JUL 2024

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Stack

'24
25th Week
173-193

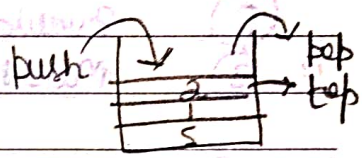
JUNE
21
FRIDAY

→ It is linear data structure that follows a particular order in which the operation are performed.

→ This strategy states that the element that is inserted last will come out first. LIFO / FILO

Basic operation

(a) isEmpty() : Return True if stack is empty else False.
(b) push(x) : insert an item to the top of the stack.
(c) pop() : Remove an item from the top.
(d) peek() : Return the top item.
(e) Size() : Return the size of stack.



In python ——— other language.

(a) append()
(b) pop()
(c) [-1] — peek index
(d) len(s) — size
(e) empty or not (a) if s: else not s: (b) if s == []:

→ Underflow :- when pop() or peek() called empty stack. [top != -1]
→ Overflow :- when push called on a full stack. [top == -1]

push() — O(1)
pop() — O(1)
isEmpty — O(1)
Size — O(1)
peek — O(1)

Types of stacks.

(a) Register stack = memory element present in the memory unit and can handle a small amount of data only.
(b) Memory Stack = It can handle large amount of memory data.

JUNE
22
SATURDAY
'24
25th Week
174-192

The pointer through which the element are accessed, inserted, and deleted in the Stack is called top of the Stack.

S	M	T	W	T	F	S
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

JUN 2024

Application of Stack

08 (1) Function Calls [Recursion]
(2) Balanced Parenthesis and backtracking
09 (3) Reversing items
(4) Infix to Postfix / Prefix
10 (5) Evaluation of Postfix / Prefix
(6) Stock Span Problem and its Variants.
11 (7) undo / Redo (or) forward / Backward. [ctrl z (or) ctr y]
(8) Evaluating Expression
12 (9) Matching HTML & XML

Reallife Application.

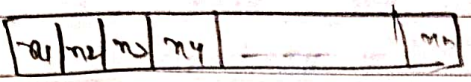
(1) CD/DVD Stand
02 (2) Stack of books in a book shop.
(3) History of web browser.
03 (4) Calls log, Emails, google photoes in any gallery.
(5) youtube download Notification.
04 (6) Pile of plates.

05 Implementations.

(a) using list
06 (2) using Collection. deque
(3) using queue. LIFO Queue [multithreaded environment uses]
07 (4) using our own implementation.

List implementation

push() — append()
pop() — pop()



```
stack = []
stack.append(10)   # [10]
stack.append(20)   # [10,20]
stack.append(30)   # [10,20,30]
print(stack.pop())  # [10,20]
top = stack[-1]  # to get top item.
print[top]
size = len(stack)
print(size)
```

TC = O(1)

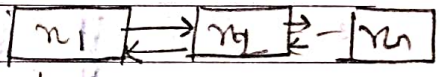
cache friendly

o/p = 30
20
2

Collection.deque implementation.

```
from collections import deque
stack = deque()      # deque([])
stack.append(10)     # deque([10])
stack.append(20)     # deque([10,20])
stack.append(30)     # deque([10,20,30])
print(stack.popl())  # deque([10,20])
top = stack[-1]
print(top)
size = len(stack)
print(size)
```

Not cache friendly



o/p — 30
20
2

TC = O(1)

Various function available in this module :—

- maxsize — No. of items allowed in the queue.
- empty() — return True if the queue is empty, False otherwise
- full() — return True if there are maxsize items in the queue.
- get() — remove & return an items from the queue.
- put() — put an item into the queue.
- qsize() — return the No of items in the queue.

JUNE
24
'24
MONDAY
26th Week
176-190

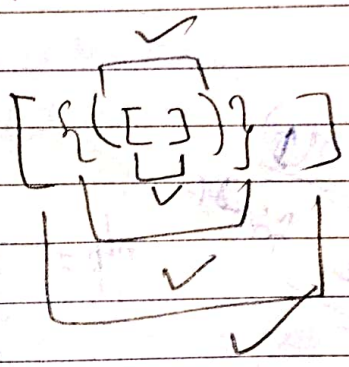
S	M	T	W	T	F	S
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

JUN 2024

Balanced Parentheses.

eg {[]{()}} — balanced.

eg [{}{}(] → Unbalanced.



