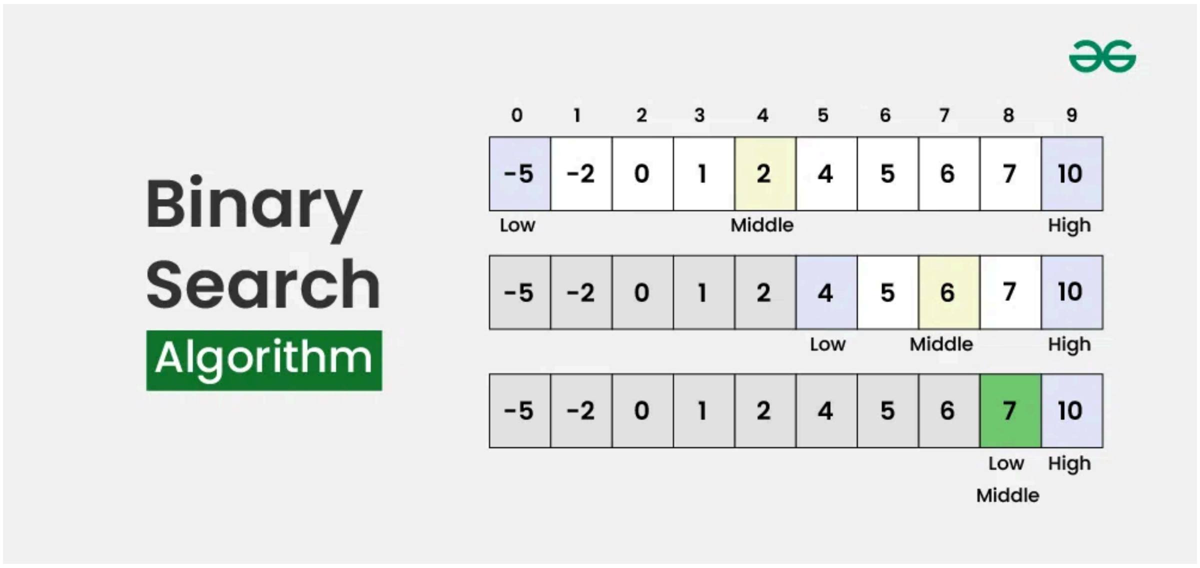
Binary Search Algorithm – Iterative and Recursive Implementation

Last Updated : 04 Sep, 2024

Binary Search Algorithm is a searching algorithm 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

Table of Content

What is Binary Search Algorithm?

Binary search is a search algorithm used to find the position of a target value within a **sorted** array. It works by repeatedly dividing the search interval in half until the target value is found or the interval is empty. The search interval is halved by comparing the target element with the middle value of the search space.

Conditions to apply Binary Search Algorithm in a Data Structure

To apply Binary Search algorithm:

- The data structure must be sorted.
- Access to any element of the data structure should take constant time.

Binary Search Algorithm

Below is the step-by-step algorithm for Binary Search:

- Divide the search space into two halves by **finding the middle index "mid"**.
- Compare the middle element of the search space with the **key**.
- If the **key** is found at middle element, the process is terminated.
- If the **key** is not found at middle element, choose which half will be used as the next search space.
    - If the **key** is smaller than the middle element, then the **left** side is used for next search.
    - If the **key** is larger than the middle element, then the **right** side is used for next search.
- This process is continued until the **key** is found or the total search space is exhausted.

