# Implementing Binary Search Algorithm

Following are the steps of implementation that we will be following:

1. Start with the middle element:
    - If the **target** value is equal to the middle element of the array, then return the index of the middle element.
    - If not, then compare the middle element with the target value,
        - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
        - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
2. When a match is found, return the index of the element matched.
3. If no match is found, then return `-1`

Binary Search Algorithm can be implemented in two ways which are discussed below.

1. Iterative Method
2. Recursive Method

## Algorithm Binary Search

### Iteration Method

```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > A[mid]) // x is on the right side
        low = mid + 1
    else                       // x is on the left side
        high = mid - 1
```

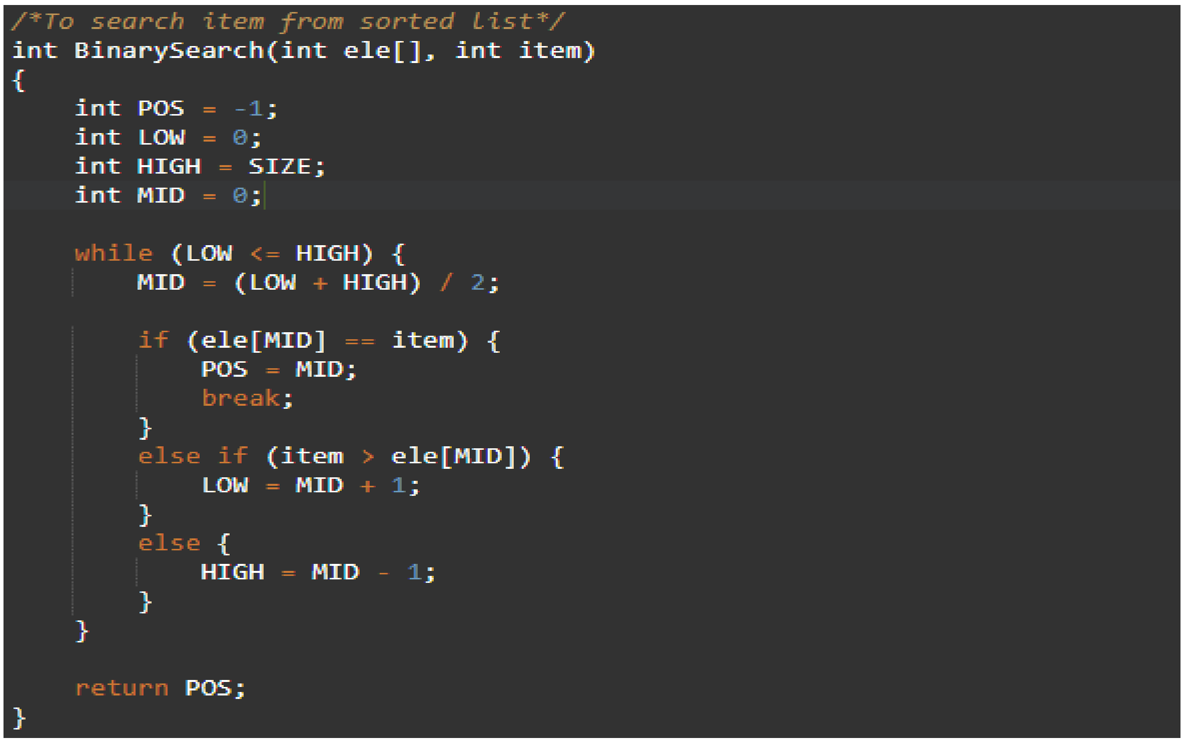
## Recursive Method

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x < data[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                               // x is on the right side
            return binarySearch(arr, x, low, mid - 1)
```

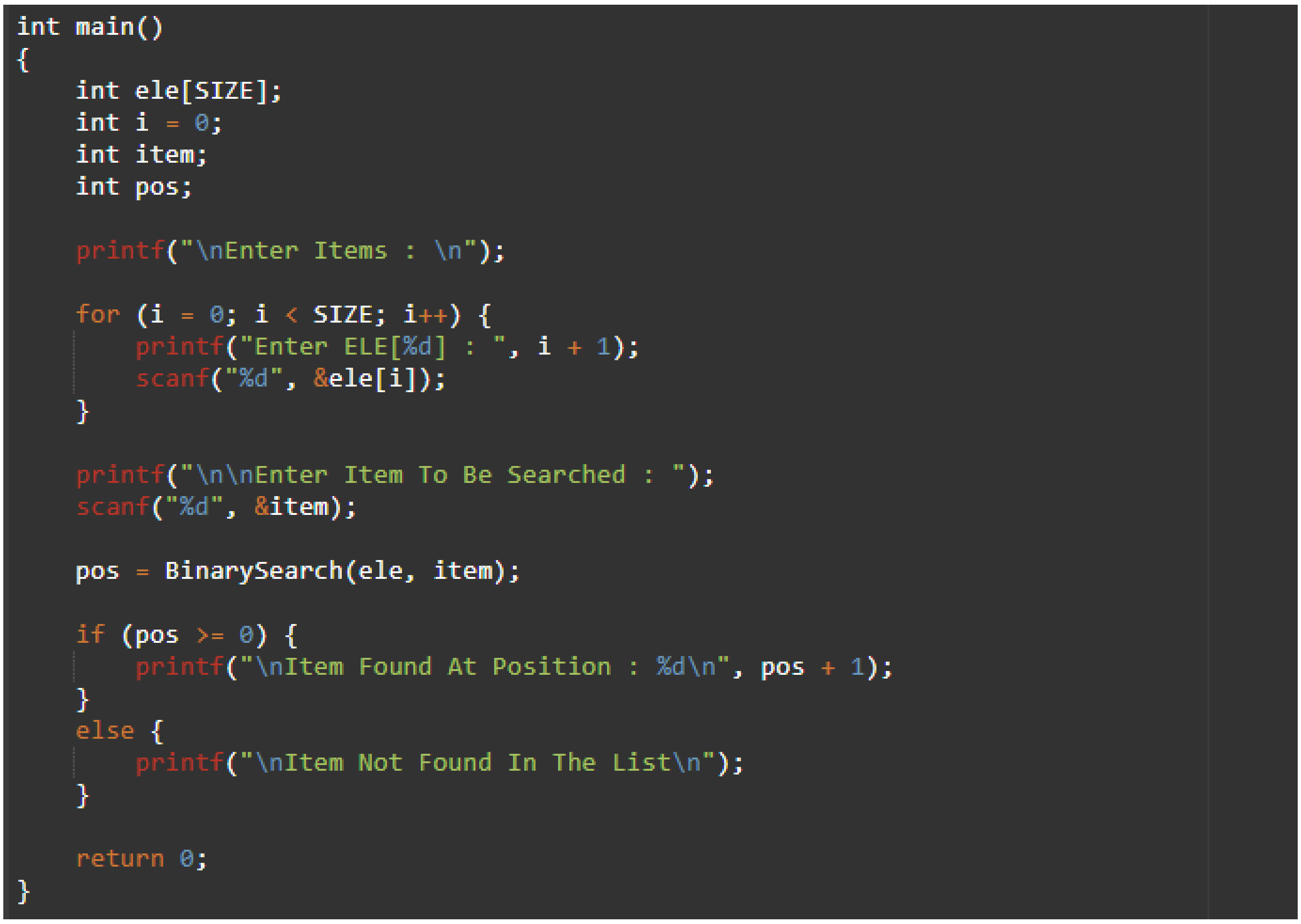
## Complexity

| SN | Performance | Complexity |
|---|---|---|
| 1 | Worst case | O(log n) |
| 2 | Best case | O(1) |
| 3 | Average Case | O(log n) |
| 4 | Worst case space complexity | O(1) |

## Iteration

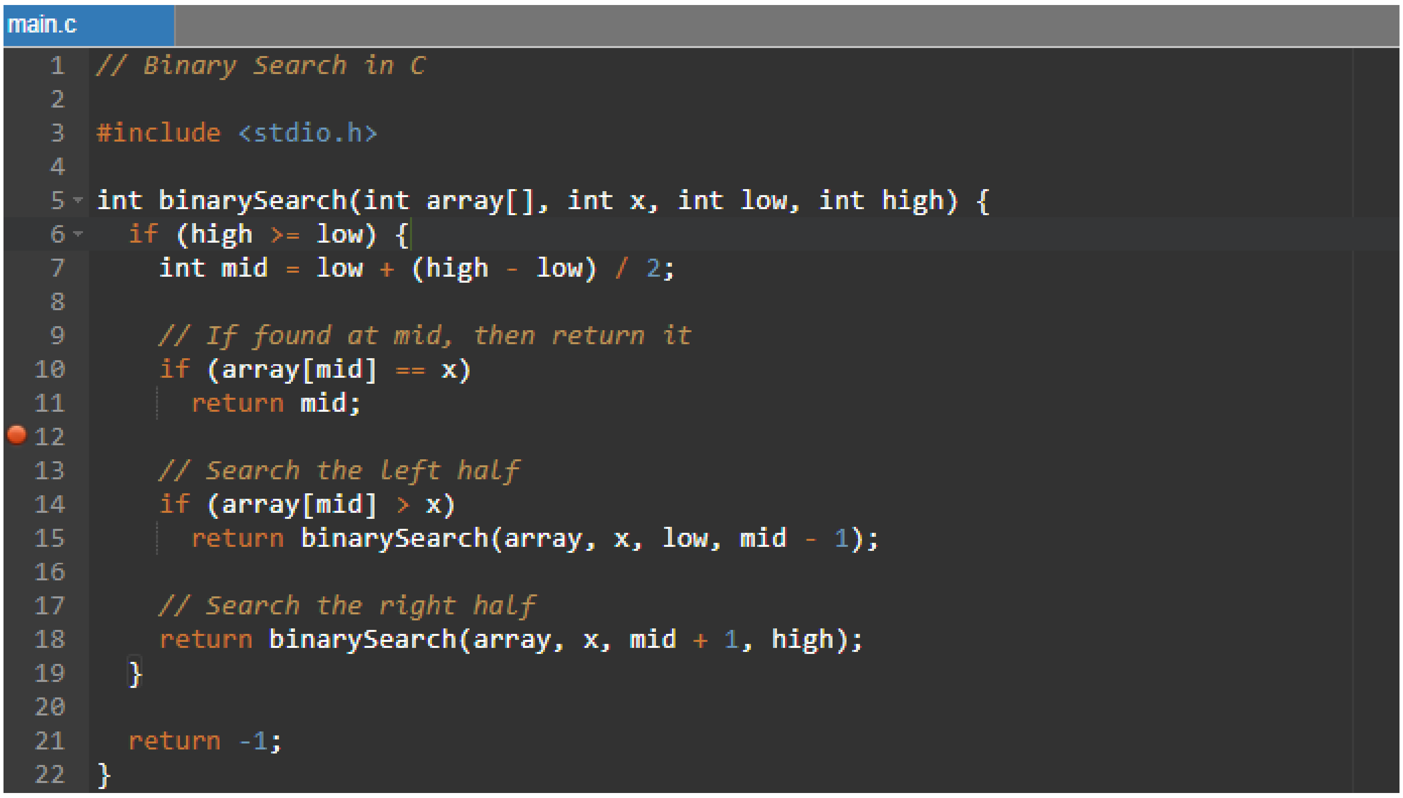


```c
/*To search item from sorted list*/
int BinarySearch(int ele[], int item)
{
    int POS = -1;
    int LOW = 0;
    int HIGH = SIZE;
    int MID = 0;

    while (LOW <= HIGH) {
        MID = (LOW + HIGH) / 2;

        if (ele[MID] == item) {
            POS = MID;
            break;
        }
        else if (item > ele[MID]) {
            LOW = MID + 1;
        }
        else {
            HIGH = MID - 1;
        }
    }

    return POS;
}
```



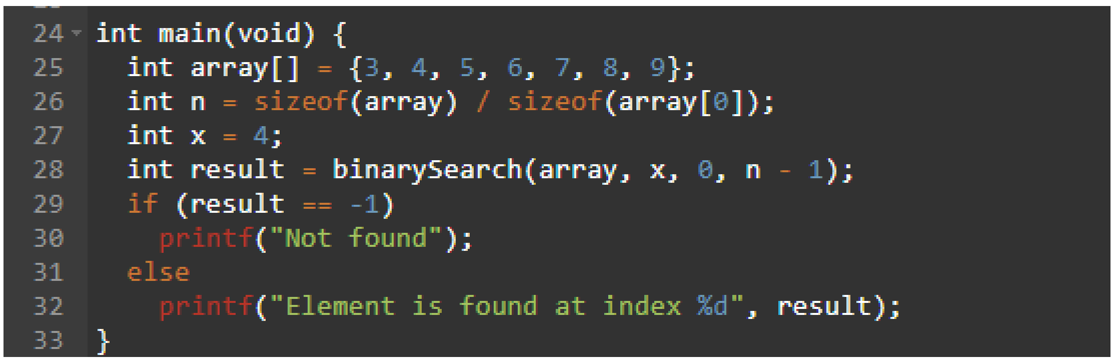
```c
int main()
{
    int ele[SIZE];
    int i = 0;
    int item;
    int pos;

    printf("\nEnter Items : \n");

    for (i = 0; i < SIZE; i++) {
        printf("Enter ELE[%d] : ", i + 1);
        scanf("%d", &ele[i]);
    }

    printf("\n\nEnter Item To Be Searched : ");
    scanf("%d", &item);

    pos = BinarySearch(ele, item);

    if (pos >= 0) {
        printf("\nItem Found At Position : %d\n", pos + 1);
    }
    else {
        printf("\nItem Not Found In The List\n");
    }

    return 0;
}
```

## Recursion



main.c

```c
// Binary Search in C

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;

    // If found at mid, then return it
    if (array[mid] == x)
      return mid;

    // Search the left half
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);

    // Search the right half
    return binarySearch(array, x, mid + 1, high);
  }

  return -1;
}
```



```c
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
}
```

**You can use sizeof in iteration method as well it just a way of use it**