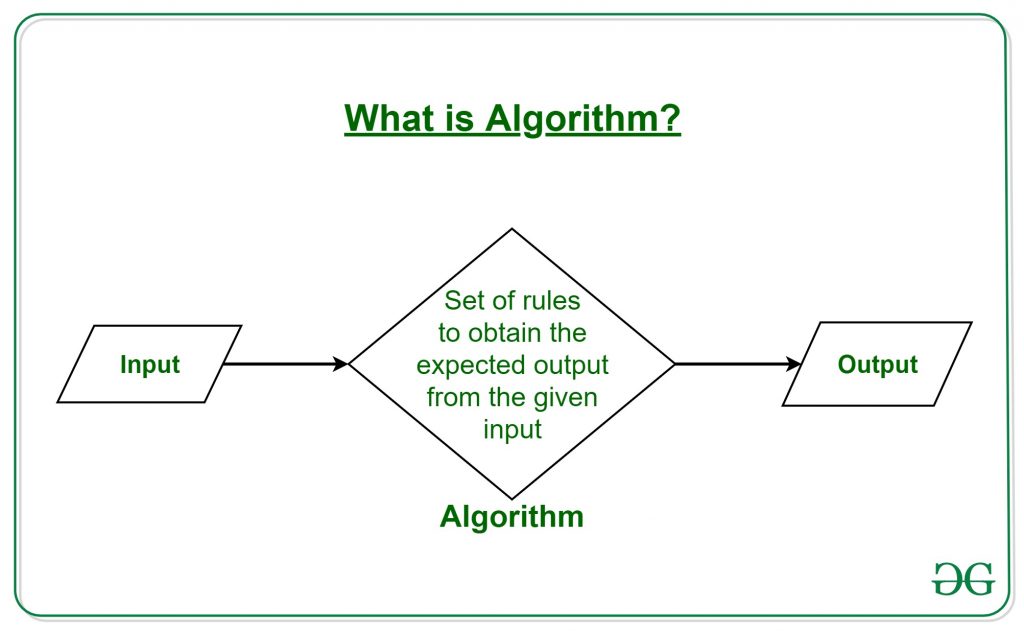
**Algorithms Tutorial**

**Algorithm**is a step-by-step procedure for solving a problem or accomplishing a task. In the context of data structures and algorithms, it is a set of well-defined instructions for performing a specific computational task. Algorithms are fundamental to computer science and play a very important role in designing efficient solutions for various problems. Understanding algorithms is essential for anyone interested in mastering data structures and algorithms.



## How do Algorithms Work?

Algorithms typically follow a logical structure:

* **Input:**The algorithm receives input data.
* **Processing:** The algorithm performs a series of operations on the input data.
* **Output:**The algorithm produces the desired output.

## Characteristics of an Algorithm:

* **Clear and Unambiguous:** The algorithm should be unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
* **Well-defined Inputs:**If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
* **Well-defined Outputs:**The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
* **Finiteness:**The algorithm must be finite, i.e. it should terminate after a finite time.
* **Feasible:**The algorithm must be simple, generic, and practical, such that it can be executed using reasonable constraints and resources.
* **Language Independent:**Algorithm must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

## What is the Need for Algorithms?

Algorithms are essential for solving complex computational problems efficiently and effectively. They provide a systematic approach to:

* **Solving problems:** Algorithms break down problems into smaller, manageable steps.
* **Optimizing solutions:** Algorithms find the best or near-optimal solutions to problems.
* **Automating tasks:** Algorithms can automate repetitive or complex tasks, saving time and effort.

## Examples of Algorithms

Below are some example of algorithms:

* **Sorting algorithms:**Merge sort, Quick sort, Heap sort
* **Searching algorithms:** Linear search, Binary search, Hashing
* **Graph algorithms:**Dijkstra’s algorithm, Prim’s algorithm, Floyd-Warshall algorithm
* **String matching algorithms:** Knuth-Morris-Pratt algorithm, Boyer-Moore algorithm

## How to Write an Algorithm?

To write an algorithm, follow these steps:

* **Define the problem:**Clearly state the problem to be solved.
* **Design the algorithm:** Choose an appropriate algorithm design paradigm and develop a step-by-step procedure.
* **Implement the algorithm:** Translate the algorithm into a programming language.
* **Test and debug:** Execute the algorithm with various inputs to ensure its correctness and efficiency.
* **Analyze the algorithm:** Determine its time and space complexity and compare it to alternative algorithms.

