문제

길이 N의 정수 배열 α, β가 주어진다.

$$Result = \alpha[0] * \beta[N-1] + \alpha[1] * \beta[N-2] + .. + \alpha[N-1] * \beta[0]$$

α 배열을 재배열하여, 위의 식의 Result 값이 최소가 되게 끔 α 배열을 재배열 합니다. Result의 최솟값을 출력하는 프로그램을 작성해주세요. 단, β 배열은 고정 시키도록 합니다.

입력

첫째 줄에 길이 N이 주어집니다. (1 ≤ N ≤ 100)

둘째 줄에 α 배열의 N개의 수가 순서대로 주어지고, 셋째 줄에는 β 배열의 N개가 수가 주어집니다.

출력

첫째 줄에 최소가 되는 Result를 출력하세요.

제한사항

Heap을 이용한 풀이를 작성하세요.

입력 예시

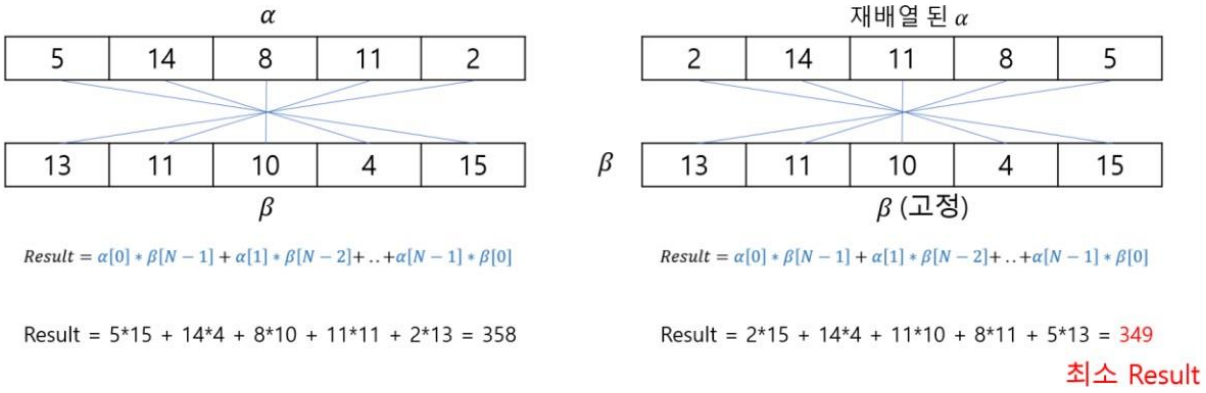
5

5 14 8 11 2

13 11 10 4 15

출력 예시

349



C / C++ 를 사용하시는 학생 분들은 아래의 폼을 참고해서 작성해 주셔야 기본적인 컴파일 에러를 방지할 수 있습니다. 또한 C 언어의 경우 표준 컴파일러에서는 scanf_s 또는 printf_s 등과 같이 "_s"를 붙이는 경 우 컴파일 에러가 발생하기 때 문에 "_s"를 제거한 scanf / printf 등의 함수를 사용하시기 바랍니다.

C:

```
ti n‹ luae ‹ st dl o. n ›

1nt   maln ( ) (
    y "  TG   i •/

    return 0;
```

C++:

```
r 1nc lude ‹1osrrean›
us ing namespace std;

int main() {

    ret urn B ;
```

모범 답안(python)

```
class MaxHeap(object):


    def __init__(self):
        self.queue = []


    def insert(self, n):
        self.queue.append(n)
        last_index = len(self.queue) - 1
        while 0 <= last_index:
            parent_index = self.parent(last_index)
            if 0 <= parent_index and self.queue[parent_index] < self.queue[last_index]:
                self.swap(last_index, parent_index)
                last_index = parent_index
            else:
                break


    def delete(self):
        last_index = len(self.queue) -1
        if last_index < 0:
            return -1
        self.swap(0, last_index)
        maxv = self.queue.pop()
        self.maxHeapify(0)
        return maxv


    def maxHeapify(self, i):
        left_index = self.leftchild(i)
        right_index = self.rightchild(i)
        max_index = i
```

```
        if left_index <= len(self.queue) -1 and self.queue[max_index] < self.queue[left_index]:
            max_index = left_index
        if right_index <= len(self.queue) -1 and self.queue[max_index] < self.queue[right_index]:
            max_index = right_index

        if max_index != i:
            self.swap(i, max_index)
            self.maxHeapify(max_index)
    def swap(self, i, parent_index):
        self.queue[i], self.queue[parent_index] = self.queue[parent_index], self.queue[i]
    def parent(self, index):
        return (index -1) // 2
    def leftchild(self, index):
        return index*2 + 1
    def rightchild(self, index):
        return index*2 + 2
    def root(self):
        return self.queue[0]

class MinHeap(object):
    def __init__(self):
        self.queue = []
    def insert(self, n):
        self.queue.append(n)
        last_index = len(self.queue) - 1
        while 0 <= last_index:
            parent_index = self.parent(last_index)
            if 0 <= parent_index and self.queue[parent_index] > self.queue[last_index]:
                self.swap(last_index, parent_index)
                last_index = parent_index
            else:
```

```
            break
    def delete(self):
        last_index = len(self.queue) -1
        if last_index < 0:
            return -1
        self.swap(0, last_index)
        minv = self.queue.pop()
        self.minHeapify(0)
        return minv

    def minHeapify(self, i):
        left_index = self.leftchild(i)
        right_index = self.rightchild(i)
        min_index = i

        if left_index <= len(self.queue) -1 and self.queue[min_index] > self.queue[left_index]:
            min_index = left_index
        if right_index <= len(self.queue) -1 and self.queue[min_index] > self.queue[right_index]:
            min_index = right_index

        if min_index != i:
            self.swap(i, min_index)
            self.minHeapify(min_index)

    def swap(self, i, parent_index):
        self.queue[i], self.queue[parent_index] = self.queue[parent_index], self.queue[i]

    def parent(self, index):
        return (index -1) // 2
```

```
    def leftchild(self, index):
        return index*2 + 1

    def rightchild(self, index):
        return index*2 + 2

    def root(self):
        return self.queue[0]

def solution(A, B):
    N = len(A)

    maxheap = MaxHeap()
    minheap = MinHeap()

    for i in range(0, N):
        minheap.insert(A[i])
        maxheap.insert(B[i])

    res = 0
    for i in range(0, N):
        a = minheap.delete()
        b = maxheap.delete()
        res = res + (a*b)

    return res
n = int(input())
a = list(map(int, input().split()))
b = list(map(int, input().split()))

print(solution(a,b))
```