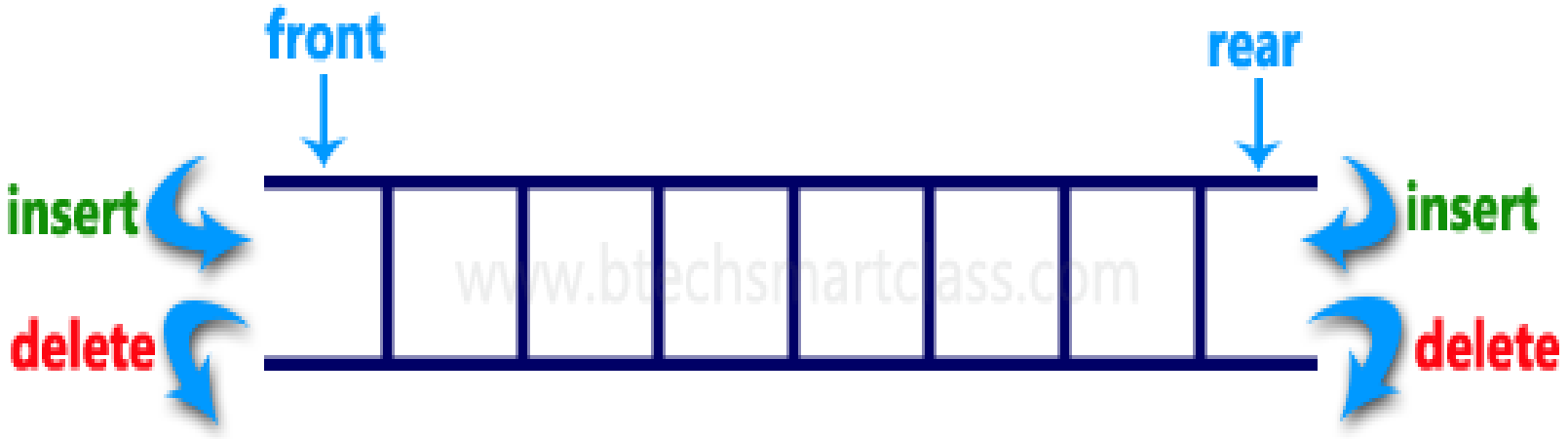
Data Structures – CST 201
Module - 2

Syllabus

- Polynomial representation using Arrays
- Sparse matrix
- Stacks
  - Evaluation of Expressions
- Queues
  - Circular Queues
  - Priority Queues
  - **Double Ended Queues**
- Linear Search
- Binary Search

DOUBLE ENDED QUEUE(DEQUE)

- Both insertion and deletion operation can be performed on both ends



- DEQUE can be used as a stack as well as queue

DEQUE- Various States

1. Deque is Empty: FRONT=-1 & REAR=-1
2. Deque is Full: FRONT = 0 and REAR=SIZE-1
3. Deque contains only one element: FRONT=REAR
4. Total elements in the Deque: REAR-FRONT+1

DEQUE- Operations

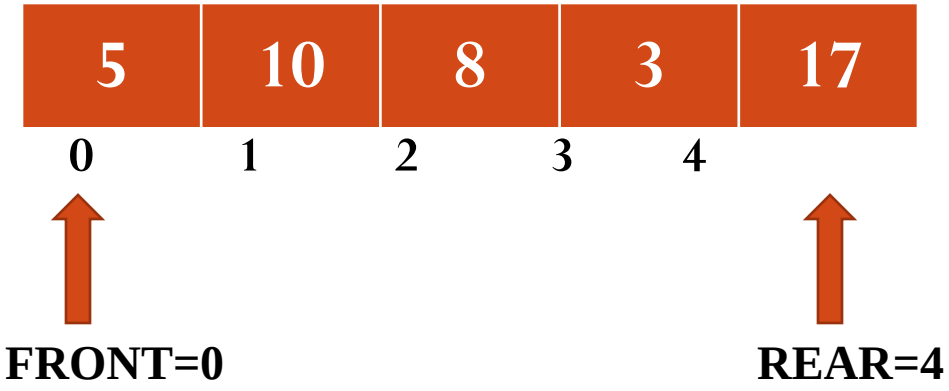
- **PUSH**: ENQUEUE at the FRONT end
- **POP**: DEQUEUE from the FRONT end
- **INJECT:** ENQUEUE at the REAR end
- **EJECT:** DEQUEUE from the REAR end
- **DISPLAY**: Display the contents of the Deque

DEQUE – PUSH

PUSH_DQ(10)

Case 1: FRONT=0 and REAR=SIZE-1

Deque is FULL

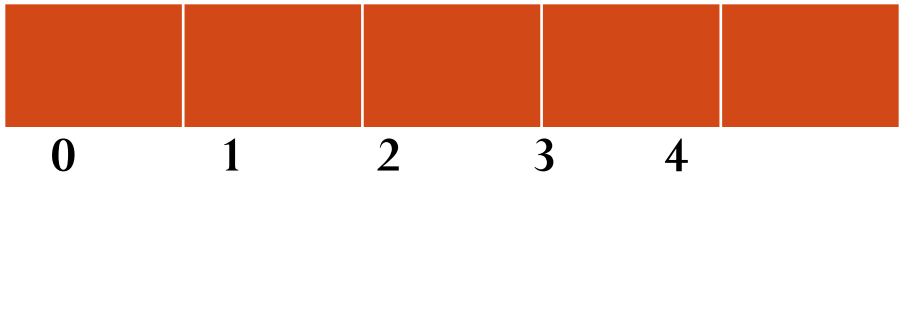


PUSH_DQ(10)