**Searching algorithms** are essential tools in computer science used to locate specific items within a collection of data. These algorithms are designed to efficiently navigate through data structures to find the desired information, making them fundamental in various applications such as **databases, web search engines**, and more.

## What is Searching?

Searching is the fundamental process of **locating a specific element or item within a collection of data**. This collection of data can take various forms, such as arrays, lists, trees, or other structured representations. The primary objective of searching is to determine whether the desired element exists within the data, and if so, to identify its precise location or retrieve it. It plays an important role in various computational tasks and real-world applications, including information retrieval, data analysis, decision-making processes, and more.

## Searching terminologies:

### Target Element:

In searching, there is always a specific target element or item that you want to find within the data collection. This target could be a value, a record, a key, or any other data entity of interest.

### Search Space:

The search space refers to the entire collection of data within which you are looking for the target element. Depending on the data structure used, the search space may vary in size and organization.

### Complexity:

Searching can have different levels of complexity depending on the data structure and the algorithm used. The complexity is often measured in terms of time and space requirements.

### Deterministic vs. Non-deterministic:

Some searching algorithms, like[binary search](https://www.geeksforgeeks.org/binary-search/), are deterministic, meaning they follow a clear, systematic approach. Others, such as linear search, are non-deterministic, as they may need to examine the entire search space in the worst case.

## Importance of Searching in DSA:

* **Efficiency:** Efficient searching algorithms improve program performance.
* **Data Retrieval:**Quickly find and retrieve specific data from large datasets.
* **Database Systems:** Enables fast querying of databases.
* **Problem Solving:**Used in a wide range of problem-solving tasks.

## Applications of Searching:

Searching algorithms have numerous applications across various fields. Here are some common applications:

* **Information Retrieval:**Search engines like Google, Bing, and Yahoo use sophisticated searching algorithms to retrieve relevant information from vast amounts of data on the web.
* **Database Systems:** Searching is fundamental in database systems for retrieving specific data records based on user queries, improving efficiency in data retrieval.
* **E-commerce:**Searching is crucial in e-commerce platforms for users to find products quickly based on their preferences, specifications, or keywords.
* **Networking:** In networking, searching algorithms are used for routing packets efficiently through networks, finding optimal paths, and managing network resources.
* **Artificial Intelligence:**Searching algorithms play a vital role in AI applications, such as problem-solving, game playing (e.g., chess), and decision-making processes
* **Pattern Recognition:** Searching algorithms are used in pattern matching tasks, such as image recognition, speech recognition, and handwriting recognition.

**Linear Search Algorithm – Data Structure and Algorithms Tutorials**

**Linear Search** is defined as a sequential [search algorithm](https://www.geeksforgeeks.org/searching-algorithms/) that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. In this article, we will learn about the basics of Linear Search Algorithm, Applications, Advantages, Disadvantages, etc. to provide a deep understanding of Linear Search.

## What is Linear Search?

**Linear Search**is a method for searching an element in a collection of elements. In Linear Search, each element of the collection is visited one by one in a sequential fashion to find the desired element. Linear Search is also known as **Sequential Search.**

## Algorithm for Linear Search:

The algorithm for linear search can be broken down into the following steps:

* **Start:** Begin at the first element of the collection of elements.
* **Compare:** Compare the current element with the desired element.
* **Found:** If the current element is equal to the desired element, return true or index to the current element.
* **Move:**Otherwise, move to the next element in the collection.
* **Repeat:** Repeat steps 2-4 until we have reached the end of collection.
* **Not found:** If the end of the collection is reached without finding the desired element, return that the desired element is not in the array.

