# Stack MCQs

Four Corners

GATE | GATE CS 1996 | Question 12

Consider the following statements:

i. First-in-first out types of computations are efficiently supported by STACKS.

ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.

iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.

iv. Last-in-first-out type of computations are efficiently supported by QUEUES.

Which of the following is correct?

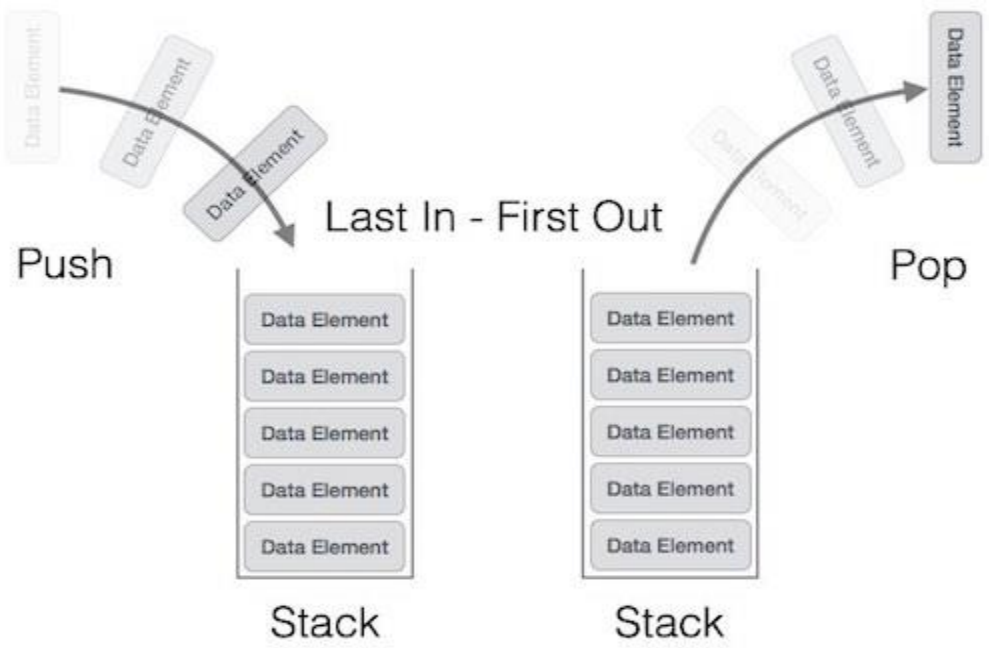
**(A)** (ii) and (iii) are true
**(B)** (i) and (ii) are true
**(C)** (iii) and (iv) are true
**(D)** (ii) and (iv) are true

# Queue

- Our software queues have counterparts in real world queues.
- We wait in
  - queues to buy pizza,
  - to enter movie theaters,
  - to drive on a turnpike, and
  - to ride on a roller coaster.

# Stacks

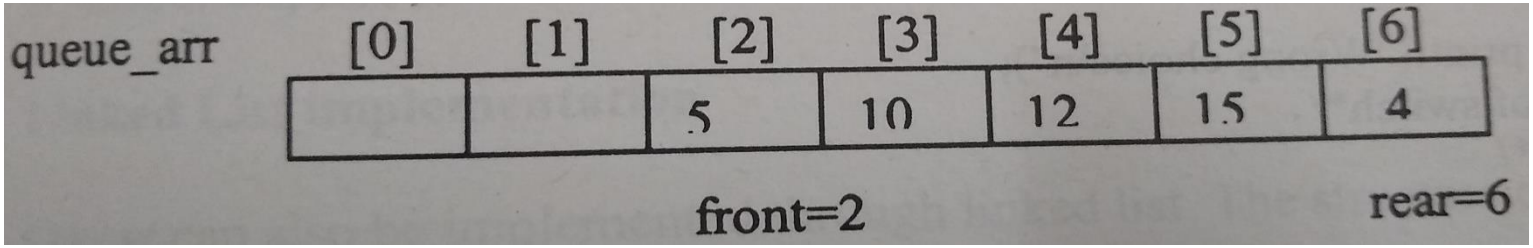
- Stack is a linear data structure which follows a particular order in which the operations are performed.
- The order may be LIFO(Last In First Out) or FILO(First In Last Out).



# Problem-

- A situation arises when
  - rear is at the last position of array and
  - front is not at the $0^{th}$ position.

- But we cannot add element any element in queue because rear is at the n-1th position.




- There are 2 spaces for adding elements in queue but
  - we cannot add any element in queue because rear is at the last position of array

GATE | GATE CS 1996 | Question 12

Consider the following statements:

i. First-in-first out types of computations are efficiently supported by STACKS.

ii. Implementing LISTS on linked lists is more efficient than implementing LISTS on an array for almost all the basic LIST operations.

iii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.

iv. Last-in-first-out type of computations are efficiently supported by QUEUES. Which of the following is correct?

**(A)** (ii) and (iii) are true
**(B)** (i) and (ii) are true
**(C)** (iii) and (iv) are true
**(D)** (ii) and (iv) are true

**Answer: (A)**

**Explanation:** i -STACK is the data structure that follows Last In First Out (LIFO) or First In Last Out (FILO) order, in which the element which is inserted at last is removed out first.

ii – Implementing LISTS on linked lists is more efficient than implementing it on an array for almost all the basic LIST operations because the insertion and deletion of elements can be done in O(1) in Linked List but it takes O(N) time in Arrays.

iii- Implementing QUEUES on a circular array is more efficient than implementing it on a linear array with two indices because using circular arrays, it takes less space and can reuse it again.

iv – QUEUE is the data structure that follows First In First Out (FIFO) or Last In Last Out (LILO) order, in which the element which is inserted first is removed first.

only (ii) and (iii) are TRUE.
Option (A) is correct.

**GATE | GATE-CS-2002 | Question 44**

To evaluate an expression without any embedded function calls:

**(A)** One stack is enough

**(B)** Two stacks are needed

**(C)** As many stacks as the height of the expression tree are needed

**(D)** A Turing machine is needed in the general case

**GATE | GATE-CS-2002 | Question 44**

To evaluate an expression without any embedded function calls:
**(A)** One stack is enough
**(B)** Two stacks are needed
**(C)** As many stacks as the height of the expression tree are needed
**(D)** A Turing machine is needed in the general case

**Answer: (A)**

**Explanation:**
Any expression can be converted into Postfix or Prefix form.
Prefix and postfix evaluation can be done using a single stack.

**Data Structures | Stack | Question 8** (GATE CS 2004)

A single array A[1..MAXSIZE] is used to implement two stacks. The two stacks grow from opposite ends of the array. Variables top1 and top2 (topl< top 2) point to the location of the topmost element in each of the stacks. If the space is to be used efficiently, the condition for “stack full” is

**(A)** (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)
**(B)** top1 + top2 = MAXSIZE
**(C)** (top1= MAXSIZE/2) or (top2 = MAXSIZE)
**(D)** top1= top2 -1

**Data Structures | Stack | Question 8** (GATE CS 2004)

**(A)** (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)
**(B)** top1 + top2 = MAXSIZE
**(C)** (top1= MAXSIZE/2) or (top2 = MAXSIZE)
**(D)** top1= top2 -1

| 1 | 2 | 3 | 4 | 5 | 6 | MAXSIZE=7 |
|---|---|---|---|---|---|---|
| top1 | | | | | | top2 |

| 1 | 2 | 3 | 4 | 5 | 6 | MAXSIZE=7 |
|---|---|---|---|---|---|---|
| | | | top1 | top2 | | |

Option c)

top1+top2=MAXSIZE

Here Stack is full, top1=4, top2=5, top1+top2=9 Not Equal to MAXSIZE, thus FALSE

Option d)

top1=top2-1, TRUE in every case

**Data Structures | Stack | Question 8** (GATE CS 2004)

**(A)** (top1 = MAXSIZE/2) and (top2 = MAXSIZE/2+1)
**(B)** top1 + top2 = MAXSIZE
**(C)** (top1= MAXSIZE/2) or (top2 = MAXSIZE)
**(D)** top1= top2 -1

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | MAXSIZE=8 |
|---|---|---|---|---|---|---|---|
| top1 | | | | | | | top2 |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | MAXSIZE=8 |
|---|---|---|---|---|---|---|---|
| | | | top1 | top2 | | | |

Option a)

If more elements were inserted in Stack1 and less elements were inserted in Stack2

top1=MAXSIZE/2=8/2=4

top2=MAXSIZE/2+1=8/2+1 =5

But here top1 and top2 will have constant values, FALSE

For Option c)

top1=MAXSIZE/2=8/2=4

top2=MAXSIZE=8, FALSE

GATE | Gate IT 2007 | Question 32

Consider the following C program:

```
#include
        #define EOF -1
        void push (int); /* push the argument on the stack */
        int pop  (void); /* pop the top of the stack */
        void flagError ();
        int main ()
       {        int c, m, n, r;
                 while ((c = getchar ()) != EOF)
                { if   (isdigit (c) )
                            push (c);
                  else if ((c == '+') || (c == '*'))
                 {          m = pop ();
                            n = pop ();
                            r = (c == '+') ? n + m : n*m;
                            push (r);
                   }
                   else if (c != ' ')
                            flagError ();
          }
           printf("% c", pop ());
```

What is the output of the program for the following input ?

5 2 * 3 3 2 + * +

**(A)** 15

**(B)** 25

**(C)** 30

**(D)** 150

```
#include
        #define EOF -1
        void push (int); /* push the argument on the stack */
        int pop  (void); /* pop the top of the stack */
        void flagError ();
        int main ()
       {        int c, m, n, r;
                 while ((c = getchar ()) != EOF)
                { if   (isdigit (c) )
                            push (c);
                  else if ((c == '+') || (c == '*'))
                 {          m = pop ();
                            n = pop ();
                            r = (c == '+') ? n + m : n*m;
                            push (r);
                   }
                   else if (c != ' ')
                            flagError ();
          }
           printf("% c", pop ());
```

```
#include
        #define EOF -1
        void push (int); /* push the argument on the stack */
        int pop  (void); /* pop the top of the stack */
        void flagError ();
        int main ()
       {        int c, m, n, r;
                 while ((c = getchar ()) != EOF)
                { if  (isdigit (c) )
                          push (c);
                 else if ((c == '+') || (c == '*'))
                {         m = pop ();
                          n = pop ();
                          r = (c == '+') ? n + m : n*m;
                          push (r);
                  }
                  else if (c != ' ')
                          flagError ();
          }
           printf("% c", pop ());
```

**Explanation:**

- The function of the program is:-

1) If the current character is a digit it pushes into stack

2) Else if the current character is operator + or *, it pops two elements and then performs the operation.

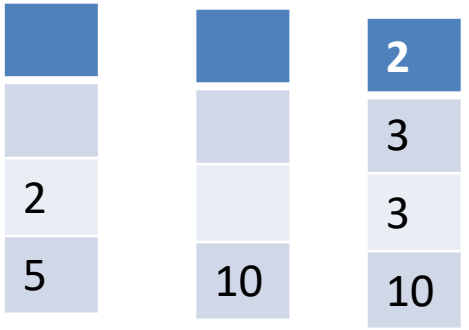
3) Finally it pushes the resultant element into stack.

```
#include
    #define EOF -1
    void push (int); /* push the argument on the stack */
    int pop  (void); /* pop the top of the stack */
    void flagError ();
    int main ()
{       int c, m, n, r;
        while ((c = getchar ()) != EOF)
        { if  (isdigit (c) )
                push (c);
         else if ((c == '+') || (c == '*'))
        {       m = pop ();
                n = pop ();
                r = (c == '+') ? n + m : n*m;
                push (r);
          }
          else if (c != ' ')
                flagError ();
  }
   printf("% c", pop ());
```

**Explanation:**

Initially stack s is empty.

Scan 5 2 * 3 3 2 + * +

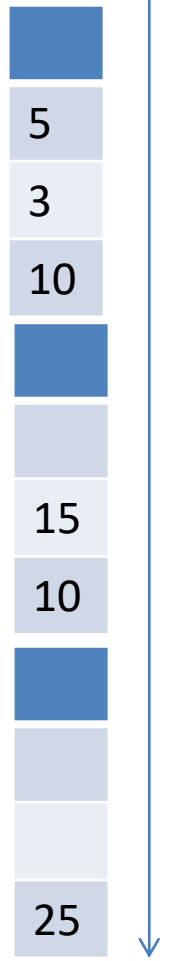



1) 5 -> Push
2) 2 -> Push
3) * -> It pops two elements m = 2, n=5

n*m =5*2= 10 ,

Push 10

4) 3 -> Push
5) 3 -> Push
6) 2 -> Push
7) + -> Pop m=2, Pop n=3, n+m=3+2=5, Push 5 into s
8) * -> m=5, n=3 n*m=5*3=15 Push 15 into s
9) + -> m=15, n=10 n+m =10+15=25 Push 25




Finally the result value is the only element present in stack.

GATE | GATE-CS-2015 (Set 3) | Question 65

The result evaluating the postfix expression (10) 5 + (60) 6 / * 8 – is

**(A)** 284
**(B)** 213
**(C)** 142
**(D)** 71

GATE | GATE-CS-2015 (Set 3) | Question 65

The result evaluating the postfix expression 10 5 + 60 6 / * 8 – is

**(A)** 284

**(B)** 213

**(C)** 142

**(D)** 71

| Stack | Stack | Stack | Stack | Stack | Stack |
|---|---|---|---|---|---|
| | | | | | |
| | | 6 | | | |
| 5 | | 60 | 10 | | 8 |
| 10 | 15 | 15 | 15 | 150 | 150 |

**Answer: (C)**

**Push 10,5**

**On Scanning +, 10 + 5=15, Push 15, 60, 6**

**On Scanning /, 60/6=10, Push 10**

**On Scanning *, 15*10=150, Push 150, 8**

**On Scanning -, 150-8=142 (Ans)**

**ISRO | ISRO CS 2017 – May | Question 57**

The best data structure to check whether an arithmetic expression has balanced parenthesis is a

**(A)** Queue

**(B)** Stack

**(C)** Tree

**(D)** List

**ISRO | ISRO CS 2017 – May | Question 57**

The best data structure to check whether an arithmetic expression has balanced parenthesis is a

**(A)** Queue
**(B)** Stack
**(C)** Tree
**(D)** List

**Answer: (B)**

**Explanation:** Stacks can check equal pair/ balanced pair of parenthesis efficiently. Whenever we get an opening parenthesis we can push it on the stack and when we get the corresponding closing parenthesis, we can pop it. After performing all push and pop operations, if at the end of the expression stack becomes empty then the expression has a balanced parenthesis.

ISRO | ISRO CS 2014 | Question 43

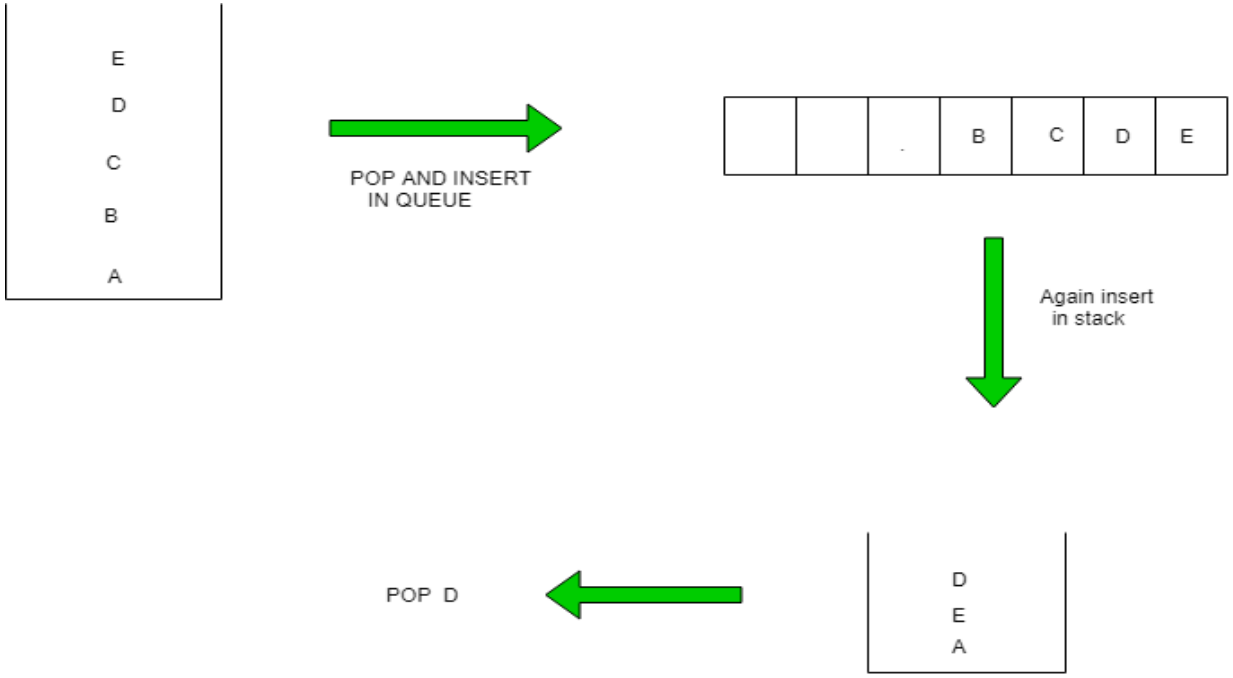
The five items: A, B, C, D, and E are pushed in a stack, one after other starting from A. The stack is popped four items and each element is inserted in a queue. The two elements are deleted from the queue and pushed back on the stack. Now one item is popped from the stack. The popped item is

**(A)** A
**(B)** B
**(C)** C
**(D)** D

ISRO | ISRO CS 2014 | Question 43

The five items: A, B, C, D, and E are pushed in a stack, one after other starting from A. The stack is popped four items and each element is inserted in a queue. The two elements are deleted from the queue and pushed back on the stack. Now one item is popped from the stack. The popped item is

**(A)** A
**(B)** B
**(C)** C
**(D)** D




**Answer: (D)**

# ISRO | ISRO CS 2015 | Question 11

If the sequence of operations – push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop are performed on a stack, the sequence of popped out values

**(A)** 2,2,1,1,2

**(B)** 2,2,1,2,2

**(C)** 2,1,2,2,1

**(D)** 2,1,2,2,2

## ISRO | ISRO CS 2015 | Question 11

If the sequence of operations – push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop are performed on a stack, the sequence of popped out values

**(A)** 2,2,1,1,2
**(B)** 2,2,1,2,2
**(C)** 2,1,2,2,1
**(D)** 2,1,2,2,2

| Operation | Stack | Pop Sequence |
|---|---|---|
| Push 1 | 1 | |
| Push 2 | 1, 2 | |
| Pop | 1 | 2 |
| Push 1 | 1, 1 | |
| Push 2 | 1, 1, 2 | |
| Pop | 1, 1 | 2, 2 |
| Pop | 1 | 2, 2, 1 |
| Pop | Empty | 2, 2, 1, 1 |
| Push 2 | 2 | |
| Pop | Empty | 2, 2, 1, 1, 2 |

**Answer: (A)**

**Explanation:** The pop sequence can be seen from the following table:

**GATE 1994**

Which of the following permutation can be obtained in the output( in the same order) using a stack assuming that input is the sequence 1,2,3,4,5 in that order?

**(A) 3,4,5,1,2**

**(B) 3,4,5,2,1**

**(C) 1,5,2,3,4**

**(D) 5,4,3,1,2**

**GATE 1994**

Which of the following permutation can be obtained in the output( in the same order) using a stack assuming that input is the sequence 1,2,3,4,5 in that order?

**(A) 3,4,5,1,2**

**(B) 3,4,5,2,1**

**(C) 1,5,2,3,4**

**(D) 5,4,3,1,2**

**Ans B)**

**ISRO | ISRO CS 2017 | Question 47**

Which of the following permutation can be obtained in the same order using a stack assuming that input is the sequence 5, 6, 7, 8, 9 in that order?

**(A)** 7, 8, 9, 5, 6

**(B)** 5, 9, 6, 7, 8

**(C)** 7, 8, 9, 6, 5

**(D)** 9, 8, 7, 5, 6

**ISRO | ISRO CS 2017 | Question 47**

Which of the following permutation can be obtained in the same order using a stack assuming that input is the sequence 5, 6, 7, 8, 9 in that order?

**(A)** 7, 8, 9, 5, 6
**(B)** 5, 9, 6, 7, 8
**(C)** 7, 8, 9, 6, 5
**(D)** 9, 8, 7, 5, 6

**Explanation:**

Lets try operations as-

Push 5
Push 6
Push 7
Push 8
Push 9
Pop All

The sequence will be 9,8,7,6,5

Option D is FALSE, 5 cannot be popped before 6

**Explanation:**

Option B is FALSE,

9 cannot be popped before 6

PUSH 5,6

POP both

Sequence=6,5

PUSH 7 etc

**ISRO | ISRO CS 2017 | Question 47**

**(A)** 7, 8, 9, 5, 6
**(B)** 5, 9, 6, 7, 8
**(C)** 7, 8, 9, 6, 5
**(D)** 9, 8, 7, 5, 6

**Answer: (C)**
**Explanation:** The sequence given in option (C) is one of the only possible sequence which can be obtained.
We can obtain the sequence by performing operations in the manner:
Push 5
Push 6
Push 7
Pop 7
Push 8
Pop 8
Push 9
Pop 9
Pop 6
Pop 5.
hence the sequence will be 7,8,9,6,5.

