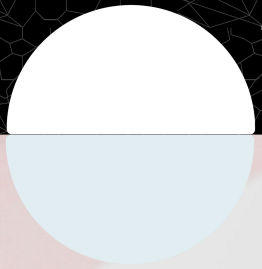
GuideBook

How to Solve Algorithm Problems?

Make Coding Interview Preparation Less Painful



WALEED KHAMIES

How to Solve Algorithm Problems

Make Coding Interview Preparation Less Painful

WALEED KHAMIES

This book is for sale at
http://leanpub.com/how-to-solve-algorithm-problems-book

This version was published on 2023-05-21 ISBN 978-1-7390105-1-5

Tweet This Book!

Please help WALEED KHAMIES by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Exciting news! Just got my copy of ”How to Solve Algorithm Problems” by @khamiesw! I can’t wait to dive into the strategies and techniques shared in this essential guide. Join me on this coding journey and grab your copy today! Happy coding! #TechBooks #NewRelease

The suggested hashtag for this book is #how-to-solve-algorithm-problems.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#how-to-solve-algorithm-problems

To my lovely parents, Ahmed, and Salha,

No words can fully describe the sincere gratitude and love I have for everything you have done for us. I won't be here without you, so thank you!

To the gang, my brothers,

Who are facing immense danger and hardship due to the ongoing military conflict in Sudan between the RSF militia and the Sudanese army. Please hold on, things will get better soon.

To my only land, Sudan,

I lose a piece of my heart every single day. You have been through a lot, so much pain and suffering, with rampant corruption and violence. But I believe that we will overcome these challenges and emerge stronger than before.

To my beloved reader,

I hope this guide serves as a valuable resource throughout your career. May it empower you to achieve your goals and reach new heights of success.

Table of Contents

GuideBook

How to Solve Algorithms Problems?

Make Coding Interview Preparation Less Painful

by Waleed Khamies

This book is also for sale on the following platforms:

Leanpub: https://leanpub.com/how-to-solve-algorithm-problems-book

Khamies' Store: https://shop.waleedkhamies.com/b/hsap

This version was published on 2023-05-21

www.waleedkhamies.com, info@waleedkhamies.com

Book Cover By : Author on Canva.

"" Clear Mind → Better Judgement → Better Outcome.""

NAVAL RAVIKANT

Preface

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

How to Solve Algorithm Problems

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

How Should You Read This Guide?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Notion Template: Learn Algorithms By Doing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

What Is The Goal Of This Guide?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

What Should You Expect After Completing This Guide?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Learning Cycle & Interview Assessment Factors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1 Understanding the Learning Cycle

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.1 Beginner Phase

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.1.1 General Characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.1.2 Focus Points

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.2 Experienced Phase

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.2.1 General Characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.2.2 Focus Points

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.3 Senior Phase

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.3.1 General Characterstics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.1.3.2 Focus Points

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2 Interview Assessment Factors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.1 Understanding of Algorithms and Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.2 Problem-Solving Skills

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.3 Attention To Detail

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.4 Code Efficiency

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.5 Time Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.6 Modular Code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.7 Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2.2.8 Communication

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Solving Algorithm Problems

This chapter discusses the key steps involved in solving algorithm problems. The first section covers essential steps, including understanding algorithms and data structures, problem-solving skills, attention to detail, code efficiency, time complexity analysis, modular code, debugging, and communication.

The second section focuses on applying these factors to solve problems related to the KSum Family. The chapter explores three critical problems in this family, providing step-by-step solutions that emphasize the key concepts discussed in the first section.

After completing this chapter, readers will have a better understanding of how to approach algorithm problems and solve them efficiently.

3.1 Steps to Solve Algorithm Problems

When attempting to solve an algorithm problem, it is important to adopt a systematic approach that assists in developing a robust solution. This means creating a solution that takes into account various corner cases and is efficient in terms of speed and memory usage. To achieve this, here are the steps to follow when solving an algorithm problem:

3.1.1 Understand the Problem

The first step in solving any problem is to understand it. You cannot solve something that you do not understand, and as they say: "Reading the question is half of the answer." It is important to pay attention to the details when encountering an algorithm problem.

3.1.2 Formalize the Problem

In this step, the job is to convert the problem information to a single question. This question will be in an input-output format, where one specifies what the input to the problem is and what the expected output is when solving the problem.

3.1.3 Repeat Reading the Question Yourself

Repeating a question several times guarantees that one will not miss any hidden information between the lines. In the binary search example, knowing the array is sorted helped in developing an extremely efficient algorithm.

3.1.4 Bring Input Examples

After understanding and formalizing the problem as a question, it is time to bring a handful of examples. These examples will serve as the expected inputs and outputs of the developed algorithm. The number of these input examples depends on the person, but it is better to have three examples, each one of them serving a specific goal, as follows:

3.1.4.1 Example 1: An Empty-Case Input

The algorithm will expect to receive an empty input such as an empty string, an empty list, or a number with a null value. Bringing this type of input includes these corner cases in the algorithm design process.

3.1.4.2 Example 2: A Medium-Case Input

This type of input example is dedicated to testing the algorithm in its most general flow. In other words, these are the inputs that the algorithm will usually deal with. There will be a concrete example of this type of input in the next section.

3.1.4.3 Example 3: A Corner-Case Input

The algorithm will expect to handle some special input examples that the general flow of the algorithm will not expect to see frequently. Examples of such corner cases include:

- Duplicated values in an array when the algorithm should expect to receive unique values.
- Negative inputs when only the algorithm should expect to receive positive inputs.

3.1.5 Develop a Brute-Force Solution

Now, it is time to develop a quick, dirty, and not practical solution for the problem. In this stage of solving the problem, there is no need to write an efficient code; only a code that works is required.

3.1.6 Analyze Time and Space Complexities For the Brute-Force Solution

After developing the brute-force solution, one has to analyze the time and space complexities. The first reason for this step is that it will tell the interviewer that one knows how much the algorithm will cost in terms of time and space. The second reason is that it will assist in optimizing the code in later stages.

3.1.7 Optimize The Brute-Force Solution

This step is the difference between a beginner candidate and an experienced candidate. The interviewer will try to see if you could optimize the brute-force solution and produce a better result. In this stage, you have to go over your brute-force solution line-by-line and look for operations that take too much time and space if the input size becomes very big.

3.1.8 Analyze Time and Space Complexities For the Optimized Solution

Again, you have to estimate the time and space complexities of your optimized solution. If you reached this stage, congratulations! That means you have passed your technical interview.

3.2 KSum Family Problems

In this section, we will delve into the process of solving algorithmic problems while considering the factors discussed earlier. We will closely examine three essential problems that belong to the KSum Family[1], and walk through their solutions while highlighting these principles.

KSum is one of the most asked questions among interviewers, because It contains multiple solution patterns that you can see among almost any other algorithm problem. Also, it allows the interviewers to easily extend their questions from one type of KSum problem to another version of the KSum problem. For these reasons, KSum represents a good example to study closely these assessement factors.

3.2.1 The 2Sum Problem

3.2.1.1 Problem Description

We are given a list of unique numbers and want to find the index of a number that matches a given target.

[1] https://bit.ly/uacifds-ksum

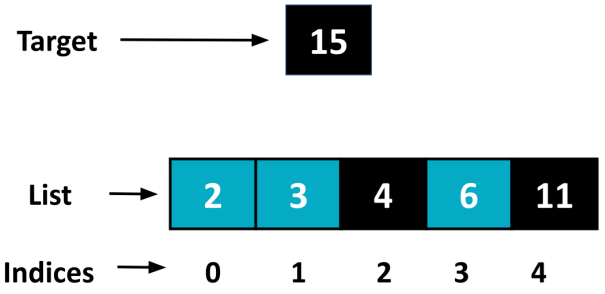


Figure 3.1: 2Sum Problem.

3.2.1.2 Problem Understanding

We have a list of integers that have duplicates, and we are interested to find k numbers from this list whose sum will be equal to a target.

Important Keywords: *1. non-unique integers*, *2. unique combination sets.*

3.2.1.3 I/O Examples

In/Out Examples.

```
Input: list of integers and a target.
Output: A collection of unique sets, where
each set has length = k.

Examples:
---------

Example 1: (Empty-Case Example)
Input: array = [ ], k = 4, target = 0
Output: [ ]

Example 2: (Medium-Case Example)
Input: array = [ 2, 4, 6,9,15 ], k = 2, target = 15
Output: [ [ 6,9] ]

Example 3: (Corner-Case Example)
Input: array = [1, 3, 5, 6, 7, 8, 10, 2, 2, 13], k = 4, target = 15
output: [[1, 2, 2, 10], [1, 2, 5, 7],
        [1, 3, 5, 6], [2, 2, 3, 8],
        [2, 2, 3, 6, 1]
```

3.2.1.4 Brute-Force Solution

To solve this problem, we see that we have access to a target. This target represents the sum of two numbers from the given array. Then, a simple solution to this problem will be by making two loop variables **(i, j)**.

Then, we look for a combination of two numbers that add up to the target value. In the following code, we can see the implementation of this algorithm.

The Brute-force solution of 2Sum problem.

```python
def two_sum_brute(nums, target):

    n = len(nums)

    for i in range(n):
        for j in range(i + 1, n):  # we start the second loop
                                   # from j = i+1.
            if nums[i] + nums[j] == target:
                return [i, j]
    return []
```

3.2.1.5 Complexity Analysis

Time Complexity

The time complexity of this solution will be ***O(n²)*** because we used two nested loops, which will result in this time complexity in the worst-case scenario.

Space Complexity

However, space complexity is ***O(1)*** which is great, because this solution will always use constant space regarding the size of the input. As we can see, ***O(n²)*** is not a good time complexity, especially if you want to run this code on an array with millions of numbers. In other words, If the algorithm takes 1 second to process 10 numbers, then that means it will take 100,000 seconds (~28 hours) to process 1 million numbers.

3.2.1.6 Optimized Solution

To optimize the brute-force code, we need to examine the algorithm line-by-line to see what operations take more resources. Below, we can see a table that outlines the time complexity of the main operations involved in the brute-force solution for the 2Sum problem.

ID	Operation	Time Complexity
1	**n = len(nums)**	**O(1)**
2	**for i in range(n):**	**O(n)**
3	**for j in range(i+1, n):**	**O(n)**
4	**if nums [i] + nums [j] == target:**	**O(1)**
5	**return [i,j]**	**O(1)**

From the table, we can see clearly that our algorithm has a bottleneck because of operations 2, and 3, because in the worst-case senario, operation 3 will have ***O(n.n) = O(n²)*** time complexity.

How we can remove this bottleneck and optimize these operations?

One of the most useful data structures that programmers like to use is the hashmap data structure. Hashmap has ***O(1)*** time complexity for the search operation of almost any element inside it. Then, we can work around the inefficiency of the brute-force solution by relying on space complexity.

Optimization Steps

- Store all the numbers inside a hashmap.
- Loop over the list items, and for each element, we will query our hashmap using a key with a value equal to ***target -nums[i]***, where i is the loop variable, and nums is the list of numbers.

Here is the optimized version of the code:

The optimized solution of 2Sum problem.

```
def twoSum_optimized(nums, target):

    mapper = {}
    for (i, e) in enumerate(nums):

       # Store the numbers inside the hashmap,
       # where the keys are the numbers,
       # and the values are the coressponding indicies.

        mapper[e] = i

    for i in range(len(nums)):
        b = target - nums[i]  # get the key value

        if b in mapper and mapper[b] != i:

            # if the key is existed in the hashmap and the its
            # value
            # does not equal to the index of the second number,
            #  we return
            # the indicies.

            return (i, mapper[b])
```

3.2.1.7 Complexity Analysis

Time Complexity

The time complexity of this solution will be ***O(n)*** because we used only one loop variable to iterate over the array elements.

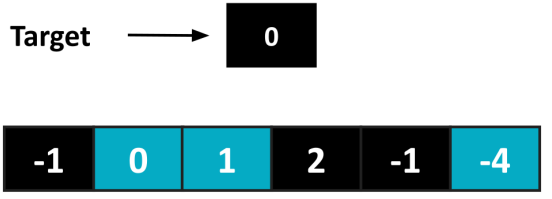
Space Complexity

But, space complexity is ***O(n)*** because we rely on the hashmap to increase the time efficiency of our algorithm.

3.2.2 The 3Sum Problem

3.2.2.1 Problem Description

We are given a list of non-unique integers, and we want to find three numbers that add up to zero. In such a way, the triplets should be all unique.



A Figure Illustrates 3Sum Problem.

3.2.2.2 Problem Understanding

We have a list of integers that are non-unique, and we are interested to find three numbers from this list whose sum will be equal to zero.

Important Keywords: *1. non-unique integers*, *2. unique triplets.*

3.2.2.3 I/O Examples

In/Out Examples.

```
Input: list of integers and a target.

Output: A collection of unique triplets.

Examples:
--------

Example 1: (Empty-Case Example)

Input: array = [ ], target = 0
Output: [ ]

Example 2: (Medium-Case Example)

Input: array = [2,3,4,-1,-2], target = 0
Output: [(-1,-2, 3)]

Example 3: (Corner-Case Example)

Input: array = [-1,0,1,2,-1,-4], target = 0
output: [ [-1,-1,2], [-1,0,1] ]
not [ [-1,-1,21, [-1,0,1], [1,0,-1] ]
```

3.2.2.4 Brute-Force Solution

The intuitive approach to solving the problem is making three loops variables **(i, j, k)**. Then, we look for a combination of three numbers that add up to zero.

In the following code lines, notice how the sorting we did initially allowed the Python set "result" to recognize duplicate combinations.

The Brute-force solution of 3Sum problem.

```
def three_sum_brute(nums):

    nums = sorted(nums)  # sort nums.
    n = len(nums)
    target = 0
    result = set()

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):

                combination_sum = nums[i] + nums[j] + nums[k]
                if combination_sum == target:
                    result.add((nums[i], nums[j], nums[k]))

    return result
```

3.2.2.5 Complexity Analysis

Time Complexity

The time complexity of this solution will be $O(n^3)$ because we used three nested loops. Notice that this is one of the worst time complexities, and the need for the optimization step is mandatory in this case.

Space Complexity

Space complexity is $O(n)$ which is fair enough because the code will use a linear space.

3.2.2.6 Optimized Solution

The next table outlines the time complexity of the main operations involved in the brute-force solution for the 3Sum problem. Let us examine our brute-force solution line-by-line to see what operations take more resources.

ID	Operation	Time Complexity
1	**nums = sorted(nums)**	**O(nlogn)**
2	**n = len(nums)**	**O(1)**
3	**for i in range(n):**	**O(n)**
4	**for j in range(i+1, n):**	**O(n)**
5	**for k in range(j+1, n):**	**O(n)**
6	**if combination_sum == target:**	**O(1)**
7	**result.add((nums[i], nums[j], nums[k]))**	**O(1)**

From the previous table, we can see clearly that our algorithm has a bottleneck because of operations 4 and 5. Because of the triple for-loops, the overall time complexity for this solution cost will cost ***O(n.n.n) = O(n³)***.

How can we speed up this naive solution, and make it robust in terms of time and memory?

One of the most useful algorithm techniques to solve algorithm problems is Two Pointers. We are going to solve this problem as follows:

1. Sort the array of integers in increasing order.
2. Create a set to hold the combinations of triplets called "result."
3. While looping over the array using a variable **(i)**:
    - Initialize two pointers, a left pointer with a value equal to i+1 and a right pointer with a value equal to n-1, where n = array size.
    - Make a nested loop and traverse the array using the two pointers. Because we are looking for nums[left] + nums[right] + nums[i]= 0, we will set our target = -nums[i].
    - If nums[left] + nums[right] = target, we will add this to the set of results.

- If nums[left] + nums[right] < target, that means we need to increase the left pointer to increase the value of this summation.
- Finally, if nums[left] + nums[right] > target, it means we need to decrease the right pointer to decrease the value of the summation and push it toward zero.
- In the end, we return the "result."

Here is how we can code this optimized version of the 3Sum problem:

The optimized solution of 2Sum problem.

```
def three_sum_optimized(nums):

    nums = sorted(nums)
    n = len(nums)
    result = set()

    for i in range(n):

        left = i + 1
        right = n - 1
        target = 0 - nums[i]

        while left < right:

            combination_sum = nums[left] + nums[right]
            if combination_sum == target:
                result.add((nums[i], nums[left], nums[right]))
                left += 1
                right -= 1
            elif combination_sum < target:

                left += 1
            else:
                right -= 1

    return result
```

3.2.2.7 Complexity Analysis

Time Complexity

We used two main operations, Sorting ***O(nlogn)*** and Searching ***O(n²)*** (two nested loops). However, the time complexity is still ***O(n²)***, because ***O(nlogn)*** + ***O(n²)*** = ***O(n²)***.

Space Complexity

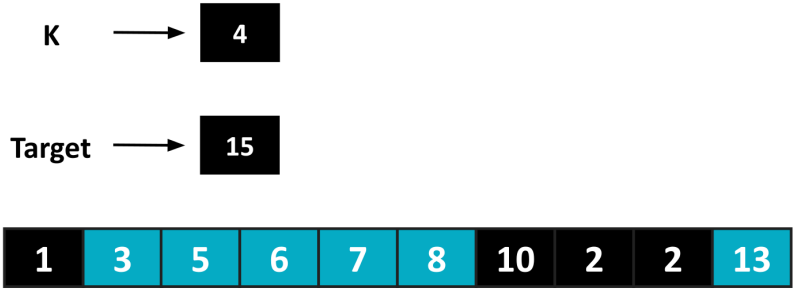
Space complexity is ***O(n)*** because we used the Python set **result** to store the triplets.

3.2.3 KSum

3.2.3.1 Problem Description

Given a list of non-unique integers and a target, we want to find any set of K numbers from this list that add up to that target, where those sets are all unique.

$$a_1 + a_2 + a_3 + ... + a_k = target$$



A Figure Illustrates KSum Problem.

3.2.3.2 Problem Understanding

We have a list of integers that have duplicates, and we are interested to find K numbers from this list whose sum will be equal to a target.

Important Keywords: *1. **non-unique integers**, 2. **unique combination sets**.*

3.2.3.3 I/O Examples

In/Out Examples.

```
Input: list of integers and a target.
Output: A collection of unique sets, where
each set has length = k.

Examples:
---------

Example 1: (Empty-Case Example)

Input: array = [ ], k = 4, target = 0

Output: [[ ]]

Example 2: (Medium-Case Example)

Input: array = [ 2, 4, 6,9,15 ], k = 2, target = 15

output: [[ 6,9 ]]

Example 3: (corner-Case Example)

Input: array = [1,3,5,6,7,8,10, 2, 2,13], k = 4, target = 15

Output: [[1, 2, 2, 10], [1, 2, 5, 7],
        [1, 3, 5, 6], [2, 2, 3, 8], [2, 2, 5, 6]]
```

3.2.3.4 Solution

As you can see in 3Sum, even though we have duplicates in the array, our code should handle such a situation by keeping all the sets unique. Also, we provide different ***K*** values. This is important later when you test your code against different KSum problems.

This problem is the general form of the 2Sum problem. To be able to infer the KSum solution from our previous solutions, we need to understand a very important thing first. Any KSum problem can be solved with this simple idea:

"Reduce KSum to 2Sum problem, then solve it."

First, let us define some terms:

$$t_n = \text{The uppermost n degree target}$$

$$K = \text{The degree of KSum problem}$$

Mathematically, the 2Sum problem could be formalized as:

$$a_1 + a_2 = t_1$$

While 3Sum is formalized as:

$$a_1 + a_2 + a_3 = t_2$$

And 4Sum is formalized as:

$$a_1 + a_2 + a_3 + a_4 = t_3$$

Then to solve 4Sum, we can rearrange these equations as follows:

$$t_2 = t_3 - a_4$$

$$t_1 = t_2 - a_3$$

$$a_1 = t_1 - a_2$$

Then, here is the summary of the KSum algorithm:

- Starting with the uppermost degree target

$$K = n$$

$$t = t_n$$

- Recursively, calculate the upper-degree targets (i.e t_3, t_2) and the missing numbers (i.e. a_4, a_3).
- When $K = 2$, call the 2Sum function to calculate the last missing numbers (a_1, a_2).

To code **KSum** in Python, we defined two functions, we will introduce two functions. The first function handles the 2Sum operation, while the second function handles the KSum operation.

The 2Sum function:

2Sum using Two Pointers technique

```
def two_sum_2pointers(
    nums,
    target,
    path,
    result,
    ):

    left = 0
    right = len(nums) - 1

    while left < right:

        if nums[left] + nums[right] == target:
            result.append(path + [nums[left], nums[right]])

      # ----------- To filter the duplicate cases ---------

            while left < right and nums[left] == nums[left + 1]:
                left += 1
            while left < right and nums[right] == nums[right - 1]:
                right -= 1

      # -----------------  End of Filtering ----------------
```

```
24
25              left += 1
26              right -= 1
27
28          if nums[left] + nums[right] < target:
29              left += 1
30
31          if nums[left] + nums[right] > target:
32              right -= 1
```

As you can see, we use the Two Pointers technique to implement this version of 2Sum. In chapter 5, we will learn more about this technique in detail.

