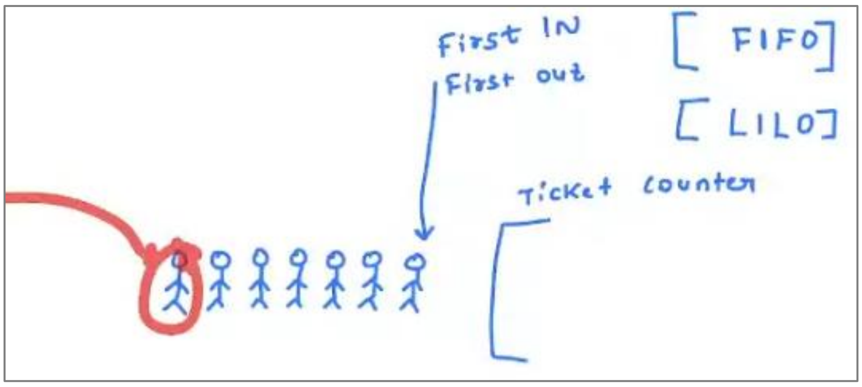
Queues: Implementation & Problems

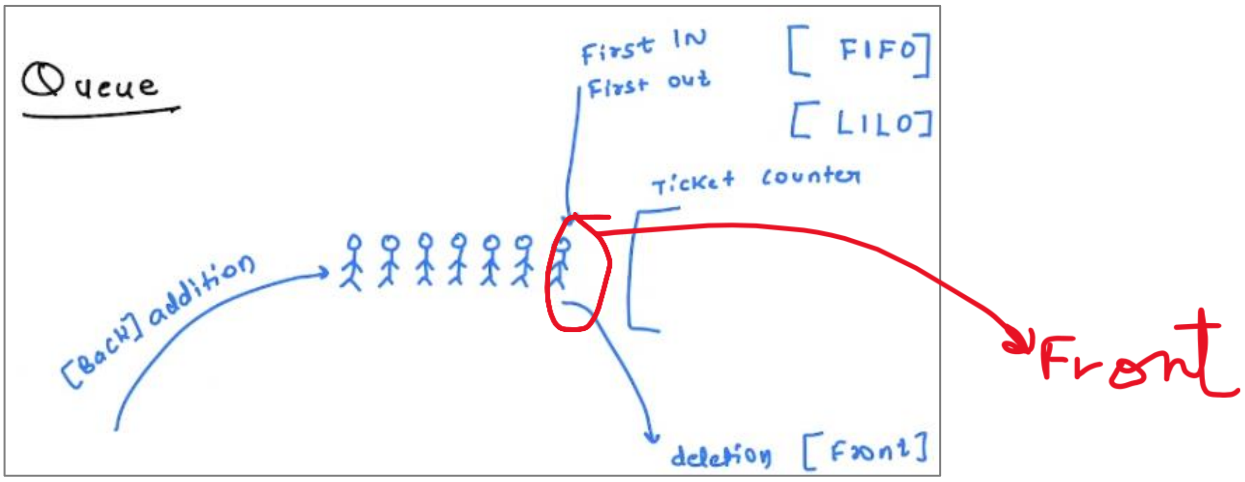
What is Queue?

Suppose in a ticket counter, people are standing in a queue. Who came first in the queue

Whoever is getting first in, will get first out



Unlike in stack where deletion and addition was done from one end only. Deletion happens form the front and addition happens at the back of the queue



FIFO: First in First out, or

LILO: Last in Last Out

Methods available [APIs]:

1. Enqueue(x) → Similar to push in stack → Insert element x in the back (rear) of queue.
2. Dequeue → Delete element from the front of the queue
3. Peek() or Front() → Return the front element
4. Size()
5. isEmpty()→If queue is empty or not → True/False

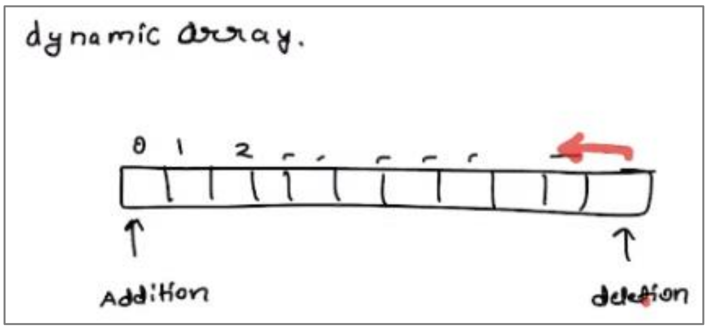
Just like in stack, you cannot access element from the middle

All these methods will be O(1) operations → TC.

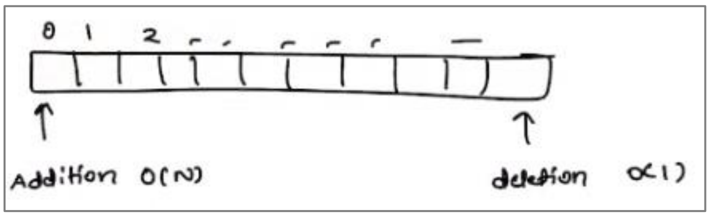
Implementation using (Dynamic) Arrays

(can be implemented using Static arrays as well) (A dynamic array is a variable-sized list data structure that can automatically resize itself when needed)

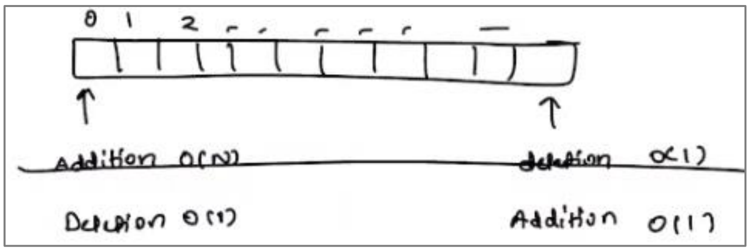
Here we will do insertion at one end and deletion at other end.



If we add in the 0th index, we have to move every element to the right so add an element to 0th index takes O(N) time



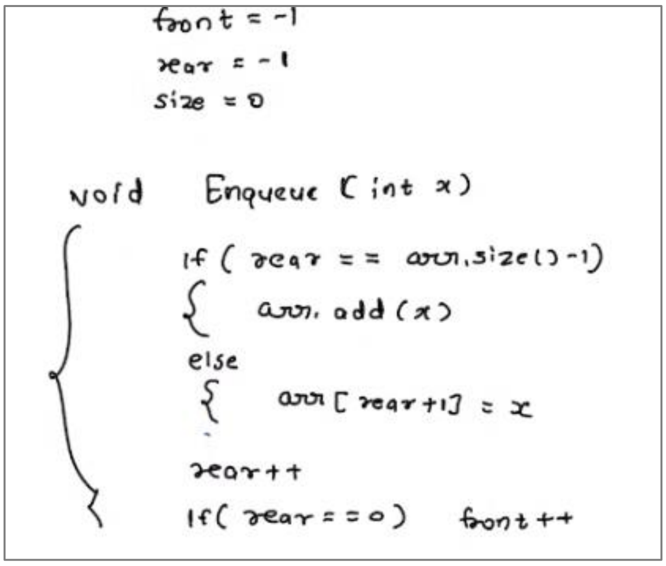
In deletion at 0th index, I will just move my front pointer to next element, i need not move/shift every element one place to the left, so this way my TC will be O(1)



In deletion we will increment our pointer from 0th index to 1st index which will take O(1) time. However we will waste some space but we will get O(1) TC on both deletion(at 0th index) and insertion (at end).

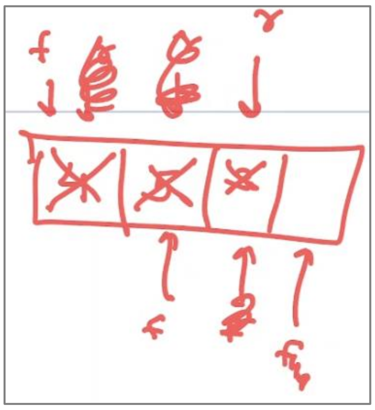
Initially, front = -1, rear = -1

If we want to add an element since we are at -1, we can add by increment an element. Since we are adding in arraylist, so we will use .add method



```
front = -1
rear = -1
size = 0

void    Enqueue ( int x)
{
        if ( rear == arr.size()-1)
        {   arr.add(x)
        else
        {   arr[rear+1] = x

        rear++
        if( rear==0)    front++
}
```

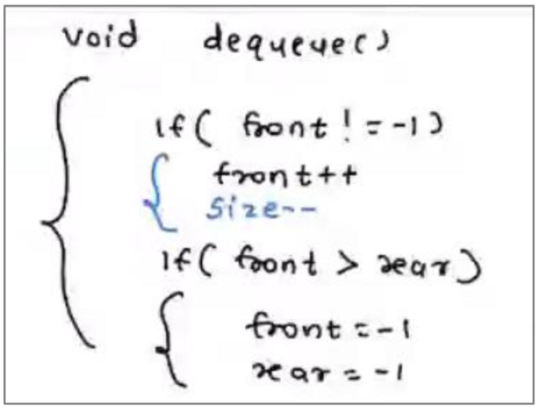
If rear is referencing the last index, then you are going to add element after the last element else add rear +1 = (-1 +1) = 0th index.



When front was at 0, rear was at 0. When we add an element, rear = 1, add an element rear =2.

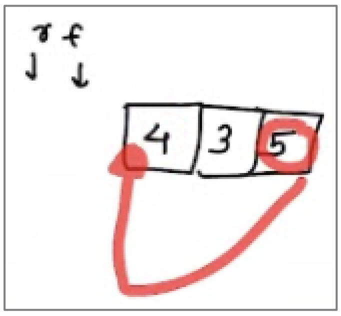
Now if we delete an element, front =1, delete an element, front = 2, delete an element front =3.

Now we see front > rear (3 >2) that means queue is empty

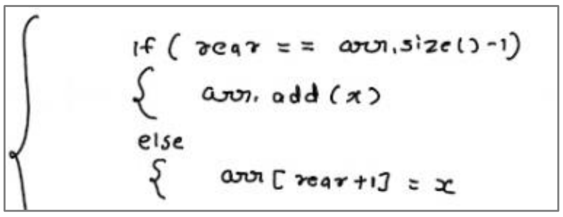


```
void    dequeue()
{
        if( front != -1)
        {   front++
            size--
        if( front > rear)
        {   front = -1
            rear = -1
```

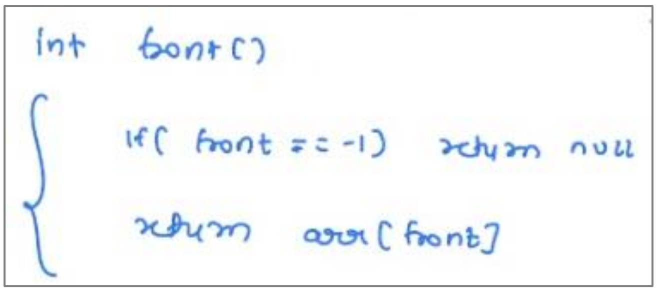
Delete element when front !=-1, coz it queue is empty then front would be -1. When queue is empty set front & rear =-1.



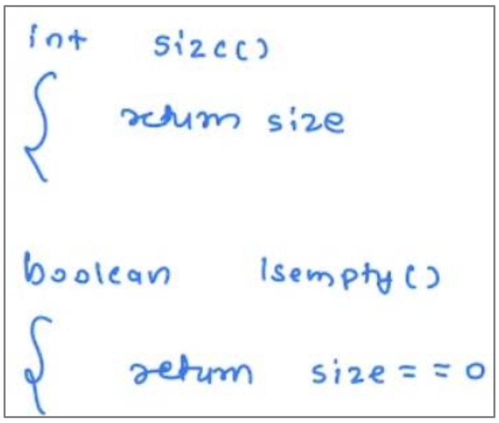
Initially array was at 0 length when we add an element 4, rear = 0, add an element, 5 rear =1. delete an element, 4 front =1, delete an element, 3 front = 2. Now since front > rear, set front & rear =-1. But when we insert new element in the array, it will add in the last element → at 2nfd index since our array size = 2. to prevent this



```
{
        if ( rear == arr.size()-1)
        {   arr.add(x)
        else
         {   arr[rear+1] = x
```



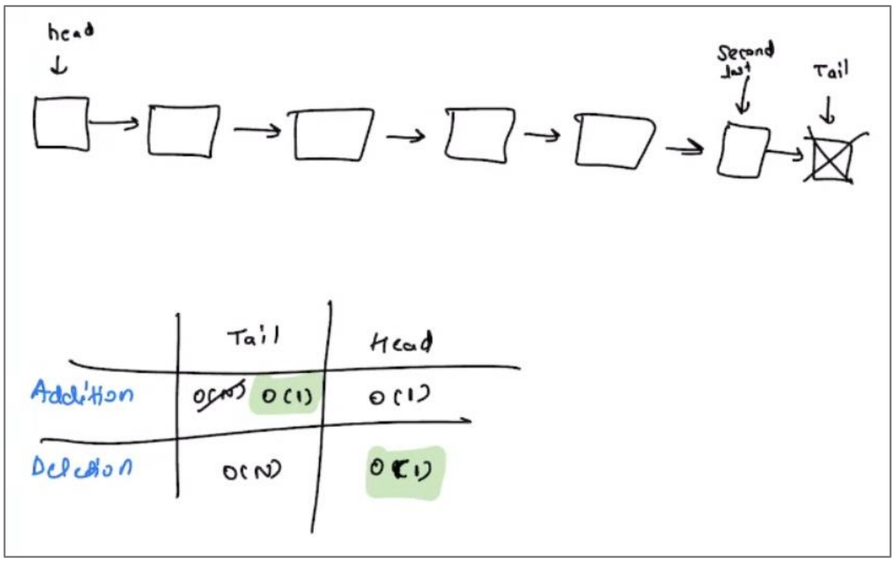
```
int  front()
{
      if( front == -1)   return  null
      return  arr[front]
```



```
int   size()
{    return size

boolean    Isempty()
{    return   size == 0
```

TC = O(1)

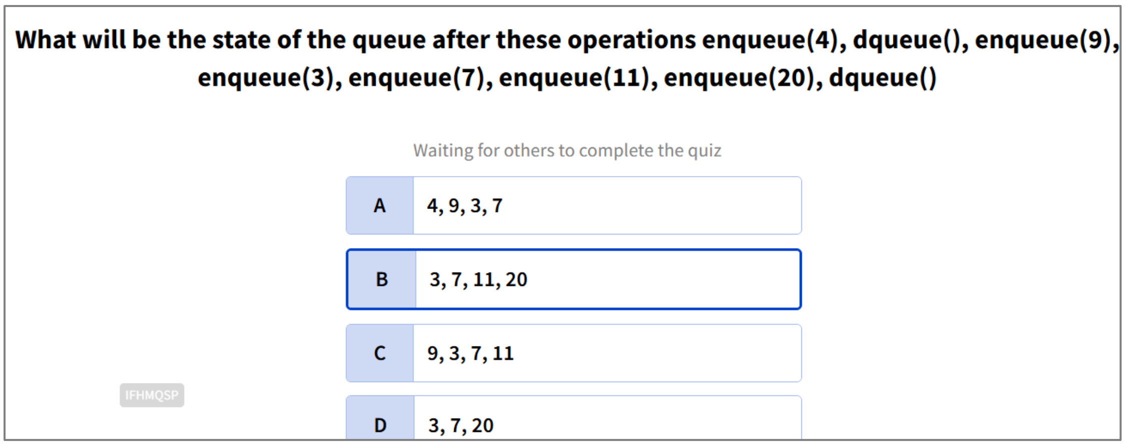
Implementation using Linked list



	Tail	Head
Addition	~~O(N)~~ O(1)	O(1)
Deletion	O(N)	O(1)

Since deletion at tail is O(N), we will use Deletion at head and Addition at Tail

Quiz

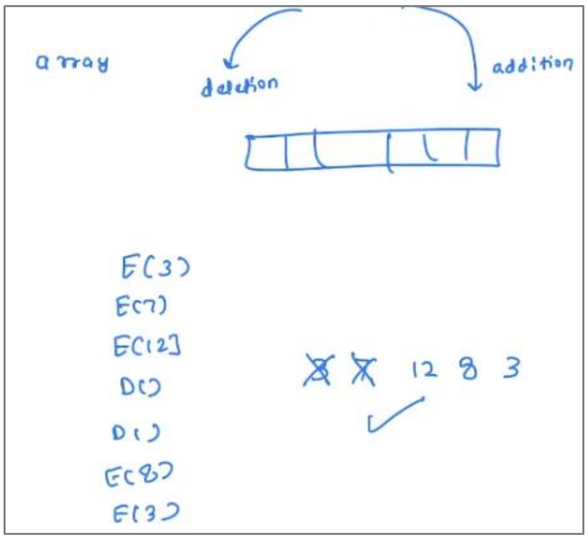


What will be the state of the queue after these operations enqueue(4), dqueue(), enqueue(9), enqueue(3), enqueue(7), enqueue(11), enqueue(20), dqueue()

