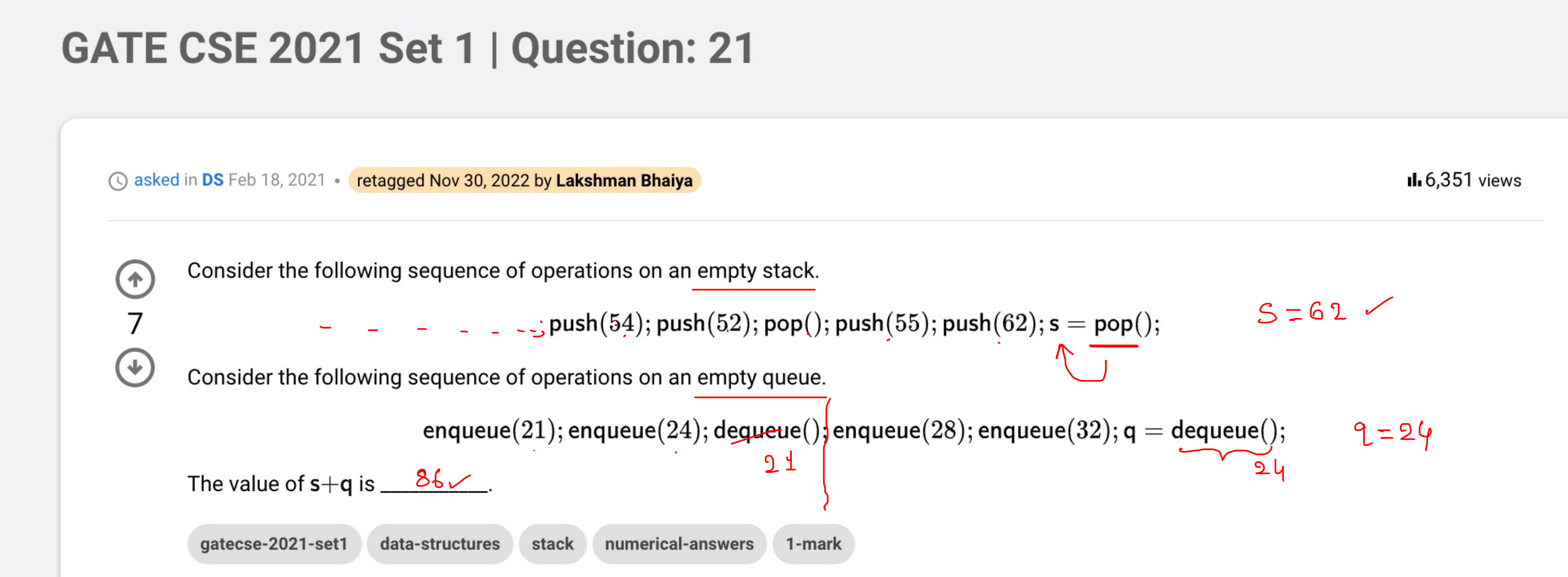
S3-Class2[Stack-2]



GATE CSE 2021 Set 1 | Question: 21

asked in DS Feb 18, 2021 • retagged Nov 30, 2022 by **Lakshman Bhaiya**

6,351 views

7

Consider the following sequence of operations on an empty stack.

$$\text{push}(54);\ \text{push}(52);\ \text{pop}();\ \text{push}(55);\ \text{push}(62);\ \text{s} = \text{pop}();$$

S = 62 ✓

Consider the following sequence of operations on an empty queue.

$$\text{enqueue}(21);\ \text{enqueue}(24);\ \underset{21}{\text{dequeue}()};\ \text{enqueue}(28);\ \text{enqueue}(32);\ \text{q} = \underbrace{\text{dequeue}()}_{24};$$

q = 24

The value of s+q is __86✓__.

gatecse-2021-set1 data-structures stack numerical-answers 1-mark

1) Stock Span problem

tomato :-

1 problem

4 more ques.

↳ NO-Duplicates (All values are dist)

n=7

price ↳

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

span ↳

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

By default

1+

1+ see it's prev day's for How many day's price is continiously smaller

n=7

P ↳

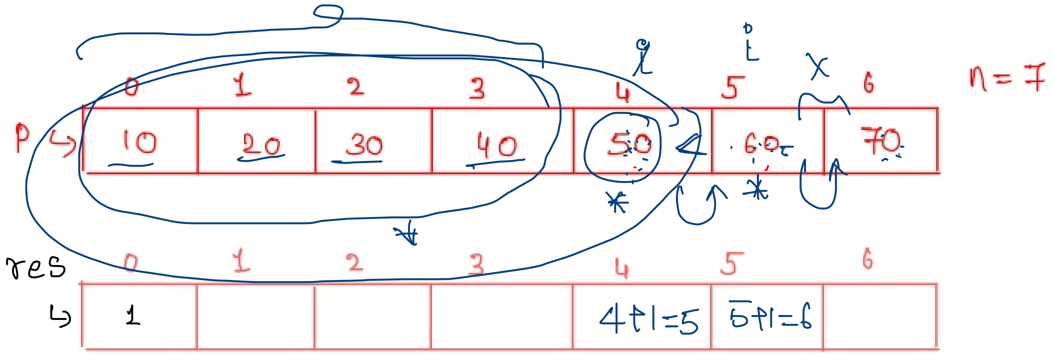
0	1	2	3	4	5	6
10	20	30	40	50	60	70

span ↳

0	1	2	3	4	5	6
1	2	3	4	5	6	7

BF :-

```
✓ int[] findSpan(int p[], int n)
  {
          int res[]=new int[n];
          res[0]=1;
          for(int i=1;i<n;i++)
          {       count =1;
            → for(int j=i-1; j>=0; j--)
              {
                    if( p[i] > p[j] )
                    {
                            count++;
                    }
                    else
                    {
                            break;
                    }
              }
              res[i] = count;
          }
          return res;
  }
```



	0	1	2	3	4	5	6
p ↳	10	20	30	40	50	60	70
res ↳	1				4+1=5	5+1=6	

j=1 j=2 j=3
✓ ✓ ✓
j=0
✓

when
{ i=4
p[4]=50

20 30 40 50
10

{ i=5 when
P[5]=60

top(big → delete()
small → don't pop()

p

	0	1	2	3	4	5	6
p	100	~~80~~ 50	60	70	60	75	85
res	1	? X	X				

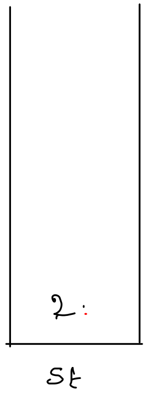
P[st.peek()] v/s P[i]

>

<

wrong one

↳ Correct one



a[i]

left

> a[i]

< a[i]

$\underbrace{P[st.peek()] > P[i]}_{\text{deleting}}$

X

```
int[] findSpan(int p[], int n)
{
        int res[]=new int[n];
        res[0]=1;
     -> for(int i=1;i<n;i++)
        {
                while(p[st.peek()]<p[i])
                        st.pop()
        }
}
```

* once you are done with arr[i], push the index into stack

usefull : $\underbrace{\text{see all left}}_{\text{side ele's}}$ a[i]

↳ these are smaller, then I need it
so i will pop()

useless :

*

```
int[] findSpan(int p[], int n)
{
        int res[]=new int[n];
        res[0]=1;
        for(int i=1;i<n;i++)
        {
            while(p[st.peek()]<p[i])  XXx
            {
                    st.pop();
            }
    * * * res[i]=i-st.peek();
            st.push(i);
        }
}
```

stack-ele

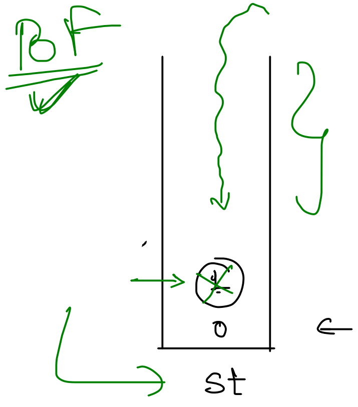
arr-ele

Inc.

