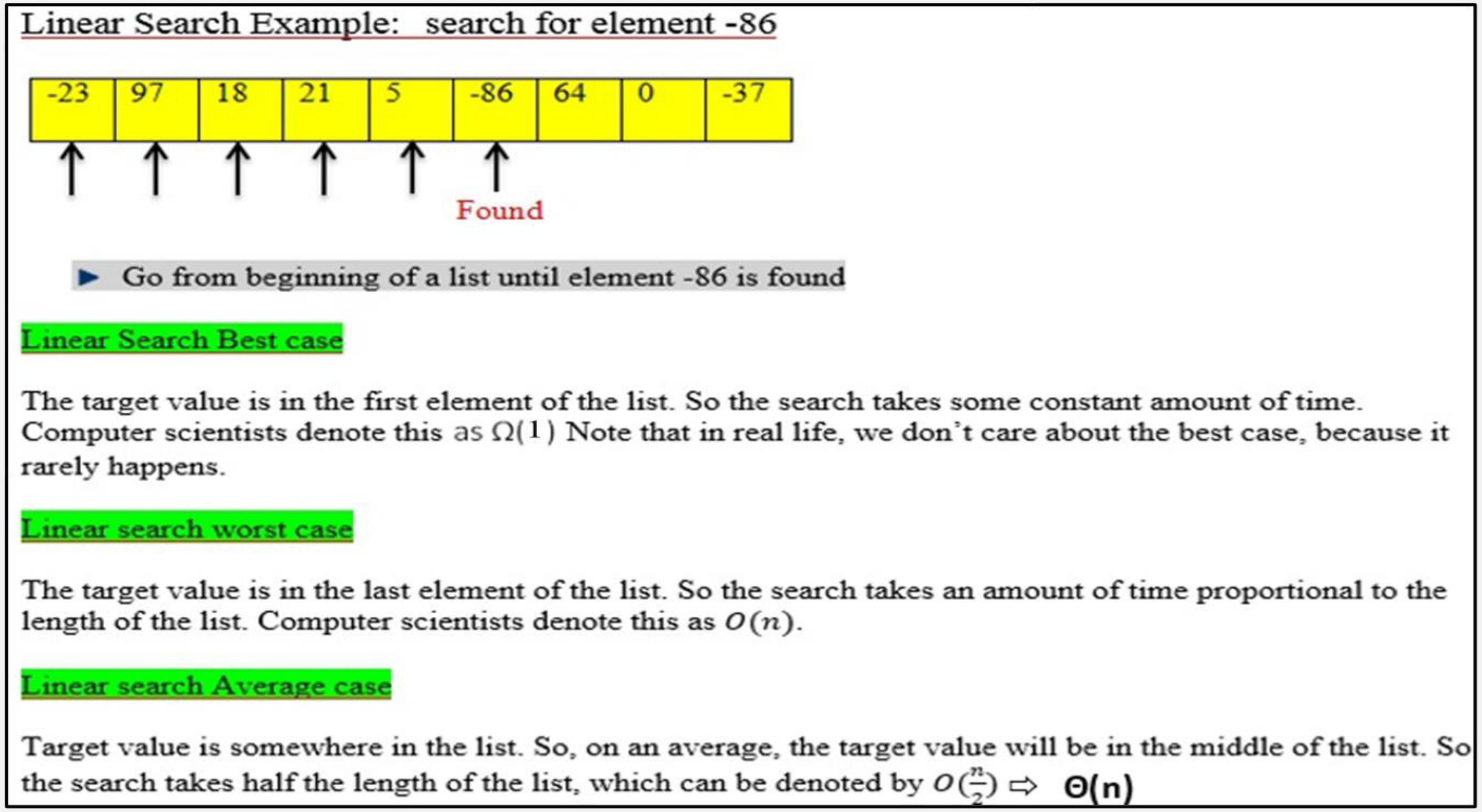
SEARCHING ALGORITHMS

- The searching algorithms are used to search or find one or more than one element from a dataset. These type of algorithms are used to find elements from a specific data structures.
- There are three popular algorithms available:
  - ✓ Linear Search
  - ✓ Binary Search
  - ✓ Jump Search

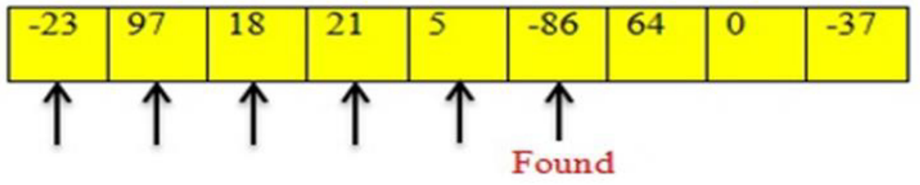
LINEAR SEARCH

- Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that item is returned, otherwise the search continues till the end of the data collection.
- **Inputs:**
  - ✓ Data Structure ➔ ex. array
  - ✓ Search Value

LINEAR SEARCH EXAMPLE



Linear Search Example: search for element -86



► Go from beginning of a list until element -86 is found

Linear Search Best case

The target value is in the first element of the list. So the search takes some constant amount of time. Computer scientists denote this as $\Omega(1)$ Note that in real life, we don't care about the best case, because it rarely happens.

