10 - Searching & Sorting

Ex. No. : 10.1 **Date: 30.05.2024**

Register No.: 230701264 **Name Ridhanya J**

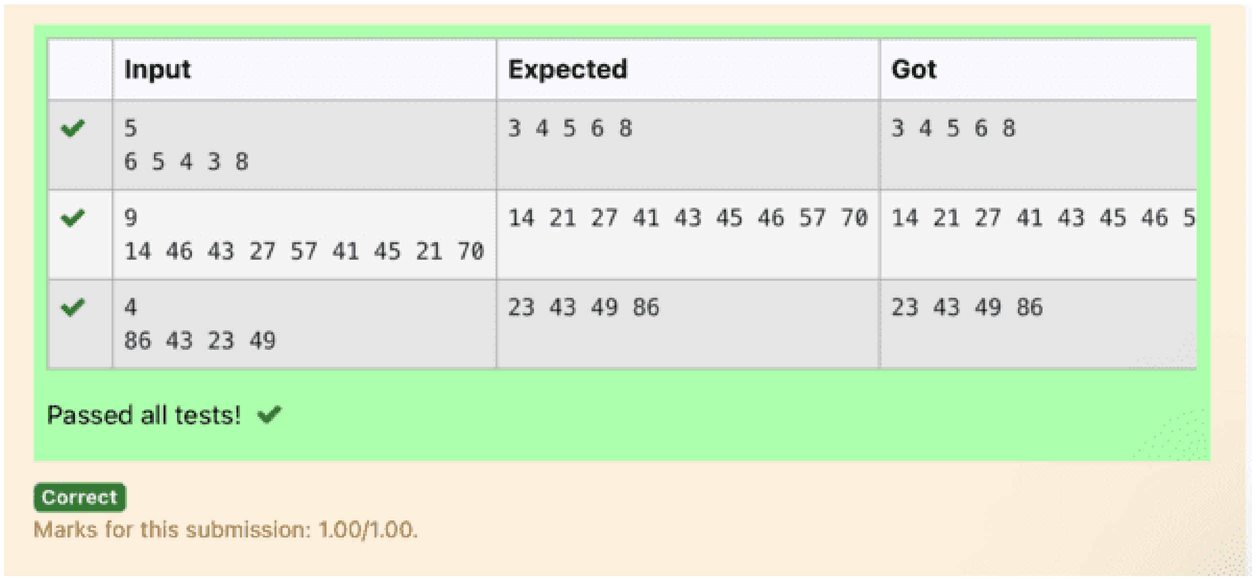
Merge Sort

Write a Python program to sort a list of elements using the merge sort algorithm.

For example:

Input	Result
5 6 5 4 3 8	3 4 5 6 8

```
a=int(input())
l=[]
l.extend(input().split())
for i in range(a-1):
    for j in range(a-1):
        if(int(l[j])>int(l[j+1])):
            t=int(l[j])
            l[j]=int(l[j+1])
            l[j+1]=t
for i in range(a):
    print(int(l[i]),end=" ")
```



	Input	Expected	Got
✓	5 6 5 4 3 8	3 4 5 6 8	3 4 5 6 8
✓	9 14 46 43 27 57 41 45 21 70	14 21 27 41 43 45 46 57 70	14 21 27 41 43 45 46 5
✓	4 86 43 23 49	23 43 49 86	23 43 49 86

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.2 **Date: 30.05.2024**

Register No.: 230701264 **Name: Ridhanya J**

Bubble Sort

Given an listof integers, sort the array in ascending order using the *Bubble Sort* algorithm above. Once sorted, print the following three lines:

1. List is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.
2. First Element: firstElement, the *first* element in the sorted list.
3. Last Element: lastElement, the *last* element in the sorted list.

For example, given a worst-case but small array to sort: a=[6,4,1]. It took 3 swaps to sort the array. Output would be

```
Array is sorted in 3 swaps.
First Element: 1
Last Element: 6
```

Input Format

The first line contains an integer,n , the size of the list a . The second line contains n, space-separated integers a[i].

Constraints

- $2<=n<=600$
- $1<=a[i]<=2x10^6$.

Output Format

You must print the following three lines of output:

1. List is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.
2. First Element: firstElement, the *first* element in the sorted list.
3. Last Element: lastElement, the *last* element in the sorted list.

Sample Input 0

```
3
1 2 3
```

Sample Output 0

```
List is sorted in 0 swaps.
First Element: 1
```

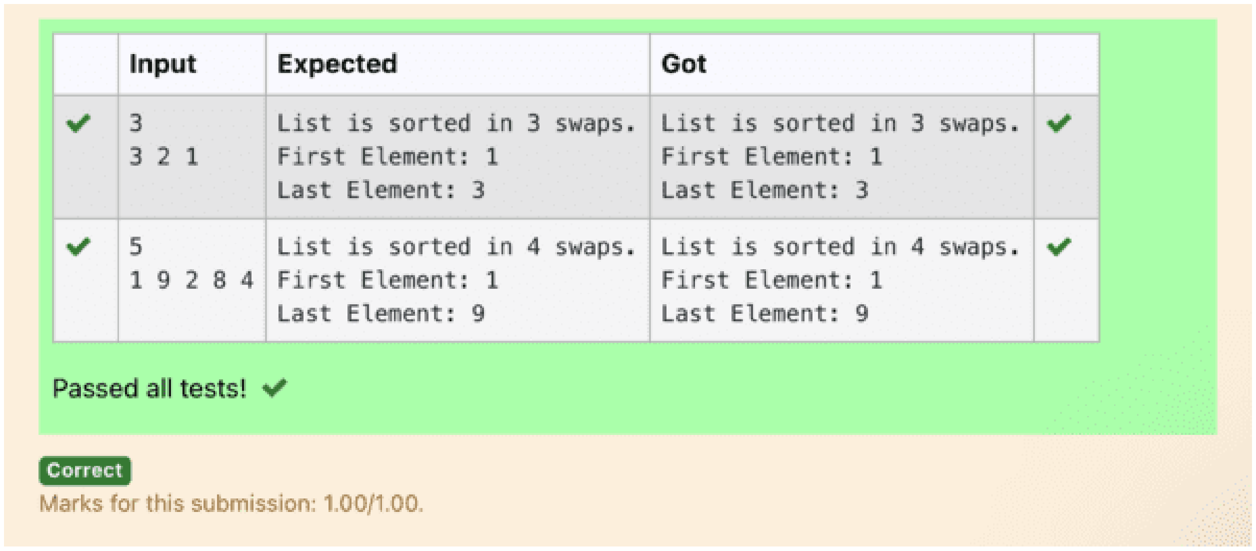
Last Element: 3

For example:

Input	Result
3 3 2 1	List is sorted in 3 swaps. First Element: 1 Last Element: 3
5 1 9 2 8 4	List is sorted in 4 swaps. First Element: 1 Last Element: 9

```
def bubble_sort(arr):
    n = len(arr)
    swaps = 0

    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j + 1]:
                # Swap elements
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swaps += 1

    return swaps

# Input the size of the list
n = int(input())

# Input the list of integers
arr = list(map(int, input().split()))

# Perform bubble sort and count the number of swaps
num_swaps = bubble_sort(arr)
```

```
# Print the number of swaps
print("List is sorted in", num_swaps, "swaps.")

# Print the first element
print("First Element:", arr[0])

# Print the last element
print("Last Element:", arr[-1])
```



	Input	Expected	Got	
✓	3 3 2 1	List is sorted in 3 swaps. First Element: 1 Last Element: 3	List is sorted in 3 swaps. First Element: 1 Last Element: 3	✓
✓	5 1 9 2 8 4	List is sorted in 4 swaps. First Element: 1 Last Element: 9	List is sorted in 4 swaps. First Element: 1 Last Element: 9	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.3 **Date: 30.05.2024**

Register No.: 230701264 **Name: Ridhanya J**

Peak Element

Given an list, find peak element in it. A peak element is an element that is greater than its neighbors.

An element a[i] is a peak element if

A[i-1] <= A[i] >=a[i+1] for middle elements. [0<i<n-1]

A[i-1] <= A[i] for last element [i=n-1]

A[i]>=A[i+1] for first element [i=0]

Input Format

The first line contains a single integer n , the length of A .
The second line contains n space-separated integers,A[i].

Output Format

Print peak numbers separated by space.

