Data Structure & Algorithm

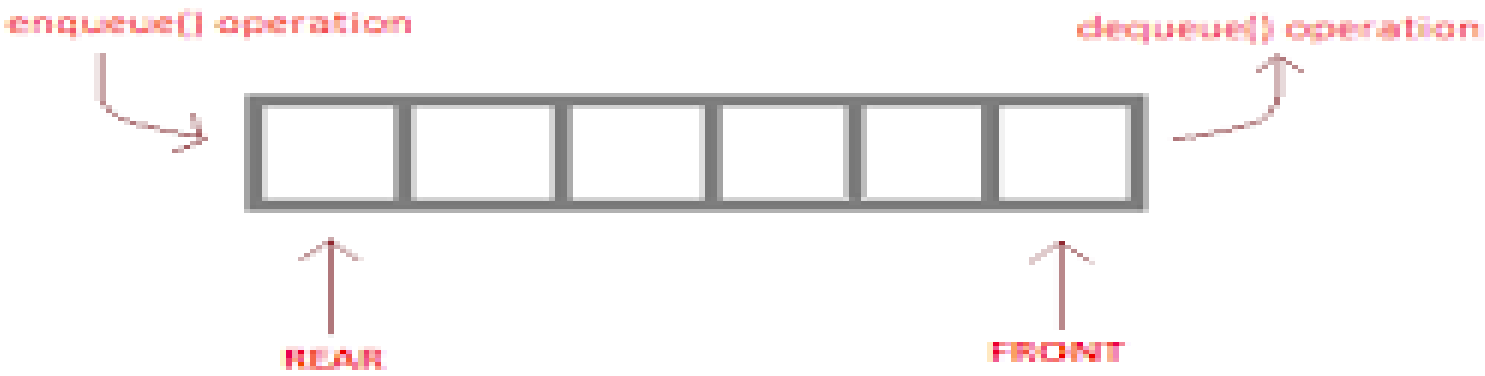
Unit 3: Queue

Linear Data Structure

- It is data structure in which elements are arranged in linear order
- Data items can be traversed in a single run
- Elements are accessed or placed in contiguous memory locations

Queue

- **It is** an abstract data type or a linear data structure, in which the first element is inserted from one end called the **REAR**(also called **tail**), and the removal of existing element takes place from the other end called as **FRONT**(also called **head**).
- It follows FIFO principle
- It is a homogeneous collection of elements in which
  - ✓ new elements are added at one end called rear,
  - ✓ and the existing elements are deleted from other end called front.



enqueue( ) is the operation for adding an element into Queue.
dequeue( ) is the operation for removing an element from Queue .

QUEUE DATA STRUCTURE

Let us illustrate it with animation:

https://www.youtube.com/watch?v=UpvDOm3prfI

Lets see a video for detail explanation about Queue as an ADT

https://youtu.be/zp6pBNbUB2U

- The basic operations (Primitive Operations) also called as Queue ADT that can be performed on queue are

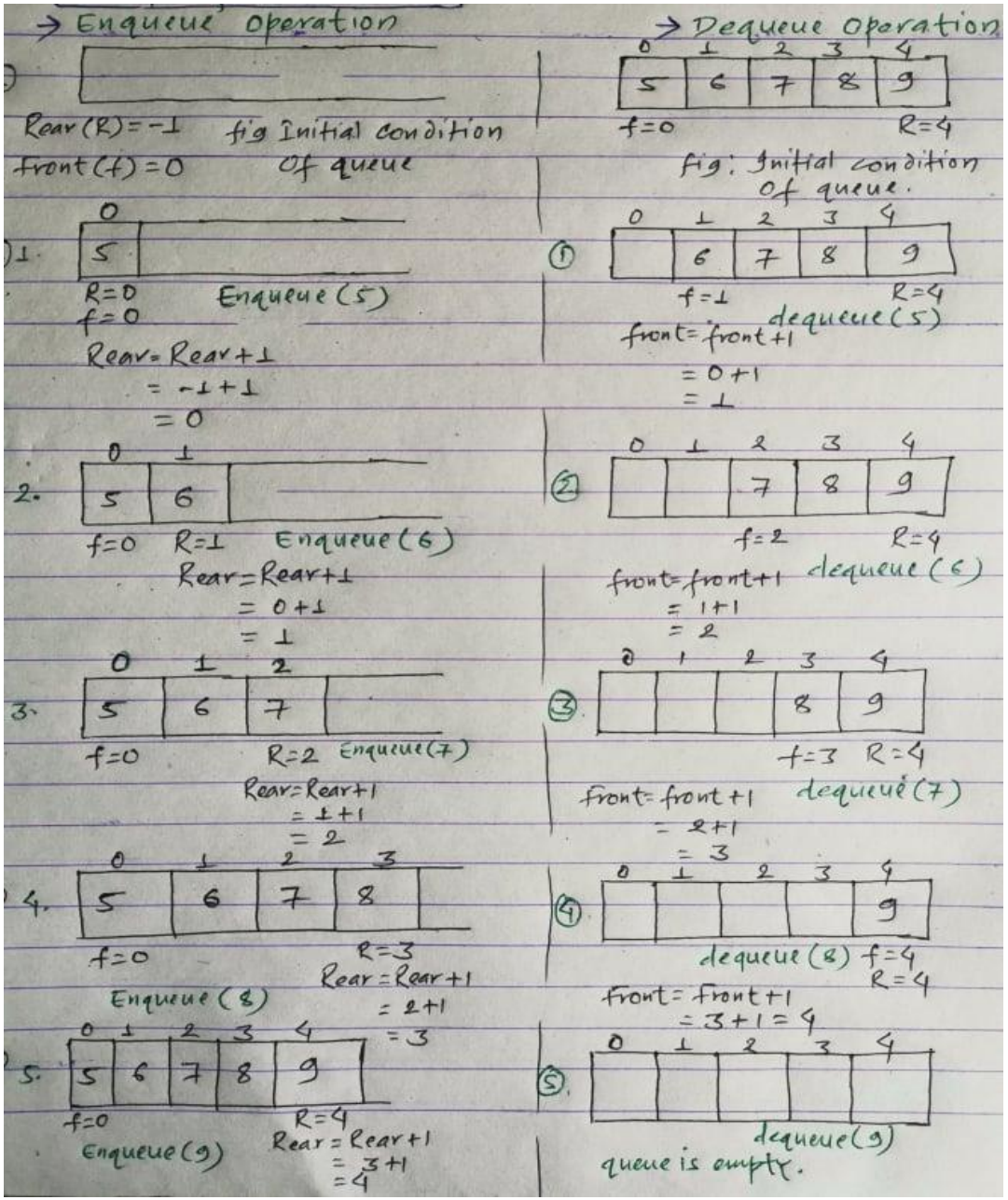
1. Enqueue() or Insert() (or add) an element to the queue (push) from the rear end.

2. Dequeue() or Delete() (or remove) an element from a queue (pop) from front end.

3. Front: return the object that is at the front of the queue without removing it.

4. Empty: return true if the queue is empty otherwise return false.

5. Size: returns the number of items in the queue.


→ Enqueue Operation
Rear(R) = -1
front(f) = 0
fig Initial condition of queue
1.
R=0
f=0
Enqueue (5)
Rear = Rear + 1
= -1 + 1
= 0
2.
f=0 R=1 Enqueue (6)
Rear = Rear + 1
= 0 + 1
= 1
3.
f=0 R=2 Enqueue (7)
Rear = Rear + 1
= 1 + 1
= 2
4.
f=0 R=3
Enqueue (8)
Rear = Rear + 1
= 2 + 1
= 3
5.
f=0 R=4
Enqueue (9)
Rear = Rear + 1
= 3 + 1
= 4
→ Dequeue Operation
f=0 R=4
fig: Initial condition of queue.
1
f=1 R=4
dequeue (5)
front = front + 1
= 0 + 1
= 1
2
f=2 R=4
dequeue (6)
front = front + 1
= 1 + 1
= 2
3
f=3 R=4
dequeue (7)
front = front + 1
= 2 + 1
= 3
4
dequeue (8) f=4
R=4
front = front + 1
= 3 + 1 = 4
5
dequeue (9)
queue is empty.

TYPES OF QUEUE

i) Linear Queue or Simple queue

ii) Circular queue

iii) Double ended queue (de-queue)

iv) Priority queue: Priority queue is generally implemented using linked list

Linear Queue Operations

1. Enqueue operation will insert (or add) an element to queue, at the rear end, by incrementing the array index.
2. Dequeue operation will delete (or remove) from the front end by decrementing the array index and will assign the deleted value to a variable.
3. Initially front is set to 0 and rear is set to -1.
4. The queue is empty whenever rear < front or rear = -1 and front =0
5. Total number of elements in the queue at any time is equal to rear-front+1, when implemented using arrays.
6. Below are the few operations in queue:

- Note. During the insertion of first element in the queue, we always increment the front by one.
- Queue can be implemented in two ways: 1. Using arrays (static) 2. Using pointers (dynamic)
- If we try to dequeue (or delete or remove) an element from queue when it is empty, underflow occurs.
- It is not possible to delete (or take out) any element when there is no element in the queue.

- Suppose maximum size of the queue (when it is implemented using arrays) is 50. If we try to Enqueue (or insert or add) an element to queue, overflow occurs.
- When queue is full it is naturally not possible to insert any more elements.
- **Overflow Condition:**
- If Rear = MAX-1
- **Underflow Condition:**
- If front = 0 and rear = -1 (Initial Condition) or rear < front
- One Element: If rear = front.
- Number of Elements present in a Queue : rear – front +1

Applications of Queue

- Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
- Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.
- Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
- Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
- Queues are used in operating systems for handling interrupts.

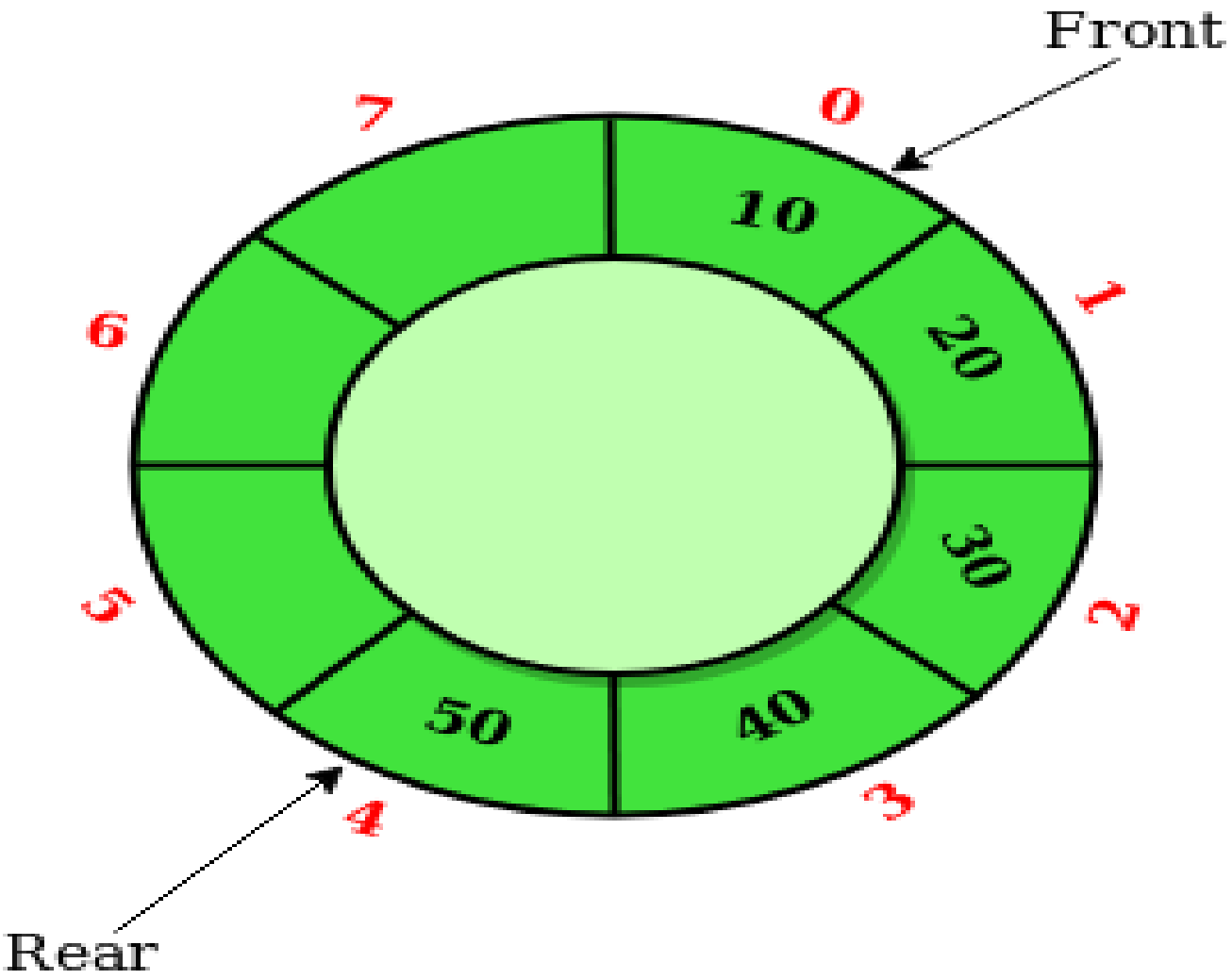
- **Memory management:** The circular queue provides memory management. As we have already seen that in linear queue, the memory is not managed very efficiently. But in case of a circular queue, the memory is managed efficiently by placing the elements in a location which is unused.
- **CPU Scheduling:** The operating system also uses the circular queue to insert the processes and then execute them.
- **Traffic system:** In a computer-control traffic system, traffic light is one of the best examples of the circular queue. Each light of traffic light gets ON one by one after every jinterval of time. Like red light gets ON for one minute then yellow light for one minute and then green light. After green light, the red light gets ON.

