Question - 1
Project Estimates

SCORE: **50 points**

Binary Search | Easy | Data Structures | Algorithms | Arrays | Problem Solving | Theme: Finance

A number of bids are being taken for a project. Determine the number of distinct pairs of project costs where their absolute difference is some target value. Two pairs are distinct if they differ in at least one value.

Example

n = 3

projectCosts = [1, 3, 5]

target= 2

There are *2* pairs *[1,3], [3,5] that* have the target difference *target = 2,* therefore a value of *2* is returned.

Function Description

Complete the function *countPairs* in the editor below.

countPairs has the following parameter(s):
  int projectCosts[n]: array of integers
  int target: the target difference

Returns
  int: the number of distinct pairs in *projectCosts* with an absolute difference of *target*

Constraints

- $5 \le n \le 10^5$
- $0 < projectCosts[i] \le 2 \times 10^9$
- Each *projectCosts[i]* is distinct, i.e. unique within *projectCosts*
- $1 \le target \le 10^9$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *projectCosts*.
The next *n* lines each contain an element *projectCosts[i]* where 0 ≤ i < n.
The next line contains the integer *target*, the target difference.

▼ Sample Case 0

Sample Input 0

```
STDIN      Function
-----      --------
5      →   projectCosts[] size n = 5
1      →   projectCosts = [1, 5, 3, 4, 2]
5
3
4
```

```
2
2       →   target = 2
```

Sample Output 0

```
3
```

Explanation 0

Count the number of pairs in *projectCosts* whose difference is *target = 2*. The following three pairs meet the criterion: *(1, 3)*, *(5, 3)*, and *(4, 2)*.

▼ Sample Case 1

Sample Input 1

```
STDIN           Function
-----           --------
10          →   projectCosts[] size n = 10
363374326   →   projectCosts = [363374326, 364147530, 61825163, 107306571, 128124602, 139946991, 428047635, 491595254, 879792181, 106926279]
364147530
61825163
107306571
128124602
139946991
428047635
491595254
879792181
106926279
1           →   target = 1
```

Sample Output 1

```
0
```

Explanation 1

Count the number of pairs in p*rojectCosts* whose difference is target *= 1*. Because no such pair of integers exists, return *0*.

▼ Sample Case 2

Sample Input 2

```
STDIN      Function
-----      --------
6      →   projectCosts[] size n = 6
2      →   projectCosts = [2, 4, 6, 8, 10, 12]
4
6
8
10
12
2      →   target = 2
```

Sample Output 2

```
5
```

Explanation 2

Count the number of pairs in *projectCosts* whose difference is *target = 2*. The following five pairs meet the criterion: *(2, 4)*, *(4, 6)*, *(6, 8)*, *(8, 10)*, and *(10, 12)*.

Question - 2
Merge 2 Arrays

SCORE: **50 points**

Easy | Data Structures | Algorithms | Arrays | Problem Solving | Interviewer Guidelines

Given two sorted arrays, merge them to form a single, sorted array with all items in non-decreasing order.

Example

a = [1, 2, 3]

b = [2, 5, 5]

Merge the arrays to create array *c* as follows:

```
        a[0] < b[0] → c = [a[0]] = [1]
        a[1] = b[0] → c = [a[0], b[0]] = [1, 2]
        a[1] < b[1] → c = [a[0], b[0], a[1]] = [1, 2, 2]
        a[2] < b[1] → c = [a[0], b[0], a[1], a[2]]= [1, 2, 2, 3]
        No more elements in a → c = [a[0], b[0], a[1], a[2], b[1], b[2]] = [1, 2, 2, 3, 5, 5]
```

Elements were alternately taken from the arrays in the order given, maintaining precedence.

Function Description

Complete the function *mergeArrays* in the editor below.

mergeArrays has the following parameter(s):

int a[n]: a sorted array of integers

int b[n]: a sorted array of integers

Returns

int[n]: an array of all the elements from both input arrays in non-decreasing order

Constraints

- $1 < n < 5 \times 10^5$
- $0 \le a[i], b[i] \le 10^9$, where $0 \le i < n$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the size of the array *a*.

The next *n* lines each contain an element *a[i]* where 0 ≤ i < n.

The next line contains an integer *n*, the size of the array *b*.