i i n = 7

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

BF



← pushing array indices

Doubt:-

60 < 70

↳ required for us, BUT pop()

are we not going to loss the info

Note:- In Brute force approach, un necessary comparisons are happening

In stack approach, the comparisons are happening w.r.t to stack element and array element,

if the elemnets let's say smaller than p[i], then if you put that element in stack, which again leads to
BRUTEFORCE, So that is the reason if element is samller than p[i]
delete it from stack ==> put only the elements which are greater than p[i]

so that comparison are happens only with greater elements

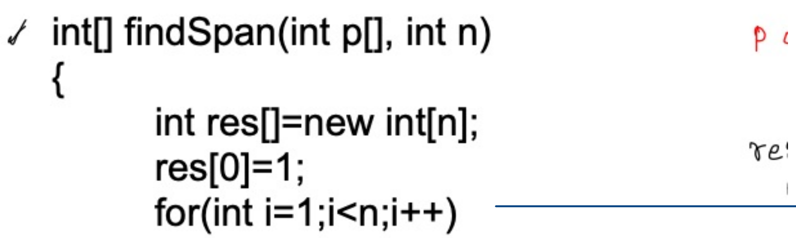
```
int[] findSpan(int p[], int n)
{
        int res[]=new int[n];
        res[0]=1;
        st.push(0)
        for(int i=1;i<n;i++)
        {
                while( !st.isEmpty()&&p[st.peek()]<p[i] )
                {
                        st.pop();
                }
                if(!st.isEmpty())
                {
                        res[i]=i-st.peek();
                }
                else
                {
                        res[i]= ?? // hoemwork
                }
                st.push(i);
        }
}
```

p ↳

0	1	2	3	4	5	6
10	20	30	40	50	60	70

n=7

12:50pm



```
✓ int[] findSpan(int p[], int n)                    p (
  {
          int res[]=new int[n];
          res[0]=1;                                  re
          for(int i=1;i<n;i++)          ——————————→ i
          {       count = 1;
        → for(int j=i-1; j>=0; j--)     ——→ j
          {                                  (i-1)
                  if( p[i] > p[j] )
                  {
                          count++;
                  }
                  else
                  {
                          break;
                  }
          }
          res[i] = count;
          }
          return res;
  }
```

Dep
↳ un-rolling

$i=0$ ×

$i=1 \rightarrow j=1$

$i=2 \rightarrow j=2$

$i=3 \rightarrow j=3$

⋮

$i=n \rightarrow j=n$

$$1+2+\dots+n = \frac{n(n+1)}{2} \Rightarrow O(n^2)$$

↳ TLE

↳ NOT in-terms of value, {interms of closeness

2) Nearest / Immediate Smaller Element to its Left [NSE Left]

L → R a[i]

n=8

i/p

	0	1	2	3	4	5	6	7
a ↳	7	2	4	6	9	3	8	11

BF

↳ 2loops ✓

$O(n^2)$

o/p

	0	1	2	3	4	5	6	7
res ↳	-1	-1	2	4	6	2	3	8

	0	1	2	3	4	5	6	7
a ↳	7	2	1	4	9	8	21	5

	0	1	2	3	4	5	6	7
res ↳	-1	-1	-1	1	4	4	8	4

✓

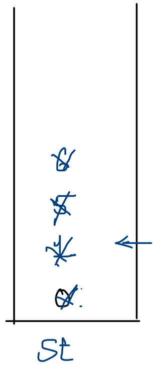
j ~~> i

arr ↳

0	1	2	3	4	5	6	7
7	2	4	6	9	3	~~8~~ 4	X 1

res ↳

0	1	2	3	4	5	6	7
-1	-1	? 2	4	6	2	3	-1



a[i]

* Among All i't's left side ele's
smaller ele I need,
so should I push/pop

* you will get
in stack

arr[st.peek()] > arr[i] ⇒ NOT required
so I will pop()

```
int[] NSELeft(int arr[], int n)
{
        int res[]=new int[n];
        res[0]=-1;
        st.push(0)
        for(int i=1;i<n;i++)
        {
                while( !st.isEmpty()&&arr[st.peek()]>arr[i] )
                {
                        st.pop();
                }
                if(!st.isEmpty())
                {
                        res[i]=arr[st.peek()];
                }
                else
                {
                        res[i]= -1;
                }
                st.push(i);
        }
}
```

