Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

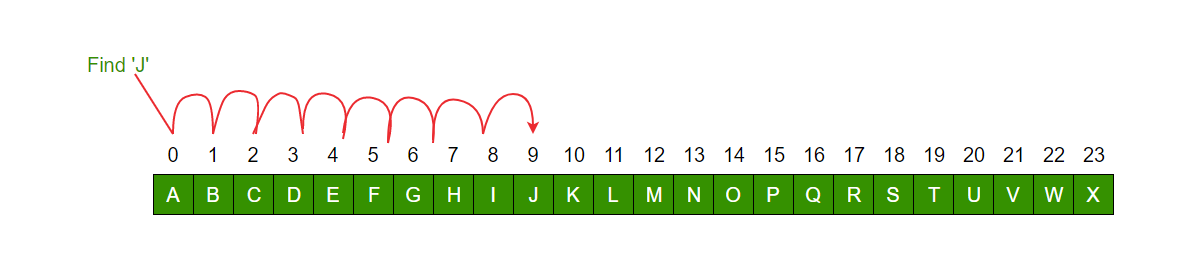
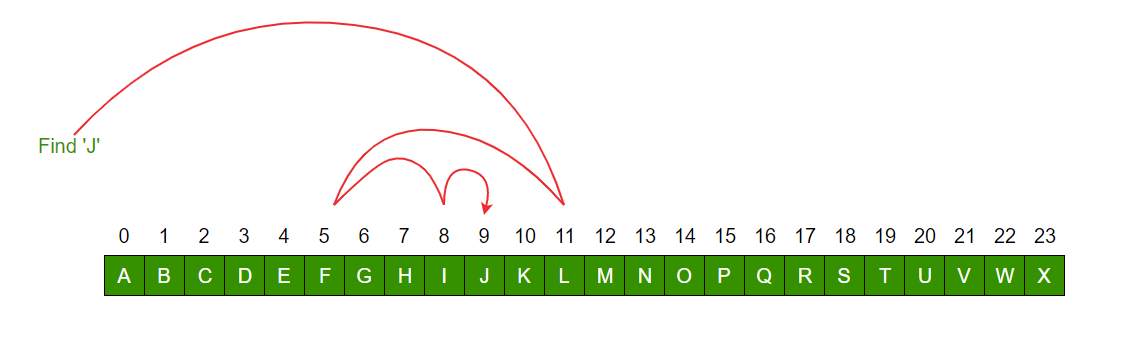
<https://www.geeksforgeeks.org/linear-search/>

<https://www.geeksforgeeks.org/searching-algorithms/?ref=shm>

<https://www.geeksforgeeks.org/introduction-to-arrays/>

1. **Sequential Search**: In this, the list or array is traversed sequentially and every element is checked. For example: [Linear Search](https://www.geeksforgeeks.org/linear-search/).
2. **Interval Search**: These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the center of the search structure and divide the search space in half. For Example: [Binary Search](https://www.geeksforgeeks.org/binary-search/).

|  |  |
| --- | --- |
| **Linear Search** | **Binary Search** |
| In linear search input data need not to be in sorted. | In binary search input data need to be in sorted order. |
| It is also called sequential search. | It is also called half-interval search. |
| The time complexity of linear search **O(n)**. | The time complexity of binary search **O(log n)**. |
| Multidimensional array can be used. | Only single dimensional array is used. |
| Linear search performs equality comparisons | Binary search performs ordering comparisons |
| It is less complex. | It is more complex. |
| It is very slow process. | It is very fast process. |

1. Let us look at an example to compare the two:
2. **Linear Search to find the element “J” in a given sorted list from A-X**
3. [](https://media.geeksforgeeks.org/wp-content/uploads/Linear.png)
4. **Binary Search to find the element “J” in a given sorted list from A-X**
5. [](https://media.geeksforgeeks.org/wp-content/uploads/binary-3.png)
6. **LINER SEARCHING EXAMPLE:**

|  |
| --- |
| /\*package whatever //do not write package name here \*/  import java.io.\*;    class GFG {      public static int liner(int arr[], int x)      {          for (int i = 0; i < arr.length; i++) {              if (arr[i] == x)                  return i;          }          return -1;      }      public static void main(String[] args)      {          int arr[] = { 1, 3, 5, 7, 9, 8 };          int search = liner(              arr,              10); // Here we are searching for 10 element in                   // the array which is not present in the                   // array so, it will print -1          System.out.println(search);      }  } |