M-1

```
def isbal(expr):
    stack = []
    for x in expr:
        if x in('(','{','['):
            stack.append(x)
        else:
            if not stack:
                return False
            elif ismatching(stack[-1],x)==False:
                return False
            else:
                stack.pop()
    if stack:
        return False
    else:
        return True

def ismatching(a,b):
    if (a=='(' and b==')') or
       (a=='{' and b=='}') or
       (a=='[' and b==']'):
        return True
    else
        return False
```

T.C = O(n)

M2

```
def check (my_string):
    brackets = ['()', '{}', '[]']
    while any( x in my_string for x in brackets):
        for br in brackets:
            my_string = my_string.replace( br, '')
    return not my_string.
string = "{[]{()}}";
print (string, "-", "balanced"
    if check (string) else 'unbalanced')
```

TC - O(n2)

M3

```
def check (exp):
    open_tup = tuple ('({[')
    close_tup = tuple (')}]')
    map = dict (zip (open_tup, close_tup))
    queue = []
    for i in exp:
        if i in open_tup:
            queue.append (map[i])
        elif i in close_tup:
            if not queue or i != queue.pop():
                return "unbalanced"
    if not queue:
        return 'balanced'
    else:
        return 'unbalanced'.

String = '{[]{()}}'
print (string, '-', check (string)) —— Balanced.
```

JUNE
26
'24
26th Week
178-188
WEDNESDAY

S	M	T	W	T	F	S
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

JUN 2024

best

```
def valpar(s):
    l = ['(', '{', '[']
    stack = []
    for i in s:
        if i in l:
            stack.append(i)
        else:
            if not stack:
                return False
            if i == ')' and stack[-1] == '(':
                stack.pop()
            elif i == ']' and stack[-1] == '[':
                stack.pop()
            elif i == '}' and stack[-1] == '{':
                stack.pop()
            else:
                return False
    return not stack
```

Advantages of stack

- Stacks cleanup object automatically.
- It is used in many virtual machine like PVM (or) JVM.
- Stack are the best for systematic memory management.
- Stack allows control over memory allocation & deallocation.
- Stacks are more secure and reliable as they do not get corruped easily.

Disadvantages of stack

- Stack memory is of limited size.
- The total size of stack must be defined before.
- Random accessing not possible
- Too many object are created then it can lead to stack overflow
- If stack falls outside the memory it can lead to abnormal termination.

Infix, Prefix & Postfix

08 **Infix:-** An operator is written in b/w two operands. eg A+B

09 **Prefix:** An operator is written before the operands. eg +AB

10 **Postfix:** An operator is written after the operands. eg AB+

eg	Infix	Prefix	Postfix
	A+B*C+D	++A*BCD	ABC*+D+
12	(A+B)*(C+D)	*+AB+CD	AB+CD+*
	A*B+C*D	+*AB*CD	AB*CD*+
01	A+B+C+D	+++ABCD	AB+C+D+

02 **Advantages**

① Do not requires Parenthesis, Precedence rules and associativity rules.

03 ② can be evaluated by writting a program that traverse the given expression exactly one.

	Associativity
^	Right to left
*, /	left to right
+, -	left to right

Precedence ↑

04

Infix to Postfix

05

i/p: "a+b*c" i/p: "(a+b)*c"

06 o/p: "bc*a+" o/p: "ab+c*"

07 i/p: "a^b^c" i/p: "(a+b)*(c+d)"

o/p: "abc^^" o/p: ab+cd+*

Naive

① First fully Parenthesis the given expression.

② After in which that expression are going to be evaluated so convert that into same order of its Postfix form.