The next *n* lines each contain an element *b[i]* where 0 ≤ i < n.

▼ Sample Case 0

Sample Input 0

```
STDIN       Function
-----       --------
4      →    a[] size n = 4
1      →    a = [1, 5, 7, 7]
1
5
7
7
4      →    b[] size n = 4
0      →    b = [0, 1, 2, 3]
```

```
1
2
3
```

Sample Output 0

```
0
1
1
2
3
5
7
7
```

Explanation

The *mergedArray* function returns the following merged, non-decreasing array: [*0, 1, 1, 2, 3, 5, 7, 7*]

▼ Sample Case 1

Sample Input 1

```
STDIN      Function
-----      --------
5      →   a[] size n = 5
2      →   a = [2, 4, 5, 9, 9]
4
5
9
9
5      →   b[] size n = 5
0      →   b = [0, 1, 2, 3, 4]
1
2
3
4
```

Sample Output 1

```
0
1
2
2
3
4
4
5
9
9
```

Explanation

The *mergedArray* function returns the following merged, non-decreasing array: [*0, 1, 2, 2, 3, 4, 4, 5, 9, 9*]

Question - 3
River Records

SCORE: 50 points

Algorithms | Arrays | Problem Solving | Easy

Given an array of integers, without reordering, determine the maximum difference between any element and any prior smaller element. If there is never a lower prior element, return -1.

Example
arr = [5, 3, 6, 7, 4]

There are no earlier elements than *arr[0]*.
There is no earlier reading with a value lower than *arr[1]*.
There are two lower earlier readings with a value lower than *arr[2] = 6:*

- *arr[2] - arr[1] = 6 - 3 = 3*
- *arr[2] - arr[0] = 6 - 5 = 1*

There are three lower earlier readings with a lower value than *arr[3] = 7:*

- *arr[3] - arr[2] = 7 - 6 = 1*
- *arr[3] - arr[1] = 7 - 3 = 4*
- *arr[3] - arr[0] = 7 - 5 = 2*

There is one lower earlier reading with a lower value than *arr[4] = 4:*

- *arr[4] - arr[1] = 4 - 3 = 1*

The maximum trailing record is *arr[3] - arr[1] = 4*.

Example
arr = [4, 3, 2, 1]

No item in *arr* has a lower earlier reading, therefore return *-1*

Function Description
Complete the function *maximumTrailing* in the editor below.

maximumTrailing has the following parameter(s):
int arr[n]: an array of integers

Returns:
int: the maximum trailing difference, or *-1* if no element in *arr* has a lower earlier value

Constraints

- $1 \le n \le 2 \times 10^5$
- $-10^6 \le arr[i] \le 10^6$ and $0 \le i < n$

▼ Input Format For Custom Testing

Input from stdin will be processed as follows and passed to the function:

The first line contains a single integer, *n*, the number of elements in the array *arr*.
Each of the *n* subsequent lines contains a single integer, each an element *arr[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input 0

```
STDIN          Function
-----          --------
7       →      arr[] size n = 7
2       →      arr = [2, 3, 10, 2, 4, 8, 1]
3
10
2
4
8
1
```

Sample Output

```
8
```

Explanation

Differences are calculated as:

- *3 - [2] = [1]*
- *10 - [3, 2] = [7, 8]*
- *4 - [2, 3, 2] = [2, 1, 2]*
- *8 - [4, 2, 3, 2] = [4, 6, 5, 6]*

The maximum trailing difference is *10 - 2 = 8*.

▼ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
6       →      arr[] size n = 6
7       →      arr = [7, 9, 5, 6, 3, 2]
9
5
6
3
2
```

Sample Output

