Priority Queue

STL Priority Queue:

The STL priority queue is an implementation of a priority queue data structure in C++. It is part of the Standard Template Library (STL) and provides a simple and efficient way to manage elements with priorities. The priority queue automatically arranges its elements based on their priorities, allowing for efficient retrieval of the element with the highest priority.

Key Features:

1. **Priority-based ordering:** The priority queue automatically orders its elements based on their priorities. The element with the highest priority is always at the top of the queue.

2. **Efficient operations:** The priority queue provides efficient operations for insertion, deletion, and retrieval of elements. Insertion and deletion take logarithmic time complexity O(log n), while retrieval of the top element takes constant time complexity O(1).

3. **Container adaptability:** The underlying container used by the priority queue is customizable. By default, it uses the **std::vector** container, but you can also use other containers like **std::deque** or **std::list** by specifying them as template parameters.

4. **Custom comparators:** The priority queue allows you to define custom comparators to determine the ordering of elements. By default, it uses **std::less** to order elements in ascending order, but you can provide your own comparison function or lambda expression to specify custom ordering.

5. **No random access:** The priority queue does not support random access to its elements. You can only access the element with the highest priority (top element) using the **top()** member function.

Max Heap: `priority_queue<int> maxHeap;`

In contrast, a max heap is structured such that for every node i, the value of i is greater than or equal to the values of its children. This ensures that the maximum element is always located at the root of the heap. Max heaps are useful for quickly accessing the maximum element in a collection.

Max heaps have applications in scenarios such as:

- Implementing priority queues where the highest priority corresponds to the largest value.
- Efficiently finding the k largest (or smallest) elements in a collection.
- Implementing the Heap Sort algorithm.

Example: [50, 30, 20, 15, 10, 8, 16]

```
        50

      /    \

    30       20

   /  \     /  \

  15    10  8    16
```

Min Heap: `priority_queue<int, std::vector<int>, std::greater<int>> minHeap;`

In a min heap, for every node **i**, the value of **i** is less than or equal to the values of its children. This means that the minimum element is always located at the root of the heap. The values in the heap are arranged in a way that allows for efficient retrieval of the minimum element.

Min heaps have various applications, including:

- Implementing priority queues where the highest priority corresponds to the smallest value.
- Efficiently finding the k smallest (or largest) elements in a collection.
- Implementing algorithms like Dijkstra's algorithm and Prim's algorithm

Example: [5, 10, 15, 20, 30, 40, 50]

```
        5

      /    \

    10       15

   /  \     /  \

  20    30  40    50
```

Usage of Priority Queue:

1. **Dijkstra's Algorithm:** The priority queue is often used in the implementation of Dijkstra's algorithm for finding the shortest path in a graph. The priority queue is used to efficiently select the next vertex with the minimum distance during the algorithm's execution.
2. **Job Scheduling:** In job scheduling problems, where tasks or jobs have different priorities or deadlines, a priority queue can be used to efficiently schedule and process the jobs based on their priorities or deadlines.

3. **Event-driven Simulations:** Priority queues are widely used in event-driven simulations, where events are scheduled to occur at different times. The priority queue can be used to order the events based on their scheduled time, allowing for efficient simulation execution.
4. **Huffman Coding:** In data compression algorithms like Huffman coding, a priority queue can be used to build an optimal prefix code. The elements in the priority queue represent the frequencies of characters, and the priority queue is used to merge the least frequent elements repeatedly until a Huffman tree is constructed.
5. **Task Prioritization:** In task management systems, a priority queue can be used to manage and prioritize tasks based on their urgency or importance. Tasks with higher priority are processed or assigned first.
6. **Event-driven Systems:** In event-driven systems, such as real-time systems or message processing systems, a priority queue can be used to handle incoming events based on their priorities. Higher priority events are processed before lower priority events.
7. **Data Stream Processing:** Priority queues can be used in scenarios where real-time data streams need to be processed, and the most important or relevant data needs to be handled first. Examples include processing network packets, sensor data, or financial market data.
8. **Scheduling Algorithms:** Priority queues are employed in various scheduling algorithms, such as process scheduling in operating systems or task scheduling in parallel computing systems. The priority queue helps determine the order in which processes or tasks are executed based on their priorities.

- **push()**: Inserts an element into the priority queue.
    - Time Complexity: O(log N)
    - Space Complexity: O(1)
- **pop()**: Removes the element with the highest priority from the priority queue.
    - Time Complexity: O(log N)
    - Space Complexity: O(1)
- **top()**: Accesses the element with the highest priority in the priority queue.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
- **size()**: Returns the number of elements in the priority queue.
    - Time Complexity: O(1)
    - Space Complexity: O(1)

- **empty()**: Checks if the priority queue is empty.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
- **emplace()**: Constructs and inserts an element into the priority queue in-place.
    - Time Complexity: O(log N)
    - Space Complexity: O(1)
- **swap()**: Swaps the contents of two priority queues.
    - Time Complexity: O(1)
    - Space Complexity: O(1)