First Approach

Find all the pairs and check whether sum of pairs is equal to x or not

Complexity O(n^2)

In [13]:
```
def findPairs(arr,x):
    arr_len = len(arr)
    for i in range(arr_len):
        for j in range(i,arr_len):
            if ((arr[i] + arr[j])==x):
                print("sum {0} is found by {1} and {2}".format(x,arr[i],arr[j]))
```

In [14]:
```
arr = [7,5,4,8,1]
x = 12
findPairs(arr,x)
# ans
# sum 12 is found by 7 and 5
# sum 12 is found by 4 and 8
```

```
sum 12 is found by 7 and 5
sum 12 is found by 4 and 8
```

Second Approach

First sort the array. Do the sum of first and last element . If it is equal to x then return 1 and pair. else increment left and right based on condition

Complexity O(nlogn) nlogn -->sorting

In [15]:
```
def findPairs_BS(arr,left,right,x):
#     left = 0
#     right = len(arr)-1
    result_list = []
    arr = sorted(arr) # implement sorting later
    while left < right :
        pair_sum = arr[left]+arr[right]
        if pair_sum == x:
            return 1,(arr[left],arr[right])
        elif pair_sum < x:
            left = left+1
        elif pair_sum > x:
            right = right -1
    return 0,(0,0)
```

In [16]:
```
arr = [7,5,4,8,1]
x = 12
findPairs_BS(arr,left = 0,right = 4, x = x)
# ans (1, (4, 8))
```

Out[16]:
```
(1, (4, 8))
```

Third Approach

Use Hash map . X = 8 and a = 2 => b = X-a = 6 . search for b in remaining elements. Complexity Time - O(n) space - O(k) where k is range of elements

In [17]:
```
def findPairs_HM(arr,x):
    hash_map = [0]*(max(max(arr),x)+1)
    for ix,i in enumerate(arr):
        temp = x-i
        if hash_map[temp] == 1:
            print("pair for sum {0} found is {1} and {2}".format(x,i,temp))
        hash_map[i] = 1
```

In [19]:
```
arr = [7,5,4,8,1]
x = 12
findPairs_HM(arr,x)
# ans
# pair for sum 12 found is 5 and 7
# pair for sum 12 found is 8 and 4
```

```
pair for sum 12 found is 5 and 7
pair for sum 12 found is 8 and 4
```

In [ ]:

In [ ]:

In [2]:

In [ ]: