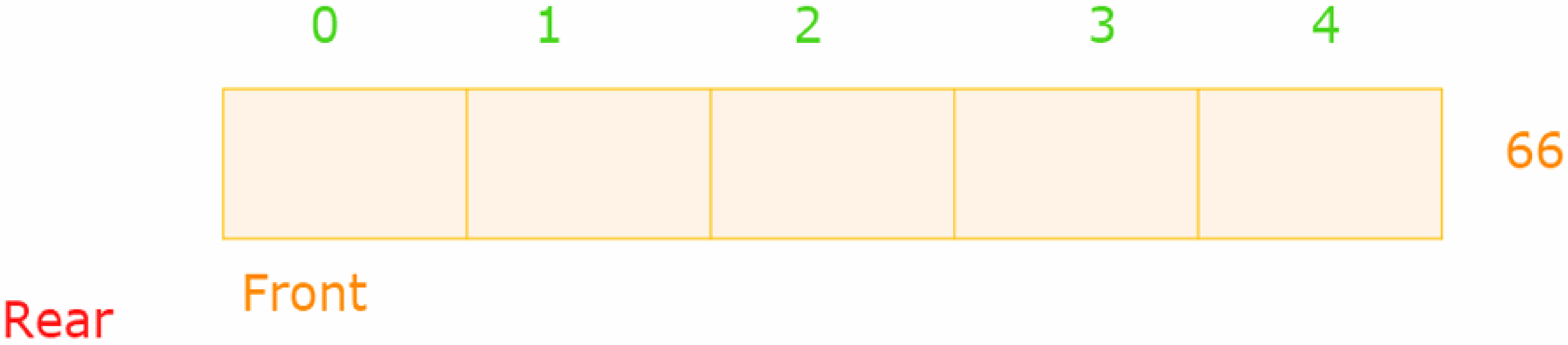
Queue

Kiran Waghmare

Queue

- Ordered collection of homogeneous elements.
- Non-primitive linear data structure.
- A new element is added at one end called rear end and the existing elements are deleted from the other end called front end.
- This mechanism is called First-In-First-Out (FIFO)
- Total no. of elements in queue = rear-front+1

Queue Operations



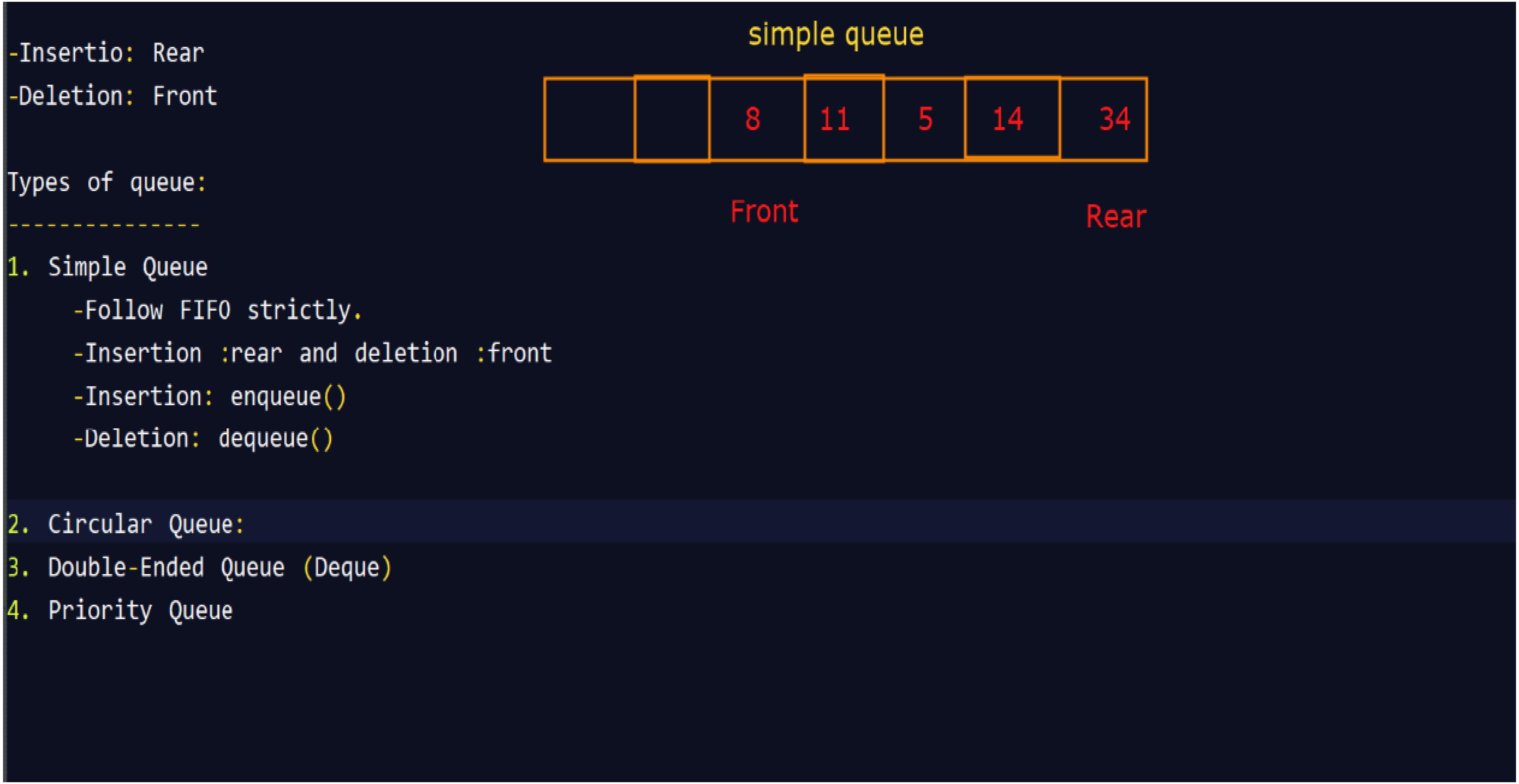
0 1 2 3 4

66

Front

Rear

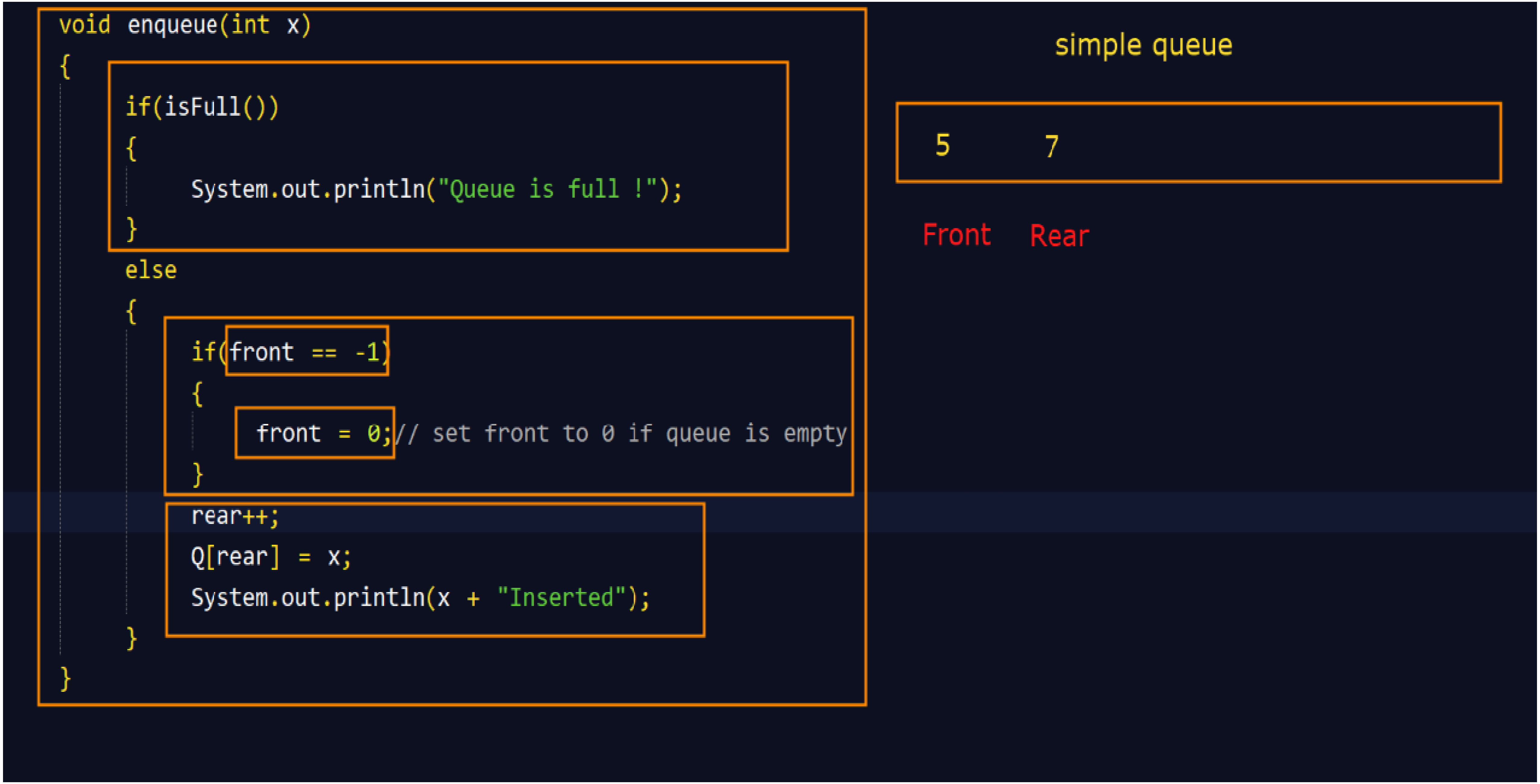
Insert: 1,2,3,4,5
Deletion:1,2, 3, 4,5



```
-Insertio: Rear
-Deletion: Front

Types of queue:
---------------
1. Simple Queue
     -Follow FIFO strictly.
     -Insertion :rear and deletion :front
     -Insertion: enqueue()
     -Deletion: dequeue()

2. Circular Queue:
3. Double-Ended Queue (Deque)
4. Priority Queue
```



```
void enqueue(int x)
{
    if(isFull())
    {
        System.out.println("Queue is full !");
    }
    else
    {
        if(front == -1)
        {
            front = 0;// set front to 0 if queue is empty
        }
        rear++;
        Q[rear] = x;
        System.out.println(x + "Inserted");
    }
}
```

Queues

- Unlike arrays, where insertion and deletion can happen at any end.

- In Queues, insertion (Enqueue) and deletion (Dequeue) can happen at only one end each.

- Insertion happens at the rear end and deletion happens at the front

- Queues follow FIFO First in First out structure, i.e. the element added first (Enqueued) will go out of the queue first(Dequeued)

- Unlike stack, which follows, LIFO, last in first out, and stack where both insertion and deletion happens as one end.