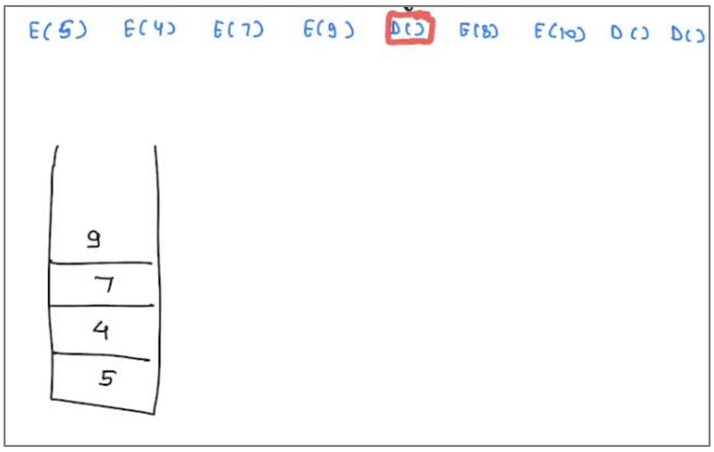
Waiting for others to complete the quiz

A	4, 9, 3, 7
B	3, 7, 11, 20
C	9, 3, 7, 11
D	3, 7, 20

IFHMQSP

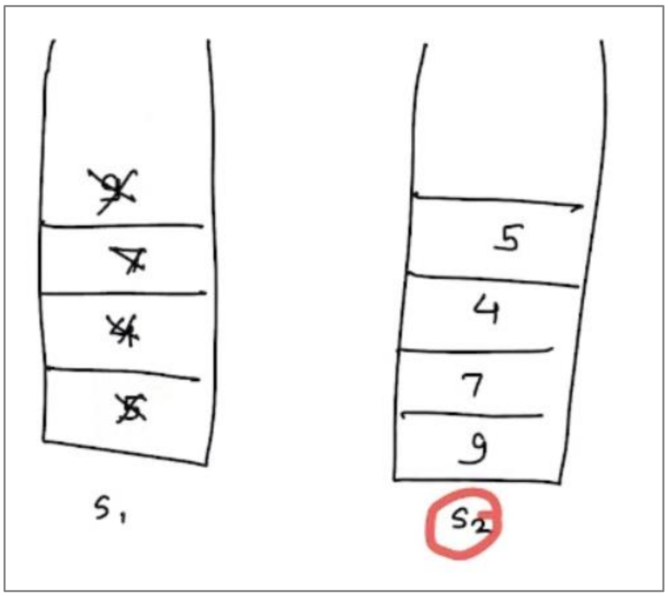


Implementation Using Stacks

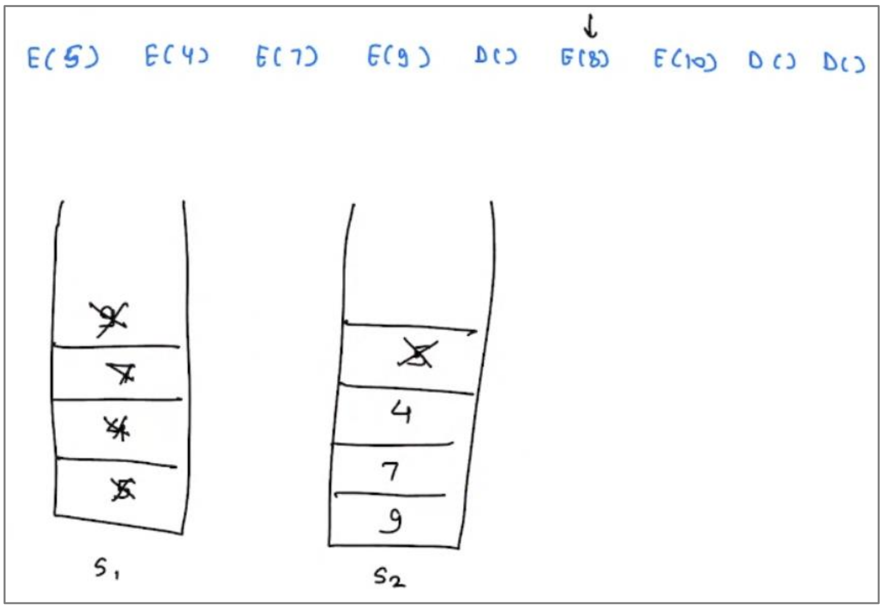


In queue element which was insert at first was deleted, now in stack how we will delete?

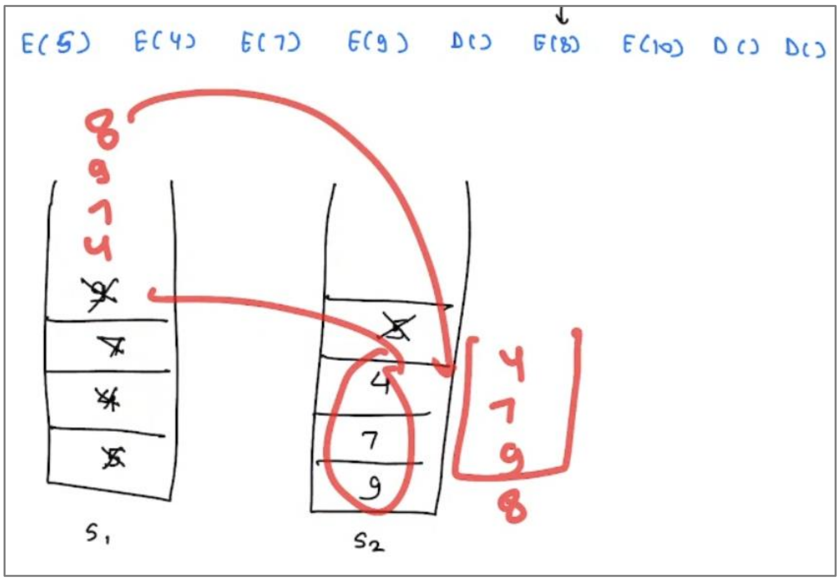
SO I have to take another stack where I'll delete from Stack 1 and add in Stack 2 till I get the first element.



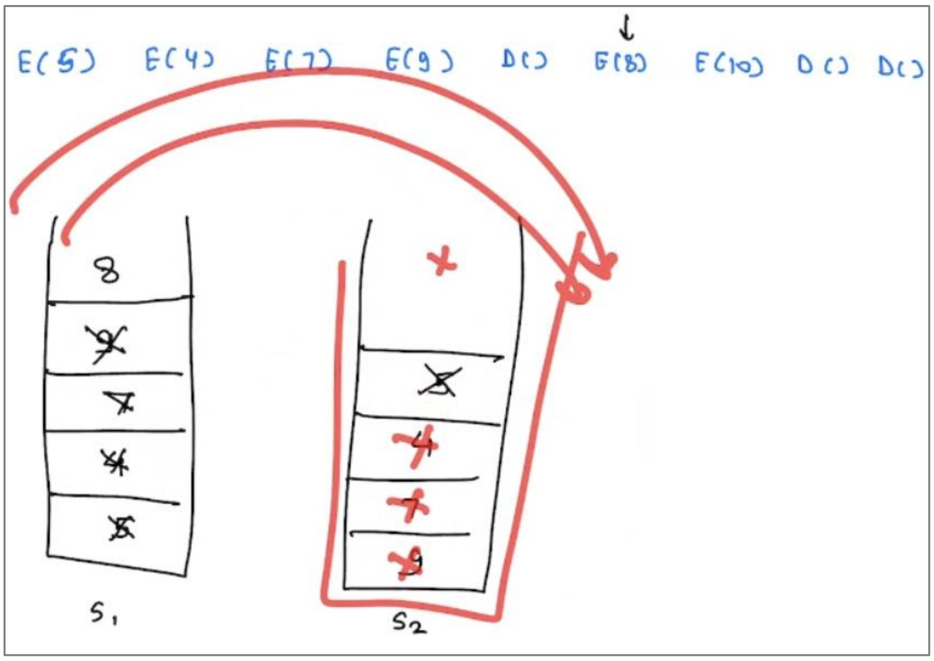
Now in stack 2, I have the right way which is to deleted. So I'll delete element 5.

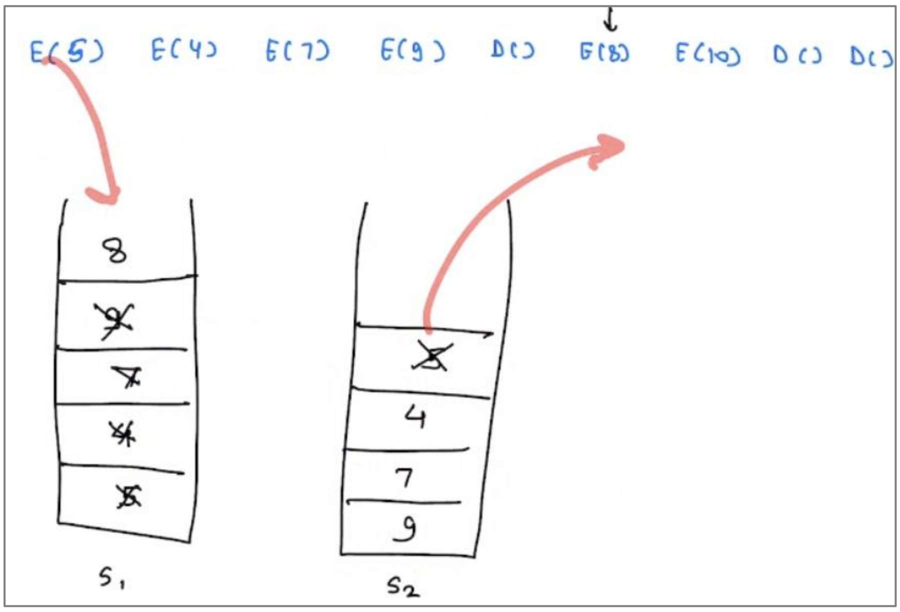


Now next is add element is 8.

