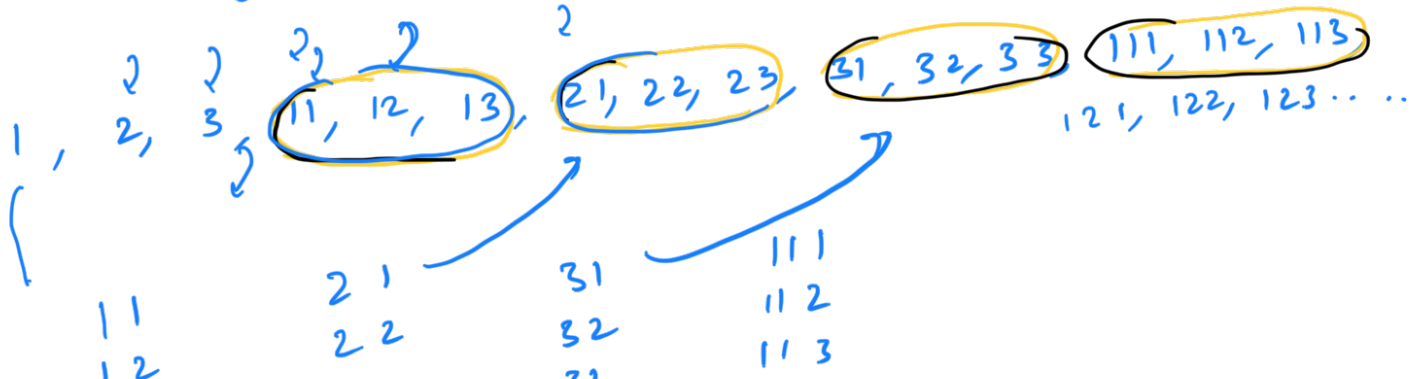
Queues

Question: Generate k numbers in sorted using only digits 1, 2, 3

Print the first k numbers

k = 15

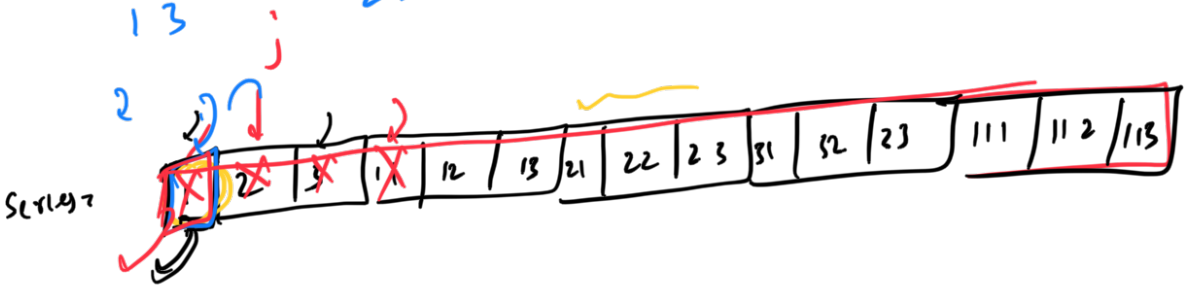


11
12
13

21
22
23

31
32
33

111
112
113

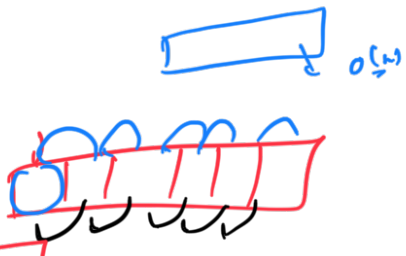


1 2 3 11

Operations,

1) Deleting first element of the array => O(n)

2) Append numbers to the end => O(1)



O(1)

Queue DS

Does operations in just O(1)

Queue :

A sequence of items / people waiting for their turn to be processed.