Binary Search Visualizer

Visualization of Binary Search

2 5 8 12 16 23 38 56 72 91

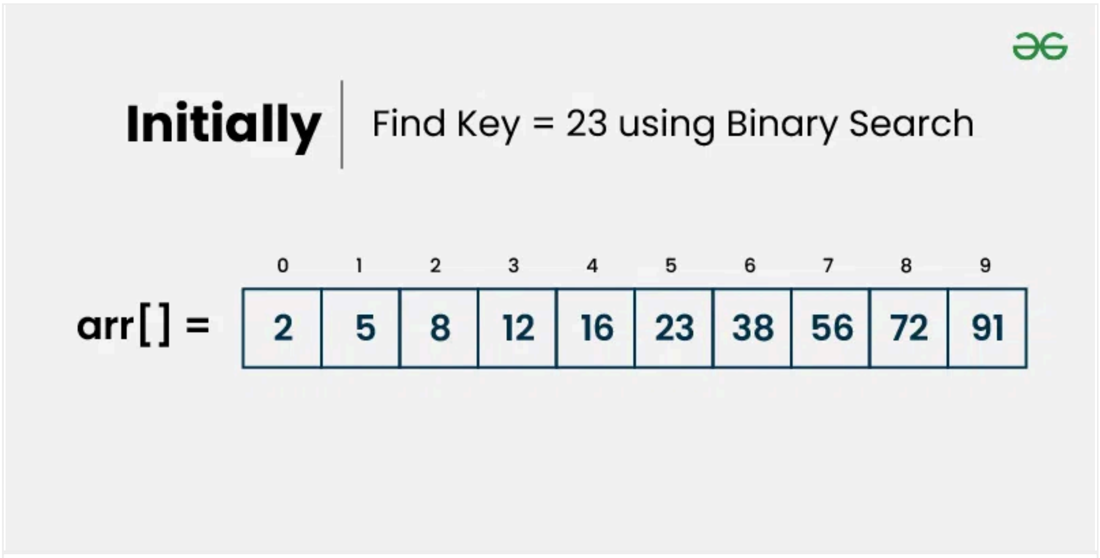
Enter a number to sear Start Search

Iterations: 0

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**.



1 / 4

How to Implement Binary Search Algorithm?

The **Binary Search Algorithm** can be implemented in the following two ways

- Iterative Binary Search Algorithm
- Recursive Binary Search Algorithm

Given below are the pseudocodes for the approaches.

Iterative Binary Search Algorithm:

Here we use a while loop to continue the process of comparing the key and splitting the search space in two halves.

C++ C Java Python C# JavaScript PHP

```python
# Python3 code to implement iterative Binary
# Search.


# It returns location of x in given array arr
def binarySearch(arr, low, high, x):

    while low <= high:

        mid = low + (high - low) // 2

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            low = mid + 1

        # If x is smaller, ignore right half
        else:
            high = mid - 1

    # If we reach here, then the element
    # was not present
    return -1


# Driver Code
if __name__ == '__main__':
    arr = [2, 3, 4, 10, 40]
    x = 10

    # Function call
    result = binarySearch(arr, 0, len(arr)-1, x)
    if result != -1:
        print("Element is present at index", result)
    else:
        print("Element is not present in array")
```

Output

```
Element is present at index 3
```

Time Complexity: O(log N)
Auxiliary Space: O(1)

Recursive Binary Search Algorithm:

> *Create a recursive function and compare the mid of the search space with the key. And based on the result either return the index where the key is found or call the recursive function for the next search space.*

C++ C Java Python C# JavaScript PHP

```
# Python3 Program for recursive binary search.


# Returns index of x in arr if present, else -1
def binarySearch(arr, low, high, x):

    # Check base case
    if high >= low:

        mid = low + (high - low) // 2

        # If 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
        elif arr[mid] > x:
            return binarySearch(arr, low, mid-1, x)

        # Else the element can only be present
        # in right subarray
        else:
            return binarySearch(arr, mid + 1, high, x)

    # Element is not present in the array
    else:
        return -1


# Driver Code
if __name__ == '__main__':
    arr = [2, 3, 4, 10, 40]
    x = 10

    # Function call
    result = binarySearch(arr, 0, len(arr)-1, x)

    if result != -1:
        print("Element is present at index", result)
    else:
        print("Element is not present in array")
```

Output

```
Element is present at index 3
```

Complexity Analysis of Binary Search Algorithm

- **Time Complexity:**
    - Best Case: O(1)
    - Average Case: O(log N)
    - Worst Case: O(log N)
- **Auxiliary Space:** O(1), If the recursive call stack is considered then the auxiliary space will be O(logN).

Applications of Binary Search Algorithm

- Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model.
- It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping.
- It can be used for searching a database.

Advantages of Binary Search

- Binary search is faster than linear search, especially for large arrays.
- More efficient than other searching algorithms with a similar time complexity, such as interpolation search or exponential search.
- Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.

Disadvantages of Binary Search

- The array should be sorted.
- Binary search requires that the data structure being searched be stored in contiguous memory locations.
- Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered.

Frequently Asked Questions(FAQs) on Binary Search

1. What is Binary Search?

Binary search is an efficient algorithm for finding a target value within a sorted array. It works by repeatedly dividing the search interval in half.

