**Searching Algorithms**

**Searching algorithms** are essential tools in computer science used to locate specific items within a collection of data. In this tutorial, we are mainly going to focus upon searching in an array. When we search an item in an array, there are two most common algorithms used based on the type of input array.

* [**Linear Search**](https://www.geeksforgeeks.org/dsa/linear-search/): It is used for an unsorted array. It mainly does one by one comparison of the item to be search with array elements. It takes linear or O(n) Time.
* Linear Search, we iterate over all the elements of the array and check if it the current element is equal to the target element. If we find any element to be equal to the target element, then return the index of the current element. Otherwise, if no element is equal to the target element, then return -1 as the element is not found. Linear search is also known as **sequential search**.
* [**Binary Search**](https://www.geeksforgeeks.org/dsa/binary-search/) : It is used for a sorted array. It mainly compares the array's middle element first and if the middle element is same as input, then it returns. Otherwise it searches in either left half or right half based on comparison result (Whether the mid element is smaller or greater). This algorithm is faster than linear search and takes O(Log n) time.
* **Binary Search (Sorted array needed)**is a [searching algorithm](https://www.geeksforgeeks.org/dsa/searching-algorithms/)that operates on a sorted or monotonic search space, repeatedly dividing it into halves to find a target value or optimal answer in logarithmic time O(log N).

A screenshot of a computer program

AI-generated content may be incorrect.

Eg. **Array:**

Index: 0 1 2 3 4 5 6 7 8 9

Value: 2 5 8 12 16 23 38 56 72 91

We are searching for **23**.

**Step 1: Initial boundaries**

* low = 0 → index of first element (2)
* high = 9 → index of last element (91)

**Step 2: Find middle**

mid = Math.floor((low + high) / 2) = Math.floor((0 + 9)/2) = 4

* arr[4] = 16
* Compare with key 23:  
  23 > 16 → key must be in the **right half**.

So, update boundaries:  
low = mid + 1 = 5, high = 9.

**Step 3: Find new middle**

mid = Math.floor((5 + 9) / 2) = 7

* arr[7] = 56
* Compare with key 23:  
  23 < 56 → key must be in the **left half**.

So, update boundaries:  
low = 5, high = mid - 1 = 6.

**Step 4: Find new middle**

mid = Math.floor((5 + 6) / 2) = 5

* arr[5] = 23
* This matches the key ✅

**🔹 Result**

Key 23 is found at **index 5**.

**Two Pointers Technique**

The Two-Pointers Technique is a simple yet powerful strategy where you use two indices (pointers) that traverse a data structure—such as an array, list, or string—either toward each other or in the same direction to solve problems more efficiently

Two pointers is really an easy and effective technique that is typically used for [Two Sum in Sorted Arrays](https://www.geeksforgeeks.org/dsa/pair-with-given-sum-in-sorted-array-two-sum-ii/), [Closest Two Sum](https://www.geeksforgeeks.org/dsa/two-elements-whose-sum-is-closest-to-zero/), [Three Sum](https://www.geeksforgeeks.org/dsa/find-a-triplet-that-sum-to-a-given-value/), [Four Sum](https://www.geeksforgeeks.org/dsa/find-four-numbers-with-sum-equal-to-given-sum/), [Trapping Rain Water](https://www.geeksforgeeks.org/dsa/trapping-rain-water/) and many other popular interview questions.

**When to Use Two Pointers:**

* **Sorted Input :**If the array or list is already sorted (or can be sorted), two pointers can efficiently find pairs or ranges. Example: Find two numbers in a sorted array that add up to a target.
* **Pairs or Subarrays :**When the problem asks about two elements, subarrays, or ranges instead of working with single elements. Example: Longest substring without repeating characters, maximum consecutive ones, checking if a string is palindrome.
* **Sliding Window Problems :**When you need to maintain a window of elements that grows/shrinks based on conditions. Example: Find smallest subarray with sum ≥ K, move all zeros to end while maintaining order.
* **Linked Lists (Slow–Fast pointers) :**Detecting cycles, finding the middle node, or checking palindrome property. Example: Floyd’s Cycle Detection Algorithm (Tortoise and Hare).

***Input****: arr[] = [10, 20, 35, 50], target =70****Output****: true****Explanation*** *: There is a pair (20, 50) with given target.*

// Function to check whether any pair exists

// whose sum is equal to the given target value

function twoSum(arr, target) {

let n = arr.length;

// Iterate through each element in the array

for (let i = 0; i < n; i++) {

// For each element arr[i], check every

// other element arr[j] that comes after it

for (let j = i + 1; j < n; j++) {

// Check if the sum of the current pair

// equals the target

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

return true;

}

}

}

// If no pair is found after checking

// all possibilities

return false;

}

let arr = [0, -1, 2, -3, 1];

let target = -2;

// Call the twoSum function and print the result

if (twoSum(arr, target))

console.log("true");

else

console.log("false");