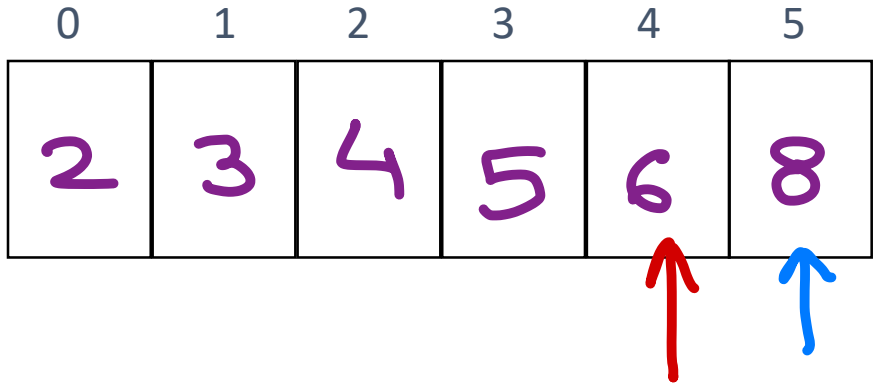
Data Structure & Algorithms

Nilesh Ghule

Selection Sort

5 6 3 8 2 4

```
for(i=0; i<n-1; i++){
    for(j=i+1; j<n; j++){
        if(a[i] > a[j]){
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}
```



Total itrs = $(n-1)+(n-2)+\dots+1$

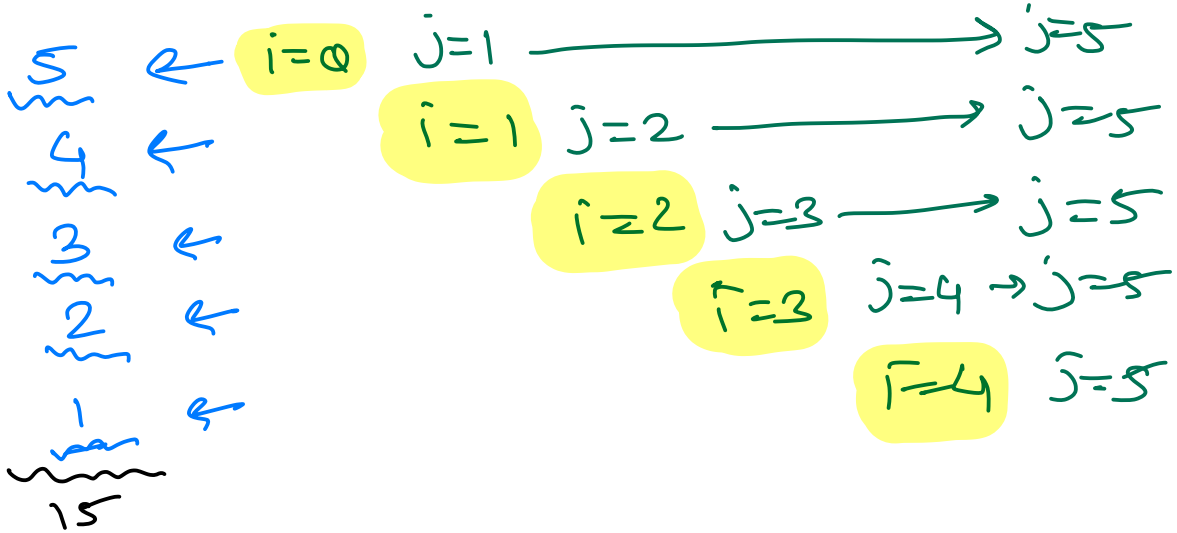
itrs = $\frac{n(n-1)}{2}$

$T \propto \frac{n(n-1)}{2}$

$T \propto n^2 - n$

if $n >> 1$, $n^2 >>> n$

$T \propto n^2 \rightarrow O(n^2)$



Bubble Sort

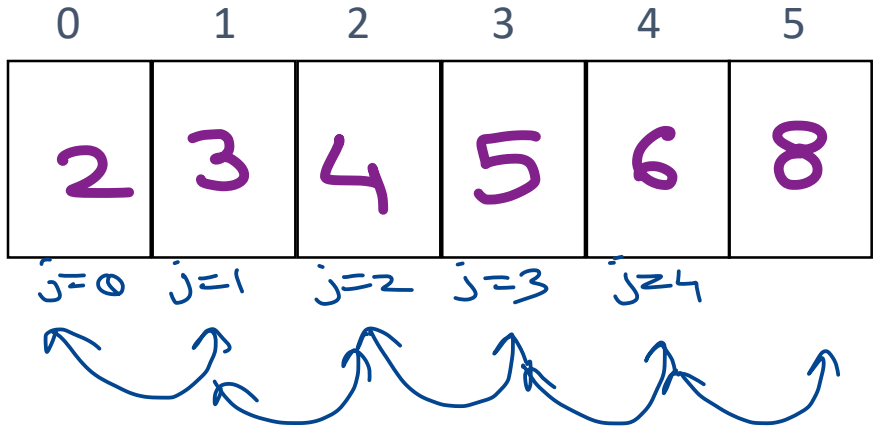
5 6 3 8 2 4

```
for(i=0; i<n-1; i++) {
    for(j=0; j<n-1; j++) {
        if(a[j] > a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
```

3

3

3



Pass 1 → 5

Pass 2 → 5

Pass 3 → 5

Pass 4 → 5

Pass 5 → 5

itr = (n-1)*(n-1)

$T \propto (n-1)^2$

$T \propto n^2 - 2n + 1$

$n >> 1, n^2 >>>> n$

$T \propto n^2$

$O(n^2)$

Improved Bubble Sort

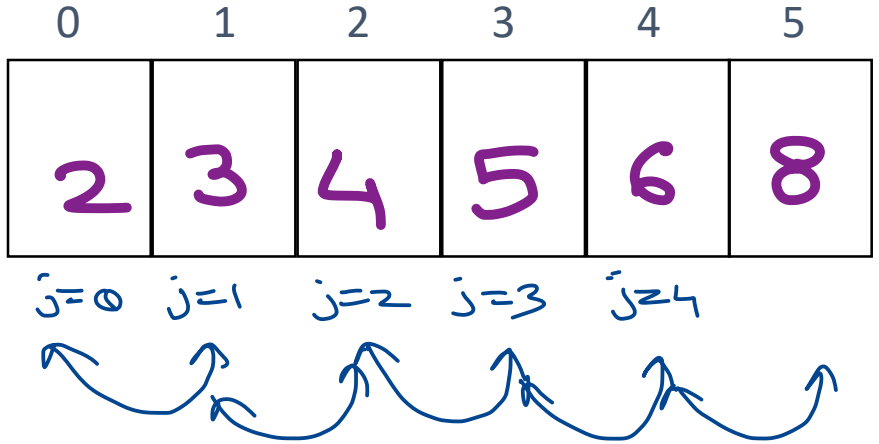
5 6 3 8 2 4

```
for(i=0; i<n-1; i++) {
    for(j=0; j<n-1-i; j++) {
        if(a[j] > a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
```

3

3

3



Pass 1 → 5

Pass 2 → 4

Pass 3 → 3

Pass 4 → 2

Pass 5 → 1

itr = (n-1) + (n-2) + ... + 1

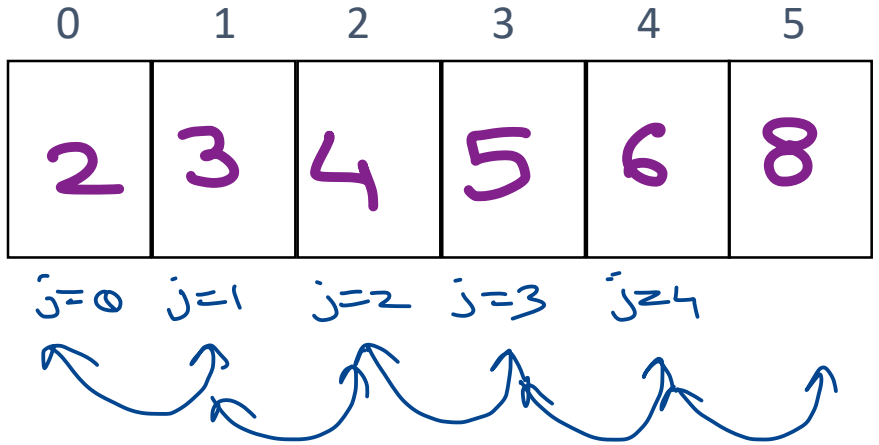
$T \propto \frac{n(n-1)}{2}$

$T \propto n^2$

$O(n^2)$

Improved Bubble Sort

```
for(i=0; i<n-1; i++) {
    flag = 0;
    for(j=0; j<n-1-i; j++) {
        if(a[j] > a[j+1]) {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
            flag = 1;
        }
    }
    if(flag == 0)
        break;
}
```



if array is already sorted, in first pass no swapping is done.

