Ternary Search

Like the binary search, it also separates the lists into sub-lists. This procedure divides the list into three parts using two intermediate mid values. As the lists are divided into more subdivisions, so it reduces the time to search a key value.

Ternary search is a divide-and-conquer search algorithm. It is mandatory for the array (in which you will search for an element) to be sorted before we begin the search. In this search, after each iteration it neglects **2/3** part of the array and repeats the same operations on the remaining **1/3**.

So, it can be visualized as follows: every time after evaluating the function at two mid-points ***mid1*** and ***mid2***, we are essentially ignoring about one third of the interval, either the left or right one. Thus the size of the search space is 2*n*/3 of the original one.

Algorithm

The steps involved in this algorithm are:
(The list must be in sorted order)

- **Step 1**: Divide the search space (initially, the list) in three parts (with two mid-points: `mid1` and `mid2`)
- **Step 2**: The target element is compared with the edge elements that is elements at location `mid1`, `mid2` and the end of the search space. If element matches, go to step 3 else predict in which section the target element lies. The search space is reduced to `1/3`rd. If the element is not in the list, go to step 4 or to step 1.
- **Step 3**: Element found. Return index and exit.
- **Step 4**: Element not found. Exit.

Complexity

- Worst case time complexity: $O(\log_3 n)$
- Space complexity: `O(1)`

Pseudo Code:

```
ternarySearch(array, start, end, key)
```

Input: An sorted array, start and end location, and the search key

Output: location of the key (if found), otherwise wrong location.

```
Begin
   if start <= end then
      midFirst := start + (end - start) /3
      midSecond := midFirst + (end - start) / 3
      if array[midFirst] = key then
         return midFirst
      if array[midSecond] = key then
         return midSecond
      if key < array[midFirst] then
         call ternarySearch(array, start, midFirst-1, key)
      if key > array[midSecond] then
         call ternarySearch(array, midSecond+1, end, key)
      else
         call ternarySearch(array, midFirst+1, midSecond-1, key)
   else
      return invalid location
End
```

Implementation:

Source Code {Using Recursion}

```
#include<stdio.h>

int ternarySearch(int array[], int start, int end, int key)
{
        if(start <= end)
        {
                int midFirst = (start + (end - start) /3);
                //mid of first and second block
                int midSecond = (midFirst + (end - start) /3);
                //mid of first and second block
                if(array[midFirst] == key)
                        return midFirst;
                if(array[midSecond] == key)
                        return midSecond;
                if(key < array[midFirst])
                        return ternarySearch(array, start, midFirst-1, key);
                if(key > array[midSecond])
                        return ternarySearch(array, midSecond+1, end, key);
                return ternarySearch(array, midFirst+1, midSecond-1, key);
        }
return -1;
}

int main()
{
int n, searchKey, loc;
printf("Enter number of items: ");
scanf("%d", &n);
int arr[n]; //create an array of size n
printf("Enter items: \n");

for(int i = 0; i< n; i++)
{
        scanf("%d", &arr[i]);
}

printf("Enter search key to search in the list: ");
scanf("%d", &searchKey);
if((loc = ternarySearch(arr, 0, n, searchKey)) >= 0)
        printf("Item found at location: %d \n", loc);
else
        printf("Item is not found in the list.\n");
}
```

Output

```
Enter number of items: 8
Enter items:
12 25 48 52 67 79 88 93
Enter search key to search in the list: 52
Item found at location: 3
```

Source Code (C) {Using Loop}

```
#include <stdio.h>
int ternarySearch(int array[], int left, int right, int x)
{
   if (right >= left) {
     int intvl = (right - left) / 3;
     int leftmid = left + intvl;
     int rightmid = leftmid + intvl;
     if (array[leftmid] == x)
        return leftmid;
     if (array[rightmid] == x)
        return rightmid;
     if (x < array[leftmid]) {
       return ternarySearch(array, left, leftmid, x);
     }
     else if (x > array[leftmid] && x < array[rightmid]) {
       return ternarySearch(array, leftmid, rightmid, x);
     }
     else {
       return ternarySearch(array, rightmid, right, x);
     }
   }
   return -1;
}
int main()
{
   int array[] = {1, 2, 3, 5};
   int size = sizeof(array)/ sizeof(array[0]);
   int find = 3;
   printf("Position of %d is %d\n", find, ternarySearch(array, 0, size-1,
find));
   return 0;
}
```

Applications

- This concept is used in *unimodal functions to determine the maximum or minimum value* of that function. Unimodal functions are functions that, have a single highest value.
- Can be used to search for where the derivative is zero in Newton's method as an optimization.