A Priority Queue is a data type that can help maintaining the dataset, and should be perceived as an extension to a regular queue. Priority queues have defined rules to set priorities amongst the group of items they entail. The highest priority items are moved to the front of the priority queue, and the lowest priority elements to the end. The best example of the way a priority queue works is by imagining multiple patients at a hospital, waiting to be be seen. Their illness/ailments will define their priority in the queue.

Priority Queues have the following characteristics:

- Each item has some priority associated with it.
- An item with the highest priority is moved at the front and deleted first.
- If two elements share the same priority value, then the priority queue follows the first-in-first-out principle for de queue operation.

The most efficient way to implement a priority queue is by using a binary heap. See the time complexities below:

- Insert(enqueue): 0(log(N))
- Remove(dequeue): 0(log(N))
- Lookup: O(1)

In Python, it is necessary to import the “heapq” module to implement a priority queue using the heap method. The operations we can perform on the heap using the "heapq" module are:

- heapify(iterable): converts the iterable into a heap, i.e. in heap order.

- heappush(heap, ele): inserts an element into the heap; the order is adjusted to maintain the heap structure.

- heappop(heap): removes and returns the smallest element from heap; the order is adjusted to maintain the heap structure.

- heappushpop(heap, ele): combines the functioning of both push and pop operations in one statement, increasing the heap's efficiency; the order is kept after this operation.

- heapreplace(heap, ele): inserts and pops element in one statement, but the smallest element is popped before the new element is pushed.

- heapreplace(): returns the smallest value originally in the heap regardless of the pushed element, as opposed to heappushpop().

In [2]:

```python
import heapq

# Initializing list
new_list = [5, 7, 9, 1, 3]

# Convert list into heap using heapify
heapq.heapify(new_list)

# Printing newly created heap
print(f"The created heap is : ", end="")
print(list(new_list))
```

```
The created heap is : [1, 3, 9, 7, 5]
```

In [3]:

```python
# Push elements into heap using heappush()
heapq.heappush(new_list, 4)

# Printing modified heap
print("The modified heap after push is : ", end="")
print(list(new_list))
```

```
The modified heap after push is : [1, 3, 4, 7, 5, 9]
```

In [4]:

```python
# Pop smallest element using heappop()
print("The popped and smallest element is : ", end="")
print(heapq.heappop(new_list))
```

```
The popped and smallest element is : 1
```

In [5]:

```python
# Creating two identical heaps to demonstrate the difference between heappushpop() and heapreplace()
list1 = [5, 7, 9, 4, 3]
list2 = [5, 7, 9, 4, 3]
heapq.heapify(list1)
heapq.heapify(list2)

# Push and pop items simultaneously using heappushpop()
print("The popped item using heappushpop() is : ", end="")
print(heapq.heappushpop(list1, 2))
```

```
The popped item using heappushpop() is : 2
```

In [6]:

```python
# Push and pop items simultaneously using heapreplace()
print("The popped item using heapreplace() is : ", end="")
print(heapq.heapreplace(list2, 2))
```

```
The popped item using heapreplace() is : 3
```