S Abhishek AM.EN.U4CSE19147

Data Structures

1. Implement the priority queue using unsorted list.

```
class list:
    def __init__(self):
        self.s = []
        self.size = 0

    def display(self):
        if self.size == 0:
            print("Display : The List is Empty")
            return
        else:
            print(self.s)

    def enqueue(self):
        while True:

            x = int(input("Press 1 to Insert or any other Key to Exit : "))
            if x == 1:
                a = int(input("Enter the Element to Insert : "))
                b = int(input("Enter the Priority to Insert : "))
                self.s.append((a, b))
                self.size = self.size + 1
                self.display()

            else:
                return

    def dequeue(self):
        while True:
            x = int(input("Press 1 to Delete or any other Key to Exit : "))
```

```python
        if x == 1:
            if self.size == 0:
                print("List is Empty")
                return
            else:
                self.s = sorted(self.s, key=lambda x: x[1])
                temp = self.s.pop(0)
                print("Poped Element : {}".format(temp))
                self.size = self.size - 1
        else:
            return


if __name__ == '__main__':
    l = list()
    l.enqueue()
    l.dequeue()
```

```
Press 1 to Insert or any other Key to Exit : 1
Enter the Element to Insert : 2
Enter the Priority to Insert : 6
[(2, 6)]
Press 1 to Insert or any other Key to Exit : 1
Enter the Element to Insert : 2
Enter the Priority to Insert : 4
[(2, 6), (2, 4)]
Press 1 to Insert or any other Key to Exit : 1
Enter the Element to Insert : 3
Enter the Priority to Insert : 2
[(2, 6), (2, 4), (3, 2)]
Press 1 to Insert or any other Key to Exit : 2
Press 1 to Delete or any other Key to Exit : 1
Poped Element : (3, 2)
Press 1 to Delete or any other Key to Exit : 1
Poped Element : (2, 4)
Press 1 to Delete or any other Key to Exit : 1
Poped Element : (2, 6)
Press 1 to Delete or any other Key to Exit : 1
List is Empty
```

2. Implement the priority queue using sorted list.

```
class list:
    def __init__(self):
        self.s = []
        self.size = 0

    def display(self):
        if self.size == 0:
            print("Display : The List is Empty")
            return
        else:
            print(self.s)

    def enqueue(self):
        while True:

            x = int(input("Press 1 to Insert or any other Key to Exit : "))
            if x == 1:
                a = int(input("Enter the Element to Insert : "))
                b = int(input("Enter the Priority to Insert : "))
                self.s.append((a, b))
                self.s = sorted(self.s, key=lambda x: x[1])
                self.size = self.size + 1
                self.display()

            else:
                return

    def dequeue(self):
        while True:
            x = int(input("Press 1 to Delete or any other Key to Exit : "))
            if x == 1:
                if self.size == 0:
                    print("List is Empty")
                    return
```

```python
            else:
                self.s.pop(0)
                self.display()
                self.size = self.size - 1
        else:
            return


if __name__ == '__main__':
    l = list()
    l.enqueue()
    l.dequeue()
```

```
Press 1 to Insert or any other Key to Exit : 1
Enter the Element to Insert : 2
Enter the Priority to Insert : 5
[(2, 5)]
Press 1 to Insert or any other Key to Exit : 1
Enter the Element to Insert : 7
Enter the Priority to Insert : 3
[(7, 3), (2, 5)]
Press 1 to Insert or any other Key to Exit : 2
Press 1 to Delete or any other Key to Exit : 1
[(2, 5)]
Press 1 to Delete or any other Key to Exit : 1
[]
Press 1 to Delete or any other Key to Exit : 1
List is Empty
```

3. Implement the priority queue using heap

```
class Heap:
    @classmethod
    def push(cls, iterable, val):
        iterable.append(val)
        Heap._bubble_up(iterable, 0, len(iterable) - 1)

    @classmethod
    def pop(cls, iterable):
        last_item = iterable.pop()
        res = iterable[0]
        iterable[0] = last_item
        Heap._bubble_to_bottom(iterable)
        return res

    @classmethod
    def _bubble_to_bottom(cls, iterable, start=0):
        end_index = len(iterable)
        current_index = start
        out_of_place_item = iterable[current_index]
        child_index = (2 * current_index) + 1
        while child_index < end_index:
            right_child_index = (2 * current_index) + 2
            if right_child_index < end_index and iterable[right_child_index] <=
iterable[child_index]:
                child_index = right_child_index

            iterable[current_index] = iterable[child_index]
            current_index = child_index
            child_index = (2 * current_index) + 1

        iterable[current_index] = out_of_place_item
        Heap._bubble_up(iterable, start, current_index)

    @classmethod
```

```python
    def heapify(cls, iterable):
        for i in range((len(iterable) // 2) - 1, -1, -1):
            Heap._bubble_to_bottom(iterable, i)

    @classmethod
    def _bubble_up(cls, iterable, stop, start):
        current_index = start
        new_item = iterable[current_index]
        while current_index > stop:
            parent_index = (current_index - 1) >> 1
            if new_item < iterable[parent_index]:
                iterable[current_index] = iterable[parent_index]
                current_index = parent_index
                continue
            break
        iterable[current_index] = new_item

a = []
Heap.push(a, 7)
Heap.push(a, 4)
Heap.push(a, 1)
Heap.push(a, 3)
print("Heap : {}".format(a))
print("Heap pop : {}".format(Heap.pop(a)))
print("Heap pop : {}".format(Heap.pop(a)))
print("Heap : {}".format(a))
```

```
Heap : [1, 3, 4, 7]
Heap pop : 1
Heap pop : 3
Heap : [4, 7]
```

One Drive : [Click Me!!]()

Thankyou