```python
#Arrays: Left Rotation
def array_left_rotation(a, n, k):
  r = k % n
  return a[r:] + a[:r]

n, k = map(int, raw_input().strip().split(' '))
a = map(int, raw_input().strip().split(' '))
answer = array_left_rotation(a, n, k);
print ' '.join(map(str,answer))

#Strings: Making Anagrams
def number_needed(a, b):
    cTable = [0]*26
    for i in a:
        cTable[ord(i)-97] += 1
    for i in b:
        cTable[ord(i)-97] -= 1

    return sum([abs(i) for i in cTable])

a = raw_input().strip()
b = raw_input().strip()
print number_needed(a, b)

#Hash Tables: Ransom Note
def ransom_note(magazine, ransom):
    d = {}

    for i in magazine:
        if (i not in d):
            d[i] = 1
        else:
            d[i] += 1

    for i in ransom:
        if (i not in d or d[i] == 0):
            return False
        if (i in d):
            d[i] -= 1

    return True

m, n = map(int, raw_input().strip().split(' '))
magazine = raw_input().strip().split(' ')
ransom = raw_input().strip().split(' ')
answer = ransom_note(magazine, ransom)
if(answer):
    print "Yes"
else:
    print "No"

#Linked Lists: Detect a Cycle
def has_cycle(head):
    a = head
    b = head

    while (True):
        if (a.next == None): return 0

        a = a.next
        for i in xrange(2):
            if (b.next != None): b = b.next
            else: return 0

        if (a.data == b.data): return 1
```

```

#Stacks: Balanced Brackets
def is_matched(expression):
    stack = []

    for i in expression:
        if i == '[':
            stack.append(']')
        elif i == '(':
            stack.append(')')
        elif i =='{':
            stack.append('}')
        elif i in [']', '}', ')']:
            if len(stack) == 0 or stack.pop() != i: return False
    return len(stack) == 0



t = int(raw_input().strip())
for a0 in xrange(t):
    expression = raw_input().strip()
    if is_matched(expression) == True:
        print "YES"
    else:
        print "NO"


#Queues: A Tale of Two Stacks
class MyQueue(object):
    def __init__(self):
        self.labels = ['first', 'second']
        self.cur = 0
        self.first = []
        self.second = []

    def peek(self):
        if (not len(self.first) and not len(self.second)): return

        if (not len(self.second)):
            while (len(self.first) > 0):
                self.second.append(self.first.pop())

        res = self.second[-1]

        return res

    def pop(self):
        if (not len(self.first) and not len(self.second)): return

        if (not len(self.second)):
            while (len(self.first) > 0):
                self.second.append(self.first.pop())

        res = self.second.pop()

        return res

    def put(self, value):
        self.first.append(value)
```

```python

queue = MyQueue()
t = int(raw_input())
for line in xrange(t):
    values = map(int, raw_input().split())

    if values[0] == 1:
        queue.put(values[1])
    elif values[0] == 2:
        queue.pop()
    else:
        print queue.peek()


#Trees: Is this a binary search tree?
def check_binary_search_tree_(root):
  def traverse(node, low, high):
    if ((low == None or node.data > low) and (high == None or node.data < high)):
      left = True if node.left == None else traverse(node.left, low, node.data)
      right = True if node.right == None else traverse(node.right, node.data, high)

      return left and right
    else: return False
  return traverse(root, None, None)


def bfs(graph, start):
    visited, queue = set(), [start]
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited




```