In [6]:
```
# Function to print all combinations of pairs of intergers in a unique list
# [1,2,3] --> (1,2),(1,3),(2,3) -> 3C2 --> 3!/((3-2)!*2!) --> NCR

#[1,2,3,4] --> two elements (1,2),(1,3),(1,4),(2,3),(2,4),(3,4) --> tuplets

def Combinations(li):
    for i in range(len(li)-1):
        for j in range(i+1,len(li)):
            print(li[i],li[j])
li = [1,2,3,4]
Combinations(li)
```

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

In [11]:
```
# Function to print all combinations of pairs of intergers in a unique list

#[1,2,3,4]  --> (1,2,3),(1,2,4),(1,3,4),(2,3,4)

def Combinations(li):
    for i in range(len(li)-2):
        for j in range(i+1,len(li)-1):
            for k in range(j+1,len(li)):
                print(li[i],li[j],li[k])
li = [1,2,3,4,5]
Combinations(li)
```

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

In [ ]:

```
def medium(li,k):
    while(True):
        li3 = differencePairs(li)
        if li3[0]==li3[1]:
            break
    if(len(li3[0]))>=k:
        print(li)
        return sorted(li3[0],reverse=True)[k-1]
    else:
        return -1

# Function to identify differences of all
# pairs of numbers and add those differences
# to same list.
# this function returns updated list and original list
def differencePairs(li):
    cli=li[:]    # Coping the li values into cli
    newelements = []

    for i in range(len(li)-1):
        for j in range(i+1,len(li)):
            d = abs(int(li[i])-int(li[j]))
            if d not in li and d not in newelements:
                newelements.append(str(d))
    li.extend(newelements)
    return [cli,li]

with open('DataFiles/medium-input.txt','r') as f:
    t=int(f.readline())
    for i in range(t):
        n=f.readline()
        li=f.readline().split()
        k=int(f.readline())
        medium(li,k)

#li = [3,6,9]
#differencePairs(li)


```

In [45]:

```
# List Data referencing vs Data copy
a = [1,2,3]
b= [1,3,2]
a = b.copy() # Data Copy through indirect referencing
a=b[:] # Data copy through direct referencing

a = b # Data Referencing

b.append(4)
a.append(5)
b
```

Out[45]: [1, 3, 2, 4, 5]

In [25]:
```
r=[1,2,3,4,5,6]
s=[7,8]
r.extend(s)
r
```

Out[25]: [1, 2, 3, 4, 5, 6, 7, 8]

In [40]:
```
# Below lists are the examples of constant list
# When we do the difference we will get same elements in the list
# Don't need to do any append

[4,8]
[20,40,60]
[4,8,12,16]
[3,6,9,12]
# Convert the list into an
# Arithmetic Progression  (differences are same here in those lists)

[3,8,15]
[3,8,15,5,2,1,4,6,7,9,10,11,12,13,14]
```

Out[40]: [4, 8]

In [ ]:
```
def medium(li,k):
    while(True):
        li3 = differencePairs(li)
        if li3[0] == li3[1]:
            break
    if(len(li3[0]))>=k:
        return sorted(li3[0],reverse=True)[k-1]
    else:
        return -1
def differencePairs(li):
    c = li.copy()
    newelements = []
    for i in range(len(li)-1):
        for j in range(i+1,len(li)):
            d = abs(li[i]-li[j])
            if d not in li and d not in newelements:
                newelements.append(d)
    li.extend(newelements)
    return li

li = [2,3,6,9,12,1,4,7,10,5,8,11]
differencePairs(li)
medium(li,2)
```

Set - Data Structure in python

- Represented by {}
- it contains only unique data reaped elements will remove automatically
- Set mutable
- index operations will not work here in set

In [4]:
```
a = {1,2,3,4,5,6,6,5}

a.add(7)   # Adding a single element to set
a
```

Out[4]: {1, 2, 3, 4, 5, 6, 7}

In [5]:
```
for i in a:
    print(i,end=' ')    #--> Accessing elements in a set

```

```
1 2 3 4 5 6 7
```

In [9]:
```
# adding
b = {7,8,9,1,2,3}    # Remove duplicate elements adds only unique elements
li=[22,33]
a.update(b)
a
```

