Agenda

0) Max Frequency stack
1) Max and Min
2) Implement Queue using stacks
3) First non-repeating char

Question:

Design a Data Structure which supports the following operations

Max Frequency stack

push(x): Pushes element on the stack

pop(): Removes and returns the latest element with max frequency

freq(5) = 2
freq(7) = 2
freq(4) = 1

push 5 7 5 7 5 4

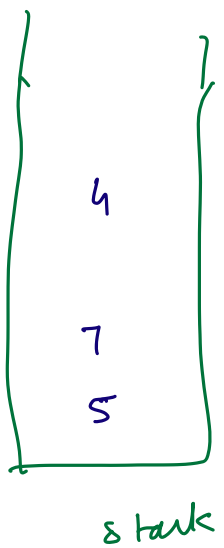
pop → 5

pop → 7

push 7 ✓

pop ✓ 7

pop ✓ 5

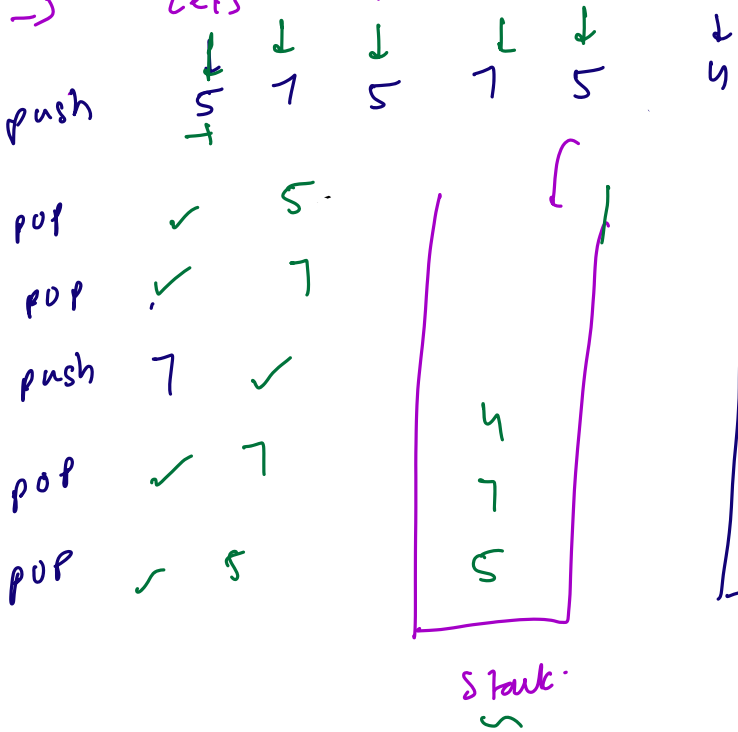


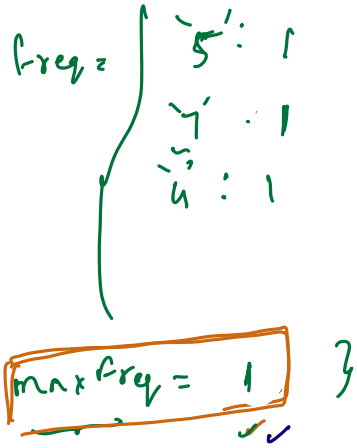
stack

Approach 1:

-> Lets maintain a hashmap for frequencies

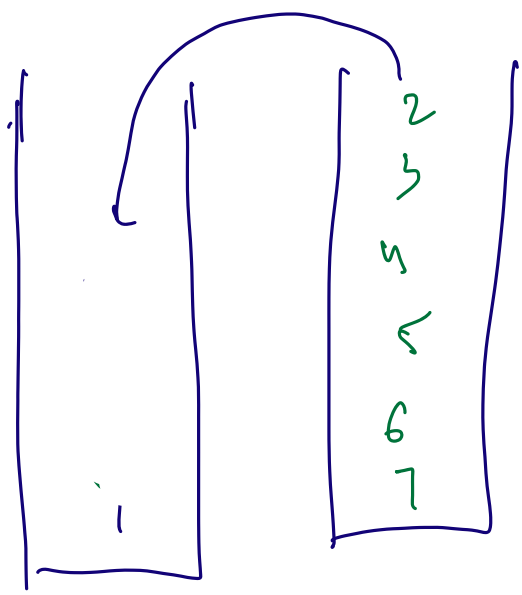
-> Lets have a variable maxFreq





Push: O(1)

Pop: O(n)



freq(2) = 3

O(N), O(1)?

maxFreq = 2

1) update freq of the element -> O(1)

2) Push on top of stack -> O(1)

Optimised Approach

- We still need frequency Map
- During pop operation, we are only concerned about elements which have maximum frequency.
- store all elements with map freq.