```
2
```

Explanation

Differences are calculated as:

- *9 - [7] = 2*
- *6 - [5] = 1*

The maximum trailing difference is *2*.

Question - 4

Triangle Or Not?

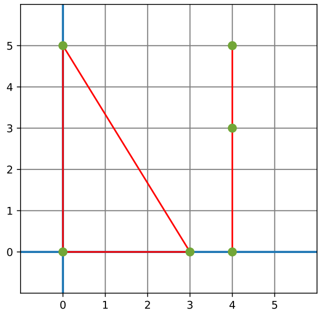
SCORE: 50 points

Loops | Easy | Arrays | Problem Solving

Given 3 lines, find out if they can form a non-degenerate triangle. If the 3 lines are placed with tips joined such that they form a triangle with non-zero angles at each vertex, then a non-degenerate triangle is formed.

Example

Lines with lengths *[3, 4, 5]* make a non-degenerate triangle (left), lines with lengths [2, 3, 5] make a degenerate triangle (right), and lines with lengths *[1, 1, 5]* cannot form a triangle.



Function Description

Complete the function *triangleOrNot* in the editor below.

triangleOrNot has the following parameter(s):

int a[n]: an array where each index *i* describes the length of side *a* for triangle *i*

int b[n]: an array representing lengths of sides *b[i]*

int c[n]: an array representing lengths of sides *c[i]*

Returns:

String arr[n]: an array where the value at each index *i* is '*Yes*' if *a[i]*, *b[i]*, and *c[i]* can form a non-degenerate triangle, otherwise, the value is *'No'*

Constraints

- $1 \le n \le 10^5$
- $1 \le a[i], b[i], c[i] \le 10^3$, where $0 \le i < n$

▼ Input Format For Custom Testing

Locked stub code reads input from stdin and passes it to the function.

The first line contains an integer, *n*, denoting the number of elements in *a[]*.
Each of the next *n* lines contains an integer *a[i]*.
The next line contains an integer, *n*, denoting the number of elements in *b[]*.
Each of the next *n* lines contains an integer *b[i]*.
The next line contains an integer, *n*, denoting the number of elements in *c[]*.
Each of the next *n* lines contains an integer *c[i]*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
3      →  a[] size n = 3
7      →  a = [7, 10, 7]
10
7
3      →  b[] size n = 3
2      →  b = [2, 3, 4]
3
4
3      →  c[] size n = 3
2      →  c = [2, 7, 4]
7
4
```

Sample Output

```
No
No
Yes
```

Explanation

Check the following $n = 3$ possible triangles using the values given by $a = [7, 10, 7]$, $b = [2, 3, 4]$, and $c = [2, 7, 4]$:

0. $a[0] = 7$, $b[0] = 2$, and $c[0] = 2$ do not form a valid, non-degenerate triangle, so store '*No*' in index *0* of the return array.
1. $a[1] = 10$, $b[1] = 3$, and $c[1] = 7$ do not form a valid, non-degenerate triangle, so store '*No*' in index *1* of the return array.
2. $a[2]= 7$, $b[2]= 4$, and $c[2] = 4$ do form a valid, non-degenerate triangle, so store '*Yes*' in index *2* of the return array.

Then return the array *['No', 'No', 'Yes']* as the answer.

Question - 5
Cutting Metal Surplus

SCORE: 75 points

Data Structures | Medium | Algorithms | Arrays | Problem Solving

The owner of a construction company has a surplus of rods of arbitrary lengths. A local contractor offers to buy any of the surplus, as long as all the rods have the same exact integer length, referred to as *saleLength*. The number of sellable rods can be increased by cutting each rod zero or more times, but each cut has a cost denoted by *costPerCut*. After all cuts have been made, any leftover rods having a length other than *saleLength* must be discarded for no profit. The owner's total profit for the sale is calculated as:

$$totalProfit = totalUniformRods \times saleLength \times salePrice - totalCuts \times costPerCut$$

where *totalUniformRods* is the number of sellable rods, *salePrice* is the per unit length price that the contractor agrees to pay, and *totalCuts* is the total number of times the rods needed to be cut.

Example

lengths = [30, 59, 110]

costPerCut = 1

salePrice = 10 per unit length

The following are tests based on lengths that are factors of *30*, the length of the shortest bar. Factors of other lengths might also be tested, but this demonstrates the methodology.

```
                            Cuts
                    ---------------------
Length  Rod     Extra   Regular Total   Pieces
----------------------------------------------
30      30      0       0       0       1
        59      1       0       1       1
        110     1       2       3       3
        Revenue = (10*5*30)-(4*1) = 1496
15      30      0       1       1       2
        59      1       2       3       3
        110     1       6       7       7
        Revenue = (10*12*15)-(11*1) = 1789
10      30      0       2       2       3
        59      1       4       5       5
        110     0       10      10      11
        Revenue = (10*19*10)-(17*1) = 1883
6       30      0       4       4       5
        59      1       8       9       9
        110     1       17      18      18
        Revenue = (10*32*6)-(31*1) = 1889
5       30      0       5       5       6
        59      1       10      11      11
        110     0       21      21      21
        Revenue = (10*39*5)-(37*1) = 1913
3       30      0       9       9       10
        59      1       18      19      19
        110     1       35      36      36
        Revenue = (10*65*3)-(64*1) = 1886
```

Working through the first stanza, *length = 30*, it is the same length as the first rod, so no cuts are required and there is *1* piece. For the second rod, cut and discard the excess *29* unit rod. No more cuts are necessary and another *1* piece is left to sell. Cut *20* units off the *110* unit rod to discard leaving *90* units, then make two more cuts to have *3* more pieces to sell. Finally sell *5 totalUniformRods* , *saleLength = 30* at *salePrice = 10* per unit length for *1500*. The cost to produce was *totalCuts = 4* times *costPerCut = 1* per cut, or *4*. Total revenue = *1500-4=1496*. The maximum revenue among these tests is obtained at length *5* for *1913*.

Function Description

Complete the function *maxProfit* in the editor below.

maxProfit has the following parameter(s):

costPerCut: cost to make a cut

salePrice: per unit length sales price

lengths[n]: integer rod lengths

Returns:

int: maximum possible profit

Constraints

- $1 \le n \le 50$
- $1 \le lengths[i] \le 10^4$
- $1 \le salePrice, costPerCut \le 1000$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer, *costPerCut*.

The second line contains an integer, *salePrice*.

The next line contains an integer *n*, the size of the array *lengths*.

Each of the next *n* lines contains an integer *lengths[i]* where $0 \le i < n$.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
1       →  costPerCut = 1
10      →  salePrice = 10
3       →  lengths[] size n = 3
26      →  lengths = [26, 103, 59]
103
59
```

Sample Output

```
1770
```

Explanation

Since *costPerCut = 1* is very inexpensive, a large number of cuts can be made to reduce the number of wasted pieces. The optimal rod length for maximizing profit is *6*, and the rods are cut as shown:

- `lengths[0] = 26`: Cut off a piece of length *2* and discard it, resulting in a rod of length *24*. Then, cut this rod into *4* pieces of length *6*.
- `lengths[1] = 103`: Cut off a piece of length *1* and discard it, resulting in a rod of length *102*. Then, cut this rod into *17* pieces of length *6*.
- `lengths[2] = 59`: Cut off a piece of length *5* and discard it, resulting in a rod of length *54*. Then, cut this rod into *9* pieces of length *6*.

After performing $totalCuts = (1 + 3) + (1 + 16) + (1 + 8) = 30$ cuts, there are $totalUniformRods = 4 + 17 + 9 = 30$ pieces of length $saleLength = 6$ that can be sold at $salePrice = 10$. This yields a total profit of $salePrice \times totalUniformRods \times saleLength - totalCuts \times costPerCut = 10 \times 30 \times 6 - 30 \times 1 = 1770$.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-----     -----
100    →  costPerCut = 100
10     →  salePrice = 10
3      →  lengths[] size n = 3
26     →  lengths = [26, 103, 59]
103
59
```

Sample Output

```
1230
```

Explanation

Since *costPerCut = 100*, cuts are expensive and must be minimal. The optimal rod length for maximizing profit is *51*, and the rods are cut as shown:

- `lengths[0] = 26`: Discard this rod entirely.
- `lengths[1] = 103`: Cut off a piece of length *1* and discard it, resulting in a rod of length *102*. Then, cut this rod into *2* pieces of length *51*.
- `lengths[2] = 59`: Cut off a piece of length *8* and discard it, resulting in a rod of length *51*.

After performing *totalCuts = (0) + (1 + 1) + (1) = 3* cuts, there are *totalUniformRods = 0 + 2 + 1 = 3* pieces of length *saleLength = 51* that can be sold at *salePrice = 10* each. This yields a total profit of *salePrice × totalUniformRods × saleLength − totalCuts × costPerCut = 10 × 3 × 51 − 3 × 100 = 1230*.