**Triplet Sum in Array**

**Company Tags**

[**Accolite**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Accolite) [**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon) [**Microsoft**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Microsoft) [**OYO Rooms**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=OYO%20Rooms) [**Samsung**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Samsung)[**CarWale**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=CarWale)

Given an array arr of size n and an integer X. Find if there's a triplet in the array which sums up to the given integer X.

**Example 1:**

**Input**:

n = 6, X = 13

arr[] = [1 4 45 6 10 8]

**Output**:

1

**Explanation**:

The triplet {1, 4, 8} in

the array sums up to 13.

**Example 2:**

**Input**:

n = 5, X = 10

arr[] = [1 2 4 3 6]

**Output**:

1

**Explanation**:

The triplet {1, 3, 6} in

the array sums up to 10.

The algorithm used in the code is as follows:

1. Sort the array **arr** in non-decreasing order using the **Arrays.sort()** method. Sorting the array allows us to apply a two-pointer approach efficiently.
2. Initialize a variable **sum** to 0.
3. Iterate over the array from the first element to the second-to-last element (up to **n-2** index) using a variable **i**:
   * Set a pointer **left** to **i + 1** (the next element after **i**).
   * Set a pointer **right** to the last element of the array.
4. While the **left** pointer is less than the **right** pointer, repeat the following steps:
   * Calculate the sum of the elements at indices **i**, **left**, and **right**: **sum = arr[i] + arr[left] + arr[right]**.
   * If the **sum** is equal to the target sum **X**, return **true** as a triplet is found.
   * If the **sum** is less than **X**, increment the **left** pointer to explore larger elements.
   * If the **sum** is greater than **X**, decrement the **right** pointer to explore smaller elements.
5. If no triplet is found after the iterations, return **false**.

The code uses the two-pointer technique to find the triplet efficiently with a time complexity of O(n^2), where n is the length of the input array.

### JAVA CODE

### import java.util.\*;

### import java.io.\*;

### import java.lang.\*;

### class Main

### {

### public static void main (String[] args) throws IOException {

### BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

### int t = Integer.parseInt(br.readLine().trim()); //Inputting the testcases

### while(t-->0){

### String inputLine[] = br.readLine().trim().split(" ");

### int n = Integer.parseInt(inputLine[0]);

### int X = Integer.parseInt(inputLine[1]);

### int A[] = new int[n];

### inputLine = br.readLine().trim().split(" ");

### for(int i=0; i<n; i++){

### A[i] = Integer.parseInt(inputLine[i]);

### }

### Solution ob=new Solution();

### boolean ans = ob.find3Numbers(A, n, X);

### System.out.println(ans?1:0);

### }

### }

### }

### //Actual code

### class Solution

### {

### //Function to find if there exists a triplet in the

### //array A[] which sums up to X.

### public static boolean find3Numbers(int arr[], int n, int X) {

### 

### Arrays.sort(arr);

### int sum=0;;

### for(int i=0;i<n-2;i++){

### int left=i+1;

### int right=n-1;

### while(left<right){

### sum=arr[i]+arr[left]+arr[right];

### if(sum==X){

### return true;

### }

### else if(sum<X)

### left++;

### else

### right--;

### }

### }

### return false;

### }

}