Ternary Search

Last Updated : 29 Apr, 2024

Computer systems use different methods to find specific data. There are various search algorithms, each better suited for certain situations. For instance, a **binary search** divides information into **two parts**, while a **ternary search** does the same but into **three equal parts**. It's worth noting that ternary search is only effective for sorted data. In this article, we're going to uncover the secrets of **Ternary Search** – how it works, why it's faster in some situations.

DSA Practice Searching Algorithms MCQs on Searching Algorithms Tutorial on Searching Algorithms Linear Sea

Table of Content

- What is the Ternary Search?
- When to use Ternary Search
- Working of Ternary Search
- Implementation of Ternary Search
- Complexity Analysis of Ternary Search
- Binary search Vs Ternary Search
- Advantages
- Disadvantages
- Summary

What is the Ternary Search?

Ternary search is a search algorithm that is used to find the position of a target value within a sorted array. It operates on the principle of dividing the array into **three parts** instead of two, as in binary search. The basic idea is to narrow down the search space by comparing the target value with elements at two points that divide the array into **three equal parts**.

- **mid1** = l + (r-l)/3
- **mid2** = r – (r-l)/3

When to use Ternary Search:

- When you have a large ordered array or list and need to find the position of a specific value.
- When you need to find the maximum or minimum value of a function.
- When you need to find bitonic point in a bitonic sequence.
- When you have to evaluate a quadratic expression

Working of Ternary Search:

The concept involves dividing the array into **three equal segments** and determining in which segment the **key** element is located. It works similarly to a binary search, with the distinction of reducing time complexity by dividing the array into three parts instead of two.

Below are the step-by-step explanation of working of Ternary Search:

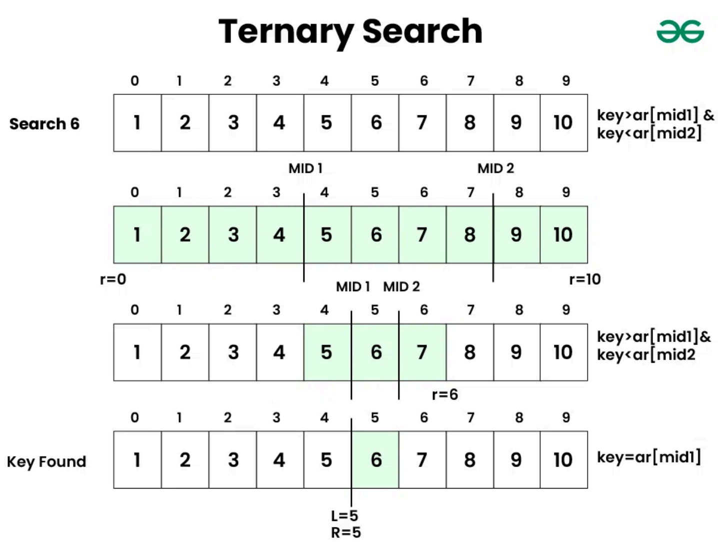
1. **Initialization:**
   - Set two pointers, **left** and **right**, initially pointing to the first and last elements of our search space.

2. **Divide the search space:**
   - Calculate two midpoints, **mid1** and **mid2**, dividing the current search space into three roughly equal parts:
   - mid1 = left + (right – left) / 3
   - mid2 = right – (right – left) / 3
   - The array is now effectively divided into **[left, mid1]**, **(mid1, mid2**), and **[mid2, right]**.

3. **Comparison with Target:**.
    - If the **target** is equal to the element at **mid1** or **mid2**, the search is successful, and the index is returned
    - If the **target** is less than the element at **mid1**, update the **right** pointer to **mid1 – 1**.
    - If the **target** is greater than the element at **mid2**, update the **left** pointer to **mid2 + 1**.
    - If the **target** is between the elements at **mid1** and **mid2**, update the **left** pointer to **mid1 + 1** and the **right** pointer to **mid2** – 1.

4. **Repeat or Conclude:**
    - Repeat the process with the reduced search space until the **target** is found or the search space becomes empty.
    - If the search space is empty and the **target** is not found, return a value indicating that the **target** is not present in the array.

Illustration:



Ternary Search

Implementation:

Below is the implementation of Ternary Search Approach:

C++ C Java Python3 C# JavaScript PHP

```
# Python3 program to illustrate
# recursive approach to ternary search
```

```python
import math as mt

# Function to perform Ternary Search
def ternarySearch(l, r, key, ar):

    if (r >= l):

        # Find the mid1 and mid2
        mid1 = l + (r - l) //3
        mid2 = r - (r - l) //3

        # Check if key is present at any mid
        if (ar[mid1] == key):
            return mid1

        if (ar[mid2] == key):
            return mid2

        # Since key is not present at mid,
        # check in which region it is present
        # then repeat the Search operation
        # in that region
        if (key < ar[mid1]):

            # The key lies in between l and mid1
            return ternarySearch(l, mid1 - 1, key, ar)

        elif (key > ar[mid2]):

            # The key lies in between mid2 and r
            return ternarySearch(mid2 + 1, r, key, ar)

        else:

            # The key lies in between mid1 and mid2
            return ternarySearch(mid1 + 1,
                                 mid2 - 1, key, ar)

    # Key not found
    return -1

# Driver code
l, r, p = 0, 9, 5
```

```
# Get the array
# Sort the array if not sorted
ar = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

# Starting index
l = 0

# end element index
r = 9

# Checking for 5

# Key to be searched in the array
key = 5

# Search the key using ternarySearch
p = ternarySearch(l, r, key, ar)

# Print the result
print("Index of", key, "is", p)

# Checking for 50

# Key to be searched in the array
key = 50

# Search the key using ternarySearch
p = ternarySearch(l, r, key, ar)

# Print the result
print("Index of", key, "is", p)

# This code is contributed by
# Mohit kumar 29
```

