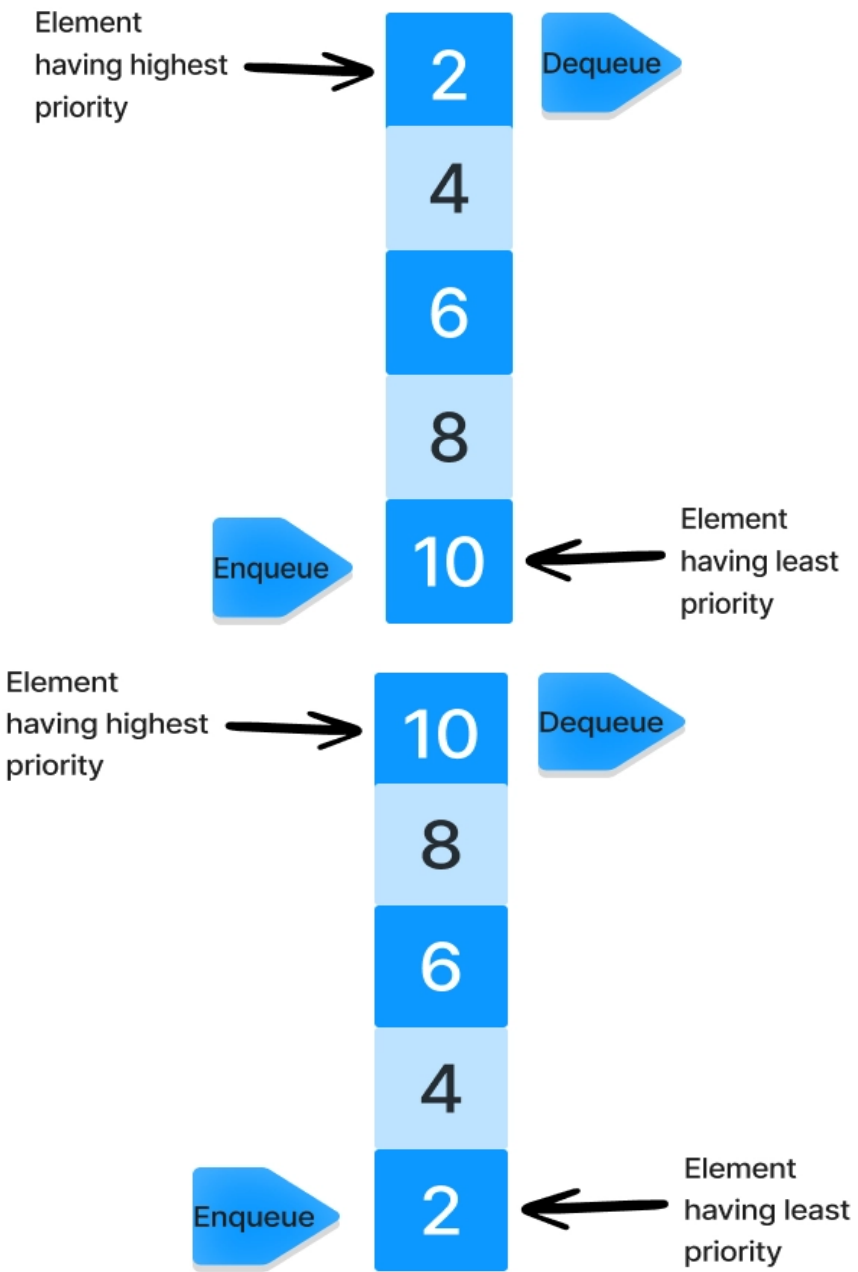
# Priority Queues

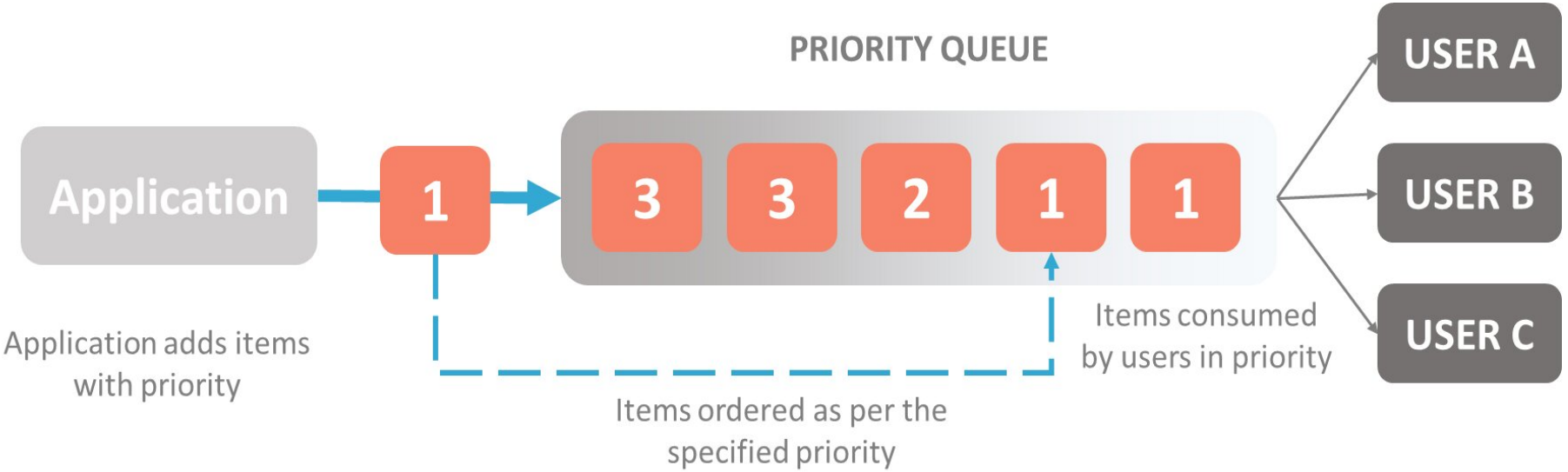
# Priority Queue

- A Priority Queue – a different kind of queue.
- Similar to a regular queue:
  - insert at rear,
  - remove from front.
- Items in priority queue are ordered by some key
- Item with the lowest key / highest key is always at the front from where they are removed.