Queue Operations

- Enqueue: Adding a new item to the Queue Data Structure, in other words, enqueuing new item to Stack DS.

If the Queue is full, then it is said to be in an overflow condition

- Dequeue: Removing an item from the Queue, i.e. dequeuing an item out.

If a Queue is empty then it is said to be in an underflow condition

- IsEmpty: This returns True If the Queue is empty else returns False

Representation of Queue

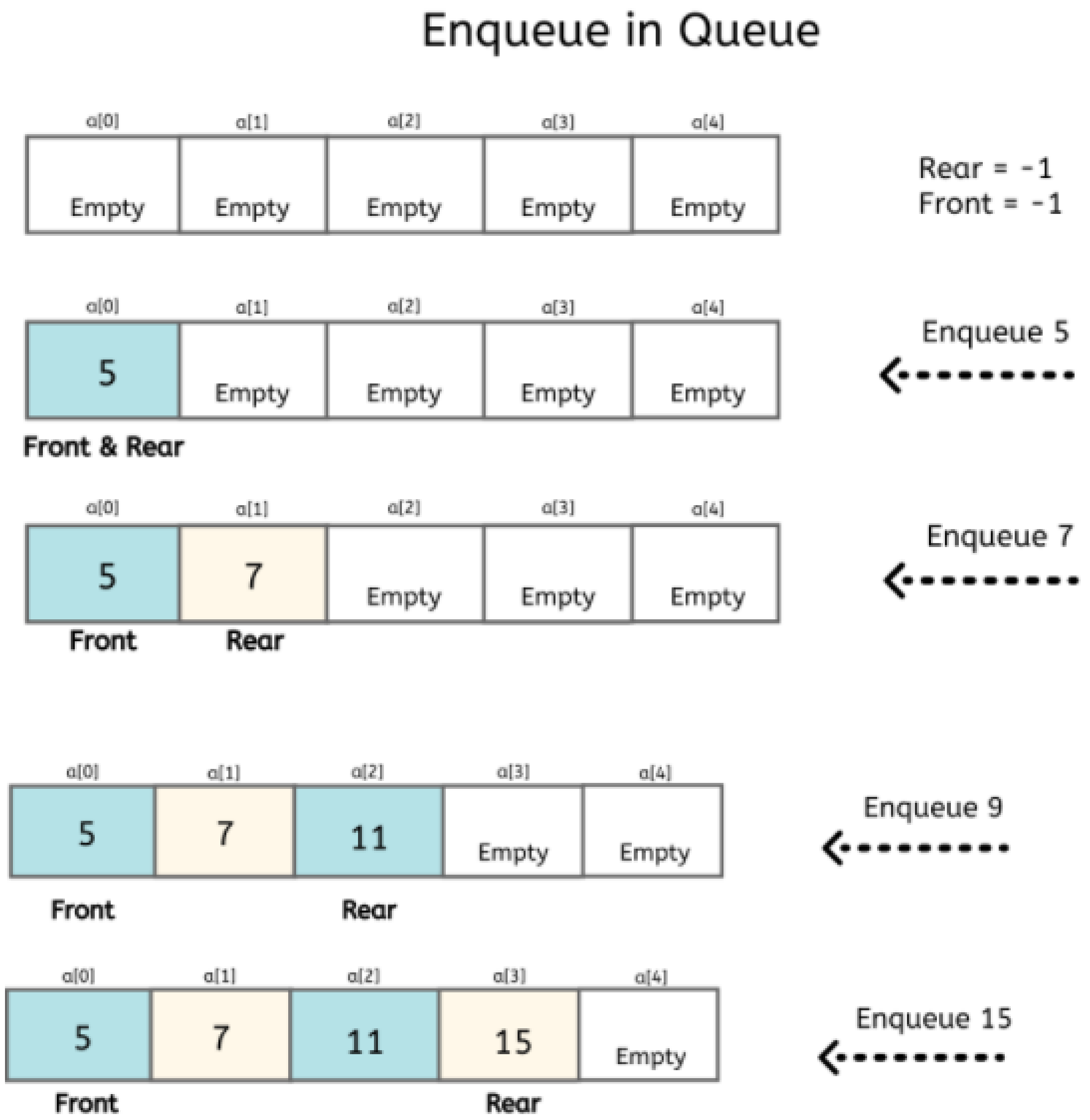
Queue as a data structure can be represented in two ways.

- Stack as an Array (Most popular)
- Stack as a struct (Popular)
- Stack as a Linked List.

1. Enqueue()

When we requirea to add an element to the Queue we perform Enqueue() operation.

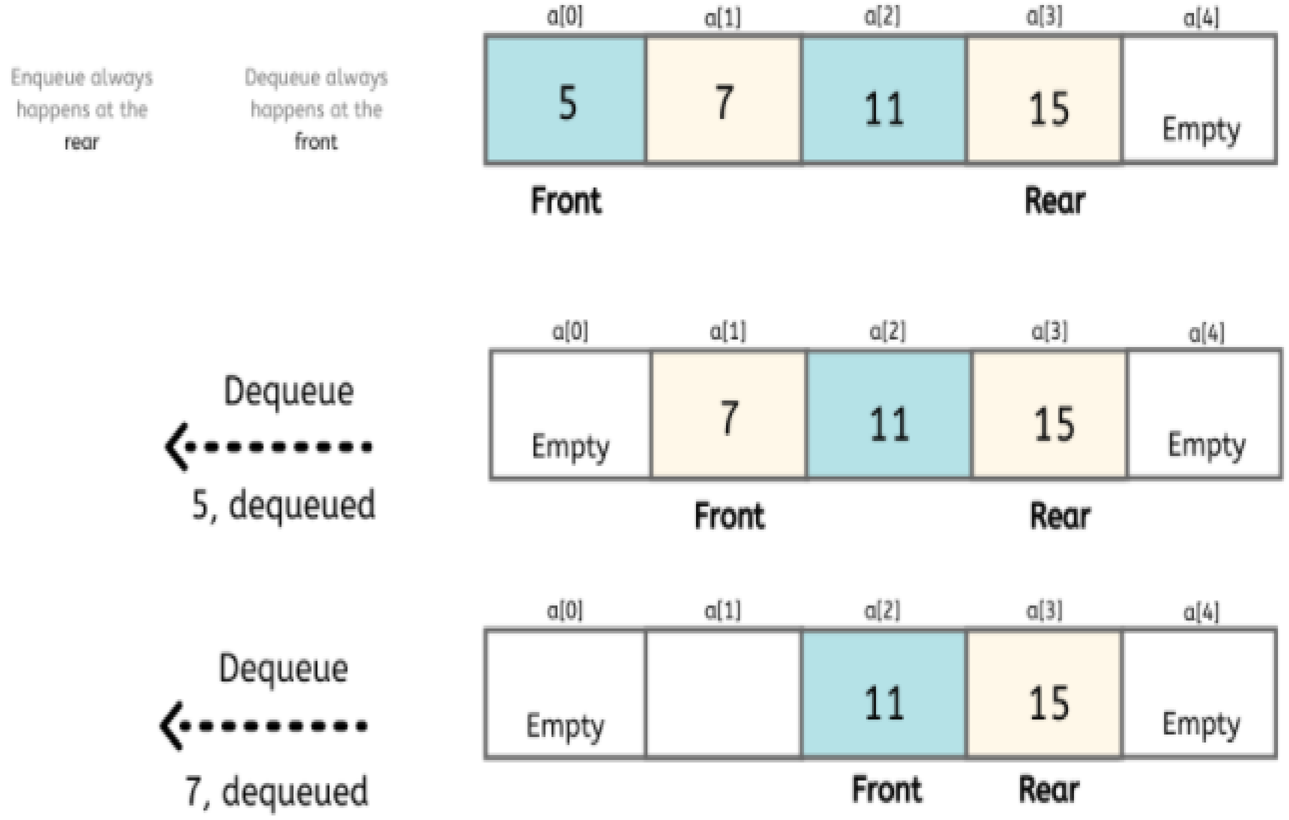
Push() operation is synonymous of insertion/addition in a data structure.



2. Dequeue()

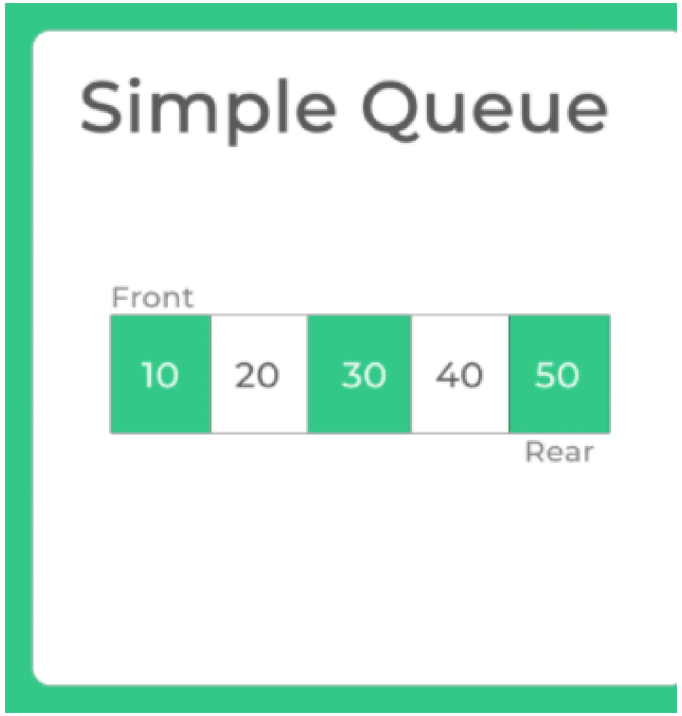
When we require to delete/remove an element to the Queue we perform Dequeue() operation.

Dequeue() operation is synonymous of deletion/removal in a data structure.



Simple Queue

A simple queue are the general queue that we use on perform insertion and deletion on FIFO basis i.e. the new element is inserted at the rear of the queue and an element is deleted from the front of the queue.



Applications of Simple Queue

The applications of simple queue are:-

- CPU scheduling
- Disk Scheduling
- Synchronization between two process.

Representation of Queue

Queue as a data structure can be represented in two ways.

- Stack as an Array (Most popular)
- Stack as a Linked List.