Sample Input

5

8 9 10 2 6

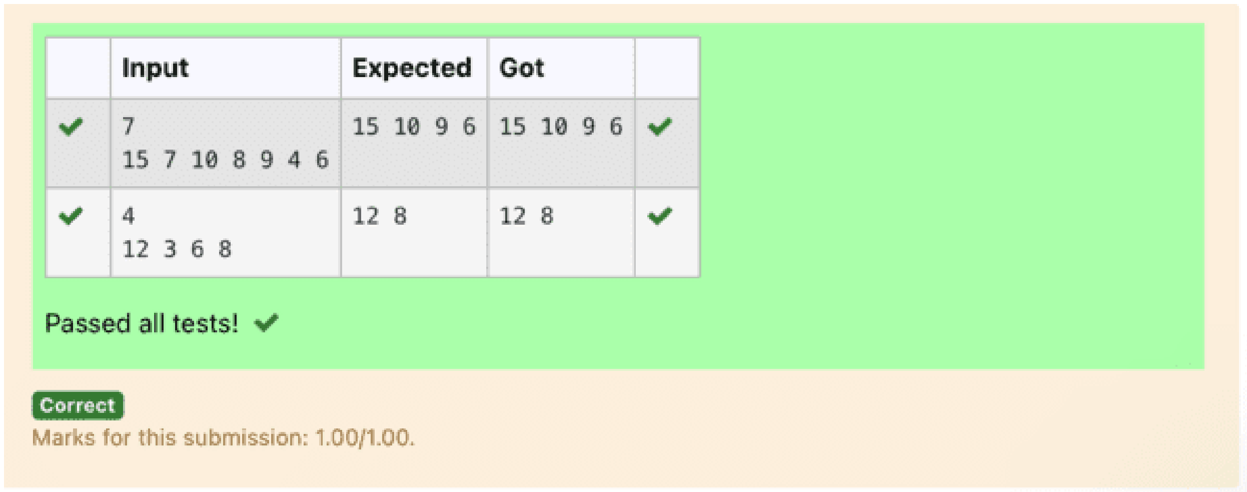
Sample Output

10 6

For example:

Input	Result
4 12 3 6 8	12 8

```
def find_peak(arr):
    peak_elements = []
    # Check for the first element
    if arr[0] >= arr[1]:
        peak_elements.append(arr[0])

    # Check for middle elements
    for i in range(1, len(arr) - 1):
        if arr[i - 1] <= arr[i] >= arr[i + 1]:
            peak_elements.append(arr[i])

    # Check for the last element
    if arr[-1] >= arr[-2]:
        peak_elements.append(arr[-1])

    return peak_elements

# Input the length of the list
n = int(input())

# Input the list of integers
arr = list(map(int, input().split()))

# Find peak elements and print the result
peak_elements = find_peak(arr)
print(*peak_elements)
```



	Input	Expected	Got	
✓	7 15 7 10 8 9 4 6	15 10 9 6	15 10 9 6	✓
✓	4 12 3 6 8	12 8	12 8	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.4 **Date: 30.05.2024**

Register No.: 230701264 **Name: Ridhanya J**

Binary Search

Write a Python program for binary search.

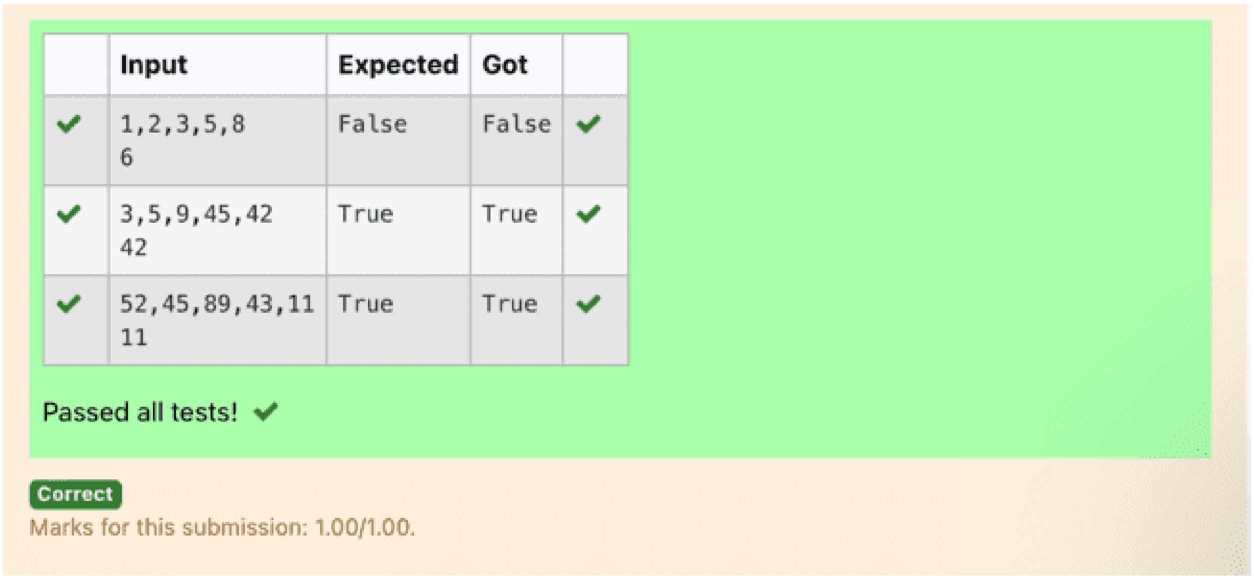
For example:

Input	Result
1 2 3 5 8 6	False
3 5 9 45 42 42	True

a = input().split(",")

b = input()

print(b in a)



	Input	Expected	Got	
✓	1,2,3,5,8 6	False	False	✓
✓	3,5,9,45,42 42	True	True	✓
✓	52,45,89,43,11 11	True	True	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.5 **Date: 30.05.2024**

Register No.: 230701264 **Name: Ridhanya J**

Frequency of Elements

To find the frequency of numbers in a list and display in sorted order.

Constraints:

1<=n, arr[i]<=100

Input:

1 68 79 4 90 68 1 4 5

output:

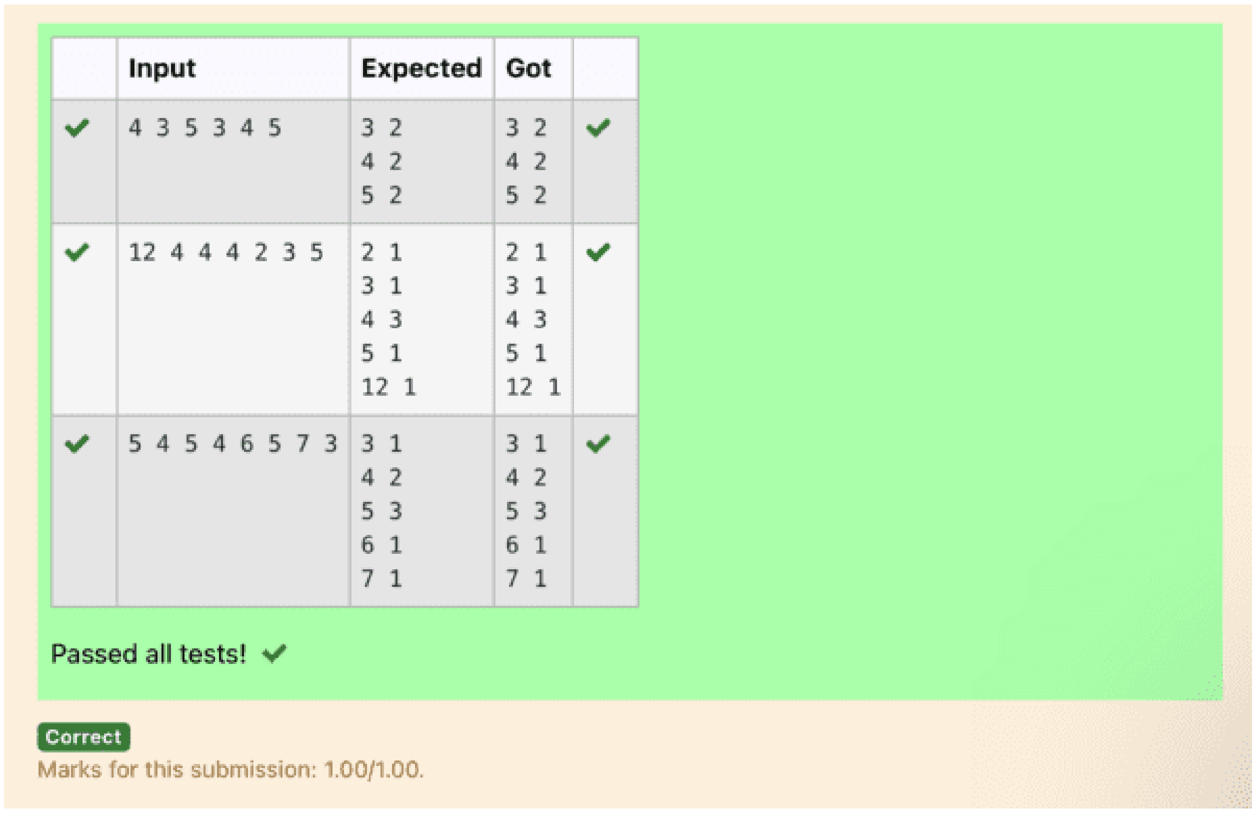
```
 1 2
 4 2
 5 1
 68 2
 79 1
90 1
```

For example:

Input	Result

4 3 5 3 4 5	3 2
	4 2
	5 2

```
def count_frequency(arr):
    frequency = {}

    # Count the frequency of each number in the list
    for num in arr:
        frequency[num] = frequency.get(num, 0) + 1
    # Sort the dictionary based on keys
    sorted_frequency = sorted(frequency.items())

    # Print the frequency of each number
    for num, freq in sorted_frequency:
        print(num, freq)

# Input the list of numbers
arr = list(map(int, input().split()))

# Count the frequency and print the result
count_frequency(arr)
```



	Input	Expected	Got	
✓	4 3 5 3 4 5	3 2 4 2 5 2	3 2 4 2 5 2	✓
✓	12 4 4 4 2 3 5	2 1 3 1 4 3 5 1 12 1	2 1 3 1 4 3 5 1 12 1	✓
✓	5 4 5 4 6 5 7 3	3 1 4 2 5 3 6 1 7 1	3 1 4 2 5 3 6 1 7 1	✓

Passed all tests! ✓

Correct
Marks for this submission: 1.00/1.00.

WEEK 11:

1.

Write a Python program that asks the user for their age and prints a message based on the age. Ensure that the program handles cases where the input is not a valid integer.

Input Format: A single line input representing the user's age.

Output Format: Print a message based on the age or an error if the input is invalid.

For example:

Input	Result
twenty	Error: Please enter a valid age.
25	You are 25 years old.
-1	Error: Please enter a valid age.

```
try:
  a=input()
  if(len(a)==0):
    print("Error: Please enter a valid age.")
  elif a.isnumeric():
    print("You are",a,"years old.")
  else:
    print("Error: Please enter a valid age.")
except:
  print("Error: Please enter a valid age.")
```

OUTOUT:

	Input	Expected	Got	
	twenty	Error: Please enter a valid age.	Error: Please enter a valid age.	
	25	You are 25 years old.	You are 25 years old.	
	-1	Error: Please enter a valid age.	Error: Please enter a valid age.	
	150	You are 150 years old.	You are 150 years old.	
		Error: Please enter a valid age.	Error: Please enter a valid age.	

Passed all tests!

Correct

2.

Problem Description:

Write a Python program that asks the user for their age and prints a message based on the age. Ensure that the program handles cases where the input is not a valid integer.

Input Format:

A single line input representing the user's age.

Output Format:

Print a message based on the age or an error if the input is invalid.

For example:

Input	Result
25	You are 25 years old.
rec	Error: Please enter a valid age.
-5	Error: Please enter a valid age.

```
try:
   a=input()
   if(len(a)==0):
      print("Error: Please enter a valid age.")
   elif a.isnumeric():
        print("You are",a,"years old.")
    else:
        print("Error: Please enter a valid age.")
except:
    print("Error: Please enter a valid age.")
```

OUTPUT:

	Input	Expected	Got	
	25	You are 25 years old.	You are 25 years old.	
	rec	Error: Please enter a valid age.	Error: Please enter a valid age.	
	!@#	Error: Please enter a valid age.	Error: Please enter a valid age.	

Passed all tests!

Correct

3.

Problem Description:

Write a Python script that asks the user to enter a number within a specified range (e.g., 1 to 100). Handle exceptions for invalid inputs and out-of-range numbers.

Input Format:

User inputs a number.

Output Format:

Confirm the input or print an error message if it's invalid or out of range.

For example:

Input	Result
1	Valid input.
101	Error: Number out of allowed range
rec	Error: invalid literal for int()

```
def main():

    min_range = 1
```

```
 max_range = 100


 try:
     num = int(input())
     if num < min_range or num > max_range:
         print("Error: Number out of allowed range")
     else:
         print("Valid input.")
 except ValueError:
     print("Error: invalid literal for int()")



if __name__ == "__main__":
```

OUTPUT:

	Input	Expected	Got	
	1	Valid input.	Valid input.	
	100	Valid input.	Valid input.	
	101	Error: Number out of allowed range	Error: Number out of allowed range	

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

4.

Develop a Python program that safely performs division between two numbers provided by the user. Handle exceptions like division by zero and non-numeric inputs.

Input Format: Two lines of input, each containing a number.

Output Format: Print the result of the division or an error message if an exception occurs.

For example:

Input	Result
10 2	5.0
10 0	Error: Cannot divide or modulo by zero.
ten 5	Error: Non-numeric input provided.

```
def main():
    try:
        num1 = float(input())
        num2 = float(input())


        division_result = num1 / num2
        modulo_result = num1 % num2


        print(division_result)


    except ValueError:
        print("Error: Non-numeric input provided.")
    except ZeroDivisionError:
        print("Error: Cannot divide or modulo by zero.")




if __name__ == "__main__":
    main()
```

OUTPUT:

	Input	Expected	Got	
	10 2	5.0	5.0	
	10 0	Error: Cannot divide or modulo by zero.	Error: Cannot divide or modulo by zero.	
	ten 5	Error: Non-numeric input provided.	Error: Non-numeric input provided.	

Passed all tests!

Correct

5.

Problem Description:

Develop a Python program that safely calculates the square root of a number provided by the user. Handle exceptions for negative inputs and non-numeric inputs.

Input Format:

User inputs a number.

Output Format:

Print the square root of the number or an error message if an exception occurs.

For example:

Input	Result
16	The square root of 16.0 is 4.00
-4	Error: Cannot calculate the square root of a negative number.
rec	Error: could not convert string to float

```
try:
  a=float(input())
  if(a<0):
    print("Error: Cannot calculate the square root of a negative number.")
    else:
        print("The square root of",a,"is {:.2f}".format(a**0.5))
except:
    print("Error: could not convert string to float")
```

OUTPUT:

	Input	Expected	Got	
	16	The square root of 16.0 is 4.00	The square root of 16.0 is 4.00	
	0	The square root of 0.0 is 0.00	The square root of 0.0 is 0.00	
	-4	Error: Cannot calculate the square root of a negative number.	Error: Cannot calculate the square root of a negative number.	

Passed all tests!

Correct

WEEK 12:

1.

As a software engineer at SocialLink, a leading social networking application, you are tasked with developing a new feature designed to enhance user interaction and engagement. The company aims to introduce a system where users can form connections based on shared interests and activities. One of the feature's components involves analyzing pairs of users based on the activities they've participated in, specifically looking at the numerical difference in the number of activities each user has participated in.

Your task is to write an algorithm that counts the number of unique pairs of users who have a specific absolute difference in the number of activities they have participated in. This algorithm will serve as the backbone for a larger feature that recommends user connections based on shared participation patterns.

Problem Statement

Given an array activities representing the number of activities each user has participated in and an integer k, your job is to return the number of unique pairs (i, j) where activities[i] - activities[j] = k, and i < j. The absolute difference between the activities should be exactly k.

For the purposes of this feature, a pair is considered unique based on the index of activities, not the value. That is, if there are two users with the same number of activities, they are considered distinct entities.

Input Format

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

The second line contains n space-separated integers, nums[i].

The third line contains an integer, k.

Output Format

Return a single integer representing the number of unique pairs (i, j)

where | nums[i] - nums[j] | = k and i < j.

Constraints:

$1 \leq n \leq 10^5$

$-10^4 \leq nums[i] \leq 10^4$

$0 \leq k \leq 10^4$

For example:

Input	Result

5
1 3 1 5 4
0
	1

4
1 2 2 1
1
	4

```
def count_pairs_with_difference_k(activities, k):

    count = 0

    n = len(activities)

    for i in range(n):
```

2.

```
        for j in range(i + 1, n):
            if abs(activities[i] - activities[j]) == k:
                count += 1
    return count

# Reading input
n = int(input())
activities = list(map(int, input().split()))
k = int(input())

# Calling function and printing the result
print(count_pairs_)
```

OUTPUT:

	Input	Expected	Got	
	4 1 2 3 4 1	3	3	
	5 1 3 1 5 4 0	1	1	
	4 1 2 2 1 1	4	4	

Passed all tests!

Correct

2.

Given an integer `n`, print `true` *if it is a power of four. Otherwise, print* `false`.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4`^{x}.

For example:

Input	**Result**
16	True
5	False

```
def is_power_of_four(n):
    if n <= 0:
        return False
    while n > 1:
        if n % 4 != 0:
            return False
        n //= 4
    return True


# Test the function
n = int(input())
print(is_power_of_four(n))
```

OUTPUT:

	Input	**Expected**	**Got**	
	16	True	True	
	5	False	False	
	1	True	True	
	-1	False	False	

Passed all tests!

Correct

3. Background:

Dr. John Wesley maintains a spreadsheet with student records for academic evaluation. The spreadsheet contains various data fields including student IDs, marks, class names, and student names. The goal is to develop a system that can calculate the average marks of all students listed in the spreadsheet.

Problem Statement:

Create a Python-based solution that can parse input data representing a list of students with their respective marks and other details, and compute the average marks. The input may present these details in any order, so the solution must be adaptable to this variability.

Input Format:

The first line contains an integer N, the total number of students.

The second line lists column names in any order (ID, NAME, MARKS, CLASS).

The next N lines provide student data corresponding to the column headers.

Output Format:

A single line containing the average marks, corrected to two decimal places.

Constraints:

$1 \le N \le 100$

Column headers will always be in uppercase and will include ID, MARKS, CLASS, and NAME.

Marks will be non-negative integers.

For example:

Input	Result

Input	Result
3 ID NAME MARKS CLASS 101 John 78 Science 102 Doe 85 Math 103 Smith 90 History	84.33
3 MARKS CLASS NAME ID 78 Science John 101 85 Math Doe 102 90 History Smith 103	84.33

```
def calculate_average_marks(data):
    total_marks = 0
    num_students = 0

    for student in data:
        if 'MARKS' in student:
            total_marks += int(student['MARKS'])
            num_students += 1

    if num_students == 0:
        return 0

    return total_marks / num_students

# Read input
N = int(input())
columns = input().split()

# Initialize data structure to store student records
```

```
students = []

# Read student data
for _ in range(N):
    student_data = input().split()
    student_record = {columns[i]: student_data[i] for i in range(len(columns))}
    students.append(student_record)

# Calculate average marks
average_marks = calculate_average_marks(students)

# Print average marks with two decimal places
print("{:.2f}".format(average_marks))
```

OUTPUT:

	Input	Expected	Got	
	3 ID NAME MARKS CLASS 101 John 78 Science 102 Doe 85 Math 103 Smith 90 History	84.33	84.33	
	3 MARKS CLASS NAME ID 78 Science John 101 85 Math Doe 102 90 History Smith 103	84.33	84.33	

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

4.

Background:

Raghu owns a shoe shop with a varying inventory of shoe sizes. The shop caters to multiple customers who have specific size requirements and are willing to pay a designated amount for their desired shoe size. Raghu needs an efficient system to manage his inventory and calculate the total revenue generated from sales based on customer demands.

Problem Statement:

Develop a Python program that manages shoe inventory and processes sales transactions to determine the total revenue generated. The program should handle inputs of shoe sizes available in the shop, track the number of each size, and match these with customer purchase requests. Each transaction should only proceed if the desired shoe size is in stock, and the inventory should update accordingly after each sale.

Input Format:

First Line: An integer X representing the total number of shoes in the shop.

Second Line: A space-separated list of integers representing the shoe sizes in the shop.

Third Line: An integer N representing the number of customer requests.

Next N Lines: Each line contains a pair of space-separated values:

The first value is an integer representing the shoe size a customer desires.

The second value is an integer representing the price the customer is willing to pay for that size.

Output Format:

Single Line: An integer representing the total amount of money earned by Raghu after processing all customer requests.

Constraints:

$1 \le X \le 1000$ — Raghu's shop can hold between 1 and 1000 shoes.

Shoe sizes will be positive integers typically ranging between 1 and 30.

$1 \le N \le 1000$ — There can be up to 1000 customer requests in a single batch.

The price offered by customers will be a positive integer, typically ranging from $5 to $100 per shoe.

For example:

Input	Result

Input	Result

10
2 3 4 5 6 8 7 6 5
18
6
6 55
6 45
6 55
4 40
18 60
10 50
	200

5
5 5 5 5 5
5
5 10
5 10
5 10
5 10
5 10
	50

Answer:(penalty regime: 0 %)

```
class ShoeInventory:
    def __init__(self, shoe_sizes):
        self.inventory = {size: 0 for size in shoe_sizes}

    def add_shoes(self, size, quantity):
        self.inventory[size] += quantity

    def sell_shoes(self, size):
        if self.inventory.get(size, 0) > 0:
            self.inventory[size] -= 1
            return True
        return False

class TransactionManager:
    def __init__(self, shoe_inventory):
        self.shoe_inventory = shoe_inventory
        self.total_revenue = 0

    def process_transactions(self, customer_requests):
        for size, price in customer_requests:
            if self.shoe_inventory.sell_shoes(size):
                self.total_revenue += price

# Read input
X = int(input())  # Total number of shoes
shoe_sizes = list(map(int, input().split()))  # Shoe sizes available
N = int(input())  # Number of customer requests

# Initialize shoe inventory
inventory = ShoeInventory(shoe_sizes)

# Populate initial inventory
for size in shoe_sizes:
    inventory.add_shoes(size, X // len(shoe_sizes))

# Initialize transaction manager
transaction_manager = TransactionManager(inventory)

# Process transactions
for _ in range(N):
    size, price = map(int, input().split())
    transaction_manager.process_transactions([(size, price)])
```

Print total revenue

print(transaction_manager.total_revenue)

OUTPUT:

	Input	Expected	Got	
	10 2 3 4 5 6 8 7 6 5 18 6 6 55 6 45 6 55 4 40 18 60 10 50	200	200	
	5 5 5 5 5 5 5 5 10 5 10 5 10 5 10 5 10	50	50	
	4 4 4 6 6 5 4 25 4 25 6 30 6 55 6 55	135	135	

Passed all tests!

Correct

Marks for this submission: 1.00/1.00.

5.

Background:

Rose manages a personal library with a diverse collection of books. To streamline her library management, she needs a program that can categorize books based on their genres, making it easier to find and organize her collection.

Problem Statement:

Develop a Python program that reads a series of book titles and their corresponding genres from user input, categorizes the books by genre using a dictionary, and outputs the list of books under each genre in a formatted manner.

Input Format:

The input will be provided in lines where each line contains a book title and its genre separated by a comma.

Input terminates with a blank line.

Output Format:

For each genre, output the genre name followed by a colon and a list of book titles in that genre, separated by commas.

Constraints:

Book titles and genres are strings.

Book titles can vary in length but will not exceed 100 characters.

Genres will not exceed 50 characters.

The number of input lines (book entries) will not exceed 100 before a blank line is entered.

For example:

Input	Result

`Introduction to Programming, Programming`
`Advanced Calculus, Mathematics`
	`Programming: Introduction to Programming`
	`Mathematics: Advanced Calculus`

`Fictional Reality, Fiction`
`Another World, Fiction`
	`Fiction: Fictional Reality, Another World`

```
def categorize_books_sorted():

  books = {}



  while True:

    try:

      user_input = input().strip()
```

```
        except EOFError:
            break

        if not user_input:
            break

        title, genre = user_input.split(',')
        genre = genre.strip()

        if genre in books:
            books[genre].append(title)
        else:
            books[genre] = [title]

    for genre in sorted(books.keys()):
        print(f"{genre}: {', '.join(books[genre])}")

categorize_books_sorted()
```

OUTPUT:

	Input	Expected	Got	

	`Introduction to Programming, Programming`
	`Advanced Calculus, Mathematics`
		`Programming: Introduction to Programming`
		`Mathematics: Advanced Calculus`
			`Programming: Introduction to Programming`
			`Mathematics: Advanced Calculus`	

	`Fictional Reality, Fiction`
	`Another World, Fiction`
		`Fiction: Fictional Reality, Another World`	`Fiction: Fictional Reality, Another World`	

Passed all tests!

Correct