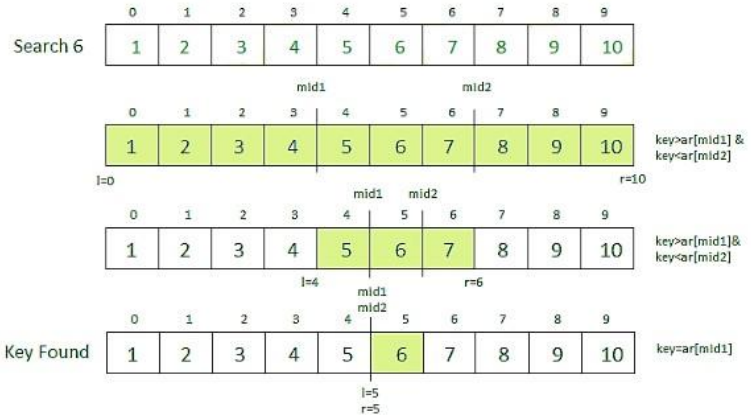
Ternary Search

Ternary Search is a Divide and Conquer Algorithm used to perform search operation in a **sorted array**. This algorithm is similar to the Binary Search algorithm but rather than dividing the array into two parts, it divides the array into three equal parts.

In this algorithm, the given array is divided into three parts and the key (element to be searched) is compared to find the part in which it lies and that part is further divided into three parts.

We can divide the array into three parts by taking mid1 and mid2 which can be calculated as shown below. Initially, l and r will be equal to 0 and N-1 respectively, where N is the length of the array.

```
mid1 = l + (r-l)/3
mid2 = r - (r-l)/3
```



Note: The array must be sorted in order to perform the Binary Search or Ternary Search operation.

Steps to perform Ternary Search:

- First, we compare the key with the element at mid1. If found equal, we return mid1.
- If not, then we compare the key with the element at mid2. If found equal, we return mid2.
- If not, then we check whether the key is less than the element at mid1. If yes, then recur to the first part.
- If not, then we check whether the key is greater than the element at mid2. If yes, then recur to the third part.
- If not, then we recur to the second (middle) part.

Implementation: The Ternary Search Algorithm can be implemented in both recursive and iterative manner. Below is the implementation of both recursive and iterative function to perform Ternary Search on an array *arr[]* of size *N* to search an element *key*.

- **Recursive Function**:

```
// Recursive Function to perform Ternary Search , Initially,l = 0, starting index of array , r = N-1, ending index of array.
int ternarySearch(int l, int r, int key, int ar[])
{
    if (r >= l)
    {
        // Find mid1 and mid2
        int mid1 = l + (r - l) / 3;
        int 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);
        }
        else if (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; }
```

- Iterative Function:

```
// Iterative Function to perform Ternary Search Initially, l = 0, starting index of array, r = N-1,
//ending index of array.
int ternarySearch(int l, int r, int key, int ar[])
{
    while (r >= l) { // Find mid1 and mid2
        int mid1 = l + (r - l) / 3;
        int 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
            r = mid1 - 1;
        }
        else if (key > ar[mid2]) {
            // The key lies in between mid2 and r
            l = mid2 + 1;
        }
        else {
            // The key lies in between mid1 and mid2
            l = mid1 + 1;
            r = mid2 - 1;
        }
    }// Key not found
    return -1; }
```

Time Complexity: $O(Log_3N)$, where N is the number of elements in the array.