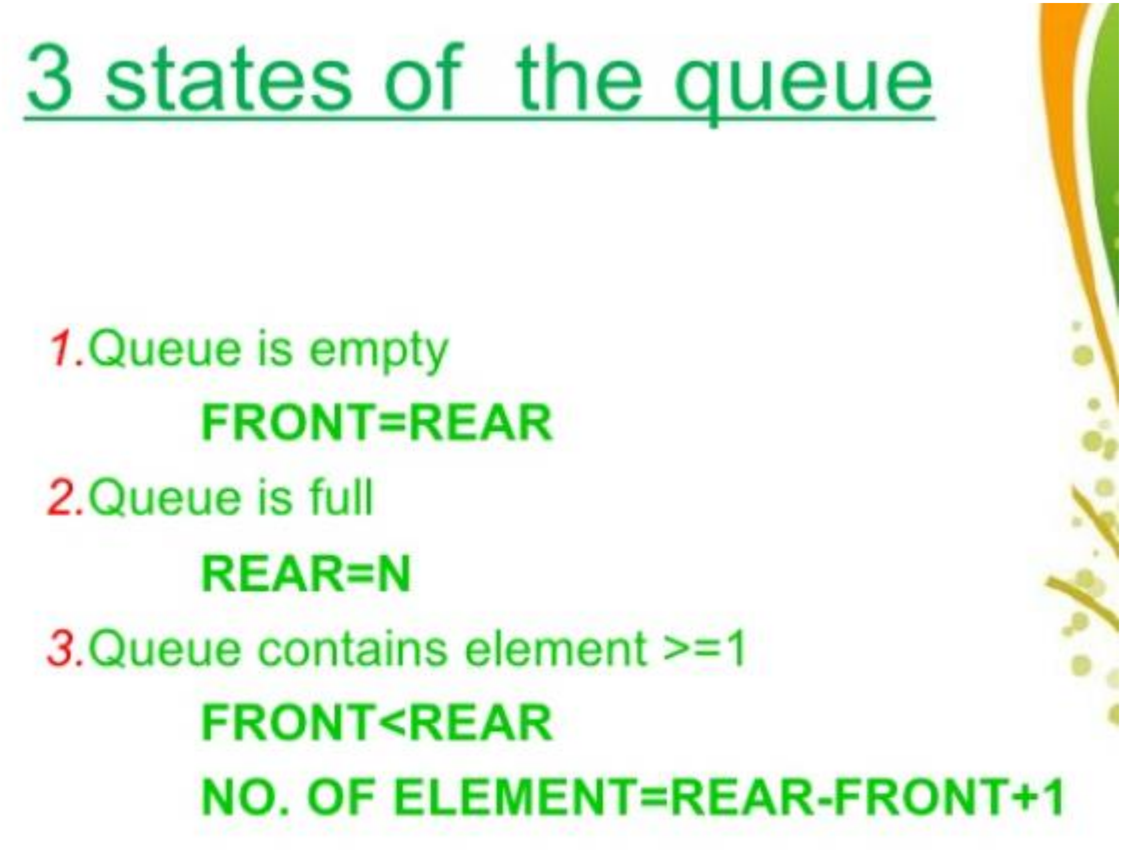
Applications and uses for Queues

- Heavily used in almost all applications of the operating system, to schedule processes, moving them in or out of process scheduler.

- FCFS, SJF etc

- Asynchronously i.e. when data resource may be the same but not received at the same rate.

- Anything that has to do with process and schedule, in the system or code.



3 states of the queue

1. Queue is empty

   FRONT=REAR

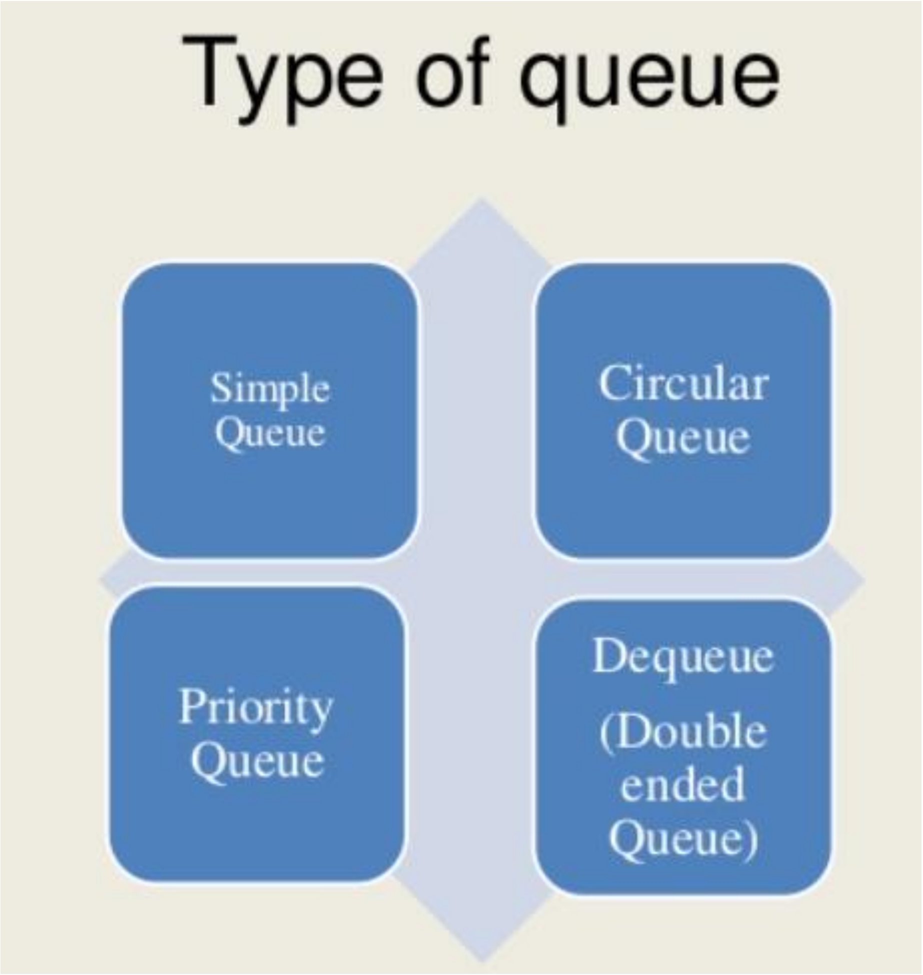
2. Queue is full

   REAR=N

3. Queue contains element >=1

   FRONT<REAR

   NO. OF ELEMENT=REAR-FRONT+1



Type of queue

Simple Queue

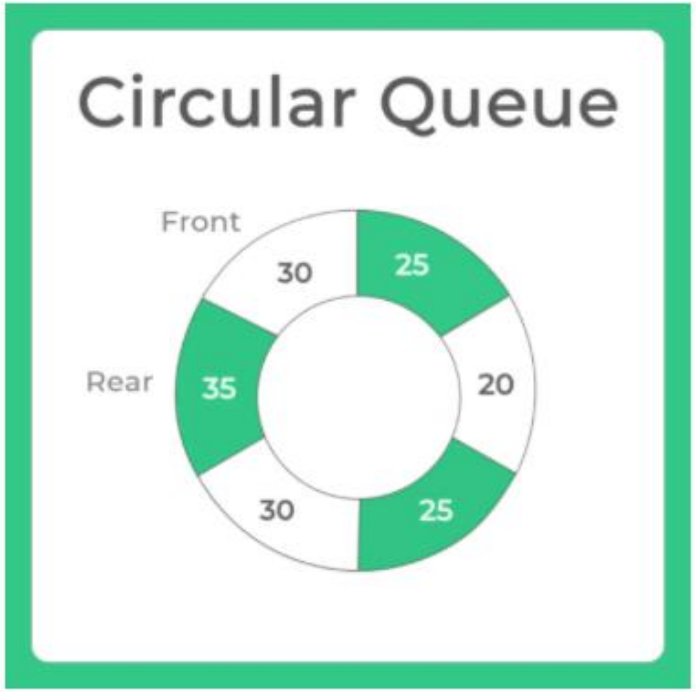
Circular Queue

Priority Queue

Dequeue (Double ended Queue)

Circular Queue

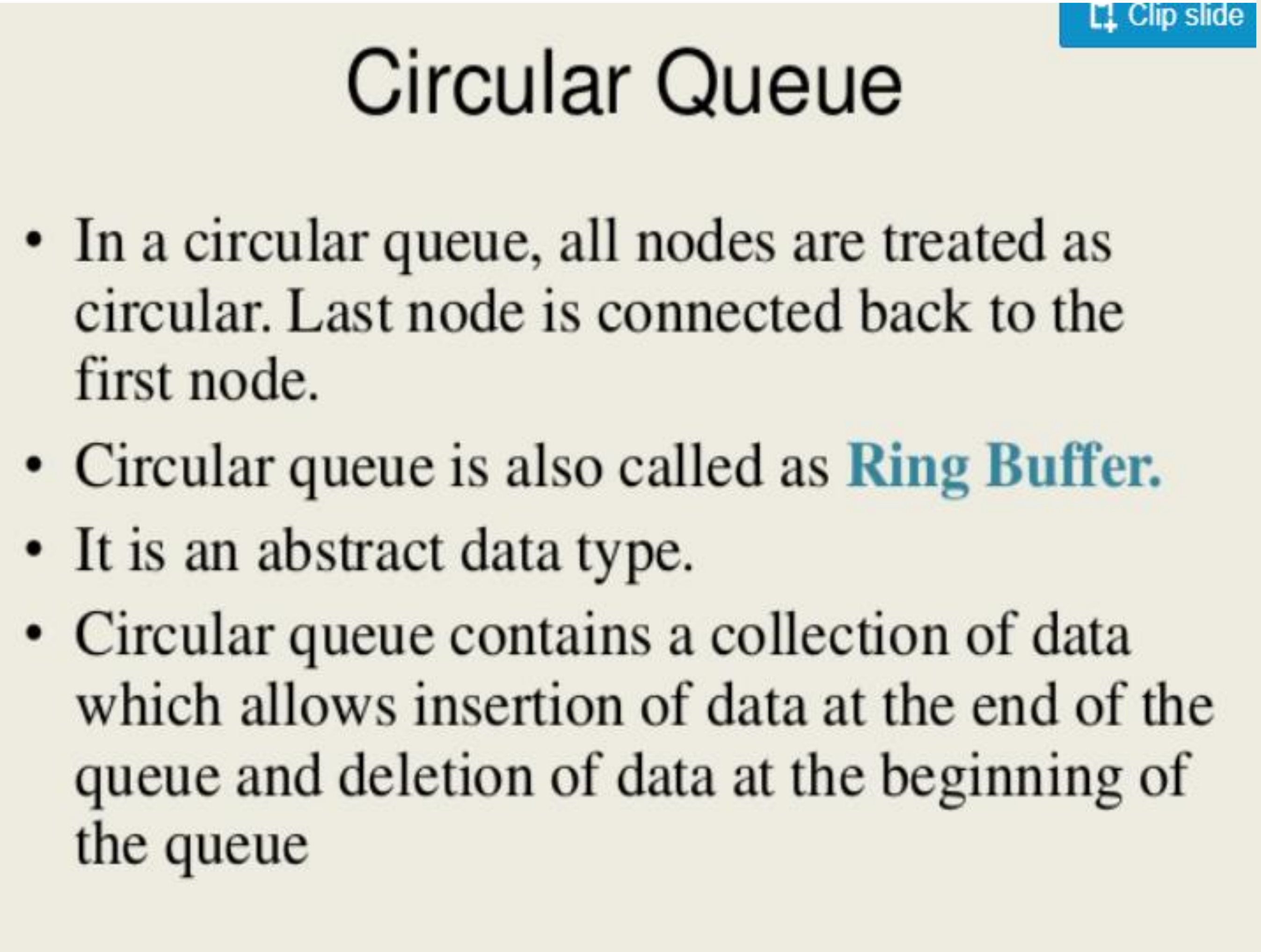
Circular queue is a type of queue in which all nodes are treated as circular such that the first node follows the last node. It is also called ring buffer. In this type of queue operations are performed on first in first out basis i.e the element that has inserted first will be one that will be deleted first.