## How Does Linear Search Algorithm Work?

In Linear Search Algorithm,

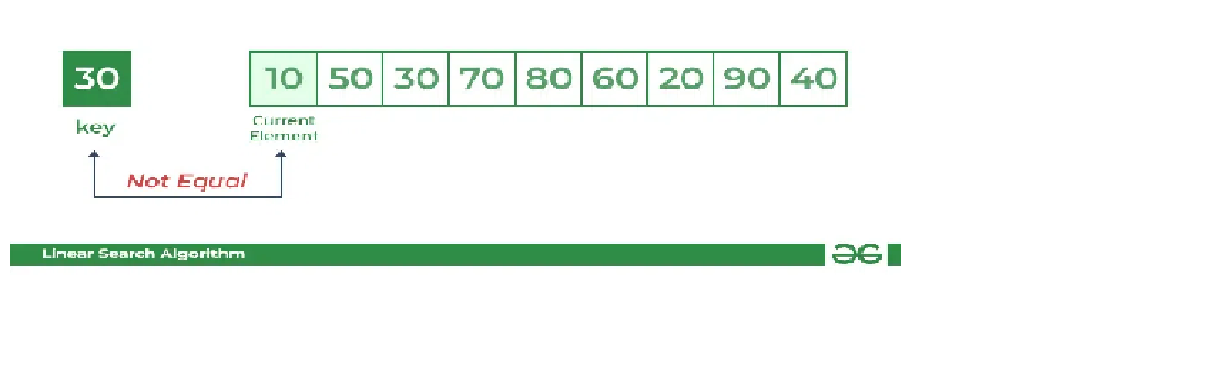
* Every element is considered as a potential match for the key and checked for the same.
* If any element is found equal to the key, the search is successful and the index of that element is returned.
* If no element is found equal to the key, the search yields “No match found”.

**For example:** Consider the array **arr[] = {10, 50, 30, 70, 80, 20, 90, 40}** and **key** = 30

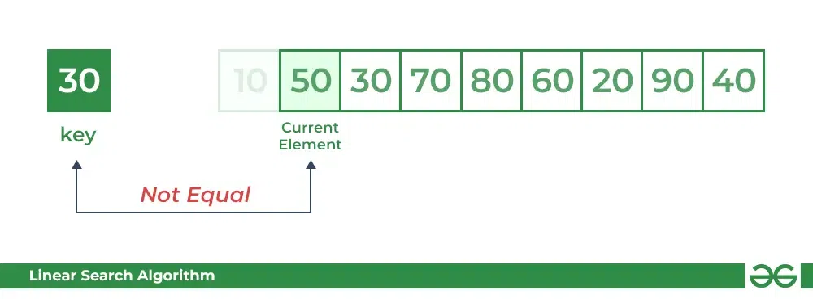
***Step 1:****Start from the first element (index 0) and compare****key****with each element (arr[i]).*

* *Comparing key with first element arr[0]. SInce not equal, the iterator moves to the next element as a potential match.*

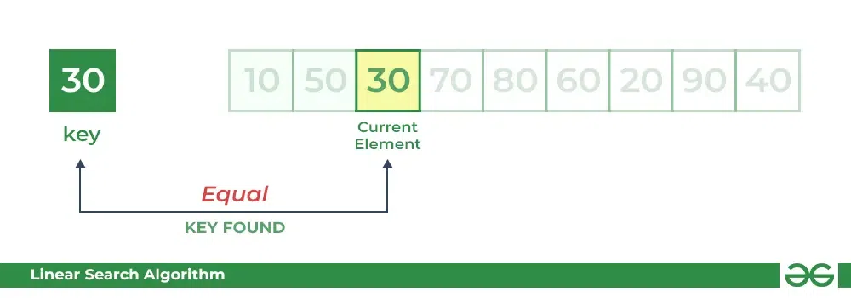
*Compare key with arr[2*



* *Comparing key with next element arr[1]. SInce not equal, the iterator moves to the next element as a potential match.*



***Step 2:****Now when comparing arr[2] with key, the value matches. So the Linear Search Algorithm will yield a successful message and return the index of the element when key is found (here 2).*



### Brute Force Algorithms

* A brute force algorithm solves a problem through exhaustion: it goes through all possible choices until a solution is found.
* The time complexity of a brute force algorithm is often proportional to the input size.
* Brute force algorithms are simple and consistent, but very slow.

