Data Structure & Algorithms

Nilesh Ghule

Insertion Sort

6 5 3 8 2 4

$itr = 1 + 2 + 3 + \dots + (n-1)$

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

$T \propto n^2 - n$

$T \propto n^2$

$O(n^2)$

pass 1 (1)
pass 2 (2)
pass 3 (3)
pass 4 (4)
pass 5 (5)

total : 15

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

	i=1	i=2	i=3	i=4	i=5
5	6				
3	5	6			
3	5	6	8		
2	3	5	6	8	
2	3	4	5	6	8

```
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;
}
```

① Calculate time complexity - general case

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;
}
```

itr = n-1

$T \propto n-1$

$T \propto n$

$O(n)$ ← best case

② Calculate time complexity – best case

1, 2, 3, 4, 5, 6

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

1 2

1 2 3

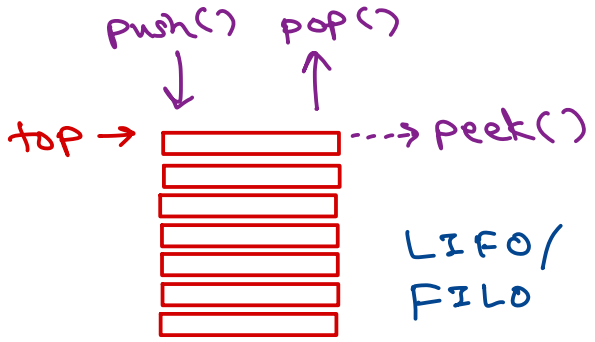
1 2 3 4

1 2 3 4 5

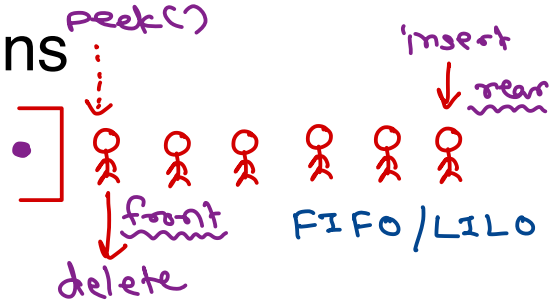
1 2 3 4 5 6

Stack and Queue

- Stack & Queue are utility data structures. – data processing (not storage)
- Can be implemented using array or linked lists.
- Usually time complexity of stack & queue operations is O(1). T=k
- 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 – Double ended queue
  - Priority queue → ele with highest priority comes out first. (internally eles are maintained in order) time complexity O(1)X
    - efficiently impl using heap DS.

Linear Queue – using array

init:

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

push:

```
r++;
arr[r] = val;
```

pop:

```
f++;
val = arr[f];
```

peek:

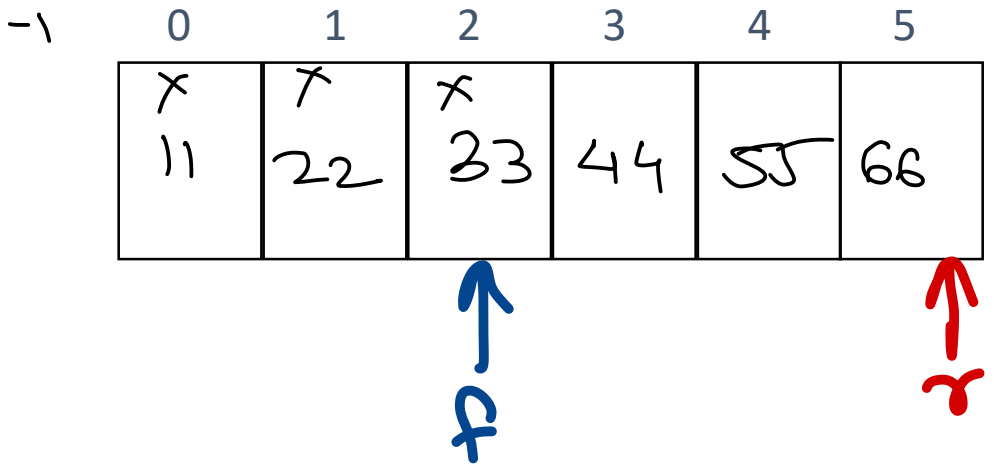
```
val = arr[f+1];
```

full:

```
r == MAX-1
```

empty:

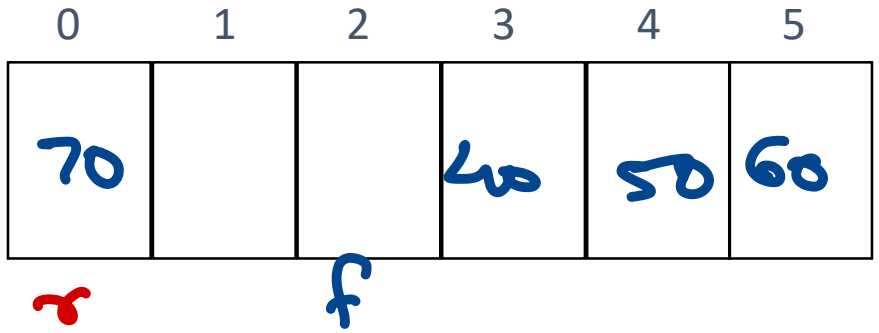
```
f == r
```



In linear queue, when rear reach last ele (max-1), then further eles cannot be added i.e. queue full condition.
In this case there may be few spaces empty at start of array.
Thus lin que not doing mem. utilization properly → limitation.

Circular Queue

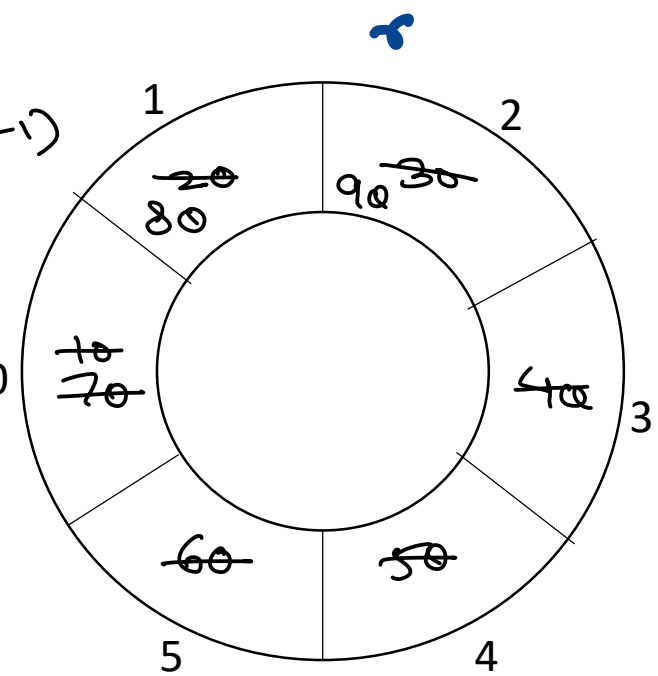
- In linear queue (using array) when *rear* reaches last index, further elements cannot be added, even If space is available due to deletion of elements from *front*. Thus space utilization is poor.
- Circular queue allows adding elements at the start of array if *rear* reaches last index and space is 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 incr
r++;
if (r==max-1)
    r=0;
```

```
circular incr
r=(r+1)%max
```

(-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

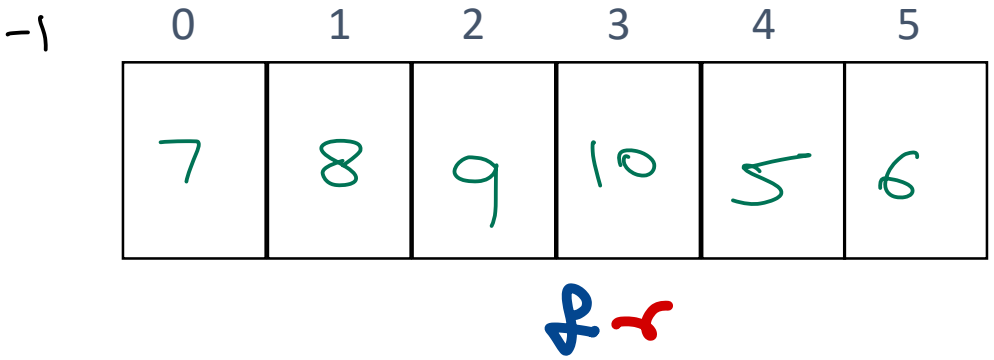
effective queue: f+1 to r

init:

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

peek:

```
i = (f+1) % max
val = arr[i];
```



push:

```
r = (r+1) % max
arr[r] = val;
```

empty:

```
f == r
```

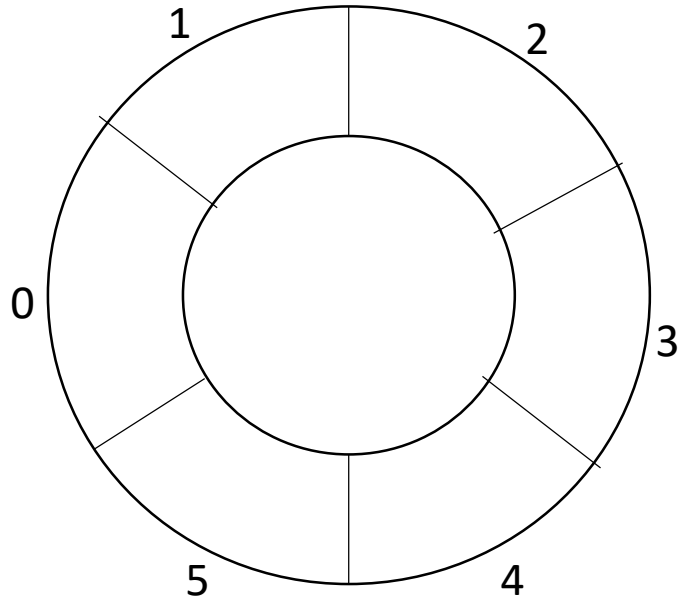
pop:

```
f = (f+1) % max
val = arr[f];
```