The KSum function:

KSum using Two Pointers technique

```
1  def k_sum(
2      nums,
3      k,
4      target,
5      path,
6      result,
7      ):
8      # stop early if the number of summation variables is less
9      # than 2 or greater than the length of input array (nums).
10     if len(nums) < k or k < 2:
11         return
12
13     if k == 2:  # Call 2Sum_2pointers function when degree (k) = 2
14         two_sum_2pointers(nums, target, path, result)
15     else:
16
17     # recursively reduce k (degree) value and estimate
18     # upper-degree targets (target-nums[i]) till we reach
19     # k =2 to solve it as a two_sum problem.
20
```

```
        for i in range(len(nums)):
        # remove the duplicates in nums.
            if i == 0 or i > 0 and nums[i - 1] != nums[i]:
                k_sum(nums[i + 1:], k - 1, target - nums[i], path
                      + [nums[i]], result)
```

The ***k_sum*** function keeps calculating the required numbers by reducing the value of k by one till ***K = 2***. Then it calls the ***two_sum_2pointers*** function to calculate the last two missing numbers.

3.2.3.5 Complexity Analysis

Time Complexity

The time complexity of this solution will be:

$$O(n^{k-1})$$

because we used recursive function calls to solve for the k elements.

Space Complexity

In general, the code will have a space complexity equal to ***O(n)*** which represents the space needed to store the results. (ignoring the function stack memory and the path variable)

4 | FGCC FRAMEWORK

PHOTO BY SHARON PITTAWAY ON UNSPLASH ON UNSPLASH

FGCC FRAMEWORK

4.1. FGCC Framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.1.1 What is FGCC?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.1.2 A Mental Framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.1.3 Steps To Apply FGCC Framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2. FGCC In Practice

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.1 Introduction to Backtracking Technique

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.1.1 Finding Permutations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.1.2 Finding Combinations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.1.3 Letter Combinations of a Phone Number

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.2 The Pillars of the FGCC Framework

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.2.1 Focus (F)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

1. Problem Formulation Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

2. Solution Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

A. Finding Permutations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

B. Finding Combinations

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

C. Letter Combinations of a Phone Number

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.2.2 Group (G) & Convert (C)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

4.2.2.3 Communication

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5. Top #3 Algorithm Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.1 Two Pointers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.1.1 Code Example

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.1.2 Usage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.1.3 Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.2 Breadth-First Search (BFS)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.2.1 Code Example

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.2.2 Usage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.2.3 Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.3 Depth-First Search (DFS)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.3.1 Usage

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

5.3.2 Data Structures

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6. Supplements

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.1 Detect a Linked List Cycle

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.1.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.1.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.1.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.1.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.2 Remove the Nth Node From the End of a Linked List

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.2.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.2.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.2.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.2.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.3 Swapping Linked List Node Pairs

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.3.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.3.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.3.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.3.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.4 Validate Binary Search Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.4.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.4.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.4.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.4.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.5 Same Binary Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.5.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.5.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.5.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.5.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.6 Symmetric Binary Tree

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.6.1 Problem Description

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.6.2 I/O Examples

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.6.3 Solution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

6.6.4 Complexity Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Time Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Space Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

References

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/how-to-solve-algorithm-problems-book

Acknowledgments

I am grateful to my supervisors for their support and mentorship, my friends for their encouragement, and my family for their unconditional love. I also appreciate the readers for their valuable feedback and support. Thank you all for being part of my journey.

About the Author

Waleed Khamies is an applied machine learning scientist with a deep passion for exploring the realm of semantic representations in multimodal data. With a strong background in research and engineering, Waleed has gained invaluable experience as a research engineer intern at Mila and a research associate intern at Brown University's Robotics Lab.

Holding a master's degree in Mathematical Sciences/Machine Learning from AMMI and a bachelor's degree in Electrical and Electronics Engineering from UofK, Waleed is recognized for his expertise in reinforcement learning and the development of cutting-edge deep learning models for robotics applications. His work has been published in reputable workshops, and he actively shares his knowledge and insights through his engaging blog and newsletter.

For more information about his work and interests, please visit his website at waleedkhamies.com.