

Stacks & Queues

- Karun Karthik

Contents

0. Introduction
1. Implementation of stack using Linkedlist
2. Implementation of queue using Linkedlist
3. Implementation of stack using queue
4. Implementation of queue using stack
5. Valid Parenthesis
6. Asteroid Collision
7. Next Greater Element
8. Next Smaller Element
9. Stock Span Problem
10. Celebrity Problem
11. Largest Rectangle in Histogram
12. Sliding Window Maximum

Stack → Linear data structure

- follows LIFO, last in first out.
- Operations → push : insert into top of stack
pop : delete from top of stack.

Applications →

- by compilers to check for parenthesis
- to evaluate postfix expression
- to convert infix to postfix/ prefix form.
- to store values during recursion & context during function call.
- to implement DFS of graph

Queue → Linear data structure

- follows FIFO, first in first out.
- Operations → enqueue : insert element at end of queue
dequeue : delete element at start of queue

Applications →

- schedule jobs by CPU.
- to carry out FIFO basis like printing jobs.
- to implement BFS of graph

Types →

- Queue
- Circular Queue
- Doubly ended Queue
- Priority Queue.

① Implement a stack using Linkedlist →

code →

```
● ● ●

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct Node{
5     int data;
6     Node* next;
7 };
8
9 Node* top;
10
11 void push(int data){
12     Node* temp = new Node();
13     if (!temp){
14         cout << "\nStack Overflow";
15         exit(1);
16     }
17     // add at the top and change top as new node
18     temp->data = data;
19     temp->next = top;
20     top = temp;
21 }
22
23 int isEmpty(){
24     // if top is null then empty
25     return top == NULL;
26 }
27
28 int peek(){
29     // if stack is not empty then return top node's data
30     if (!isEmpty())
31         return top->data;
32     else
33         exit(1);
34 }
35
36 void pop(){
37     Node* temp;
38     if (top == NULL){
39         cout << "\nStack Underflow" << endl;
40         exit(1);
41     } else {
42         temp = top;
43         top = top->next;
44         free(temp);
45     }
46 }
47
```

② Implement a Queue using Linkedlist →

Code →

```
● ● ●  
1 class Node {  
2     int data;  
3     Node* next;  
4     Node(int d){  
5         data = d;  
6         next = NULL;  
7     }  
8 };  
9  
10 class Queue {  
11     Node *front, *rear;  
12  
13     Queue(){  
14         front = rear = NULL;  
15     }  
16  
17     void enqueue(int x)  
18     {  
19         Node* temp = new Node(x);  
20         // if empty then node is both front and rear  
21         if (rear == NULL) {  
22             front = rear = temp;  
23             return;  
24         }  
25         // else add at end  
26         rear->next = temp;  
27         rear = temp;  
28     }  
29  
30     void dequeue()  
31     {  
32         // if empty then return NULL  
33         if (front == NULL)  
34             return;  
35         // store front node  
36         Node* temp = front;  
37         front = front->next;  
38  
39         // if front is NULL => no Nodes, change rear to NULL  
40         if (front == NULL)  
41             rear = NULL;  
42         // free node  
43         delete (temp);  
44     }  
45 };
```

③ Implement a Stack using Queue →

If push, push into queue from rear end & pop & push all elements
if pop, pop from queue from front end.

Code →

```
● ● ●  
1 class Stack {  
2     queue <int> q;  
3  
4     public:  
5  
6         // push operation  
7         void Push(int x) {  
8             int n = q.size();  
9             q.push(x);  
10            for (int i = 0; i < n; i++)  
11            {  
12                int value = q.front();  
13                q.pop();  
14                q.push(value);  
15            }  
16        }  
17  
18        // pop operation  
19        int Pop() {  
20            int value = q.front();  
21            q.pop();  
22            return value;  
23        }  
24  
25        // accessing top value  
26        int Top() {  
27            return q.front();  
28        }  
29  
30        // finding size of stack  
31        int Size() {  
32            return q.size();  
33        }  
34    };  
35
```

④ Implement a Queue using Stack →

→ use 2 stacks.

→ while pop(), shift all elements in 1 stack to another.
& return top value.