3) Nearest / Immediate Smaller Element to its Right [NSE Right]

L i R

n = 8

BF
↳ 2loops ; $O(n^2)$

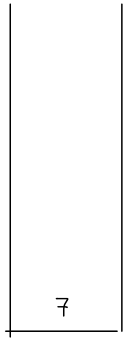
arr ↳

0	1	2	3	4	5	6	7
7	2	4	6	9	3	8	11

res ↳

0	1	2	3	4	5	6	7

x



$a[i]\ \underbrace{\qquad\qquad\qquad}$

↑

$\text{smaller}[\underbrace{\text{Right side}}_{\text{in stack}}\ \text{ele}]$ ⇒

$\underline{arr[st.peek()]} > arr[i]$ ⇒ pop() ✓

```
int[] NSERight(int arr[], int n)
{
        int res[]=new int[n];
        res[n-1]=-1;
        st.push(n-1)
        for(int i=n-2;i>=0;i--)
        {
                while( !st.isEmpty()&&arr[st.peek()]>arr[i] )
                {
                        st.pop();
                }
                if(!st.isEmpty())
                {
                        res[i]=arr[st.peek()];
                }
                else
                {
                        res[i]= -1;
                }
                st.push(i);
        }
}
```

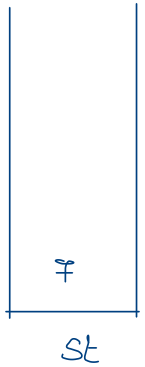
4) Nearest / Immediate Greater Element to its Right [NSE Right]

i

n = 8

	0	1	2	3	4	5	6	7
arr	7	2	4	6	9	3	8	11

	0	1	2	3	4	5	6	7
res	9	4	6	9	11	8	11	-1



$a[i]\ \underbrace{\qquad\qquad\qquad}_{\text{right}}$

arr[st.peek()] > arr[i] ⇒ required ✓

arr[st.peek()] < arr[i] ⇒ pop()

not required

```
int[] NSERight(int arr[], int n)
{
        int res[]=new int[n];
        res[n-1]=-1;
        st.push(n-1)
        for(int i=n-2;i>=0;i--)
        {
                while( !st.isEmpty()&&arr[st.peek()]<arr[i] )
                {
                        st.pop();
                }
                if(!st.isEmpty())
                {
                        res[i]=arr[st.peek()];
                }
                else
                {
                        res[i]= -1;
                }
                st.push(i);
        }
}
```

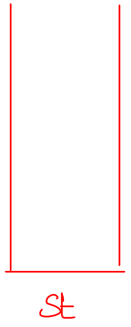
5) Nearest / Immediate Greater Element to its Left   [NSE Left]

arr

0	1	2	3	4	5	6	7
7	2	4	6	9	3	8	11

res

0	1	2	3	4	5	6	7
-1	7	7	7	-1	9	9	-1



a[i]

{left ele need}

arr[st.peek()] > arr[i] => reaired

arr[st.peek()] < arr[i] => pop()

```
int[] NSELeft(int arr[], int n)
{
        int res[]=new int[n];
        res[0]=-1;
        st.push(0)
        for(int i=1;i<n;i++)
        {
                while( !st.isEmpty()&&arr[st.peek()]<arr[i] )
                {
                        st.pop();
                }
                if(!st.isEmpty())
                {
                        res[i]=arr[st.peek()];
                }
                else
                {
                        res[i]= -1;
                }
                st.push(i);
        }
}
```

1. Stock - span problem

2,3 ↳ NSE [left + right]

4,5 ↳ NGE [left + right]

⇒ 5 problems

OJ
↳ may be slight variation ✓

3) Nearest / Immediate Greater Element to its Left [NGE Left]

4) Nearest / Immediate Greater Element to its Right   [NGE Right]