Case 2: FRONT=-1 Or REAR=-1

FRONT=REAR=0

A[FRONT]=10



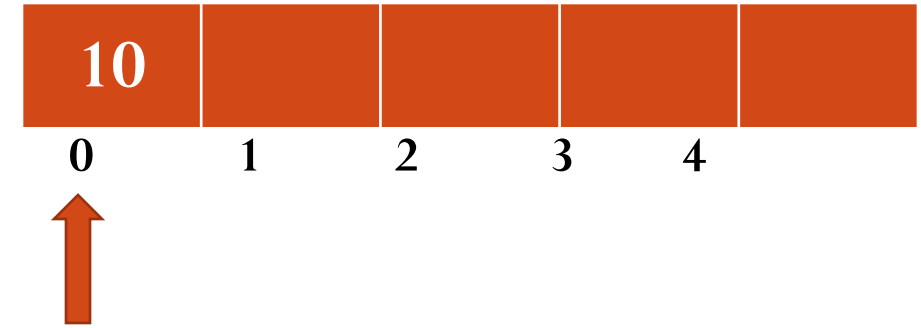
FRONT=-1
REAR=-1

PUSH_DQ(10)

Case 2: FRONT=-1 Or REAR=-1

FRONT=REAR=0

A[FRONT]=10



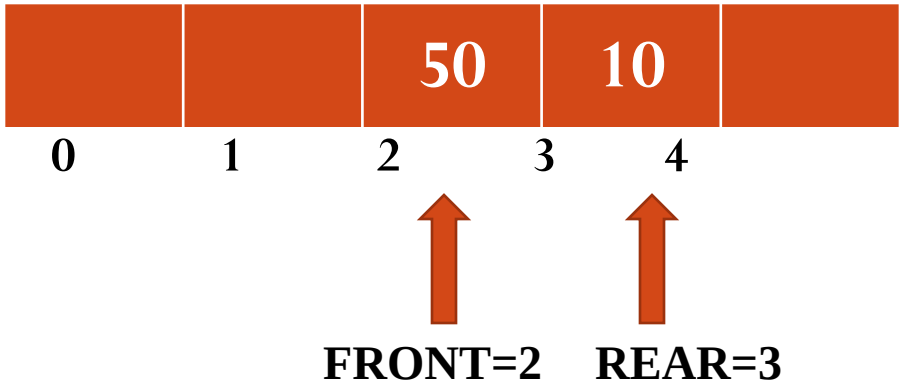
FRONT=0
REAR=0

PUSH_DQ(10)

Case 3: FRONT>0

FRONT=FRONT-1

A[FRONT]=10

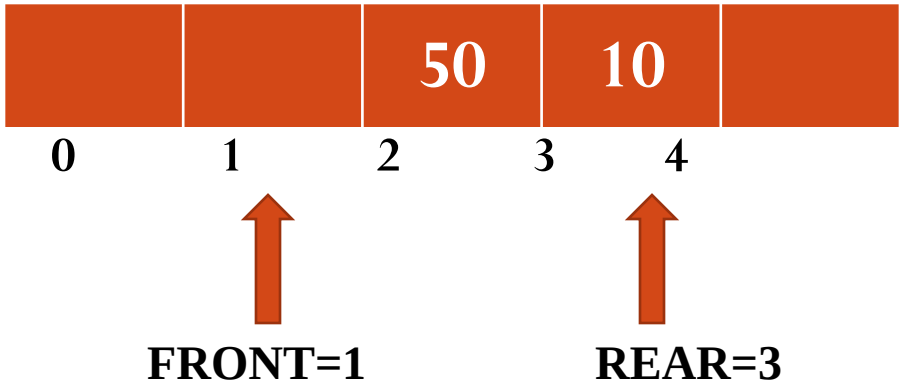


PUSH_DQ(10)

Case 3: FRONT>0

FRONT=FRONT-1

A[FRONT]=10

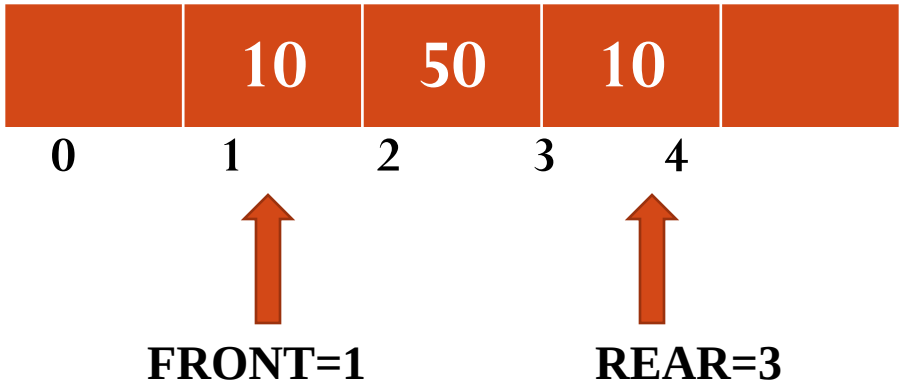


PUSH_DQ(10)

Case 3: FRONT>0

FRONT=FRONT-1

A[FRONT]=10



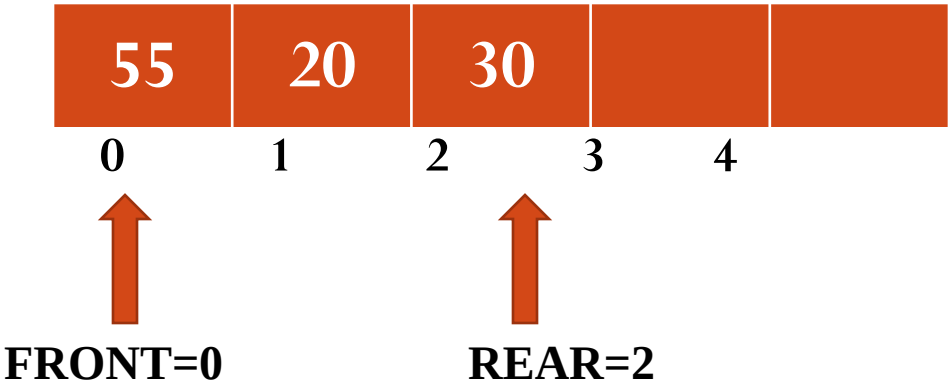
PUSH_DQ(10)

