**Question 3. PnC**

Divyansh Mathur, 210357

1. **Characterise all the permutations A(Π0) = [aΠ0(1), aΠ0(2), aΠ0(3) . . . , aΠ0(n) ] of A such that S(A(Π0)) = min Π S(A(Π))**

To minimize the score, we need to minimize the sum .

Point to note is that the maximum element will always come out of the modulus as +ve and the smallest element will come out of the modulus as negative so we will have

Our aim remains to minimize this sum S, this sum can be easily visualised as 0 when the array is sorted in **increasing and decreasing order.**

When in decreasing order

Score=

=

(since sorted in decreasing order)

{

When in increasing order

Score=

=

( since sorted in decreasing order)

{

**Good permutations : Array sorted in 1. Increasing and 2. Decreasing order.**

1. **Algorithm which computes the minimum cost required to transform the given array A into a good permutation:**

In previous part we have seen that the good permutation is just the increasing or decreasing order of the array.

Idea is to sort the array (in both increasing and decreasing fashion) and find the minimum cost to sort the array.

*Observations:*

Consider the maximum element, if this element is not located on its required place then it is to be swapped and whichever element it is swapped with, the cost will be the maximum element. So we first place this maximum element on its required position i.e. at the last. Now our working space has been reduced, this element will not interfere with the cost anymore. Again we place the maximum element of the remaining array on its correct position and so on.

*Algorithm:*

Let the given Array be A[n]

First sort the array and store it in another array say B[n].

Create two arrays: temp[n] and pos[n] ,

temp array: stores where is the element in the sorted array such that A[i]= B[temp[i]]

pos array : stores where is the element in the original array such that B[i]=A[pos[i]]

temp array and pos array are related such that pos[temp[i]]=i

temp array can be easily filled using binary search on the sorted array for each element.

Ex. A[5]= 7,2,5,4,1

Sorting it: B[5]= 1,2,4,5,7

Filling temp array{

Find A[0] in B array using binary seach: 5 th position ( 4 in 0 based indexing),

Similarly finding other elements and filling temp array:}

temp[5]= 4,1,3,2,0

pos[5]=4,1,3,2,0

verifying: B[0]=1, pos[0]=4, yes 1 is A[4]

B[1]=2, pos[1]=1, yes 2 is A[1] and so on..

Sorting takes nlogn time  
binary search for each element takes logn time and nlogn for all  
Now for our algorithm, we need to move from right to left in our B array and see where this element should have been, If it is at its position than ok, else swap it and add the maximum value to cost.  
This takes O(n) time since for each B[i] we are taking constant time to swap

Overall Time complexity = O(nlogn)

*Swapping*

Say, temp[i]=k and pos[i]=j  
implies pos[k]=i and temp[j]=i  
To swap we need to bring A[j] to A[i]  
We can’t just use the swap function because then our pos and temp array will not be functional as they don’t map value to index they map index to index.

Now DO,  
temp[j]=k and temp[i]=i  
pos[k]=j and pos[i]=i

A: A[0] , A[1], A[2]….A[j]…. A[i]….A[n-1]

B: B[0], B[1], B[2] ….B[k]…..B[i],…..B[n-1]

**NEED TO CHECK FOR BOTH INCREASING AND DECREASING ORDER:**

We will do the same analysis for decreasing order also and report   
min(cost\_increasing, cost\_decresing)