Hashmap < freq, Elements with this freq >

Array / LL / stack / Queue

stack

int

Hashmap < freq, stack <int> >

stack


push 5 7 5 7 5 4
pop ✓ 5
pop ✓ 7
push 7 ✓
pop ✓ 7
pop ✓ 5
pop 4
maxfreq = 1
freqMap :
5 : 1
7 : 2
4 : 1
stackMap :
1 :
2 :
7
5

T.C :

Pop Operation

→ Pop element with max freq from the respective stack.

stackMap[maxFreq].pop() ⟹ O(1)

→ update freq Map ⟹ O(1)

T.C: O(1)

(7) 1 2 3 (6)

[1,2] ⟹ 1
[1,2,5] ⟹ 2
[2,3] ⟹ 1

Question: max and Min

Find summation of |max − min| of all the subarrays.

Array of size N: $\frac{N(N+1)}{2}$

N = 3

$\frac{3(4)}{2} = 6$

A =	max	min	diff
[2]	2	2	0
[2,5]	5	2	3
[2,5,3]	5	2	3
[5]	5	5	0
[5,3]	5	3	2
[3]	3	3	0

3+3+2 = 8

Brute Force

→ Consider all $\frac{N(N+1)}{2}$ subarrays

→ Find max & min for every subarray : O(N)

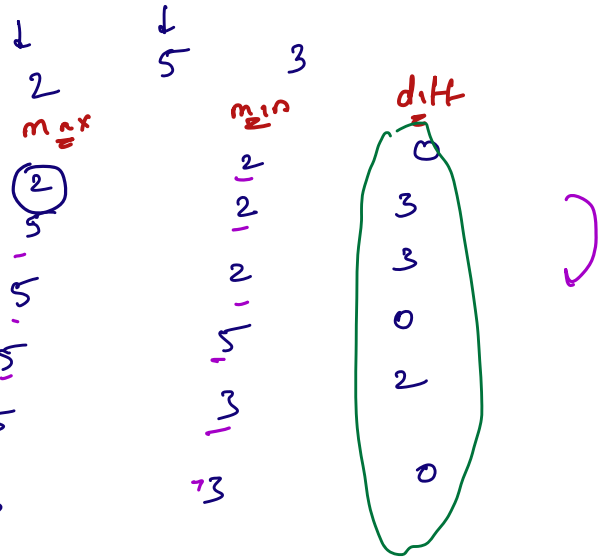
Total T.C : $\frac{N(N+1)}{2} \times O(N) = O(N^3)$

Carry Forward

T.C : $O(N^2)$

Optimised Approach

A = [2] [2,5] [2,5,3] [5] [5,3] [3]



Max = 2 1 1 5 4

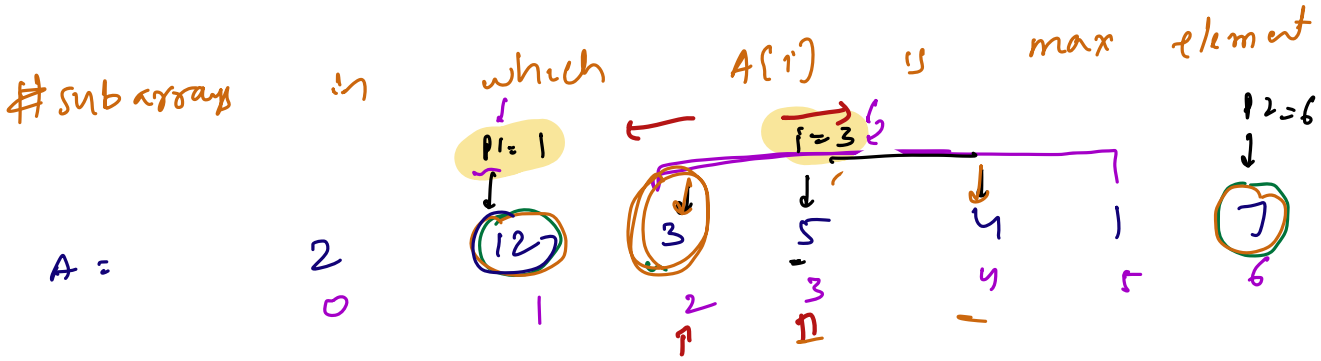
Min = 3 2 1

Contribution = $2^*1 - 2^*3 = 2(1-3)$ + $5^*1 - 3^*2$ $3(1-2)$ + $5^*4 - 5^*1$ $5(4-1)$

ans = sum of [A[i] * (#max - #min)] for i

How to find # subarrays in which an element is max / min

subarrays in which A[i] is max element



[3, 5] [3, 5, 4] [3, 5, 4, 1] } = 6

[5] [5, 4] [5, 4, 1]

Accepted starting points : 2, 3 ⇒ 2 starting

(i - p1) = 3 - 1 = 2

Accepted Ending point : 3, 4, 5 ⇒ 3 ending point

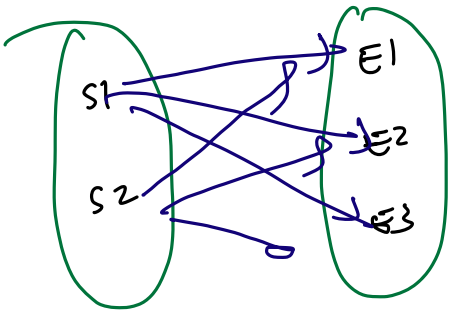
i = 3, p2 = 6

(p2 - i)

2 Start points

3 end points

2 * 3

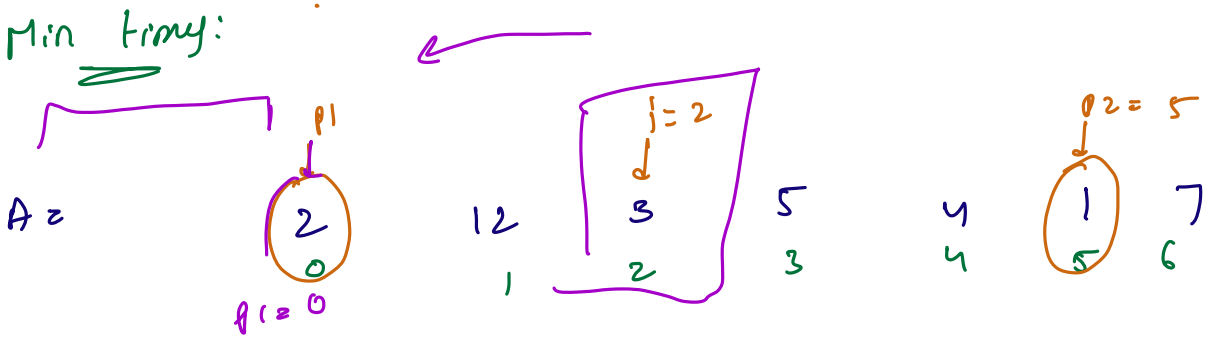


2 * 3 = 6

subarrays in which A[i] is max = $(i-p1) \times (p2-i)$

ans += $(i-p1) \times (p2-i) \times A[i]$;

Min trivy:



[12, 3] [12, 3, 5] [12, 3, 5, 4] } = 6

[3] [3, 5) [3, 5, 4]

starting = $(i-p1)$ = $(2-0) = 2$

endly = $(p2-i)$ = $(5-2) = 3$

sub arrays = $2 * 3 = 6$

ans += $(i-p1) \times (p2-i) \times A[i]$;

```
int maxMin(int a[], int N){
int ans = 0;
for(i=0; i<N; i++){
// #subarrays in which A[i] is max
p1 = prevGreater[i];
p2 = nextGreater[i];
ans += (i-p1)*(p2-i)*A[i];
// #subarrays in which A[i] is min
p1 = prevSmaller[i];
p2 = nextSmaller[i];
ans -= (i-p1)*(p2-i)*A[i]
}
return ans;
}
```

O(n)

1) Construct 4 arrays => T.C O(N) => O(N)

2)

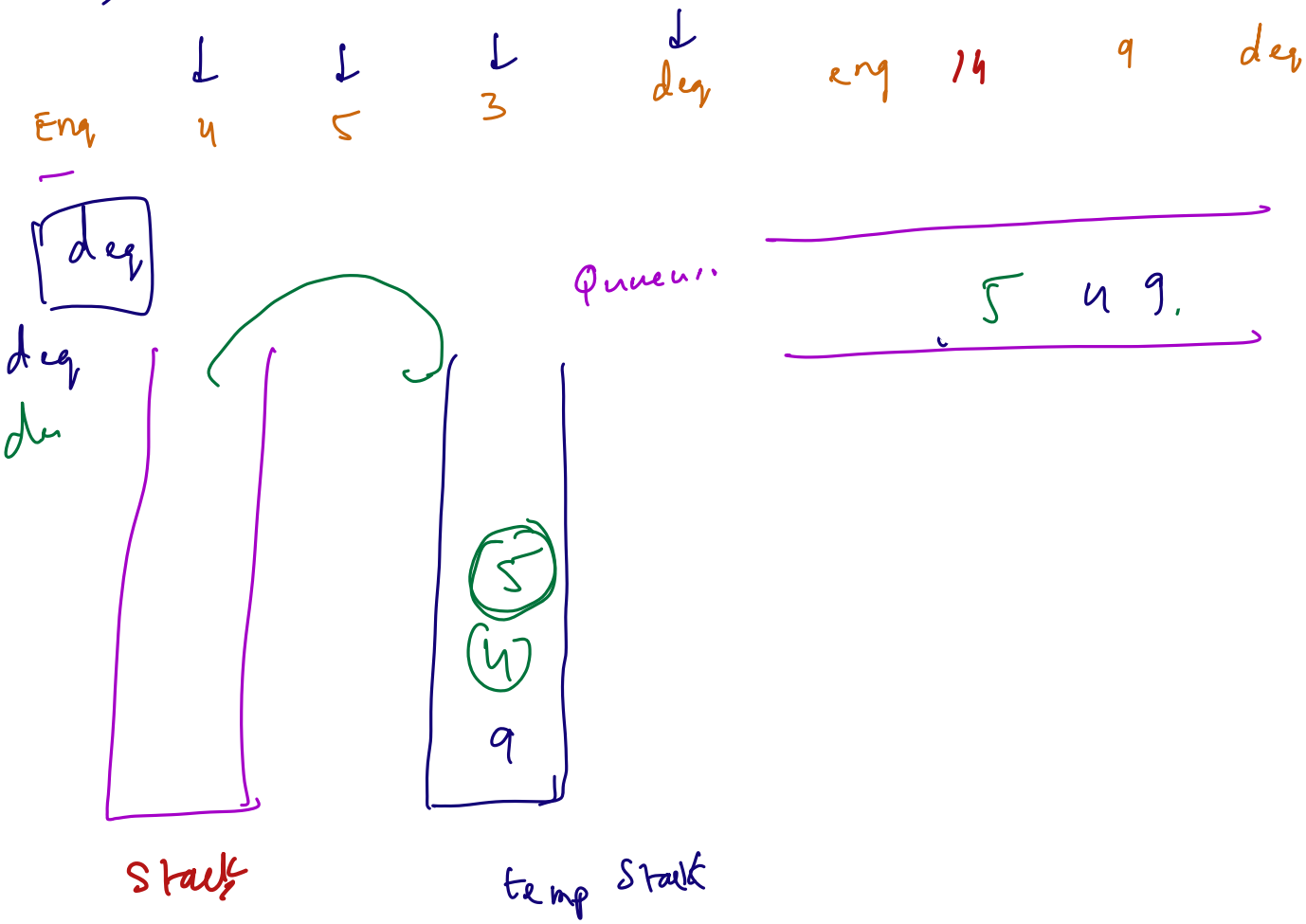
O(N)+ O(N)

T.C: O(N)

S.C: O(N)

Implement Queue using Stacks FIFO

→ Enqueue()
→ Dequeue()



T.C for enqueue : O(1)

T.C for dequeue : O(N)

Approach2 :