**ISRO | ISRO CS 2017 | Question 47**

**(A)** 7, 8, 9, 5, 6
**(B)** 5, 9, 6, 7, 8
**(C)** 7, 8, 9, 6, 5
**(D)** 9, 8, 7, 5, 6

**Answer: (C)**
**Explanation:**

5 cannot be popped before 6

Option A) is FALSE

ISRO | ISRO CS 2008 | Question 67

Stack A has the entries a, b, c (with a on top). Stack B is empty. An entry popped out of stack A can be printed immediately or pushed to stack B.

An entry popped out of the stack B can be only be printed.

In this arrangement, which of the following permutations of a, b, c are not possible?

(A) b a c

(B) b c a

(C) c a b

(D) a b c

ISRO | ISRO CS 2008 | Question 67

Stack A has the entries a, b, c (with a on top). Stack B is empty. An entry popped out of stack A can be printed immediately or pushed to stack B. An entry popped out of the stack B can be only be printed. In this arrangement, which of the following permutations of a, b, c are not possible?

(A) b a c

(B) b c a

(C) c a b

(D) a b c

Answer: (C)

Option d)

Pop three times from Stack A and print

Sequence=a b c

Possible

Explanation:

Option (A):

Pop a from stack A

Push a to stack B

Print b from Stack A

Print a from stack B

Print c from stack A

Order = b a c

Option (B):

Pop a from stack A& Push to stack B

Print b from stack A

Print c from stack A

Print a from stack A

Order = b c a

Option (C):

Pop a from stack A & Push to stack B

Pop b from stack A & Push to stack B

Print c from stack A

Now, printing a will not be possible.

GATE | Gate IT 2005 | Question 13

A function f defined on stacks of integers satisfies the following properties. f(Ø) = 0 and f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

**(A)** 6
**(B)** 4
**(C)** 3
**(D)** 2

- *i*: Element to be pushed
- Initial State *f*(∅)=0.
- For Empty Stack *f*(*S*) is 0
- Then we push each element (*i*)one by one and calculate *f*(*s*) for each insertion as given
- *fnew*(*S*)=max(*fprevious*(*S*),0)+*i*, the function to compute *f*(*S*) for each insertion

- INSERT 2 on to Stack
  *fprevious*(*S*)=0 [Stack was empty ]
  *i*=2 (inserting element is i )
  *fnew*(*S*)=max(*fprevious*(*S*),0)+*i*
  *fnew*(*S*)=max(0,0)+2=2

- INSERT −3 on to Stack
  *fprevious*(*S*)=2
  *i*=−3 (inserting element is i )
  *fnew*(*S*)=max(*fprevious*(*S*),0)+*i*
  *fnew*(*S*)=max(2,0)+−3=−1

GATE | Gate IT 2005 | Question 13

A function f defined on stacks of integers satisfies the following properties.

f(∅) = 0

f (push (S, i)) = max (f(S), 0) + i for all stacks S and integers i.

If a stack S contains the integers 2, -3, 2, -1, 2 in order from bottom to top, what is f(S)?

**Answer: (C)**

**Explanation:**

f(S) = 0, max(f(S), 0) = 0, i = 2

$f(S)_{new}$ = max(f(S), 0) + i = 0 + 2 = 2

f(S) = 2, max(f(S), 0) = 2, i = -3

$f(S)_{new}$ = max(f(S), 0) + i = 2 – 3 = -1

f(S) = -1, max(f(S), 0) = 0, i = 2

$f(S)_{new}$ = max(f(S), 0) + i = 0 + 2 = 2

f(S) = 2, max(f(S), 0) = 2, i = -1

$f(S)_{new}$ = max(f(S), 0) + i = 2 – 1 = 1

f(S) = 1, max(f(S), 0) = 1, i = 2

$f(S)_{new}$ = max(f(S), 0) + i = 1 + 2 = 3

Thus, option (C) is correct.

GATE | GATE MOCK 2017 | Question 19

Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)?

**(A)** O(1) for insertion and O(n) for deletion
**(B)** O(1) for insertion and O(1) for deletion
**(C)** O(n) for insertion and O(1) for deletion
**(D)** O(n) for insertion and O(n) for deletion

GATE | GATE MOCK 2017 | Question 19

Suppose a stack is to be implemented with a linked list instead of an array. What would be the effect on the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)?

**(A)** O(1) for insertion and O(n) for deletion
**(B)** O(1) for insertion and O(1) for deletion
**(C)** O(n) for insertion and O(1) for deletion
**(D)** O(n) for insertion and O(n) for deletion

**Answer: (B)**

**Explanation:** Stack can be implemented using link list having O(1) bounds for both insertion as well as deletion by inserting and deleting the element from the beginning of the list.

# The END!!