EXERCISE SOLUTION

1. Given N friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out the total number of ways in which friends can remain single or can be paired up.

Note: Since answer can be very large, return your answer mod 10^9+7.

Example:

```
Input: N = 3 Output: 4 Explanation:\{1\}, \{2\}, \{3\}: All single\{1\}, \{2,3\}: 2 and 3 paired but 1 is single.\{1,2\}, \{3\}: 1 and 2 are paired but 3 is single.\{1,3\}, \{2\}: 1 and 3 are paired but 2 is single.Note that \{1,2\} and \{2,1\} are considered same.
```

```
# Returns count of ways
# n people can remain
# single or paired up.
def friend(n):
    c = [0 for i in range(n + 1)]
```

Filling dp[] in bottom-up manner using # recursive formula explained above.

```
for i in range(n + 1):
    if(i <= 2):
      c[i] = i
    else:
       c[i] = c[i - 1] + (i - 1) * c[i - 2]
  return c[n]
# Driver code
n = 3
print(friend(n))
m
Time Complexity: O(n)
Auxiliary Space : O(n)
Another approach: (Using recursion)
111
```

111

2. Given a gold mine of n*m dimensions. Each field in this mine contains a positive integer which is the amount of gold in tons. Initially the miner is at first column but can be at any row. He can move only (right->,right up /,right down\)

that is from a given cell, the miner can move to the cell diagonally up towards the right or right or diagonally down towards the right. Find out maximum amount of gold he can collect.

Example:

111

$$MAX = 100$$

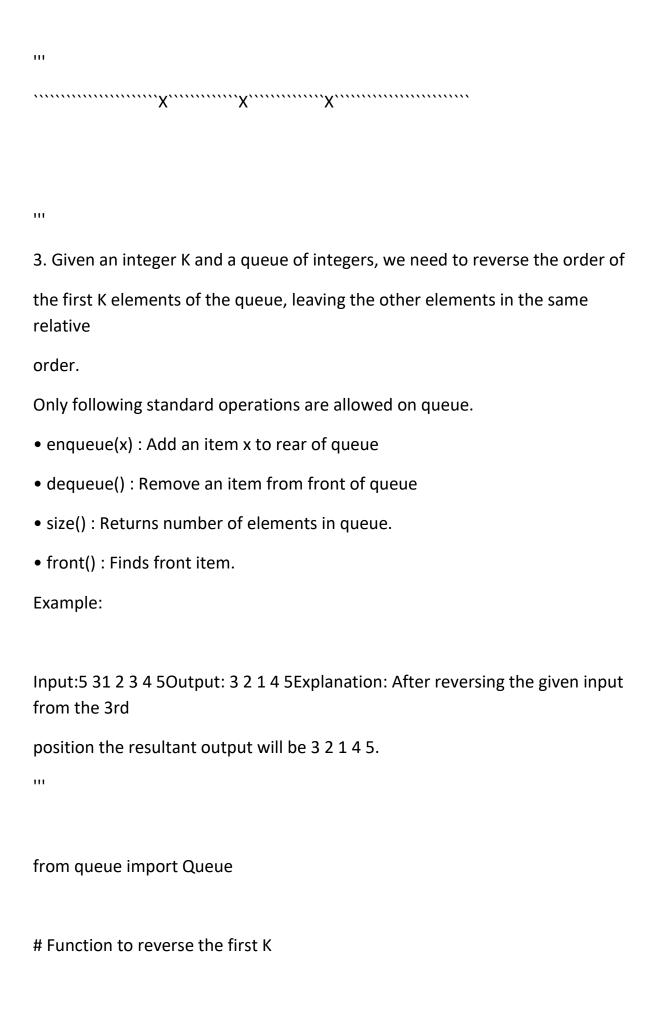
Returns maximum amount of

gold that can be collected

```
# when journey started from
# first column and moves
# allowed are right, right-up
# and right-down
def getMaxGold(gold, m, n):
  # Create a table for storing
  # intermediate results
  # and initialize all cells to 0.
  # The first row of
  # goldMineTable gives the
  # maximum gold that the miner
  # can collect when starts that row
  goldTable = [[0 for i in range(n)]
             for j in range(m)]
  for col in range(n-1, -1, -1):
    for row in range(m):
      # Gold collected on going to
      # the cell on the right(->)
      if (col == n-1):
         right = 0
```

