In [9]:
```
# Function to print all combinations of pairs of integers in a unique list
# [1,2,3]->(1,2),(1,3),(2,3)->3C2 --> 3!/(3-2)!*2! = 3

# [1,2,3,4] -> 1,2 1,3 1,4 2,3 2,4 3,4
#[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)-1):
        for j in range(i+1,len(li)):
                print(li[i],li[j])

    return
combinations([1,2,3])


```

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

In [14]:
```
def combinations3(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])

    return
combinations3([1,2,3,4])
```

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

In [ ]:
```
def medium(li,k):
    while(True):
        #li3=[[],li
        count=1
        if count==1:
            li3=differencePairs(li)
            if li3[0]==li3[1]:
                break

    if len(li3[0])>=k:
        return sorted(li3[0],reverse=True)[k-1]
    return -1

    return li3[0]

# Function to identify differences of all pairs of numbers
# Pairs of numbers and add those differences
# to the same list
# It returns the updated list and original list

def differencePairs(li):
    c=li.copy()
    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 [c,li]
with open('DataFiles/medium.txt','r') as f:
    t=int(f.readline())
    for i in range(t):
        f.readline()
        li=f.readline().split()
        k=f.readline()
        print(medium(li,k))

```

In [ ]:
```
[4,8]
[20,40,60]
[4,8,12,16]
[3,6,9,12]

# Convert the list into an Arithmetic Progression
a=[1,2,3]
b=[1,3,2]
a=b.copy() #
a=b   # data Referencing
[3,8]

```

Set - Data Structure in Python

Represented by '{}'

In [1]:
```
a={1,2,3,4,5,6,6}
a.add(7)  # Adding a single element to a set
a

#for i in a:
    #print(i,end=" ") -> Accessing elements in a set

b = {7,8,1,2,3,9}
li = [11,12,13,1]

a.update(b,li)

a
```

Out[1]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13}

In [3]:
```
a.discard(12)
a
```

Out[3]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13}

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

a.union(b)

b.union(a)
```

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

In [14]:
```
a={10,1,2,3,4,5,6}
b={7,8,9,1,2,3}
c={111,123}

a.intersection(b)
```

Out[14]: {1, 2, 3}

In [15]:
```
a.isdisjoint(c) # No common elements b/w A and C
```

Out[15]: True

In [16]:
```
a-b  # All elements of a which are not in b -> A-(AnB)
```

Out[16]: {4, 5, 6, 10}

In [17]:
```
b-a  # All elements of b which are not in a
```

Out[17]: {7, 8, 9}

In [18]:
```
sorted(a) # sorting of elements in a set
```

Out[18]: [1, 2, 3, 4, 5, 6, 10]

In [19]:
```
a={10,1,2,3,4,5,6}
b={7,8,9,1,2,3}
a^b # Elements either in a or b
```

Out[19]: {4, 5, 6, 7, 8, 9, 10}

In [20]:
```
d=set()  # Creates an empty set
d
```

Out[20]: set()

In [21]:
```
li=[1,2,3,4,2,1,2,3,4]
u=set(li)
u
```

Out[21]: {1, 2, 3, 4}

In [ ]:
```

```

Procedural: C

Object Oriented : Java

Scripting : PHP, Python, Javascript, Shell, Perl

Functional : Pyhton, Haskell, Scala

Logic : Prolog, Lisp

List Comprehensions

In [27]:
```
# List of N natural Numbers in a list

n=10
li=[]
for i in range(1,n+1):
    li.append(i)
li

```

Out[27]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [29]:
```
li=[i for i in range(1,11)]
li
```

Out[29]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [31]:
```
# Apply list comprehension to strore the cubes of n natural numbers
li=[i**3 for i in range(1,11)]
li
```

Out[31]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

In [38]:
```
# Function to calculate the factorial

def factorial(n):
    if n==0 or n==1:
        return 1
    return n*factorial(n-1)
factorial(5)

# Apply a list comprehension to calculate the factorial of N natural numbers

n=7
factorialList=[factorial(i) for i in range(1,n+1)]
factorialList

```

Out[38]: [1, 2, 6, 24, 120, 720, 5040]

In [42]:
```
# Store cumulative sum of numbers till n in a list
# n= 5--> [1,3,6,10,15]

def sumcumulative(n):
    sum=0
    for i in range(1,n+1):
        sum=sum+i
    return sum
sumcumulative(5)

n=5
cumulativesumList=[sum(range(1,i+1)) for i in range(1,n+1)]
cumulativesumList

```

Out[42]: [1, 3, 6, 10, 15]

In [50]:
```
# List Comprehension to store only leap years in a given time period

def leapYear(year):
    if year%400==0 or (year%4==0 and year%100!=0):
        print(year)
#leapYear(1972)

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[50]: [1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016]

In [1]:
```
li=[1,2,3,2,1,4,4,6,5,5]
li.sort()
u2=[]
unique=[]
unique=[li[i] for i in range(0,len(li)-1) if (li)[i]!=(li)[i+1]]
#[u2.append(i) for i in li if i not in u2]
unique
```

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

In [ ]:
```

```

Iterators

Iterable - String,Lists,Tuples, Sets, Dictionaries

Convert iterable to iterator--> iter()

for loop : We can not break until some condition is reached

Iterator : We can stop at anytime(There is a pause in iterable process)

In [2]:
```
it=iter('Python')
print('1:')
print(next(it))
print('\n')
print('2:')
print(next(it))
```

```
1:
P


2:
y
```

Generators

Generator is a user defined function

yield is like a return

In [5]:
```
def generator():
    n=2
    for i in range(1,5):
        n**=3
        yield n

a=generator()
next(a)
next(a)

```

Out[5]:
```
512
```

In [20]:
```
# for infinite loop

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

a=generator()
next(a)
b=next(a)**2
b*=next(a)
b

for i in range(5):
    print(next(a))
```

```
512
134217728
241785163922925834941235 2
141347765182270746366663800059433481266198711750049516649728496103409582 08
282401395870821749694910884220462786335135391185157752468340193086269383036119 8
499905873920995229996970897865498283996578123296865878390947626553088486946106 4
307960914827161205726320724927035277237573594788345303657349 12
```

In [ ]:
```

```

In [ ]:
```

```