Case 4: No space in FRONT but space in REAR

Shift elements one position to right

A[FRONT]=10

REAR=REAR+1



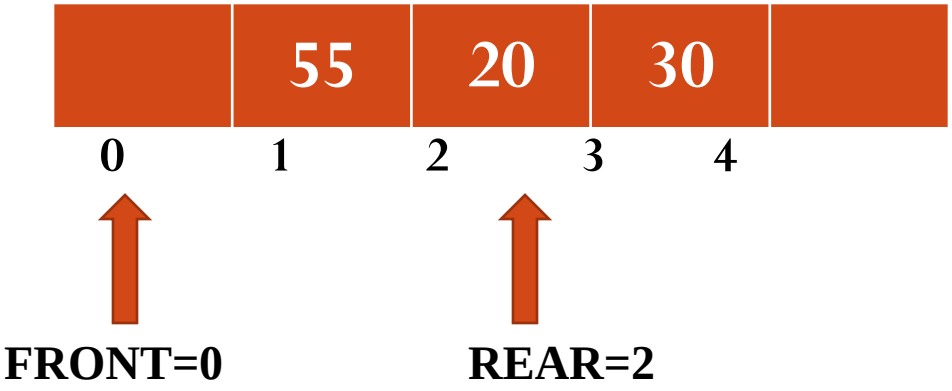
PUSH_DQ(10)

Case 4: No space in FRONT but space in REAR

Shift elements one position to right

A[FRONT]=10

REAR=REAR+1



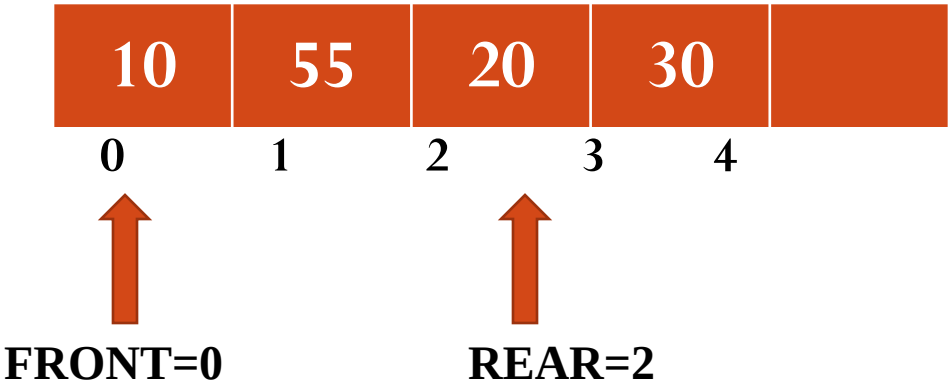
PUSH_DQ(10)

Case 4: No space in FRONT but space in REAR

Shift elements one position to right

A[FRONT]=10

REAR=REAR+1



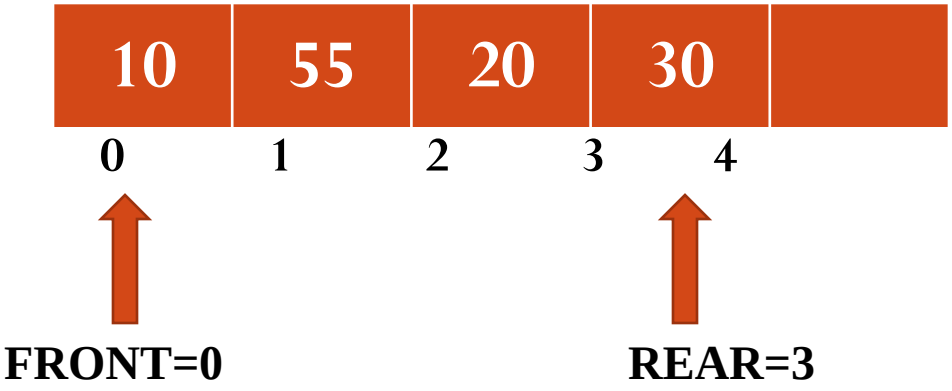
PUSH_DQ(10)

Case 4: No space in FRONT but space in REAR

Shift elements one position to right

A[FRONT]=10

REAR=REAR+1



DEQUE – PUSH Algorithm

Algorithm PUSH_DQ(ITEM)

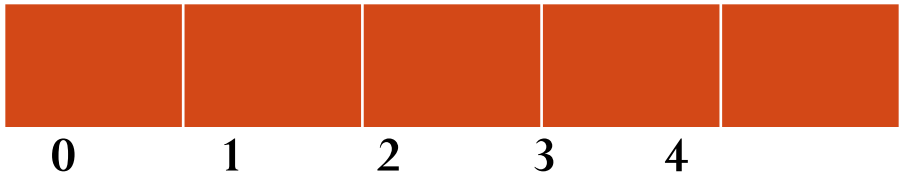
```
{       if  FRONT=0 and REAR=SIZE-1 then
                Print "Deque is FULL"
        else if FRONT=-1 and REAR=-1 then
        {       FRONT=REAR=0
                A[FRONT]=ITEM
        }
        else if FRONT>0 then
        {       FRONT=FRONT-1
                A[FRONT]=ITEM
        }
                                        else
                                        {       for i=REAR to 0 do
                                                        A[i+1]=A[i]
                                                A[FRONT]=ITEM
                                                REAR=REAR+1
                                        }
                                }
```

DEQUE – POP

POP_DQ()