Code →

```
● ● ●  
1 class Queue {  
2     public:  
3         stack <int> in;  
4         stack <int> out;  
5  
6         // push operation  
7         void Push(int x) {  
8             in.push(x);  
9         }  
10  
11         // pop operation  
12         int Pop() {  
13             // shift in to out  
14             if (out.empty()){  
15                 while (in.size()){  
16                     out.push(in.top());  
17                     in.pop();  
18                 }  
19             }  
20             int x = out.top();  
21             out.pop();  
22             return x;  
23         }  
24  
25         // peek operation  
26         int Top() {  
27             if (out.empty()){  
28                 while (in.size()){  
29                     out.push(in.top());  
30                     in.pop();  
31                 }  
32             }  
33             return out.top();  
34         }  
35  
36         int Size() {  
37             return in.size()+out.size();  
38         }  
39     };
```

⑤ Valid parenthesis

$s = \{\}$ → T

$s = \{\}[]\}$ → T

$s = ()\{\}$ → T

$s =)[]$ → F

Eg $s = \{\[()\{}\)(\])$ → True.

→ if match found then pop, else push.

stack : [

] $s = \{\[(){}\)(\])$

stack : [\

] $s = \{\[\){}\)(\])$

stack : [\[

] $s = \{\[\(\){}\)(\])$

stack : [\[\]

] $s = \{\[\(\)\{}\)(\])$

stack : [\(\)

] $s = \{\[\(\)\{}\)(\])$

stack : [\(\)

] $s = \{\[\(\)\{}\)(\])$

stack : [\(\)

] $s = \{\[\(\)\{}\)(\])$

stack : [()

] $s = \{\[\(\)\{}\)(\])$

Code →

```
1  class Solution {
2 public:
3     bool isValid(string s) {
4         stack<char> st;
5         for(auto i : s)
6         {
7             if (st.empty() || i == '(' || i == '{' || i == '[')
8             {
9                 st.push(i);
10            }
11            else
12            {
13                if ((i == ')' && st.top() != '(') ||
14                    (i == ']' && st.top() != '[') ||
15                    (i == '}' && st.top() != '{')){
16                    return false;
17                }
18                st.pop();
19            }
20        }
21        return st.empty();
22    }
23};
```

$Tc \rightarrow O(n)$

$Sc \rightarrow O(1)$

⑥ Asteroid Collision

only consider magnitude

+ve sign \Rightarrow right direction

If $x \& y$ collide then $\min(x, y)$ will be removed

-ve sign \Rightarrow left direction

If $x = y$ then both will be removed.

Eg $[5, 10, -5]$

$5, 10$ will not collide

$10, -5$ will collide & -5 will be removed

$$\text{result} = [5, 10]$$

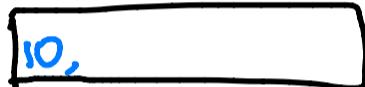
Eg $[10, 6, -8, -8, 8, 9]$

stack



$[10, 6, -8, -8, 8, 9]$

stack



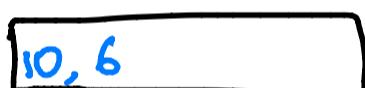
$[10, 6, -8, -8, 8, 9]$

stack



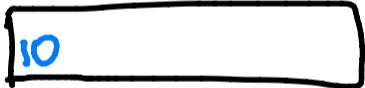
$[10, 6, -8, -8, 8, 9]$ as 6 is +ve push

stack



$[10, 6, -8, -8, 8, 9]$ as 6 & 8 will collide
(opp directions), 6 will be removed

stack



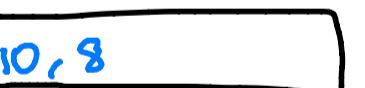
$[10, 6, -8, -8, 8, 9]$ as 10 & 8 will collide
(opp directions), 8 will be removed

stack



$[10, 6, -8, -8, 8, 9]$ as 10 & 8 will collide
(opp directions), 8 will be removed

