The algorithm to generate permutations can have two cases . For eg. if Set Pd = {(1,1), (1,2),(2,2),(2,3), (3,3)}

Then

1. Considering the swaps in order i.e (i1,j1) = (1,2), (i2,j2) = (2,3), (i3,j3) = (3,3) and so on
2. Considering the swaps out of order (i1,j1) = (2,3),(i2,j2) = (3,3), (i3,j3) = (1,2)

The task has specifically asked to present the pseudo-code for permutation generation,

**Assumptions**

Let set **Pd** be swapsSet,

**swap**(swap\_pair, permutation) be a function that takes the pair (i,j) and the permutation as parameters and perform the swap (i and j th number ) and return the swapped permutation,

**permutationsSet -** each generated permutation is added in the set and then this set is returned,

**SwapsSet -**  Pd set having swap elements (i1,j1) , (i2,j2) , (i3,j3).......,

**set\_1.add(set\_2) -** takes a set **set2** as parameter and add its elements to the set **set\_1,**

**set\_1.remove(element) -** function that takes an element as a parameter and remove it form the set **set\_1,**

**generatePermutationSets(P, swapsSet, l)-** function/procedure that takes permutation **P, swapsSet** & value of **l**

**Case -1**

Considering the swaps to be taken in order of their occurrence in the Pd set,

We can have a recursive function that can generate permutations for given values of d, l , σ (permutation; lets denote **σ** as P)

generatePermutationSets (P, swapsSet ,l)

Begin :

If ( l<=0)

return P

permutationsSet = empty[] //empty list

swapsSetCopy = swapsSet // copying the contents of swapsSet Pd to another set

For each element of swapsSet // iterating over each element of swapsSet

Begin:

P\_new = swap (element, P)

swapsSetCopy.remove(element) // remove the used swap pair(i,j) from the swaps set

PermutationSet.add ( generatePermutationSets(P\_new, swapsSetCopy, l-1))

End:

return permutationsSet

End :

The above recursive function can be used to generate permutations (d,l) reachable from **σ (permutation P)** by performing the swaps in order of their occurrence in the set Pd.

**Case- 2**

If the order of the swaps is not to be taken into account and they can be applied without order like performing (2,3) swap before(1,2) swap on permutation where (1,2) appears before (2,3) in the ordered pair set of swaps (set Pd) **,**

Then the following recursive function can be used to generate (d,l) - reachable permutations for given values of d, l , σ (permutation; lets denote **σ** as P)

Pseudo code -

generatePermutationSets (P, swapsSet ,l)

Begin :

If ( l<=0)

Return p

permutationsSet = empty[] //empty list

For each element of swapsSet

Begin:

P\_new = swap (element, P)

swapsSetCopy = swapsSet

// copying the contents of swapsSet Pd to another set

swapsSetCopy.remove(element) // remove the used swap pair(i,j) from the swaps set

PermutationSet.add ( generatePermutationSets(P\_new, swapsSetCopy, l-1))

End:

return permutationsSet

End :