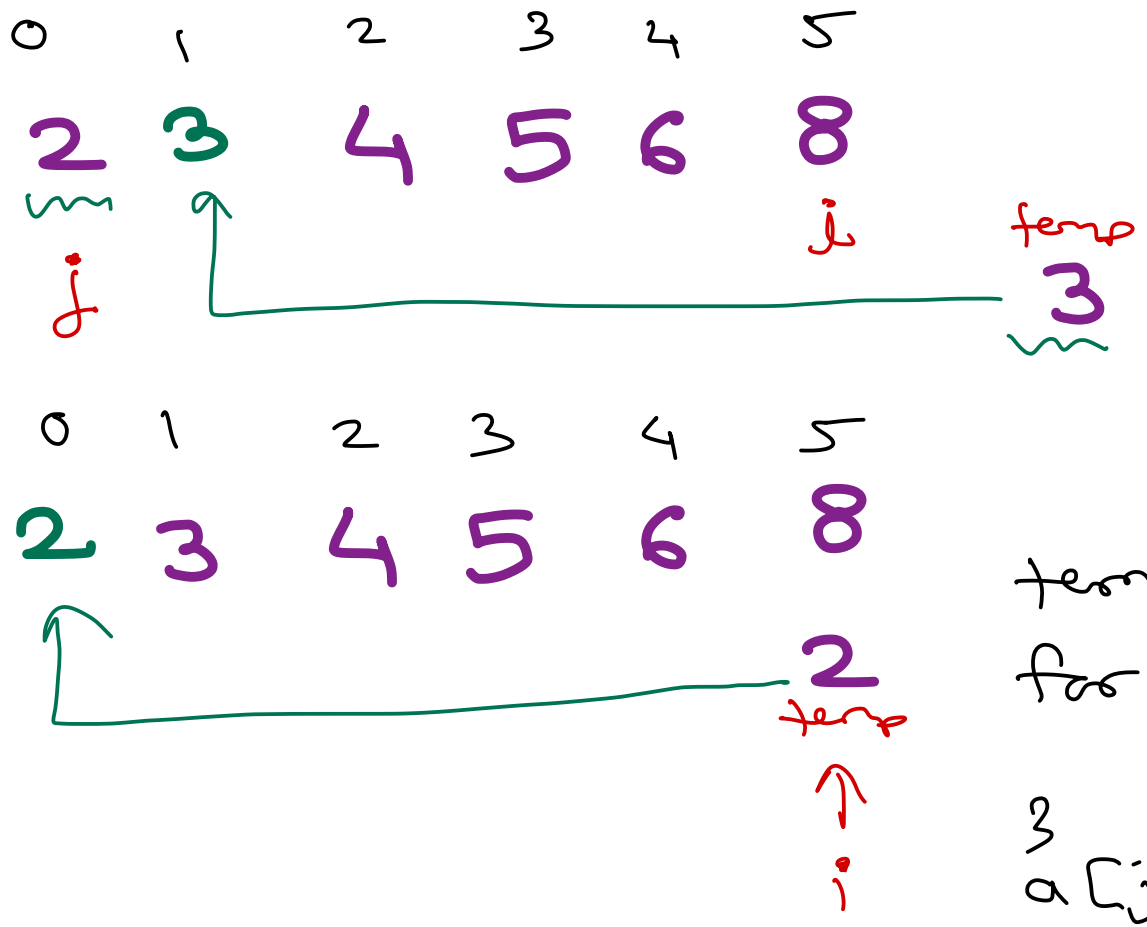
Best Case

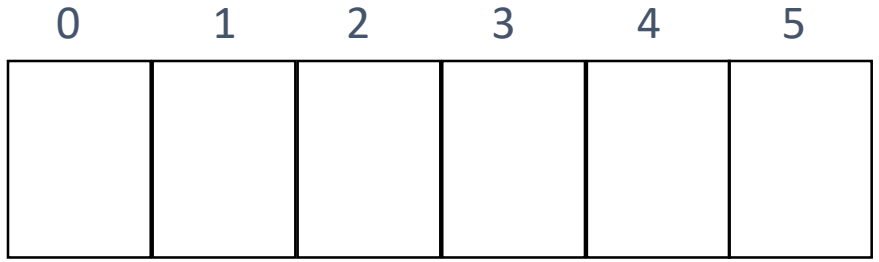
pass 1 → 5

itr = n

O(n)

Insertion Sort






```
temp = a[i];
for(j=i-1; j>=0 && a[j]>temp ; j--){
    a[j+1] = a[j];
}
a[j+1] = temp;
```

Insertion Sort

6 5 3 8 2 4

```
for(i= 1; i<n; i++){
    temp = a[i];
    for(j=i-1; j>=0 && a[j]>temp ; j--){
        a[j+1] = a[j];
    }
    a[j+1]= temp;
}
```

0	1	2	3	4	5
6	5	3	8	2	4

$itr = (n-1) + (n-2) + \dots 1$

$itr = \frac{n(n-1)}{2}$

$T \propto n^2$

$O(n^2)$

pass 1 → 1

5 6 ← i=1

Pass 2 → 2

3 5 6 ← i=2

pass 3 = 3

3 5 6 8 ← i=3

pass 4 → 4

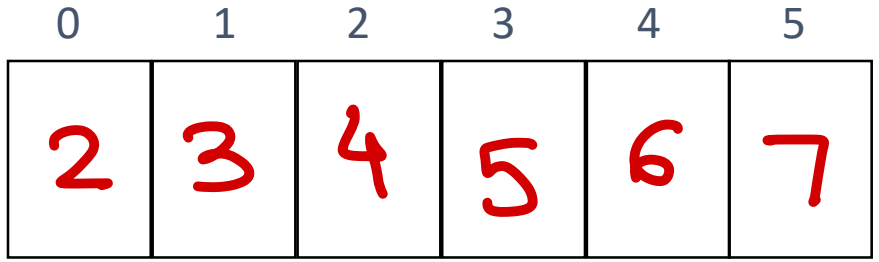
2 3 5 6 8 ← i=4

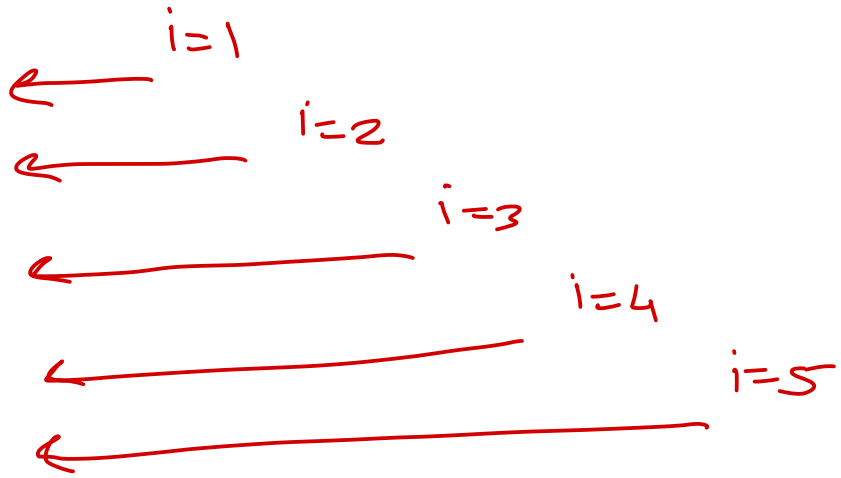
pass 5 → 5

2 3 4 5 6 8 ← i=5

Insertion Sort

```
for(i=1; i<n; i++){
    temp = a[i];
    for(j=i-1; j>=0 && a[j]>temp; j--){
        a[j+1] = a[j];
    }
    a[j+1] = temp;
}
```





best case → array already sorted

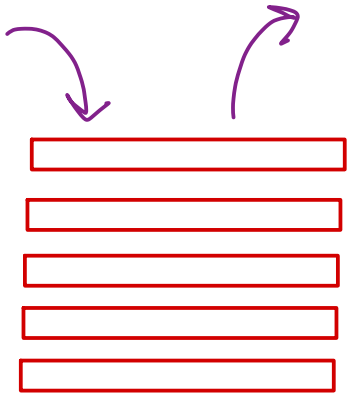
itr = n

$T \propto n$

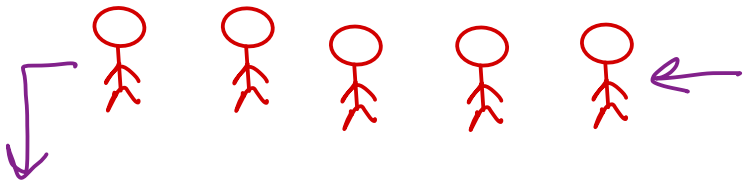
$O(n)$

Stack and Queue

- Stack & Queue are utility data structures.
- Can be implemented using array or linked lists.
- Usually time complexity of stack & queue operations is O(1).
- Stack is Last-In-First-Out structure.
- Stack operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()*



- Simple queue is First-In-First-Out structure.
- Queue operations
  - push()
  - pop()
  - peek()
  - isEmpty()
  - isFull()*



- Queue types
  - Linear queue
  - Circular queue
  - Deque
  - Priority queue

Stack / Queue using Linked List

- Stack can be implemented using linked list.
  - add first
  - delete first
  - is empty

- Queue can be implemented using linked list.
  - add last
  - delete first
  - is empty

Linear Queue

- front → pop
- rear ← push

arr

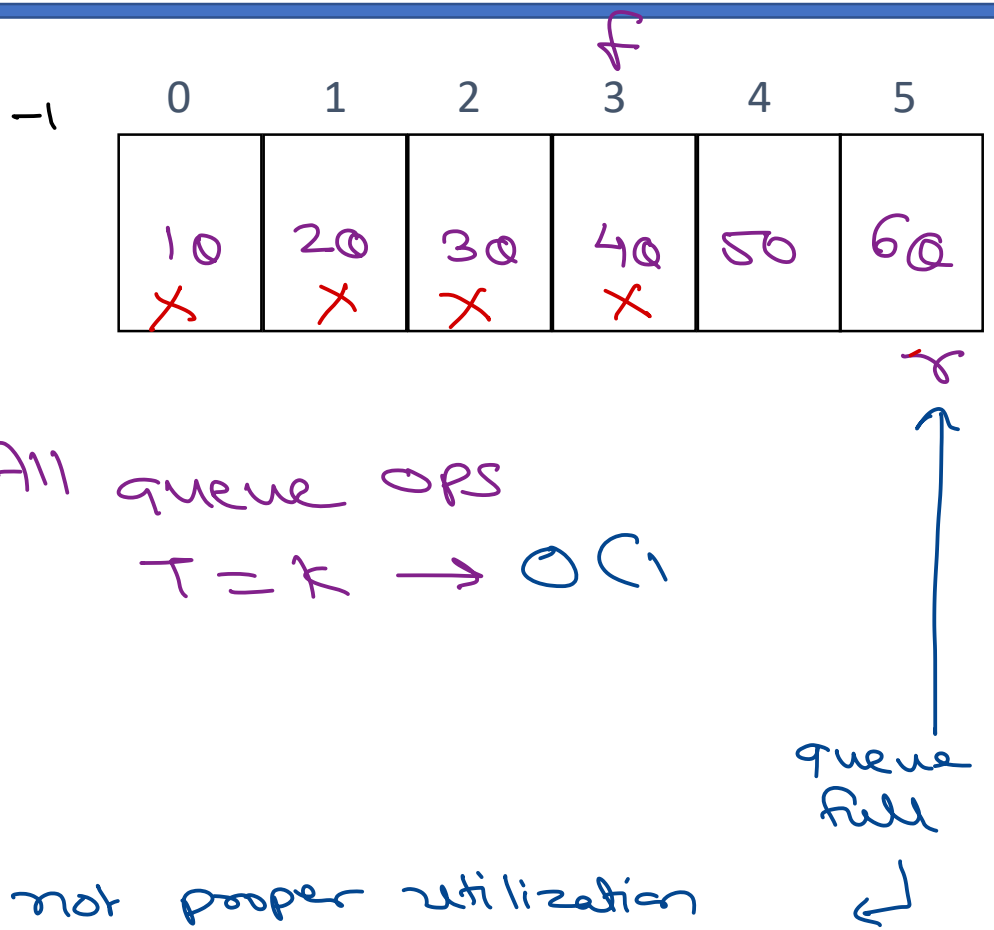
```
init:
    f = -1;
    r = -1;

push:
    r++;
    arr[r] = val;

pop:
    f++;

peek:
    return arr[f+1];
```

full:

$r == MAX - 1$

empty:

$f == r$



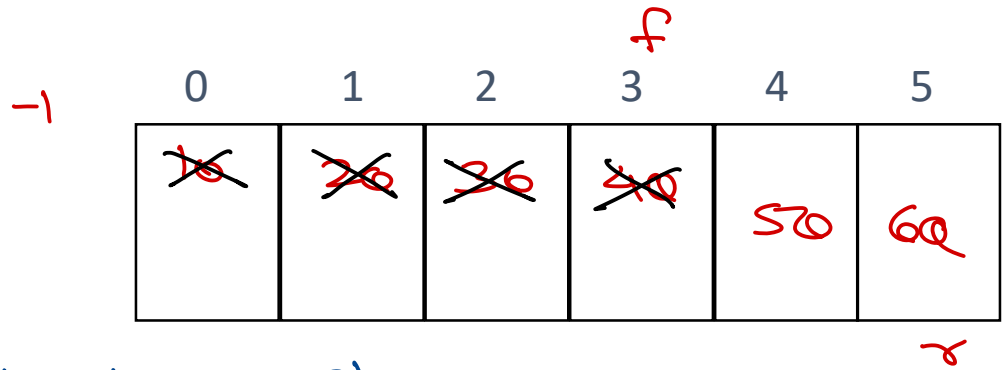
All queue ops

$T = k \rightarrow O(1)$

not proper utilization of memory.

Circular Queue

- In linear queue (using array) when *rear* reaches last index, further elements cannot be added, even If space is available due to deletion of elements from *front*. Thus space utilization is poor.
- Circular queue allows adding elements at the start of array if *rear* reaches last index and space is free at the start of the array.
- Thus *rear* and *front* can be incremented in circular fashion i.e. 0, 1, 2, 3, …, n-1. So they are said to be circular queue.
- However queue full and empty conditions become tricky.

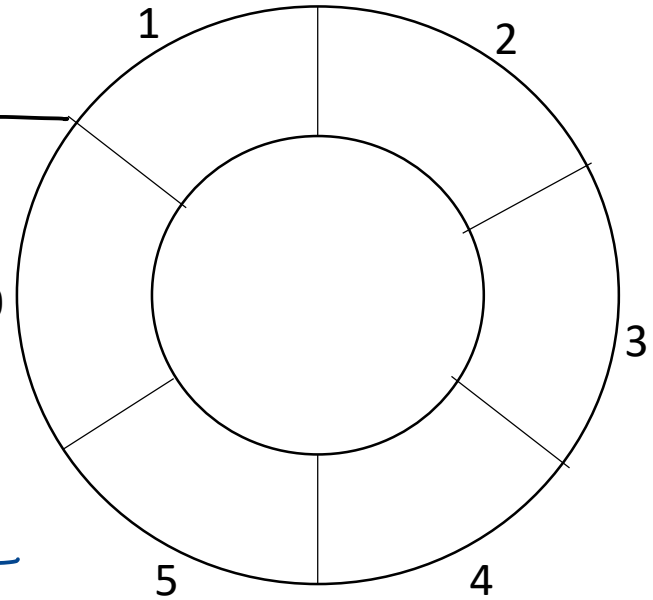


circular increment

```
if(r == max-1)
    r = 0;
else
    r++;
```

```
r = (r+1) % max;
```

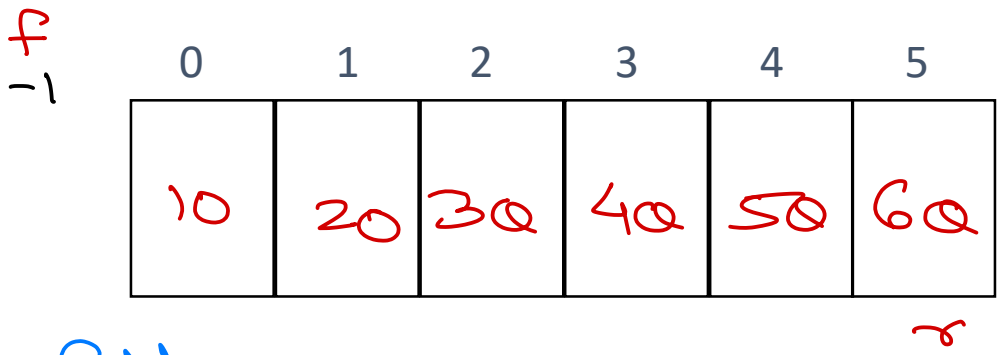
r	max = 6
-1 + 1 →	% 6 = 0
0 + 1 →	% 6 = 1
1 + 1 →	% 6 = 2
2 + 1 →	% 6 = 3
3 + 1 →	% 6 = 4
4 + 1 →	% 6 = 5
5 + 1 →	% 6 = 0



