Q.1 **Maximum Positivity**

Given an array, return the maximum size subarray with only **non-negative** elements.

$$A = \begin{array}{cccccc} [\,5 & 6 & -1 & 7 & 3 & 8\,] \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

$$A = \begin{array}{cccccccccc} [-5 & -3 & 1 & 7 & 3 & 4 & -9 & -10 & 8 & 12\,] \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

observation: we are interested in consecutively coming non-negative elements.

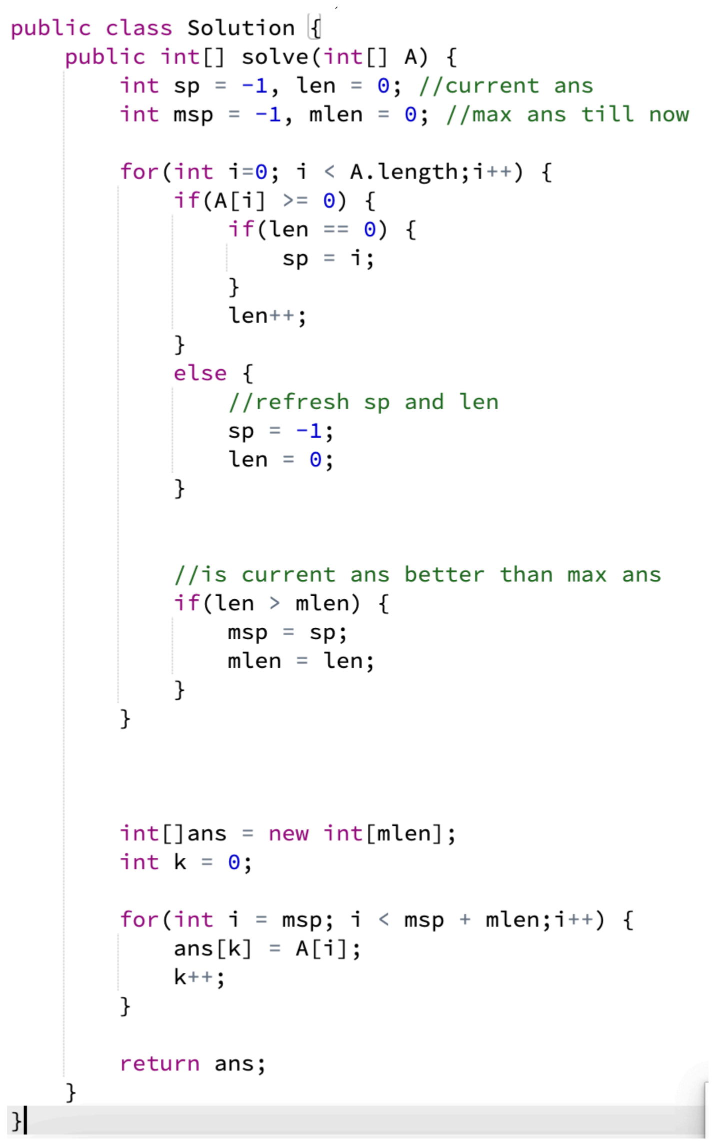
$\swarrow$

$$A = \begin{array}{cccccc} [\,5 & 6 & -1 & 7 & 3 & 8\,] \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

$sp = \cancel{-1}\ \cancel{0}\ \cancel{-1}\ 3$ $\qquad len = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{0}\ \cancel{1}\ \cancel{2}\ 3$ $\qquad$ [current state]

$msp = \cancel{-1}\ \cancel{0}\ 3$ $\qquad mlen = \cancel{0}\ \cancel{1}\ \cancel{2}\ 3$ $\qquad$ [max ans till now]

$$A = \begin{array}{cccccccccc} [-5 & -3 & 1 & 7 & 3 & 4 & -9 & -10 & 8 & 12] \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$



```
public class Solution {
    public int[] solve(int[] A) {
        int sp = -1, len = 0; //current ans
        int msp = -1, mlen = 0; //max ans till now

        for(int i=0; i < A.length;i++) {
            if(A[i] >= 0) {
                if(len == 0) {
                    sp = i;
                }
                len++;
            }
            else {
                //refresh sp and len
                sp = -1;
                len = 0;
            }


            //is current ans better than max ans
            if(len > mlen) {
                msp = sp;
                mlen = len;
            }
        }



        int[]ans = new int[mlen];
        int k = 0;

        for(int i = msp; i < msp + mlen;i++) {
            ans[k] = A[i];
            k++;
        }

        return ans;
    }
}
```

$sp = \cancel{-1}\ \cancel{2}$
$\quad \cancel{-1}\ 8$

$len = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{3}\ \cancel{4}$
$\quad \cancel{0}\ \cancel{1}\ 2$

$msp = \cancel{-1}\ 2$

$mlen = \cancel{0}\ \cancel{1}\ \cancel{2}\ \cancel{3}\ 4$

$$ans = \begin{array}{cccc} [1 & 7 & 3 & 4] \\ 0 & 1 & 2 & 3 \end{array}$$

i	k
2	0
3	1
4	2
5	3

TC : O(N)

Q.2 Little ponny and Maximum element.

Given an array, in one operation we can make ele = -1.
Find min no. of operations required to make B the max of array.

$$A = \begin{array}{ccccccc} [3 & 9 & 5 & 4 & 1 & 0 & 5] \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

B = 4
ans = 3

$$A = \begin{array}{ccccccc} [5 & 8 & 1 & 12 & 6 & 10 & 2] \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

B = 6
ans = 3

A = [5 8 1 10]

B = 4
ans = -1

Obs: ele > B are stopping it to becoming of the array.

```java
public class Solution {
    public int solve(int[] A, int B) {
        boolean temp = false;
        int cnt = 0;

        for(int i=0; i < A.length;i++) {
            if(A[i] == B) {
                temp = true;
            }
            else if(A[i] > B) {
                cnt++;
            }
        }


        if(temp == false) {
            //B is not present in the array
            return -1;
        }
        else {
            return cnt;
        }
    }
}
```

$B = 6$

$$A = \begin{array}{ccccccc} [5 & 8 & 1 & 12 & 6 & 10 & 2] \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

temp = ~~false~~ true

cnt = $\cancel{0}\ \cancel{1}\ \cancel{2}\ 3$

TC : O(N)

Q.3 Vowels in Range

Given a string and Q queries, for every query find out the no. of vowels in the range L to R.

