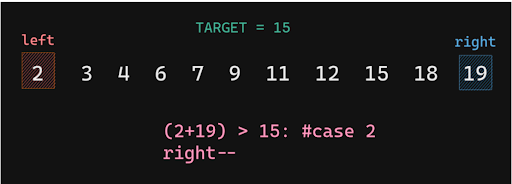
**1. PROBLEM DISCUSSION :**

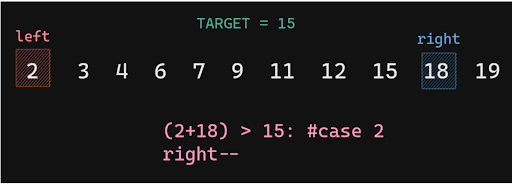
In this problem, we are given an array of distinct integers and a target. We have to find all the pairs of integers from the given array the sum of which equals the given target. For example, consider the given array : [ 7, 15, 3, 18, 6, 4, 19, 2, 12, 11, 9 ] Let’s suppose the target given to us is 15. Then the valid pairs are : [ 3, 12 ] [ 4, 11 ] [ 6, 9 ] The sum of all these pairs is equal to the target given i.e. 15. So we have to find all such pairs from the given array.

**2. APPROACH :**

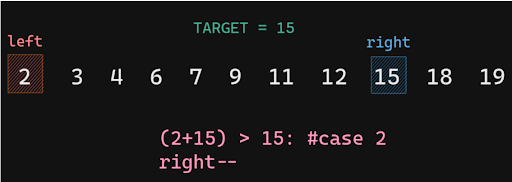
Let's consider the same example array where the numbers are: [ 7, 15, 3, 18, 6, 4, 19, 2, 12, 11, 9 ] Let’s suppose the target sum is 15. First, we have to sort the given array. Why to sort this array? We will come to the answer of this question later. So, the sorted array will be : [ 2, 3, 4, 6, 7, 9, 11, 12, 15, 18, 19 ] Let me explain an intuition before we get into what we'll do with the sorted array. Consider the following three numbers: a, b, and c, where a + b equals c. For instance, 5 + 10 = 15. In this case, a = 5, b = 10, and c = 15. If I increase 'a’ by some factor, then to maintain the expression a + b = c, what do you believe we should do about 'b'? We can't modify the target 'c.' c is constant here. Clearly, we need to reduce the value of 'b.' As a result, if 'a' is decreased to 7, 'b' should be reduced to 8. The conclusion is that if two numbers add up to a specific sum, increasing one will cause the other to decrease. We use two pointers. One to the leftmost element, and the other one to the rightmost element. Since the array is sorted the leftmost is going to be the smallest number and the rightmost is going to be the largest number. For this reason, we have to sort the given array so that when the sum is greater than the target value, we can reduce the sum by moving the right pointer to the left and vice versa. Now at every instant, we have to compare the sum of the elements pointed by the left and the right pointer with that of the target value. if sum == target: # CASE 1 left ++ right -- else if sum > target: # CASE 2 right -- else if sum < target # CASE 3 left ++ # CASE 1 If the sum is already equal to the target, we are pretty sure no other elements can form pairs with arr[left] and arr[right]. So we have to update both the pointers. That's why we have updated both left and right. Now the left was pointing to 0 initially. So we must always increment left. And right was pointing to the last index. So we must always decrement right. # CASE 2 Since the sum is already greater than the target. We have two options to normalize it i.e make it equal or at least try to bring it closer to target. Now arr[left+1] > arr[left]. So increasing left makes no sense as it makes the sum even larger. So ideally we should decrease the right. Hence right-- is performed. # CASE 3 This is the complete opposite of Case 2. Here since the sum is already less we must try to make it closer i.e increase the value. And we know arr[left+1] > arr[left]. So, incrementing left will be ideal. Let's dry run on our example array (already sorted):

****

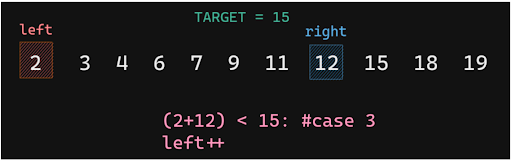
Here arr[left] + arr[right] > 15. Hence we have to decrement right.(that is our case 2)

****

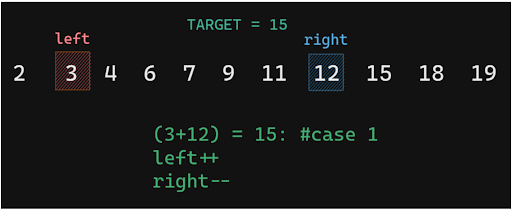
Now if you notice the prev sum was 21. Now it is reduced to 20. So we are gradually coming closer to our target sum. Hence our operations are working. Let's keep moving.

****

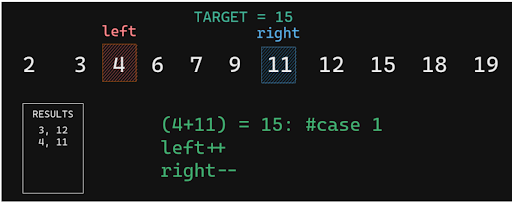
Still 2 + 15 > 15. So right should be decremented.

****

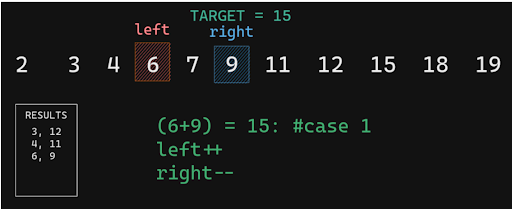
Now 2 + 12 is 14 which is less than 15. So this time according to case 3 we should increment left.

****

Bingo, we finally got our one pair. 3 + 12 is 15. So we will save it as a result. Also this time according to case 1. Left will be incremented as well as right will be decremented.

****

We again got another pair. 4, 11 which adds up to the target value.

****

And got our third pair as well. Now the question is how long should we keep doing this. Well, as long as left < right. So if now left and right points to the same value so we break from the loop. And we got our results. 3, 12 4, 11 6,9 I hope the approach to solve this problem is clear to you.

**3. CODE :**

Note : Now the approach to solve this problem is clear to you. So before reading the Code, I recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void targetSumPair(int[] arr, int target) {

Arrays.sort(arr);

int left = 0, right = arr.length - 1;

while (left < right) {

if (arr[left] + arr[right] == target) {

System.out.println(arr[left] + ", " + arr[right]);

left++;

right--;

} else if (arr[left] + arr[right] > target) {

right--;

} else {

left++;

}

}

}

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

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

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

arr[i] = scn.nextInt();

}

int target = scn.nextInt();

targetSumPair(arr, target);

}

}

For more clarity of the question, watch the question video

Play Video

**4. ANALYSIS :**

Time Complexity :

There are two parts to this. Sorting : The built-in sort function sorts in O(nlogn). Loop : The loop runs only once through all the elements. So this will contribute O(n) to the complexity. Hence overall complexity will be O(n + n logn) = O(nlogn). But if you are already given a sorted array then you can ignore the sorting and get the results in O(n).

Space complexity :

We do not use any auxiliary space hence the space complexity of this solution is constant i.e. space complexity is O(1).

Hope that you liked the article on “Target Sum Pair - 1”. Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website or in the content section on NADOS. You can suggest any improvements to the article on our telegram Channel, in the Nados doubt section or on the youtube channel’s comment section.