Applications of Circular Queue

The applications of circular queue are:-

- CPU scheduling
- Memory management
- Traffic Management



Circular Queue

- In a circular queue, all nodes are treated as circular. Last node is connected back to the first node.
- Circular queue is also called as **Ring Buffer.**
- It is an abstract data type.
- Circular queue contains a collection of data which allows insertion of data at the end of the queue and deletion of data at the beginning of the queue

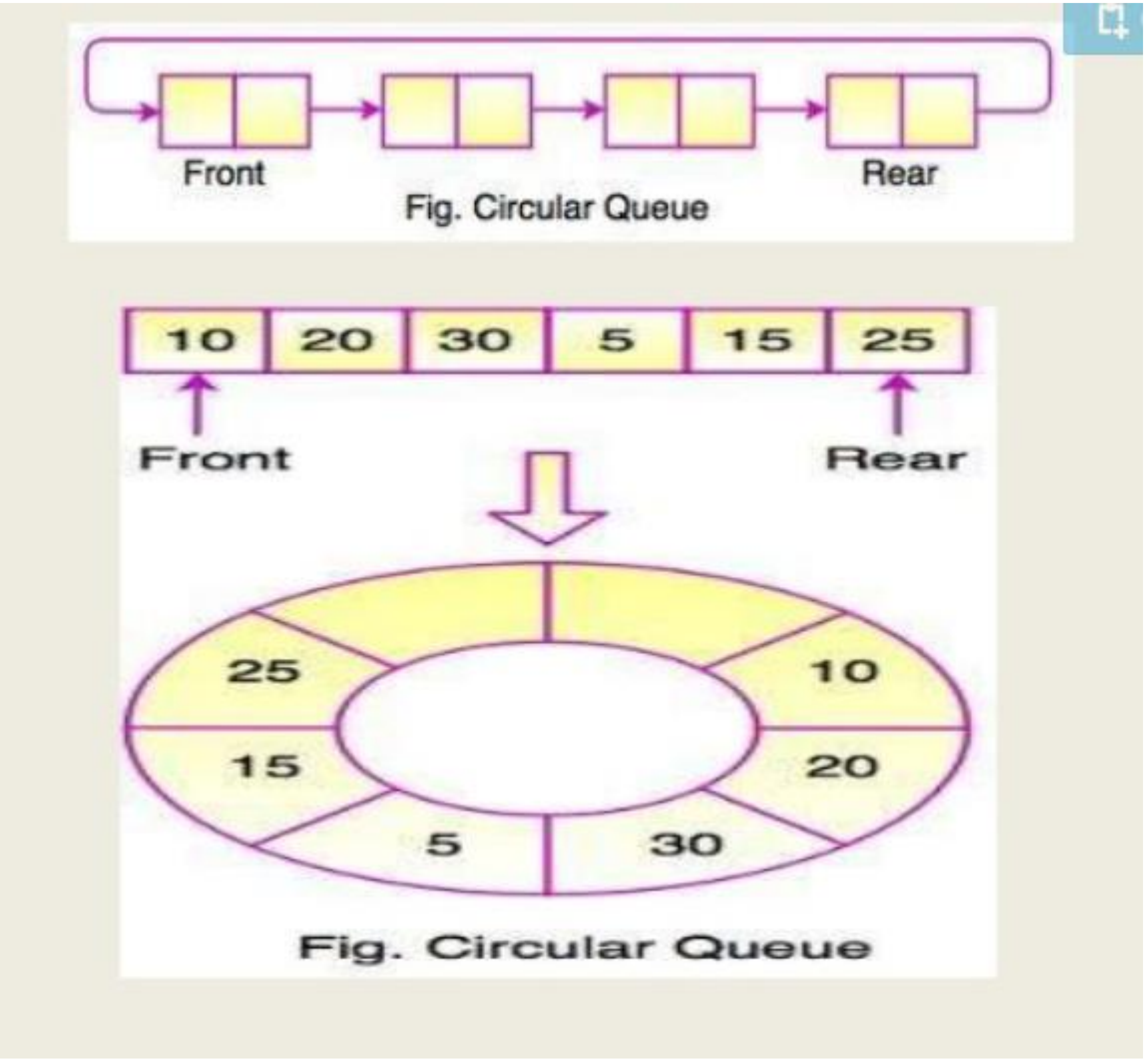
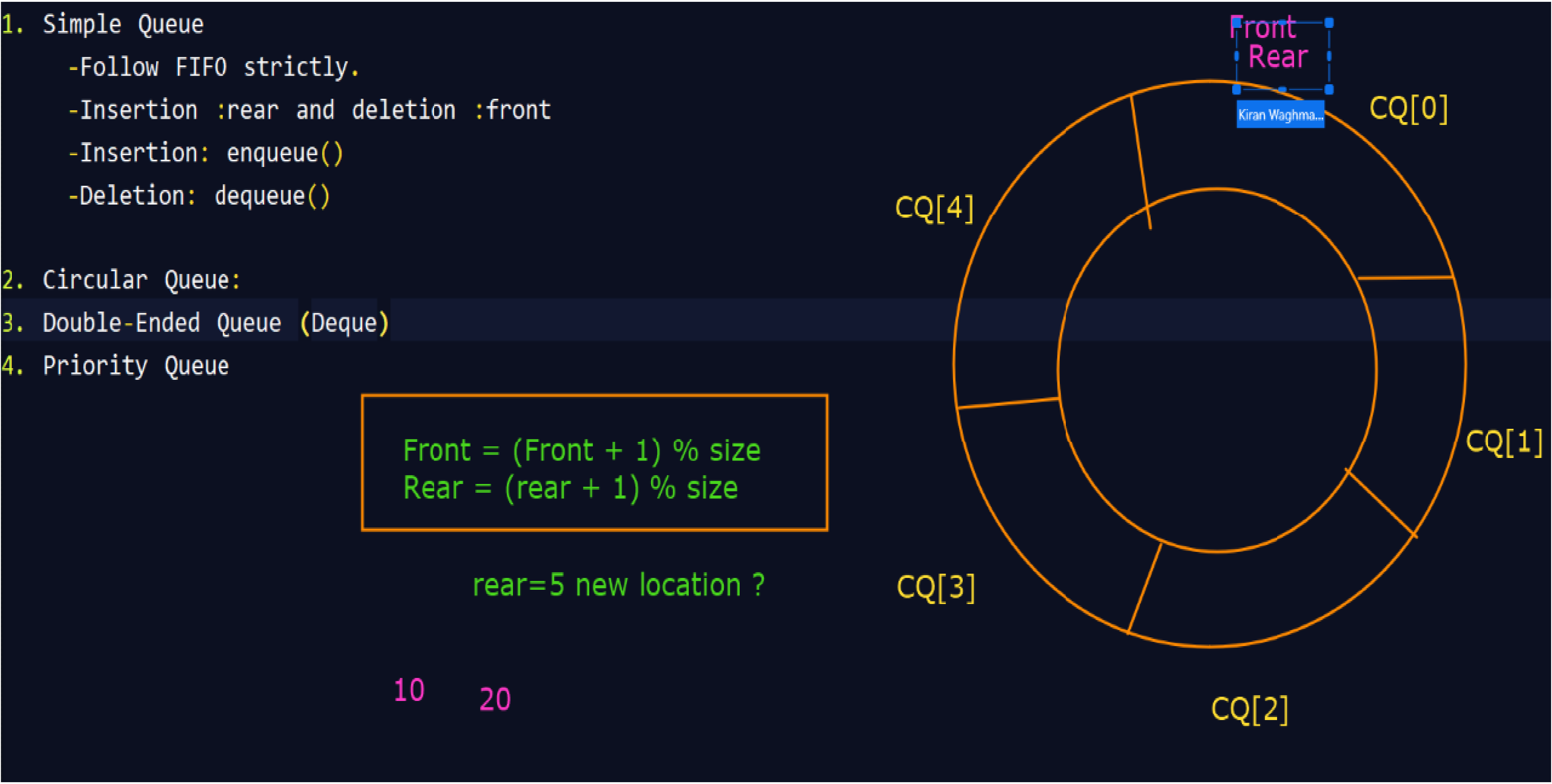


Fig. Circular Queue

Fig. Circular Queue



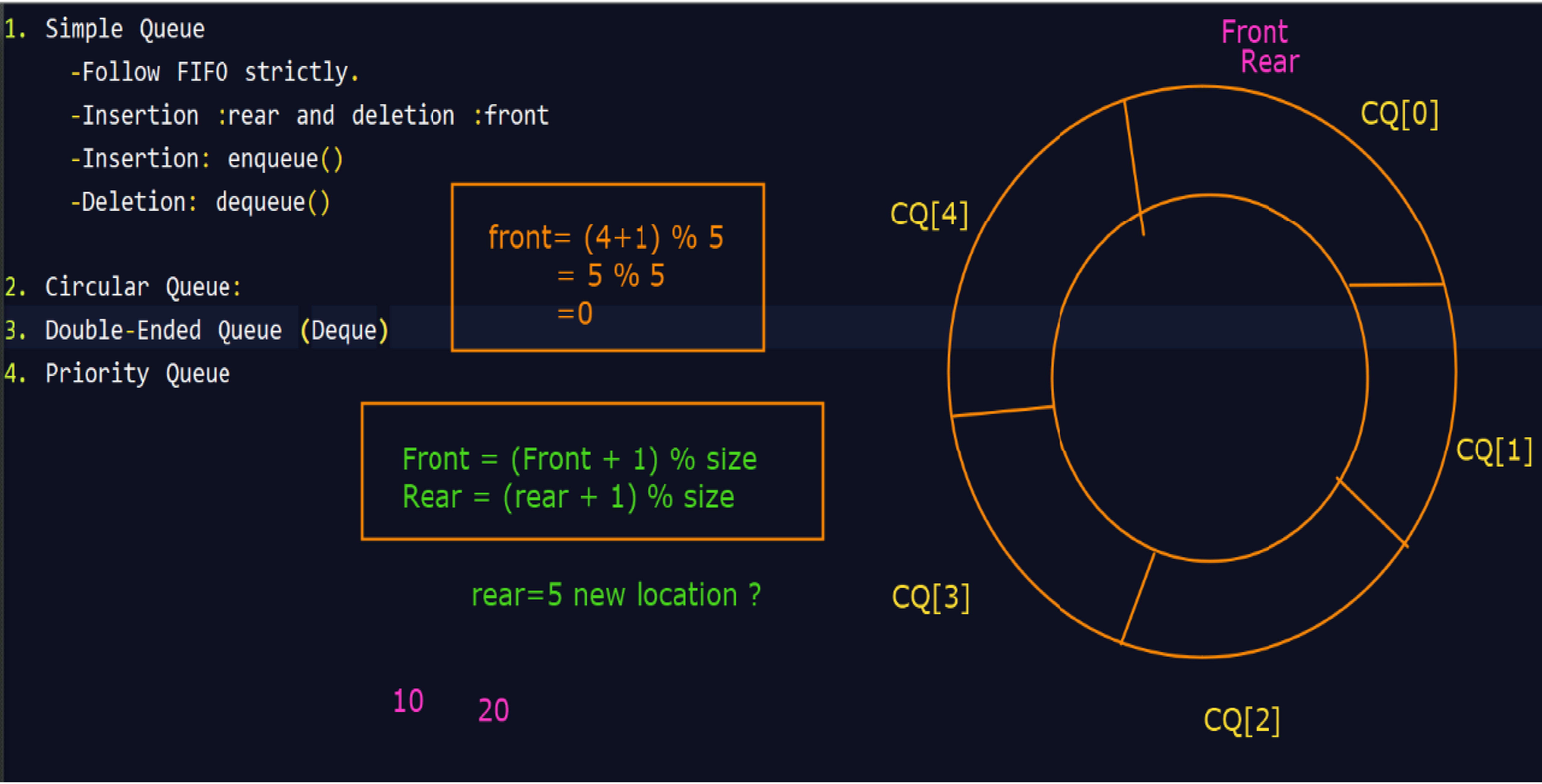
1. Simple Queue
   - -Follow FIFO strictly.
   - -Insertion :rear and deletion :front
   - -Insertion: enqueue()
   - -Deletion: dequeue()

2. Circular Queue:
3. Double-Ended Queue (Deque)
4. Priority Queue

Front = (Front + 1) % size
Rear = (rear + 1) % size

rear=5 new location ?

10 20



```
1. Simple Queue
     -Follow FIFO strictly.
     -Insertion :rear and deletion :front
     -Insertion: enqueue()
     -Deletion: dequeue()

2. Circular Queue:
3. Double-Ended Queue (Deque)
4. Priority Queue
```

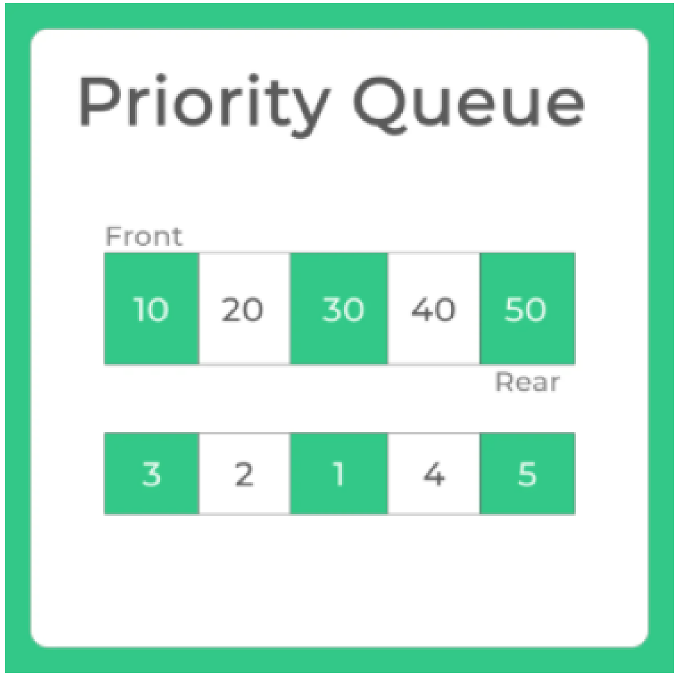
front= (4+1) % 5
= 5 % 5
=0

Front = (Front + 1) % size
Rear = (rear + 1) % size

rear=5 new location ?

Priority Queue

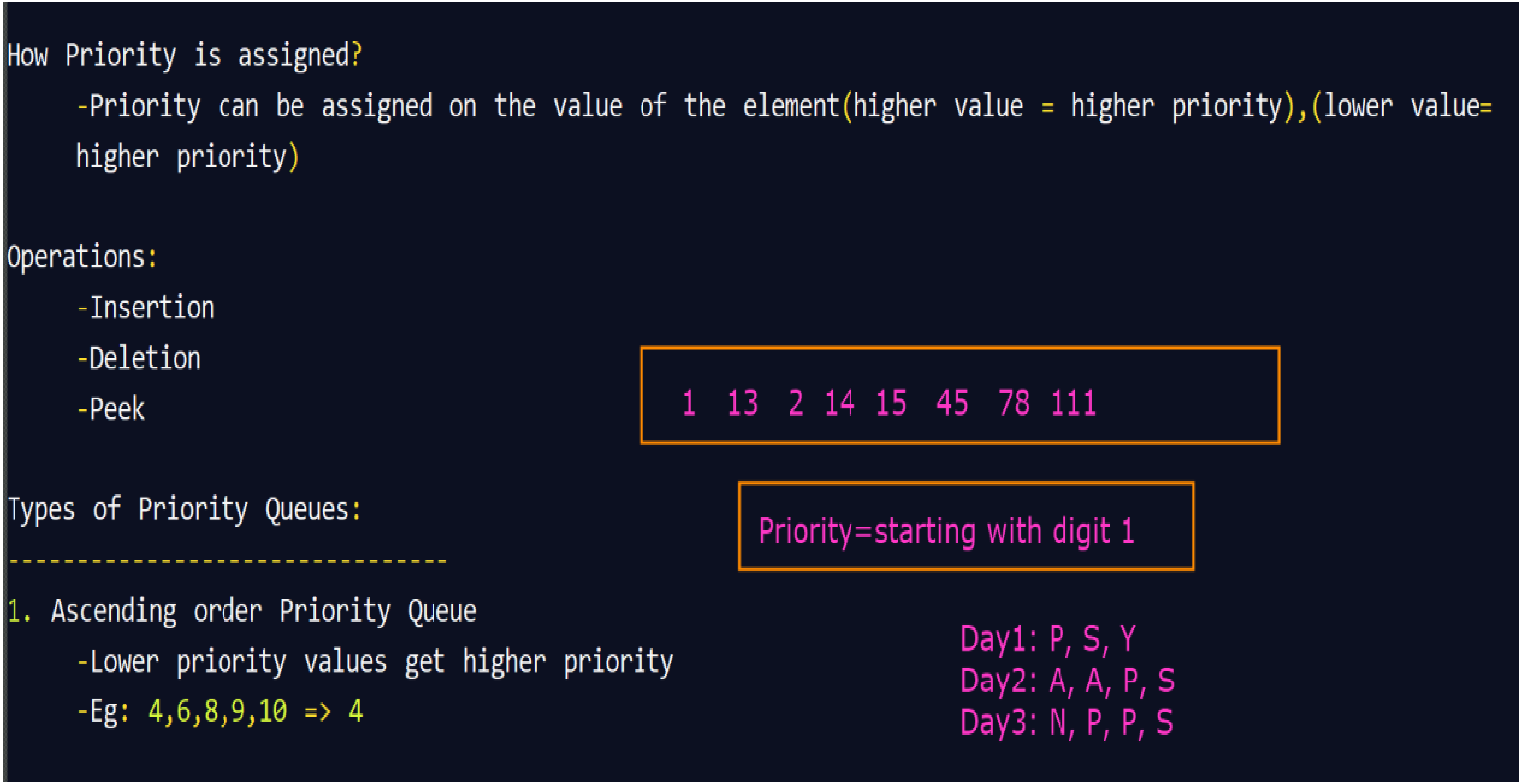
Priority Queue is a special type of queue in which elements are treated according to their priority. Insertion of an element take place at the rear of the queue but the deletion or removal of an element take place according to the priority of the element. Element with the highest priority is removed first and element with the lowest priority is removed last.



Applications of Priority Queue

The applications of priority queue are:-

- Dijkstra's shortest path algorithm
- Data compression in huffman codes.
- Load balancing and interrupt handling in operating system.
- Sorting heap.



```
How Priority is assigned?
     -Priority can be assigned on the value of the element(higher value = higher priority),(lower value=
     higher priority)

Operations:
     -Insertion
     -Deletion
     -Peek

Types of Priority Queues:
--------------------------------
1. Ascending order Priority Queue
     -Lower priority values get higher priority
     -Eg: 4,6,8,9,10 => 4
```

1 13 2 14 15 45 78 111

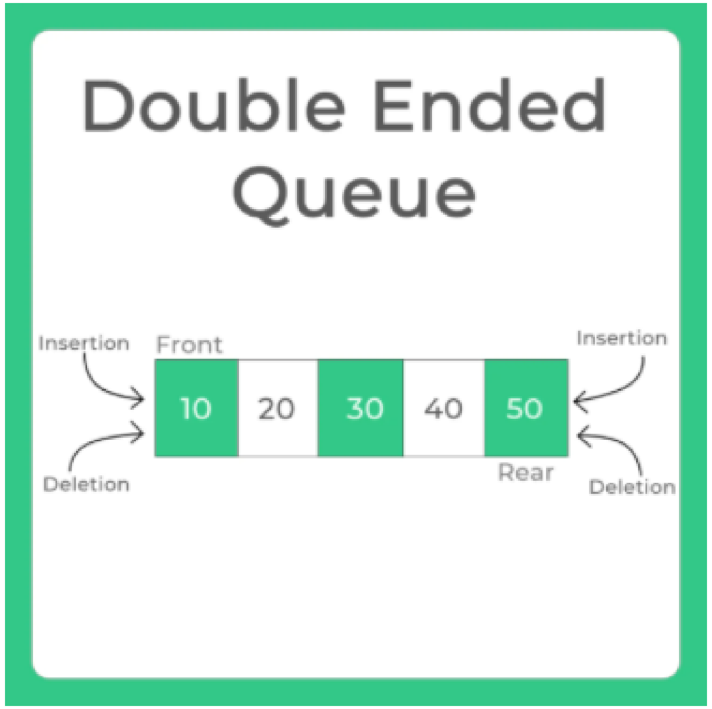
Priority=starting with digit 1

Day1: P, S, Y
Day2: A, A, P, S
Day3: N, P, P, S

Double Ended Queue

Double ended queue are also known as deque. In this type of queue insertion and deletion of an element can take place at both the ends. Further deque is divided into two types:-

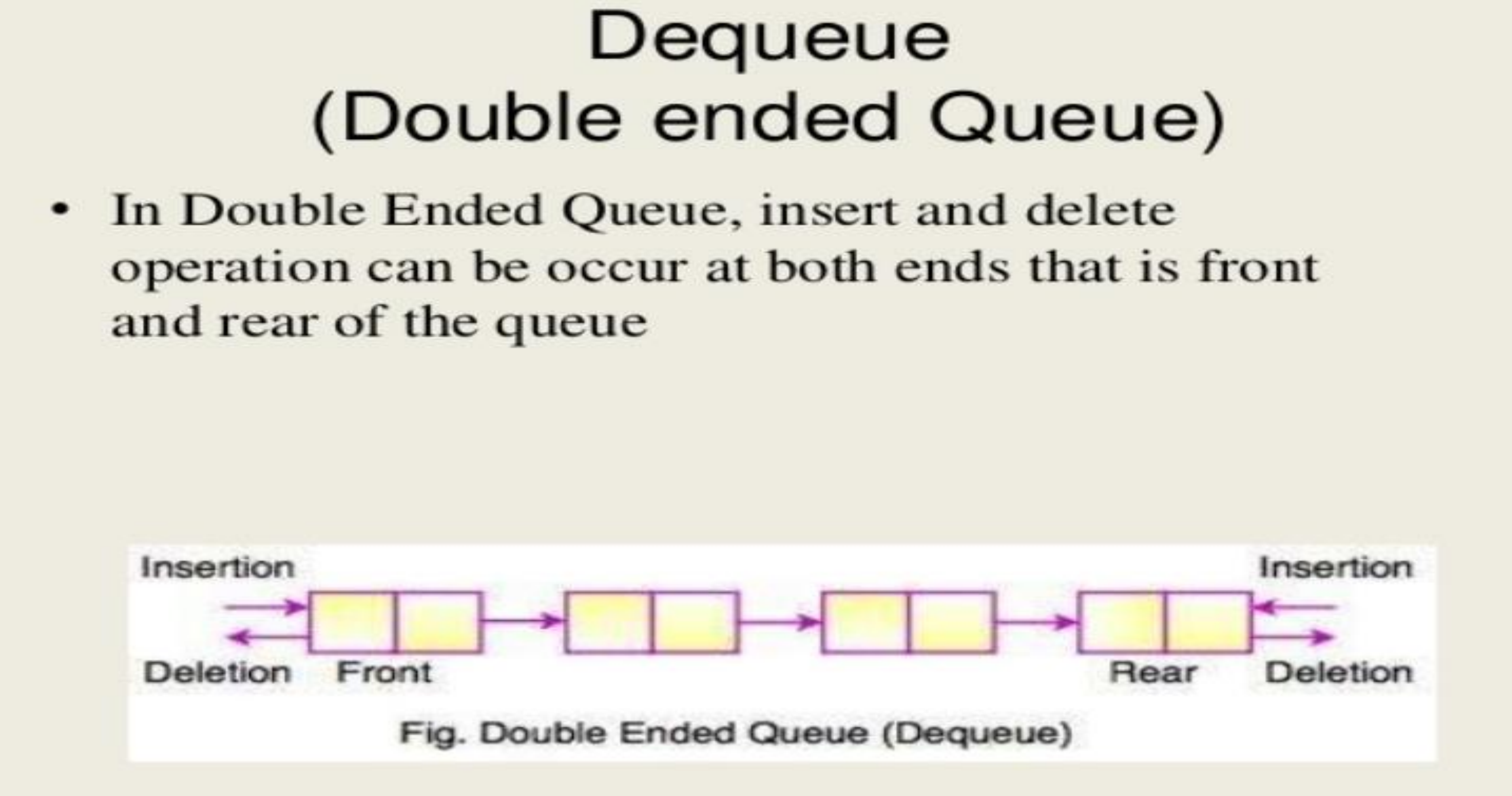
- Input Restricted Deque :- In this, input is blocked at a single end but allows deletion at both the ends.
- Output Restricted Deque :- In this, output is blocked at a single end but allows insertion at both the ends.



Applications of Double Ended Queue

The applications of double ended queue are:-

- To execute undo and redo operation.
- For implementing stacks.
- Storing the history of web browsers.



Dequeue (Double ended Queue)

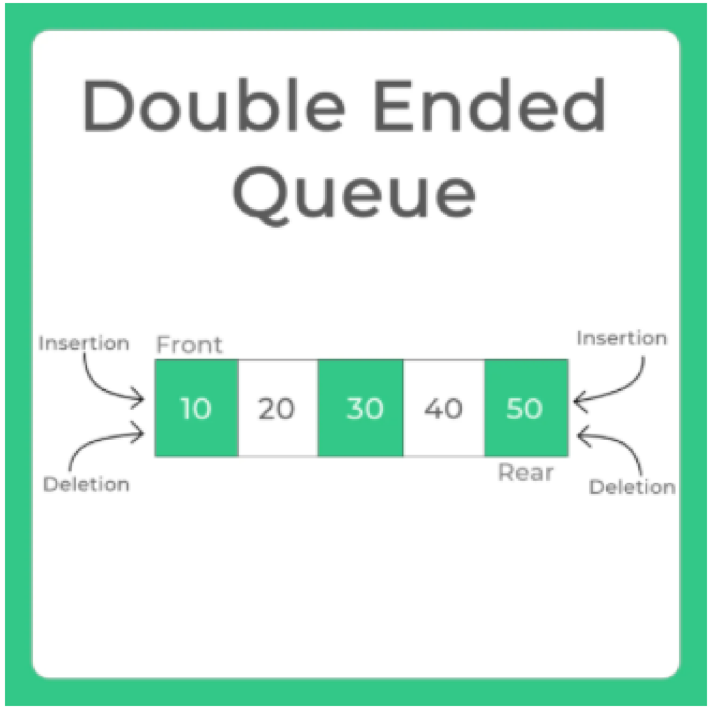
- In Double Ended Queue, insert and delete operation can be occur at both ends that is front and rear of the queue

Fig. Double Ended Queue (Dequeue)

Double Ended Queue

Double ended queue are also known as deque. In this type of queue insertion and deletion of an element can take place at both the ends. Further deque is divided into two types:-

- Input Restricted Deque :- In this, input is blocked at a single end but allows deletion at both the ends.
- Output Restricted Deque :- In this, output is blocked at a single end but allows insertion at both the ends.



Applications of Double Ended Queue

The applications of double ended queue are:-

- To execute undo and redo operation.
- For implementing stacks.
- Storing the history of web browsers.

Heap

Module I

Kiran Waghmare

Definition in Data Structure

- **Heap:**
  - A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).
- **Max-Heap:**
  - root node has the largest key. A max tree is a tree in which the key value in each node is no smaller than the key values in its children. A max heap is a complete binary tree that is also a max tree.
- **Min-Heap:**
  - root node has the smallest key. A min tree is a tree in which the key value in each node is no larger than the key values in its children. A min heap is a complete binary tree that is also a min tree.
- **Complete Binary Tree:**
  - A complete binary tree is a binary tree in which every level, *except possibly the last*, **is completely filled, and all nodes are as far left as possible**

Heap

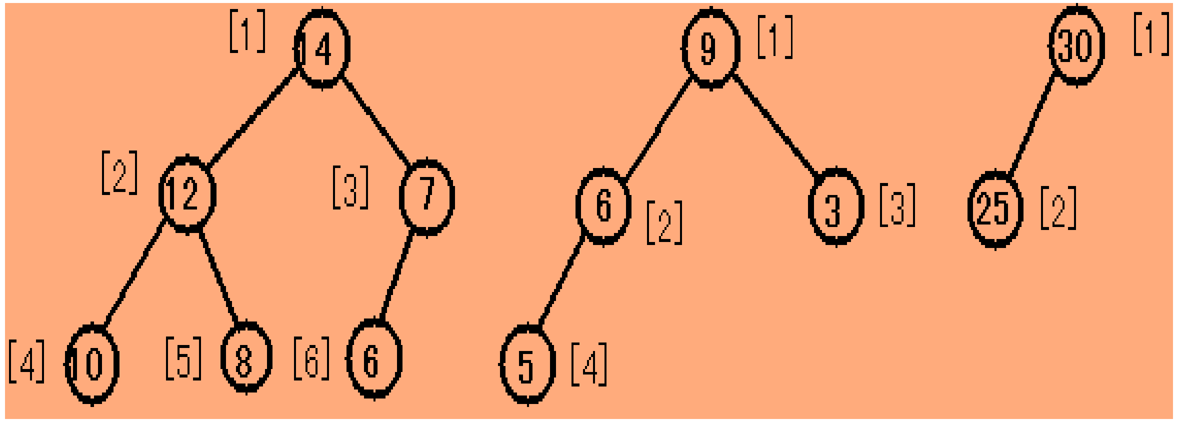
- **Definition in Data Structure**
  - **Heap:** A special form of complete binary tree that key value of each node is no smaller (larger) than the key value of its children (if any).

- **Max-Heap: root node has the largest key.**
  - A *max tree* is a tree in which the key value in each node is no smaller than the key values in its children. A *max heap* is a complete binary tree that is also a max tree.

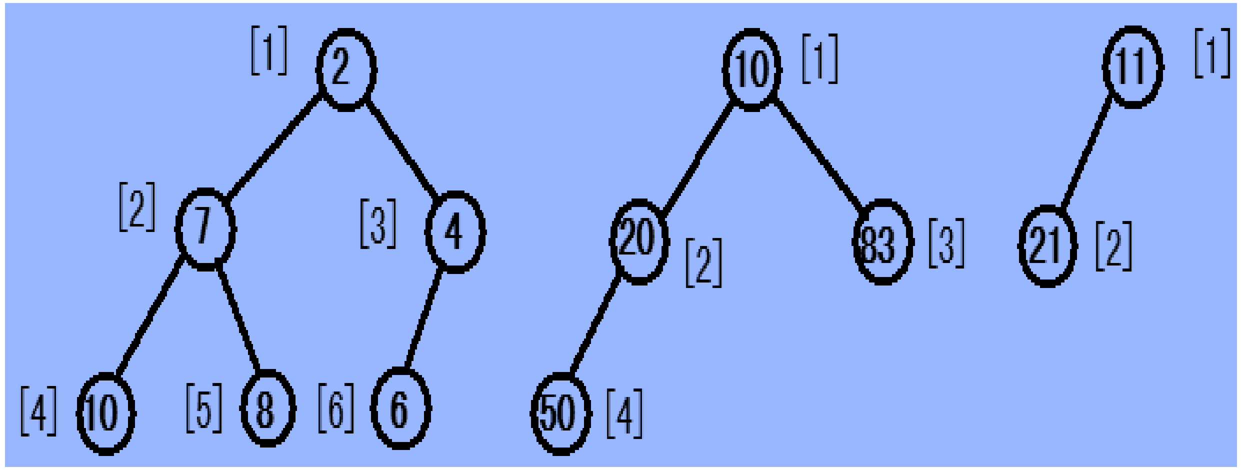
- **Min-Heap: root node has the smallest key.**
  - A *min tree* is a tree in which the key value in each node is no larger than the key values in its children. A *min heap* is a complete binary tree that is also a min tree.