JUNE
28
'24
26th Week
180-186
FRIDAY

S	M	T	W	T	F	S	
30						1	JUN 2024
2	3	4	5	6	7	8	
9	10	11	12	13	14	15	
16	17	18	19	20	21	22	
23	24	25	26	27	28	29	

(i) a+b*c ⇒ (a+(b*c)) ⇒ (a+(bc*)) ⇒ (abc*+)

(ii) (a+b)*c ⇒ ((a+b)*c) ⇒ ((ab+)*c) ⇒ ab+c*

(iii) (a+b)*(c+d) ⇒ ((a+b)*(c+d)) ⇒ ((ab+)*(cd+)) ⇒ ab+cd+*

Algorithm

1. Create an empty stack, st.
2. Do following for every character x from left to right.
3. If x is
   - (a) operand : output it.
   - (b) left Parenthesis : push to st.
   - (c) right Parenthesis : Pop from st until left Parenthesis is found. output the popped operator.
   - (d) operator : If st is empty, push x to st. else compare with st top
     - (i) Higher Precedence (than st top), push to st.
     - (ii) lower Precedence, pop st top and output until a higher precedence operator is found. Then push x to st.
     - (iii) Equal Precedence, use associativity.
4. pop and output everything from st.

operand आया print कर दो और operator के लिए ऊपर बताए rules follow करो।

x = a+b*c

Symbol	Stack	Result
a		a
+	+	a
b	+	ab
*	* +	ab
c	* +	abc
	+	abc*
		abc*+

→ For Code See ChatGPT.

Evaluation of Postfix

i/p: 102*3+ [10*2+3 ⇒ ((10×2)+3) ⇒ ((10 2×)+3)
o/p: 23 [102×3+]

i/p: 102+3* [(10+2)×3 ⇒ ((10+2)×3) ⇒ ((10 2+)×3)
o/p: 36 ⇒ [10 2 + 3*]

i/p: 10 2 3 ^1
o/p: 100000000

→ Postfix expression don't required Precedence & associativity even brackets.
→ And postfix expression is evaluated in Traversal

Algorithms

i/p: 10 2*3 5*+9-
[Infix: (10×2)+(3×5)-9]

1. Create an empty Stack st.
2. Traverse through every symbol x of given Postfix.
   a. If x is an operand, push to st.
   b. else (x is an operator)
      i. op1 = st.pop();
      ii. op2 = st.pop();
      iii. Compute op2(x)op1 and push the result to st.
3. Return st.top()

→ For code see chatGPT.

Input Symbol(x)	Stack(st)
10	10
2	2 / 10
*	20
3	3 / 20
5	5 / 3 / 20
*	15 / 20
+	35
9	9 / 35
-	26

JUNE
30
'24
27th Week
182-184
SUNDAY

30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

2024 JUN

Evaluation of Prefix

i/p: +*10 23 [10*2+3 ⇒ ((10*2)+3) ⇒ ((*102)+3) ⇒ +*10 23

o/p: 23

i/p: *+10 23 [(10+2)*3 ⇒ ((10+2)*3) ⇒ ((+102)*3) ⇒ [*+10 23]

o/p: 36

Prefix & Postfix expression can be evaluated faster than an infix expression. This is because we don't need to process any bracket or follow operator precedence rules. In postfix and prefix expression whichever operator comes before will be evaluated first, irrespective of its priority.
Also there is no bracket in these expression.

Algorithm

(I) Put a pointer P at the end of the end.
(II) If character at P is an operand push it to stack.
(III) If the character at P is an operator pop two elements from the stack. operate on these element according to the operator, and push the result back to the stack.
(IV) Decrement P by 1 and go to step 2 as long as there are characters left to be scanned in the expression.
(V) The result is stored at the top of stack, return it.

Code See ChatGPT.

Infix to Prefix Conversion

operators	Associativity
^	Right to left
* /	left to right
+ -	left to right
	(Precedence)

i/p: infix = x+y*z
o/p: Prefix = +x*yz

i/p: (x+y)*z
o/p: *+xyz

i/p: x^y^z
o/p: ^x^yz

i/p: (x+y)*(z+w)
o/p: (*+xy+zw)

Two steps

1. Fully parenthesis.
2. Start converting from innermost to outermost.

(i) x+y*z ⇒ (x+(y*z)) ⇒ (x+(*yz)) ⇒ +x*yz.