Linear search worst case

The target value is in the last element of the list. So the search takes an amount of time proportional to the length of the list. Computer scientists denote this as $O(n)$.

Linear search Average case

Target value is somewhere in the list. So, on an average, the target value will be in the middle of the list. So the search takes half the length of the list, which can be denoted by $O(\frac{n}{2}) \Rightarrow$ $\Theta(n)$

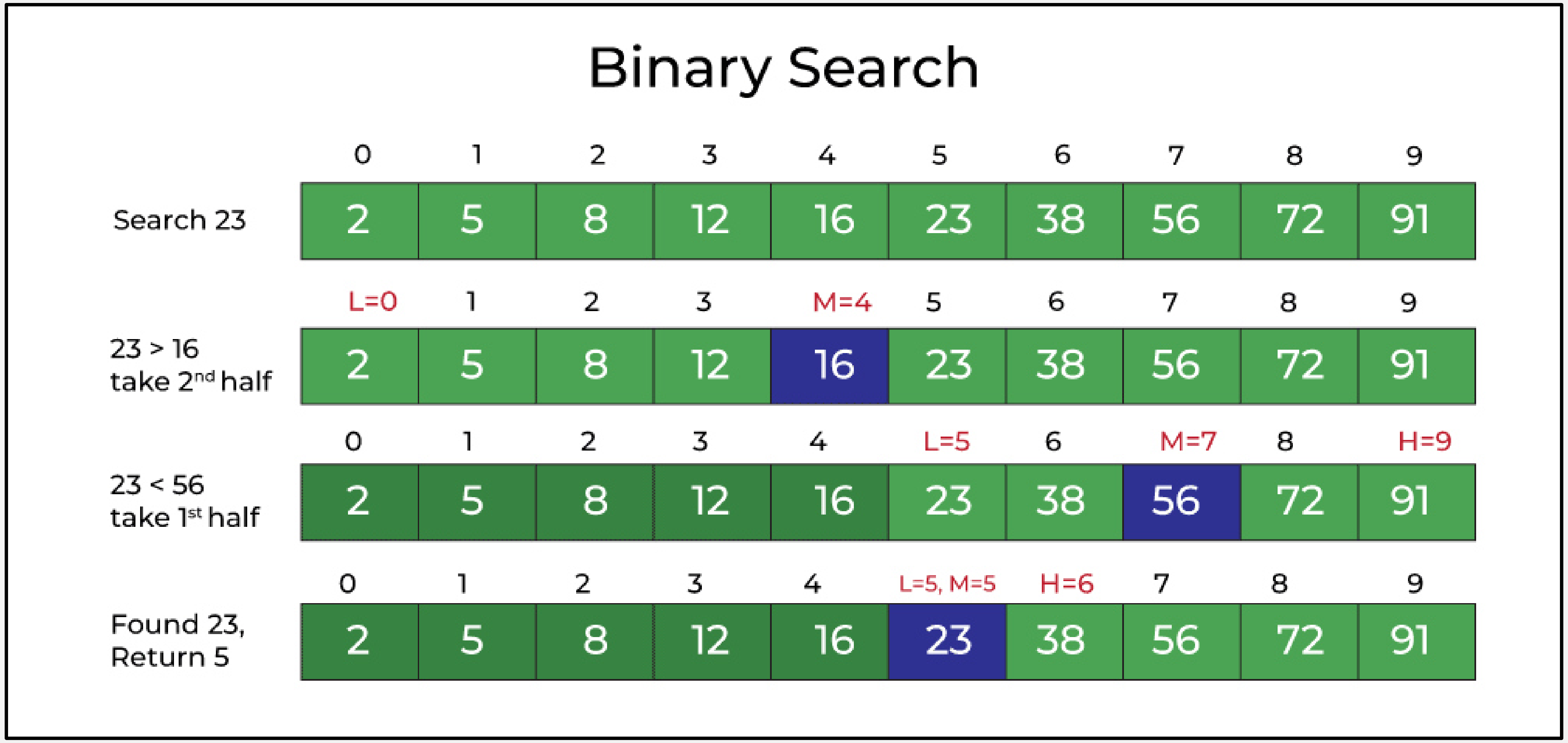
LINEAR SEARCH CODE

```
lst = []
items = int(input("Enter the number of items: "))
for n in range(items):
    numbers = int(input("Enter the %d number: " %n))
    lst.append(numbers)
keyValue = int(input("Enter number to search for: "))
found = False
for i in range(len(lst)):
    if lst[i] == keyValue:
        found = True
        print("%d found at location %d" % (keyValue, i))
        break
if not found:
    print("%d is not in list" % keyValue)
input()
```