→ We need 2 stacks

→ deq

→ deq

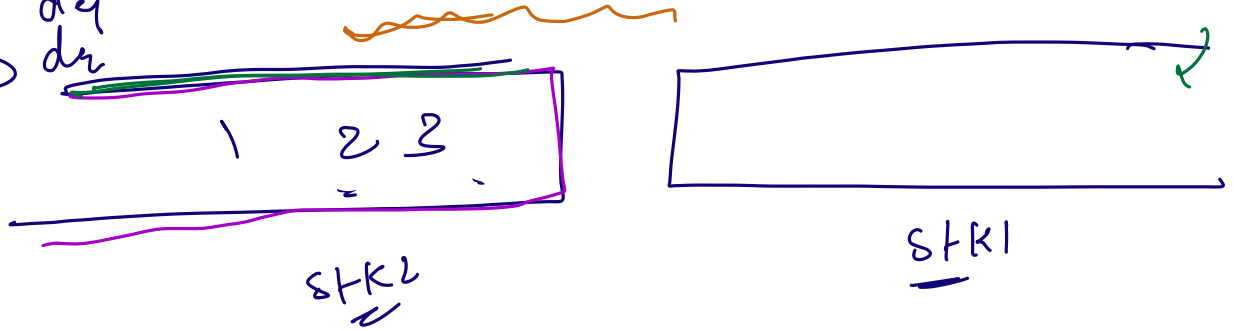
→ enq 1, 2, 3

→ deq

→ deq

Queue

9 , 1, 2, 3



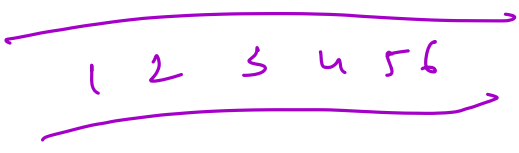
(N)

deq :

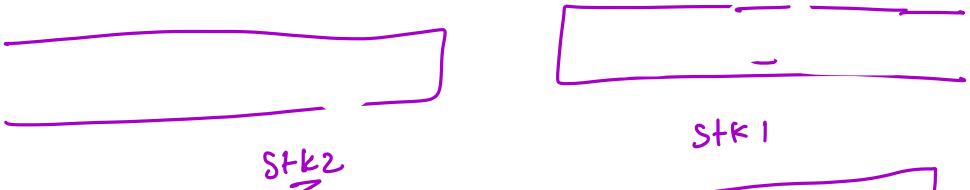
O(1) worst Case

Enqueue :

stk1.push() ⇒



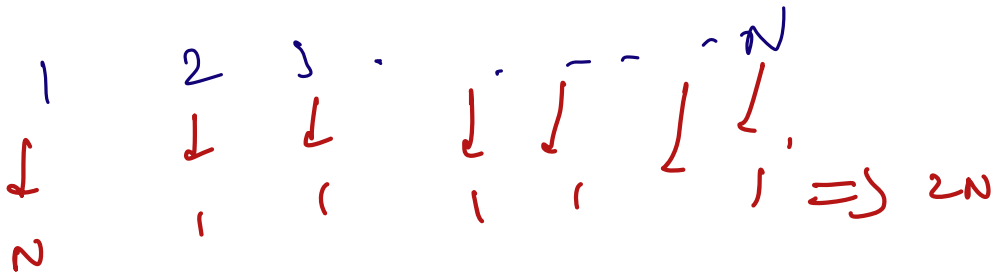
Dequeue :



O(N) worst Case

deq :

deq() :