(ii) (x+y)*z ⇒ ((x+y)*z) ⇒ ((+xy)*z) ⇒ *+xyz.

(iii) x^y^z ⇒ (x^(y^z)) ⇒ (x^(^yz)) ⇒ ^x^yz.

(iv) x+y*(z-w) ⇒ (x+(y*(z-w))) ⇒ (x+(y*(-zw)))
⇒ (x+(*y-zw)) ⇒ (+x*y-zw)

eg i/p: x+y*z

Input Symbol	Stack	Prefix (reverse)
z	⌊⌋	z
*	|*|	z
y	|*|	zy
+	|+|	zy*
x	|+|	zy*x
.	⌊⌋	zy*x+

Prefix = Reverse of "zy*x+" → (+x*zy)

JULY
02
'24
27th Week
184-182
TUESDAY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Algorithms

$TC = \Theta(n)$
$SC = O(n)$

1. Create an empty stack, st.
2. Create an empty string, Prefix.
3. Do following for every character C from right to left.
4. If C is:
   - (a) operand: Push it to Prefix.
   - (b) right parenthesis: Push to st.
   - (c) Left parenthesis: Pop from st until right parenthesis is found. Append the popped character to prefix.
   - (d) operator: If st is empty, Push c to st.
     else compare with st top.
     - (i) Higher Precedence (than st top): push c to st.
     - (ii) lower Precedence: pop st top and append the popped item to prefix until a higher precedence operator is found (or st become empty). push C to st.
     - (iii) Equal Precedence: use Associativity.
5. Pop everything from st and append to prefix.
6. Return reverse of Prefix.

For Code see ChatGPT.

Previous Greater Element

→ Given an array of distinct integers, find the closest (positive wise) greater on left of every element. If there is not greater element on left then Print -1.

i/p: arr[] = [15, 10, 18, 12, 4, 6, 2, 18]
o/p: -1 15 -1 18 12 12 6 12

i/p: arr[] = [8 10 12]
o/p: -1 -1 -1

Naive

```
def PrevGreat(arr):
    for i in range(len(arr)):
        Pg = -1
        for j in range(i-1, -1, -1):
            if arr[j] > arr[i]:
                Pg = arr[j]
                break
        print(Pg, end=" ")
```

TC = O(n²)
SC = O(1)

Efficient

```
① def PrevGreater(arr):
② st = []
③ for i in range(len(arr)):
④ while len(st)>0 and
        st[-1] <= arr[i]:
⑤ {O(n) O(n)}  st.pop()
⑥ Pg = -1 if (len(st)==0)
        else st[-1]
⑦ Print(pg, end=" ")
⑧ st.append(arr[i])
```

Next Greater Element

→ The next greater element for an element x is the first greater element on the right side of x in the array. Element for which no greater element exist, consider the next greater element as -1.

i/p: arr[] = [5, 15, 10, 8, 6, 12, 7]
o/p: = 15 -1 12 12 12 -1 -1

i/p: arr[] = [10, 15, 20, 25]
o/p: [15, 20, 25, -1]

Naive

```
from collections import deque
def PrintGreater(arr):
    for i in range(len(arr)):
        ng = -1
        for j in range(i+1, len(arr)):
            if arr[j] > arr[i]:
                ng = arr[j]
                break
        print(ng, end=" ")
```

O(n²)

Efficient

```
from collection import deque
def PrintGreater(arr):
    st = [] n = len(arr)
    res = [None] * len(arr)
    for i in range(n-1, -1, -1):
        while len(st)>0 and st[-1]
        <= arr[i]: st.pop()
        res[i] = -1 if len(st)<=0
        else st[-1]
        st.append(arr[i])
    for x in res:
        print(x, end=" ")
```

O(n)

JULY
04 '24
27th Week
186-180
THURSDAY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

Design a stack that support getMin() :—

Design a Data structure special stack that support all the stack operation like Push(), pop(), isEmpty(), isFull() and an additional operation getmin() which should return minimum element from the special stack. All these operation of special stack must have a time & space complexity of O(1).