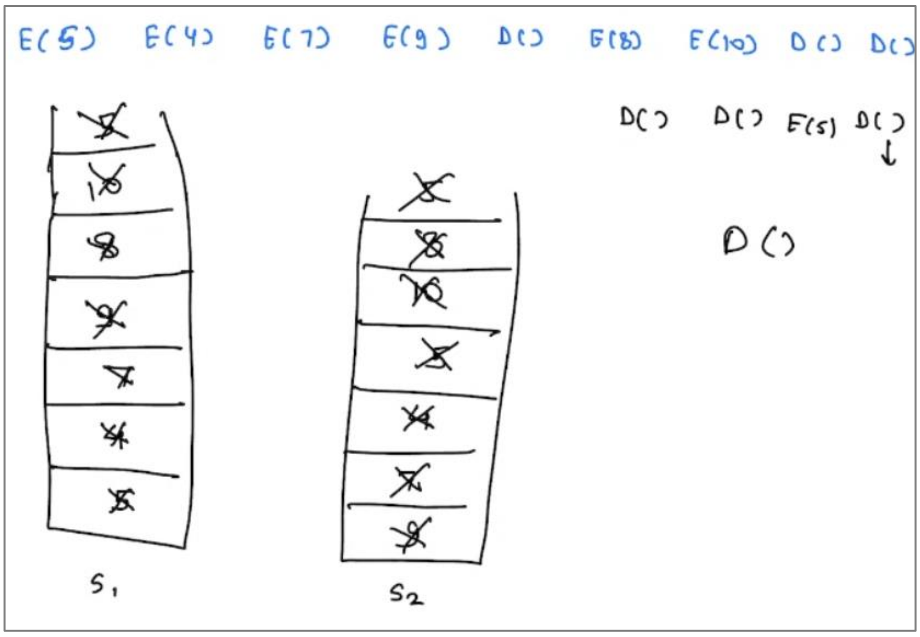


Add 8 in Stack 1 and keep all other element in Stack 2, and when we dequeue we will do in Stack 2 – since they are in right order in which we want to remove. When all element are empty in Stack2, take everything from Stack 1 and add in Stack2 and so on.

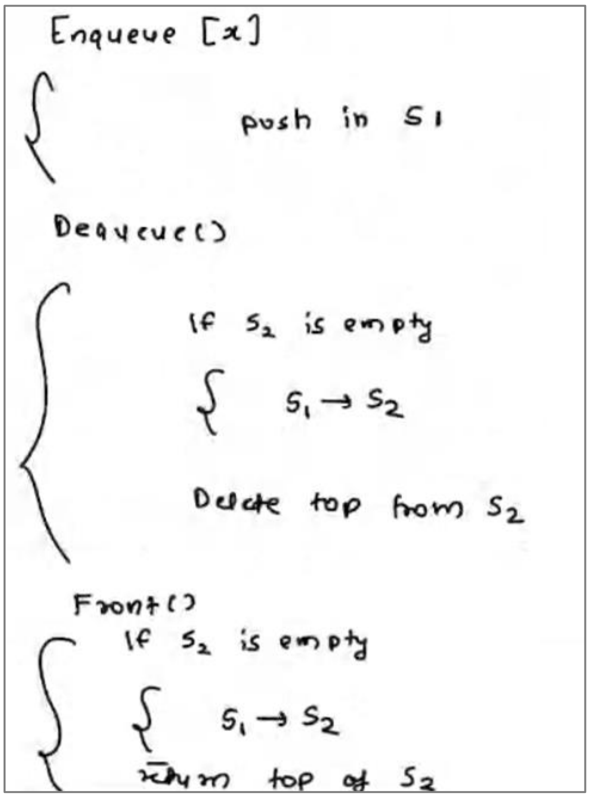




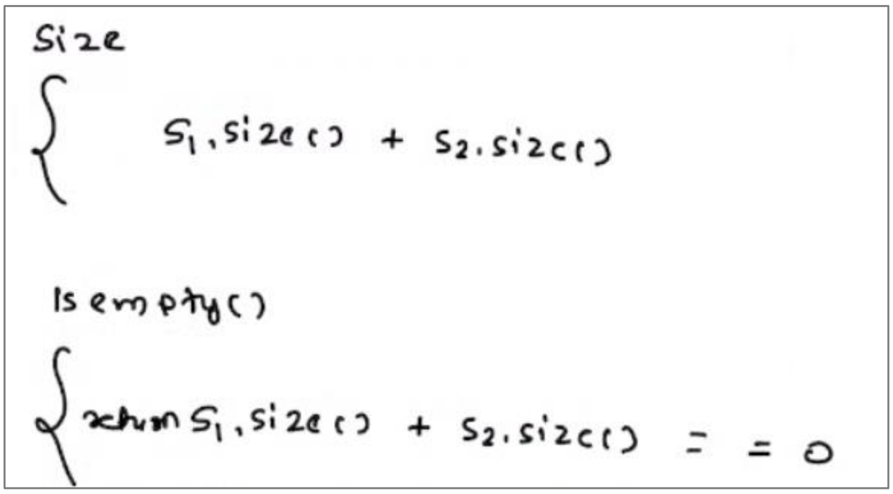
Add element in Stack 1 and delete element from Stack 2



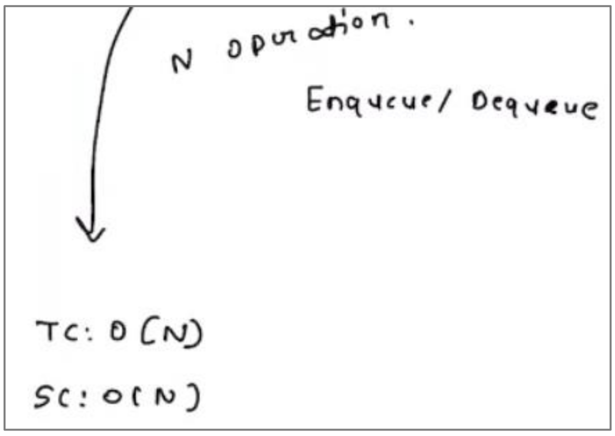
Psuedocode



```
Enqueue [x]
{
        push in S1

Dequeue()
{
        If S2 is empty
        {   S1 → S2

        Delete top from S2

Front()
{   If S2 is empty
    {   S1 → S2
    return top of S2
```



```
Size
{
    s1.size() + s2.size()

Isempty()
{return s1.size() + s2.size() == 0
```



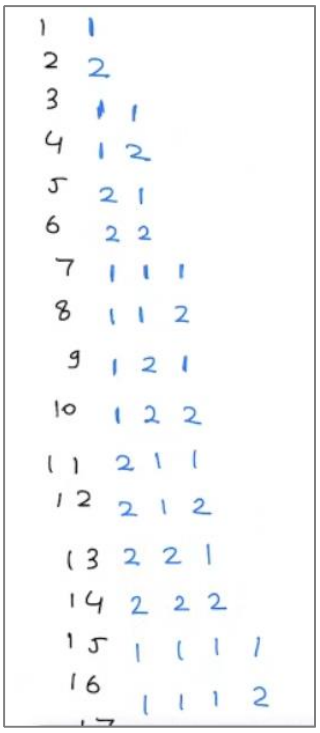
N operation.
Enqueue/ Dequeue

TC: O(N)

SC: O(N)

1.Problem Statement

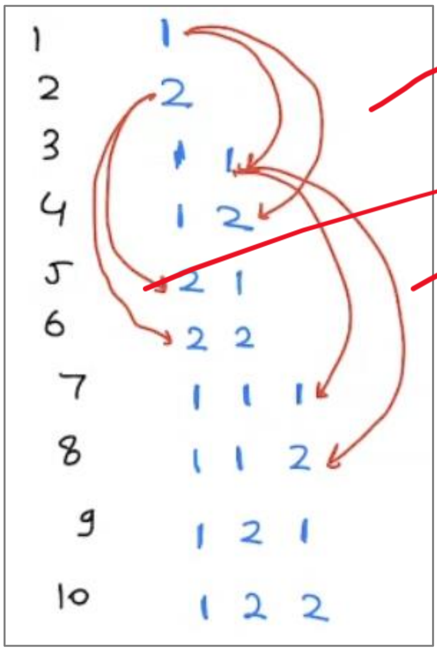
Find nth Perfect number (numbers made up of 1 or 2 only) Example: 1, 2, 11, 12, 21,22, 111, 112, 122, 211, 212, 221, 222, 1111...and so on.



1	1
2	2
3	1 1
4	1 2
5	2 1
6	2 2
7	1 1 1
8	1 1 2
9	1 2 1
10	1 2 2
11	2 1 1
12	2 1 2
13	2 2 1
14	2 2 2
15	1 1 1 1
16	1 1 1 2

If we say if you have to find 5th perfect #: 21 or 15th perfect #: 1111

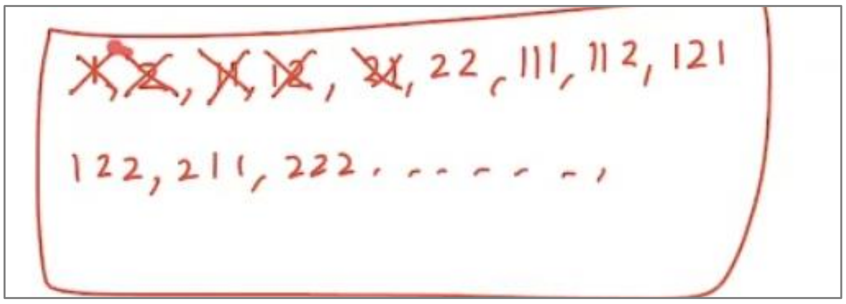
N = 5; ans = 21



Take 1 → add 1 after it and add 2 after it → next perfect numbers

Take 2 → add 1 after it and add 2 after it → next perfect numbers

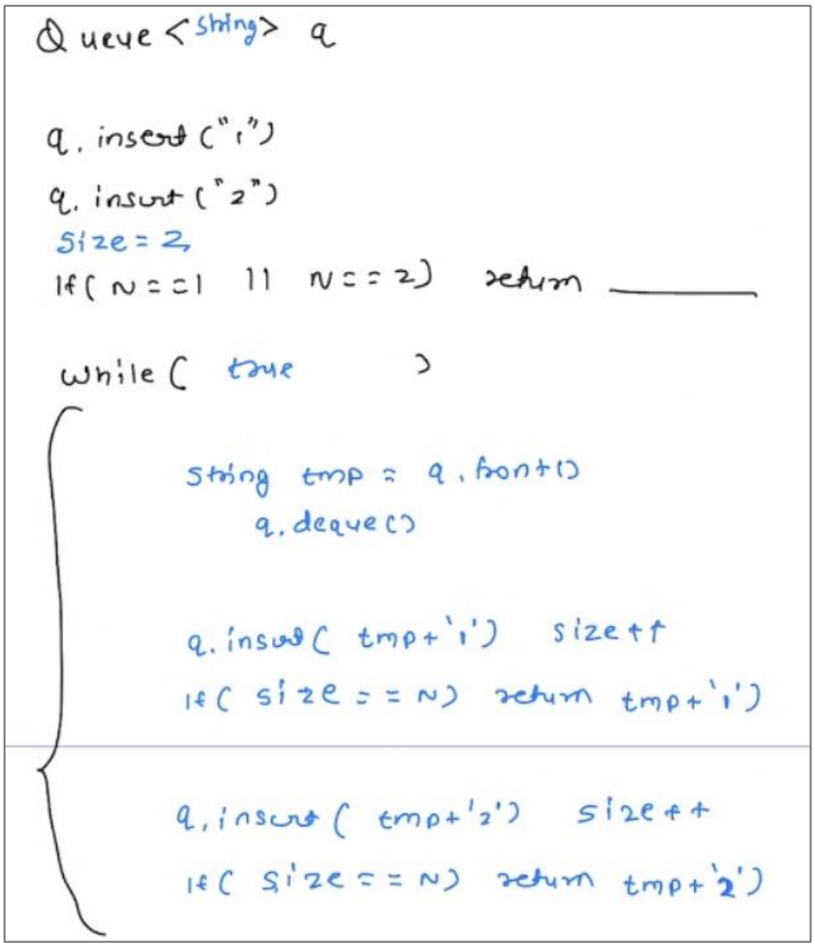
Take 11 → add 1 after it and add 2 after it



~~1~~ ~~2~~, ~~11~~, ~~12~~, ~~21~~, 22, 111, 112, 121

122, 211, 222, ... ,

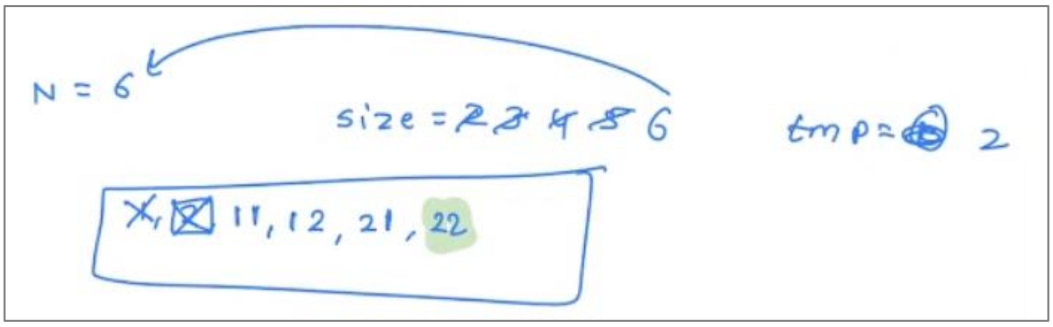
In a queue, Take 1, add 1 after it 11, add 2 after it 12 → Delete 1
Delete 2 → add 1 after it 21; add 2 after it 22
Insert at tail and delete at head



```
Queue<String> q

q.insert("1")
q.insert("2")
size = 2,
If( N==1  ||  N==2)   return ______

while(  true          )
{
        String tmp = q.front()
            q.deque()

        q.insert( tmp+'1')    size++
        if( size == N)  return  tmp+'1')

        q.insert( tmp+'2')    size++
        if( size == N)  return  tmp+'2')
}
```

Initially add 1 & 2.When size of q is < N→ keep adding elements
first save the first element
You have maintain the size since queue size keeps on changing
Dry run:

N = 6

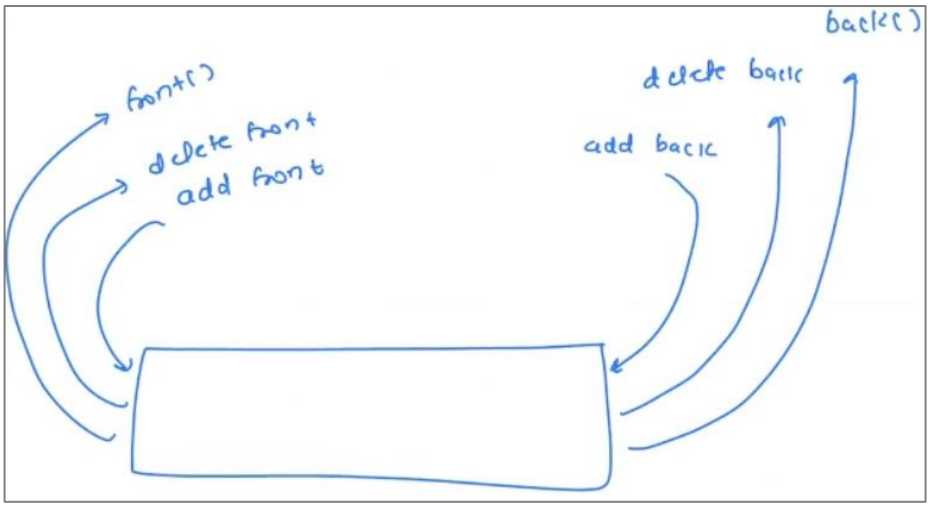


TC = O(N)

SC = O(N) –> you are adding two elements and deleting one element so when N == size, there are N/2 elements in the queue.

Double Ended Queue (Deque)

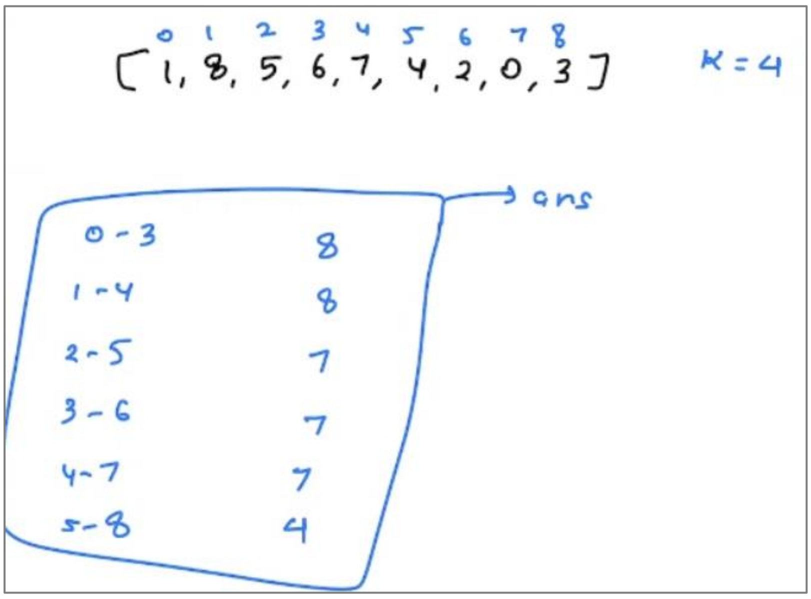
Add an element at the front & delete & checks an element at the front. Add an element at the back, delete an element at the back and checks an element at the back.



Deque may have SC as 3N or 4N so complex the Data structure, Space will also increase but we can use Deque in place of Satck and Queue.

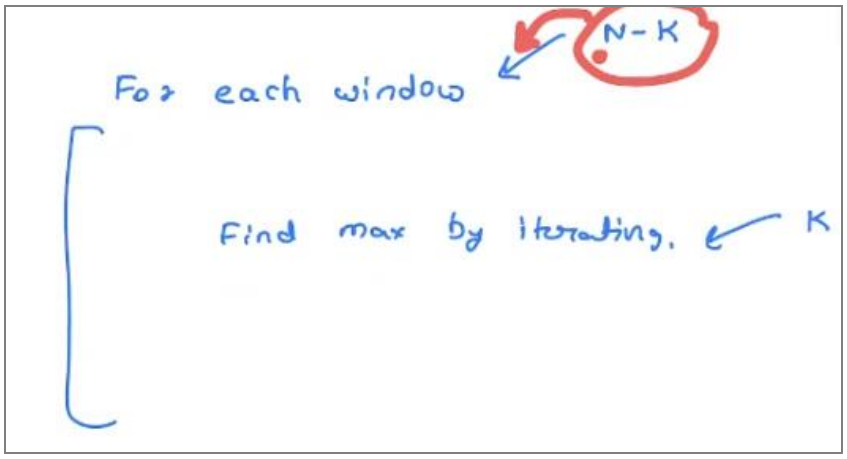
2.Problem Statement

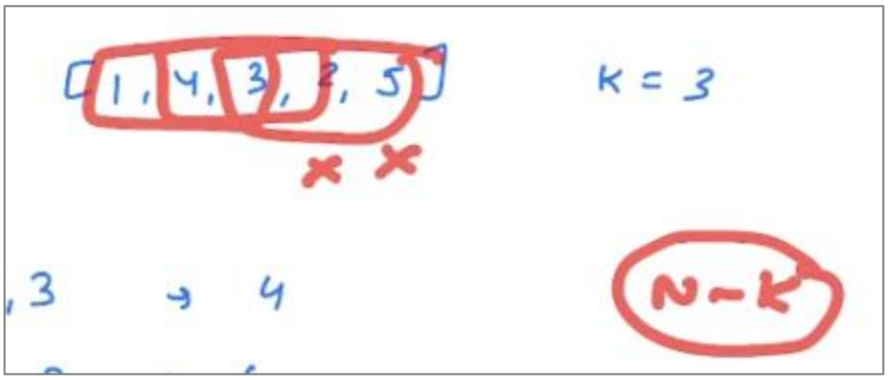
Given an array A and window size k. Find the max element of every window of size K



Brute Force

For every window, find max by iterating it



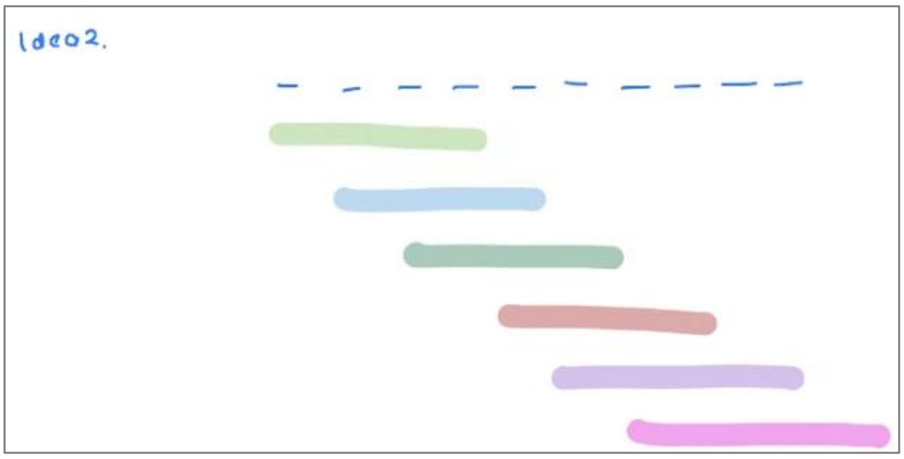


Last two elements will not be included

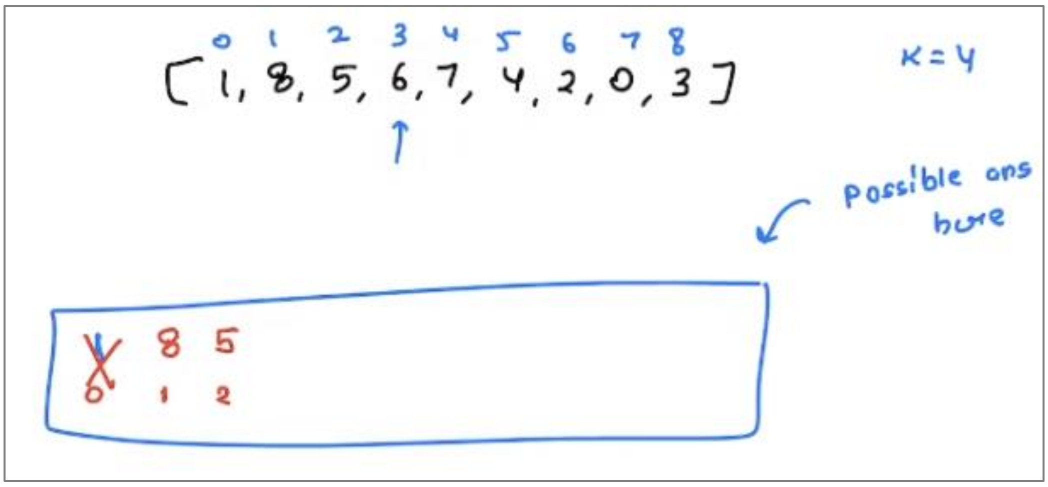
TC= O((N-K) * K) = O(K*N)

Idea -2

Sliding Window – is good for calculating sum but here we need max so this will not do

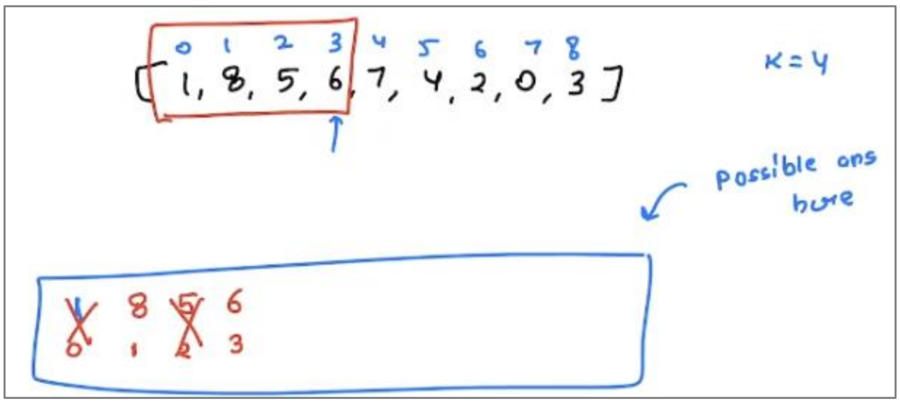


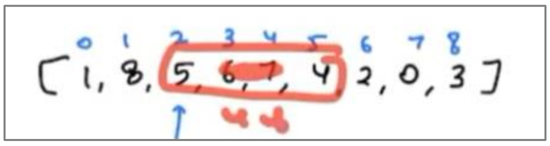
Dry Run using Queue



Add element in the rear and delete element in the front.

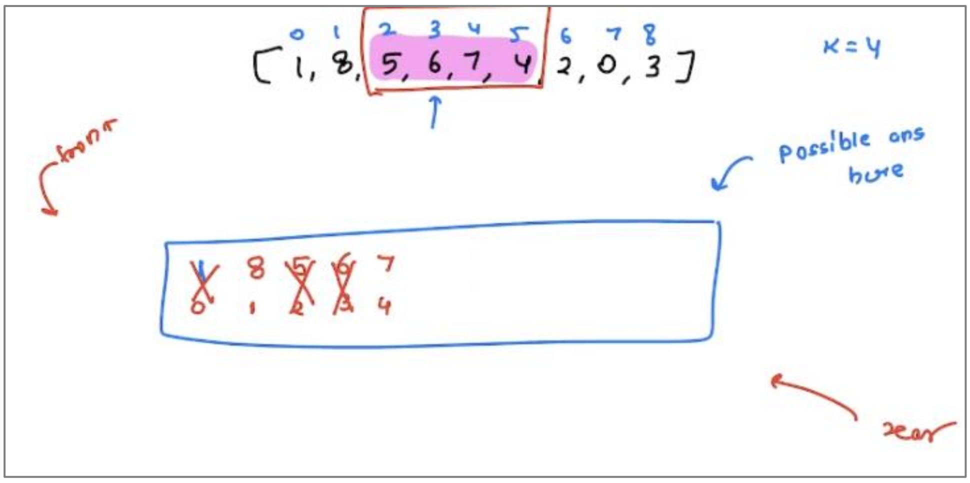
Add 1, If 8 is there, can there be any other element max in the subarray if 8 is present which is left of 8–> No. When you add an element , delete all element which is smaller than 8.