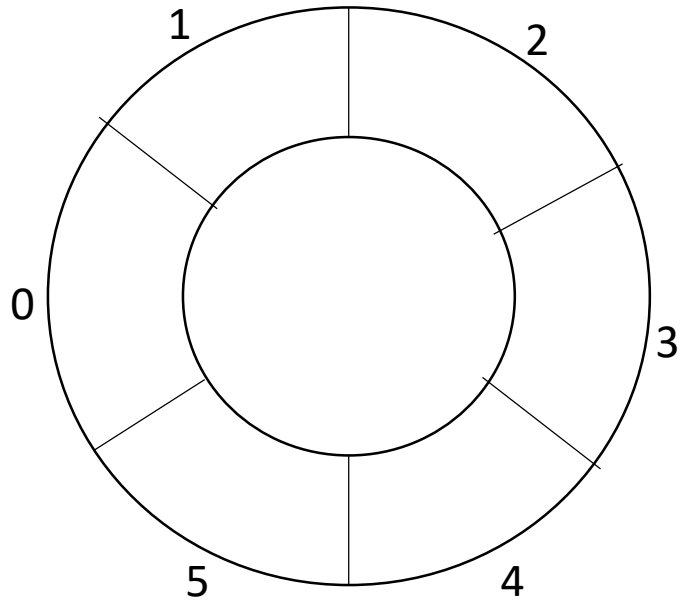
Circular Queue

init :
f = -1
r = -1

push :
r = (r+1) % MAX
arr[r] = val



queue full.
f == -1 && r == MAX-1

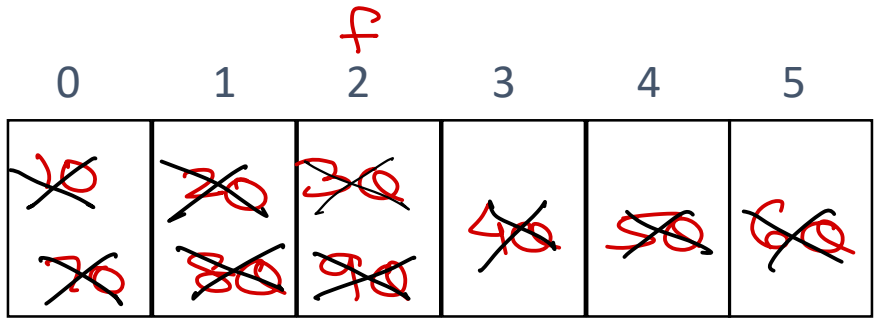


Circular Queue

```
init :
    f = -1
    r = -1

push :
    r = (r+1) % MAX
    arr[r] = val

pop :
    f = (f+1) % MAX
    if (f == r) {
        f = -1;
        r = -1;
    }

peek :
    next = (f+1) % MAX;
    return arr[next];
```

queue empty :