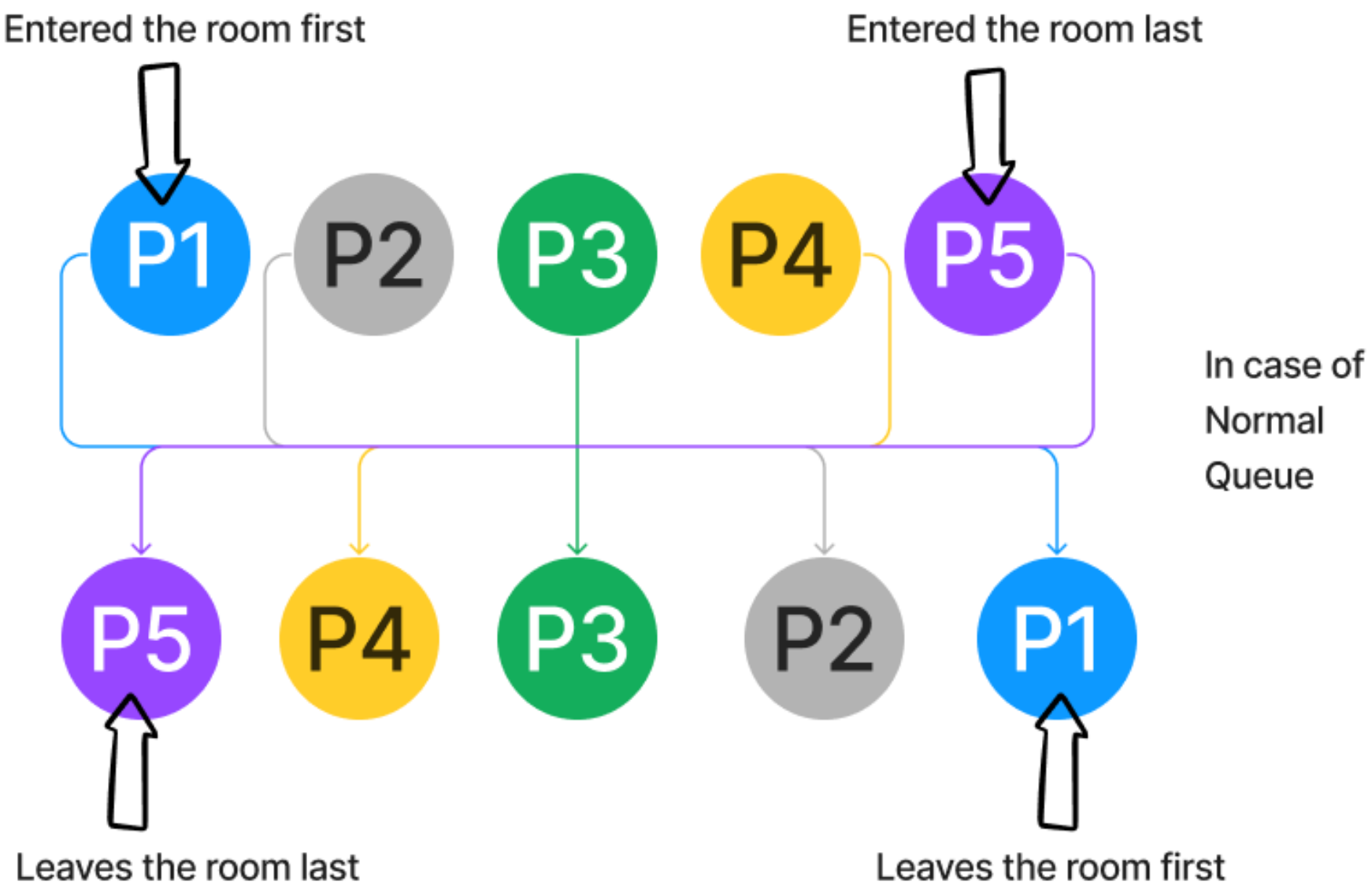


# Priority Queue

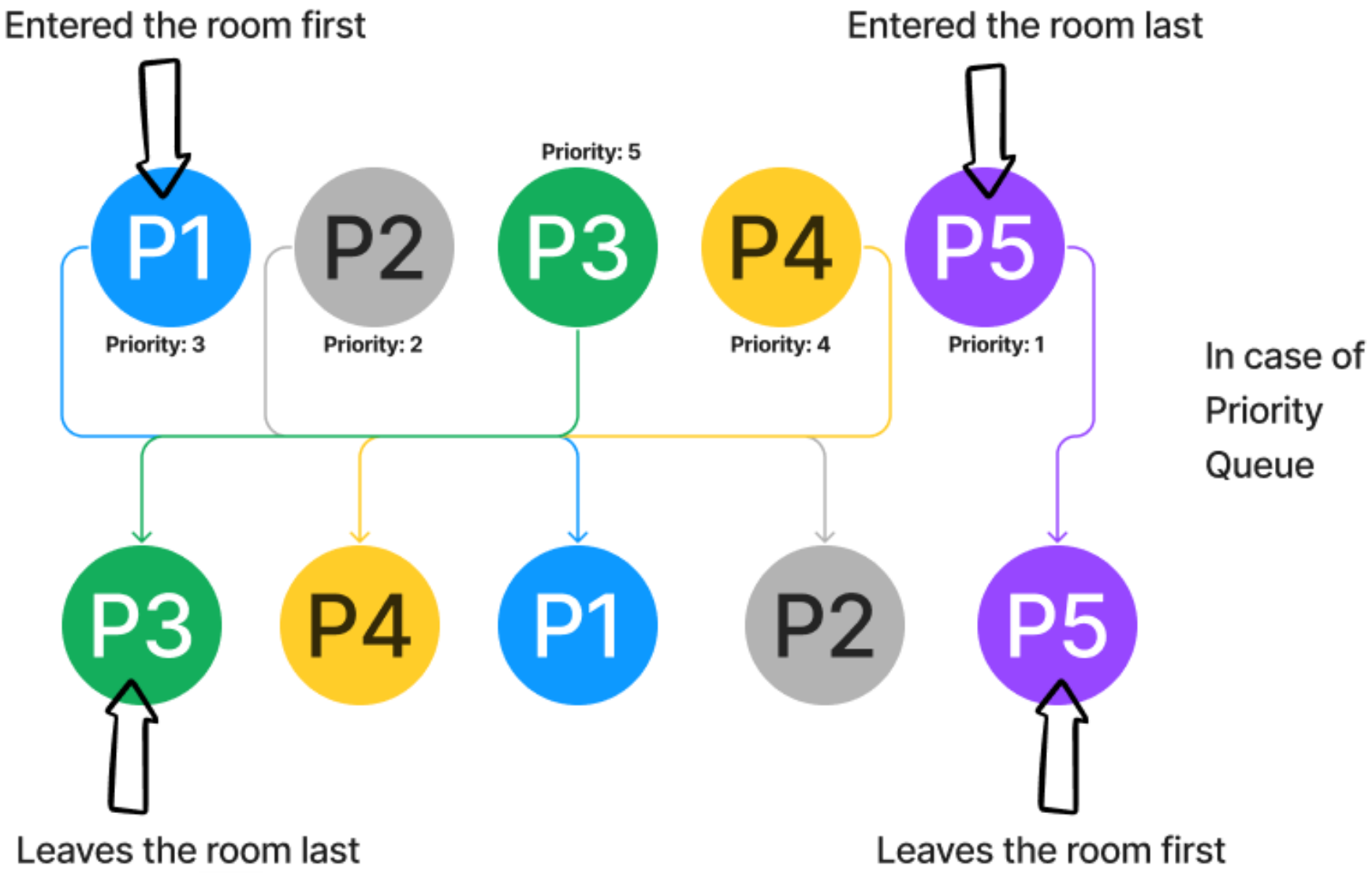
- Items then ‘inserted’ in ‘proper’ position
- Idea behind the Priority Queue is simple:
  - Is a queue
  - But the items are ordered by a key.
  - Implies your ‘position’ in the queue may be changed by the arrival of a new item.



# Normal Queue



# Priority Queue



# Applications of Priority Queues

- Many, many applications.
  - Scheduling queues for a processor, print queues, transmit queues, disk scheduler etc.

Note: a priority queue is no longer FIFO!

You will still remove from the front of the queue, but insertions are governed by a priority.

# Priority Queues: Access

- remove()
  - So, the first item has priority and can be retrieved (removed) quickly and returned to calling environment.
  - Hence, 'remove()' is easy and will take O(1) time
- insert()
  - But, we want to insert quickly. Must go into proper position.

# Priority Queues: Access

Implementing using array:

- slow to insert( ), but this is the simplest and best approach where
  - Number of items in the pqueue small, and
  - Insertion speed is not critical.

# Priority Queues: Types

- **Min Priority Queue**
  - Minimum value gets the highest priority and
  - Maximum value gets the lowest priority.

  **Also called Ascending Order Priority Queue**

- **Max Priority Queue**:
  - Maximum value gets the highest priority and
  - Minimum value gets the lowest priority

  **Also called Descending Order Priority Queue**

# Priority Queues: Implementation

- Priority Queue can be implemented in two ways:
  - Using ordered Array: In ordered array enqueue operation takes O(n) time complexity because it enters elements in sorted order in queue. Deletion takes O(1) time complexity.
  - Using unordered Array: In unordered array deletion takes O(n) time complexity because it search for the element in Queue for the deletion and enqueue takes O(1) time complexity.

# Max Priority Queue Implementation(Unordered)

- Enqueue – Insert the item at the end of the priority queue takes O(1) time
- Dequeue – Remove the item with the highest priority
- Peek – Return item with highest priority

# Max Priority Queue (Unordered) Implementation in C

# C program to demonstrate Priority Queue

```
#include<stdio.h>
#include<limits.h>
#define MAX 100
int idx = -1;
// denotes where the last item in priority queue is
// initialized to -1 since no item is in queue
int pqVal[MAX];
// pqVal holds data for each index item
int pqPriority[MAX];
// pqPriority holds priority for each index item
```

# C program to demonstrate Priority Queue

```
int isEmpty( )
{    return idx == -1;
}

int isFull( )
{     return idx == MAX - 1;
}
```

# C program to demonstrate Priority Queue

```
//enqueue adds item to the end of the priority queue O(1)
void enqueue(int data, int priority)
{   if(!isFull())
    {   idx++;
        pqVal[idx] = data;
       // Insert the element in priority queue
        pqPriority[idx] = priority;
    }
}
```

# C program to demonstrate Priority Queue

```
// peek returns item with highest priority
// Max Priority Queue High priority number
// means higher priority | O(N)
int peek( )
{ // Max Priority, so assigned min value as initial value
    int maxPriority = INT_MIN;
    int indexPos = -1;
```

# C program to demonstrate Priority Queue

```
// Linear search for highest priority
  for (int i = 0; i <= idx; i++)
  { // If two items have same priority choose the one
    // with higher data value
     if ( maxPriority == pqPriority[i] && indexPos > -1
              && pqVal[indexPos] < pqVal[i] )
      {   maxPriority = pqPriority[i];
          indexPos = i;
      }
```

# C program to demonstrate Priority Queue

```
// MAX Priority so higher priority number means
// higher priority
        else if (maxPriority < pqPriority[i])
        {   maxPriority = pqPriority[i];
            indexPos = i;
        }
    }
  return indexPos;
}
```

# C program to demonstrate Priority Queue

```
// dequeue() removes the element with highest priority
// from the priority queue | O(N)
void dequeue( )
{  if( !isEmpty() )
    {  // Get element with highest priority
       int indexPos = peek();
// reduce size of priority queue by first shifting all elements
// one position left from index where the
// highest priority item was found
```

# C program to demonstrate Priority Queue

```
for (int i = indexPos; i < idx; i++)
   {      pqVal[i] = pqVal[i + 1];
          pqPriority[i] = pqPriority[i + 1];
       }
 // reduce size of priority queue by 1
       idx--;
   }
}
```

# C program to demonstrate Priority Queue

```
void display( )
{   for (int i = 0; i <= idx; i++)
    {  printf ( "(%d, %d) \n",
                    pqVal[i], pqPriority[i]);
    }
}
```

# C program to demonstrate Priority Queue

```
int main( )
{   enqueue(5, 1);     enqueue(10, 3);    enqueue(15, 4);
     enqueue(20, 5);    enqueue(500, 2);
    printf("Before Dequeue : \n");
    display();
    // Dequeue the top element
    dequeue();    // 20 dequeued
    dequeue();    // 15 dequeued
     printf("\nAfter Dequeue : \n");
    display();
}
```

# Max Priority Queue Implementation(Ordered)

- Dequeue – Remove the item from the end takes O(1) time
- Enqueue – Insert item according to their priority, lowest priority at the start and highest priority at the end. Items are arranged in ascending order of their priority value
- Peek – Return item with highest priority. Last item in the array itself will have highest priority

# Max Priority Queue (Ordered) Implementation in C

```
#include<stdio.h>
#include<limits.h>
#define MAX 100
// denotes where the last item in priority queue is
// initialized to -1 since no item is in queue
int idx = -1;
// pqVal holds data for each index item
// pqPriority holds priority for each index item
int pqVal[MAX];
int pqPriority[MAX];
```

```
int isEmpty( )
{     return idx == -1;
}

int isFull( )
{    return idx == MAX - 1;
}
```

# // Insert the element maintaining the sorted order of priority

```
void enqueue(int data, int priority)
{    if( !isFull( ) )
     {  if( idx == -1 )    //  If empty, insert the New item
        {       idx++;
                pqVal[idx] = data;
                pqPriority[idx] = priority;
                return;
         }
        else {     idx++;
```

```
// shift all items rightwards with higher
// priority than the element we trying to insert
  for(int i = idx-1; i >= 0;i--)
       {  if (pqPriority[i] >= priority)
          {   pqVal[i+1] = pqVal[i];
              pqPriority[i+1] = pqPriority[i];
           }
```

```
      else
        {   // insert item just before where
          // lower priority index was found
           pqVal[i+1] = data;
                  pqPriority[i+1] = priority;
                  break;
          }     // else
        }       // for
      }         // outer else
   }            // if
}               // enqueue
```

```
// peek( ) returns item with highest priority
// note highest priority in max priority queue is last
//  item in array
int peek( )
{    return   idx;
}
```

```
void dequeue( )
{       idx--;
}
// just reducing index would mean we have dequed the
// value would be still there but we can say that
// no more than a garbage value
void display( )
{   for (int i = 0; i <= idx; i++)
    {   printf ( "(%d, %d)\n",
                pqVal[i], pqPriority[i]);
    }
}
```

```
int main()
{  // To enqueue items as per priority
   enqueue(25, 1); enqueue(10, 10);  enqueue(15, 50);
   enqueue(20, 100);   enqueue(30, 5);   enqueue(40,
7);
    printf("Before Dequeue : \n");      display();
    // // Dequeue the top element
   dequeue(); // 20 dequeued
   dequeue(); // 15 dequeued
   printf("\nAfter Dequeue : \n");     display();
}
```