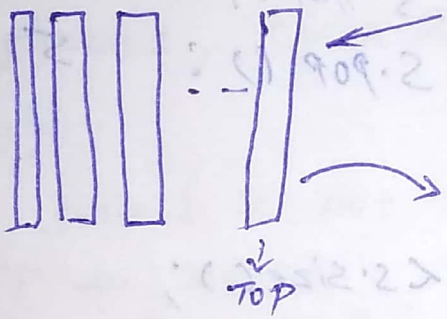
```
for(int i=0; i< colour.size(); i++)
    cout << colour[i] << "\n";
}
```

Stack Data Structure

linear data structure follows particular order in which operation are performed. Order may be LIFO (Last in first out) or FILO (First in last out)

Stack insertion and deletion happen on same side

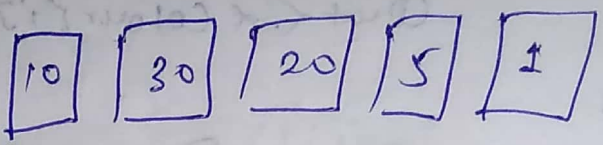


Last in, First out

- empty() :— Returns whether stack is empty — time complexity O(1)
- size() :- Returns size of stacks — time complexity O(1)
- top() :- Returns ~~reference to~~ top most element of stack — Time complexity O(1)
- push(g) :- Add element 'g' at top of stack. time complexity : O(1)
- pop() — deletes top most element of stack time Complexity : O(1)

```
#include <bits/stdc++.h>
using namespace std;
Void showstack (stack<int> s)
{ while (!s.empty())
  { cout << ' ' << s.top();
    s.pop();
  }
```

```
    cout << "\n";
}
int main ()
{ stack <int> s;
  s.push(10);
  s.push(30);
  s.push(20);
  s.push(5);
  s.push(1);
  cout << "the stack is:";
  showstack(s);
  cout << "\n s.size() : " << s.size();
  cout << "\n s.top() : " << s.top();
  cout << "\n s.pop() : ";
  s.pop();
  showstack(s);
  return 0;
}
```



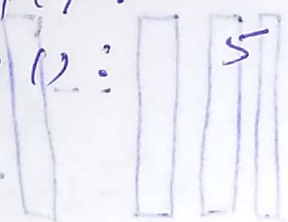
Output:-

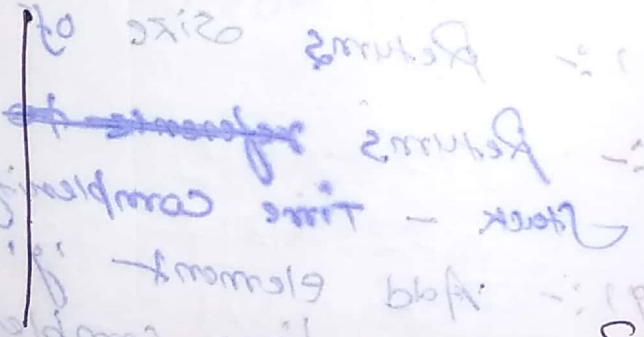
The stack is : 1 5 20 30 10

S.size() : 5

S.top() : 1

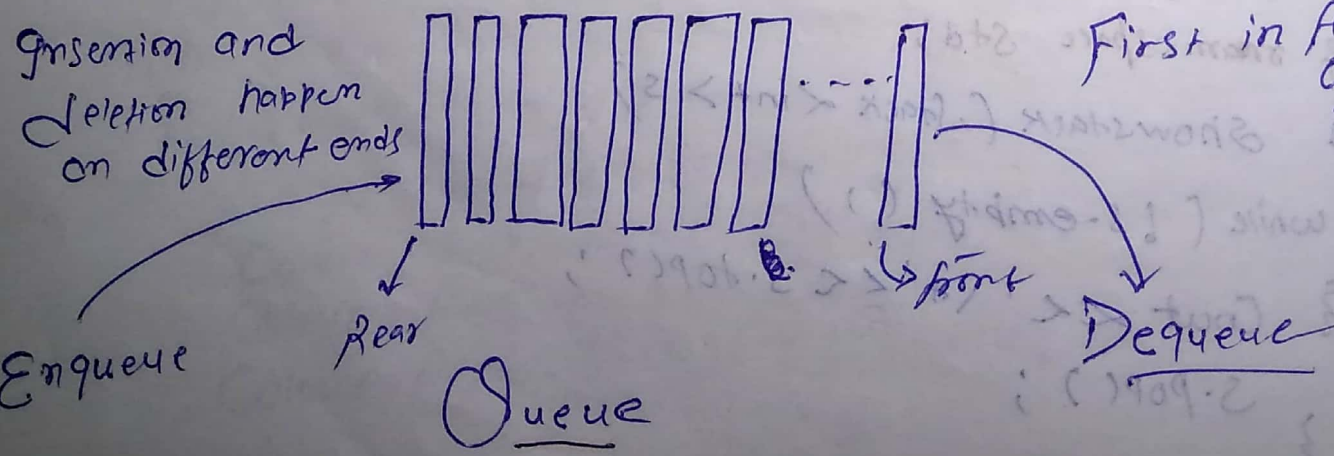
S.pop() :





Implementation of Queue using Stacks

Given, Stacks data structures and we have to implement Queue using pop() and push() operation.



Queue can be implemented using two stacks. Stack 1 and stack 2. q (queue) can be implemented into two ways :-

Method-1 (By making enqueue operation costly) :-

This method make sure that oldest entered element is at always top of stack 1, so that dequeue operation just pops from stack 1. To put element at top of stack 1, stack 2 is used.

enQueue (q, x):

→ while stack 1 is not empty, push everything from stack 1 to stack 2.

→ push x to stack 1 (assuming size of stack is unlimited)

→ push everything back to stack 1

Here time complexity O(n)

dequeue (q):

→ if stack 1 is empty then error

→ pop an element from stack 1 and return it.

time complexity O(1)

```
#include <bits/stdc++.h>
using namespace std;
Stack Queue
{ Stack<int> s1, s2;
  void enQueue (int x)
  { while (!s1.empty())
    { s2.push(s1.top());
      s1.pop();
```

```
    S1.push(x);
    while (!S2.empty())
    { S1.push(S2.top());
      S2.pop();
    }
}
int deQueue()
{ if(S1.empty())
  { cout << "Q is Empty";
    exit(0);
  }
  int x = S1.top();
  S1.pop();
  return x;
}
};
int main()
{ Queue q;
  q.enQueue(1);
  q.enQueue(2);
  q.enQueue(3);
  cout << q.deQueue() << '\n';
  cout << q.deQueue() << '\n';
  cout << q.deQueue() << '\n';
  return 0;
}
```

Time Complexity :-

Push operation : O(N)

Pop operation : O(1)

Auxiliary space : O(N)

use of stack for storing values.