i/p: push(20), push(10), getMin(), push(5), getMin(), pop(), getMin(), pop(), getMin()
o/p: 10 5 10 20

i/p: push(5), push(4), push(3), getMin(), pop(), getmin(), push(2), getMin()
o/p: 3 4 2

push(n)
ms.push(n);
if (as.top() >= ms.top())
as.push(n)

pop()
if (ms.top() == as.top())
as.pop()
ms.pop()

Largest Rectangle with all 1's.

i/p: mat = [[0, 1, 1, 0],
[1, 1, 1, 1],
[1, 1, 1, 1],
[1, 1, 0, 0]]

o/p: 8

i/p: mat = [[0, 0],
[0, 0]]

o/p: 0.

Naive
→ Consider Every cell as a starting point.
→ Consider all sizes of rectangles with current cell as a starting point.
→ For the current rectangle, check if it has all 1's. If yes then update the res if the current size is more.

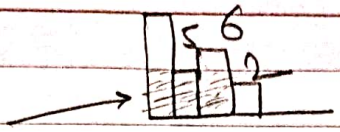
TC: $O(R^3 * C^3)$

AUG 2024

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

JULY '24
27th Week 187-179
05 FRIDAY

i/p: arr = [10, 5, 6, 2]
o/p: 15



Recap of the largest rectangular area Prob

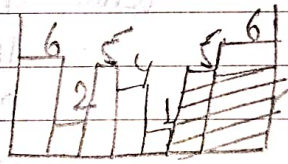
Run a loop from 0 to R-1
(i) Update the histogram for the current row.
(ii) Find the largest area in the histogram and update the result if required.

```
def maxRectangle(mat):
    res = largestHist(mat[0])
    for i in range(1, len(mat)):
        for j in range(len(mat[i])):
            if mat[i][j]:
                mat[i][j] += mat[i-1][j]
        res = max(res, largestHist(mat[i]))
    return res
```

TC = O(R×C)

Largest Area in a Histogram: — (Rectangular)

i/p: arr = [6, 2, 5, 4, 1, 5, 6]
o/p: 10



consider

Naive

```
def getmaxArea(arr):
    res = 0
    for i in range(n):
        curr = arr[i]
        for j in range(i-1, -1, -1):
            if arr[j] >= arr[i]:
                curr += arr[i]
            else: break
        for j in range(i+1, n):
            if arr[j] >= arr[i]:
                curr += arr[i]
            else: break
        res = max(res, curr)
    return res
```

TC = O(n²)
SC = O(1)

arr = [6, 2, 5, 4, 1, 5, 6]
initially: res = 0

i = 0: curr = 6, res = 6
i = 1: curr = 8, res = 8
i = 2: curr = 5, "
i = 3: curr = 8, "
i = 4: curr = 7, "
i = 5: curr = 10, res = 10
i = 6: curr = 6,

Don't count your chickens before they are hatched

JULY
06
'24
SATURDAY
27th Week
188-178

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

Efficient Solution:-

(1) Initialize : res = 0
(2) Find previous smaller element for every element.
(3) Find next smaller element for every element.
(4) Do following for every element arr[i]

```
curr = arr[i];
curr += (i - ps[i] - 1) * arr[i]
curr += (ns[i] - i - 1) * arr[i]
res = max(res, curr);
```

(5) return res

[C++ O(n)]

Efficient

```
def getmaxArea(arr):
    st = []
    res = 0
    for i in range(len(arr)):
        while st and arr[st[-1]] >= arr[i]:
            tp = st[-1]
            st.pop()
            curr_width = (i - st[-1] - 1) if st else i
            res = max(res, curr_width * arr[tp])
        st.append(i)
    while st:
        tp = st[-1]
        st.pop()
        curr_width = (len(arr) - st[-1] - 1) if st else len(arr)
        res = max(res, curr_width * arr[tp])
    return res
```

Obs)

Design a stack with getmin() in O(1) space :-

1st method (Assuming all element Positive)

Push()
pop()
peek()
getmin()
} O(1) [TC]
O(1) [SC]

We assume all elements are positive when you see a greater element than current-minimum, simply push it. We see smaller element than current-minimum then you need to store the difference & update the minimum.

```
Void push (int x)
    if (s.isEmpty())
        min = x
        s.push(x)
    elseif (x <= min)
        s.push(x - min)
        min = x
    else :
        s.push(x)
```

```
int getMin()
    return min
int pop()
    t = s.pop()
    if (t <= 0)
        res = min
        min = min - t
        return res
    else : return t.
```

```
int Peek()
    t = s.peek()
    return (t <= 0) ?
        min : t.
```

2nd method [Handle Negative Case]

```
Void push (int x)
    if (s.isEmpty())
        min = x
        s.push(x)
    elseif (x <= min)
        s.push(2*x - min)
        min = x
    else :
        s.push(x)
```

```
int getMin()
    return min
int peek()
    t = s.peek()
    return (t <= min) ? min : t
```

```
int pop()
    t = s.pop()
    if (t <= min)
        res = min
        min = 2*min - t
        return res
    else
        return t.
```

JULY
08
'24
28th Week
190-176
MONDAY

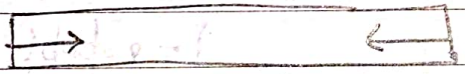
S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

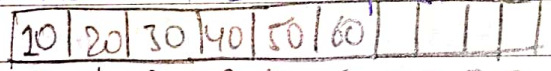
JUL 2024

Implement two stacks in array:—

→ Create a data structure twoStacks that represent two stack.
→ Implementation of twoStack should use only one array, both stacks should use the same array for storing elements.
→ push1(int x) – pushes x to first stack, push2(int x) – push x to 2nd stacks
→ pop1() – pop from 1st stack, pop2() – pop from 2nd stack.

Naive
→ We divide the array from middle, use 1st half for stack 1 and 2nd half for stack 2.
→ Inefficient use of space.
If we add 5 items to stack 1, and no items to stack 2, then we can't add any more items to stack 1, even if we have space in the array.

Efficient
→ 
→ Begins both stacks from the two corners of the array.
→ now we can insert items in any stack as long as we have space.



top1 = 5
top2 = 10

```
Class TwoStacks:
    def __init__(self,n):
        self.size = n
        self.arr = [None]*n
        self.top1 = -1
        self.top2 = self.size

    def push1(self, n):
        if self.top1 < self.top2 - 1:
            self.top1 = self.top1 + 1
            self.arr[self.top1] = n
            return True
        return False

    def push2(self, n):
        if self.top1 < self.top2 - 1:
            self.top2 = self.top - 1
            self[arr.top2] = n
            return True
        return False

    def size1(self):
        return self.top1 + 1

    def size2(self):
        return self.size - self.top2
```

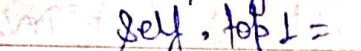
TC – O(1)

```
def pop1(self):
    if self.top1 >= 0:
        x = self.arr[self.top1]
        self.top1 = self.top1 - 1
        return x
    return None
```

TC = O(1)

```
def pop2(self):
    if self.top2 < self.size:
        x = self.arr[self.top2]
        self.top2 = self.top2 + 1
        return x
    return None
```

ts = TwoStack(5)

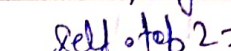


top1 = -1 top2 = 5

ts.push1(10)

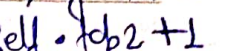
top1 = 0, top2 = 5

ts.push2(20)



top1 = 0 top2 = 4

ts.pop1()



top1 = -1 top2 = 4

Implement K stacks in an Array.

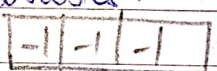
Create a data structure KStack that represent K stacks. Implementation of KStack should use only one array. K stacks should use the same array for storing elements.

We maintain two arrays and an extra variables.

arr =

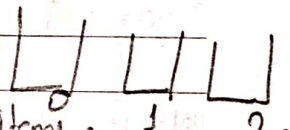
Given array n = 6

top =

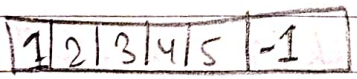


to maintain index of top items.

-1 means empty



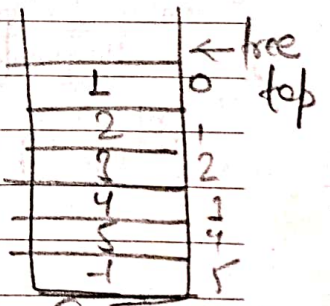
next =



to maintain index of next item (or item just below) in the stacks.

free_top = 0

to maintain top of free stacks (stack to maintain free slots)



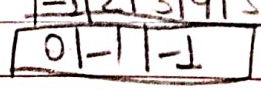
Initial State

Push (0, 10)

arr

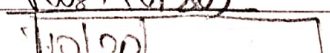
next

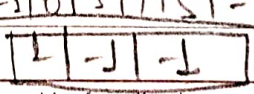
top



free_top = -1

Push (0, 20)





free_top = 2

JULY
10 '24
28th Week
192-174
WEDNESDAY

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUL 2024

Class KStacks:

```
08  def __init__(self, n, k):
        self.cap = n
09      self.k = k
        self.arr = [None]*n
10      self.top = [-1] * k
        self.next = [i+1 for i in range(n)]
11      self.next[n-1] = -1
        self.free_top = 0
12
```

```
def push(self, sn, x):
01      i = self.free_top
        self.free_top = self.next[self.free_top]
02      self.arr[i] = x
        self.next[i] = self.top[sn]
03      self.top[sn] = i
```

```
def pop(self, sn):
    prev_top = self.top[sn]
    self.top[sn] = next[prev_top]
    self.next[prev_top] = self.free_top
    self.free_top = prev_top
    return self.arr[prev_top]
```

```
def isEmpty(self, sn):
    return self.top[sn] == -1
```

04 arr = [| | | | | |], top = [-1 | -1], free_top = 0, next = [1 | 2 | 3 | 4 | -1 | 6]
n = 6, k = 2, 0 1 2 3 4 5

05 push(1, 100):
arr = [100 | | | | | |], top = [-1 | 0], free_top = 1, next = [-1 | 2 | 3 | 4 | -1 | 6]

06 push(1, 200):
arr = [100 | 200 | | | | |], top = [-1 | 1], free_top = 2, next = [-1 | 0 | 3 | 4 | -1 | 6]

07 pop(1):
arr = [100 | | | | | |], top = [-1 | 0], free_top = 1, next = [-1 | 0 | 3 | 4 | -1 | 6]

Stock Span Problem

→ We are Given array of integers. this array represents Prices of Stock on n-consecutive days. our task is to find out Span of Stock on every day.

→ Span on a day is number of consecutive days including the current day and days before it which have value equal to (or) smaller. [current item is always included]

i/p : arr[] = [13, 15, 12, 14, 16, 8, 6, 4, 10, 30]
o/p : 1 2 1 2 5 1 1 1 4 10

i/p : arr[] = {10, 20, 30, 40}
o/p : 1 2 3 4

i/p : arr = [30, 20, 25, 28, 27, 29]
o/p : 1 1 2 3 1 5

Naive solution

```
def PrintSpan(arr):
    for i in range(len(arr)):
        Span = 1
        J = i-1
        while J>=0 and arr[i] >= arr[J]:
            Span += 1
            J -= 1
        Print(Span, end = " ")
```

O(n2)
O(1)

arr = [18, 12, 13, 14]
i=0 : Span = 1
i=1 : Span = 1
i=2 : Span = 2
i=3 : Span = 3

Efficient solution

0	1	2	3	4	5	6	7	8
[60	10	20	40	35	30	50	70	65]
1	(1-0)	(2-0)	(3-0)	(4-3)	(5-4)	(6-0)	(7+1)	

Span = { (Index of current Element) − (Index of closest greater element on left side) } If there is a greater element on left side.
Index of current element + 1 } otherwise

```
def PrintSpan(arr):
    St = []
    St.append(0)
    Print(1, end = " ")
    for i in range(1,n):
        while (len(St) >=0 and arr[St[-1]]) <= arr[i]:
            St.pop()
        Span = (i+1) if len(St)==0 else i-St[-1]
        Print(Span, end = " ")
        St.append(i)
```

TC = O(n)