1. **BINARY SEARCHING EXAMPLE:**

|  |
| --- |
| /\*package whatever //do not write package name here \*/    public class GFG {      public static int binary(int arr[], int x)      {          int start = 0;          int end = arr.length - 1;          while (start <= end) {              int mid = (start + end) / 2;              if (x == arr[mid]) {                  return mid;              }              else if (x > arr[mid]) {                  start = mid + 1;              }              else {                  end = mid - 1;              }          }          return -1;      }      public static void main(String[] args)      {          int arr[] = { 2, 4, 5, 17, 14, 7, 11, 22 };          int search = binary(arr, 22);          System.out.println(search);      }  } |

**Linear Search Algorithm**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 23 Sep, 2022

 Read

 Discuss

Linear Search is defined as a sequential search algorithm 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. It is the easiest searching algorithm



Given an array **arr[]** of **N** elements, the task is to write a function to search a given element **x** in **arr[]**.

**Examples:**

**Input:** arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170}, x = 110;  
**Output:** 6  
**Explanation:** Element x is present at index 6

**Input:** arr[] = {10, 20, 80, 30, 60, 50,110, 100, 130, 170}, x = 175;  
**Output:** -1  
**Explanation:** Element x is not present in arr[].

Recommended Problem

Search an Element in an array

[Arrays](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Arrays&sortBy=submissions)

[Searching](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Searching&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/search-an-element-in-an-array-1587115621/1" \o "Permalink to Search an Element in an array)

Submission count: 1.1L

Follow the below idea to solve the problem:

Iterate from 0 to N-1and compare the value of every index with x if they match return index

 Follow the given steps to solve the problem:

* Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
* If x matches with an element, return the index.
* If x doesn’t match with any of the elements, return -1.

Below is the implementation of the above approach:

|  |
| --- |
| // Java code for linearly searching x in arr[]. If x  // is present then return its location, otherwise  // return -1    class GFG {      public static int search(int arr[], int x)      {          int N = arr.length;          for (int i = 0; i < N; i++) {              if (arr[i] == x)                  return i;          }          return -1;      }        // Driver's code      public static void main(String args[])      {          int arr[] = { 2, 3, 4, 10, 40 };          int x = 10;            // Function call          int result = search(arr, x);          if (result == -1)              System.out.print(                  "Element is not present in array");          else              System.out.print("Element is present at index "                               + result);      }  } |

**Output**

Element is present at index 3

**Time complexity:** O(N)  
**Auxiliary Space:** O(1)

**Linear Search Recursive Approach:**

 Follow the given steps to solve the problem:

* If the size of the array is zero then, return -1, representing that the element is not found. This can also be treated as the base condition of a recursion call.
* Otherwise, check if the element at the current index in the array is equal to the key or not i.e,arr[size – 1] == key
  + If equal, then return the index of the found key.

Below is the implementation of the above approach:

|  |
| --- |
| // Java Recursive Code For Linear Search  import java.io.\*;    class Test {      static int arr[] = { 5, 15, 6, 9, 4 };        // Recursive Method to search key in the array      static int linearsearch(int arr[], int size, int key)      {          if (size == 0) {              return -1;          }          else if (arr[size - 1] == key) {              // Return the index of found key.              return size - 1;          }          else {              return linearsearch(arr, size - 1, key);          }      }        // Driver method      public static void main(String[] args)      {          int key = 4;            // Function call to find key          int index = linearsearch(arr, arr.length, key);          if (index != -1)              System.out.println(                  "The element " + key + " is found at "                  + index + " index of the given array.");            else              System.out.println("The element " + key                                 + " is not found.");      }  }    // This Code is submitted by Susobhan Akhuli |

**Output**

The element 4 is found at 4 index of the given array.

**Time Complexity:** O(N)  
**Auxiliary Space:** O(N), for using recursive stack space.

**Binary Search**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 13 Oct, 2022

 Read

 Discuss

**Problem:** Given a sorted array **arr[]** of **n** elements, write a function to search a given element **x** in **arr[]** and return the index of x in the array.

                 Consider array is 0 base index.

**Examples:**

**Input:** arr[] = {10, 20, 30, 50, 60, 80, 110, 130, 140, 170}, x = 110  
**Output:** 6  
**Explanation:** Element x is present at index 6.

**Input:** arr[] = {10, 20, 30, 40, 60, 110, 120, 130, 170}, x = 175  
**Output:** -1  
**Explanation:** Element x is not present in arr[].

**Linear Search Approach**: A simple approach is to do a [**linear search**](https://www.geeksforgeeks.org/linear-search/)**.** The time complexity of the Linear search is O(n). Another approach to perform the same task is using *Binary Search*.

**Binary Search Approach:**

**Binary Search** 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).

**Binary Search Algorithm:** The basic steps to perform Binary Search are:

* Begin with the mid element of the whole array as a search key.
* If the value of the search key is equal to the item then return an index of the search key.
* Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
* Otherwise, narrow it to the upper half.
* Repeatedly check from the second point until the value is found or the interval is empty.

Binary Search Algorithm can be implemented in the following two ways

1. Iterative Method
2. Recursive Method

1. Iteration Method

binarySearch(arr, x, low, high)

repeat till low = high

mid = (low + high)/2

if (x == arr[mid])

return mid

else if (x > arr[mid]) // x is on the right side

low = mid + 1

else // x is on the left side

high = mid - 1

2. Recursive Method (The recursive method follows the divide and conquer approach)

binarySearch(arr, x, low, high)

if low > high

return False

else

mid = (low + high) / 2

if x == arr[mid]

return mid

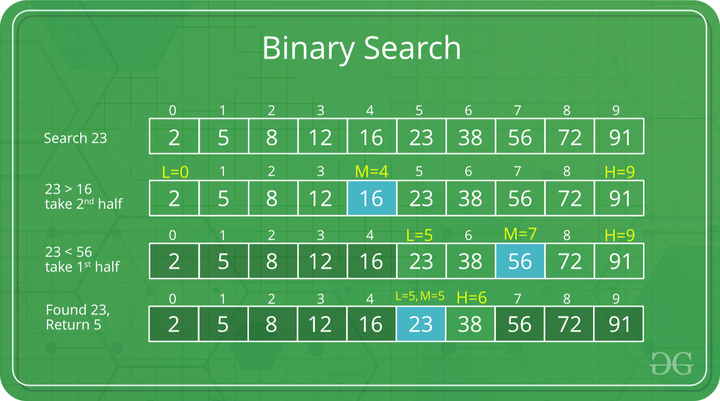
else if x > arr[mid] // x is on the right side

return binarySearch(arr, x, mid + 1, high)

else // x is on the left side

return binarySearch(arr, x, low, mid - 1)

**Illustration of Binary Search Algorithm:**



Example of Binary Search Algorithm

Recommended Problem

Searching an element in a sorted array

[Searching](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Searching&sortBy=submissions)

[Binary Search](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Binary%20Search&sortBy=submissions)

[Paytm](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Paytm&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/who-will-win-1587115621/1" \o "Permalink to Searching an element in a sorted array)

Submission count: 77.5K

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Step-by-step Binary Search Algorithm:** We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we recur for the right half.
4. Else (x is smaller) recur for the left half.

**Recursive implementation of Binary Search**:

|  |
| --- |
| // Java implementation of recursive Binary Search  class BinarySearch {      // Returns index of x if it is present in arr[l..      // r], else return -1      int binarySearch(int arr[], int l, int r, int x)      {          if (r >= l) {              int mid = l + (r - l) / 2;                // If the element is present at the              // middle itself              if (arr[mid] == x)                  return mid;                // If element is smaller than mid, then              // it can only be present in left subarray              if (arr[mid] > x)                  return binarySearch(arr, l, mid - 1, x);                // Else the element can only be present              // in right subarray              return binarySearch(arr, mid + 1, r, x);          }            // We reach here when element is not present          // in array          return -1;      }        // Driver method to test above      public static void main(String args[])      {          BinarySearch ob = new BinarySearch();          int arr[] = { 2, 3, 4, 10, 40 };          int n = arr.length;          int x = 10;          int result = ob.binarySearch(arr, 0, n - 1, x);          if (result == -1)              System.out.println("Element not present");          else              System.out.println("Element found at index "                                 + result);      }  }  /\* This code is contributed by Rajat Mishra \*/ |

**Output**

Element is present at index 3

**Time Complexity:** O(log n)