Output :- 1
2
3

Stack introduction and Identification

7 Question :-

1> Nearest	Greater	to left	→ same concept
2> Nearest	"	to Right	(Next largest element)
3> "	Smaller	" Left	(Nearest Smaller ")
4> "	"	to Right	

5> stock Spain Problem
6> maximum area of Histogram

7> max Area of Rectangle in binary matrix

6 question :-

8> Rain water trapping
9>. Implementing a min Stack
10>. Implementing Stack using Heap
11>. The Celebrity problem
12>. Longest valid parenthesis
13>. Iterative ToH (Tower of Height) implementation

Identification :- i> array
of stack
question ii> $O(n^2)$ ← brute force.

Second loop → I loop dependent on i

then we always use Stack in this question for better Solution.

Next largest element / Nearest greater to Right

arr[] : 1 3 2 4

o/p :- 3 4 4 -1

arr[] : 1 3 0 0 1 2 4

o/p :- 3 4 1 1 2 4 -1

```
for(int i=0; i<n; i++)
  for(int j=i+1; j<n; j++)
  {
  }
```

-> So this can be implemented using stacks.


```
Vector<int> v;
Stack<int> s;
for(int i=n-1; i>=0; i--)
{ if( s.size()==0)
      v.push_back(-1);
  Else if ( s.top() > arr(i) )
    { v.push_back(s.top());
    }
  Else if ( s.top() <= arr(i) )
    { while( s.size()>0 && s.top() <= arr(i) )
        { s.pop();
        }
      if ( s.size()==0)
          v.push_back(-1);
      else
          v.push_back ( s.top());
    }
```

S.push(arr[i]);

}

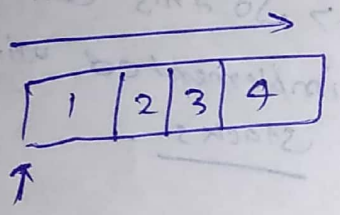
Reverse vector v;

Nearest greater to left

arr[]: 1 3 2 4

↑ ↑ ↑ ↑

-1 -1 3 -1



Brute:- ✓ dependent on i.

```
for(int i=0; i<n; i++)
    for(int j=i-1; j>=0; j--)
    {

    }
```

Change → i> left to Right

i> no need for changes.

```
vector<int> v;
stack<int> s;
for(int i=0; i<size; i++)
{ if( s.size() == 0)
      v.push_back(-1);
  else if( s.top() > arr[i])
      v.push_back( s.top());
  else if( s.size()>0 && s.top()<=arr[i])
  { while( s.size()>0 && s.top()<=arr[i])
    { s.pop();
    }
    if( s.size()==0)
      v.push_back(-1);
    else
      v.push_back( s.top());
  }
  s.push(arr[i]);
}
```

Nearest Smaller to left / Nearest Smaller element

arr[]:- 4 5 2 10 8

-1 4 -1 2 2

Stack vector

```
for(int i=0; i<size; i++)
{   if(S.size()==0)
        v.push_back(-1);
    Else if(S.size()>0 && S.top()<arr[i])
        v.push_back(S.top());
    Else if(S.size()>0 && S.top()>=arr[i])
    {   while(S.size()>0 && S.top>=arr[i])
        {   S.pop();
        }
        if(S.size()==0)
            v.push_back(-1);
        Else
            v.push_back(S.top());
    }
    S.push(arr[i]);
}
```

Nearest Smaller to Right

Traverse from right

Reverse ✓

arr[] : 4 5 2 10 8

Stack ←

Normal →

```
for(int i= size-1; i>=0; i--)
{   if(S.size()==0)
        v.push_back(-1);
```

```
Else if( S.top() <= arr[i])
        V.Push_back(S.top());
Else
{   while( S.size()>0 && S.top()>=arr[i])
        {S.pop()
        }
    if(S.size()==0)
      V.push_back(-1);
    Else
      V.push_back(S.top());
    }
  S.push(arr[i]);
}
reverse(V.begin(), V.end());
}
```

Stock Span Problem

arr: 100 80 60 70 60 75 85

Day 1 1 1 2 4 6

Consecutive smaller or equal elements → before i

↓ Span value of day i

left to Right

We have to find Nearest greater to left

100 80 60 70 60 75 85

80 70 75

Nearest greater to left

100	80	60	70	60	75	85
0	1	2	3	4	5	6
-1	100	80	80	70	80	100

6-0 = 6

4-3 = 1

5-1 = 4

⇒ arr :-

0	1	2	3	4	5	6
100	80	60	70	60	75	85

V :-

-1	0	1	1	3	1	0

← Nearest greater to left

1 1 1 2 1 4 6

Stack <int> s ⟶ stack < pair < int, int > > s;

NGE, index

vector <int> v;

for (int i=0; i < size; i++)

{ if (s.size() == 0)

v.push_back(-1);

else if (s.top().first > arr[i])

v.push_back (s.top().second);

else

{ while (s.size() > 0 && s.top().first <= arr[i])

{ s.pop(); }

if (s.size() == 0)

v.push_back (-1);

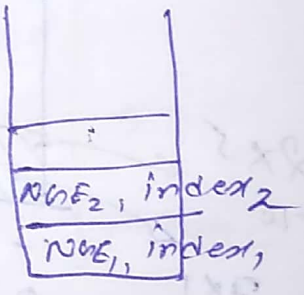
else

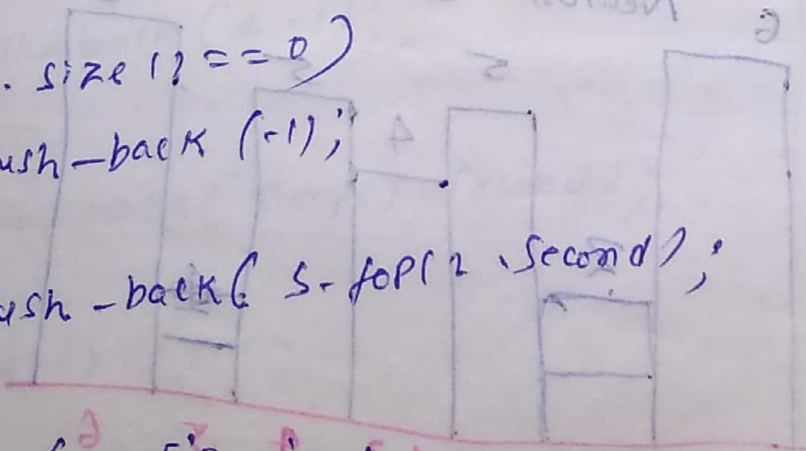
v.push_back (s.top().second);

}

s.push (arr[i], i);

}





```
for(int i=0; i<size(); i++)
{ v[i] = 1 - v[i];
}
return v;
}
```

Sum up :-

Stock Span Problem → Consecutive Smaller or equal to left → NGL index → Next Greater Left → i - NGL index.

