Priority Queue

Created by Chandra Prakash

Structure:

```
priority_queue<datatype, container, comparator> pq
```

- datatype: Type of data stored in the queue (e.g., `int` , `pair<int,int>` , `string`).
- container: The underlying container used for storage (usually `vector`).
- comparator: Defines the rule for prioritizing elements (e.g., `less` , `greater`).

Behavior and Access

- `.push(val)` → Insert element
- `.top()` → Get highest-priority (according to comparator)
- `.pop()` → Remove top
- `.empty()` , `.size()` → Queue state

Performance

- Internally uses a **binary heap**
- Operations:
  - `push()` → `O(log n)`
  - `pop()` → `O(log n)`
  - `top()` → `O(1)`

By default:

```
priority_queue<int> pq;
// is same as:
priority_queue<int, vector<int>, less<int>> pq;
```

1. `datatype` – What you're storing

Common Examples:

- `int`
- `pair<int, int>`
- `string`
- `custom struct` → For full control using custom comparators

2. `container` – Where values are stored

Supported Containers:

- `vector<T>` (default, efficient heap)
- `deque<T>` (alternative; rarely needed)
- `list<T>` or `set<T>` → Not supported

Examples:

```
priority_queue<int, vector<int>> pq;
```

Using deque:

```
priority_queue<int, deque<int>> pq;
```

Output & behavior will remain like default `vector<int>` usage.

3. `comparator` – How values are prioritized

This determines the **heap behavior**:

Comparator	Heap Type	Top Element
`less<T>` (default)	Max-heap	Largest
`greater<T>`	Min-heap	Smallest
Custom Struct	Custom	User-defined

Built-in Comparators

Max-Heap (default)

```
priority_queue<int> pq;
```

Top: Largest element.

Min-Heap

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

Top: Smallest element.

Min-Heap with pairs

```
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
```

Custom Comparator Struct Example

Sort by the second value of a pair (min-heap):

```
struct CompareSecond {
    bool operator()(pair<int, int> a, pair<int, int> b) {
        return a.second > b.second; // smaller .second gets higher priority
    }
};

priority_queue<pair<int,int>, vector<pair<int,int>>, CompareSecond> pq;
```

Lambda Comparator (C++14+)

Same custom behavior with lambda:

```
auto cmp = [](pair<int,int> a, pair<int,int> b) {
    return a.second > b.second;
};

priority_queue<pair<int,int>, vector<pair<int,int>>, decltype(cmp)> pq(cmp);
```

Integer-based Priority Queues

1. Max-heap

```
priority_queue<int> pq;
```

- **Sorting Rule:** Largest integer comes first (default `less<int>`)

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<int> pq;
    pq.push(10); pq.push(5); pq.push(20);
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
}
```

Output: `20 10 5`

2. Min-heap

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

- **Sorting Rule:** Smallest integer comes first (`greater<int>`)

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<int, vector<int>, greater<>> pq;
    pq.push(10); pq.push(5); pq.push(20);
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
}
```

Output: `5 10 20`

Note:

We must write the container `vector<int>` while using the min heap `priority_queue<int, vector<int>, greater<>>`

Reason:

- `priority_queue` has 3 template parameters:

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

- If you want to change the **comparator** (e.g., use `greater<>` for min-heap), you **must also specify the container** (`vector<int>`), even if it's the default.

If you skip the container and write like `priority_queue<int, greater<>> pq;` , The compiler treats `greater<>` as the container type, causing a type mismatch error.

3. Max-heap using deque

```
priority_queue<int, deque<int>> pq;
```

- **Sorting Rule:** Same as default, just uses `deque` instead of `vector`

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<int, deque<int>> pq;
    pq.push(15); pq.push(3); pq.push(8);
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
}
```

Output: `15 8 3`

Pair-based Priority Queues

1. Max-heap of pairs

```
priority_queue<pair<int, int>> pq;
```

- **Sorting Rule:** Highest `first`, then `second` in descending

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<pair<int, int>> pq;
    pq.push({1, 5}); pq.push({3, 4}); pq.push({3, 7});
    while (!pq.empty()) {
        cout << "(" << pq.top().first << "," << pq.top().second << ") ";
        pq.pop();
    }
}
```

Output: `(3,7) (3,4) (1,5)`

2. Min-heap of pairs

```
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
```

- **Sorting Rule:** Smallest `first`, then `second` in ascending

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    pq.push({1, 5}); pq.push({3, 4}); pq.push({3, 2});
    while (!pq.empty()) {
        cout << "(" << pq.top().first << "," << pq.top().second << ") ";
        pq.pop();
    }
}
```

Output: `(1,5) (3,2) (3,4)`

String-based Priority Queues

1. Max-heap

```
priority_queue<string> pq;
```

- **Sorting Rule:** Lexicographically largest string first

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<string> pq;
    pq.push("apple"); pq.push("banana"); pq.push("grape");
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
}
```

Output: `grape banana apple`

2. Min-heap

```
priority_queue<string, vector<string>, greater<>> pq;
```

Note: vector<string> in the container.

- **Sorting Rule:** Lexicographically smallest string first

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    priority_queue<string, vector<string>, greater<>> pq;
    pq.push("apple"); pq.push("banana"); pq.push("grape");
    while (!pq.empty()) {
        cout << pq.top() << " ";
        pq.pop();
    }
}
```

Output: `apple banana grape`

4. Custom Comparator (Struct-based)

Custom Sorting Rule

Example: Min-heap of `pair<int,int>` by second value

```
#include <bits/stdc++.h>
using namespace std;

struct CompareSecond {
    bool operator()(pair<int,int> a, pair<int,int> b) {
        return a.second > b.second; // min-heap by .second
    }
};
```

```
int main() {
    priority_queue<pair<int,int>, vector<pair<int,int>>, CompareSecond> pq;
    pq.push({1, 30}); pq.push({2, 10}); pq.push({3, 20});
    while (!pq.empty()) {
        cout << "(" << pq.top().first << "," << pq.top().second << ") ";
        pq.pop();
    }
}
```

Output: `(2,10) (3,20) (1,30)`

THE END