tree, AVL Tree, etc.)** → Rarely used.
    - **Unordered/Ordered Arrays or Linked Lists** → Less efficient.

Priority Queue in C++ (Using priority_queue)

C++ provides a **priority_queue** in the <queue> library, which is **implemented using a Max Heap** by default.

Basic Usage

```
#include <iostream>
#include <queue>  // Priority queue is in <queue>
using namespace std;

int main() {
  priority_queue<int> pq;  // Max Heap (default behavior)

  pq.push(10);
  pq.push(30);
  pq.push(20);
  pq.push(5);

  cout << "Top element (Highest Priority): " << pq.top() << endl; // 30
```

```
    pq.pop();  // Removes 30
    cout << "Next highest priority: " << pq.top() << endl; // 20

    return 0;
}
```

Output

Top element (Highest Priority): 30

Next highest priority: 20

Min Heap (Custom Priority)

By default, priority_queue in C++ is a **Max Heap** (higher numbers have higher priority). If you want a **Min Heap**, you have to use greater<int>.

```
priority_queue<int, vector<int>, greater<int>> minHeap;
```

Example of Min Heap

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

int main() {
    priority_queue<int, vector<int>, greater<int>> pq;  // Min Heap

    pq.push(10);
    pq.push(30);
    pq.push(20);
    pq.push(5);

    cout << "Top element (Lowest Value has Highest Priority): " << pq.top() << endl; // 5

    pq.pop();  // Removes 5
    cout << "Next priority: " << pq.top() << endl; // 10
```

```
    return 0;
}
```

Output

```
Top element (Lowest Value has Highest Priority): 5
Next priority: 10
```

Custom Comparator in Priority Queue

Sometimes, you may want to set custom rules for priority.

Example: Custom Comparator for Struct

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

struct Task {
    int id;
    int priority;

    Task(int i, int p) : id(i), priority(p) {}

    // Custom comparator: Higher priority value comes first
    bool operator<(const Task& other) const {
        return priority < other.priority;
    }
};

int main() {
    priority_queue<Task> pq;

    pq.push(Task(1, 3));  // Task ID 1, Priority 3
    pq.push(Task(2, 5));  // Task ID 2, Priority 5
    pq.push(Task(3, 2));  // Task ID 3, Priority 2
```

```
    cout << "Highest priority task ID: " << pq.top().id << endl; // Task 2 (priority 5)


    return 0;
}
```

Output

Highest priority task ID: 2

Complexity Analysis

Operation	Complexity
push()	**O(log N)** (heap insertion)
pop()	**O(log N)** (heap deletion)
top()	**O(1)** (peek top element)

Use Cases of Priority Queues

- **Dijkstra's Algorithm (Shortest Path)**
- **Huffman Encoding (Compression Algorithms)**
- **Task Scheduling (Operating Systems)**
- *A Algorithm (Pathfinding in AI)**
- **Load Balancing in Networks**

Final Thoughts

- **Use priority_queue<int> when you want a max heap (default).**
- **Use greater<int> to create a min heap.**
- **For complex objects, define a custom comparator.**
- **Priority queues are best for scenarios where the highest priority item must be processed first.**