BFS: Queue

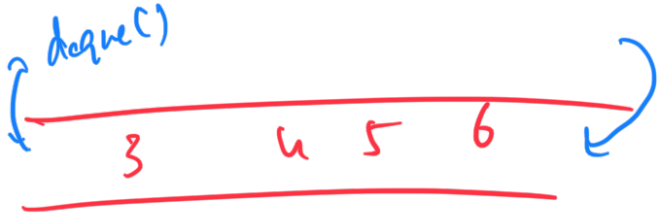
Operations

1) Enqueue(x) : Inserts an element at end of Queue
2) Dequeue() : Deletes the first element from queue
3) front() : Returns the first element
4) isEmpty(): Is the Queue Empty or not

Implementation:

enq 5, 7, 3,

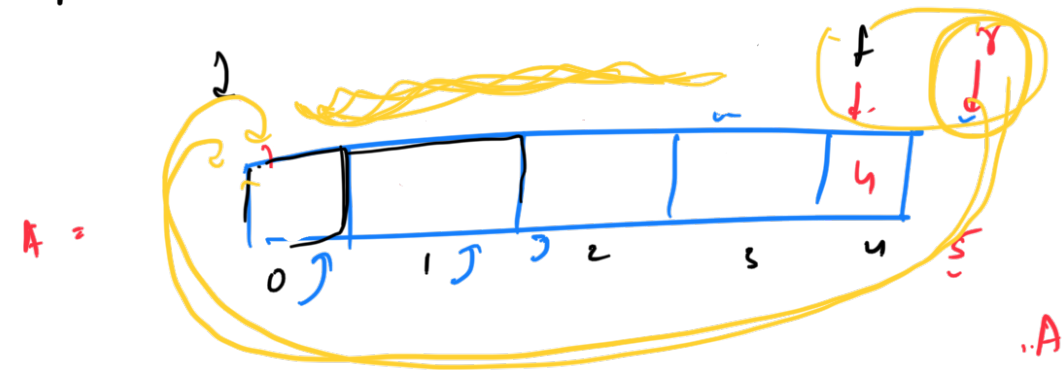
deq(), enq(2), deq, deq, isEmpty() → False
deq(), isEmpty() → true.



Which D.S ?

1) Array

Enq 5, 7, deq(), isEmpty() deq() isEmpty() enq(2) 8 4, deq(), front()



f = 0 } (f = = r)
r = 0

A[f] = 5

Enqueue :

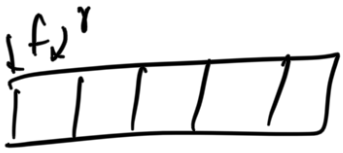
```
{ if(r == N){
    "Overflow";
  }
  A[r] = x;
  r++;
```

A[r] = 5 ✓

```
frontH(){
  if( isEmpty() )
      UNDERFLOW
  else
      return A[f]
```

Dequeue :

```
if( isEmpty() )
    "UNDERFLOW"
else
    f++;
```



Issues

1) Unused space

Circular Queue

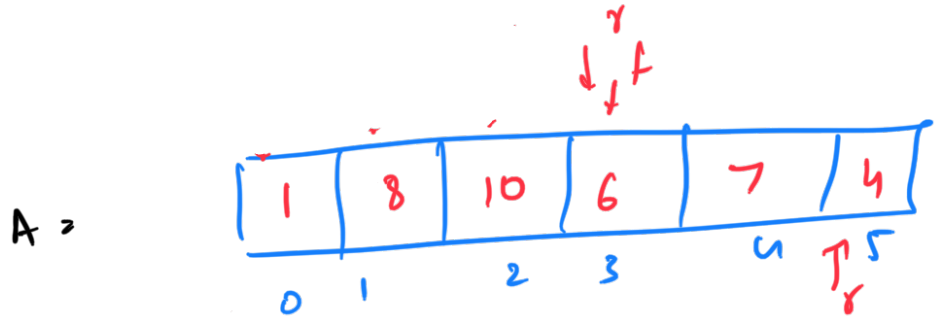
enq 3 2 5 6 7

deq deq deq() enq 4

enq 1 enq 8 enq 10

r = 6

r = (r + 1) % N



flag = 0

if(f == r and flag == 0)
Queue is Empty

if(f == r && flag == 1)
Queue is FULL

int flag = 0;
f = 0, r = 0;

isEmpty( ) {
if(f == r && flag == 0)
return true;
else false. O(1)
}

isFull() {
if(f==r && flg==1)
return true
return false.
O(1)
}

Enqueue (x) {
if(isFull()) {
overFlow; O(1)
}
A[r] = x;
r = (r+1) % N;
if(f == r) flag = 1;
}

