

/\

Interview: Patterns

Search Course

Pair with Target Sum (easy)

Remove Duplicates (easy)

Squaring a Sorted Array (easy)

Triplet Sum to Zero (medium)

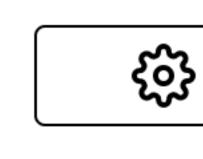
Triplet Sum Close to Target

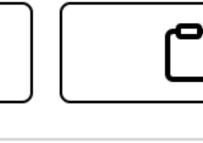
Introduction

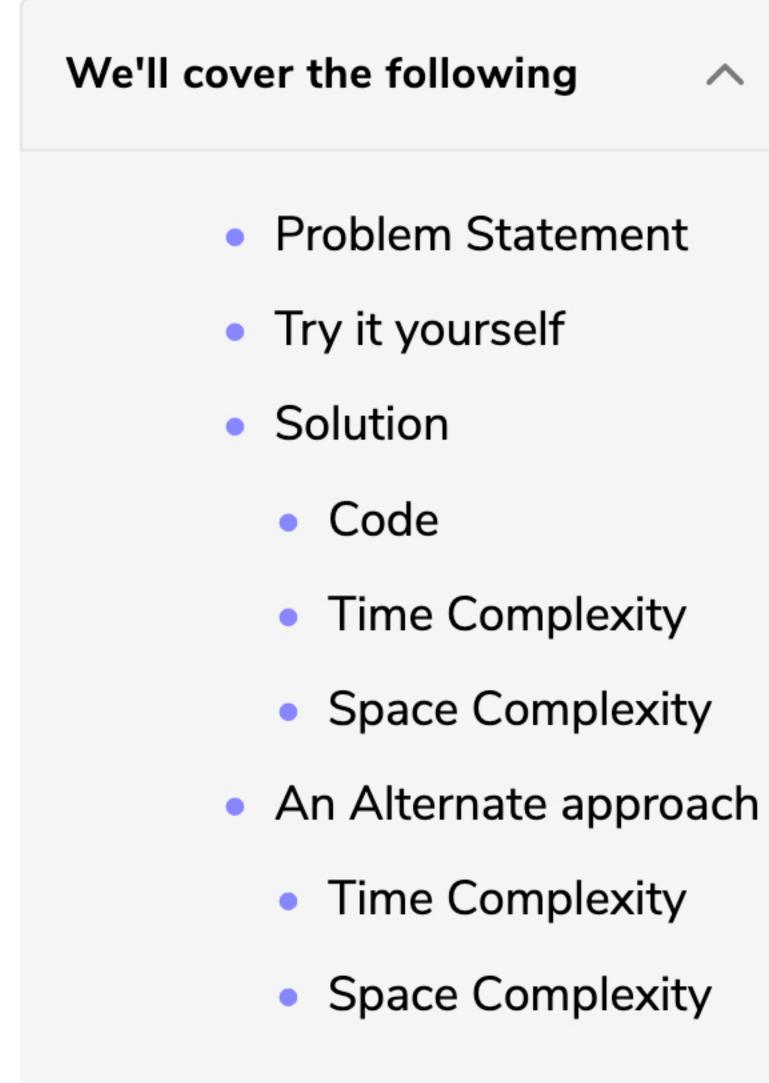
6% completed

for Coding Questions

? Ask a Question







Problem Statement

Given an array of sorted numbers and a target sum, find a pair in the array whose sum is equal to the given target.

Write a function to return the indices of the two numbers (i.e. the pair) such that they add up to the given target.

Example 1:

```
Input: [1, 2, 3, 4, 6], target=6
Output: [1, 3]
Explanation: The numbers at index 1 and 3 add up to 6: 2+4=6
```

Example 2:

```
Input: [2, 5, 9, 11], target=11
Output: [0, 2]
Explanation: The numbers at index 0 and 2 add up to 11: 2+9=11
```

Try it yourself# Try solving this question here:



Since the given array is sorted, a brute-force solution could be to iterate through the array, taking one

Solution

algorithm will be O(N*logN). Can we do better than this? We can follow the **Two Pointers** approach. We will start with one pointer pointing to the beginning of the array and another pointing at the end. At every step, we will see if the numbers pointed by the two pointers

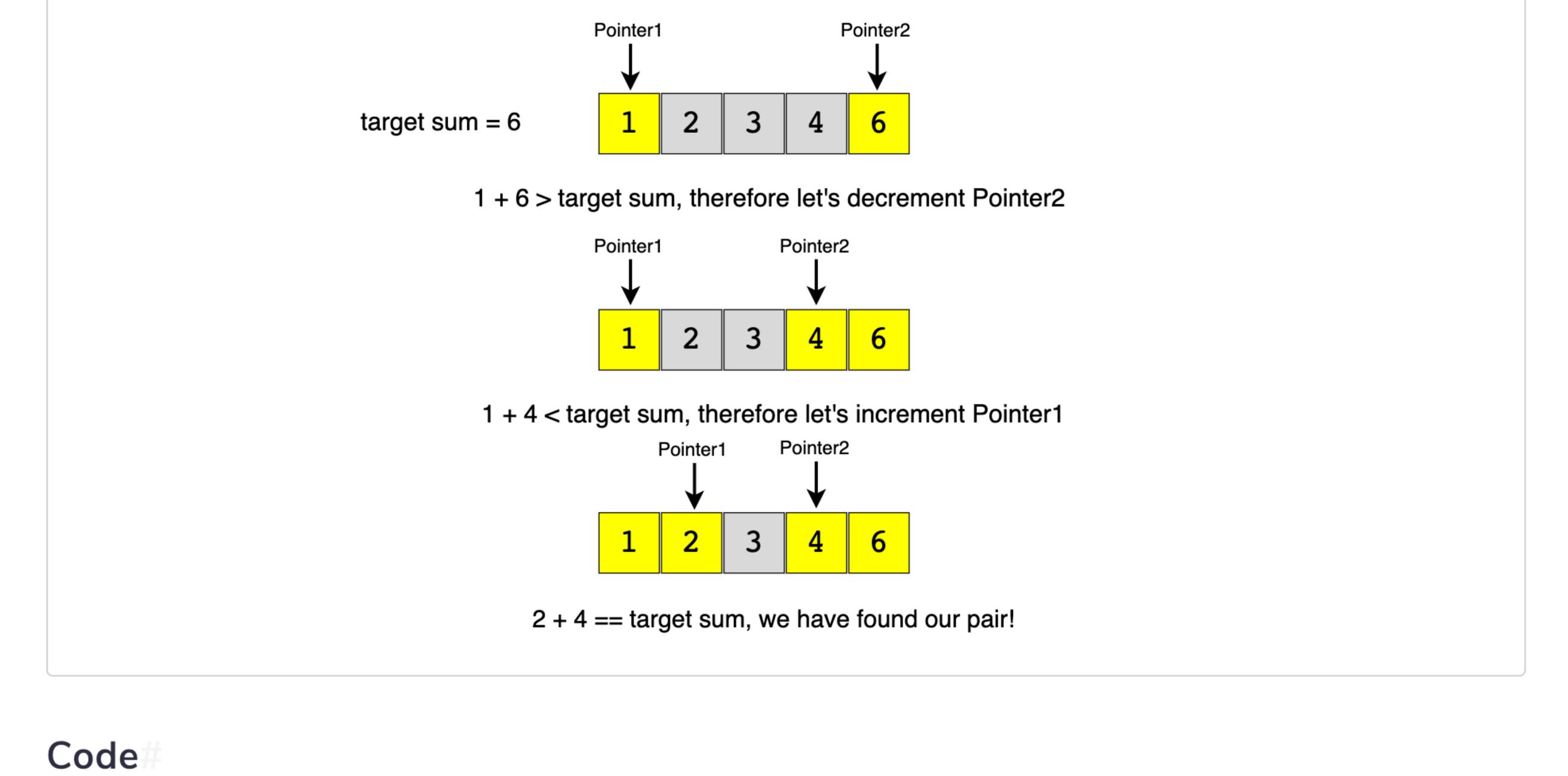
add up to the target sum. If they do, we have found our pair; otherwise, we will do one of two things:

number at a time and searching for the second number through Binary Search. The time complexity of this

1. If the sum of the two numbers pointed by the two pointers is greater than the target sum, this means that we need a pair with a smaller sum. So, to try more pairs, we can decrement the end-pointer.

- 2. If the sum of the two numbers pointed by the two pointers is smaller than the target sum, this means that we need a pair with a larger sum. So, to try more pairs, we can increment the start-pointer.
- that we need a pair with a larger sum. So, to try more pairs, we can increment the start-pointer.

 Here is the visual representation of this algorithm for Example-1:



Here is what our algorithm will look like:

Java
Python3
Class PairWithTargetSum {

```
<u>C</u> ∓
         public static int[] search(int[] arr, int targetSum) {
           int left = 0, right = arr.length - 1;
           while (left < right) {</pre>
             int currentSum = arr[left] + arr[right];
             if (currentSum == targetSum)
               return new int[] { left, right }; // found the pair
    9
             if (targetSum > currentSum)
   10
   11
               left++; // we need a pair with a bigger sum
   12
             else
   13
               right--; // we need a pair with a smaller sum
   14
           return new int[] \{-1, -1\};
   15
   16
   17
         public static void main(String[] args) {
           int[] result = PairWithTargetSum.search(new int[] { 1, 2, 3, 4, 6 }, 6);
           System.out.println("Pair with target sum: [" + result[0] + ", " + result[1] + "]");
   20
           result = PairWithTargetSum.search(new int[] { 2, 5, 9, 11 }, 11);
   21
           System.out.println("Pair with target sum: [" + result[0] + ", " + result[1] + "]");
   22
   23
   24
   Run
                                                                                                  Reset
                                                                                         Save
Time Complexity
The time complexity of the above algorithm will be O(N), where 'N' is the total number of elements in the
```

Space Complexity# The algorithm runs in constant space O(1).

given array.

An Alternate approach#

Instead of using a two-pointer or a binary search approach, we can utilize a HashTable to search for the

required pair. We can iterate through the array one number at a time. Let's say during our iteration we are at number 'X', so we need to find 'Y' such that "X+Y==Target". We will do two things here:

required pair.

public static int[] search(int[] arr, int targetSum) {

1 import java.util.HashMap;

class PairWithTargetSum {

2. Otherwise, insert "X" in the **HashTable**, so that we can search it for the later numbers. Here is what our algorithm will look like:

<u>C</u> →

2.2s

(!) Report an Issue

1. Search for 'Y' (which is equivalent to "Target-X") in the **HashTable**. If it is there, we have found the

```
HashMap<Integer, Integer> nums = new HashMap<>(); // to store numbers and their indices
 6
        for (int i = 0; i < arr.length; i++) {</pre>
          if (nums.containsKey(targetSum - arr[i]))
            return new int[] { nums.get(targetSum - arr[i]), i };
10
          else
            nums.put(arr[i], i); // put the number and its index in the map
11
12
        return new int[] \{ -1, -1 \}; // pair not found
13
14
15
      public static void main(String[] args) {
16
        int[] result = PairWithTargetSum.search(new int[] { 1, 2, 3, 4, 6 }, 6);
17
        System.out.println("Pair with target sum: [" + result[0] + ", " + result[1] + "]");
18
        result = PairWithTargetSum.search(new int[] { 2, 5, 9, 11 }, 11);
19
        System.out.println("Pair with target sum: [" + result[0] + ", " + result[1] + "]");
20
23
Run
                                                                                                 Reset
                                                                                       Save
```

Time Complexity

Output

Pair with target sum: [1, 3]

Pair with target sum: [0, 2]

Space Complexity# The space complexity will also be O(N), as, in the worst case, we will be pushing 'N' numbers in the

given array.

HashTable.

The time complexity of the above algorithm will be O(N), where 'N' is the total number of elements in the

```
← Back

Introduction

Remove Duplicates (easy)

✓ Mark as Completed
```