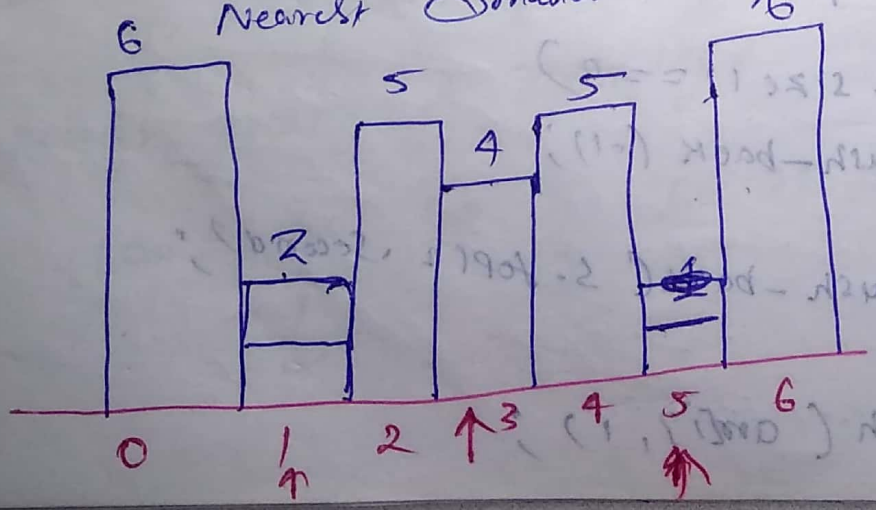
maximum area of histogram

arr[] : 6 2 5 4 5 1 6 → Height

For any building expend upto greater or equal to height on both side.

↓

Nearest smaller to left index
Nearest Smaller to right index



	0	1	2	3	4	5	6	
arr:	6	2	5	4	5	1	6	
left:	-1	-1	1	1	3	-1	5	← NSL index
Right:	1	5	3	5	5	7	7	← NSR index

Answer :- **Right - left - 1 = width[i]**

Area[i] = arr[i] (height) × width[i]

```
Stack < pair < int, int >> S1, S2
Vector < int > left
vector < int > Right
vector < int > width
For (int i=0 ; i<size; i++)
{ if ( S.size() ==0 )
    left.push_back(-1);
  Else if ( S.top().first < arr[i])
    left.push_back(S.top().second);
```

→ NSL

```
  Else
  { while ( S.top().first >= arr[i] && S.size() >0)
      { S.pop();
      }
    if (S.size() ==0)
      left.push_back(-1);
    Else
      left.push_back(S.top().second);
  }
  S.push(arr[i], i);
}
```

```
for(int i=size-1 ; i>=0 ; i--)
{ if(s.size()==0)
    right.push_back(7);
  Else if (s.top().first < arr[i])
    right.push_back(s.top().second);
  Else
    { while(s.size()>0 && s.top().first >= arr[i])
      { s.pop(); }
      if(s.size()==0)
        right.push_back(size);
      Else
        right.push_back(s.top().second);
    }
  s.push(arr[i], i);
}
reverse(right.begin(), right.end());
}
for(i=0; i<size; i++)
  width[i] = Right[i] - Left[i] - 1;
int max = width[0] * arr[0];
for(int i=1; i<size; i++)
{ if(max < width[i] * arr[i])
    max = width[i] * arr[i];
}
```

NSR

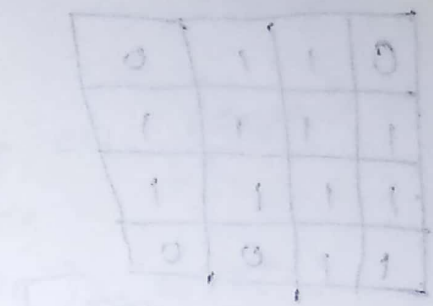
area

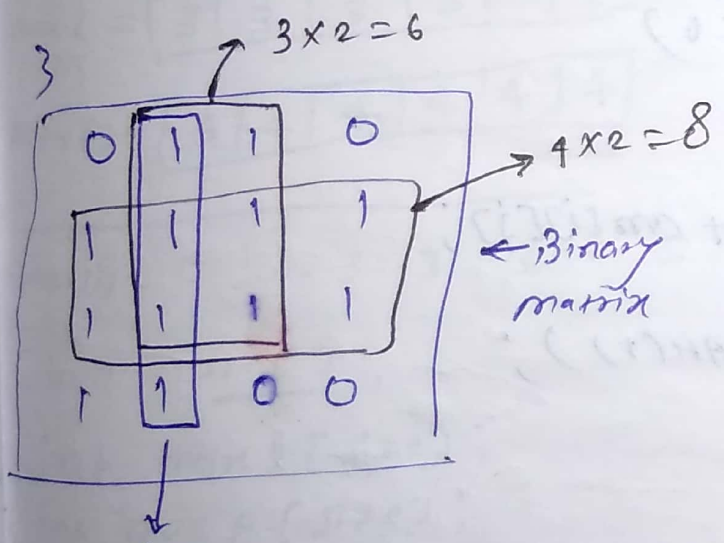
return max ;

7. Maximum area of Rectangle in binary matrix

Maximum area of histogram code :-

```
MAH ( int arr[], int size )
{ right[] → NSR (arr, size)
  left[] → NSL (arr, size)
  width[i] = Right[i] - Left[i] - 1
  area[i] = arr[i] * width[i]
  return max area[i]
```