stack



$[10, 6, -8, -8, 8, 9]$ as 8 is +ve push

stack



$[10, 6, -8, -8, 8, 9]$ as 9 is +ve push



$$\text{result} = [10, 8, 9]$$

TC $\rightarrow O(2n) \approx O(n)$ SC $\rightarrow O(n)$

worst case

Code →



```
1 class Solution {
2 public:
3     vector<int> asteroidCollision(vector<int>& asteroids) {
4
5         vector<int> res;
6
7         for(int i=0; i< asteroids.size(); i++){
8
9             if(res.empty() || asteroids[i]>0)
10                 res.push_back(asteroids[i]);
11             else {
12
13                 while(!res.empty() && res.back()>0 && res.back()<abs(asteroids[i])) {
14                     res.pop_back();
15                 }
16
17                 if(!res.empty() && res.back()+asteroids[i]==0)
18                     res.pop_back();
19                 else if(res.empty() || res.back()<0)
20                     res.push_back(asteroids[i]);
21                 }
22             }
23         return res;
24     }
25 }
```

⑦ Next greater element → [2, 4, 1, 3, 1, 6]

Eg [4, 5, 2, 25]

4 → 5 2 → 25
5 → 25 25 → -1

2 → 4 3 → 6
4 → 6 1 → 6
1 → 3 6 → -1

- iterate from last & compare its value with top of stack
- if stack is greater then its the next greater element
- else keep popping till the next greater element is found.

Eg [11, 13, 3, 10, 7, 21, 26]



Stack = []	[11, 13, 3, 10, 7, 21, 26]	
Stack = [26]	[11, 13, 3, 10, 7, 21, 26]	26 → -1
Stack = [26, 21]	[11, 13, 3, 10, 7, 21, 26]	21 → 26
Stack = [26, 21, 7]	[11, 13, 3, 10, 7, 21, 26]	7 → 21
Stack = [26, 21, 7, 10]	[11, 13, 3, 10, 7, 21, 26]	pop 7, push 10 10 → 21
Stack = [26, 21, 10]	[11, 13, 3, 10, 7, 21, 26]	3 → 10
Stack = [26, 21, 10, 3, 13]	[11, 13, 3, 10, 7, 21, 26]	pop 3, 10 push 13 13 → 21
Stack = [26, 21, 13]	[11, 13, 3, 10, 7, 21, 26]	11 → 13

Ans = [13, 21, 10, 21, 21, 26, -1]

Code →

```
1 class Solution
2 {
3     public:
4     //Function to find the next greater element for each element of the array.
5     vector<long long> nextLargerElement(vector<long long> arr, int n){
6
7         stack<long long> st;
8         vector<long long> res(n);
9
10        for(int i=n-1; i>=0 ; i--){
11            long long currVal = arr[i];
12
13            while(!st.empty() && st.top()<=currVal)
14                st.pop();
15
16            res[i] = st.empty()?-1:st.top();
17            st.push(currVal);
18        }
19        return res;
20    }
21 };
22
```

$Tc \rightarrow O(n)$

$Sc \rightarrow O(n)$

8

Next Smaller element →

→ entire approach is similar to next greater element except for comparison.

Code →

$Tc \rightarrow O(n)$

$Sc \rightarrow O(n)$



```

1  vector<int> nextSmallerElement(vector<int> &arr, int n)
2  {
3      stack<int> st;
4      vector<int> res(n);
5      for(int i=n-1; i>=0 ; i--){
6
7          long long currVal = arr[i];
8
9          while(!st.empty() && st.top()>=currVal)
10             st.pop();
11
12          res[i] = st.empty()?-1:st.top();
13          st.push(currVal);
14      }
15      return res;
16  }
```

⑨ Stock Span Problem → Given price quotes of stock for n days.
 we need to find span of stock on any particular day.

max no. of consecutive days for which price \leq curr day's price

Eg $[100, 80, 60, 70, 60, 75, 85]$

stack = [stores indexes]

span =

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

if currentElement > stack.top
 pop stack

else:
 $\text{span} = \text{currentIndex} - \text{stack.top}$

→ push index into stack after processing →

0 1 2 3 4 5 6

$[100, 80, 60, 70, 60, 75, 85]$ span of 1st element = 1

stack

[0]

span

0	1	0	0	0	0	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $80 > 100 \Rightarrow \text{false}$
 $\therefore \text{span} = 1 - 0 = 1$

[0, 1]

1	0	0	0	0	0	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $60 > 100 \Rightarrow \text{false}$
 $\therefore \text{span} = 2 - 1 = 1$

[0, 1, 2]

1	1	0	0	0	0	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $70 > 60 \Rightarrow \text{true} \therefore \text{pop}$
 $70 > 80 \Rightarrow \text{false}$
 $\therefore \text{span} = 3 - 1 = 2$

[0, 1, 3]

1	1	1	2	0	0	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $60 > 70 \Rightarrow \text{false}$
 $\therefore \text{span} = 4 - 3 = 1$

[0, 1, 3, 4]

1	1	1	2	1	0	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $75 > 60 \Rightarrow \text{true} \therefore \text{pop}$
 $75 > 70 \Rightarrow \text{true} \therefore \text{pop}$
 $75 > 80 \Rightarrow \text{false}$
 $\text{span} = 5 - 1 = 4$

[0, 1, 5]

1	1	1	2	1	3	0
---	---	---	---	---	---	---

$[100, 80, 60, 70, 60, 75, 85]$ $85 > 75 \Rightarrow \text{true} \therefore \text{pop}$
 $85 > 80 \Rightarrow \text{true} \therefore \text{pop}$
 $85 > 100 \Rightarrow \text{false}$
 $\text{span} = 6 - 0 = 6$

[0, 6]

1	1	1	2	1	4	6
---	---	---	---	---	---	---

span =

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

$Tc \rightarrow O(n)$
 $Sc \rightarrow O(n)$

Code →



```
1 class Solution
2 {
3     public:
4         //Function to calculate the span of stocks price for all n days.
5         vector <int> calculateSpan(int price[], int n)
6     {
7         vector<int> span(n);
8         stack<int> st;
9
10        st.push(0);
11        span[0] = 1;
12
13        for(int i=1; i<n; i++){
14
15            int currPrice = price[i];
16
17            while(!st.empty() && currPrice >= price[st.top()])
18                st.pop();
19
20            if(st.empty()){
21                span[i] = i+1;
22            } else {
23                span[i] = i-st.top();
24            }
25
26            st.push(i);
27        }
28        return span;
29    }
30 }
31
```

⑩ Celebrity Problem

A Celebrity is a person, who is known to everyone & knows none.

Given a square matrix M & if i^{th} person knows j^{th} person
then $M[i][j] = 1$, else 0.

Eg →

$$M = \begin{bmatrix} 0 & 1 & 2 \\ 0 & [0, 1, 0], \\ 1 & [0, 0, 0], \\ 2 & [0, 1, 0] \end{bmatrix}, \quad n = 3.$$

$$\rightarrow [\stackrel{\text{stack}}{[]}] \Rightarrow [\stackrel{\text{stack}}{[0, 1, 2]}]$$

- ① create stack & push values from 0 to $n-1$.
- ② do the following till stack more than has 1 value.
 - pop 1st element & set it to A
 - pop again & set it to B
 - if A knows B then push B

$$\Rightarrow [\stackrel{\text{stack}}{[0, 1, 2]}] \quad \begin{array}{l} A = 2 \\ B = 1 \end{array} \quad \& \quad \begin{array}{l} M[2][1] = 1 \\ \text{true} \end{array} \quad \therefore \text{push } 1 \Rightarrow [\stackrel{\text{stack}}{[0, 1]}]$$

$$\Rightarrow [\stackrel{\text{stack}}{[0, 1]}] \quad \begin{array}{l} A = 1 \\ B = 0 \end{array} \quad \& \quad \begin{array}{l} M[1][0] = 1 \\ \text{false} \end{array} \quad \therefore \text{push } 1 \Rightarrow [\stackrel{\text{stack}}{[1]}]$$

\therefore as stack has only 1 element, STOP.

Now pop the stack & consider it as celebrity & check for

- anyone doesn't know celeb ($\neg M[i][\text{celeb}]$)
- if celeb knows anyone ($M[\text{celeb}][i]$)

} return -1.

\therefore from $i=0$ to 2 & celeb = 1

$$i=0 \quad (\neg M[0][1] \text{ or } M[1][0]) = 0 \quad \left. \right\}$$

$i=1$ skip as celeb is 1

$$i=2 \quad (\neg M[2][1] \text{ or } M[1][2]) = 0 \quad \left. \right\}$$

all are failed i.e. no violation of conditions.

\therefore return celeb i.e. 1

Code →

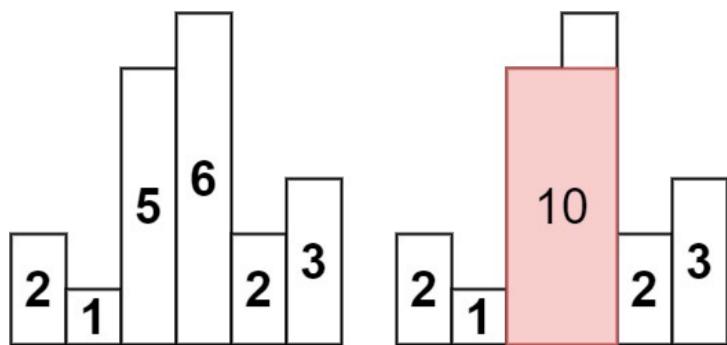
$TC = O(n)$

$SC = O(n)$

```
● ● ●

1 class Solution
2 {
3     public:
4     //Function to find if there is a celebrity in the party or not.
5     int celebrity(vector<vector<int> & M, int n) {
6
7         stack<int> s;
8
9         for(int i=0;i<n;i++)    s.push(i);
10
11        // check and if is a celebrity then push into stack
12        while(s.size()>1)
13        {
14            int a=s.top();
15            s.pop();
16            int b=s.top();
17            s.pop();
18
19            if(M[a][b]==1)
20                s.push(b);
21            else
22                s.push(a);
23        }
24
25        int celeb = s.top();
26
27        for (int i = 0; i < n; i++){
28            // if i person doesn't know celeb or celeb knows anyone else
29            // then return -1
30            if ( (i!=celeb) && (!M[i][celeb]) || M[celeb][i] )
31                return -1;
32        }
33
34        return celeb;
35    }
36};
```

11 Largest Rectangle in Histogram →



→ given an array of heights,
return area of largest rectangle

Ans = 10.

0 1 2 3 4 5

Stack .

arr = [2, 1, 5, 6, 2, 3]

[]

area = 0 maxArea = 0

i = 0 [2, 1, 5, 6, 2, 3]

[0]

area = 0 maxArea = 0

→ i = 1 [2, 1, 5, 6, 2, 3]

[0]

area = 0 maxArea = 0

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop() ↑
as stack is empty now, width = i & push(i)

∴ ht = 2 & width = 1 ∴ area = 2 & maxArea = ⚡ 2.

→ i = 2 [2, 1, 5, 6, 2, 3] [1] area = 0 maxArea = 2

now arr[st.top()] > currElement ⇒ false ∴ push(i) 2

→ i = 3 [2, 1, 5, 6, 2, 3] [1, 2] area = 0 maxArea = 2

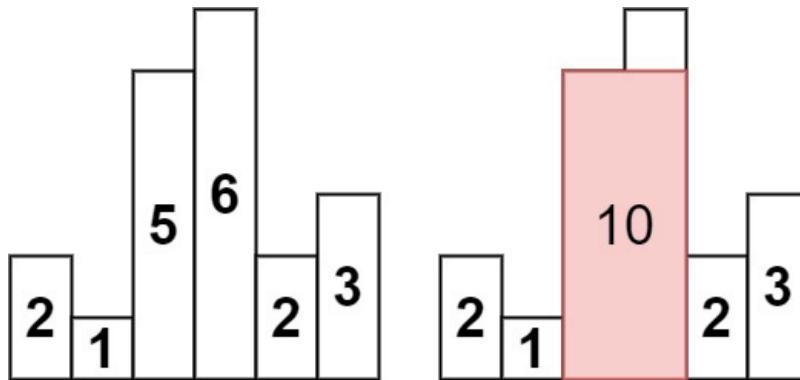
now arr[st.top()] > currElement ⇒ false ∴ push(i) 3

→ i = 4 [2, 1, 5, 6, 2, 3] [1, 2, 3] area = 0 maxArea = 2

now arr[st.top()] > currElement ⇒ ht = arr[st.top()] & st.pop() ↑ [1, 2, 3]

width = i - st.top() - 1 = 1 ∴ area = 6 * 1 = 6 maxArea = ⚡ 6.

