```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/*
A ordered list implementation of Priority Queue, the
minimum element i.e. Node with smallest priority is stored
at A[0] position of the list. Each enqueue operation is followed
by sort operation. The sorting algorithm used is quicksort due to its
fast average time. The quick sort is also implemented in this class
as auxiliary function
Since the class itself act as auxiliary class to the dijkstra's algorithm, the
implementation does not need to be generic and can only store data type of Node
*/
public class PriorityQueue
{
    // the underlying storage is a linked list
    // the list stored both the node and its priority
    // which in dijkstra's algorithm will just be its distance
    // the queue is kept as ordered list so its minimum element is
    // just the head
    private LinkedList<(Node, int)> queue;
    public PriorityQueue() {
        queue = new LinkedList<(Node, int)>();
    }

    /*
    Insert the new node into its proper position
    */
    public void enqueue(Node node, int priority) {
        // if queue is empty, insert it at beginning
        if(queue.Count == 0) {
            queue.AddFirst((node, priority));
        }
        else {
            // loop through the linked list collection from head
            //
            foreach((Node, int) v in queue) {
                if(v.Item2 > priority) {
                    queue.AddBefore(queue.Find(v), (node, priority));
                    return;
                }

            }
        }
        queue.AddLast((node, priority));
    }

    public bool isEmpty() {
        return queue.Count == 0;
    }

    /*
    delete and return the first element
    */
    public (Node, int) dequeue() {
        // check if its empty
        if(queue.Count == 0) {
            throw new System.Exception("Queue Underflow");
        }
```

```
61          (Node, int) temp = queue.First.Value;
62          queue.RemoveFirst();
63          return temp;
64      }
65
66      public (Node, int) findNode(Node node) {
67          foreach((Node, int) v in queue) {
68              if(v.Item1 == node) {
69                  return v;
70              }
71          }
72          return (null, 0);
73      }
74
75      /*
76      The function takes a node and update its priority
77      based on the given newKey and need to reset it to
78      its proper position
79      */
80      public void updateKey(Node node, int newKey) {
81          // remove the existing node from the queue
82          queue.Remove(findNode(node));
83          // call enqueue on the node with new value so its inserted at right place
84          enqueue(node, newKey);
85      }
86  }
87
```