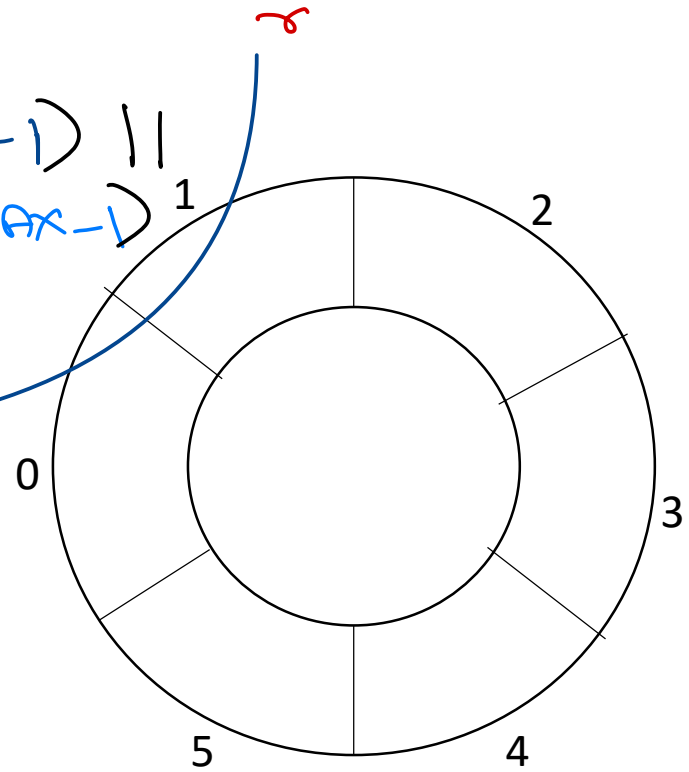
```
f == r && f == -1
```



queue full.

```
(f == r && f1 = -1) ||
(f == -1 && r == MAX-1)
```

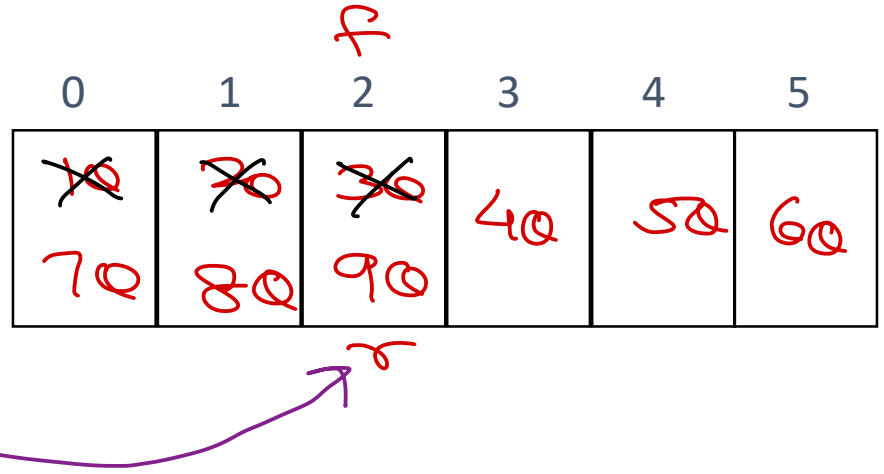
↳ see last slide.

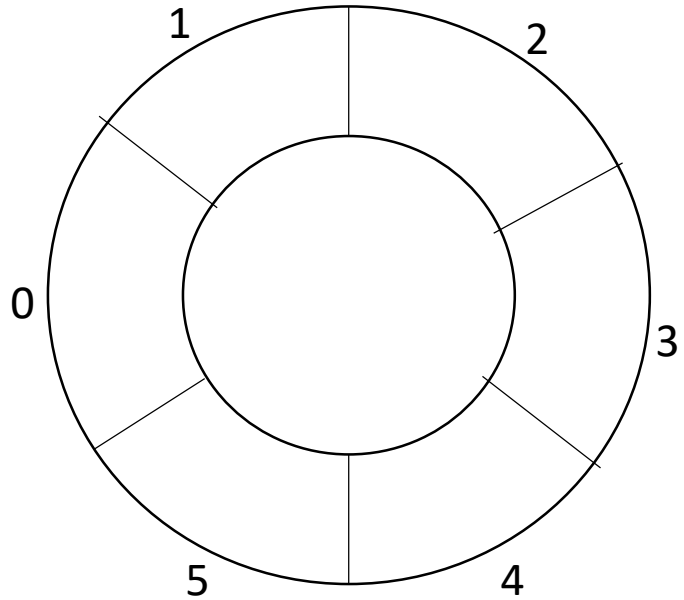


Circular Queue

Queue Full

(f == r && f != -1)

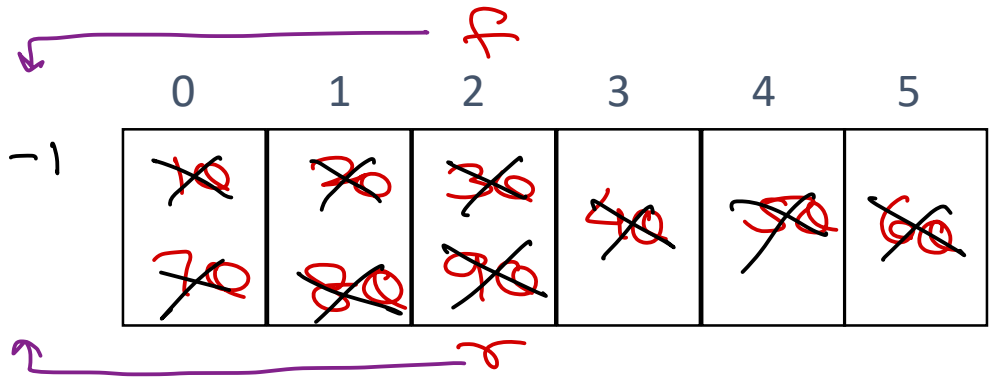




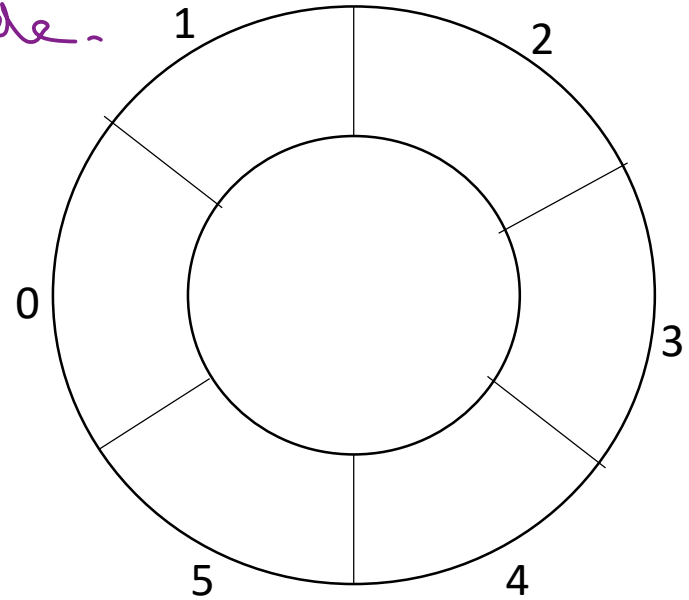
Circular Queue

queue empty

(f == r && f == -1)



reset after deleting last ele.