full:

```
f == r
```

Same condn



que full

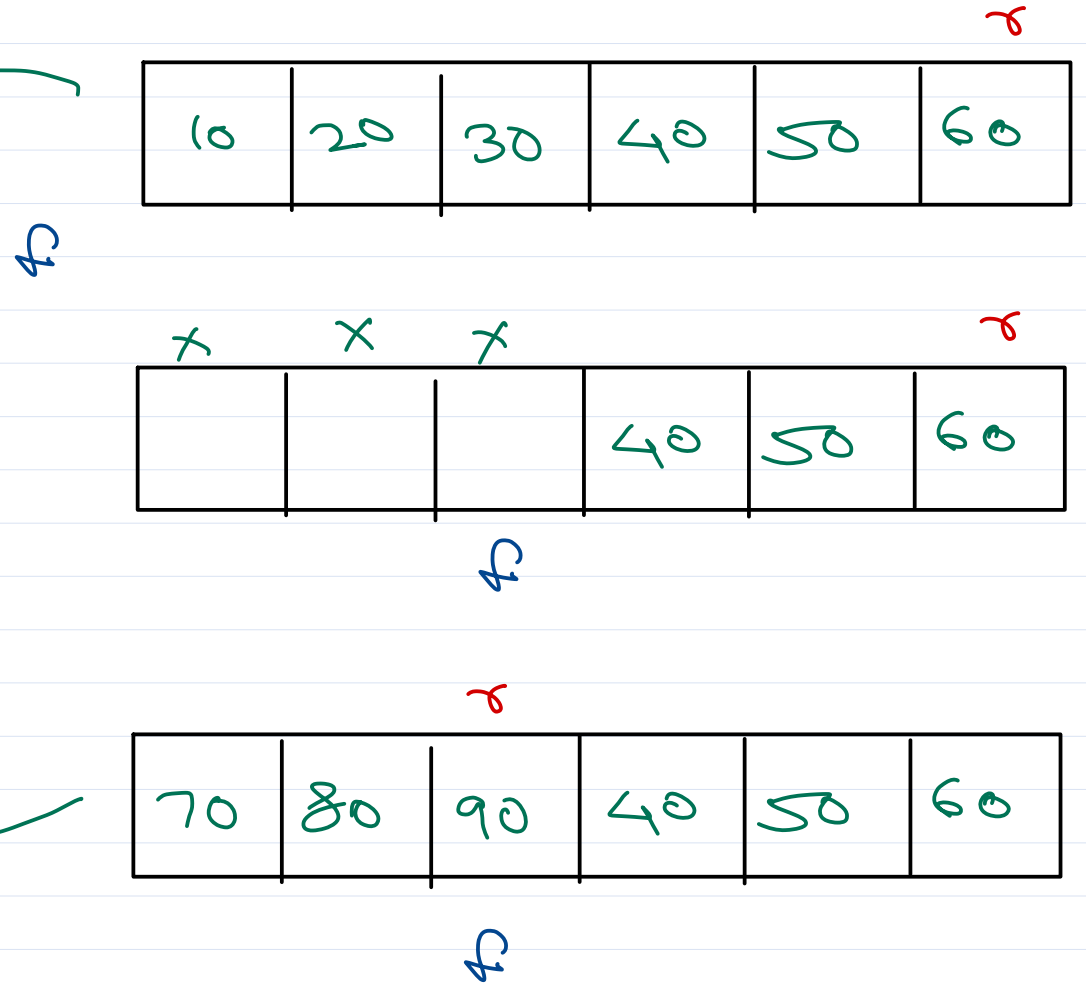
$(f == -1 \;\&\&\; r == max-1)$

OR

$(r == f \;\&\&\; r != -1)$

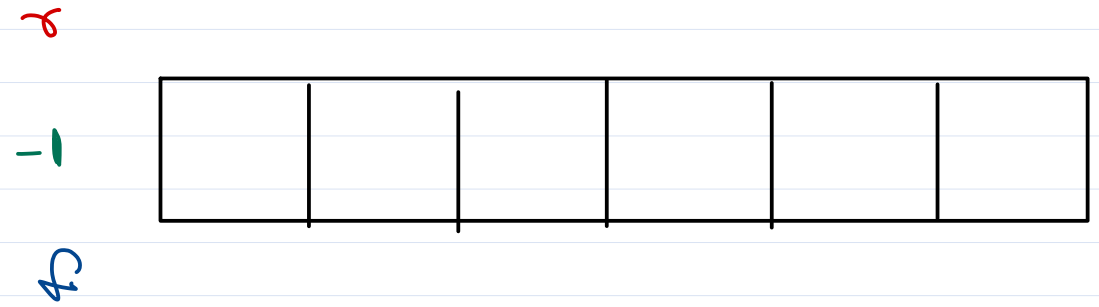
push:

```
r = (r+1) % max
arr[r] = val;
```



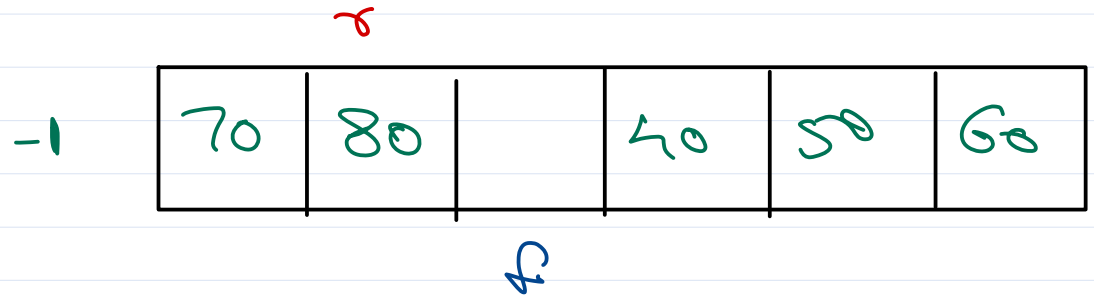
Que empty

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



pop:

```
f = (f+1) % max
val = arr[f];
if(f == r) {
    f = -1;
    r = -1;
}
```



Circular Queue

effective queue: f+1 to r

init:

```
r = -1
f = -1
cnt = 0
```

peek:

```
i = (f+1) % max
val = arr[i];
```

push:

```
r = (r+1) % max
arr[r] = val;
cnt++;
```

empty:

```
cnt == 0
```

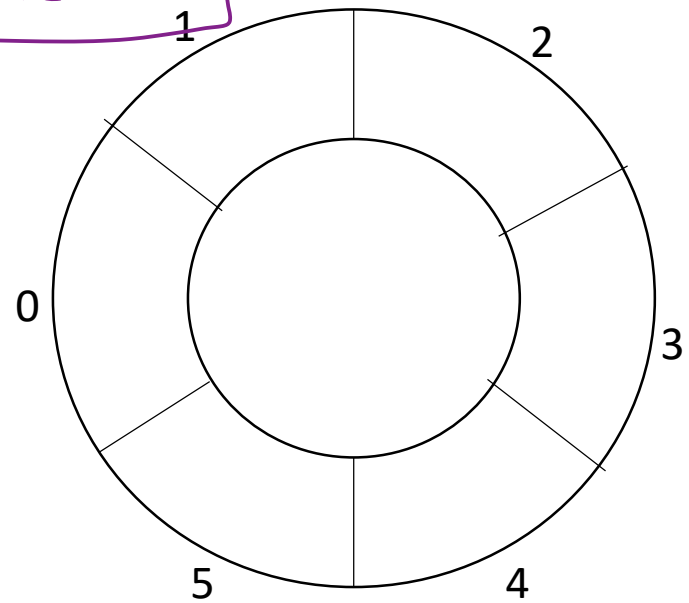
full:

```
cnt == max
```

pop:

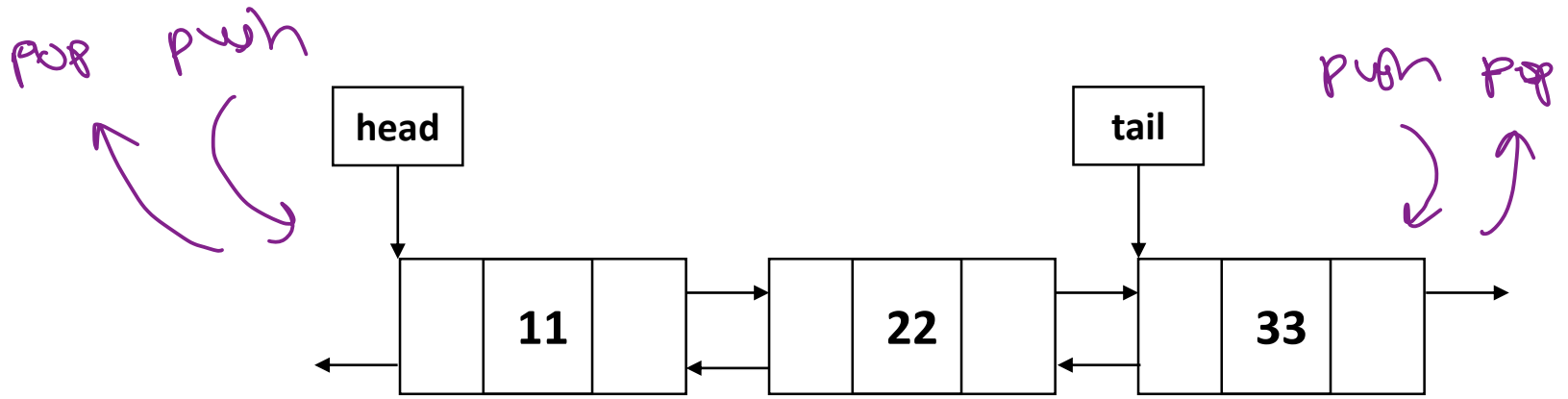
```
f = (f+1) % max
val = arr[f];
cnt--;
```

Homework



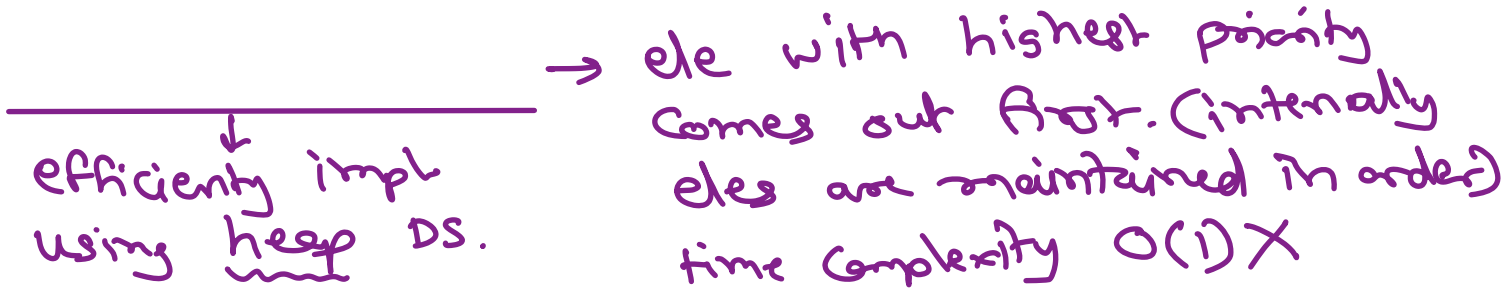
DeQueue

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



Priority queue

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



Stack effective stack ele: @ to top.

init:

```
top = -1
```

is full:

```
top == max-1
```

push:

```
top++;
arr[top] = val;
```

is empty:

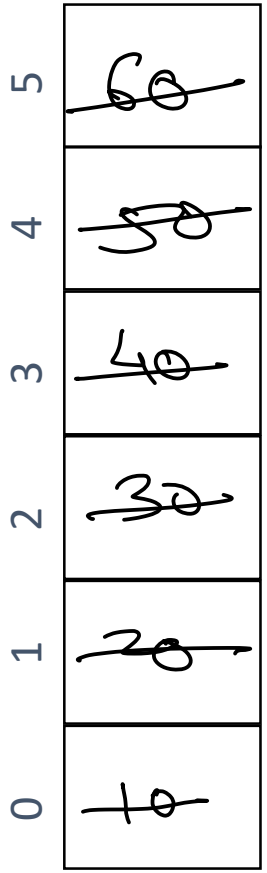
```
top == -1
```

pop:

```
val = arr[top]
top--;
```

peek:

```
val = arr[top]
```



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>