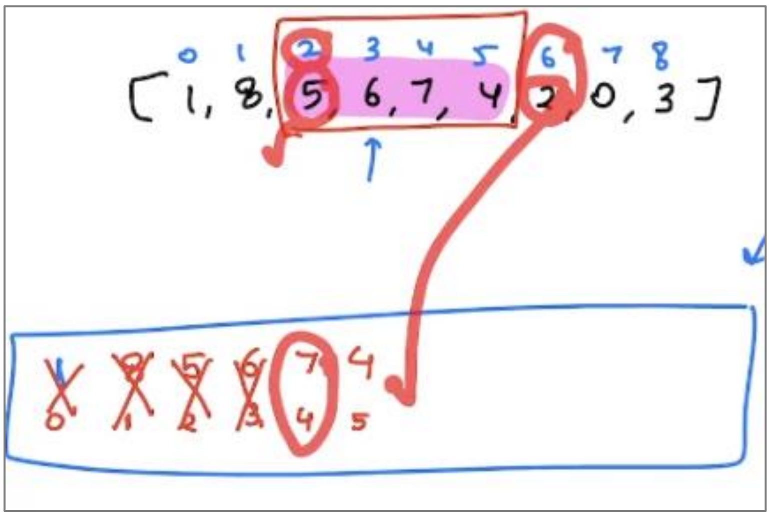
Add 5, since 5 could the max element of another window where 8 is not there and all elements are smaller than 5.

after we encounter, 6 → delete all element smaller than 6. → delete 5



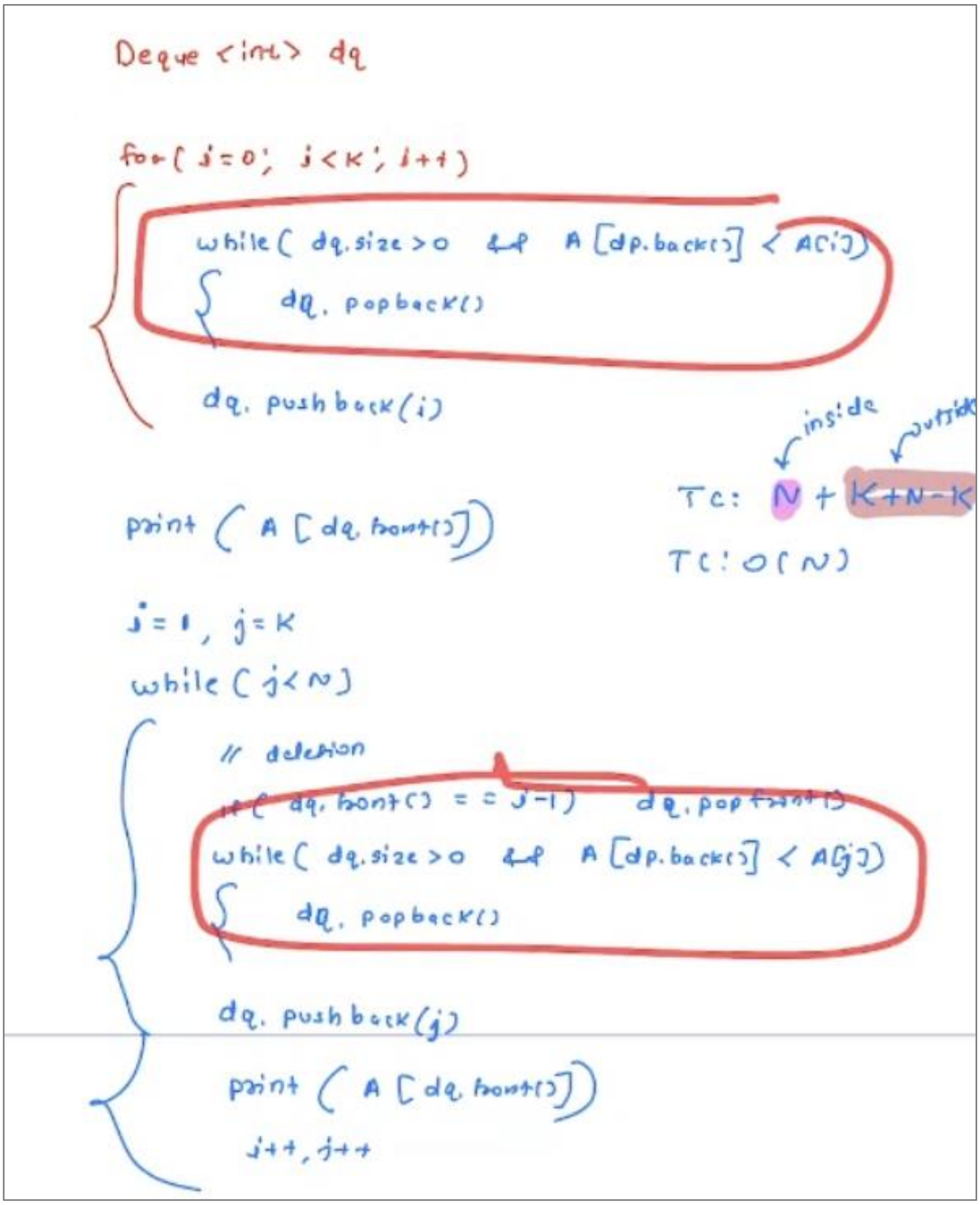
Element in the front is max. → 8. Where to look when looking for larger element? in the front or at the back. Since the format would descending, we have to look in the front

When you change the window, delete the first element if not already deleted → delete 5 and add 2 → delete elements smaller than 2.



Delete element smaller number than 2. Max element is already element in the front

Pseudocode



```
Deque <int> dq

for( i=0; i<K; i++)
{
    while( dq.size>0 && A[dp.back()] < A[i])
    {       dq.popback()

    dq.pushback(i)
}

print ( A [dq.front()])

i=1, j=K
while( j<N)
{
    // deletion
    if( dq.front() == i-1)      dq.popfront()
    while( dq.size>0 && A[dp.back()] < A[j])
    {       dq.popback()

    dq.pushback(j)
    print ( A [dq.front()])
    i++, j++
}
```

$TC: \overset{inside}{N} + \overset{outside}{K+N-K}$

$TC: O(N)$

In total , you can delete element which you added so you added N times only

SC = O(K) in worst case all would be there.

This can be used in Stock Market where you want to use max window of 1 min. You can see candlelight sticks on the laptop screen of Traders which represents max of stock price in that minute or min of stock price in that min

Problem Statement 1 - Parking Ice Cream Truck

Imagine you're an ice cream truck driver in a beachside town. The beach is divided into several sections, and each section has varying numbers of beachgoers wanting ice cream given by the array of integers **A**.

For simplicity, let's say the beach is divided into 8 sections. One day, you note down the number of potential customers in each section: [5, 12, 3, 4, 8, 10, 2, 7]. This means there are 5 people in the first section, 12 in the second, and so on.

You can only stop your truck in **B** consecutive sections at a time because of parking restrictions. To maximize sales, you want to park where the most customers are clustered together.

For all **B** consecutive sections, identify the busiest stretch to park your ice cream truck and serve the most customers. Return an array **C**, where **C[i]** is the busiest section in each of the **B** consecutive sections. Refer to the given example for clarity.
NOTE: If **B** > length of the array, return **1** element with the max of the array.
Problem Constraints
$1 <= |A|, B <= 10^6$
Input Format
The first argument given is the integer array A.
The second argument given is the integer B.
Output Format
Return an array C, where C[i] is the maximum value from A[i] to A[i+B-1].
Example Input
Input 1:
 A = [1, 3, -1, -3, 5, 3, 6, 7]
 B = 3
Input 2:
 A = [1, 2, 3, 4, 2, 7, 1, 3, 6]
 B = 6
Example Output
Output 1:
 [3, 3, 5, 5, 6, 7]
Output 2:
 [7, 7, 7, 7]
Example Explanation
Explanation 1:

```
 Window position     | Max
 --------------------|-------
```

[1 3 -1] -3 5 3 6 7 | 3
1 [3 -1 -3] 5 3 6 7 | 3
1 3 [-1 -3 5] 3 6 7 | 5
1 3 -1 [-3 5 3] 6 7 | 5
1 3 -1 -3 [5 3 6] 7 | 6
1 3 -1 -3 5 [3 6 7] | 7

Explanation 2:

```
Window position     | Max
--------------------|-------
[1 2 3 4 2 7] 1 3 6 | 7
1 [2 3 4 2 7 1] 3 6 | 7
1 2 [3 4 2 7 1 3] 6 | 7
1 2 3 [4 2 7 1 3 6] | 7
```

Actual Code

```
public class Solution {
  // DO NOT MODIFY THE LIST. IT IS READ ONLY
  public ArrayList<Integer> slidingMaximum(final List<Integer> A, int B) {
    Deque<Integer> dq = new ArrayDeque<>();
    ArrayList<Integer> C = new ArrayList<>();

    if (A.size() ==1){
      C.add(A.get(0));
      return C;
    }

    for(int i =0; i<B; i++){
      while(!dq.isEmpty() && A.get(dq.peekLast()) <= A.get(i)){
        dq.pollLast();
      }
      dq.addLast(i);
    }
    C.add(A.get(dq.peekFirst()));
    int i =1, j = B;
    while(j < A.size()){
      if(!dq.isEmpty() && dq.peekFirst() == i-1) dq.pollFirst();
      while(!dq.isEmpty() && A.get(dq.peekLast()) < A.get(j)){
        dq.pollLast();
      }
      dq.addLast(j);
      C.add(A.get(dq.peekFirst()));
      i++;
```

```
        j++;
      }
      return C;
    }
}
```