**# pseudocode that prints all divisors of n by brute force**

define printDivisors, n

for all numbers from 1 to n

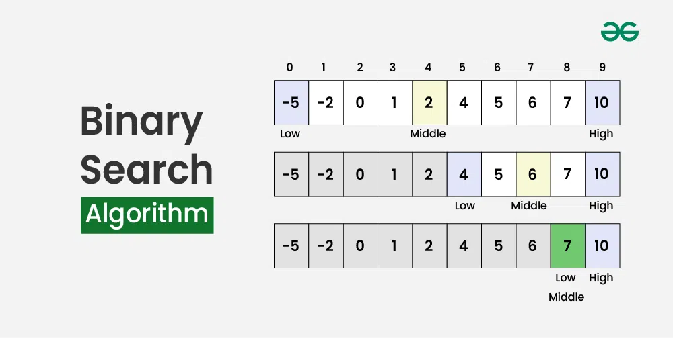
if the number is a divisor of n

print the number

### Linear Search Best and Worst Cases

The best-case performance for the Linear Search algorithm is when the search item appears at the beginning of the list and is O(1). The worst-case performance is when the search item appears at the end of the list or not at all. This would require N comparisons, hence, the worse case is O(N).

**Binary Search** **Algorithm**is a [searching algorithm](https://www.geeksforgeeks.org/searching-algorithms/) used in a sorted array by **repeatedly dividing the search interval in half**. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).



## How does Binary Search Algorithm work?

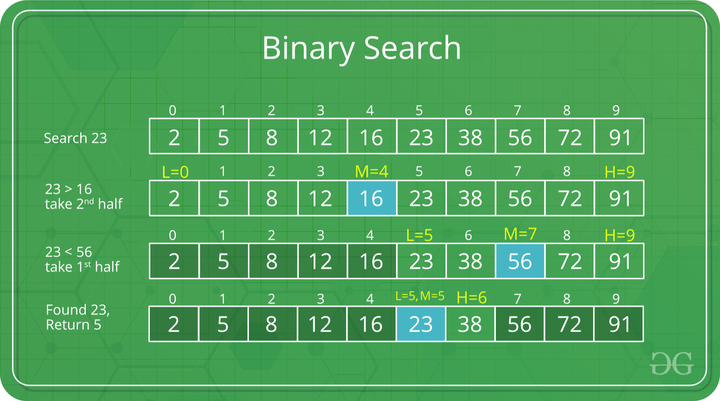
To understand the working of binary search, consider the following illustration:

*Consider an array****arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91}****, and the****target = 23****.*

***First Step:****Calculate the mid and compare the mid element with the key. If the key is less than mid element, move to left and if it is greater than the mid then move search space to the right.*

* *Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.*
* *Key is less than the current mid 56. The search space moves to the left.*

***Second Step:****If the key matches the value of the mid element, the element is found and stop search.*



# Jump Search

Like [Binary Search](https://www.geeksforgeeks.org/binary-search/), Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than [linear search](https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/)) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.  
For example, suppose we have an array arr[] of size n and a block (to be jumped) of size m. Then we search in the indexes arr[0], arr[m], arr[2m]…..arr[km], and so on. Once we find the interval (arr[km] < x < arr[(k+1)m]), we perform a linear search operation from the index km to find the element x.  
Let’s consider the following array: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). The length of the array is 16. The Jump search will find the value of 55 with the following steps assuming that the block size to be jumped is 4.   
STEP 1: Jump from index 0 to index 4;   
STEP 2: Jump from index 4 to index 8;   
STEP 3: Jump from index 8 to index 12;   
STEP 4: Since the element at index 12 is greater than 55, we will jump back a step to come to index 8.   
STEP 5: Perform a linear search from index 8 to get the element 55.

#### Performance in comparison to linear and binary search:

If we compare it with linear and binary search then it comes out then it is better than linear search but not better than binary search.

The increasing order of performance is:

linear search  <  jump search  <  binary search

**What is the optimal block size to be skipped?**   
In the worst case, we have to do n/m jumps, and if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore, the total number of comparisons in the worst case will be ((n/m) + m-1). The value of the function ((n/m) + m-1) will be minimum when m = ?n. Therefore, the best step size is m = **?n.**

#### Algorithm steps

* Jump Search is an algorithm for finding a specific value in a sorted array by jumping through certain steps in the array.
* The steps are determined by the sqrt of the length of the array.
* Here is a step-by-step algorithm for the jump search:
* Determine the step size m by taking the sqrt of the length of the array n.
* Start at the first element of the array and jump m steps until the value at that position is greater than the target value.  
  Once a value greater than the target is found, perform a linear search starting from the previous step until the target is found or it is clear that the target is not in the array.  
  If the target is found, return its index. If not, return -1 to indicate that the target was not found in the array.