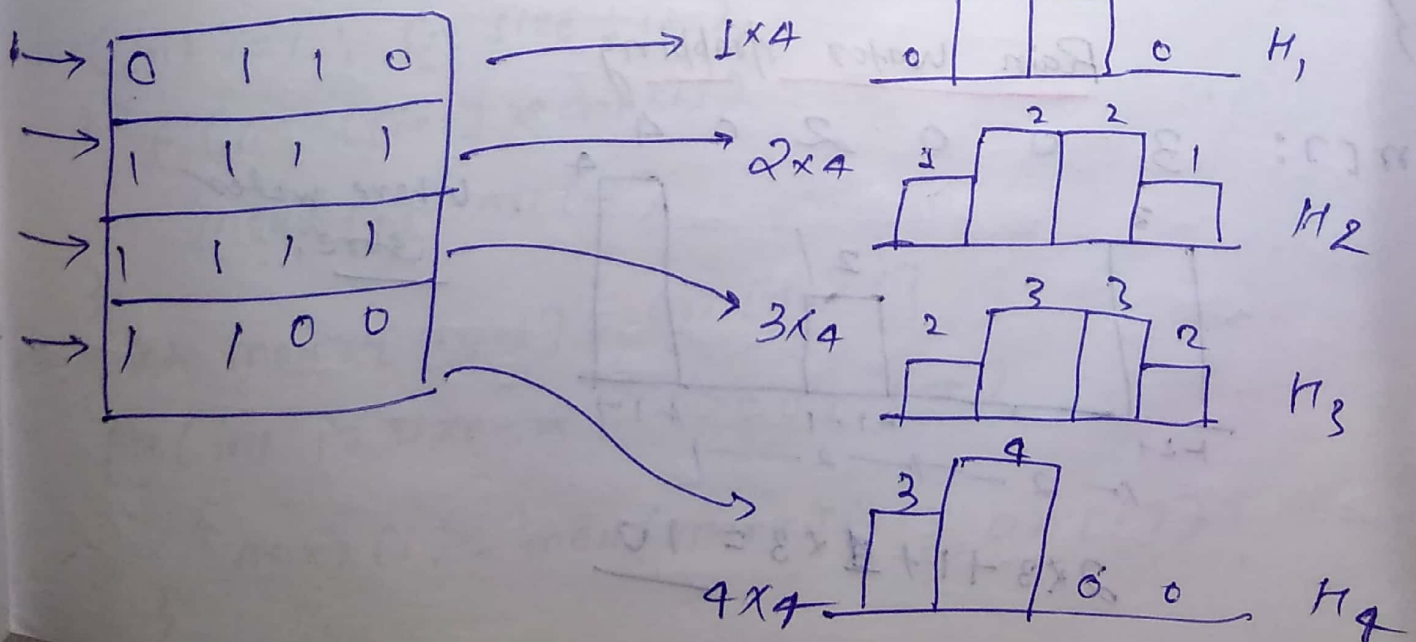


difference :-

2D array

Binary number.

Compress 2D array to 1D array



ans = max { max(H1) → 2, max(H2) → 4, max(H3) → 8, max(H4) → 6 } → max = 8

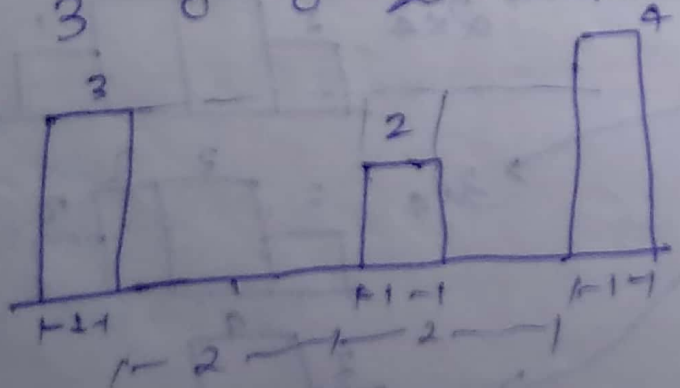
	j=0	1	2	3
i=0	0	1	1	0
1	1	1	1	1
2	1	1	1	1
3	1	1	0	0

n x m
arr[][]

```
Vector<int> v;
for(int j=0; j<m; j++)
    v.push_back(arr[0][j]);
int max = mah(v);
for(int i=1; i<n; i++)
{ for(int j=0; j<m; j++)
    { if(arr[i][j]==0)
        v[j]=0;
      else
        v[j] = v[j] + arr[i][j];
    }
  max = max(max, mAH(v));
}
return max;
}
```

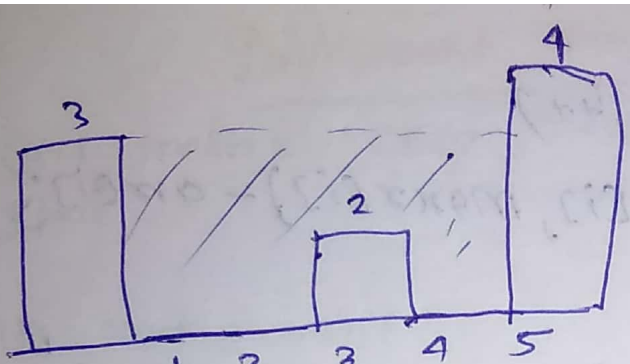
Rain water trapping

arr[]: 3 0 0 2 0 4



where water store.

2×3 + 1 + 2×3 = 10



Σ water on above each building = 0 + 3 + 3 + 1 + 3 = 10

water on any i^{th} buiding dependent on Largest building on Left and Right side:

water[i] = min(left side maxm, right side maxm) − arr[i]

3 0 0 2 0 4

maxl = 3 3 3 3 3 4

maxr[] = 4 4 4 4 4 4

water[i] = 0 3 3 1 3 0 = 10

0 1 0 2 1 0 1 3 2 1

0 1 1 2 2 2 2 3 3 3

3 3 3 3 3 3 3 3 2 1

Given :- arr[],

```
int maxl[size];
int maxR[size];
maxl[0] = 3;
for(int i=1 ; i< size ; i++)
  { if (maxl[i-1] < arr[i])
      maxl[i] = arr[i];
  }
maxr[size-1] = arr[size-1]
for(int i= size-2; i>=0; i--)
  { maxr[i] = max(maxr[i+1], arr[i]);
  }
```

```
int water[size];
for(int i=0 ; i<size ; i++)
    water[i]= min(maxl[i], maxr[i])-arr[i];

int sum=0;
for(int i=0; i<size ; i++)
    sum += water[i];
```

Return sum;

Implement minstack with extra space - O(n)

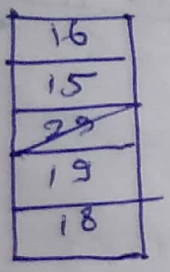
18 19 29 15 16

Find minm element in stack.

Stack (S)

Supporting Stack (SS)

top of it always contain minimum value to give min output in O(1) time.



Stack → push
→ pop

```
stack<int> s;
stack<int> ss;

void push(int a)
{
  s.push(a);
  if(ss.size()==0)
    ss.push(a);
  else if(a<=ss.top())
    ss.push(a);
  return;
}
```

```
int pop()
{ if(s.size()==0)
    return -1;
  if(s.top()==ss.top())
    ss.pop();
  s.pop();
  return s.top();
}
```

```
int getmain()
{
  if(ss.size()==0)
    return -1;
  return ss.top();
}
```

Implement minstack without using extra O(1) space

O(1) means any number of variable.

Constant

(2*x-min)
(2*min-y)

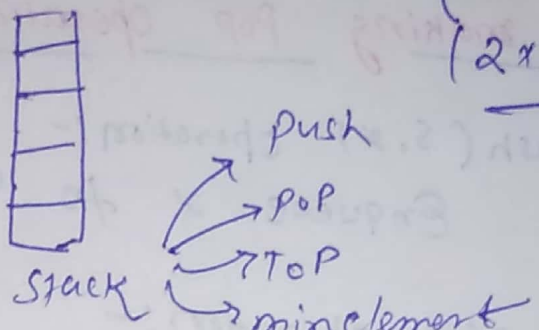
int minElement;

~~minElement = 0;~~

int getMin()
{ if (s.size() == 0)
    return -1;
  return minElement;
}

Void push(x)
{ if (s.size() == 0)
    minElement = x;
  if (x >= minElement)
    s.push(x);
  Else
  { s.push(2x - minElement);
    minElement = x;
  }
}

int top()
{ if (s.size() == 0)
    return -1;
  else { if (s.top() >= minElement)
           return s.top()
         else if (s.top() < min Element)
           return minElement;
  } }



void pop()
{ if (s.size() == 0)
    return;
  Else if (s.top() >= min Element)
    s.pop();
  Else if (s.top() < min element)
  { minE = 2*minE - s.top()
    s.pop();
  }
}

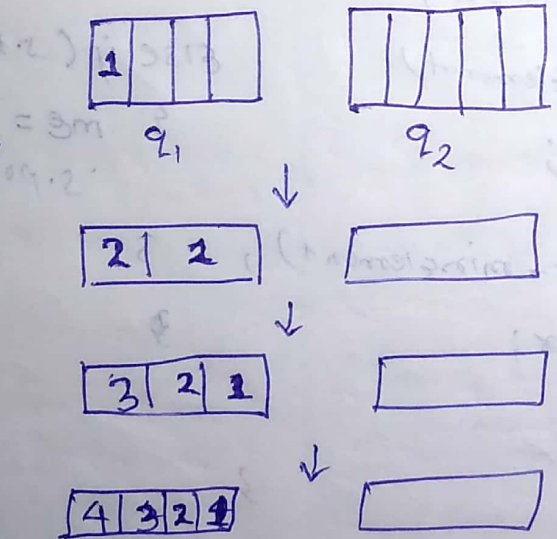
Implementation of Stack using Queue

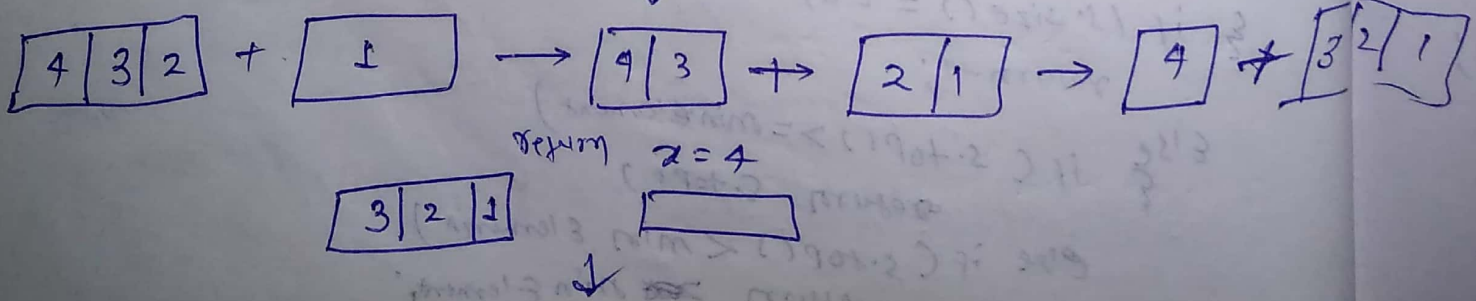
1.> By making Pop operation Costly :-

1. Push(S, x) operation:-
   Enqueue x to q_1 (assuming size of q_1 is unlimited)

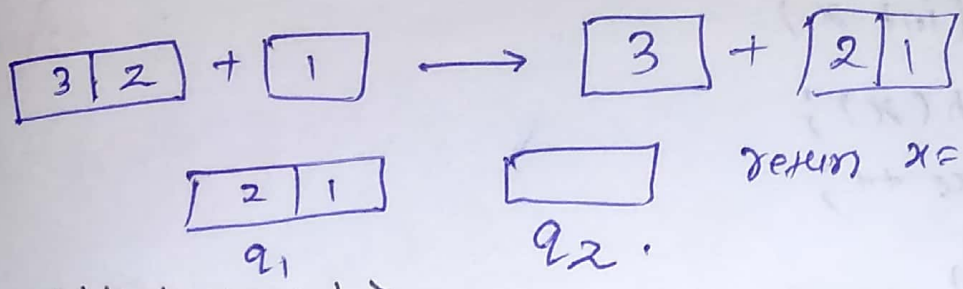
2. Pop(S) operation:-
   → One by one dequeue everything except the last element from q_1 and enqueue to q_2.
   → Dequeue last item of q_1, the dequeued item is result, store it.
   → Swap names of q_1 and q_2
   → Return item stored in step 2.

S.push(1)
S.push(2)
S.push(3)
S.push(4)
S.pop()
S.pop()





return x = 4




```
#include<bits/stdc++.h>
using namespace std;
class Stack
{
  queue<int> q1, q2;
  int curr_size;
public:
  Stack()
  { curr_size = 0;
  }
  void pop()
  { if (q1.empty())
      return;
    while (q1.push != 1)
    { q2.push(q1.front());
      q1.pop();
    }
    q1.pop();
    curr_size--;
    queue<int> q = q1;
    q1 = q2;
    q2 = q;
}
```

```
Void push(int x)
{ q1.push(x);
  curr_size++;
}
int top()
{ if (q1.empty())
    return -1;
  while (q1.size() != 1)
  { q2.push(q1.front());
    q1.pop();
  }
  int temp = q1.front();
  q1.pop();
  q2.push(temp);
  queue<int> q = q1;
  q1 = q2;
  q2 = q;
  return temp;
}
int size()
{ return curr_size;
}
};
```

```
int main()
{ Stack S;
  S.push(1);
  S.push(2);
  S.push(3);
  S.push(9);
  Cout << S.size() << endl;
  Cout << S.top() << endl;
  S.pop();
  Cout << S.top() << endl;
  S.pop();
  return;
```

Output:-

4
4
3

Implement Stack using single Queue

Push (s, x)
→ Size of q be s
→ Enqueue x to q
→ One by one Dequeue s items from queue and enqueue item.

Pop (s)
→ Dequeue an item from q.

```
#include<bits/stdc++.h>
using namespace std;
Class Stack
{ queue<int> q;
  Public:
     void push(int val);
     Void pop();
     int top();
     bool empty();
};
Void Stack::push(int val)
{ int s=q.size();
  q.push(val);
  for(int i=0; i<s; i++)
  { q.push(q.front()); q.POP();
  }}
```

```
void Stack::pop()
{ if(q.empty())
    return;
  q.pop();
}
int Stack::top()
{ return (q.empty()) ? -1 : q.front();
}
bool stack::empty()
{ return (q.empty());
}
int main()
{ Stack s;
  s.push(10);
  s.push(20);
  cout<<s.top()<<endl;
  s.pop();
  s.push(30);
  s.pop();
  cout<<s.top()<<endl;
  return 0;
}
```

Output:-

20
10

Stack (Infix to Postfix)

Infix expression:- expression of form a op b.
when operator is in-between every pair of operands.

