Exponential search algorithm

***Source:* https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms**

1. *What is an exponential search algorithm?*

Exponential search is an algorithm used to search for a target element within a sorted array or list. It works by first checking the middle element of the array, and then checking the middle element of the subarray to the left or right of the middle element, depending on whether the target element is smaller or larger than the middle element, respectively. The search continues this way, halving the size of the subarray at each step, until the target element is found or it is determined that the element is not present in the array.

One of the key features of exponential search is that it has a time complexity of **`O(log n)`,** which means that it can search through large arrays and lists relatively quickly. This makes it a useful algorithm for searching through large datasets, especially when the data is sorted.

2. *Code example.*

```
#include <iostream>

// Function to perform exponential search
int exponentialSearch(int arr[], int n, int x)
{
    // If x is present at first location itself
    if (arr[0] == x)
        return 0;

    // Find range for binary search by
    // repeated doubling
    int i = 1;
    while (i < n && arr[i] <= x)
        i = i*2;

    // Call binary search for the found range.
    return binarySearch(arr, i/2, std::min(i, n), x);
}

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is
// present, otherwise -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;
}

int main()
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = exponentialSearch(arr, n, x);
    (result == -1)? printf("Element is not present in array")
                  : printf("Element is present at index %d",
                                                    result);
    return 0;
}
```

```
Output: Element is present at index 3
```

This is because the element **`x`** with a value of 10 is present at index 3 in the array arr. When the **`exponentialSearch`** function is called, it first checks the element at the midpoint of the array (which is 4). Since 10 is greater than 4, the function searches the right half of the array. It then checks the element at the midpoint of the right half of the array (which is 10), and since this is the element we are searching for, the function returns the index 3.

3. Code explanation.

The example above shows how to implement the exponential search algorithm in C++. The exponential search algorithm is used to search for a specific element in a sorted array. It works by first checking the element at the midpoint of the array, and if the element is not found, it searches either the left or right half of the array depending on whether the element is smaller or larger than the midpoint element. This process is repeated until the element is found, or it is determined that the element is not present in the array.

The code begins by including the necessary headers and defining the **`exponentialSearch`** and **`binarySearch`** functions. The **`exponentialSearch`** function takes in an array **`arr`**, the size of the array n, and the element to be searched for x. It first checks if x is present at the first location in the array. If it is, it returns the index 0.

Next, the function performs a binary search by repeatedly doubling the index i until it is either greater than n or the element at index i is greater than x. This is done to find the range in which x might be present. Once the range has been found, the **`binarySearch`** function is called to search for x within that range.

The binarySearch function is a recursive function that takes in an array arr, the left and right indices l and r, and the element to be searched for x. It first checks if r is greater than or equal to l. If it is, it calculates the midpoint of the array using the formula **`mid = l + (r - l)/2`**. It then checks if the element at the midpoint is equal to x. If it is, the function returns the index mid.

If the element at the midpoint is not equal to x, the function checks if it is greater than x. If it is, the function calls itself with the left and midpoint-1 indices as the new left and right indices, respectively. If the element at the midpoint is not greater than x, the function calls itself with the midpoint+1 and right indices as the new left and right indices, respectively.

If r is not greater than or equal to l, the function returns -1, indicating that the element was not found in the array.

The main function declares and initializes an array arr, and calls the **`exponentialSearch`** function to search for the element x with a value of 10. The function returns the index at which x was found, and this value is printed to the console. If x is not found, the function returns -1, and a message is printed to the console indicating that the element was not found.

4. Time complexity.

The time complexity is **`O(log n).`**

The time complexity of the exponential search algorithm is O(log n), where n is the size of the array. This is because the algorithm works by repeatedly halving the size of the search space until the element is found or it is determined that the element is not present in the array.

The time complexity of the binary search function used in the example is also O(log n), as it also works by repeatedly halving the size of the search space.

Therefore, the overall time complexity of the example is O(log n), as the time complexity of the exponential search algorithm and the binary search function are both O(log n).

I wish you happy learning,

Your Rosen Rusev