```
# Returns true if the
# there is a subarray
# of arr[] with sum
# equal to 'sum'
# otherwise returns
# false. Also, prints
# the result
def subArraySum(arr, n, sum_):

    # Pick a starting
    # point
    for i in range(n):
        curr_sum = arr[i]

        # try all subarrays
        # starting with 'i'
        j = i + 1
        while j <= n:

            if curr_sum == sum_:
                print ("Sum found between")
                print("indexes % d and % d"%( i, j-1))

                return 1

            if curr_sum > sum_ or j == n:
                break

            curr_sum = curr_sum + arr[j]
            j += 1

    print ("No subarray found")
    return 0

# Driver program
arr = [1, 4, 3, 2, 6, 7]
n = len(arr)
sum_ = 12

subArraySum(arr, n, sum_)
```

```
No subarray found
0
```

```
# Returns true if the
# there is a subarray
# of arr[] with sum
# equal to 'sum'
# otherwise returns
# false. Also, prints
# the result
def subArraySum(arr, n, sum_):
```

```
        # Pick a starting
        # point
        for i in range(n):
            curr_sum = arr[i]

            # try all subarrays
            # starting with 'i'
            j = i + 1
            while j <= n:

                if curr_sum == sum_:
                    print ("Sum found between")
                    print("indexes % d and % d"%( i, j-1))

                    return 1

                if curr_sum > sum_ or j == n:
                    break

                curr_sum = curr_sum + arr[j]
                j += 1

        print ("No subarray found")
        return 0

    # Driver program
    arr = [1,2,3,7,5]
    n = len(arr)
    sum_ = 12

    subArraySum(arr, n, sum_)
```

```
     Sum found between
     indexes  1 and  3
     1
```

```
    # Returns true if the
    # there is a subarray
    # of arr[] with sum
    # equal to 'sum'
    # otherwise returns
    # false. Also, prints
    # the result
    def subArraySum(arr, n, sum_):

        # Pick a starting
        # point
        for i in range(n):
            curr_sum = arr[i]

            # try all subarrays
            # starting with 'i'
            j = i + 1
            while j <= n:
```

```
            if curr_sum == sum_:
                print ("Sum found between")
                print("indexes % d and % d"%( i, j-1))

                return 1

            if curr_sum > sum_ or j == n:
                break

            curr_sum = curr_sum + arr[j]
            j += 1

    print ("No subarray found")
    return 0

# Driver program
arr = [1,2,3,4,5,6,7,8,9,10]
n = len(arr)
sum_ = 15

subArraySum(arr, n, sum_)
```

```
 Sum found between
 indexes  0 and  4
 1
```

```
# Python program to find a pair with the given difference

# The function assumes that the array is sorted

def findPair(arr, size, n):


    mpp = {}


    for i in range(size):

        if arr[i] in mpp.keys():

             mpp[arr[i]] += 1

             if(n == 0 and mpp[arr[i]] > 1):

                return true;

        else:

             mpp[arr[i]] = 1



    if(n == 0):
```

```
        return false;

    for i in range(size):
         if n + arr[i] in mpp.keys():
            print("Pair Found: (" + str(arr[i]) + ", " + str(n + arr[i]) + ")")
            return True

    print("No Pair found")
    return False
# Driver program to test above function
arr = [12,  16,  19,  23 , 50]
size = len(arr)
n = 4
findPair(arr, size, n)
```

```
 Pair Found: (12, 16)
 True
```

```
# Python program to find a pair with the given difference
# The function assumes that the array is sorted
def findPair(arr, size, n):

    mpp = {}

    for i in range(size):
        if arr[i] in mpp.keys():
            mpp[arr[i]] += 1
            if(n == 0 and mpp[arr[i]] > 1):
               return true;
        else:
            mpp[arr[i]] = 1
```

```
    if(n == 0):

      return false;


    for i in range(size):

         if n + arr[i] in mpp.keys():

            print("Pair Found: (" + str(arr[i]) + ", " + str(n + arr[i]) + ")")

            return True



    print("No Pair found")

    return False

# Driver program to test above function

arr = [12,  16,  19,  23 , 50]

size = len(arr)

n = 5
findPair(arr, size, n)
```

```
 No Pair found
 False
```

```
def isSubsetSum(arr, n, sum):
    # Base Cases
    if sum == 0:
        return True
    if n == 0 and sum != 0:
        return False

    # If last element is greater than sum, then
    # ignore it
    if arr[n-1] > sum:
        return isSubsetSum(arr, n-1, sum)

    ''' else, check if sum can be obtained by any of
    the following
    (a) including the last element
    (b) excluding the last element'''

    return isSubsetSum(arr, n-1, sum) or isSubsetSum(arr, n-1, sum-arr[n-1])

# Returns true if arr[] can be partitioned in two
# subsets of equal sum, otherwise false
```

```
def findPartion(arr, n):
    # Calculate sum of the elements in array
    sum = 0
    for i in range(0, n):
        sum += arr[i]
    # If sum is odd, there cannot be two subsets
    # with equal sum
    if sum % 2 != 0:
        return false

    # Find if there is subset with sum equal to
    # half of total sum
    return isSubsetSum(arr, n, sum // 2)


# Driver code
arr = [1,5,11,5]
n = len(arr)

# Function call
if findPartion(arr, n) == True:
    print("Can be divided into two subsets of equal sum")
else:
    print("Can not be divided into two subsets of equal sum")
```

```
 Can be divided into two subsets of equal sum
```

```
def isSubsetSum(arr, n, sum):
    # Base Cases
    if sum == 0:
        return True
    if n == 0 and sum != 0:
        return False

    # If last element is greater than sum, then
    # ignore it
    if arr[n-1] > sum:
        return isSubsetSum(arr, n-1, sum)

    ''' else, check if sum can be obtained by any of
    the following
    (a) including the last element
    (b) excluding the last element'''

    return isSubsetSum(arr, n-1, sum) or isSubsetSum(arr, n-1, sum-arr[n-1])

# Returns true if arr[] can be partitioned in two
# subsets of equal sum, otherwise false


def findPartion(arr, n):
    # Calculate sum of the elements in array
```

```
    sum = 0
    for i in range(0, n):
        sum += arr[i]
    # If sum is odd, there cannot be two subsets
    # with equal sum
    if sum % 2 != 0:
        return false

    # Find if there is subset with sum equal to
    # half of total sum
    return isSubsetSum(arr, n, sum // 2)


# Driver code
arr = [1,3,5]
n = len(arr)

# Function call
if findPartion(arr, n) == True:
    print("Can be divided into two subsets of equal sum")
else:
    print("Can not be divided into two subsets of equal sum")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-a7f6c3c44350> in <module>()
     42
     43 # Function call
---> 44 if findPartion(arr, n) == True:
     45     print("Can be divided into two subsets of equal sum")
     46 else:

<ipython-input-7-a7f6c3c44350> in findPartion(arr, n)
     30     # with equal sum
     31     if sum % 2 != 0:
---> 32         return false
     33
     34     # Find if there is subset with sum equal to

NameError: name 'false' is not defined
```

SEARCH STACK OVERFLOW

```
def areIsomorphic(str1, str2):
    #initializing a dictionary
    #to store letters from str1 and str2
    #as key value pairs
    charCount = dict()
    #initially setting c to "a"
    c = "a"
    #iterating over str1 and str2
    for i in range(len(str1)):
        #if str1[i] is a key in charCount
        if str1[i] in charCount:
            c = charCount[str1[i]]
```

```
            if c != str2[i]:
                return False
        #if str2[i] is not a value in charCount
        elif str2[i] not in charCount.values():
            charCount[str1[i]] = str2[i]
        else:
            return False
    return True

#Driver Code
str1 = "aab"
str2 = "xxy"

#Function Call
if (len(str1) == len(str2) and areIsomorphic(str1, str2)):
    print(1)
else:
    print(0)
```

```
 1
```

```
def areIsomorphic(str1, str2):
    #initializing a dictionary
    #to store letters from str1 and str2
    #as key value pairs
    charCount = dict()
    #initially setting c to "a"
    c = "a"
    #iterating over str1 and str2
    for i in range(len(str1)):
        #if str1[i] is a key in charCount
        if str1[i] in charCount:
            c = charCount[str1[i]]
            if c != str2[i]:
                return False
        #if str2[i] is not a value in charCount
        elif str2[i] not in charCount.values():
            charCount[str1[i]] = str2[i]
        else:
            return False
    return True

#Driver Code
str1 = "aab"
str2 = "xyz"

#Function Call
if (len(str1) == len(str2) and areIsomorphic(str1, str2)):
    print(1)
else:
    print(0)
```

```
 0
```

```
# Function to group anagrams from a given list of words
def groupAnagrams(words):
    # a list to store anagrams
    anagrams = []

    # base case
    if not words:
        return anagrams

    # sort each word on the list
    nums = [''.join(sorted(word)) for word in words]

    # construct a dictionary where the key is each sorted word,
    # and value is a list of indices where it is present
    d = {}
    for i, e in enumerate(nums):
        d.setdefault(e, []).append(i)

    # traverse the dictionary and read indices for each sorted key.
    # The anagrams are present in the actual list at those indices
    for index in d.values():
        collection = tuple(words[i] for i in index)
        if len(collection) > 1:
            anagrams.append(collection)

    return anagrams


if __name__ == '__main__':
    # a list of words
    words = ['CARS', 'REPAID', 'DUES', 'NOSE', 'SIGNED', 'LANE', 'PAIRED', 'ARCS',
             'GRAB', 'USED', 'ONES', 'BRAG', 'SUED', 'LEAN', 'SCAR', 'DESIGN']

    anagrams = groupAnagrams(words)
    for anagram in anagrams:
        print(anagram)
```

```
('CARS', 'ARCS', 'SCAR')
('REPAID', 'PAIRED')
('DUES', 'USED', 'SUED')
('NOSE', 'ONES')
('SIGNED', 'DESIGN')
('LANE', 'LEAN')
('GRAB', 'BRAG')
```

```
def insertionSort(arr):


    # Traverse through 1 to len(arr)

    for i in range(1, len(arr)):
```

```
        key = arr[i]


        # Move elements of arr[0..i-1], that are

        # greater than key, to one position ahead

        # of their current position

        j = i-1

        while j >= 0 and key < arr[j] :

                arr[j + 1] = arr[j]

                j -= 1

        arr[j + 1] = key


# Driver code to test above

arr = [100,70,90,40,60,30,50]
insertionSort(arr)

for i in range(len(arr)):

    print ("% d" % arr[i])
```

```
  30
  40
  50
  60
  70
  90
  100
```

```
# Recursive implementation of the binary search algorithm to return
# the position of `target` in subarray nums[left…right]
def binarySearch(nums, left, right, target):

    # Base condition (search space is exhausted)
    if left > right:
        return -1

    # find the mid-value in the search space and
    # compares it with the target

    mid = (left + right) // 2

    # overflow can happen. Use below
    # mid = left + (right - left) / 2

    # Base condition (a target is found)
    if target == nums[mid]:
```

```
            return mid

        # discard all elements in the right search space,
        # including the middle element
        elif target < nums[mid]:
            return binarySearch(nums, left, mid - 1, target)

        # discard all elements in the left search space,
        # including the middle element
        else:
            return binarySearch(nums, mid + 1, right, target)


if __name__ == '__main__':

    nums = [100,70,90,40,60,30,50]
    target = 40

    (left, right) = (0, len(nums) - 1)
    index = binarySearch(nums, left, right, target)

    if index != -1:
        print('Element found at index', index)
    else:
        print('Element found not in the list')
```

```
 Element found at index 3
```