

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'largestRectangle' function below.

#

# The function is expected to return a LONG\_INTEGER.

# The function accepts INTEGER\_ARRAY h as parameter.

#

def largestRectangle(h):

    stack = []

    max\_area = 0

    index = 0

    n = len(h)

    while index < n:

        if not stack or h[index] >= h[stack[-1]]:

            stack.append(index)

            index += 1

        else:

            top = stack.pop()

            width = index if not stack else index - stack[-1] - 1

            max\_area = max(max\_area, h[top] \* width)

    while stack:

        top = stack.pop()

        width = index if not stack else index - stack[-1] - 1

        max\_area = max(max\_area, h[top] \* width)

    return max\_area

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

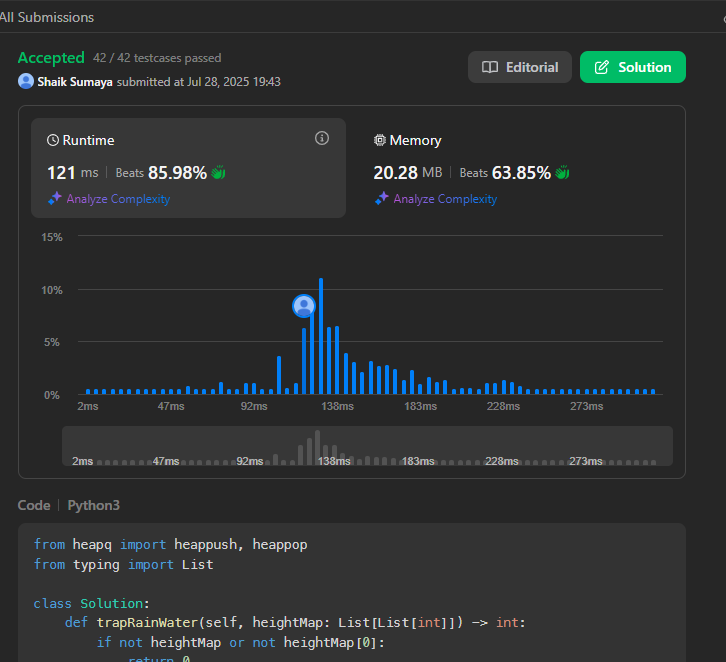
    n = int(input().strip())

    h = list(map(int, input().rstrip().split()))

    result = largestRectangle(h)

    fptr.write(str(result) + '\n')

    fptr.close()



from heapq import heappush, heappop

from typing import List

class Solution:

def trapRainWater(self, heightMap: List[List[int]]) -> int:

if not heightMap or not heightMap[0]:

return 0

m, n = len(heightMap), len(heightMap[0])

visited = [[False] \* n for \_ in range(m)]

heap = []

# Push all border cells into heap

for i in range(m):

heappush(heap, (heightMap[i][0], i, 0))

heappush(heap, (heightMap[i][n - 1], i, n - 1))

visited[i][0] = True

visited[i][n - 1] = True

for j in range(1, n - 1):

heappush(heap, (heightMap[0][j], 0, j))

heappush(heap, (heightMap[m - 1][j], m - 1, j))

visited[0][j] = True

visited[m - 1][j] = True

# Directions: up, down, left, right

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

res = 0

while heap:

height, x, y = heappop(heap)

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < m and 0 <= ny < n and not visited[nx][ny]:

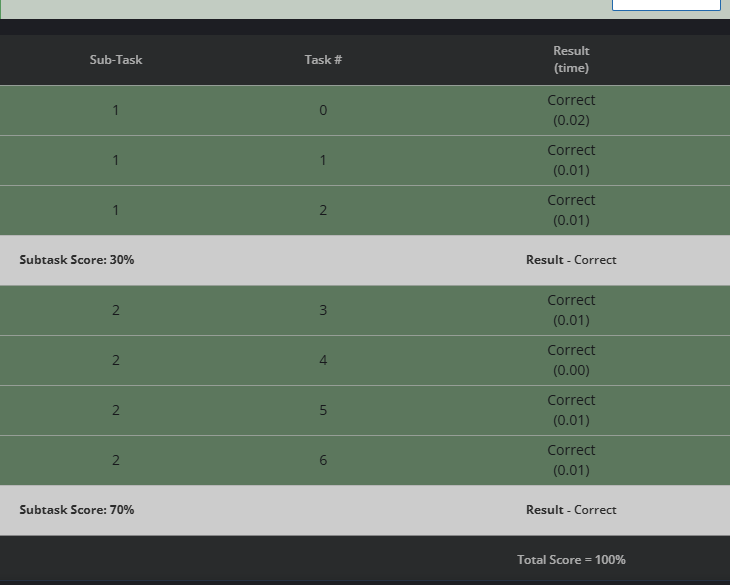
visited[nx][ny] = True

trapped = max(0, height - heightMap[nx][ny])

res += trapped

heappush(heap, (max(heightMap[nx][ny], height), nx, ny))

return res



# cook your dish here

from itertools import combinations

from math import comb

def count\_interesting\_subsequences(N, K, A):

A.sort() # Sorting helps get the smallest elements easily

min\_k\_elements = A[:K] # Smallest K elements will form the minimum sum

min\_sum = sum(min\_k\_elements)

# Count how many times each element in A appears

from collections import Counter

count = Counter(A)

# Count how many times the elements in the minimum sum appear

min\_elem\_count = Counter(min\_k\_elements)

# The number of ways to choose the elements for the minimum sum subsequence

result = 1

for val in min\_elem\_count:

result \*= comb(count[val], min\_elem\_count[val]) # Use binomial coefficient

return result

# Reading input for CodeChef

T = int(input())

for \_ in range(T):

N, K = map(int, input().split())

A = list(map(int, input().split()))

print(count\_interesting\_subsequences(N, K, A))