Priority Queue

Objective

- To learn the implementation of priority queue data structure with its basic operations.
- Implement a priority queue using both normal array and circular array.
- Also, provide a linked list based implementation.

Task

1. Array based Priority Queue

```
Class Queue<T extends Comparable<T>>{
        T[] Q;
        int F;

    Queue(){
       Q=(T[])new Comparable[10];
       F=-1;
   }

    Queue(int size){
       Q=(T[])new Comparable[size];
       F=-1;
   }

   public void Enqueue(T obj){…}

   Public T Dequeue(){…}

   public boolean isEmpty(){…}

   public boolean isFull(){…}

   public T peek(){…}

   public String toString() {…}
}// class end
```

You have to implement a program to manage a printer queue efficiently. Suppose multiple printing jobs are coming to the printer however printer can handle one job at a time while other jobs will have to wait in a queue until printer again becomes free. You have to release *shortest job first* from the queue to be assigned to the printer. On the other hand, if jobs are of same size then delete on first come first serve basis (FIFO).
You have to implement the priority queue using Array to store printer's jobs with their priorities. To manage the queue you have to implement the following methods:

- Enqueue(): Insert new job (maintain the order of shortest job first) in O(n), if queue is not full then insert.
- Dequeuer(): Delete shortest jobs first in O(1), if queue is not empty then delete.
- Peek(): Return the peek element in the queue (the element which will delete next from queue)
- toString(): display all jobs waiting in a queue.