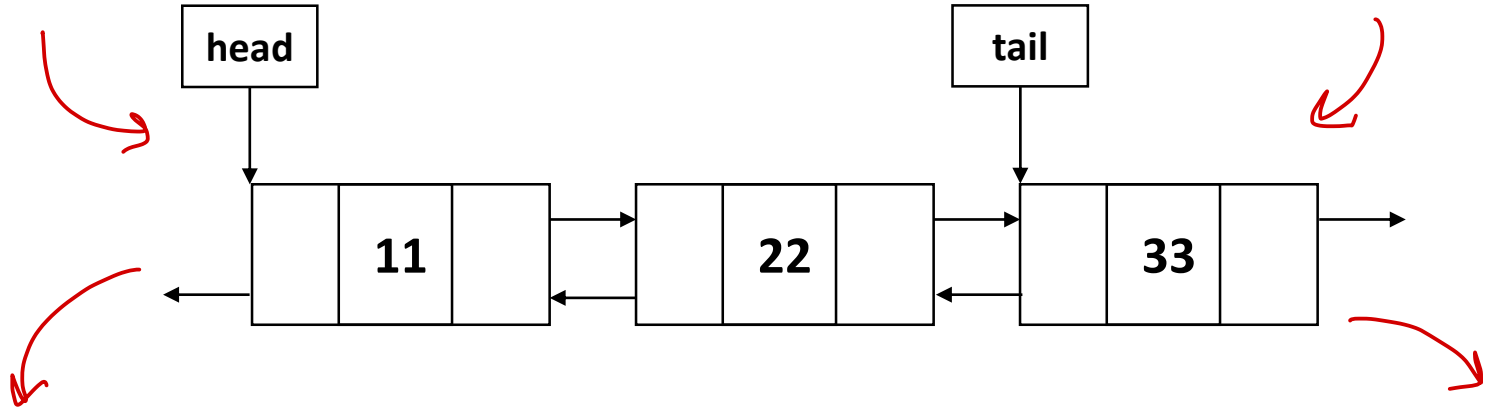


DeQueue

- In double ended queue, values can be added or deleted from front end or rear end.

push_back()
push_front()

pop_back()
pop_front()



Priority queue

- In priority queue, element with highest priority is removed first.

doesn't follow FIFO.

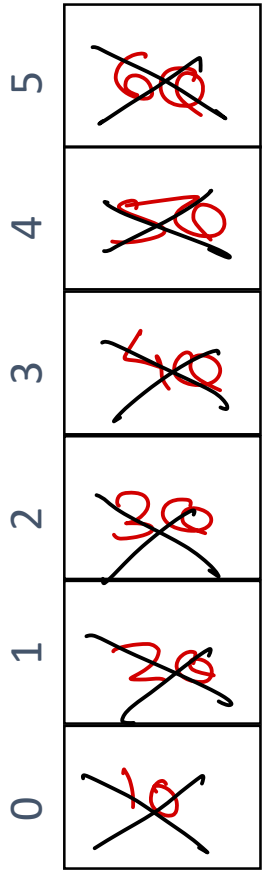
top ← push
top → pop

arr

arr

```
init :
    top = -1 ;

push :
    top++ ;
    arr[top] = val ;

peek :
    return arr[top] ;

pop :
    top-- ;
```

```
full :
    top == MAX-1

empty :
    top == -1
```



top → -1

Stack / Queue in Java collections

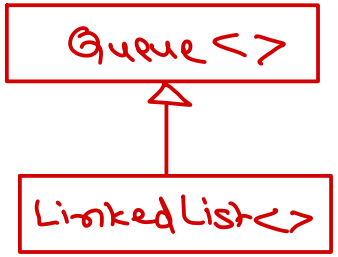
- class java.util.Stack<E>
  - E push(E);
  - E pop();
  - E peek();
  - boolean isEmpty();

isFull() X

→ java.util package.

```
Stack<Integer> s = new Stack<>();
    s.push(m);
    val = s.pop();
    if(s.isEmpty())
```

- interface java.util.Queue<E>
  - boolean offer(E e); ← push
  - E poll(); ← pop
  - E peek();
  - boolean isEmpty();



a + b → infix expr → human

+ a b → prefix expr } computing

a b + → postfix expr }

$5 + 9 - 4 * (8 - 6 / 2) + 1 \$ (7 - 3)$

priorities

power ( )

$

* /

\+ -

(6) (7) (5) (2) (1) (8) (4) (3)

$5 + 9 - 4 * (8 - 6 / 2) + 1 \$ (7 - 3)$

$5 + 9 - 4 * (8 - 62/) + 1 \$ (7 - 3)$

$5 + 9 - 4 * 862/- + 1 \$ (7 - 3)$

$5 + 9 - 4 * 862/- + 1 \$ 73-$

$5 + 9 - 4 * 862/- + 173-\$$

$5 + 9 - 4862/-* + 173-\$$

$59+ - 4862/-* + 173-\$$

$59+4862/-*- + 173-\$$

$59+4862/-*-173-\$+$

(6) (7) (5) (2) (1) (8) (4) (3)

5 + 9 - 4 * (8 - 6 | 2) + 1 $ (7 - 3)

+ - + 5 9 * 4 - 8 / 6 2 $ 1 - 7 3

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>