A = amateurship

a	m	a	t	e	u	r	s	h	i	p
0	1	2	3	4	5	6	7	8	9	10

Queries

L	R	ans
0	3	2
4	8	2
5	5	1
3	9	3

A =	a	m	a	t	e	u	r	s	h	i	p
	0	1	2	3	4	5	6	7	8	9	10

PC =	1	1	2	2	3	4	4	4	4	5	5
	0	1	2	3	4	5	6	7	8	9	10

$pc[i] = pc[i-1] + (1 \text{ or } 0)$

$pc[i]$

$\Rightarrow$ no. of vowels in the range 0 to i in string A

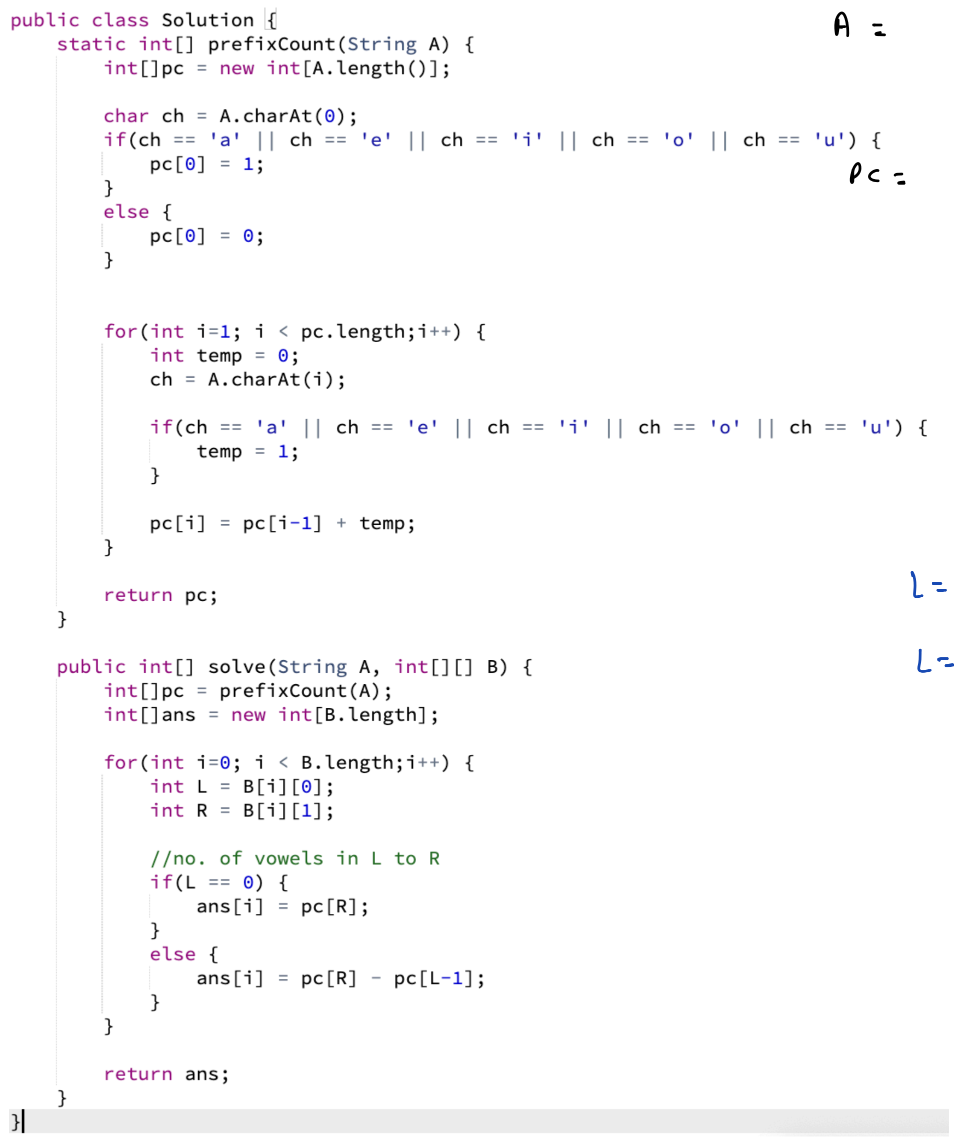
A =	a	m	a	t	e	u	r	s	h	i	p
	0	1	2	3	4	5	6	7	8	9	10

pc =	1	1	2	2	3	4	4	4	4	5	5
	0	1	2	3	4	5	6	7	8	9	10

ans(3, 7) $\Rightarrow$ pc[7] - pc[2] = 4 - 2 = 2

ans(4, 9) $\Rightarrow$ pc[9] - pc[3] = 5 - 2 = 3

$$\begin{tikzcd}
 & L, R \arrow[dl, dash, "\text{if } L=0"'] \arrow[dr, dash, "\text{else}"] & \\
pc[R] & & pc[R] - pc[L-1]
\end{tikzcd}$$



```java
public class Solution {
    static int[] prefixCount(String A) {
        int[]pc = new int[A.length()];

        char ch = A.charAt(0);
        if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            pc[0] = 1;
        }
        else {
            pc[0] = 0;
        }


        for(int i=1; i < pc.length;i++) {
            int temp = 0;
            ch = A.charAt(i);

            if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
                temp = 1;
            }

            pc[i] = pc[i-1] + temp;
        }

        return pc;
    }

    public int[] solve(String A, int[][] B) {
        int[]pc = prefixCount(A);
        int[]ans = new int[B.length];

        for(int i=0; i < B.length;i++) {
            int L = B[i][0];
            int R = B[i][1];

            //no. of vowels in L to R
            if(L == 0) {
                ans[i] = pc[R];
            }
            else {
                ans[i] = pc[R] - pc[L-1];
            }
        }

        return ans;
    }
}
```

A =	a	y	g	i	k	u
	0	1	2	3	4	5
pc =	1	1	1	2	2	3
	0	1	2	3	4	5

B = [[0, 2]
[2, 4]
]

L = 0, R = 2, pc[2] = 1

L = 2, R = 4, pc[4] - pc[1]
= 2 - 1 = 1

TC: O(N+Q)

SC: O(N)