NEED TO MASTER THESE PATTERNS

**1. 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(N^2) 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 it’s corresponding value squared in the second array. The frequency of values must be the same.

**2. Multiple pointers**

Creating **pointers** 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 function should find the **first** pair where the sum is 0. Return an array that includes both values that sum to zero or undefined if a pair does not exist.

sumZero([-3, -2, -1, 0, 1, 2, 3]) 🡪 return [-3,3]

sumZero([-2, 0, 1, 3]) 🡪 return undefined

sumZero([1, 2, 3]) 🡪 return undefined

**3. 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 track of a subset of data in array/string etc.

An example: Write a function called maxSubarraySum 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.

maxSubarraySum([1, 2, 5, 2, 8, 1, 5], 2) 🡪 return 10

maxSubarraySum([1, 2, 5, 2, 8, 1, 5], 4) 🡪 return 17

maxSubarraySum([4, 2, 1, 6], 1) 🡪 return 6

maxSubarraySum([4, 2, 1, 6, 2], 4) 🡪 return 13

maxSubarraySum([], 4) 🡪 return null

**4. Divide and Conquer**

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 Search([1 ,2 , 3, 4 ,5 ,6], 4) 🡪 return 3

Search([1 ,2 , 3, 4 ,5 ,6], 6) 🡪 return 5

Search([1 ,2 , 3, 4 ,5 ,6], 11) 🡪 return -1