LAB # 04

ImplementING PRIORITY QUEUE

# OBJECTIVE

Manage a set of records with priority queue using queue and heapq module in python.

# THEORY

**Priority Queue**

A priority queue is an abstract data type (ADT) which is like a regular queue or stack data structure, but where additionally each element has a priority associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.

While priority queues are often implemented with heaps, they are conceptually distinct from heaps. A priority queue is an abstract concept like a list or a map; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods such as an unordered array.

**ALGORITHM: Priority Queue**

A priority queue is a specialized type of queue

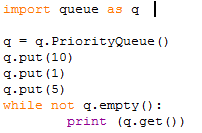
* Items added to queue are assigned an order of rank
* Items of higher priority are removed before those of lower priority
* Items of equal priority are removed in FIFO order
* Items A has a higher priority than item B if A<B

**Module in Python which support Priority queue:**

**Queue:**

The queue module provides a first-in, first-out (FIFO) data structure suitable for multi-threaded programming. As we can see from the output, the queue stores the elements by priority not by the order of element creation. Note that depending on the Python versions, the name of the priority queue is different.

**Example: Output:**

In queue module,

* PriorityQueue ( ) is a subclass of Queue; retrieves entries in priority order (lowest first). Entries are typically tuples of the form: (priority number, data).
* Empty ( ), return True if the queue is empty, False otherwise.
* Put (item), put an item into the queue. If the queue is full, wait until a free slot is available before adding item.
* Get ( ), remove and return an item from the queue. If queue is empty, wait until an item is available.

The module implements three types of queue, which differ only in the order in which the entries are retrieved. In a FIFO queue, the first tasks added are the first retrieved. In a LIFO queue, the most recently added entry is the first retrieved (operating like a stack). With a priority queue, the entries are kept sorted (using the heapq module) and the lowest valued entry is retrieved first.

**Heapq:**

Python has a heapq module that implements a priority queue using a binary heap. Python's heapq module implements a binary min-heap on top of a list.

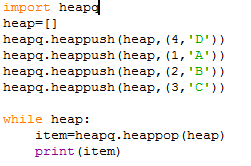
Heaps are binary trees for which every parent node has a value less than or equal to any of its children. This implementation uses arrays for which heap[k] <= heap[2\*k+1] and heap[k] <= heap[2\*k+2] for all k, counting elements from zero. For the sake of comparison, non-existing elements are considered to be infinite. The interesting property of a heap is that its smallest element is always the root, heap[0]. Heap[0] is the smallest item, and heap.sort() maintains the heap invariant

A max-heap ensures that the parent is larger than or equal to both of its children. A min-heap requires that the parent be less than or equal to its children. Python’s heapq module implements a min-heap.

**Function of heapq:**

* First initialize Heap=[ ]
* **Method: heapq.heappush(Heap,n):** add n into heap
* **Method: heapq.heapify(list):** convert list as a heap
* **Method: heapq.heappop(Heap):** pop the smallest item
* **Method: heapq.heapreplace(Heap,n):** replace the smallest item with n

**Example: Output:**

** **

**Exercise:**

1. In contrast to the standard FIFO implementation of Queue, the LifoQueue uses last-in, first-out ordering (normally associated with a stack data structure). Implement LIFO queue using queue module.
2. We have an array of 5 elements: [4, 8, 1, 7, 3] and we have to insert all the elements in the max-priority queue. First as the priority queue is empty, so 4 will be inserted initially.   
   Now when 8 will be inserted it will move to front as 8 is greater than 4. While inserting 1, as it is the current minimum element in the priority queue, it will remain in the back of priority queue. Now 7 will be inserted between 8 and 4 as 7 is smaller than 8.  
   Now 3 will be inserted before 1 as it is the 2nd minimum element in the priority queue. All the steps are represented in the diagram below:



**Home Assignment:**

Implement graph using adjacency list using list or dictionary , make a class such as Vertex and Graph then make some function such as add\_nodes , add\_edges, add\_neighbors, add\_vertex, add\_vertices and suppose whatever you want to need it.

