=

educative

Pair with Target Sum (easy)

Grokking the Coding Interview: Patterns for Coding Questions

**We'll cover the following**

7% completed

• Problem Statement

• Try it yourself

Q

Search Course

• Solution

**Pattern: Two Pointers**

• Code

• Time Complexity

Introduction

• Space Complexity

Pair with Target Sum (easy)

An Alternate approach

Remove Duplicates (easy)

• Time Complexity

Squaring a Sorted Array (easy)

• Space Complexity

Triplet Sum to Zero (medium)

Triplet Sum Close to Target (medium)

Problem Statement

Triplets with Smaller Sum (medium)

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

Subarrays with Product Less than a Target (medium)

Dutch National Flag Problem (medium)

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

Problem Challenge 1

**Example 1:**

Solution Review: Problem Challenge 1

Problem Challenge 2

Input: [1, 2, 3, 4, 6), target=6 Output: (1, 3) Explanation: The numbers at inde**x 1 and 3 add up to 6: 2+4=6,**

Solution Review: Problem Challenge 2

**Example 2:**

Problem Challenge 3 Solution Review: Problem Challenge 3

Input: (2, 5, 9, 11), target=11 Output: [0, 2] Explanation: The num**bers at index ® and 2 add up to 1**1: 2+9=11

**Pattern: Fast &** Slow **pointers**

Try it yourself

Q

Introduction

Linked List Cycle (easy)

Try solving this question here:

Start of Linked List Cycle (medium)

Java

Python3

JS US

C++

Happy Number (medium)

Middle of the Linked List (easy)

A WNP

const pair\_with\_targetsum = function(arr, target\_sum) {

*//* TODO: Write your code here

return (-1, -1]; }

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

TEST

SAVE

RESET

Solution Review: Problem Challenge 2

Solution

Problem Challenge 3

Solution Review: Problem Challenge 3

Since the given array is sorted, a brute-force solution could be to iterate through the array, taking one number at a time and searching for the second number through **Binary Search.** The time complexity of this algorithm will be *O(*N \* *log*N). Can we do better than this?

Pattern: Merge Intervals

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:

O

Introduction

Merge Intervals (medium)

Insert Interval (medium)

Intervals Intersection (medium)

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.

Conflicting Appointments (medium)

Problem Challenge 1

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

Solution Review: Problem Challenge 1

Problem Challenge 2

Pointer 1

Pointer2

Solution Review: Problem Challenge 2

target sum = 6

**3**

Problem Challenge 3

1 + 6 > target sum, therefore let's decrement Pointer2

Solution Review: Problem Challenge 3

Pointer

Pointer2

**Patter**n: Cyclic Sort

1

2

3

4

6

Introduction

1 + 4 <target sum, therefore let's increment Pointer 1

Pointer Pointer2

Cyclic Sort (easy)

Find the Missing Number (easy)

1|2|3

**3**

4

6

Find all Missing Numbers (easy)

Find the Duplicate Number (easy)

2 + 4 == target sum, we have found our pair!

Find all Duplicate Numbers (easy)

Problem Challenge 1

Code

Solution Review: Problem Challenge 1

Here is what our algorithm will look like:

Problem Challenge 2

Java

Python3

++

Solution Review: Problem

js JS

L-Il----

7

Challenge 2

**1**

Problem Challenge 3

Solution Review: Problem Challenge 3

**function pair\_with\_target\_sum arr, targetsum)**

let left = @

right = arr. Length - 1; while (left < right) {

const current Sum = arr[left] + a**rr[right];** if (currentSum **=== targetSum) {**

return [left, right);

**Pattern: In-place Reversal of a** LinkedList

if (target Sum > cur*r*entSum) {

left += 1; // we need a pair with a bigger sum } else {

right -= 1; // we need a pair with a smaller sum

Introduction

Reverse a Linked List (easy)

Q

Reverse a Sub-list (medium)

return (-1, -1);

Reverse e*v*ery K-element Sub-list (medium)

Problem Challenge 1

20 **21**

console.log(pair\_w**ith\_target\_sum ( [1, 2, 3, 4, 6), 6)); console.log(pair\_with\_target\_sum ( [2, 5, 9, 11), 11));**

Solution Review: Problem Challenge 1

Problem Challenge 2

RUN

SAVE

Solution Review: Problem Challenge 2

SAVE

RESET

RESET

Time Complexity

**Pattern: Tree Breadth**

First Search

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

O

Introduction

Space Complexity

Binary Tree Level Order Tra*v*ersal **(easy)**

The algorithm runs in constant space O(1).

Reverse Level Order Traversal (easy)

An Alternate approach

Zigzag Traversal (medium)

Level Averages in a Binary Tree **(easy)**

Instead of using a two-pointer or a binary search approach, we can utilize a Has**hTabl**e 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:

Minimum Depth of a Binary Tree **(easy)**

Level Order Successor (easy)

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

required pair. 2. Otherwise, insert "X" in the **HashTable,** so that we can search it for the later numbers.

Connect Level Order Siblings (medium)

Problem Challenge 1

Here is what our algorithm will look like:

Solution Review: Problem Challenge 1

Problem Challenge 2

Java

Python3

© C++

JS JS

Solution Review: Problem Challenge 2

1 function pair\_with\_target\_sum(arr, target Sum) {

const nums = {}; // to store numbers and their indices for (let i = ®; i < arr.length; i++) {

const num = arr[i]; if (target Sum - num in nums) {

return [nums [targetSum - **num], il;**

**Pattern: Tree Depth** First Search

MW

Introduction

nums (arr[i]] = i;

Binary Tree Path Sum (easy)

return (-1, -1];

11 }

All Paths for a Sum (medium)

**Explore**

Sum of Path Numbers (medium)

14 15

Path With Given Sequence (medium)

console.log(pair\_with\_target\_sum( [1, 2, 3, **4, 6), 6));** console.log(pair\_with\_target\_sum ( [2, 5, 9, **11), 11));**

Tracks

Count Paths for a Sum (medium)

**My Courses**

Problem Challenge 1

RUN

SAVE

RESET

0

Solution Review: Problem Challenge 1

Time Complexity

**Edpresso**

Problem Challenge 2

where 'N' is the total number of elements in the

Solution Review: Problem Challenge 2

The time complexity of the above algorithm will be *o* given array.

**Refer a Friend**

Space Complexity

**Pattern: Two Heaps**

The space complexity will also be *O*(N), as, in the worst case, we will be pushing ‘N’ numbers in the **HashTable.**

Introduction

Find the Median of a Number Stream (medium)

MARK AS COMPLETED

Sliding Window Median (hard)

Maximize Capital (hard)

+

Back

**Next**

**+**

Problem Challenge 1

Introduction

Remove Duplicates (easy)

Solution Review: Problem Challenge 1

A Report an Issue

2 Ask a Question

Pattern: Subsets

Introduction

+

Create

Subsets (easy) Subsets With Duplicates (easy)

Q