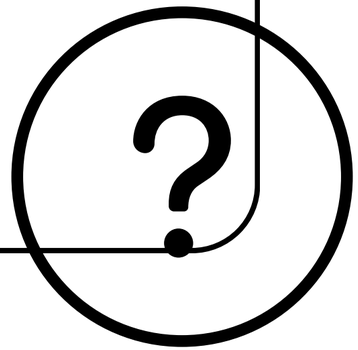
Check Pair Sum in an Array

Let's Understand the problem

-Given an array of n integers and a value targetSum, write a program to check whether there is a pair of elements in the array whose sum is equal to targetSum. If yes, return true; otherwise, return false

-Assume all elements are distinct.

-Values in the array can be both negative and positive



EX1:

Input

TargetSum =15 **Output : True**

Explanation : (7 , 8) or (-5 , 20) are the pairs with the sum of 15

EX2:

Input

X[]=

-5	4	-2	16	8	9

TargetSum =15 **Output : False**

Explanation : There is no pair of elements whose sum is equal to 15.

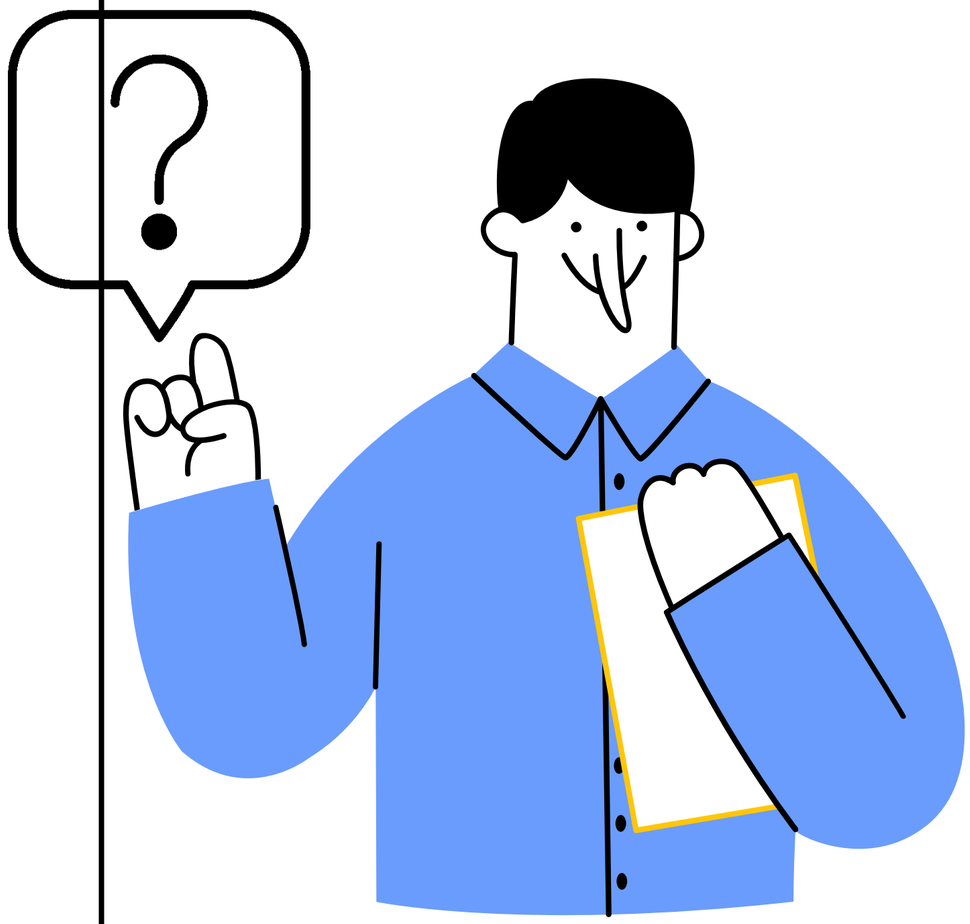
Brute force approach using nested loops

```
def checkPairSum(X, n, targetSum):
    for i in range(n - 1):
        for j in range(i + 1, n):
            if X[i] + X[j] == targetSum:
                return True
    return False
```

Discussed solution approaches:

-Efficient approach using sorting and binary search

Solution idea using sorting and binary search:



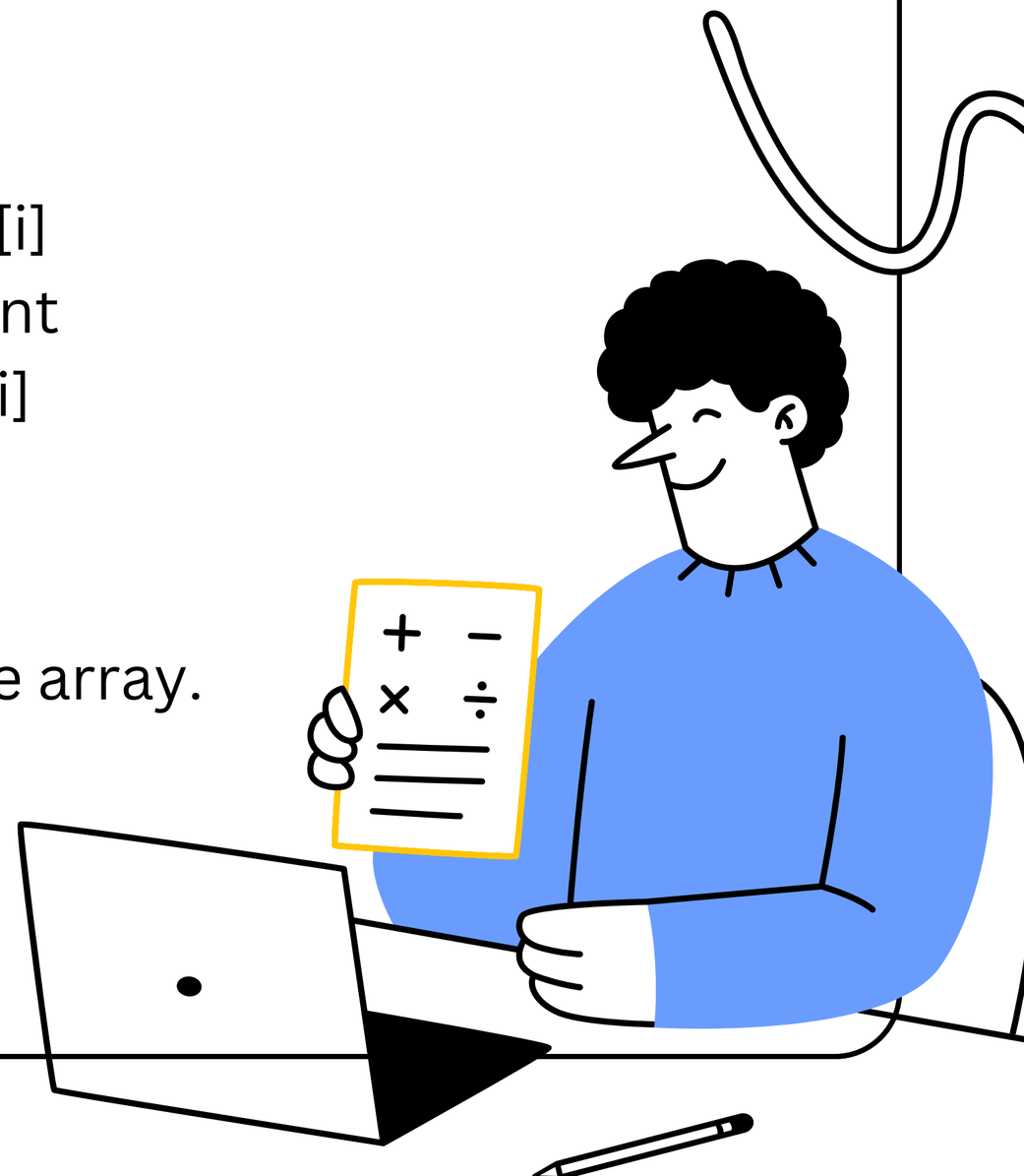
- The critical question is: How can we improve the time complexity? On a sorted array, binary search will take O(log n) time. Can we solve this problem using sorting and binary search? Let's think! For every element X[i], if targetSum - X[i] is present in the array, then there exists a pair with targetSum.
- So one idea is to sort the array and apply binary search to find targetSum - X[i] for every element X[i]. If targetSum - X[i] is present, we return true.

Solution steps:

1. We sort the array in increasing order,

2. Now we iterate over each array element X[i] and apply binary search to look for an element with a value targetSum-X[i]. If targetSum-X[i] exists, we return true. Activate W

3. We return false if there is no such a pair in the array.



Solution code python

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1

# Example usage:
X = [1, 7, 4, 15, 10]
target_sum = 17
result = pair_sum_array(X, target_sum)
print(result)
```

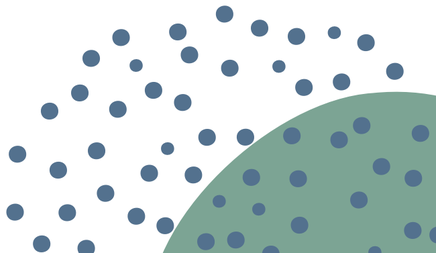
```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

X[]=

1	7	4	15	10

n = 5 **targetSum = 17**



```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

n = 5

targetSum = 17

X[]=

1	4	7	10	15

n = 5

r = 4

l = 0

key =17-1=16

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

n = 5 **targetSum = 17**

X[]=

1	4	7	10	15

r = 4

l = 0

key =17-4=13

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

n = 5

targetSum = 17

X[]=

1	4	7	10	15

r = 4

l = 0

key =17-7=10

Solution analysis

- Suppose we are using the O(nlogn) sorting algorithm heap sort and iterative binary search for the implementation. Time complexity = Time complexity of heap sort+n *Time complexity of binary search= O(nlogn)+ n* O(logn) = O(nlogn) + O(nlogn) = O(nlogn) .

- Space complexity =Space complexity of heap sort +Space complexity of the iterative binary search = O(1)+ O(1)= O(1)

Solution analysis

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

$$T(\ ) =$$

Solution analysis

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

$$T(n) =$$

Solution analysis

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False

def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

$$T(n) = \quad 1$$

Solution analysis

```python
def pair_sum_array(X, target_sum):
    X.sort()
    n = len(X)
    for i in range(n):
        k = binary_search(X, 0, n - 1, target_sum - X[i])
        if k >= 0:
            return True
    return False
```

```python
def binary_search(X, l, r, key):
    while l <= r:
        mid = l + (r - l) // 2
        if X[mid] == key:
            return mid
        elif X[mid] < key:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

$$T(n) = O(n\log n) + \sum_{i=0}^{n-1} \text{TB.S(n/2)+1}$$

Solution analysis

```
TB.S(n)=TB.S(n/2)+1          |          TB.S(1)=1

TB.S(n)=TB.S(n/2)+1                     TB.S(1)=1
        =TB.S(n/2^2)+2                  TB.S(n/2)=TB.S(n/2^2)
        =TB.S(n/2^3)+3                  solve k=log(n)
        =TB.S(n/2^k)+k
        =TB.S(n/2^log(n))+log(n)
        =TB.S(n/n)+log(n)
        =1+log(n)
         =>O(log(n))
```

$$T(n) = O(\text{nlogn}) + \sum_{i=0}^{n-1} \text{TB.S(n/2)+1}$$

$$T(n) = O(\text{nlogn}) + \sum_{i=1}^{n} O(\text{logn}) \qquad , \sum_{i=l}^{u} 1$$

$$= O(\text{nlogn}) + O(\text{logn}) * \sum_{i=1}^{n} 1$$

$$= O(\text{nlogn}) + O(\text{logn}) * n - 1 + 1 \qquad , u - l + 1$$

$$= O(\text{nlogn}) + n * O(\text{logn})$$

$$= O(\text{nlogn}) + O(\text{nlogn}) = O(\text{nlogn})$$

Comparisons of time and space complexities

Nestled inape :Time O(n^2), Space = O(1)

Sorting and binary search :Time O(nlogn), Space = O(1)

Sorting and Two pointers:Time O(nlogn), Space = O(1)

Hash Table:Time O(n), Space = O(n)

Thank You!

for Watching