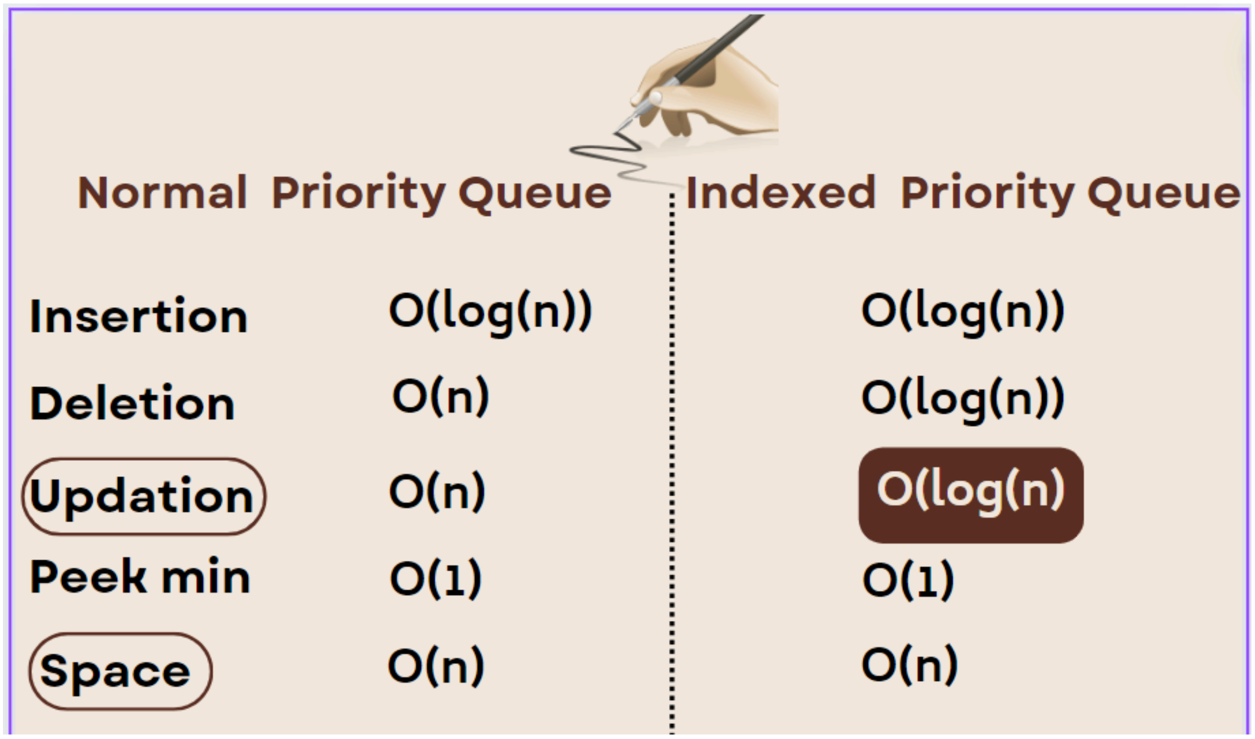
INDEXED PRIORITY QUEUE

There are many combinations of data structures which fill each other's shortcomings by their property and make those combinations more useful than useful in terms of complexity .
It might seem that why those individual data structures exist, if combinations are way better than both individuals ?? But we must notice that if a combination is providing some facility in terms of time complexity then they might be taking extra space complexity , so it is very important to think when to use which, either individual or combination.

Top 5 majorly used Data Structure Combinations
1. Hash Table (Unordered_map) + Linked List
2. **Priority Queue (Min-Heap) + Hash Map**
3. Binary Search Tree (BST) + Augmented Data
4. Trie (Prefix Tree) + Hash Map
5. Graph + Union-Find (Disjoint Set)

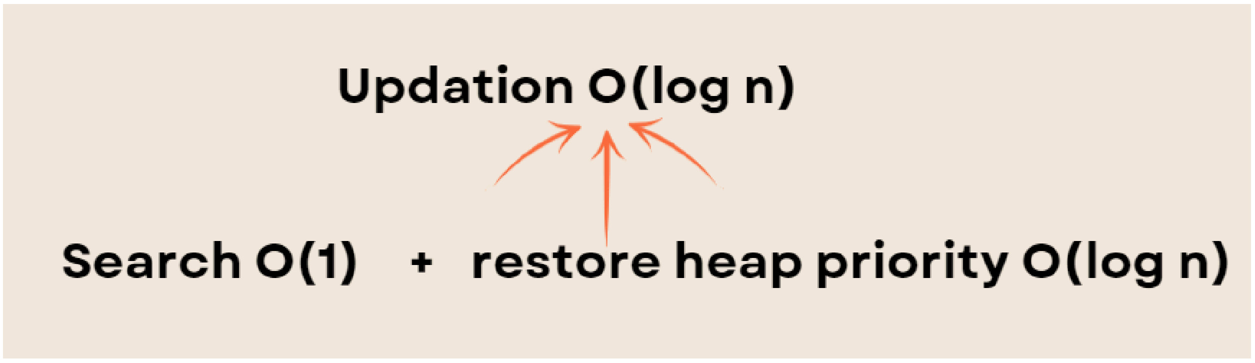
Indexed Priority Queue (IPQ) :- We are implementing Priority Queue(Min-Heap) with Hashmap . Which is used in Implementation of Dijkstra Algorithm in Graphs.



	Normal Priority Queue	Indexed Priority Queue
Insertion	O(log(n))	O(log(n))
Deletion	O(n)	O(log(n))
Updation	O(n)	O(log(n)
Peek min	O(1)	O(1)
Space	O(n)	O(n)

The main work of map is to map items and retrieve it in constant time , while min heap widely used for continuous insertion and deletion with priority based arrangement happens with less number of updation as it takes O(n) time complexity to first search that element in heap to update it.

And that O(n) can be overcome with the combination of Hashmap like properties and make it possible in O(log n) as Hashmap will give the position of element you want to update in O(1) but to maintain the property of min heap we need to float it up or down based on priority will take O(log n).



We will use this in the Dijkstra Algorithm to make time complexity better than the lazy implementation of it. In general, the eager implementation(Which uses IPQ) is often preferred because it avoids unnecessary operations and typically has better practical performance as it does not store unnecessary multiple nodes of the same type.

Code Github Link