Dequeue ( ) {
if(isEmpty())
return "UNDERFLOW"
f = (f+1) % n
if(f == r) flag = 0;
}

O(1)

(stl)

T.C:

Implement Queues using Stacks

push()
pop()
top()

enq 4 5 3 deq deq enq 14 9 deq

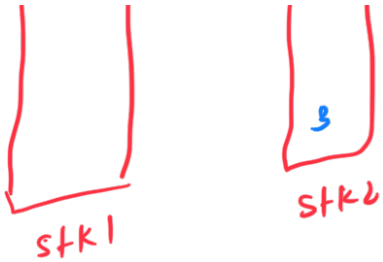
Visualization

3

queue

Approach 1:

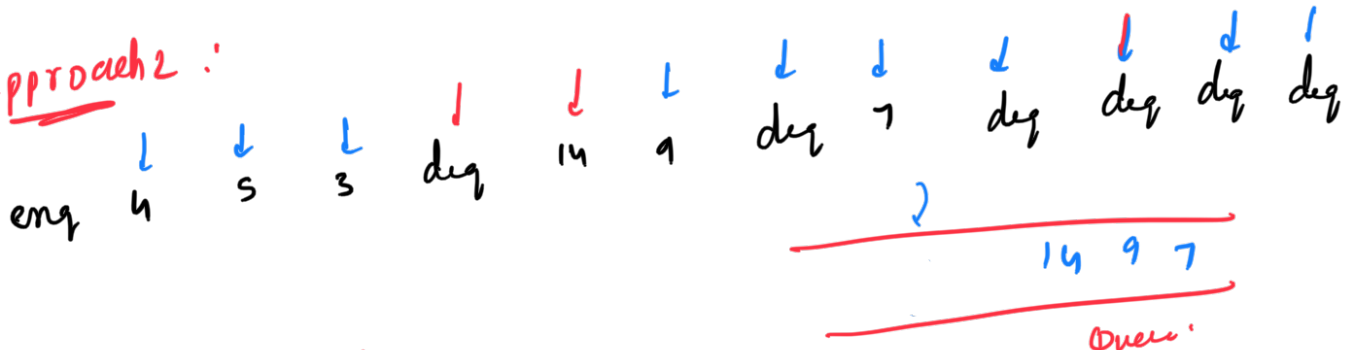
[ ] [ ] [ ]

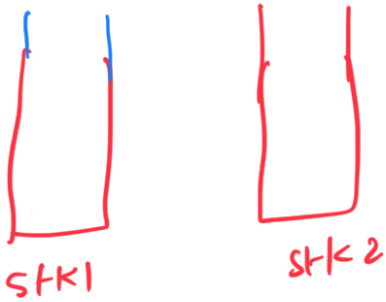


Enqueue():
push to stk1 : O(1)

Dequeue(); => O(n)

Approach 2 :





Enqueue(x):
Push to stack 1

Dequeue():
if (stack2 is not empty)
stack2.pop();

else
copy elements from stack1 to Stack2
Pop the top element.

=> total T.C for all the n operations!

Consider any element
-> Pushed to stack 1
-> Pop from stack1
stack 2

} 4 operations

-> Push to
-> Pop from Stack 2

For n elements => n operations

For one enque / deque

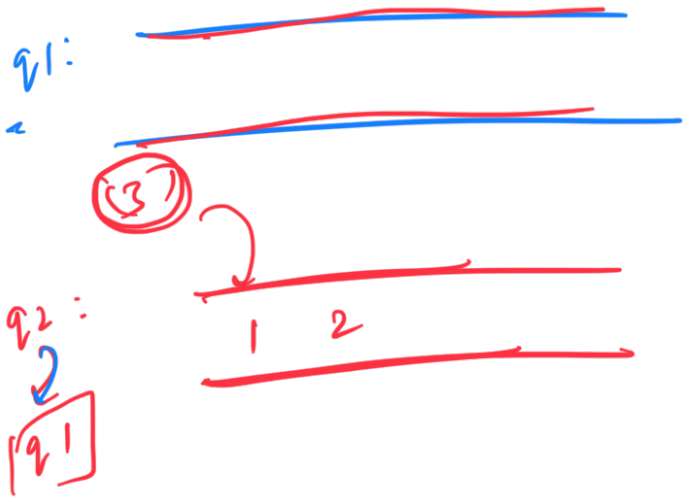
Amortized T.C : O(1)

Stacks Using Queues

Enqueue(x)
Dequeue()
front()

push 1, 2, 3, (pop)

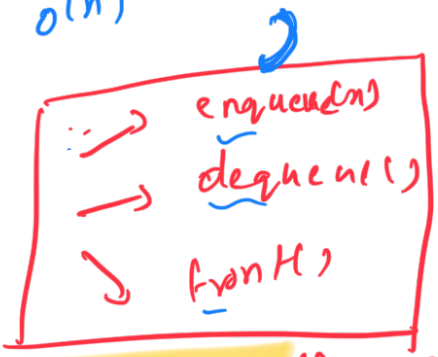
stack



Push :
Enqueue in q1
O(1)

Pop :
O(n)

Issues with Queue



eng 4 5 6 7

=> Cannot undo a wrong Enqueue / Dequeue operation

Doubly Ended Queue

→ Operations are done at both the ends
→ Insert from both the ends
→ delete from both the ends

1) Enqueue at front
2) Enqueue at rear
3) Dequeue at front
4) Dequeue at rear

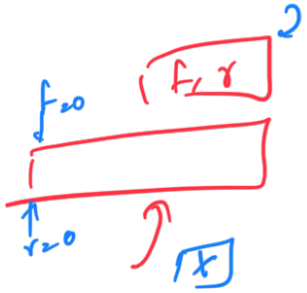
} 4 operations

EF(1) EF(3) EB(4) EB(7) EB(2) EF(-4)

DF() DF() DB()

147

Implementation (HW)



Question: Sliding window Max

K = 3
N = 6

A = 1 3 2 5 7 6

Output = [3 5 7 7]

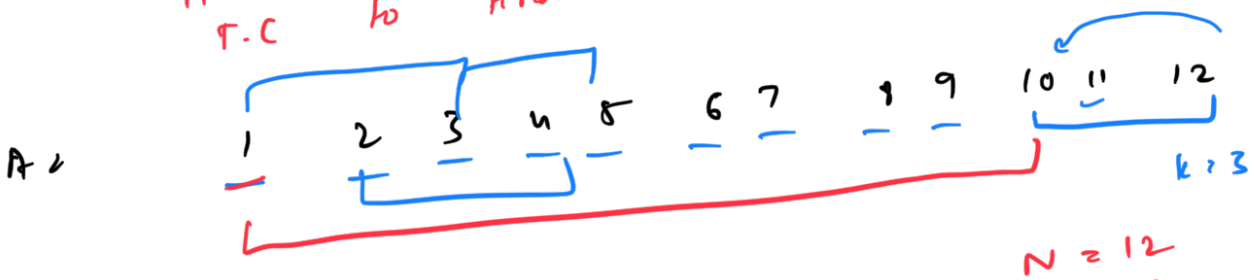
Brute Force:

For every subarray of size k, iterate and find the max value

T.C:

No. of subarrays of size K = N - K + 1

T.C to find max in a subarray = O(k)



N = 12
K = 3
N - K + 1 = 12 - 3 + 1 = 10

T.C: O((n-k) x K)

K = N/2

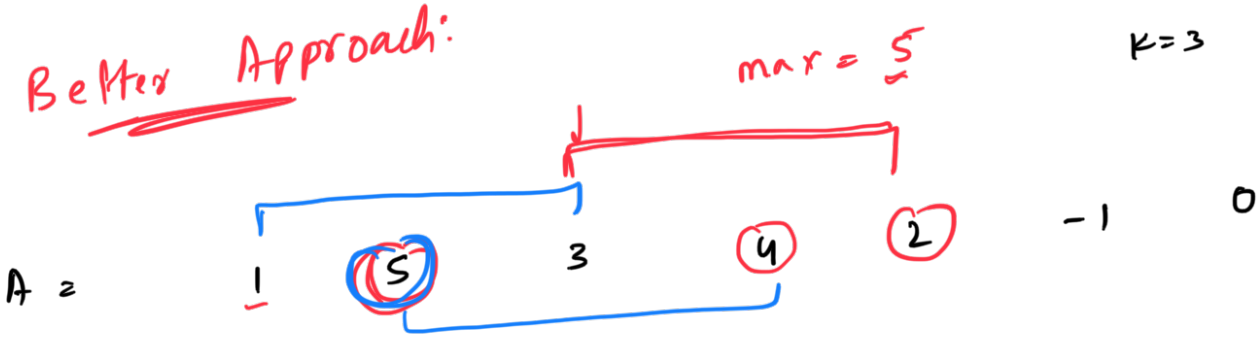
O(n/2 * n/2) => O(n^2)

K = N

O(n-k+1) x k)
O(N-N+1) x N
= O(N)

Better Approach:

K = 3



Does it make sense to have some sort of a queue to get the next max element current max goes out of window.

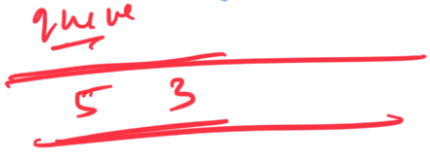
when the

Observations:

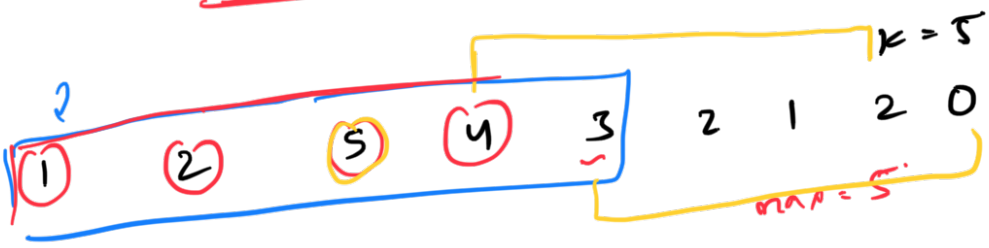
1) 1 cannot be the max of any window in future

2) 3 can be the max of a window in future

queue



Ex A =



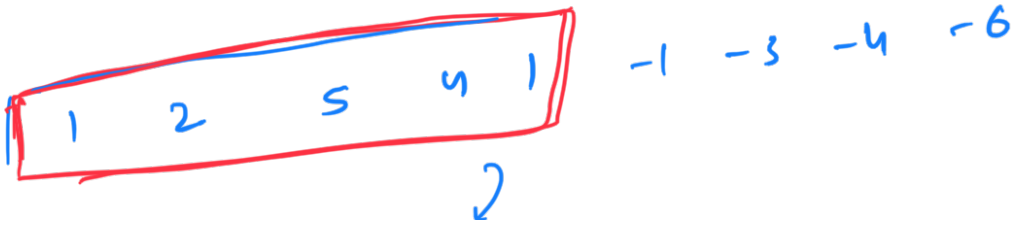
queue:

Queue is in descending order

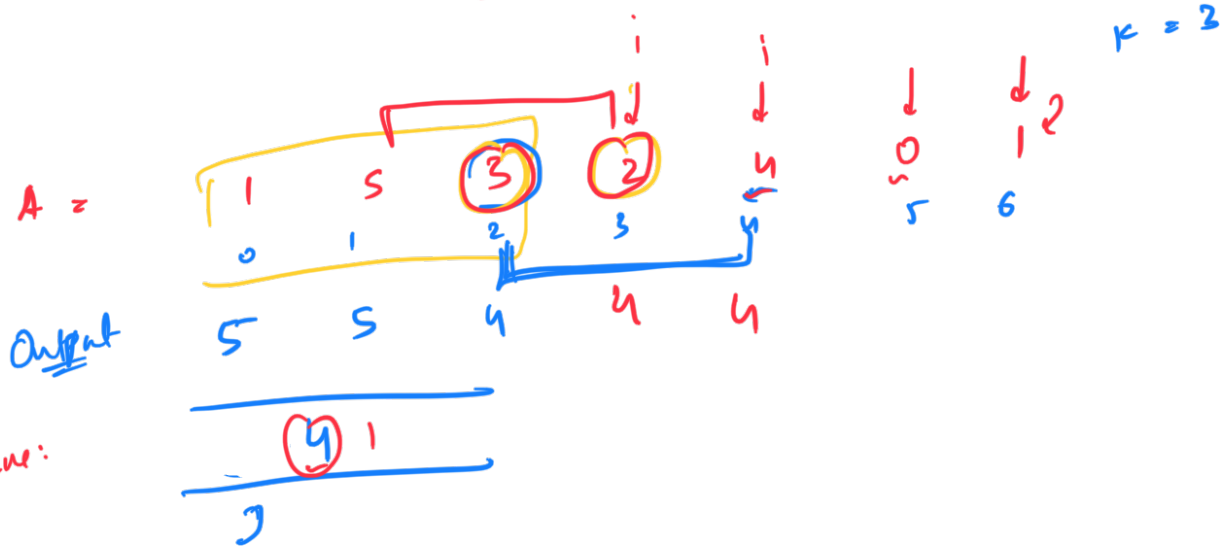
A =

queue:

A =



queue: 5 4 1



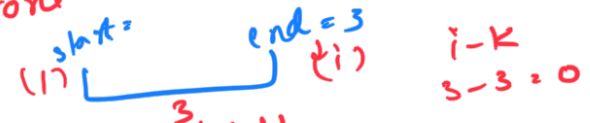
Store indices in the queue

What all operations are we doing?

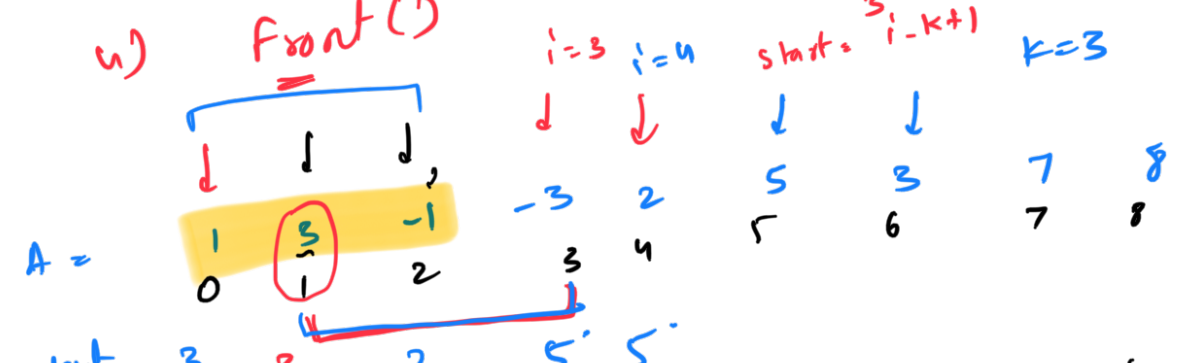
1) Enqueue at back
2) Dequeue at back
3) Dequeue from front.

} Queue!

