Binary Search

Varun Jhaveri

What is Binary Search?

Search is the process of finding a value in a list of values. In other words, Searching is the process of locating a given value position in a list of values.

A binary search is a simple searching technique which can be applied if the items to be compared are either in ascending or descending order. The general idea used in binary search is similar to the way we search for the address of a person in a address book. Obviously we don't use linear search. Instead, we open the book from the middle and the name is compared with the element at the middle of the book. If the name is found, the corresponding address is retrieved and the searching has to be stopped. Otherwise, we search either the left part of the book or right part of the book. If the name to be searched is less than the middle element, search towards left otherwise, search towards right. The procedure is repeated till Target items is found or Target item is not found.

History of searching and sorting an array.

From the beginning of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. Among the authors of early sorting algorithms around 1951 was Betty Holberton, who worked on ENIAC and UNIVAC. Bubble sort was analysed as early as 1956.Asymptotically optimal algorithms have been known since the mid-20th century – new algorithms are still being invented, with the widely used Timsort dating to 2002, and the library sort being first published in 2006.

Comparison sorting algorithms have a fundamental requirement of $\Omega(n \log n)$ comparisons (some input sequences will require a multiple of $n \log n$ comparisons, where n is the number of elements in the array to be sorted). Algorithms not based on comparisons, such as counting sort, can have better performance.

Sorting small arrays optimally (in fewest comparisons and swaps) or fast (i.e. taking into account machine specific details) is still an open research problem, with solutions only known for very small arrays (<20 elements). Similarly optimal (by various definitions) sorting on a parallel machine is an open research topic.

Different implementations of Binary Search.

There are 4 variants for binary search which are listed below:

Classic Binary Search

The classic binary search, the foundation of this algorithm, operates by comparing the target element with the middle element of a sorted array. This technique is widely recognized for its efficiency in narrowing down the search space by half with each iteration.

Interpolation Search

Interpolation search takes a different approach by estimating the target's position based on its value. It adjusts the search space according to this estimation, potentially providing faster convergence, especially when the data is uniformly distributed.

Exponential (Doubling) Binary Search

The exponential search, often referred to as doubling or galloping binary search, optimizes the search process by doubling the search range at each step. This method is particularly advantageous when dealing with unbounded datasets or when the precise location of the target is unknown.

Recursive and Iterative Implementations

Binary search can be implemented using both recursive and iterative methods. Recursive implementations offer an elegant, concise code structure but can consume more memory due to the call stack. Iterative implementations, on the other hand, often use less memory and are favored in resource-constrained environments. These two implementation approaches cater to different programming needs, allowing for flexibility in applying binary search in various scenarios.

Implementation

BINARY SEARCH ONLY WORKS ON SORTED ARRAYS.

While designing the program, the low is considered as the position of first element it is initialised to 0 and high as the position of the last element it is initialised to n-1, the middle element position can be obtained using

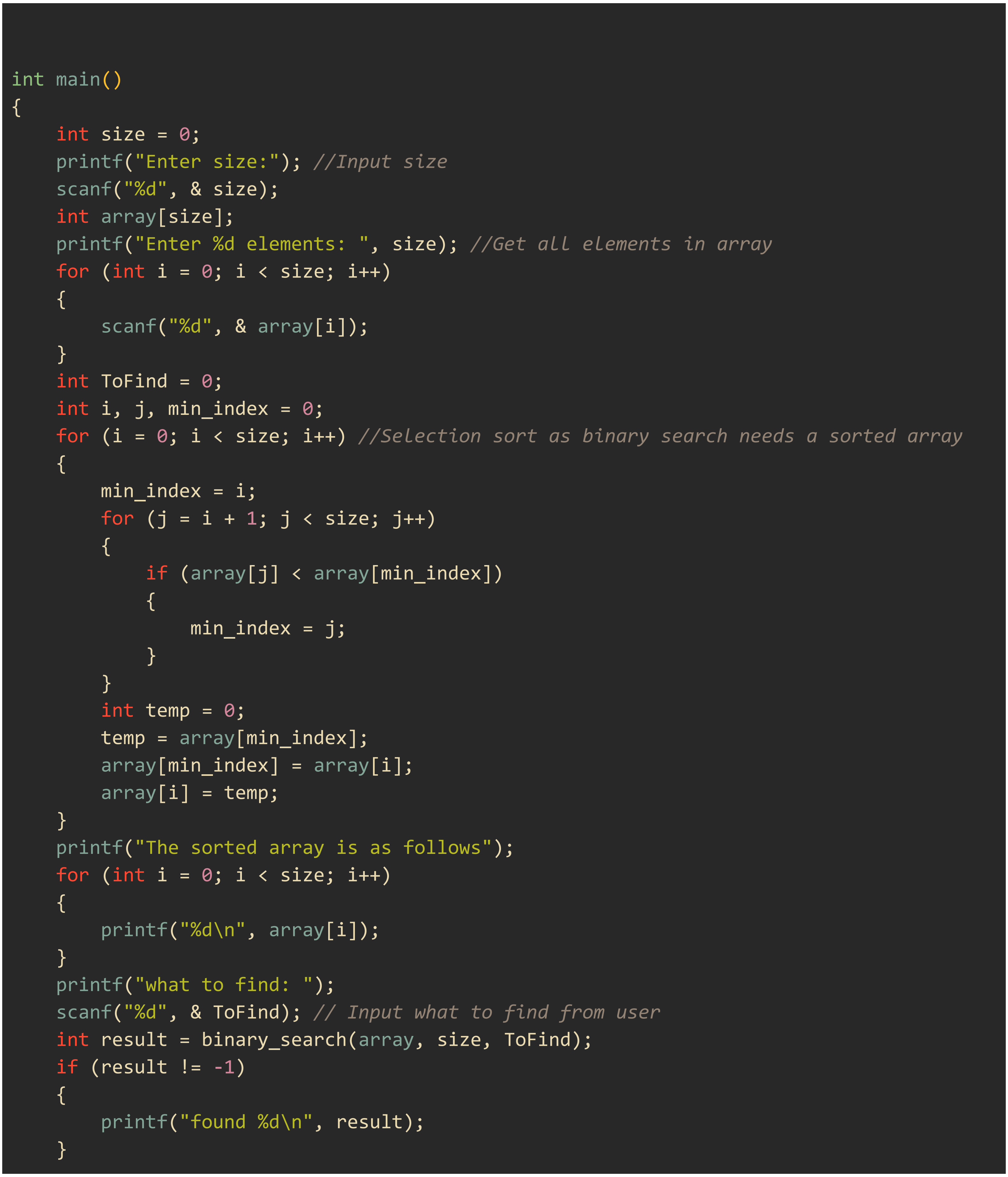
mid= (low+high)/2

The Target Element to be searched is compared with the middle element. If they are equal the position of the item in the array is returned. If the condition is false, the target element may be present in either the left part of the array or in the right part of the array. If Target element is less than the middle element then the left part of the array has to be compared from low to mid-1.Otherwise, the right part of the array has to be compared from mid+1 to high. Finally when low exceeds high, it indicates that item not found in the array

Code

```c
#include <stdio.h>
int binary_search(int array[], int size, int ToFind) //Binary search
{
    int lowerbound = 0;
    int higherbound = size - 1;

    while (lowerbound <= higherbound) {
        int midvalue = (lowerbound + higherbound) / 2;
        if (array[midvalue] == ToFind)
        {
            return midvalue;
        }
        else if (array[midvalue] < ToFind)
        {
            lowerbound = midvalue + 1;
        }
        else
        {
            higherbound = midvalue - 1;
        }
    }
    return -1;
}
```



```c
int main()
{
    int size = 0;
    printf("Enter size:"); //Input size
    scanf("%d", & size);
    int array[size];
    printf("Enter %d elements: ", size); //Get all elements in array
    for (int i = 0; i < size; i++)
    {
        scanf("%d", & array[i]);
    }
    int ToFind = 0;
    int i, j, min_index = 0;
    for (i = 0; i < size; i++) //Selection sort as binary search needs a sorted array
    {
        min_index = i;
        for (j = i + 1; j < size; j++)
        {
            if (array[j] < array[min_index])
            {
                min_index = j;
            }
        }
        int temp = 0;
        temp = array[min_index];
        array[min_index] = array[i];
        array[i] = temp;
    }
    printf("The sorted array is as follows");
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", array[i]);
    }
    printf("what to find: ");
    scanf("%d", & ToFind); // Input what to find from user
    int result = binary_search(array, size, ToFind);
    if (result != -1)
    {
        printf("found %d\n", result);
    }
    else
    {
        printf("not there\n");
    }

    return 0;
}
```

Citations

https://www.geeksforgeeks.org/variants-of-binary-search/
https://en.wikipedia.org/wiki/Binary_search_algorithm
https://ieeexplore.ieee.org/document/4797609