Postfix expression:- expression of form a b op
when operator is followed by every pair of operands.

a+b*c+d ⟶ postfix a b c * + d +
infix a+b*c+d

```
#include<bits/stdc++.h>
using namespace std;
int prec(char c)
{ if(c=='^')
   return 3;
 else if(c=='*'||c=='/')
   return 2;
 else if(c=='+'||c=='-')
   return 1;
 else
   return -1;
}
void infixToPostfix(string s)
{ std::stack<char> st;
   st.push('N');
   int l=s.length();
   string ns;
```

```
For(int i=0; i<l; i++)
{ if((s[i]>='a' && s[i]<'z') || (s[i]>='A' && s[i]<='Z'))
    ns+=s[i];
  else if (s[i]=='(')
     St.push('(');
  else if (s[i]==')')
  { while (St.top()!='N' && st.top()!='(')
      { char c = St.top();
        St.pop();
        ns+=c;
      }
    if(s.top()=='(')
       St.pop();
    }
  else
    { while (st.top()!='N' && prec(s[i])<=prec(St.top()))
      { char c = St.pop();
        St.pop();
        ns+=c;
      }
      St.Push(s[i]);
    }
}
while(St.pop()!='N')
{ char c = St.top();
  St.pop();   ns+=c;
}
```

```
    cout << ns << endl;
    }
int main()
{ string exp = "a+b*(c^d-e)^(f+g*h)-i";
  infixToPostfix(exp);
  return 0;
}
```

Output :- abcd^e-fgh*+^*+i-

Prefix to Infix Conversion

Prefix :- if operator appears in expression before operands.

Example :- *+AB-CD

= (A+B)*(C-D) infix

Algorithm :-

→ Read prefix expression in reverse order (from right to left)

→ if symbol is an operand, then push it onto stack

→ if symbol is an operator, then pop two consecutive operands from top of stack. Create a string by concatenating the two operands and operator b/w them

String = (operand 1 + Operator + operand 2)

Push resulting string into stack.

→ Repeat above step until end of prefix expression.

```
String PreToInfix ( String Pre_exp)
{ Stack <string> s;
  int length = Pre_exp.size();
for (int i= length-1; i>=0; i--)
{ if ( pre_exp[i] = '+' || pre_exp[i]= '-' ||
       Pre-exp[i] = '*' || Pre_exp[i]= '/')

  { string op1 = s.top(); s.pop();
    ||     op2 = s.top();  s.pop();
    String temp = "(" + op1 + Pre_exp[i] + op2 + ")";
    s.push (temp);
  }
 else
  { s.push ( String (1, Pre_exp[i]));
  }
}
return s.top();
}
```

Prefix to Postfix

Algorithm :-

→ read prefix in reverse order. (Right to left)
→ if symbol is operand, push it onto stack
→ if symbol is Operator, pop two operands from stack create a string

string = operand1 + operand2 + operator

Push it back to stack

→ Repeat until end of prefix expression.

```
String PreToPost ( String Pre_exp)
{ stack <string> S;
int length= Pre_exp.size();
for(int i= length-1; i>=0; i--)
{ if (isoperator.(Pre_exp[i]))
  { String op1 = S.top(); S.pop();
    String op2 = S.top(); S.pop();
  String temp= op1 + op2 + Pre_exp[i];
  S.push(temp);
  }
else { S.push ( String (', Pre_exp[i]));
  }
} return S.top();
}
```

Example postfix :- ABC/-AK/L-*

Prefix :- *-A/BC-/AKL";

Postfix to Prefix

- Read postfix expression from left to right.
- → if symbol is an operand, push it onto the stack
- → if symbol is an operator, pop two elements from stack, create a string by concatenating two operands and operator.

String = operator + operand2 + operand1

Push string back to stack.

→ Repeat until end of postfix expression.

Check for balanced Parenthesis in an expression

Example :- Input :- "[()]{}{[()()]()}"

Output :- Balanced

input :- "[(])"

Output :- Not Balanced

Algorithm :-

→ Declare a character stack S.

→ Now traverse expression string

1. if current character is starting bracket ('{', '(', '[') then push it to stack.
2. if current character is closing bracket (')', '}', ']') the pop from stack and if popped character is matching starting character then fine else else not balanced.

→ After complete traversal, if stack is not empty means not balanced else balanced.

```
bool areBalanced ( string expr)
{  stack < char > s;
   Char x;
   for ( int i=0 ; i< expr.length() ; i++)
   { if ( exp[i] == '(' || exp[i] == '{' || exp[i] == '[')
        { s.push ( exp[i]);
          Continue;
        }
   else if ( s.empty ())
        return false;
```

```
Switch ( exp[i] )
{ case ')':
        x = S.top();
        S.pop();
        if ( x != '(' )
            return false;
        break;
  Case '}':
        x = S.top();
        S.pop();
        if ( x != '{' )
            return false;
        break;
  Case ']':
        x = S.top();
        S.pop();
        if ( x != '[' )
            return false;
        break;
  }
} return ( S.empty() );
}
```

Length of longest valid substring

input:- ((()

Output:- 2 ()

input :- ()(())

output:- () (2)

input)()())

Output:- 4 :-()()

1) Create an empty stack and push -1 to it. first element of stack is used to provide a base for next valid string.

2) initialize result as 0.

3) If character is '(' push index i to stack

4) Else [if character is ')']

a) Pop an item from stack

b) if stack is not empty .

result = max (result, i - s.top());

c) if stack is empty

push current index for base for next valid substring.

5. Return Result.

```
int findmaxLen (string str)
{ int n = str.length();
  stack <int> st;
  st.push(-1);
  for (int i=0 ; i<n ; i++)
    { if (str[i] == '(')
        st.push(i);
```

```
else
{ st.pop();
  if(!st.empty())
      result = max(result, i-st.top());
  Else
    st.push(i);
}
}
return result;
```

Priority Queue

It is an extension of Queue with following properties:-

i) Every item has a priority associated with it.

ii) An element with high priority is dequeued before element with low priority.

iii) If two elements have same priority, they are served according to their oder in the queue.

Example:- Queue with priority according highest ascii value.

initial Queue = { }

Operation	Return value	Queue content
insert (C)		C
insert (O)		CO
insert (D)		COD
remove max	O	CD
insert (I)		CDI
insert (N)		CDIN
remove max	N	CDI
insert (S)		CDIS

Typical priority queue support following operations:-

i) insert (item, priority) :- insert given item with priority

ii) getHighestPriority ( ) :- Returns highest priority item.

iii) deleteHighestPriority ( ) :- Removes " " ".

Deque

Deque or Double ended Queue is generalised version of Queue data structures that allows insertion and deletion at both ends.

Operation on deque :-

i) insertFront ( ) :- Add items at front of deque.

ii) insertLast ( ) :- " " " last " "

iii) deleteFront ( ) :- delete " " front " "

iv) deleteLast ( ) :- " " " last " ".

v) getFront ( ) :- Gets front item from deque.

vi) getRear ( ) :- Gets rear (last) " " " ".

vii) isEmpty ( ) :- Returns true if it is empty.

viii) isFull ( ) :- Checks whether deque full or not.