Problem Statement 2-Queue Using Stacks

Implement a **First In First Out (FIFO) queue** using stacks only.
The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).
Implement the UserQueue class:
 void push(int X) : Pushes element **X** to the back of the queue.
 int pop() : Removes the element from the front of the queue and returns it.
 int peek() : Returns the element at the front of the queue.
 boolean empty() : Returns true if the queue is empty, false otherwise.
NOTES:
 You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
 Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.
Problem Constraints
1 <= **X** <= 10^9
At most 1000 calls will be made to **push**, **pop**, **peek**, and **empty** function.
All the calls to pop and peek are valid. i.e. pop and peek are called only when the queue is non-empty.
Example Input
Input 1:
 1) UserQueue()
 2) push(20)
 3) empty()
 4) peek()
 5) pop()
 6) empty()
 7) push(30)
 8) peek()
 9) push(40)
 10) peek()
Input 2:
 1) UserQueue()
 2) push(10)
 3) push(20)
 4) push(30)
 5) pop()

6) pop()

Example Output

Output 1:

```
false
20
20
true
30
30
```

Output 2:

```
10
20
```

Example Explanation

Explanation 1:

Queue => 20

Queue => -

Queue => 30

Queue => 30, 40

Explanation 2:

Queue => 10

Queue => 10, 20

Queue => 10, 20, 30

Queue => 20, 30

Queue => 30

Actual Code

```java
public static class UserQueue {
  /** Initialize your data structure here. */
  private static Stack <Integer> st = new Stack<>();
  private static Stack <Integer> st2 = new Stack<>();
  UserQueue() {
    Queue <Integer> q = new LinkedList<>();
  }

  /** Push element X to the back of queue. */
  static void push(int X) {
    st.push(X);
  }

  /** Removes the element from in front of queue and returns that element. */
  static int pop() {
    transferIfNecessary();
    return st2.pop();
  }

  /** Get the front element of the queue. */
  static int peek() {
        transferIfNecessary();
        return st2.peek();
    }

    /** Returns whether the queue is empty. */
    static boolean empty() {
        return st.isEmpty() && st2.isEmpty();

    }

    private static void transferIfNecessary(){
        if(st2.isEmpty()){
            while(!st.isEmpty()){
                st2.push(st.pop());
            }
        }
    }
}
```

Problem Statement 3- N integers containing only 1, 2 & 3

Given an integer, **A**. Find and Return first positive **A** integers in ascending order containing only digits 1, 2, and 3.

NOTE: All the **A** integers will fit in 32-bit integers.

Problem Constraints

1 <= A <= 29500

Input Format

The only argument given is integer A.

Output Format

Return an integer array denoting the first positive A integers in ascending order containing only digits 1, 2 and 3.

Example Input

Input 1:

A = 3

Input 2:

A = 7

Example Output

Output 1:

[1, 2, 3]

Output 2:

[1, 2, 3, 11, 12, 13, 21]

Example Explanation

Explanation 1:

Output denotes the first 3 integers that contains only digits 1, 2 and 3.

Explanation 2:

Output denotes the first 7 integers that contains only digits 1, 2 and 3.

Actual Code

```
public class Solution {
   public ArrayList<Integer> solve(int A) {
      Queue<Integer> q = new LinkedList<>();
      ArrayList<Integer> B = new ArrayList<>();
      q.add(1);
      q.add(2);
      q.add(3);
      //int n = length(A);
      while(B.size() < A){
         int tmp = q.poll();
         B.add(tmp);

         q.add(tmp*10 +1);
         q.add(tmp*10 +2);
         q.add(tmp*10 +3);
      }
      return B;
   }
}
```

Problem Statement 4- Unique Letter

Imagine you're a teacher. You ask students to call out a letter one by one. After each letter, you jot down the very first letter that's only been called out once. If all letters have been repeated, you write "#". Here's a scenario:

A student says "a". It's the first letter. You write "a".
Next, a student says "b", "a" is still unique, so you add "a". Now it's "aa".
A student says "a" again. Now, "b" is the unique one. You add "b", making it "aab".
A student says "b". All letters so far are repeated. You add "#". It becomes "aab#".
A student says "c". "c" is unique. You add "c". The final is "aab#c".
Your task? Given the sequence the students call out **A**, determine the string on the board.

Problem Constraints

1 <= |A| <= 100000

Input Format

The only argument given is string A.

Output Format

Return a string after processing the stream of lowercase alphabets A.

Example Input

Input 1:
 A = "abadbc"
Input 2:
 A = "abcabc"

Example Output

Output 1:
"aabbdd"
Output 2:
"aaabc#"

Example Explanation

Explanation 1:
"a"   -   first non repeating character 'a'
"ab"   -   first non repeating character 'a'
"aba"   -   first non repeating character 'b'
"abad"   -   first non repeating character 'b'
"abadb" -   first non repeating character 'd'
"abadbc" -   first non repeating character 'd'
Explanation 2:
"a"   -   first non repeating character 'a'
"ab"   -   first non repeating character 'a'
"abc"   -   first non repeating character 'a'
"abca"   -   first non repeating character 'b'
"abcab" -   first non repeating character 'c'
"abcabc" -   no non repeating character so '#'

Actual Code

```
public class Solution {
  public String solve(String A) {
    Queue <Character> q = new LinkedList<>();
    HashMap<Character, Integer> map = new HashMap<>();
    StringBuilder res = new StringBuilder();

    //for (int i = 0; i < str.length(); i++) {
    //char ch = A.charAt(i);
    for(char ch: A.toCharArray()){
      map.put(ch, map.getOrDefault(ch, 0) +1);
      q.add(ch);
      while(!q.isEmpty() && map.get(q.peek()) > 1){
        q.poll();
      }
      if(!q.isEmpty()){
        res.append(q.peek());
      }else{
        res.append("#");
      }

    }
    return res.toString();
  }
}
```

Problem Statement 5 - Sum of min and max

Given an array **A** of both positive and negative integers.

Your task is to compute the sum of **minimum** and **maximum** elements of all sub-array of size **B**.

NOTE: Since the answer can be very large, you are required to return the sum modulo $10^9 + 7$.

Problem Constraints

1 <= size of array A <= 10^5

-10^9 <= A[i] <= 10^9

1 <= B <= size of array

Input Format

The first argument denotes the integer array A.

The second argument denotes the value B

Output Format

Return an integer that denotes the required value.

Example Input

Input 1:

A = [2, 5, -1, 7, -3, -1, -2]

B = 4

Input 2:

A = [2, -1, 3]

B = 2

Example Output

Output 1: 18

Output 2: 3

Example Explanation

Explanation 1:

Subarrays of size 4 are :

[2, 5, -1, 7],   min + max = -1 + 7 = 6

[5, -1, 7, -3], min + max = -3 + 7 = 4

[-1, 7, -3, -1], min + max = -3 + 7 = 4

[7, -3, -1, -2], min + max = -3 + 7 = 4

Sum of all min & max = 6 + 4 + 4 + 4 = 18

Explanation 2:

Subarrays of size 2 are :

[2, -1],   min + max = -1 + 2 = 1

[-1, 3],   min + max = -1 + 3 = 2

Sum of all min & max = 1 + 2 = 3

Actual Code

```
public class Solution {
  public int solve(ArrayList<Integer> A, int B) {
    Deque <Integer> dq = new ArrayDeque<>();
    long max =0, min =0;
    long sum =0;
    long mod = 1000000007;
    for(int i =0; i<B; i++){
      while(!dq.isEmpty() && A.get(dq.peekLast()) <= A.get(i)){ //calculate max
        dq.pollLast();
      }
      dq.addLast(i);
    }
    max += A.get(dq.peekFirst());

    for(int i = B; i< A.size(); i++){
      if(!dq.isEmpty() && dq.peekFirst() == i-B) dq.pollFirst();
      while(!dq.isEmpty() && A.get(dq.peekLast()) <= A.get(i)){
        dq.pollLast();
      }
      dq.addLast(i);
      max += A.get(dq.peekFirst());
    }
    dq.clear();
    for(int k =0; k<B; k++){
      while(!dq.isEmpty() && A.get(dq.peekLast()) >= A.get(k)){ //calculate min
        dq.pollLast();
      }
      dq.addLast(k);
    }
    min += A.get(dq.peekFirst());

    for(int k = B; k <A.size(); k++){
      if(!dq.isEmpty() && dq.peekFirst() == k-B) dq.pollFirst();
      while(!dq.isEmpty() && A.get(dq.peekLast()) >= A.get(k)){
        dq.pollLast();
      }
      dq.addLast(k);
      min += A.get(dq.peekFirst());
    }
    sum = ((max % mod) + (min % mod)+mod)%mod;
    return (int) sum;
  }
}
```