Doubly ended Queue



4) Front()



(ind, val)

queue: (5,5) (6,3)

top = 1

i = 3

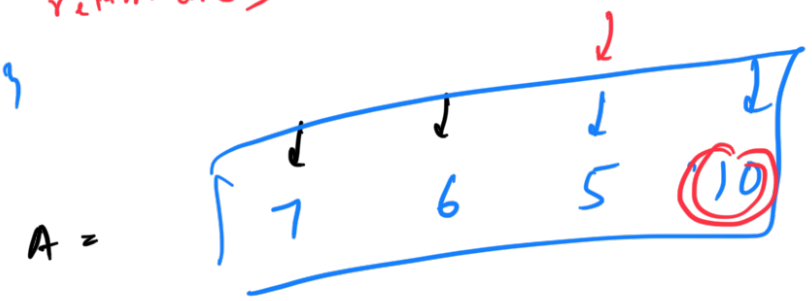
3 - 3 + 1 = (1)

start = i - k + 1

4 - 3 + 1 = (2)

```
slidingMax( int arr[], int N, int K) {
    vector<int> ans;
    deque<int> dq;
    for(i=0; i<K; i++) {
        while(!dq.empty() && a[i] >= a[dq.back()])
            dq.pop_back();
        dq.push_back(i);
    }
    ans.push( a[dq.front()] )

    for(i=K; i<n; i++) {
        start = i-K+1;
        while(!dq.empty() && a[i] >= a[dq.back()])
            dq.pop_back();
        dq.push_back(i);
        while(!dq.empty() && dq.front() < start)
            dq.pop_front();
        ans.push( a[dq.front()] );
    }
    return ans;
}
```



7 6 5 _

Queue Implementation using arrays

```
MAX_SIZE = 10;
int arr[SIZE];
f = 0, r = 0;

bool isEmpty(){
     if(f == r)
          return true;
     return false;
}

void enqueue(x){
     if(r == MAX_SIZE){
          print('Overflow');
          return;
     }
     arr[r] = x;
     r++;
}

void dequeue(){
     if(isEmpty()){
          print('Underflow');
          return;
     }
     f++;
}
```

Circular Queue

```
int arr[MAX_SIZE];
int l = 0, r = 0;
int flag = 0;
bool isEmpty(){
     if(f == r && flag == 0)
          return true;
     return false;
}
bool isFull(){
     if(f == r && flag == 1)
          return true;
     return false;
}
```

```
 void enqueue(x){
      if(isFull()){
           print('Overflow');
           return;
      }
      arr[r] = x;
      r = (r + 1)%N;
      if(f == r) flag = 1;
 }

 void dequeue(){
      if(isEmpty()){
           print('Underflow');
      }
      f = (f + 1)%N;
      if(f == r) flag = 0;
 }
```

Queues using stacks

```
stack<int> stk1, stk2;
void enqueue(int x){
     stk1.push(x);
}
void dequeue(){
     if(!stk2.isEmpty())
          stk2.pop();
     else{
          while(!stk1.isEmpty()){
               top = stk1.top();
               stk1.pop();
               stk2.push(top);
          }
          if(!stk2.isEmpty())
               stk2.pop();
          else
               return "Underflow"
     }
}
```

Stacks using queues

```
queue<int> q1, q2;
void push(int x){
     q1.push(x);
}

void pop(){
     while(!q1.isEmpty()){
          int f = q1.front();
          q1.pop();
          if(!q1.isEmpty())
               q2.push(f);
     }

}
```

Question : Sliding window maximum

```
slidingMax(int a[], int n, int k){
     vector<int> ans;
     if(n < k)
          return ans;
     deque<int> dq;
     // First window of size k
     for(int i = 0 ; i < k; i++){
          while(!dq.empty() && a[i] >= A[dq.back()])
               dq.pop_back();
          dq.push_back(i);
     }
     ans.push_back(a[dq.front()]);

     for(i = k; i < n; i++){
          while(!dq.empty() && a[i] >= A[dq.back()])
               dq.pop_back();
          dq.push_back(i);

          start = i - k + 1;
          while(!dq.empty() && dq.front() < start)
               dq.pop_front();
          ans.push_back(a[dq.front()]);
     }

return ans;
}
```