& push(i) 4



\Rightarrow Last iteration to pop stack $\Rightarrow i=6$

After Iteration i=6:

- Stack: $[2, 1, 5, 6, \text{[2, 3]}]$ (The bar at index 6 is highlighted in red.)
- Current Element: 3
- Area: $\text{area} = 3$
- Max Area: $\text{maxArea} = 10$

Calculation for i=6:

$\text{ht} = \text{arr}[\text{st.top}()] \& \text{pop}()$ as stack is not empty

Width: $\text{width} = i - \text{st.top}() - 1 = 1$ $\therefore \text{area} = 3 \times 1 = 3$ $\text{maxArea} = 10$

$\& \text{push}(i)$

After Iteration i=7:

- Stack: $[2, 1, 5, 6, \text{[2, 3]}]$ (The bar at index 7 is highlighted in green.)
- Current Element: 1
- Area: $\text{area} = 3$
- Max Area: $\text{maxArea} = 10$

Calculation for i=7:

$\text{ht} = \text{arr}[\text{st.top}()] \& \text{pop}()$ as stack is not empty

Width: $\text{width} = i - \text{st.top}() - 1 = 1$ $\therefore \text{area} = 2 \times 1 = 2$ $\text{maxArea} = 10$

After Iteration i=8:

- Stack: $[2, 1, 5, 6, \text{[2, 3]}]$ (The bar at index 8 is highlighted in yellow.)
- Current Element: 0
- Area: $\text{area} = 3$
- Max Area: $\text{maxArea} = 10$

Calculation for i=8:

$\text{ht} = \text{arr}[\text{st.top}()] \& \text{pop}()$ as stack is not empty

Width: $\text{width} = i - \text{st.top}() - 1 = 1$ $\therefore \text{area} = 2 \times 1 = 2$ $\text{maxArea} = 10$

0 1 2 3 4 5
 → [2, 1, 5, 6, 2, 3] [1,] area = 6 maxArea = 0
 ht = arr[st.top()] & pop() & as stack is empty
 width = ⁶i = 6 ⇒ ∴ area = 1 * 6 = 6 maxArea = 10
 ∵ stack is empty return maxArea = 10.

Code → $Tc \rightarrow O(n)$
 $Sc \rightarrow O(n)$

```

1 class Solution {
2 public:
3     int largestRectangleArea(vector<int>& heights) {
4         stack < int > st;
5         int maxArea = 0;
6         int n = heights.size();
7
8         for (int i = 0; i <= n; i++) {
9
10            while (!st.empty() && (i == n || heights[st.top()] >= heights[i])) {
11
12                int height = heights[st.top()];
13                st.pop();
14                int width;
15                if (st.empty()){
16                    width = i;
17                } else {
18                    width = i - st.top() - 1;
19                }
20
21                int area = width*height;
22                maxArea = max(maxArea, area);
23            }
24            st.push(i);
25        }
26        return maxArea;
27    }
28 };
29
30

```

⑫ Sliding Window Maximum →

- process first ' k ' elements before pushing into result arr.
- if $dq.front() == i - k$ then pop-front (out of boundary case)
- if $nums[dq.back()] < nums[i]$ then pop-back
(meaningless to store smaller elements in window)
- if $i \geq k - 1$ then push $nums[dq.front()]$

Eg $nums = [1, 3, -1, -3, 5, 3, 6, 7] \quad k=3 \quad res = [3, 3, 5, 5, 6, 7]$

\Rightarrow	nums	deque	res
	$[1, 3, -1, -3, 5, 3, 6, 7]$ 0 1 2 3 4 5 6 7	_____	[]
$i=0$	$\overset{0}{[1}, 3, -1, -3, 5, 3, 6, 7]$	<u>0</u>	[]
$i=1$	$\overset{0}{[1}, \overset{1}{3}, -1, -3, 5, 3, 6, 7]$ $\rightarrow dq.front == i-k \rightarrow \text{false}$ $nums[0] < nums[1]$ $\therefore \text{pop back \& push } i$	<u>0</u> <u>1</u>	[]
$i=2$	$\overset{0}{[1}, \overset{1}{3}, \overset{2}{-1}, -3, 5, 3, 6, 7]$ $\rightarrow dq.front == i-k \rightarrow \text{false}$ $nums[1] < nums[2]$ $\therefore \text{false \& push } i$	<u>1, 2</u>	$\begin{matrix} 3 \\ \uparrow \\ \backslash \end{matrix}$
			$\rightarrow \text{as } i \geq k-1$ push $nums[dq.front()]$ i.e. 3 into res

$i=3$ [1, 3, -1, -3, 5, 3, 6, 7]

$\rightarrow dq.front == i-k \rightarrow \text{false}$

$\text{num}[2] < \text{num}[i]$

$\therefore \text{false} \ \& \ \text{push } i$

1, 2, 3

[3, 3]

↑

↓

;

$\rightarrow \text{as } i >= k-1$

push $\text{num}[dq.front()]$ ie 3
into res

$i=4$ [1, 3, -1, -3, 5, 3, 6, 7]

$\rightarrow dq.front == i-k \quad \text{true} \quad \therefore \text{pop front}$

$-3 \quad 5$
 $\text{num}[3] < \text{num}[i] \quad \therefore \text{pop back}$

$-1 \quad 5$
 $\text{num}[2] < \text{num}[i] \quad \therefore \text{pop back}$

& push(i)

order & pop
① 1, 2, 3 ② 4

[3, 3, 5]

↑

;

$\rightarrow \text{as } i >= k-1$

push $\text{num}[dq.front()]$ ie 5
into res

$i=5$ [1, 3, -1, -3, 5, 3, 6, 7]

$\rightarrow dq.front == i-k \rightarrow \text{false}$

$5 \quad 3$
 $\text{num}[4] < \text{num}[i]$

$\therefore \text{false} \ \& \ \text{push}(i)$

order & pop
4, 5

[3, 3, 5, 5]

↑

;

$\rightarrow \text{as } i >= k-1$

push $\text{num}[dq.front()]$ ie 5
into res

$i=6$ [1, 3, -1, -3, 5, 3, 6, 7]

$\rightarrow dq.front == i-k \rightarrow \text{false}$

$3 \quad 6$
 $\text{num}[5] < \text{num}[i] \quad \therefore \text{pop back}$

$5 \quad 6$
 $\text{num}[4] < \text{num}[i] \quad \therefore \text{pop back}$

& push

order & pop
③ 4, 5, 6

[3, 3, 5, 5, 6]

↑

;

$\rightarrow \text{as } i >= k-1$

push $\text{num}[dq.front()]$ ie 6
into res

$i=7$ [1, 3, -1, -3, 5, 3, 6, 7] ~~order of pop~~
~~① 6, 7~~ [3, 3, 5, 5, 6, 7]

$\rightarrow dq.front == i-k \rightarrow \text{false}$ $\rightarrow \text{as } i \geq k-1$
 6 7
 $\text{num}_6 < \text{num}_i \therefore \text{pop_back}$ push $\text{num}[dq.front()]$ i.e. 7
 q push(i)

code → $Tc \rightarrow O(N)$
 $Sc \rightarrow O(K)$

```

● ● ●

1 class Solution {
2 public:
3     vector<int> maxSlidingWindow(vector<int>& nums, int k) {
4         deque <int> dq;
5         vector <int> ans;
6         for (int i = 0; i < nums.size(); i++) {
7
8             if (!dq.empty() && dq.front() == i - k)
9                 dq.pop_front();
10
11            while (!dq.empty() && nums[dq.back()] < nums[i])
12                dq.pop_back();
13
14            dq.push_back(i);
15
16            if (i >= k - 1)
17                ans.push_back(nums[dq.front()]);
18        }
19        return ans;
20    }
21 };
  
```

<https://www.linkedin.com/in/karun-karthik>