LAB # 03

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

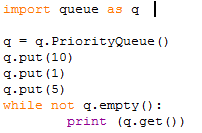
* Items added to queue are assigned an order of rank
* Items of higher priority are removed before those of lower priority
* Items of equal priority are removed in FIFO order
* Items A has a higher priority than item B if A<B

**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:**

   
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.

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.

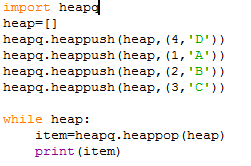
Heaps are binary trees for which every parent node has a value less than or equal to any of its children. This mplementation uses arrays for which heap[k] <= heap[2\*k+1] and heap[k] <= heap[2\*k+2] for all k, counting elements from zero. For the sake of comparison, non-existing elements are considered 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):** sort 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:**

** **

**Lab#3 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.

**Code:**

import queue as q

q = q.LifoQueue()

for i in range(9):

q.put(i)

while not q.empty():

print(q.get())

**Output:**

8

7

6

5

4

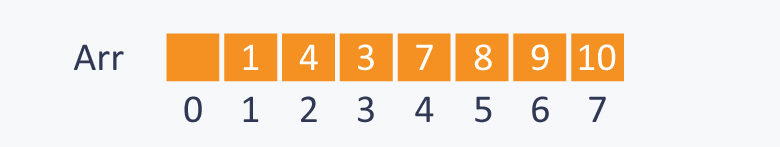
3

2

1

0

1. Suppose you have 7 elements stored in array ‘Arr’. Find the priority queue using heapq in the following array:



**Code:**

import heapq

Arr=[1,0,2,3,5,6,8,7]

heapq.heapify(Arr)

while Arr:

item =heapq.heappop(Arr)

print(item)

**Output:**

0

1

2

3

5

6

7

8

# HOME ASSIGNMENT

Q1. Implement Priority Queue using heapq module

**Code:**

import heapq

heap=[]

heapq.heappush(heap,10)

heapq.heappush(heap,1)

heapq.heappush(heap,5)

while heap:

item =heapq.heappop(heap)

print(item)

**Output:**

1

5

10

Q2. The lowest valued entries are retrieved first. A typical pattern for entries is a tuple in the form: (priority\_number, data) using queue module.

Using pattern: (8,”low”)

(1,”Very Imp”)

(10, “Very Low”)

(5,”Normal”)

(4, ”Imp”)

**Code:**

import heapq

heap=[]

heapq.heappush(heap,(8,'low'))

heapq.heappush(heap,(1,'Very Imp'))

heapq.heappush(heap,(10, 'Very Low'))

heapq.heappush(heap,(5,'Normal'))

heapq.heappush(heap,(4, 'Imp'))

while heap:

item =heapq.heappop(heap)

print(item)

**Output:**

(1, 'Very Imp')

(4, 'Imp')

(5, 'Normal')

(8, 'low')

(10, 'Very Low')