Case 1: FRONT=-1 or REAR=-1

Deque is empty

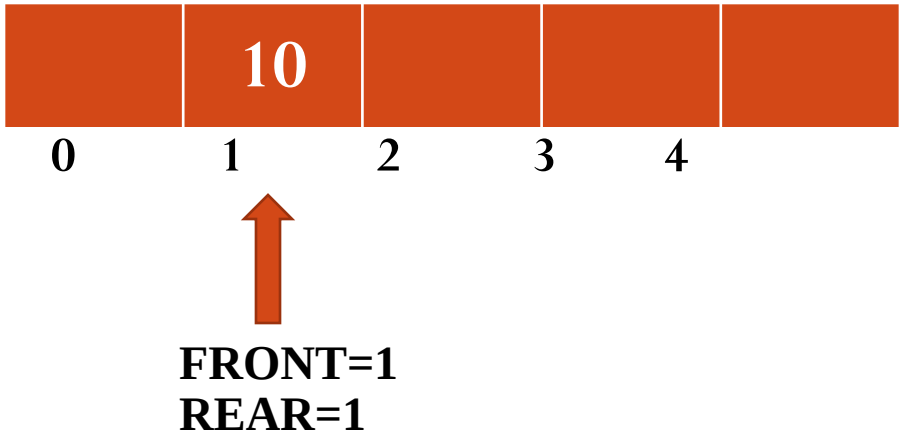


FRONT=-1
REAR=-1

POP_DQ()

Case 2: Deque contains only one element

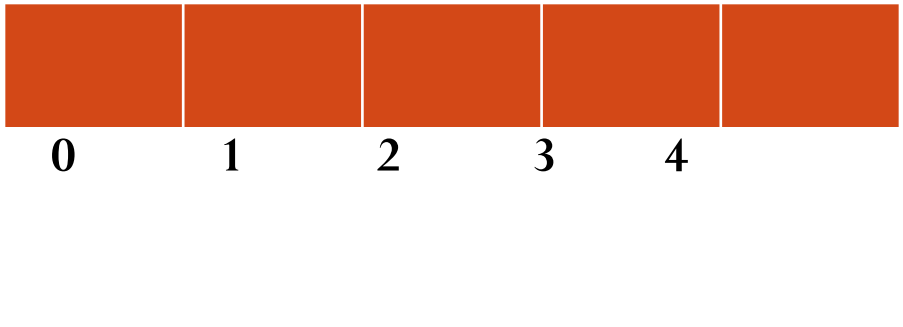
FRONT=REAR=-1



POP_DQ()

Case 2: **Deque contains only one element**

FRONT=REAR=-1

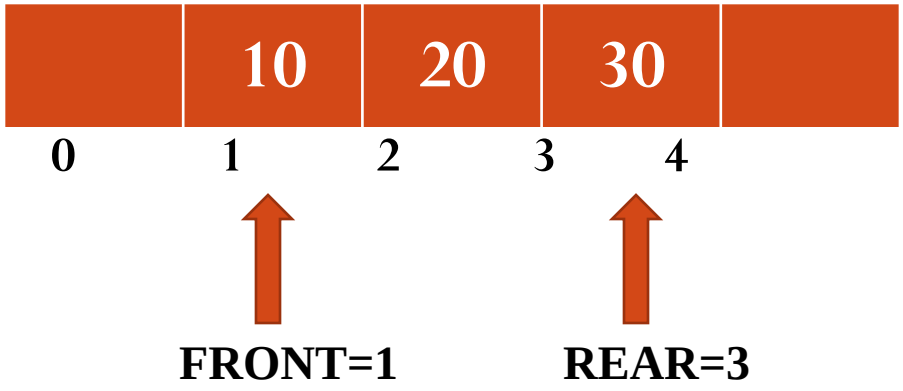


FRONT=-1
REAR=-1

POP_DQ()

Case 3: **Deque contains more than one element**

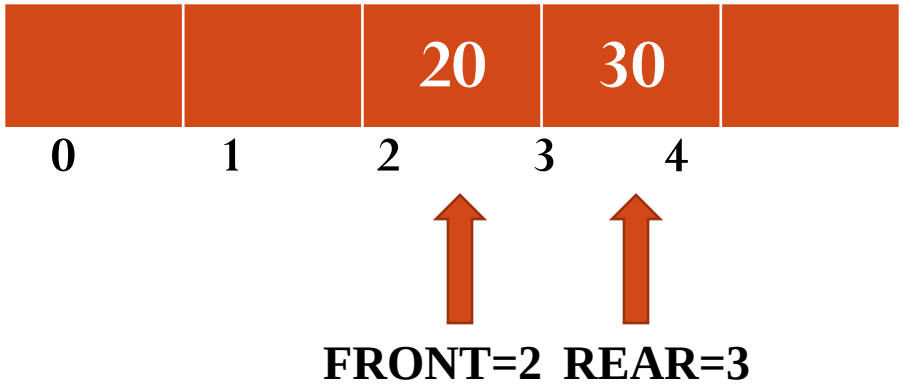
FRONT=FRONT+1



POP_DQ()

Case 3: **Deque contains more than one element**

FRONT=FRONT+1



DEQUE – POP Algorithm

Algorithm POP_DQ()

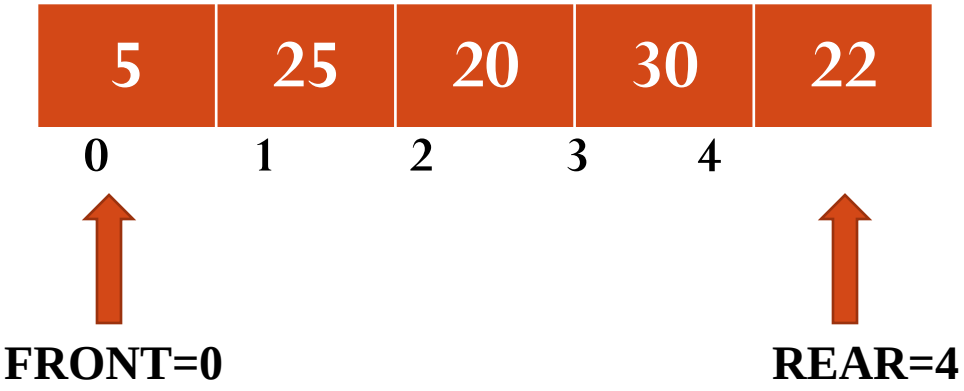
```
{       if FRONT=-1 and REAR=-1 then
                Print “Deque is EMPTY”
        else if FRONT=REAR then
        {       Print “Dequed item is “ A[FRONT]
                FRONT=REAR=-1
        }
        else
        {       Print “Dequed item is “ A[FRONT]
                FRONT=FRONT+1
        }
}
```