2. How does Binary Search work?

Binary Search compares the target value to the middle element of the array. If they are equal, the search is successful. If the target is less than the middle element, the search continues in the lower half of the array. If *the target is greater, the search continues in the upper half. This process repeats until the target is found or the search interval is empty.*

3. What is the time complexity of Binary Search?

The time complexity of binary search is $O(\log_2 n)$, where n is the number of elements in the array. This is because the size of the search interval is halved in each step.

4. What are the prerequisites for Binary Search?

Binary search requires that the array is sorted in ascending or descending order. If the array is not sorted, we cannot use Binary Search to search an element in the array.

5. What happens if the array is not sorted for binary search?

If the array is not sorted, binary search may return incorrect results. It relies on the sorted nature of the array to make decisions about which half of the array to search.

6. Can binary search be applied to non-numeric data?

Yes, binary search can be applied to non-numeric data as long as there is a defined order for the elements. For example, it can be used to search for strings in alphabetical order.

7. What are some common disadvantages of Binary Search?

The disadvantage of Binary Search is that the input array needs to be sorted to decide which in which half the target element can lie. Therefore for unsorted arrays, we need to sort the array before applying Binary Search.

8. When should Binary Search be used?

> *Binary search should be used when searching for a target value in a sorted array, especially when the size of the array is large. It is particularly efficient for large datasets compared to linear search algorithms.*

9. Can binary search be implemented recursively?

> *Yes, binary search can be implemented both iteratively and recursively. The recursive implementation often leads to more concise code but may have slightly higher overhead due to recursive stack space or function calls.*

10. Is Binary Search always the best choice for searching in a sorted array?

> *While binary search is very efficient for searching in sorted arrays, there may be specific cases where other search algorithms are more appropriate, such as when dealing with small datasets or when the array is frequently modified.*

Related Articles:

- Binary Search on Answer Tutorial with Problems
- Linear Search vs Binary Search
- How to Identify & Solve Binary Search Problems?

GeeksforGeeks 1.11k

Previous Article

Linear Search Algorithm

Next Article

Sentinel Linear Search

Similar Reads

Iterative Deepening Search(IDS) or Iterative Deepening Depth First...

There are two common ways to traverse a graph, BFS and DFS. Considering a Tree (or Graph) of huge height and width, both BFS and DFS are not very efficie...

10 min read

Search an element in a Linked List (Iterative and Recursive)

Given a linked list and a key, the task is to check if key is present in the linked list or not. Examples: Input: 14 -> 21 -> 11 -> 30 -> 10, key = 14Output:...

13 min read

Count half nodes in a Binary tree (Iterative and Recursive)

Given A binary Tree, how do you count all the half nodes (which has only one child) without using recursion? Note leaves should not be touched as they have...

12 min read

Count full nodes in a Binary tree (Iterative and Recursive)

Given A binary Tree, how do you count all the full nodes (Nodes which have both children as not NULL) without using recursion and with recursion? Note leaves...

12 min read

Merge Two Binary Trees by doing Node Sum (Recursive and Iterative)

Given two binary trees. We need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new...

15+ min read

Count consonants in a string (Iterative and recursive methods)

Given a string, count total number of consonants in it. A consonant is an English alphabet character that is not vowel (a, e, i, o and u). Examples of constants are ...

7 min read

First uppercase letter in a string (Iterative and Recursive)

Given a string find its first uppercase letterExamples : Input : geeksforgeeKs Output : K Input : geekS Output : S Method 1: linear search Using linear search,...

6 min read

Function to copy string (Iterative and Recursive)

Given two strings, copy one string to another using recursion. We basically need to write our own recursive version of strcpy in C/C++ Examples: Input : s1 =...

6 min read

Program for average of an array (Iterative and Recursive)

Given an array, the task is to find average of that array. Average is the sum of array elements divided by the number of elements. Examples : Input : arr[] = {1, ...

7 min read

Program to count vowels in a string (Iterative and Recursive)

Given a string, count the total number of vowels (a, e, i, o, u) in it. There are two methods to count total number of vowels in a string. Iterative Recursive...

7 min read

Article Tags : Divide and Conquer DSA Searching Accenture +8 More

Practice Tags : Accenture Infosys Oracle Qualcomm +6 More

Software Engineering
Digital Logic Design
Engineering Maths
Software Development
Software Testing

Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

Inteview Preparation
Competitive Programming
Top DS or Algo for CP
Company-Wise Recruitment Process
Company-Wise Preparation
Aptitude Preparation
Puzzles

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos
DSA
Python
Java
C++
Web Development
Data Science
CS Subjects