2) O(N) for 'N' element

T.C per operation = O(1) Amortized

Eg 1 2 3 4 5 6

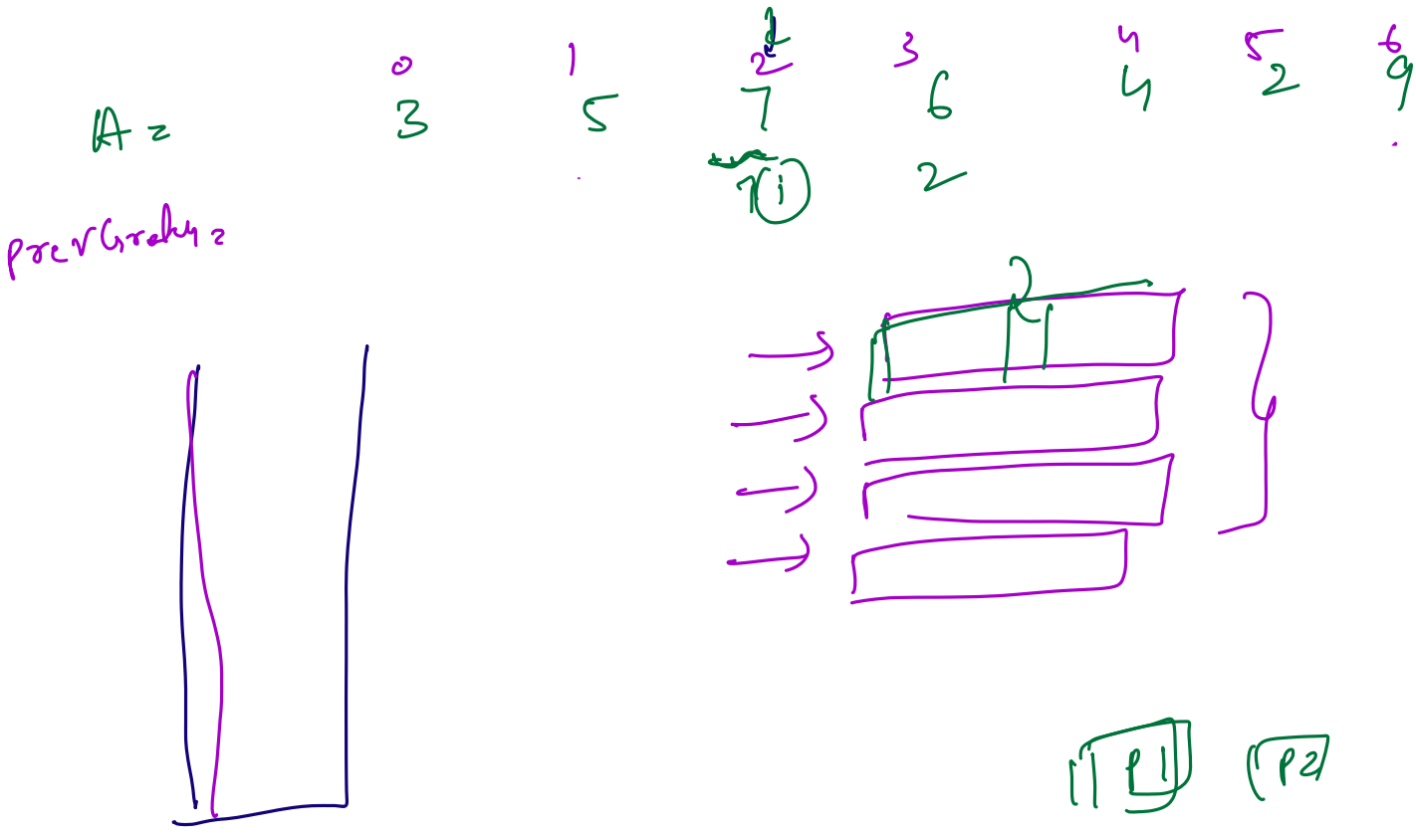
Way 2

1) Push to Stack1
2) Popped from stack1
3) Push to Stack2
4) Pop from stack2.

N element => 4N

N element => O(N)

O(1) Amortized



Question: Max and Min

```
int maxMin(int a[], int n){
      sum = 0;
      for(int i = 0 ; i < n; i ++){
            p1 = prevGreater[i];  // Get the index
            p2 = nextGreater[i];
            num_arrays = (i - p1) * (p2 - i);
            sum += num_arrays * a[i];

            p1 = prevSmaller[i];  // Get the index
            p2 = nextSmaller[i];
            num_arrays = (i - p1) * (p2 - i);
            sum -= num_arrays * a[i];
      }
return sum;
}
```

Max frequency stack

```
maxFreq = 0;
void push(int x){
      if(x in freqMap)
            freqMap[x] += 1;
      else
            freqMap[x] = 1;

      if(freqMap[x] > maxFreq)
            maxFreq = freqMap[x];

      stackMap[freqMap[x]].push(x);
}

int pop(){
      int top = stackMap[maxFreq].top();
       stackMap[maxFreq].pop();

      freqMap[top]--;
      if(stackMap[maxFreq].isEmpty() == True)
            maxFreq--;
      return top;
}
```

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”
      }
}
```