Output

```
Index of 5 is 4
Index of 50 is -1
```

Time Complexity: $O(2 * \log_3 n)$

Auxiliary Space: $O(\log_3 n)$

Iterative Approach of Ternary Search:

C Java Python3 C# JavaScript PHP C++

```
# Python 3 program to illustrate iterative
# approach to ternary search

# Function to perform Ternary Search
def ternarySearch(l, r, key, ar):
    while r >= l:

        # Find mid1 and mid2
        mid1 = l + (r-l) // 3
        mid2 = r - (r-l) // 3

        # Check if key is at any mid
        if key == ar[mid1]:
            return mid1
        if key == ar[mid2]:
            return mid2

        # Since key is not present at mid,
        # Check in which region it is present
        # Then repeat the search operation in that region
        if key < ar[mid1]:
            # key lies between l and mid1
            r = mid1 - 1
        elif key > ar[mid2]:
            # key lies between mid2 and r
            l = mid2 + 1
        else:
            # key lies between mid1 and mid2
            l = mid1 + 1
            r = mid2 - 1

    # key not found
    return -1

# Driver code
```

```python
# Get the list
# Sort the list if not sorted
ar = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Starting index
l = 0

# end element index
r = 9

# Checking for 5
# Key to be searched in the list
key = 5

# Search the key using ternary search
p = ternarySearch(l, r, key, ar)

# Print the result
print("Index of", key, "is", p)

# Checking for 50
# Key to be searched in the list
key = 50

# Search the key using ternary search
p = ternarySearch(l, r, key, ar)

# Print the result
print("Index of", key, "is", p)

# This code has been contributed by Sujal Motagi
```

Output

```
Index of 5 is 4
Index of 50 is -1
```

Time Complexity: $O(2 * \log_3 n)$, where n is the size of the array.

Auxiliary Space: O(1)

Complexity Analysis of Ternary Search:

Time Complexity:

- Worst case: $O(\log_3 N)$
- Average case: $\Theta(\log_3 N)$
- Best case: $\Omega(1)$

Auxiliary Space: $O(1)$

Binary search Vs Ternary Search:

The time complexity of the binary search is less than the ternary search as the number of comparisons in ternary search is much more than binary search. Binary Search is used to find the maxima/minima of monotonic functions where as Ternary Search is used to find the maxima/minima of unimodal functions.

Note: We can also use ternary search for monotonic functions but the time complexity will be slightly higher as compared to binary search.

Advantages:

- Ternary search can find maxima/minima for unimodal functions, where binary search is not applicable.
- Ternary Search has a time complexity of $O(2 * \log_3 n)$, which is more efficient than linear search and comparable to binary search.
- Fits well with optimization problems.

Disadvantages:

- Ternary Search is only applicable to ordered lists or arrays, and cannot be used on unordered or non-linear data sets.
- Ternary Search takes more time to find maxima/minima of monotonic functions as compared to Binary Search.

Summary:

- Ternary Search is a divide-and-conquer algorithm that is used to find the position of a specific value in a given array or list.
- It works by dividing the array into three parts and recursively performing the search operation on the appropriate part until the desired element is found.

- The algorithm has a time complexity of O(2 * $\log_3 n$) and is more efficient than a linear search, but less commonly used than other search algorithms like binary search.
- It's important to note that the array to be searched must be sorted for Ternary Search to work correctly.

Previous Article

Meta Binary Search | One-Sided Binary Search

Next Article

Jump Search

Similar Reads

Binary Search Tree vs Ternary Search Tree

For effective searching, insertion, and deletion of items, two types of search trees are used: the binary search tree (BST) and the ternary search tree (TST)....

3 min read

Why is Binary Search preferred over Ternary Search?

The following is a simple recursive Binary Search function in C++ taken from here. C/C++ Code // CPP program for the above approach #include <bits/stdc++.h>...

11 min read

Is ternary search faster than binary search?

Binary search is a widely used algorithm for searching a sorted array. It works by repeatedly dividing the search space in half until the target element is found....

3 min read

Ternary Search Visualization using JavaScript

GUI(Graphical User Interface) helps better in understanding than programs. In this article, we will visualize Ternary Search using JavaScript. We will see how...

4 min read

Ternary Search Tree meaning & definition in DSA

Ternary Search Tree is defined as a special trie data structure where the child nodes of a standard trie are ordered as a binary search tree where each node ca...

3 min read

Ternary search meaning in DSA

Ternary search is a searching algorithm which is based on divide and conquer principle that works by dividing the search interval into three parts as in binary...

2 min read

Ternary Search Tree

A ternary search tree is a special trie data structure where the child nodes of a standard trie are ordered as a binary search tree. Representation of ternary sear...

13 min read

Ternary Search Tree (Deletion)

In the SET 1 post on TST we have described how to insert and search a node in TST. In this article, we will discuss algorithm on how to delete a node from TST....

14 min read

Longest word in ternary search tree

Given a set of words represented in a ternary search tree, find the length of largest word among them. Examples: Input : {"Prakriti", "Raghav", "Rashi",...

11 min read

Ternary Search for Competitive Programming

Ternary search is a powerful algorithmic technique that plays a crucial role in competitive programming. This article explores the fundamentals of ternary...

8 min read

Article Tags : Algorithms DSA Searching Sorting

Practice Tags : Algorithms Searching Sorting

System Design Bootcamp
Interview Questions

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