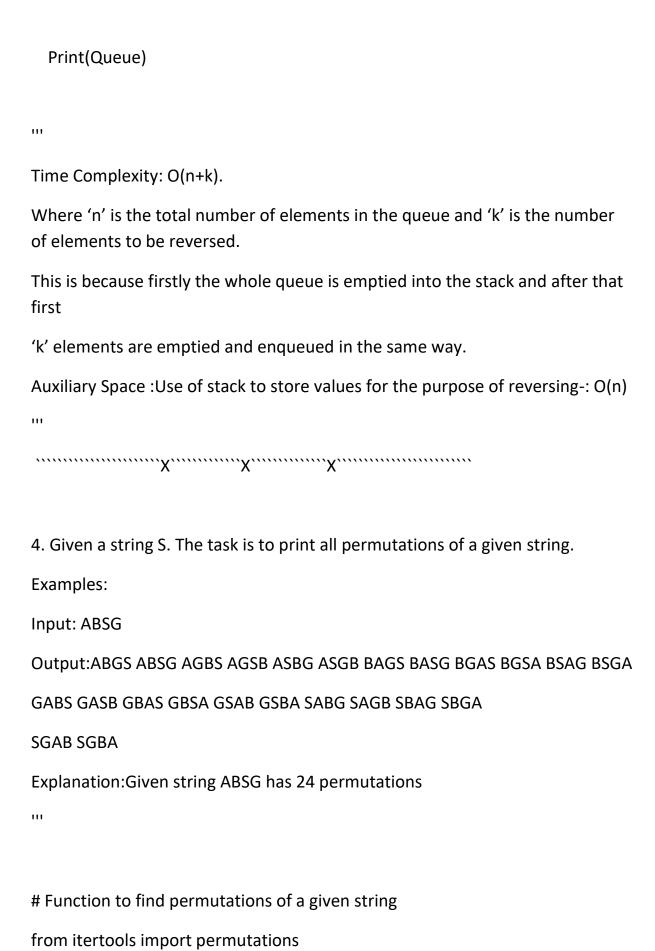
```
else:
      right = goldTable[row][col+1]
    # Gold collected on going to
    # the cell to right up (/)
    if (row == 0 or col == n-1):
      right up = 0
    else:
       right_up = goldTable[row-1][col+1]
    # Gold collected on going to
    # the cell to right down (\)
    if (row == m-1 or col == n-1):
      right_down = 0
    else:
      right_down = goldTable[row+1][col+1]
    # Max gold collected from taking
    # either of the above 3 paths
    goldTable[row][col] = gold[row][col] + max(right, right_up, right_down)
# The max amount of gold
# collected will be the max
```

```
# value in first column of all rows
  res = goldTable[0][0]
  for i in range(1, m):
    res = max(res, goldTable[i][0])
  return res
# Driver code
gold = [[10, 33, 13, 15],
  [22, 21, 4, 1],
  [5, 0, 2, 3],
  [0, 6, 14, 2]]
m = 4
n = 4
print(getMaxGold(gold, m, n))
111
Time Complexity:O(m*n)
Space Complexity: O(m*n)
```



```
# elements of the Queue
def reverseQueueFirstKElements(k, Queue):
  if (Queue.empty() == True or
       k > Queue.qsize()):
    return
  if (k \le 0):
    return
  Stack = []
  # put the first K elements
  # into a Stack
  for i in range(k):
    Stack.append(Queue.queue[0])
    Queue.get()
  # Enqueue the contents of stack
  # at the back of the queue
  while (len(Stack) != 0):
    Queue.put(Stack[-1])
    Stack.pop()
  # Remove the remaining elements and
```

```
# enqueue them at the end of the Queue
  for i in range(Queue.qsize() - k):
    Queue.put(Queue.queue[0])
    Queue.get()
# Utility Function to print the Queue
def Print(Queue):
  while (not Queue.empty()):
    print(Queue.queue[0], end =" ")
    Queue.get()
# Driver code
if __name__ == '__main__':
  Queue = Queue()
  Queue.put(1)
  Queue.put(2)
  Queue.put(3)
  Queue.put(4)
  Queue.put(5)
  k = 3
  reverseQueueFirstKElements(k, Queue)
```



```
def allPermutations(str):
  permList = permutations(str)
  # print all permutations
  for perm in list(permList):
    print (".join(perm))
# Driver program
if __name__ == "__main__":
 str = 'ABSG'
 allPermutations(str)
111
Expected Time Complexity: O(n! * n)
Expected Space Complexity: O(n)
111
.....X.....X.....X.....X......X
```

111

5. Given the root of a binary tree. Check whether it is a BST or not.

Note: We are considering that BSTs cannot contain duplicate Nodes.

A BST is defined as follows:

• The left subtree of a node contains only nodes with keys less than the node's

key.

• The right subtree of a node contains only nodes with keys greater than the

node's key.

• Both the left and right subtrees must also be binary search trees.

Example:

Input: 2 / \1 3

Output: 1 Explanation: The left subtree of root node contains node with

key lesser than the root node's key and the right subtree of root node contains

node with

key greater than the root node's key. Hence, the tree is a BST.

111

""" Program to check if a given Binary

Tree is balanced like a Red-Black Tree """

Helper function that allocates a new

node with the given data and None

left and right poers.

class newNode:

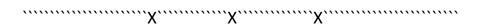
```
# Construct to create a new node
      def __init__(self, key):
             self.data = key
             self.left = None
             self.right = None
# Returns true if given tree is BST.
def isBST(root, I = None, r = None):
      # Base condition
      if (root == None):
             return True
      # if left node exist then check it has
      # correct data or not i.e. left node's data
      # should be less than root's data
      if (I != None and root.data <= I.data):
             return False
      # if right node exist then check it has
      # correct data or not i.e. right node's data
```

```
# should be greater than root's data
      if (r != None and root.data >= r.data) :
            return False
      # check recursively for every node.
      return isBST(root.left, I, root) and \
            isBST(root.right, root, r)
# Driver Code
if __name__ == '__main__':
      root = newNode(2)
      root.left = newNode(1)
      root.right = newNode(3)
      #root.right.left = newNode(1)
      #root.right.right = newNode(4)
      #root.right.left.left = newNode(40)
      if (isBST(root,None,None)):
            print("Is BST")
      else:
            print("Not a BST")
```

Time Complexity: O(n)

Auxiliary Space: O(1) if Function Call Stack size is not considered, otherwise

O(n)



6. Given a sorted array arr containing n elements with possibly duplicate elements, the task is to find indexes of first and last occurrences of an element x in the given array.

Example:

Input:n=9, x=5arr[] = { 1, 3, 5, 5, 5, 5, 67, 123, 125 }

Output: 2 5Explanation: First occurrence

of 5 is at index 2 and last occurrence of 5 is at index 5.

111

last occurrences of a number in

a given sorted array

if x is present in arr[] then

returns the index of FIRST

occurrence of x in arr[0..n-1],

otherwise returns -1

def first(arr, low, high, x, n):

```
if(high >= low):
              mid = low + (high - low) // 2
              if( ( mid == 0 \text{ or } x > arr[mid - 1]) and arr[mid] == x):
                     return mid
              elif(x > arr[mid]):
                     return first(arr, (mid + 1), high, x, n)
              else:
                     return first(arr, low, (mid - 1), x, n)
       return -1
# if x is present in arr[] then
# returns the index of LAST occurrence
# of x in arr[0..n-1], otherwise
# returns -1
def last(arr, low, high, x, n):
       if (high >= low):
              mid = low + (high - low) // 2
              if ((mid == n - 1 \text{ or } x < arr[mid + 1]) \text{ and } arr[mid] == x):
                     return mid
              elif (x < arr[mid]):
                     return last(arr, low, (mid - 1), x, n)
```

```
return -1
# Driver program
arr = [1, 3, 5, 5, 5, 5, 67, 123, 125]
n = len(arr)
x = 5
print("First Occurrence = ",
     first(arr, 0, n - 1, x, n))
print("Last Occurrence = ",
     last(arr, 0, n - 1, x, n))
111
Time Complexity: O(log n)
Auxiliary Space : O(Log n)
111
GITHUB REPOSITORY
```

return last(arr, (mid + 1), high, x, n)

else: