Play with Numbers

You are given an array of n numbers and q queries. For each query you have to print the floor of the expected value(mean) of the subarray from L to R.

- Input
  - First line contains two integers N and Q denoting number of array elements and number of queries.
  - Next line contains N space seperated integers denoting array elements.
  - Next Q lines contain two integers L and R(indices of the array).
- Output
  - print a single integer denoting the answer.

In [2]:
```
n = input().split()
n[0],n[1] = int(n[0]),int(n[1])

# Read array element
a = input().split()
sum = []

# Cumulative Sum
for i in range(0,n[0]):
    if i == 0:
        sum.append(int(a[i]))
    else:
        sum.append(int(sum[i-1]) + int(a[i]))


del a

# Read each query and calculate the average
for k in range(0,n[1]):
    inq = input().split()
    i = int(inq[0])
    j = int(inq[1])
    if i > 1:
        print((sum[j-1] - sum[i-2]) // (j-i+1))
    else:
        print(sum[j-1] // (j-i+1))
```

```
5 3
1 2 3 4 5
1 4
2
2 4
3
2 5
3
```

In [ ]:

Special Numbers

A special number is defined as a number which has atleast P distinct prime factors. Write a program to determine whether a number N is a special number.

Input Format

- First Line : P
- Second Line : T(no of test case)
- Next T lines: N

Output

- For each test case, print YES or NO depending on the result

In [3]:
```
# Function to determine if a number is special or not
# Function to check if number is prime.
# Function to determine number of Prime factors for a given number

def Prime(n):
    c=0
    if n == 2:
        return True
    for i in range(2,n//2 + 1):
        if n % i == 0:
            c=1
            return False
    if c == 0:
        return True
Prime(71)

def PrimeFactors(N):
    if Prime(N):
        return 1
    c=0
    for i in range(2,N//2+1):
        if N % i == 0 and Prime(i):
            c+=1
    return c
PrimeFactors(10) # 2,5

def specialNumber(N,NP):
    if PrimeFactors(N) >= NP:
        return True
    return False
specialNumber(10,2)

def solution2():
    p=int(input())
    t=int(input())
    for i in range(0,t):
        n=int(input())
        if specialNumber(n,p):
            print("YES")
        else:
            print("NO")
solution2()
```

```
2
3
4
NO
10
YES
30
YES
```

Highest Remainder

Write a program to find a natural number that is smaller than N gives the highest remainder when divided by that number. If there is more than one such number, print the smallest one.

N highest = 0 k<N and N % x == highest

- 10
  - 9 1
  - 8 2
  - 7 3
  - 6 4
  - 5 0
  - 4 2
  - 3 1
  - 2 0

In [6]:
```
def highestRemainder(n):
    hr = 0
    v = n
    for i in range(n-1, n // 2,-1):
        r = n % i
        if r > hr:
            hr = r
            v = i
    print(v)
    return
highestRemainder(30)
```

```
16
```

In [ ]:
```

```

Tuples

- Difference between lists and tuples
  - t1=() and l1=[]
  - lists are mutable(can be changed or modified)
    - access, modify, add and delete data
  - tuples are immutable once it is intialised
    - tuples are used to access data only
    - all slicing operations work

In [11]:
```
t1 = (1,2,3,4,5)

t1[3] # Accessing fourth element.

t1[len(t1)//2:] # Accessing elements from middle to last.
```

Out[11]: (3, 4, 5)

In [12]:
```
type(t1)
```

Out[12]: tuple

In [ ]:
```

```

Dictionaries

- It works on the concept of a Set.
- Unique Data
- Keys, Values
  - Key is the unique identifier for a value.
  - Value is the data that can be accessed with a key
- Dictionaries are mutable

In [14]:
```
di = {"k1":"value1","k2":"value2","k3":"value3"}

di["k2"]
```

Out[14]: 'value2'

In [15]:
```
di.keys() # Will provide the list of all keys in the dictionaries
```

Out[15]: dict_keys(['k1', 'k2', 'k3'])

In [16]:
```
di.values() # Will provide all the values in the dictionaries as list
```

Out[16]: dict_values(['value1', 'value2', 'value3'])

In [18]:
```
di.items() # returns the list of tuples of keys and values
```

Out[18]: dict_items([('k1', 'value1'), ('k2', 'value2'), ('k3', 'value3')])

In [20]:
```
di["k4"] = "value4" # Adding an element to the dictionary
di
```

Out[20]: {'k1': 'value1', 'k2': 'value2', 'k3': 'value3', 'k4': 'value4'}

In [23]:
```
di["k2"] = "Aman"  # Updating an element of the dictionary
di
```

Out[23]:
```
{'k1': 'value1', 'k2': 'Aman', 'k3': 'value3', 'k4': 'value4'}
```

In [29]:
```
di.pop("k2")
di
```

Out[29]:
```
{'k1': 'value1', 'k4': 'value4'}
```

In [31]:
```
"k3" in di # To check if key is present in the dictionary or not
```

Out[31]:
```
False
```

Contact Application

- Add Contact
- Search for contact
- List all contact
- Modify contact
- Remove contact

In [35]:
```
contact = {}
def addContact(name,phone):
    # Verify that the contact does not already exists
    if name not in contact:
        contact[name] = phone
        print("Contact %s added" % name)
    else:
        print("Contact %s already exists" % name)
addContact('Aman',"7906579563")
```

```
Contact Aman added
```

In [37]:
```
contact
```

Out[37]:
```
{'Aman': '7906579563'}
```

In [41]:
```
def SearchContact(name):
    if name not in contact:
        print("Contact %s does not exist" % name)
    else:
        print(name ,":",contact[name])
SearchContact("Aman")
```

```
Aman : 7906579563
```

In [42]:
```
contact["A"]="1234566771"
contact["B"]="7383903231"
contact
```

Out[42]:
```
{'Aman': '7906579563', 'A': '1234566771', 'B': '7383903231'}
```

In [46]:
```
def ListContact():
    return contact
ListContact()
```

Out[46]:
```
{'Aman': '7906579563', 'A': '1234566771', 'B': '7383903231'}
```

In [47]:
```
def Modify(name,num):
    if name in contact:
        contact[name] = num
        print("Modified")
    else:
        print("The contact not available")
Modify('Aman','9565033528')
```

```
Modified
```

In [48]:
```
contact
```

Out[48]:
```
{'Aman': '9565033528', 'A': '1234566771', 'B': '7383903231'}
```

In [49]:
```
def Delete(name):
    if name in contact:
        contact.pop(name)
        print("Contact deleted")
    else:
        print("Contact not found")
Delete('A')
```

```
Contact deleted
```

In [50]:
```
contact
```

Out[50]:
```
{'Aman': '9565033528', 'B': '7383903231'}
```

In [54]:
```
# New contact is given as a dictionary.
# Use update keyword
# Merge new contacts with existing dictionary

def importC(newContacts):
    contact.update(newContacts)
    print(len(newContacts.keys()), "contacts got added.")
newContacts = {'D':542423534534,'C':3749249249}

importC(newContacts)
```

```
2 contacts got added.
```

In [53]:
```
contact
```

Out[53]:
```
{'Aman': '9565033528', 'B': '7383903231', 'D': 542423534534, 'C': 3749249249}
```

In [ ]:
```

```

Packages and Modules in Python

Package -> Collection of Modules(Python File .py) and subpackages.

SubPackages

Modules -> A single python file containing functions.

Package -> SubPackage -> Modules -> Functions

In [56]:
```
# math will import the math package
import math

math.floor(123.456)

math.pi
```

Out[56]:
```
3.141592653589793
```

In [62]:
```
from math import floor,pi

floor(12.33244343)

pi
```

Out[62]:
```
3.141592653589793
```

In [63]:
```
from math import floor as fl

fl(21313.4323232323323)
```

Out[63]:
```
21313
```

Function to generate n random numbers in a given range

In [81]:
```
import random
# random.randint(10,20) # Generates a single number between given range
def generateNRandomNumbers(n,lb,ub):
    for i in range(0,n):
        print(random.randint(lb,ub),end=" ")
generateNRandomNumbers(10,0,100)
```

```
13 60 20 46 39 54 32 93 18 73
```

In [5]:
```
from Packages import numerical

numerical.Prime(5)

numerical.PrimeFactors(96)
```

Out[5]: 2

In [6]:
```
from Packages.numerical import Prime
Prime(5)
```

Out[6]: True

In [7]:
```
from Packages.numerical import PrimeFactors
PrimeFactors(9332)
```

Out[7]: 2

In [ ]:
```

```