Out[9]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

In [13]:
```
li = [22,44]
a.update(li)
a
```

Out[13]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 22, 44}

In [15]:
```
s='sirei'
a.update(s)
a
```

Out[15]: {1, 2, 22, 3, 4, 44, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

In [16]:
```
p=[334,488483,393]
st = 'upma'
a.update(b,p)
a
```

Out[16]: {1, 2, 22, 3, 334, 393, 4, 44, 488483, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

In [18]:
```
# To remove a element  i.e only a single element can be removed by discard f
a.discard(44)
a
```

Out[18]: {1, 2, 22, 3, 334, 393, 4, 488483, 5, 6, 7, 8, 9, 'e', 'i', 'r', 's'}

In [23]:
```
a = { 10,1,2,3,4,5,6}
b = {7,8,9,1,2,3}

a.union(b)

# A U B = B U A    ---> a union b is equal to b union a

```

Out[23]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In [24]:
```
b.union(a)
```

Out[24]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In [25]:
```
# Accessing common elememnts in both sets

a.intersection(b)
```

Out[25]: {1, 2, 3}

In [26]:
```
# If both sets are not have common elements then it will give o/p as TRUE ot
c = {111,123}
a.isdisjoint(b)
```

Out[26]: False

In [28]:
```
c = {111,123}
a.isdisjoint(c)
```

Out[28]: True

In [29]:
```
a - b   # All elements in a  which are not in b
```

Out[29]: {4, 5, 6, 10}

In [30]:
```
b - a  # Return all elements in b which are not in a
```

Out[30]: {7, 8, 9}

In [31]:
```
a
```

Out[31]: {1, 2, 3, 4, 5, 6, 10}

In [32]:
```
b
```

Out[32]: {1, 2, 3, 7, 8, 9}

In [33]:
```
sorted(a)
```

Out[33]: [1, 2, 3, 4, 5, 6, 10]

In [34]:
```
a^b   # Elements either in a or in b  i.e not common i.e not intersectioned

```

Out[34]: {4, 5, 6, 7, 8, 9, 10}

In [35]:
```
t = { 1,4,33,393,292,8,29,9}
sorted(t)
```

Out[35]: [1, 4, 8, 9, 29, 33, 292, 393]

In [36]:
```
# Creating an empty set
d = set()
d
```

Out[36]: set()

In [38]:
```
li = [32, 34,22,1,2,3,4,1,2,4,56,]
u = set(li)
u
```

Out[38]: {1, 2, 3, 4, 22, 32, 34, 56}

Functional Programming

- C Language is procedural or structural programming
- C++,Java are Object oriented languages
- Python,JavaScript,PHP are Scrpting Languages

Procedural : C

Object Oriented : Java, Python

Scripting : PHP, Python, Javascripting, Shell, Perl

Functional : Python, Haskell, Scala

Logic : Prolog, Lisp

Functional Programming

- **Def** : In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- Search in google as "**Functioanl Programming**"

List Comprehensions

- mathematical or sybalic logic into a list

In [41]:
```
# N natural numbers in a list
n = 10
li =[]
for i in range(1,n+1):
    li.append(i)
print(li)

```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

For above programm we need 5 lines by using List Comprehesion we get in single line

In [42]:
```
li = [i for i in range(1,11)]
li
```

Out[42]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [43]:
```
# Apply list comprehension to store the
# Cubes on n natural numbers

li = [i**3 for i in range(1,11)]
li
```

Out[43]:
```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

In [48]:
```
# Function to calculate the factorial
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n-1)
# factorial(5)
```

In [49]:
```
# Apply list comprehension to calculate
# Factorial of n natural numbers

n = 10
factorialList = [factorial(i) for i in range(1,n+1)]
factorialList
```

Out[49]:
```
[1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
```

In [63]:
```
# Store cummulative sum of numbers till n in a list
# n = 5  --> [1, 3, 6, 10, 15]

def cummsum(n):
    li = []
    for i in range(1,n+1):
        li.append(i)
    #return li
    s=0
    for i in li:
        sum(li([:i+1])
    return s

cummsum(5)

```

```
  File "<ipython-input-63-c0d7d9b4c166>", line 11
    s=sum(li([:i+1])
              ^
SyntaxError: invalid syntax
```

In [62]:
```
li = [1,2,3,4,5,6]
sum(li[0:4]) # ---> ouput upto 4
```

Out[62]: 10

In [65]:
```
n = 5
cummulativeSum = [sum([i]) for i in range(1,n+1)]
cummulativeSum
```

Out[65]: [1, 2, 3, 4, 5]

In [73]:
```
cummulativeSum = [sum(range(1,i+1)) for i in range(1,n+1)]
cummulativeSum
```

Out[73]: [1, 3, 6, 10, 15]

In [78]:
```
# List Comprehensions to store
# Only Leap Years in a given time period

st = 1970
et = 2019
leapYears = [i for i in range(st,et+1)
             if i%400==0 or (i%100!=0 and i%4 == 0) ]
leapYears
```

Out[78]: [1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016]

In [81]:
```
# If we have a if condition then it is after for loop condition
st = 1970
et = 2019
leapYears = [[(i%400==0 or (i%100!=0 and i%4 == 0)),i] for i in range(st,et+
leapYears
```

Out[81]:
```
[[False, 1970],
 [False, 1971],
 [True, 1972],
 [False, 1973],
 [False, 1974],
 [False, 1975],
 [True, 1976],
 [False, 1977],
 [False, 1978],
 [False, 1979],
 [True, 1980],
 [False, 1981],
 [False, 1982],
 [False, 1983],
 [True, 1984],
 [False, 1985],
 [False, 1986],
 [False, 1987],
 [True, 1988],
 [False, 1989],
 [False, 1990],
 [False, 1991],
 [True, 1992],
 [False, 1993],
 [False, 1994],
 [False, 1995],
 [True, 1996],
 [False, 1997],
 [False, 1998],
 [False, 1999],
 [True, 2000],
 [False, 2001],
 [False, 2002],
 [False, 2003],
 [True, 2004],
 [False, 2005],
 [False, 2006],
 [False, 2007],
 [True, 2008],
 [False, 2009],
 [False, 2010],
 [False, 2011],
 [True, 2012],
 [False, 2013],
 [False, 2014],
 [False, 2015],
 [True, 2016],
 [False, 2017],
 [False, 2018],
 [False, 2019]]
```

In [91]:
```
li = [1,2,3,2,1]
unique = [i for i in range(0,len(li)) if sorted(li.count(i) == 1]
unique
```

Out[91]: [3]

Iterators

- **Iterable** - Strings,Lists, Tuples, Sets, Dictionaries
- Convert Iterable to Iterator -> iter()
- **For Loop** : goes through every element untill condition reached
- **Iterator**: we can stop at anywhere and start at any time

In [96]:
```
it = iter('Python')
next(it)   # --> 'P'
next(it)   # --> 'y'
```

Out[96]: 'y'

In [97]:
```
it = iter('Python')
print(next(it))
print(next(it))
```

```
P
y
```

In [105]:
```
it = iter('Python')

for i in it:
    print(next(it))
```

```
y
h
n
```

In [106]:
```
it = iter('Python')

print('1: ')
print(next(it))
print('\n')
print('2: ')
print(next(it))
```

```
1:
P


2:
y
```

Generators

- Generator is Function

In [117]:
```
1
2  def generator():
3      n = 2
4      yield n
5      n = n ** 3
6      yield n
7
8      n = n ** 3
9      yield n
10 a = generator()
11 print(next(a))
12 print(next(a))
13 print(next(a))
14
```

```
2
8
512
```

In [120]:
```
1  def generator1():
2      n = 2
3      for i in range(1,5):
4          n **= 3
5          yield n
6  a = generator1()
7
8  print(next(a))
9  print(next(a))
```

```
8
512
```

In [127]:

```
def generator2():
    n = 2
    while(True):
        n **= 3
        yield n

a = generator2()

# print(next(a))
# b=next(a)**2
# b *= next(a)
# print(next(a))

for i in range()
```

```
8
```

```
241785163922925834941235 2
```

In [ ]:

```

```