Heap

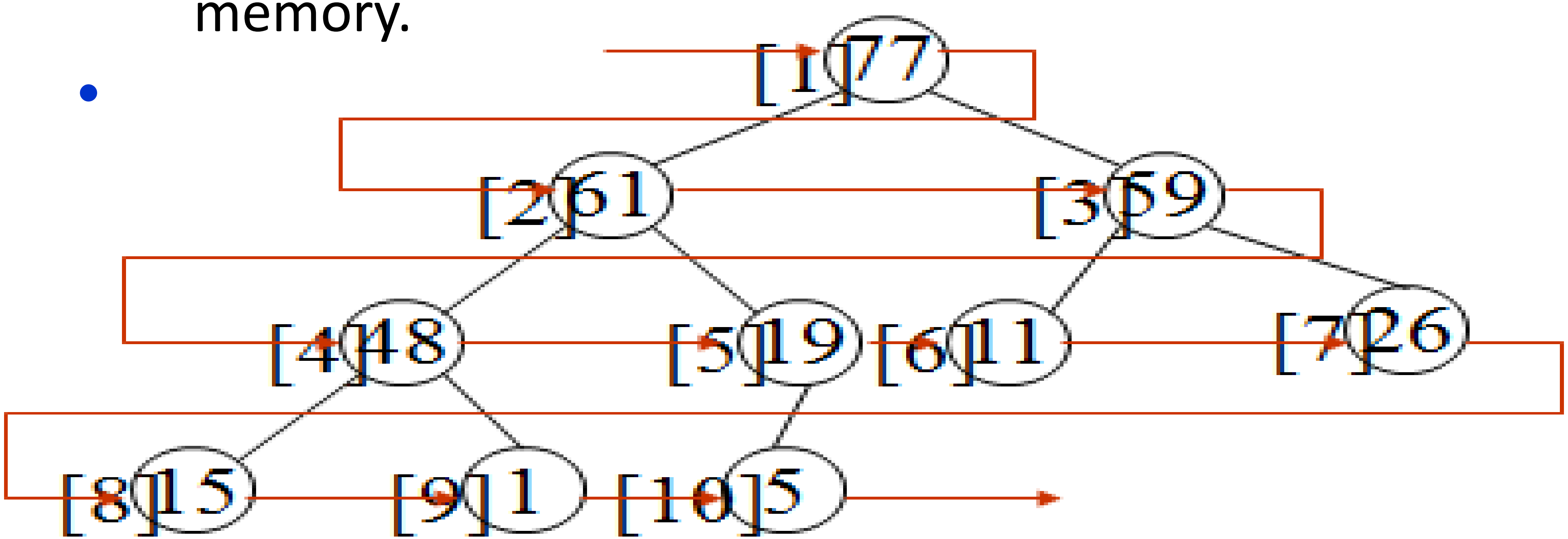
- **Example:**
  - Max-Heap



  - Min-Heap



- **Note:**
  - Heap in data structure is a complete binary tree!
    - **(Nice representation in Array)**
  - Heap in C program environment is an array of memory.
-



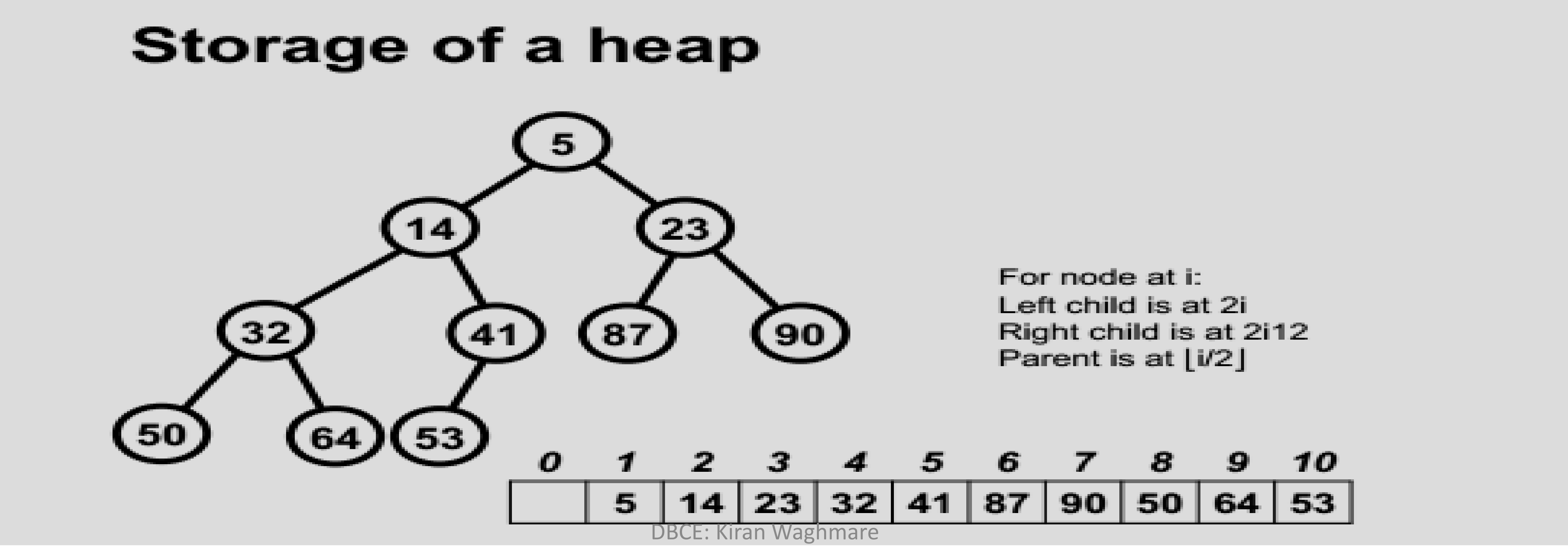
– Stored using array in C

index	1	2	3	4	5	6	7	8	9	10
value	77	61	59	48	19	11	26	15	1	5

Heap

For any node n in position i :

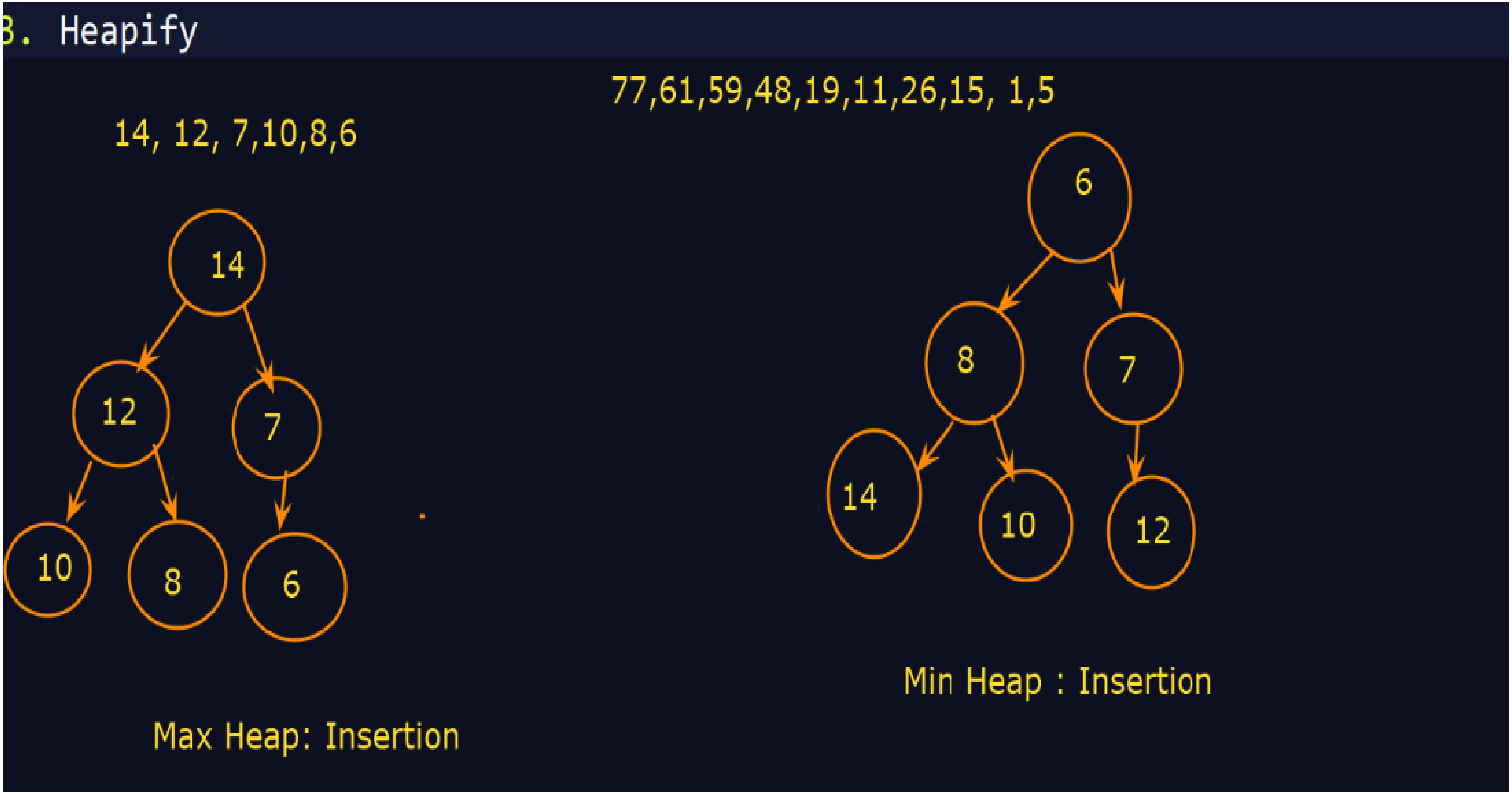
1. LeftChild(i) : return 2i
2. RightChild(i) : return 2i+1
3. Parent(i) : return i/2



Storage of a heap

For node at i:
Left child is at 2i
Right child is at 2i12
Parent is at ⌊i/2⌋

0	1	2	3	4	5	6	7	8	9	10
	5	14	23	32	41	87	90	50	64	53



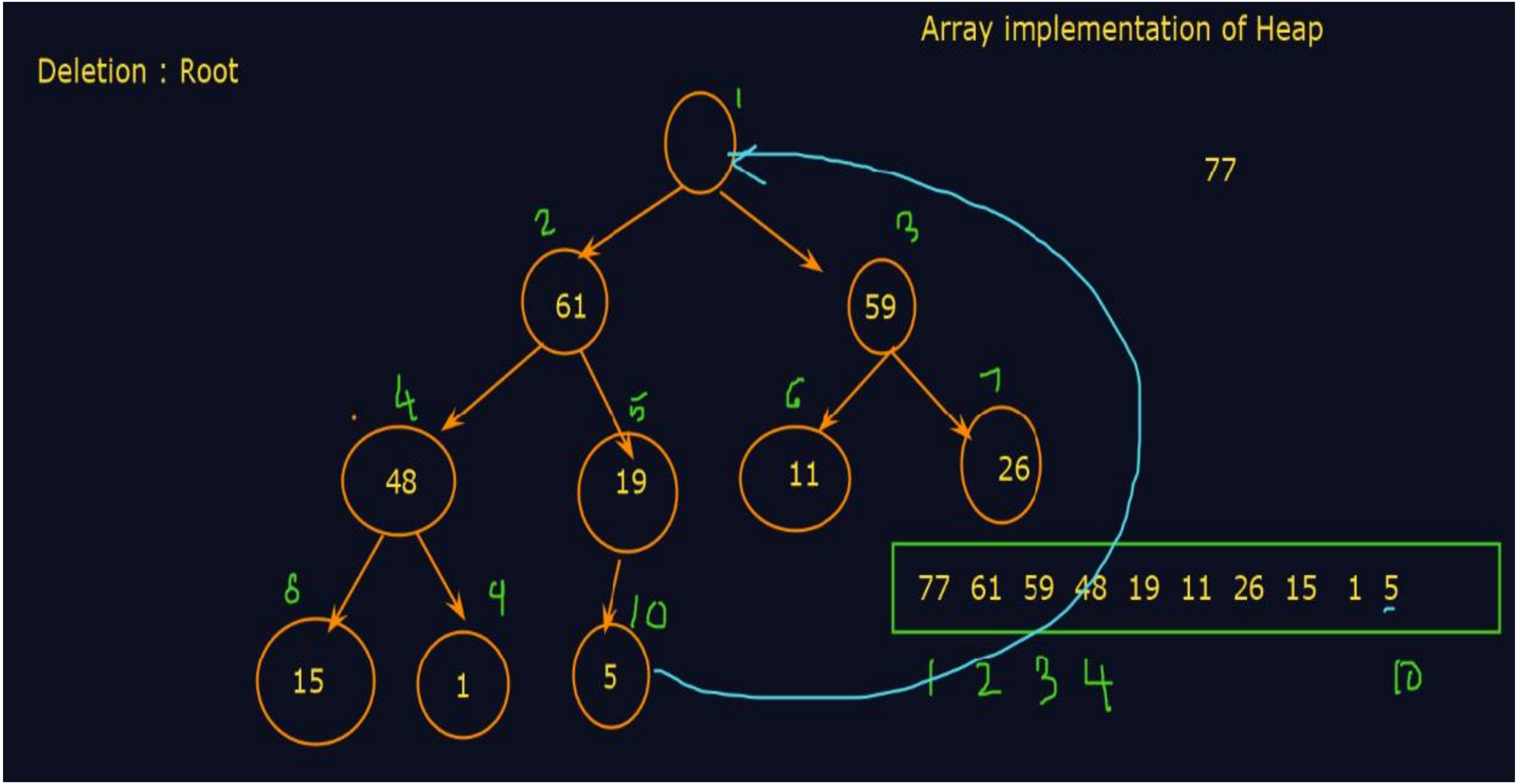
3. Heapify

77,61,59,48,19,11,26,15, 1,5

14, 12, 7,10,8,6

Min Heap : Insertion

Max Heap: Insertion



Array implementation of Heap

Deletion : Root

77

77	61	59	48	19	11	26	15	1	5
1	2	3	4						10

Heap

- **Operations**
  - Creation of an empty heap
  - **Insertion** of a new element into the heap
  - **Deletion** of the largest(smallest) element from the heap

  - Heap is **complete binary tree**, can be represented by **array.**
- **So the complexity of**
  - inserting any node or
  - deleting the root node

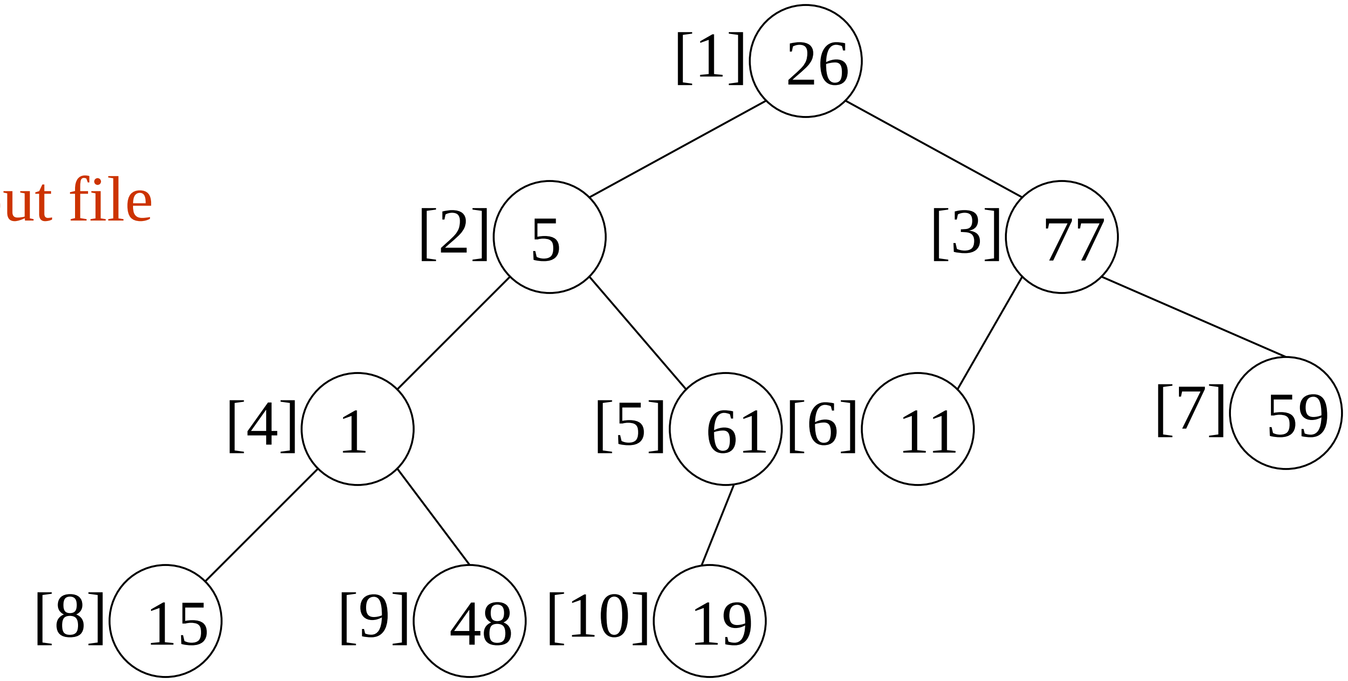
  from Heap is $\left.\begin{matrix} \\ \\ \\ \end{matrix}\right\}$ O(height) = O($\log_2 n$).

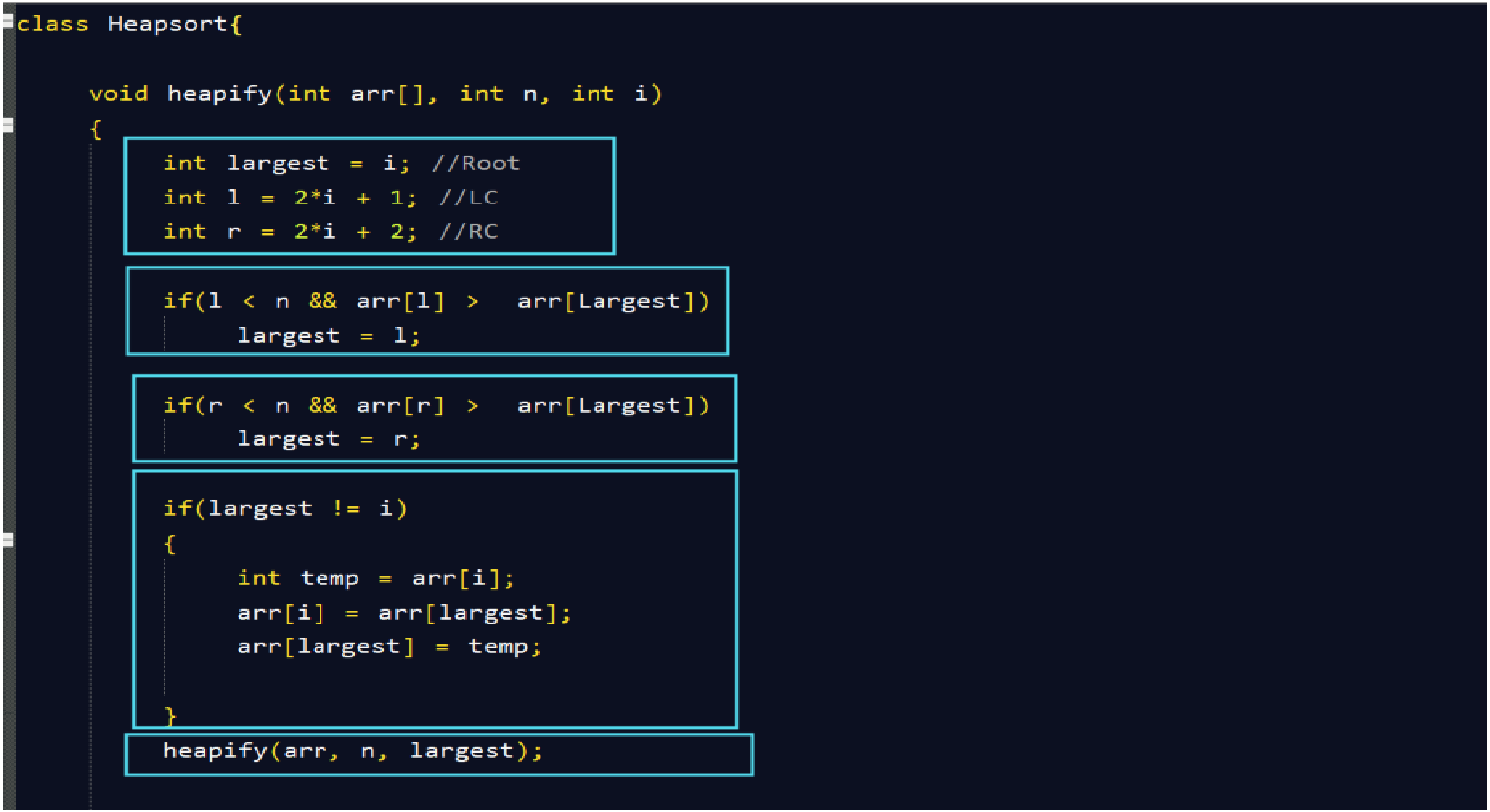
Application On Sorting: Heap Sort

- **See an illustration first**
  - Array interpreted as a binary tree

1	2	3	4	5	6	7	8	9	10
26	5	77	1	61	11	59	15	48	19

input file





```
class Heapsort{

    void heapify(int arr[], int n, int i)
    {
        int largest = i; //Root
        int l = 2*i + 1; //LC
        int r = 2*i + 2; //RC

        if(l < n && arr[l] >  arr[Largest])
            largest = l;

        if(r < n && arr[r] >  arr[Largest])
            largest = r;

        if(largest != i)
        {
            int temp = arr[i];
            arr[i] = arr[largest];
            arr[largest] = temp;

        }
        heapify(arr, n, largest);
```

Thanks

Thanks