Reverse a queue using stack

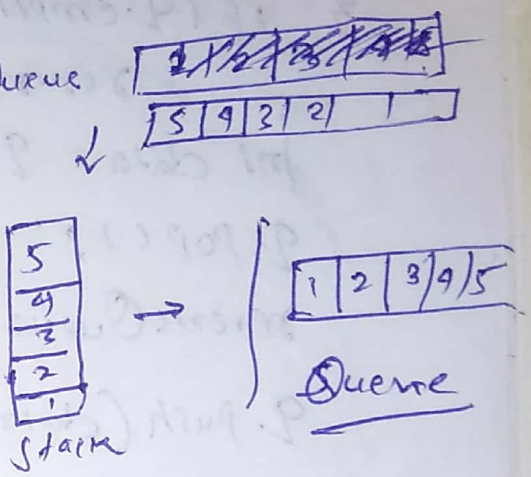
input :- [1, 2, 3, 4, 5]
Queue

Output Queue :- [5, 4, 3, 2, 1]

Approach :- i) Pop element from queue and insert into stack (first element become last)

ii) Pop element from stack and insert into queue.

```
void reverseQueue ( Queue<int> &Queue )
{ Stack <int> Stack;
  while ( Queue.size() != 0)
  { Stack.Push ( Queue.front() );
    Queue.pop();
  }
  while ( Stack.size() != 0)
  { Queue.Push ( Stack.top() );
    Stack.pop();
  }
}
```



Reversing of Queue using Recursion

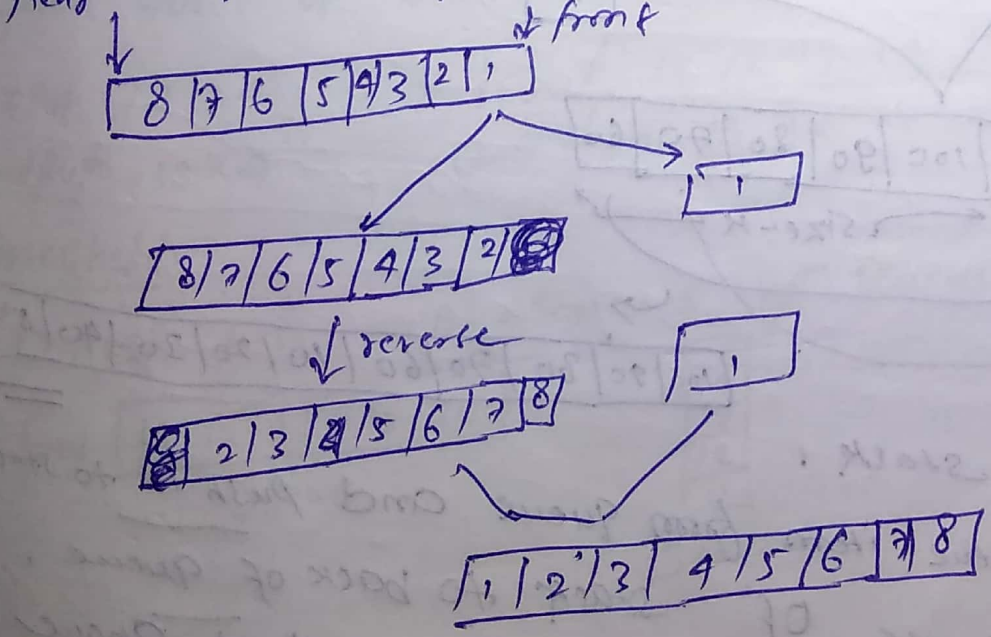
input : Q = | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Output Q = | 8 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Algorithm :- i) pop element from queue if queue has element else return empty queue

ii) Call reverseQueue function for remaing queue.

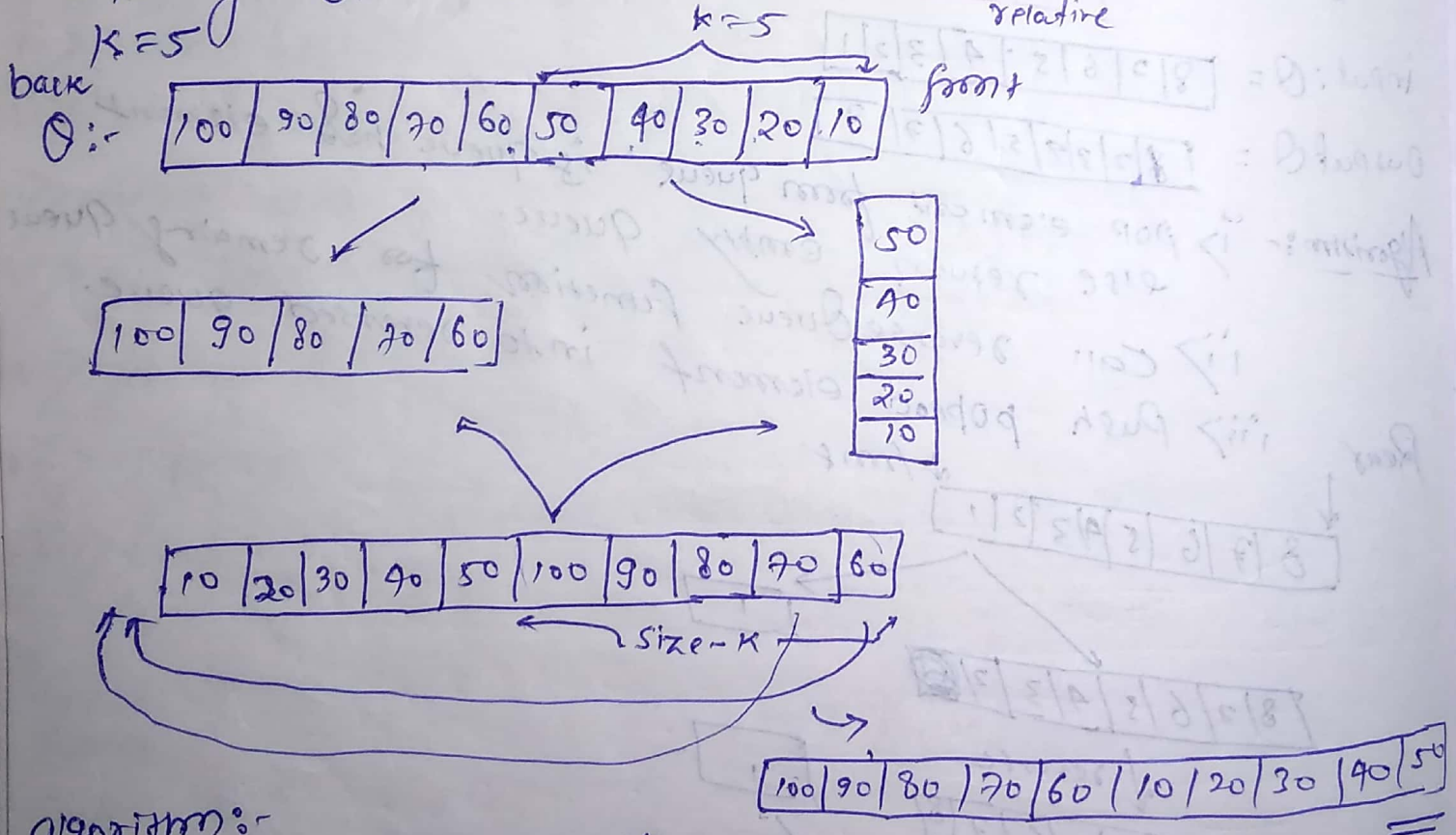
iii) Push popped element into reversed queue



```
void reverseQueue (queue<int> &q)
{ if (q.empty())
      return;
  int data = q.front();
  q.pop();
  reverseQueue (q);
  q.push(data);
}
```

Reversing first K elements of Queue

We have to reverse first K elements of queue.
- leaving other element in same order.



algorithm:-
i) Create empty stack.
ii) One by dequeue item from queue and push it to stack.
iii) Enqueue contents of stack to back of queue.
iv) Dequeue (size-k) elements from front and Enque them one by one to same queue.

```
void reverseQueueFirstKElements ( int K, Queue<int> Queue)
{ if (Queue.empty() == true || K > Queue.size() || K <= 0)
        return;
  Stack <int> S;
  for (int i=0; i<K; i++)
  { S.push (Queue.front());
    Queue.pop();
  } while (!stack.empty())
  { Queue.push (Stack.top());
    Stack.pop();
  }
for (int i=0; i< Queue.size() - K; i++)
  { Queue.push (Queue.front());
    Queue.pop();
  }
}
```

Sorting a Queue without extra space.

With extra space we do simply by dequeue element to array then sort array and enqueue.

```
int minIndex ( queue<int> &q, int sortedIndex )
{ int min_index = -1;
  int min_val = INT_MAX;
  int n = q.size();
  for(int i=0; i<n; i++)
  { int curr = q.front();
    q.pop();
    if ( curr <= min_val && i <= sortedIndex)
    { min_index = i;
      min_val = curr;
    }
    q.push(curr);
  }
  return min_index;
}
Void insertMinToRear (queue<int> &q, int min_index)
{ int min_val;
  int n = q.size();
  for (int i=0; i<n; i++)
  { int curr = q.front();
    q.pop();
    int n = q.size();
    if ( i != min_index)
      q.push(curr);
    else
      { min_val = curr; }
      break
  }
```

```
    q.push(min_val);
  }
void SortQueue( queue<int> &q)
{ for( int i=1 ; i<=q.size();i++)
   { int min_index=MinIndex( q, q.sizer()-i);
     insertMinToRear( q, min_index);
   }
}
```

Interesting method to generate Binary numbers from 1 to n using Queue

input :- n=5

output:- 1, 10, 11, 100, 101

```
Void generatePrintBinary (int n)
{ queue < string > q;
  q.push ("1");
  while (n--)
{ string s1 = q.Front();
    q.pop();
  string s2 = s1;
  q.push ( s1. append("0"));
  q.push ( s2. append("1"));
  }
}
```

Circular tour

There is a circle. There are N petrol pumps on that circle. At this two data sets are given:—
i) Amount of petrol that every petrol pump has.
ii) Distances from petrol pump to next petrol pumps.
Find a starting point where truck can start to get through complete circle without exhausting petrol in between.
Assume:- for 1 litre petrol, truck can go 1 unit of distance.

Input:- 4 → N

4,6	6,5	7,3	4,5

petrol → distances

Output :- 1 (0 indexing)

An efficient approach is use Queue to store current tour. we first enqueue first petrol pump to queue, we keep enqueueing petrol pumps till we either complete tour, or current amount of petrol becomes negative. if Curr_petrol become negative we keep - dequeuing petrol pumps until Curr_petrol become non negative.

Instead we use two pointer start and end that represents rear and front of queue.

```
int PrintTour( PetrolPump arr[], int n)
{  int start=0, end=1;
   int curr_petrol = arr[start].petrol - arr[start].dis
```

```
while (end != start || curr_petrol < 0)
{
    while (curr_petrol < 0)
    {
        curr_petrol -= arr[start].petrol - arr[start].d
        start = (start + 1) % n;
        start = start + 1;
        if (start == n)
            return -1;
    }
    curr_petrol += arr[end].petrol - arr[end].distance;
    end = (end+1) % n;
}
return start;
}
```

Sliding window maximum (maximum of all subarays of size k)

input arr[] :- {1,2,3,1,4,5,2,3,6} k=3

Output :- 3 3 4 5 5 5 6

maximum of 1,2,3 is 3
" " 2,3,1 is 3
" " 3,1,4 is 4
" " 1,4,5 is 5
" " 4,5,2 is 5
" " 5,2,3 is 5
" " 2,3,6 is 6.

1. Create deque to store k elements.
2. Run a loop and insert first k elements in deque. While inserting if element at back of queue is smaller than current element remove all those elements and then insert the elements.
3. Now run a loop from k to end of arr.

4. Print front element of array.
5. Remove element from front of queue if they are out of current window.
6) Insert next element in deque while inserting if element at back of queue is smaller than current element remove all those elements and then insert element.
7. Print maximum element of last window.

```
Void printKmax (int arr[], int n, int k)
{ deque <int> Qi(k);
  for(int i=0; i<k; i++)
    { while((!Qi.empty()) && (arr[i] >= arr[Qi.back()])
        { Qi.pop_back(); }
      Qi.push_back(i);
    }
for(int i=k; i<n; i++)
  { cout << arr[Qi.front()] << " ";
  while((!Qi.empty()) && (Qi.front() <= i-k))
      Qi.pop_front();
  while((!Qi.empty()) && (arr[i] >= arr[Qi.back()])
      Qi.pop_back();
  Qi.push_back(i);
  }
cout << arr[Qi.front()];
}
```

Minimum number of brackets reversals needed to make make an expression balanced

Given expression with only '{' and '}'.

input:- exp = "}{" Output = 2.

input :- }}{{{{}} Output :- 1

O(n) time approach using Stack:-

first remove all balanced part of expression.

Example :- "}{{}}{{{{" to "}{{{{" by removing highlighted part. let m be total number of closing bracket ('}') and n be no. of opening brackets ('{'). we need $[m/2] + [n/2]$ reversals.

Example :- }}}}{{ required $[4/2] + [2/2] = 3$

(ceil) (ceil)

```
Void CountMinReversals (String expr)
{ int len = expr.length();
  if (len % 2)
      return -1;
  Stack <char> s;
  for (int i=0; i<l; i++)
  { if (expr[i]=='}' && !s.empty())
    { if (s.top() == '{')
          s.pop();
      else
          s.push(expr[i]);
    }
```

```
else    // means  expr[i] == '}' or s.empty() == true
    s.push(expr[i]);

}
int n=0, m=0;
while(!s.empty())
{ if(s.top()=='}')
    n++;
  else
    m++;
  s.pop();
}
if(a%2==0)
  { if(b%2==0)
      cout << a/2 + b/2;
    else cout << a/2 + b/2 + 1;
  }
else
  { if(b%2==0)
      cout << a/2 + b/2 + 1;
    else
      cout << a/2 + b/2 + 2;
  }
```