# Section 5: Problem Solving Patterns

## 25. PREREQUISITES

Prerequisites For This Section

It is strongly recommended you first complete the following sections before working through this section:

* Section 1: BIG O NOTATION
* Section 4: PROBLEM SOLVING APPROACH

## 26. Intro to Problem Solving Patterns

Slides: <https://cs.slides.com/colt_steele/problem-solving-patterns>

1. **Devise** a plan for solving problems
2. **Master** common problem solving patterns

#### SOME PATTERNS…

* Frequency Counter
* Multiple Pointers
* Sliding Window
* Divide and Conquer
* Dynamic Programming
* Greedy Algorithms
* Backtracking
* Many more!

## 27.Frequency Counter Pattern

#### FREQUENCY COUNTERS

This pattern uses objects or sets to collect values/frequencies of values

This can often avoid the need for nested loops or O(N²) operations with arrays /strings

AN EXAMPLE

Write a function called same, which accepts two arrays. The function should return true if every value in the array has its corresponding value squared in the second array. The frequency of values must be the same.

##### same\_naive\_solution

function same(arr1, arr2){

if(arr1.length !== arr2.length){

return false;

}

for(let i = 0; i < arr1.length; i++){

let correctIndex = arr2.indexOf(arr1[i] \*\* 2)

if(correctIndex === -1) {

return false;

}

console.log(arr2);

arr2.splice(correctIndex,1)

}

return true;

}

same([1,2,3,2], [9,1,4,4])

##### same\_refactored\_solution

function same(arr1, arr2){

if(arr1.length !== arr2.length){

return false;

}

let frequencyCounter1 = {}

let frequencyCounter2 = {}

for(let val of arr1){

frequencyCounter1[val] = (frequencyCounter1[val] || 0) + 1

}

for(let val of arr2){

frequencyCounter2[val] = (frequencyCounter2[val] || 0) + 1

}

console.log(frequencyCounter1);

console.log(frequencyCounter2);

for(let key in frequencyCounter1){

if(!(key \*\* 2 in frequencyCounter2)){

return false

}

if(frequencyCounter2[key \*\* 2] !== frequencyCounter1[key]){

return false

}

}

return true

}

same([1,2,3,2,5], [9,1,4,4,11])

## 28. Frequency Counter: Anagram Challenge

Given two strings, write a function to determine if the second string is an anagram of the first. An anagram is a word, phrase, or name formed by rearranging the letters of another, such as cinema , formed from iceman.

function validAnagram(str1, str2){

*// assume the string contains only lowercase alphabets*

var arr1 = [...str1.toLowerCase()]

var arr2 = [...str2.toLowerCase()]

*// make object to return at end*

if (arr1.length !== arr2.length){

return false;

}

*// loop over string, for each character...*

for (let i =0; i<arr1.length; i++) {

let foundIndex = arr2.indexOf(arr1[i])

*// if the arr1 is not found in arr2 ,false*

if(foundIndex === -1) {

return false

}

console.log(arr2)

*// delete the found arr2 item in array*

arr2.splice(foundIndex,1)

}

*// if arr1 could be found in arr2. return*

return true

}

validAnagram('str1','str2')

validAnagram('qwerty','qeywrt')

## 29. Anagram Challenge Solution

function validAnagram(first, second) {

if (first.length !== second.length) {

return false;

}

const lookup = {};

for (let i = 0; i < first.length; i++) {

let letter = first[i];

*// if letter exists, increment, otherwise set to 1*

lookup[letter] ? (lookup[letter] += 1) : (lookup[letter] = 1);

}

console.log(lookup);

for (let i = 0; i < second.length; i++) {

let letter = second[i];

*// can't find letter or letter is zero then it's not an anagram*

if (!lookup[letter]) {

return false;

} else {

lookup[letter] -= 1;

}

}

return true;

}

*// {a: 0, n: 0, g: 0, r: 0, m: 0,s:1}*

validAnagram('anagrams', 'nagaramm');

## 30. Multiple Pointers Pattern

Creating **Ponters** or values that correspond to an index or position and move towards the beginning, end or middle based on a certain condition

**Very** efficient for solving problems with minimal space complexity as well

AN EXAMPLE

Write a function called **sumZero** which accepts a **sorted** array of integers. The funcion should find the **first** pair where the dum is 0. Return an array that includes both values that sum to zero or undefined if a pari does not exist

Naive Solution

function sumZero(arr){

for(let i = 0; i < arr.length; i++){

for(let j = i+1; j < arr.length; j++){

if(arr[i] + arr[j] === 0){

return [arr[i], arr[j]];

}

}

}

}

sumZero([-4,-3,-2,-1,0,1,2,5])

Time Complexity - O(N2)

Space Complexity - O(1)

Refactor

function sumZero(arr) {

let left =0

let right = arr.length -1

while (left < right) {

let sum = arr[left] + arr[right]

if(sum === 0) {

return [arr[left],arr[right]]

}else if (sum>0 ) {

right --

}else {

left++

}

}

}

sumZero([-4,-3,-2,-1,0,1,2,5])

Time Complexity - O(N)

Space Complexity - O(1)

## 31. Multiple Pointers: Count Unique Values Challenge

Implement a function called **countUniqueValues**, which acceprs a sorted array, and counts the unique values in the array. There can be negative numbers in the array , but it will always be sorted.

## 32. Count Unique Values Solution

function countUniqueValues(arr){

if(arr.length === 0) return 0;

var i = 0;

for(var j = 1; j < arr.length; j++){

if(arr[i] !== arr[j]){

i++;

arr[i] = arr[j]

}

}

return i + 1;

}

countUniqueValues([1,2,2,5,7,7,99])

## 33. Sliding Window Pattern

SLIDING WINDOW

This pattern involves creating a window which can either be an array or number from one position to another

Depending on a certain condition, the window either increases or closes(and a new window is created)

Very useful for keeping reack of a subset of data in an array/string etc.

AN EXAMPLE

Write a function called maxSubarrauSum which accepts an array of integers and a number called n. The function should calculate the maximum sum of n consecutive elements in the array.

## 34. Divide and Conquer Pattern

This pattern involves dividing a data set into smaller chunks and then repeating a process with a subset of data.

This pattern can tremendously **decrease time complexity**

AN EXAMPLE

Given a sorted array of integers, write a function called search, that accepts a value and returns the index where the value passed to the function is located. If the value is not found , return -1

###### A native solution

function search (arr, val){

for (let i =0; i< arr.length; i++) {

if (arr[i] === val) {

return i;

}

}

return -1;

}

Linear Search

Time Complexity O(N)

###### Refactor

function search (array,val) {

let min =0;

let max =array.length -1;

while (min<= max) {

let middle = Math.floor((min+max)/2);

let currentElement = array[middle];

if (array[middle]< val) {

min = middle +1;

} else if (array[middle] > val) {

max = middle -1;

} else {

return middle;

}

}

return -1;

}

Time Complexity - Log(N) - Binary Search!