Drawback of Linear Queue

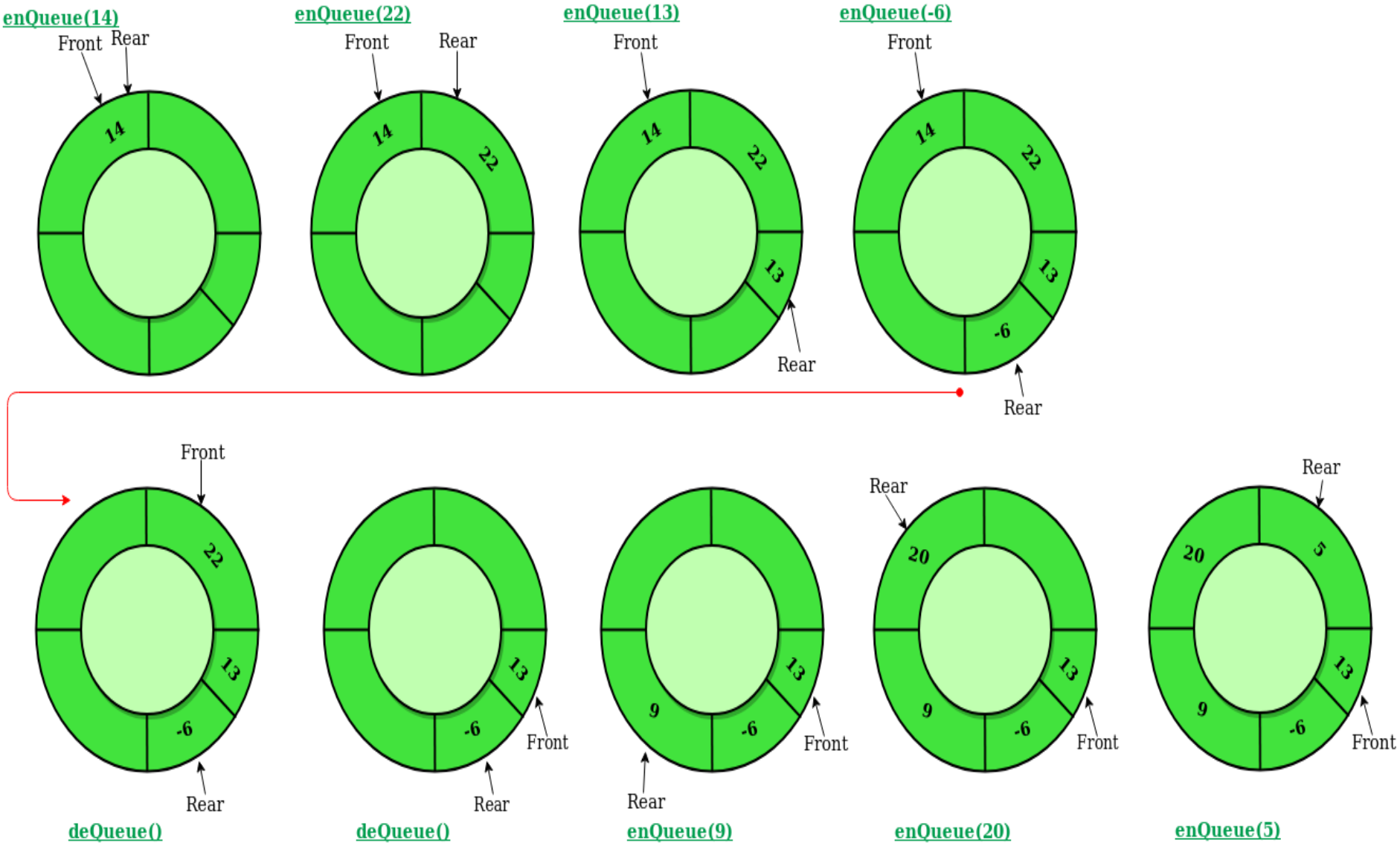
- The linear queue suffers from serious drawback that performing some operations, we cannot insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we cannot insert items into queue.

Circular Queue

- Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.
- It is also called **'Ring Buffer'**.


Front
Rear
0
1
2
3
4
5
6
7
10
20
30
40
50

- In a normal Queue, we can insert elements until queue becomes full.
- But once queue becomes full, we can not insert the next element even if there is a space in front of queue.


enQueue(14)
Front
Rear
14
enQueue(22)
Front
Rear
14
22
enQueue(13)
Front
14
22
13
Rear
enQueue(-6)
Front
14
22
13
-6
Rear
Front
22
13
-6
Rear
deQueue()
13
-6
Front
Rear
deQueue()
9
13
-6
Front
Rear
enQueue(9)
Rear
20
9
13
-6
Front
enQueue(20)
Rear
20
5
9
13
-6
Front
enQueue(5)

Operations on Circular Queue:

- **Front:** Get the front item from queue.
- **Rear:** Get the last item from queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.
  - Check whether queue is Full – Check ((rear == SIZE-1 && front == 0) || (rear == front-1)).
  - If it is full then display Queue is full. If queue is not full then, check if (rear == SIZE – 1 && front != 0) if it is true then set rear=0 and insert element.

- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position.
  - Check whether queue is Empty means check (front==-1).
  - If it is empty then display Queue is empty.
  - If queue is not empty then step 3
  - Check if (front==rear) if it is true
  - then set front=rear= -1
  - else
  - check if (front==size-1),
  - if it is true then set front=0 and
  - return the element.

Priority Queue

- A priority queue is another special type of Queue data structure in which each element has some priority associated with it.
- Based on the priority of the element, the elements are arranged in a priority queue.
- If the elements occur with the same priority, then they are served according to the FIFO principle.

- In priority Queue, the insertion takes place based on the arrival while the deletion occurs based on the priority.
- The priority Queue can be shown as:
- The above figure shows that the highest priority element comes first and the elements of the same priority are arranged based on FIFO structure.

- In priority Queue, the insertion takes place based on the arrival while the deletion occurs based on the priority.
- Priority Queue is an extension of queue with following properties.
- Every item has a priority associated with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

Applications of Priority Queue

1) CPU Scheduling

2) Graph algorithms like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree, etc

3) All queue applications where priority is involved.

Note: a priority queue is implemented using Heap.


Thank
You