DEQUE – INJECT

INJECT_DQ(10)

Case 1: FRONT=0 and REAR=SIZE-1

Deque is FULL

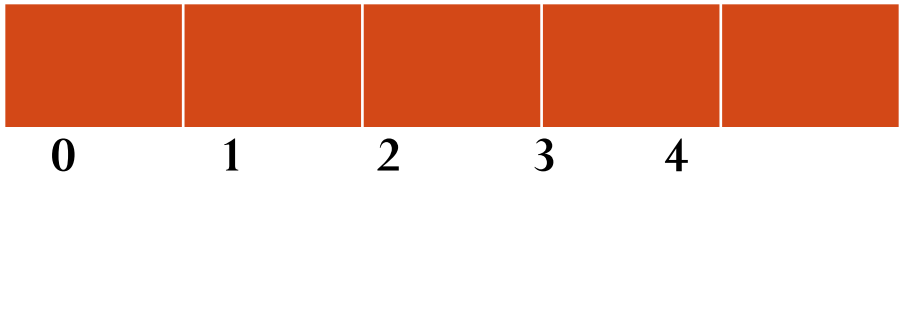


INJECT_DQ(10)

Case 2: FRONT=-1 and REAR=-1

FRONT=REAR=0

A[REAR]=10



FRONT=-1

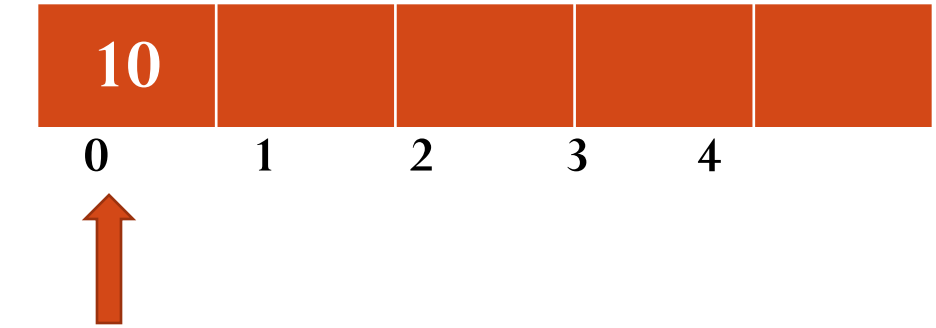
REAR=-1

INJECT_DQ(10)

Case 2: FRONT=-1 and REAR=-1

FRONT=REAR=0

A[REAR]=10



FRONT=0

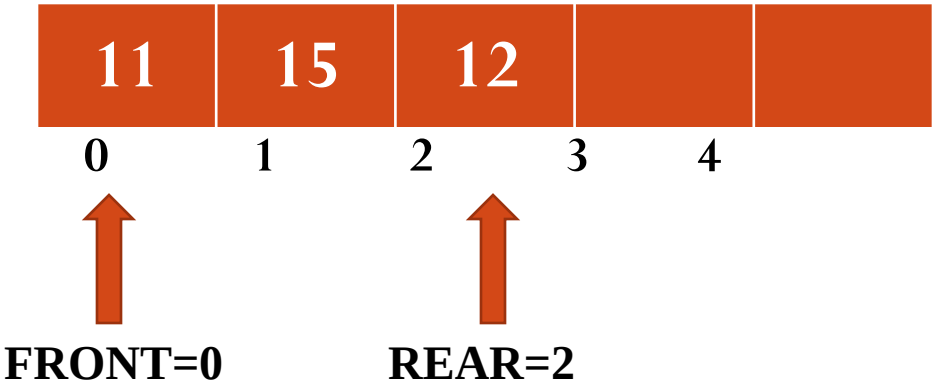
REAR=0

INJECT_DQ(10)

Case 3: Free space at REAR end

REAR=REAR+1

A[REAR]=10

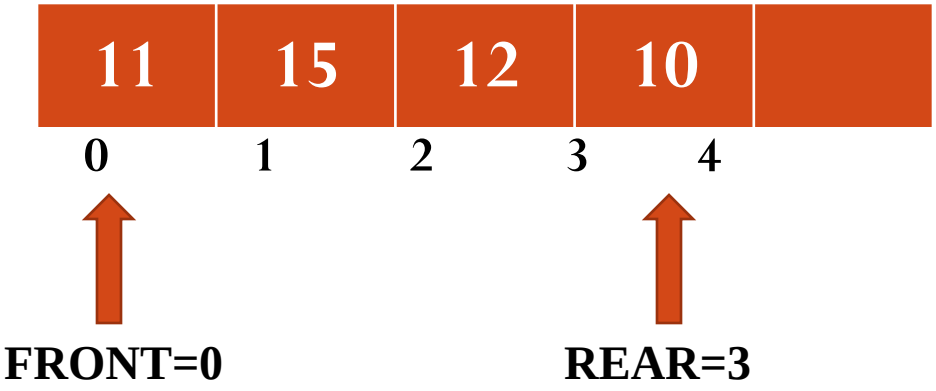


INJECT_DQ(10)

Case 3: Free space at REAR end

REAR=REAR+1

A[REAR]=10



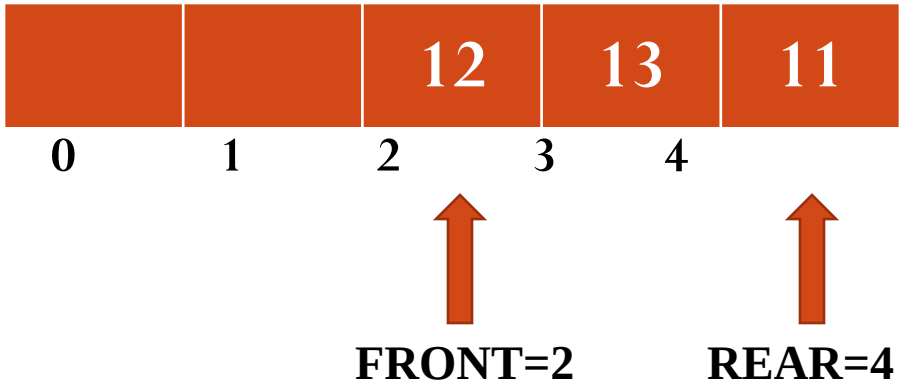
INJECT_DQ(10)

Case 4: **No space at REAR end but space at FRONT end**

Shift the elements one position to the left

A[REAR]=10

FRONT=FRONT-1



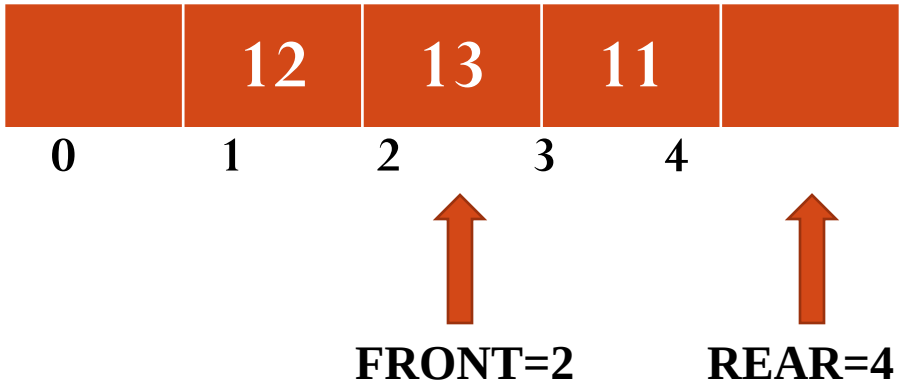
INJECT_DQ(10)

Case 4: **No space at REAR end but space at FRONT end**

Shift the elements one position to the left

A[REAR]=10

FRONT=FRONT-1



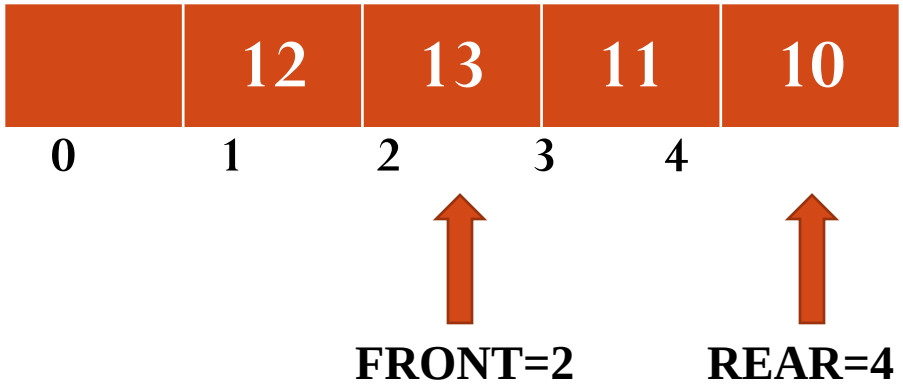
INJECT_DQ(10)

Case 4: **No space at REAR end but space at FRONT end**

Shift the elements one position to the left

A[REAR]=10

FRONT=FRONT-1



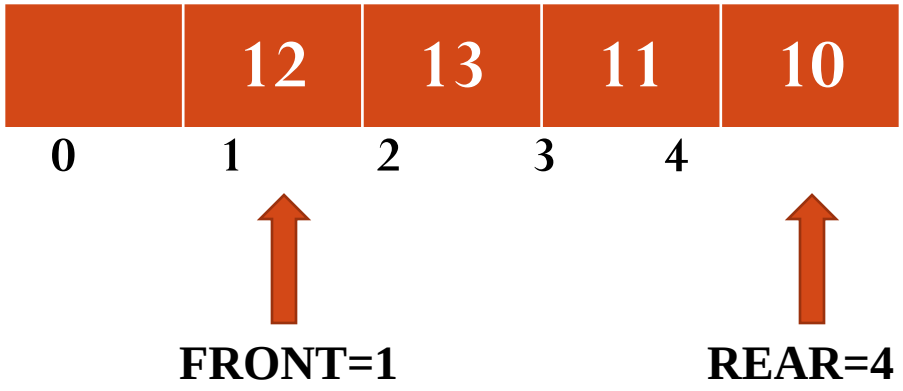
INJECT_DQ(10)

Case 4: **No space at REAR end but space at FRONT end**

Shift the elements one position to the left

A[REAR]=10

FRONT=FRONT-1



DEQUE – INJECT Algorithm

Algorithm INJECT_DQ(ITEM)

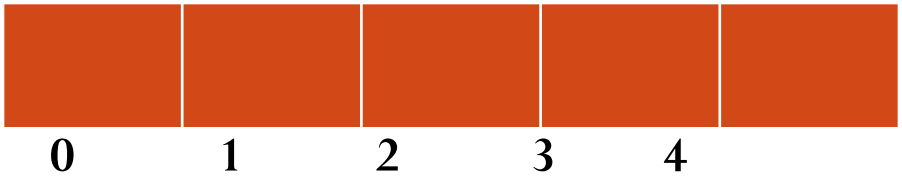
```
{       if  FRONT=0 and REAR=SIZE-1 then
                Print "Deque is FULL"
        else if FRONT=-1 and REAR=-1 then
        {       FRONT=REAR=0
                A[REAR]=ITEM
        }
        else if REAR<SIZE-1 then
        {       REAR=REAR+1
                A[REAR]=ITEM
        }
                                        else
                                        {       for i=FRONT to REAR do
                                                        A[i-1]=A[i]
                                                A[REAR]=ITEM
                                                FRONT=FRONT-1
                                        }
                                }
```

DEQUE – EJECT

EJECT_DQ()

Case 1: FRONT=-1 and REAR=-1

Deque is EMPTY

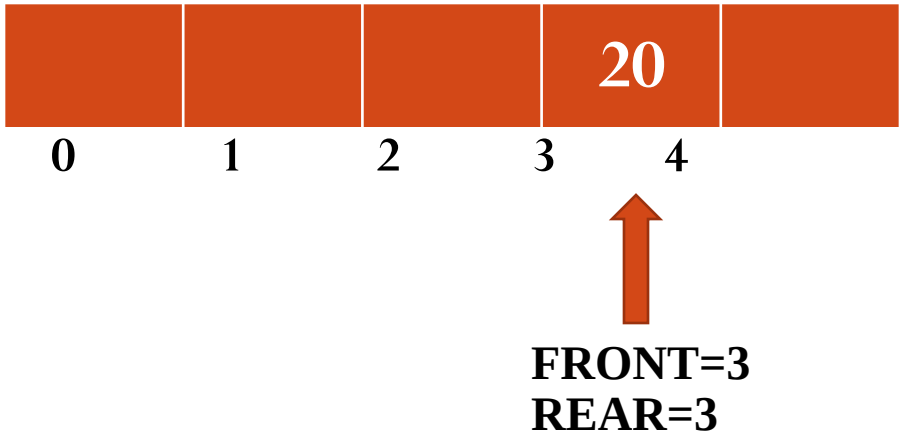


FRONT=-1
REAR=-1

EJECT_DQ()

Case 2: FRONT= REAR

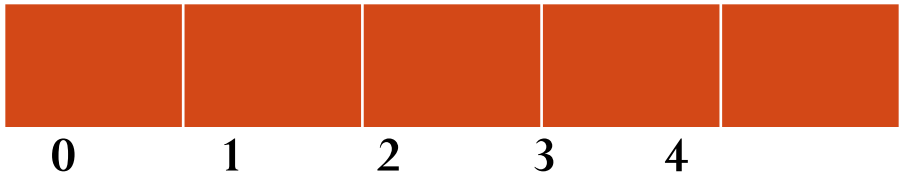
FRONT=REAR=-1



EJECT_DQ()

Case 2: FRONT= REAR

FRONT=REAR=-1

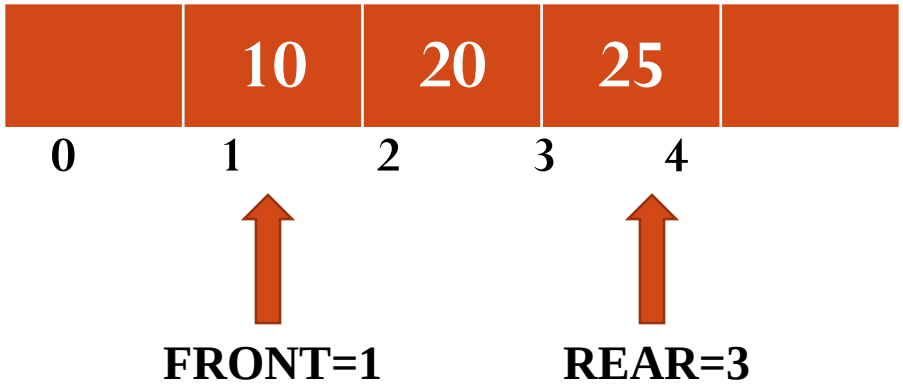


FRONT=-1
REAR=-1

EJECT_DQ()

Case 3: Deque contains more than one element

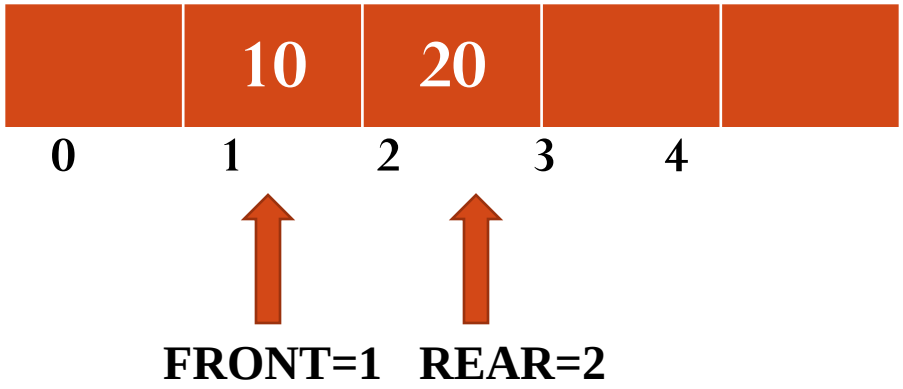
REAR=REAR-1



EJECT_DQ()

Case 3: Deque contains more than one element

REAR=REAR-1



DEQUE – EJECT Algorithm

Algorithm EJECT_DQ()

```
{       if FRONT=-1 and REAR=-1 then
                Print “Deque is EMPTY”
        else if FRONT=REAR then
        {       Print “Dequed item is “ A[REAR]
                FRONT=REAR=-1
        }
        else
        {       Print “Dequed item is “ A[REAR]
                REAR=REAR-1
        }
}
```

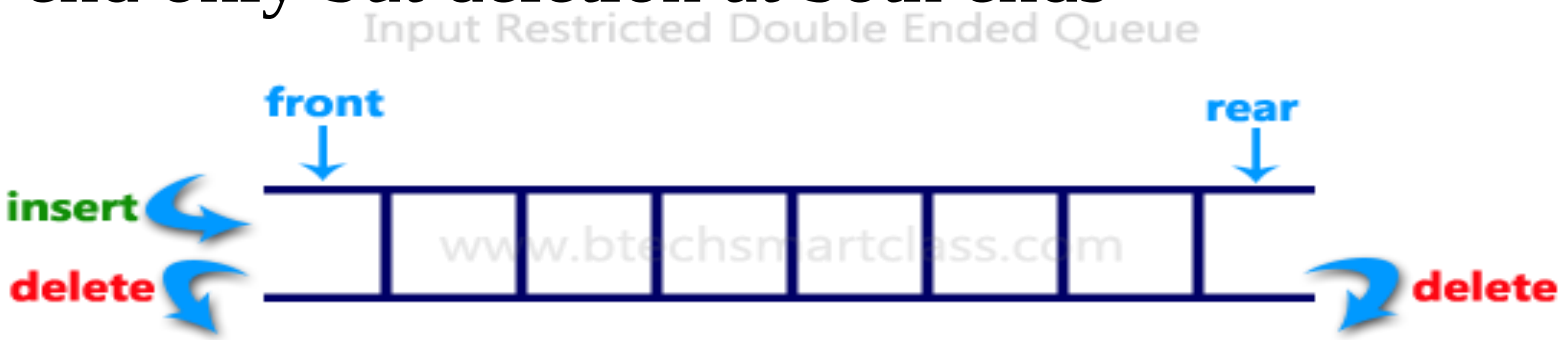
DEQUE – DISPLAY Algorithm

Algorithm DISPLAY_DQ()

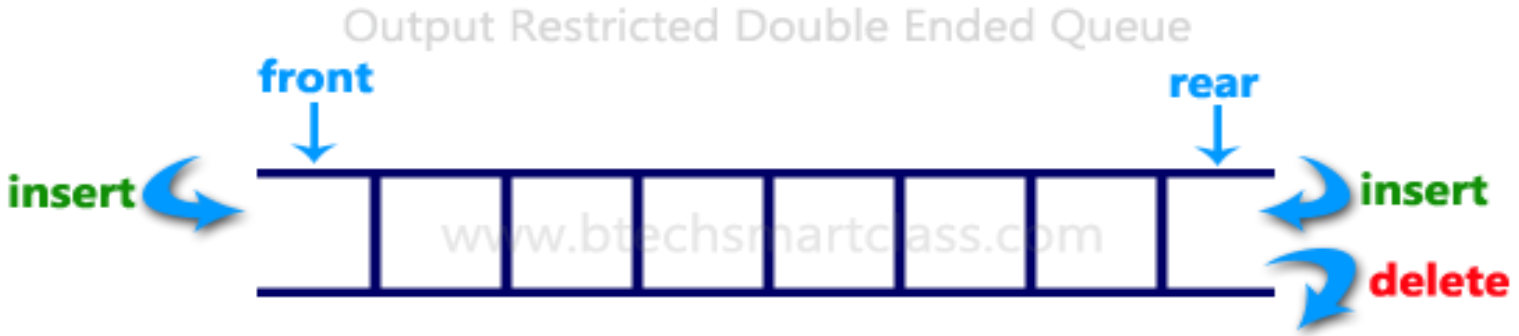
```
{
        if FRONT=-1 and REAR=-1 then
                Print “Deque is EMPTY”
        else
        {
                for i=FRONT to REAR do
                        Print A[i]
        }
}
```

DOUBLE ENDED QUEUE(DEQUE)

- There are two variations of Deque
  - **Input restricted Deque**: Allows insertion at one end only but deletion at both ends



  - **Output restricted Deque**: Allows deletion at one end only but insertion at both ends



APPLICATIONS OF DEQUE

- Palindrome checker
- A-steal job scheduling algorithm

DEQUE USING ARRAY-PROGRAM

```
#include<stdio.h>
int A[20],size,front,rear;
void PUSH_DQ(int item)
{       int i;
        if(front==0 && rear==size-1)
                printf("Deque is FULL. Insertion is not possible.");
        else if(rear==-1)
        {
                front=0;
                rear=0;
                A[front]=item;
        }
```

```
        else if(front>0)
        {
                front--;
                A[front]=item;
        }
        else
        {       for(i=rear;i>=front;i--)
                        A[i+1]=A[i];
                A[front]=item;
                rear++;
        }
}
```

```
void INJECT_DQ(int item)
{  int i;
   if(front==0 && rear==size-1)
         printf("Deque is FULL. Insertion is not possible.");
   else if(rear==-1)
   {     front=0;
         rear=0;
         A[rear]=item;
   }
   else if(rear<size-1)
   {     rear++;
         A[rear]=item;
   }
         else
         {
               for(i=front;i<=rear;i++)
                     A[i-1]=A[i];
               A[rear]=item;
               front--;
         }
   }
```

```
void POP_DQ()
{       if(front==-1)
                printf("Deque is EMPTY");
        else if(front==rear)
        {       printf("Deleted item is %d",A[front]);
                front=-1;
                rear=-1;
        }
        else
        {       printf("Deleted item is %d",A[front]);
                front++;
        }
}
```

```
void EJECT_DQ()
{       if(front==-1)
                printf("Deque is EMPTY");
        else if(front==rear)
        {       printf("Deleted item is %d",A[front]);
                front=-1;
                rear=-1;
        }
        else
        {       printf("Deleted item is %d",A[rear]);
                rear--;
        }
}
```

```
void DISPLAY_DQ()
{       int i;
        if(front==-1)
                printf("Queue is EMPTY");
        else
        {
                for(i=front;i<=rear;i++)
                        printf("%d\t",A[i]);
        }
}
```

```
void main()
{
   int item,opt;
   front=-1;
   rear=-1;
   printf("Enter the size of the Deque: ");
   scanf("%d",&size);
   do
   {
         printf("\nEnter the option: \n”1.PUSH\n2.POP\n
         3.INJECT\n4.EJECT\n5.DISPLAY\n6.EXIT\n");
         scanf("%d",&opt);
```

```
switch(opt)
{       case 1:printf("Enter the item to be inserted: ");
               scanf("%d",&item);
               PUSH_DQ(item);
               break;
        case 2:POP_DQ();
               break;
        case 3:printf("Enter the item to be inserted: ");
               scanf("%d",&item);
               INJECT_DQ(item);
               break;
        case 4:EJECT_DQ();
               break;
```

```
                case 5:         DISPLAY_DQ();
                                break;
                case 6:         break;
                default: printf("Invalid option...");
        }
   }while(opt!=6);
}
```