Maximum positivity.

[9 3 11 -2 1 2]

Find a subarray with max^m length such that it doesn't contain any -ve elements.

Ans: [9 3 11]

Idea: Generate all subarrays, and keep track of max length subarray with no -ve values.

[9 3 11 -2 1 2]
 i j

Starts with 9.
[9]
[9 3]
[9 3 11]

Starts with 3
[3]
[3 11]

Starts with 11
[11]

Starts with -2
[ ]

Starts with 1
[1]
[1 2]

Starts with 2
[2].

Idea 2: My ans is from [l..r], Both l and r must lie on the side of any negative number in the array.

[9 3 11 -2 1 2]
 j
 i

[9]
[9 3]
[9 3 11]
[1]
[1 2]

```
def maxPositivity(arr, N)
{
    i=0, j=0, stat=0, end=0.        → This will store final subarray.
    while(i<N && j<N)
    {
        if(arr[j]>=0)
            j=j+1
        else
            if((j-i) > (end-stat+1))
                stat = i
                end = j
            i=j+1
            j=j+1
    }
    return arr[stat:end]
}
```

TC: O(N)
SC: O(1)

→ To make sure we written the first subarray, in case of overlap.

Q2: Sort the permutation. [All the elements in the array will be from 1-N].

The no's that can be sorted among themselves independent of others should be colored the same.

Ex:1

A = [4 3 5 1 2]

1 2 3 4 5

[4 2 5 1 2]
5 3
3
2

5 groups.

(clubbing 4 & 1) → 4.
(clubbing 3 & 5) → 3.
(clubbing 5 & 2) → 2.

Ex: 2.

A = [1 2 4 3 5 6]

1 2 3 4 5 6

Observation: Ideal position for A[i] is $(i+1)^{th}$ index, If its not in that position then this leads to a new group.

Idea: Assume that there are 'N' independent groups, and you can decrementing the value of 'N' as you club the nos (swap the nos).

Code:

```
def sortThePermutation(arr, n)
    ans = n.
    for i in range(0,n)
        if(A[i]!=i+1)
            while( A[i]!=i+1 )
                ans = ans-1
                x = arr[i]
                A[i] = A[x-1]
                A[x-1] = x

    return ans
```

Simply swap A[i] with A[A[i]-1].

Tc: O(N).
Sc: O(1).

Reason for Tc.

(i) looking at ans.
(ii) At the max, you've n swaps.

Q3: Bus Dilemma.

B → max no. of people that bus can accomodate.

5.

Possible values for no. of people in the bus initially, in-order for the above data to be valid.

Ex: (i) Can we've 0 people in the bus → 2, 3, 0 ✓

" " 1 " " " " → 3, 4, 1 ✓

" " 2 " " " " → 4, 5, 2 ✓

" " 3 " " " " → 5, ⑥ ✗.

(ii) No. of people can't be negative initially, start with 0.

The possible values lies from [0 -- B].

Idea: Try checking for every value from [0 -- B] as shown above, The running sum you get should'nt exceed B, and should'nt be -ve.

Idea 2: Try using prefix sum.

Arr: [2 1 -3]

Pf: [2 3 0]. , B = 5.

Pfmax & Pfmin

[0 1 2 3 4 5]

(i) Running sum should'nt be more than B.

The chosen value must not exceed [0 → B - Pfmax].

[a b) = b - a + 1.

[B - Pfmax + 1] .-> ans.

(ii) Running sum shouldn't be -ve.

[-2, 1] , B = 4.

Pf [-2, 1].

0
1
2
3
4.

If Pfmin < 0,

ans = ans - Pfmin.

The sol^n lies from [0 to B - Pfmax].

Total elements = B - Pfmax + 1.

Corners.

(i) If (-Pfmin) < B, Can there be a sol^n.? No. return 0.

[-4, -5] , B = 3.

(ii) If Pfmin < 0, remove Pfmin elements.

ans = ans + Pfmin.

(iii) If Pfmax < 0, remove Pfmax elements , => ans = ans + Pfmax.

[-3, -7] , B = 20.

If [-3, -10]

ans = [10 ... 20].
= 11.

```
def busDilemma(arr, N)
{
    // create pf and populate it.
    // find Pfmax & Pfmin

    if (Pfmin < B)
        return 0

    ans = B - Pfmax + 1

    if (Pfmin < 0)
        ans = ans + Pfmin

    if (Pfmax < 0)
        ans = ans + Pfmax

    return ans
}
```

TC: O(N).
SC: O(N).