**1. Similarities:**

The name and the work of the functions of the Priority Queue are the same as that of Queue; add(), remove(), and peek().

add() function add an element to the Priority Queue.

remove() function removes an element from the Priority Queue.

peek() function returns the value of an element of the Priority Queue.

**2. Differences:**

When an element is removed from the Priority Queue, a certain "Priority" is followed. We can set this priority according to our needs.

We have two options to do so; we can either set it to rank order i.e. number with highest rank (lowest in value) or to score order i.e. number with highest score (greatest in value).

By default this priority is set to rank order. That means if you call remove() function then the number with lowest value will be removed from the Priority Queue.

Same is the case for the peek() function. It also returns the value of the element with highest priority.

Unlike Queue, the add() and the remove() functions in the Priority Queue have a time complexity of O(log n).

You should watch this part of the lecture video for more clarity.

**Let's take an example:**

Suppose we added 10, 80, 9, 33 and 2 in a Priority Queue, pq.

* And we call the peek() function.

**What do you think it should and have returned?**

**Yes, Correct! 2. As it is the highest priority value in terms of rankings.**

And on calling remove() function. 2 gets removed from the pq.

Now peek() will return 9 and remove() will remove 9 from pq.

Now peek() will return 10 and remove() will remove 10 from pq.

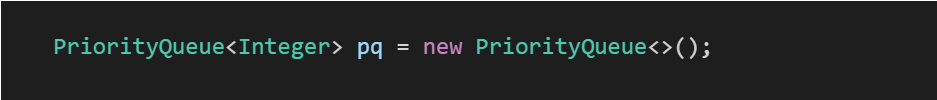
Now peek() will return 33 and remove() will remove 33 from pq.

Now peek() will return 80 and remove() will remove 80 from pq.

**How to use "Priority Queue" in java?**

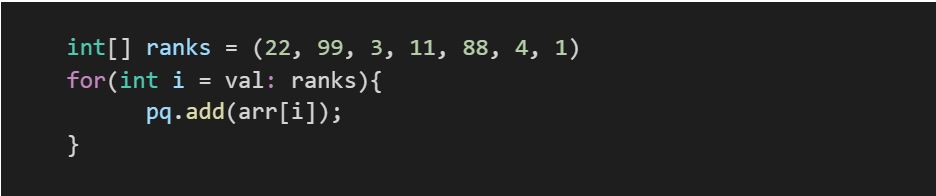
First of all we need to define a Priority Queue. Let's define a Priority Queue, say pq, of type Integer.

By default pq's priority will be set to rank wise order, that is it will have more priority for smaller values.



Now consider an array, ranks = {22, 99, 3, 11, 88, 4, 1}.

Now we will add all the values of array ranks in the Priority Queue, pq using a for loop and function add(value).



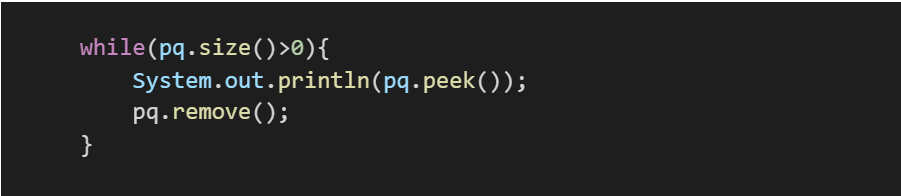
**What do you think is the time complexity of the above for loop?**

To remind you, the time complexity of the add(value) function is O(log n).

And since the for loop runs n times, therefore time complexity will be n\*O(log n) or O(n log n)

Moving further, to use peek() and remove() function, we run a while loop until the pq empties out.

And first print the peek element by calling peek() function and then call remove() function to remove element.



**What do you think is the time complexity of the above while loop?**

To remind you, the time complexity of remove() function is O(log n) and peek() function is constant.

And since the for loop runs n times, therefore time complexity will be n\*O(log n) or O(n log n).

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

PriorityQueue< Integer> pq = new PriorityQueue< >();

int[] ranks = (22, 99, 3, 11, 88, 4, 1)

for (int i = val : ranks) {

pq.add(arr[i]);

}

while (pq.size() > 0) {

System.out.println(pq.peek());

pq.remove();

}

}

}

So I hope that you have reached the stage where you can easily guess the output of the above code.

**Can you guess it?**

Yes! It's: **1 3 4 11 22 88 99** (in different lines, of course!).

**So, as it is noticeable that we get sorted values by using this way. As we can do sorting this way, it's also called Heap Sort. And the time complexity of this sorting is O(n logn) as the time complexity of each loop is O(n logn) which sum up to be 2 O(n log n) which is O(n log n) only.**

However, Heap Sort is of two types, this one is where space complexity is O(n). And in another one constant space is used.

If you remember, we mentioned that we can set the priority order according to our needs. And there are only two ways to set priority. Out of which "rank "one (where smaller value has more priority) is default.

**But what if we need to change this priority order?**

We can change this order by simply adding some magical words in the parenthesis of the Priority Queue constructor while defining any Priority Queue. And those magical words are **Collections.reverseOrder()**



**What will be the output of the above discussed code if this Priority Queue is used?**

Exactly! The order of output will get reversed.

And it would look like: **99 88 22 11 4 3 1** (in different lines, of course!).