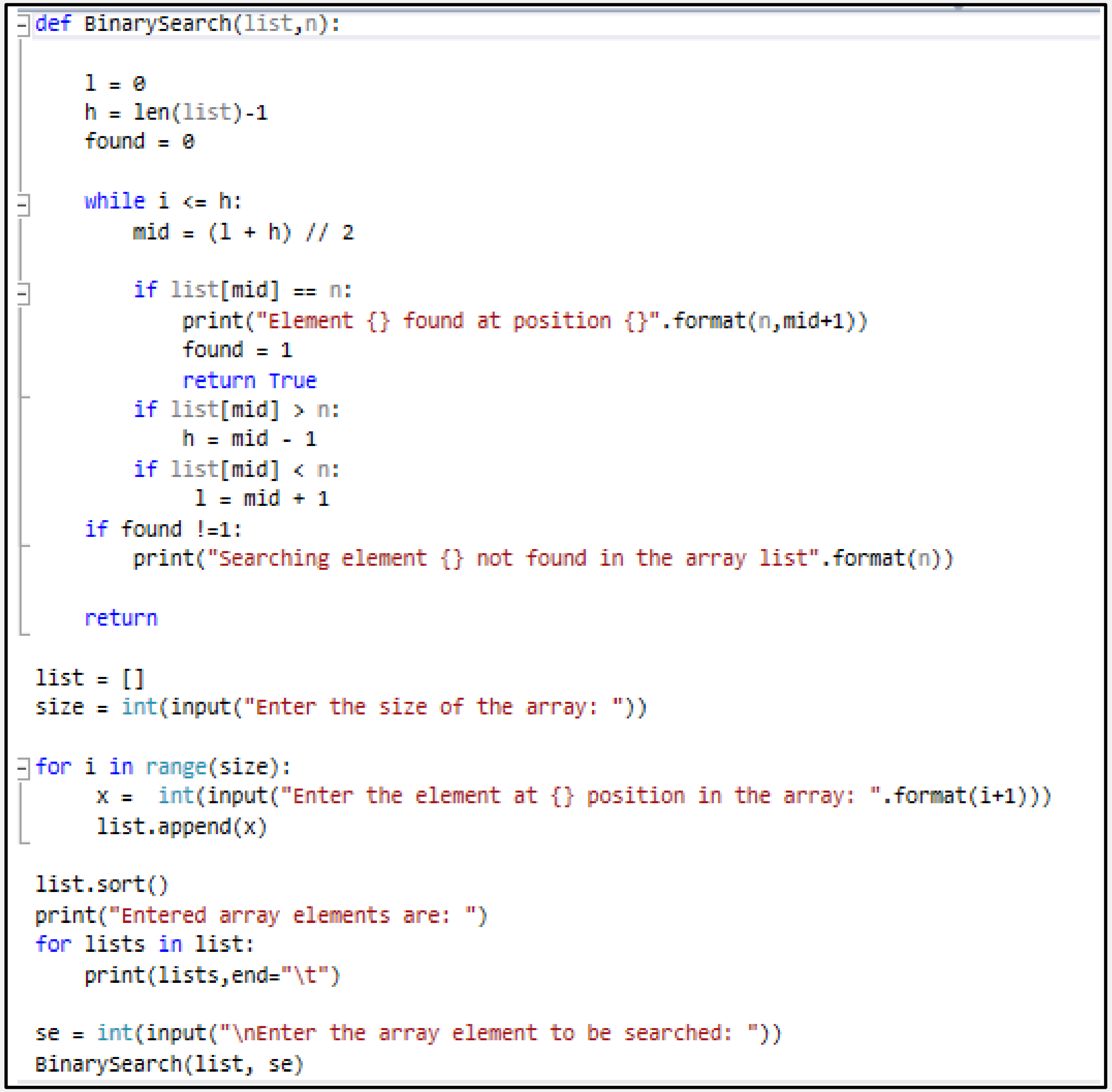
BINARY SEARCH

- Binary search is a search algorithm that finds the position of a target value within **a sorted array.**
- A binary search begins by comparing the middle element of the array with the target value. If the target value matches the middle element, its position in the array is returned. If the target value is less or more than the middle element, the search continues the lower or upper half of the array respectively with a new middle element, eliminating the other half from consideration.
- **Inputs:**
  - ✓ Data Structure ➔ ex. array
  - ✓ Search Value
- Best case time complexity ➔ O(1), worst and average case time complexity ➔ O(logn)

BINARY SEARCH EXAMPLE

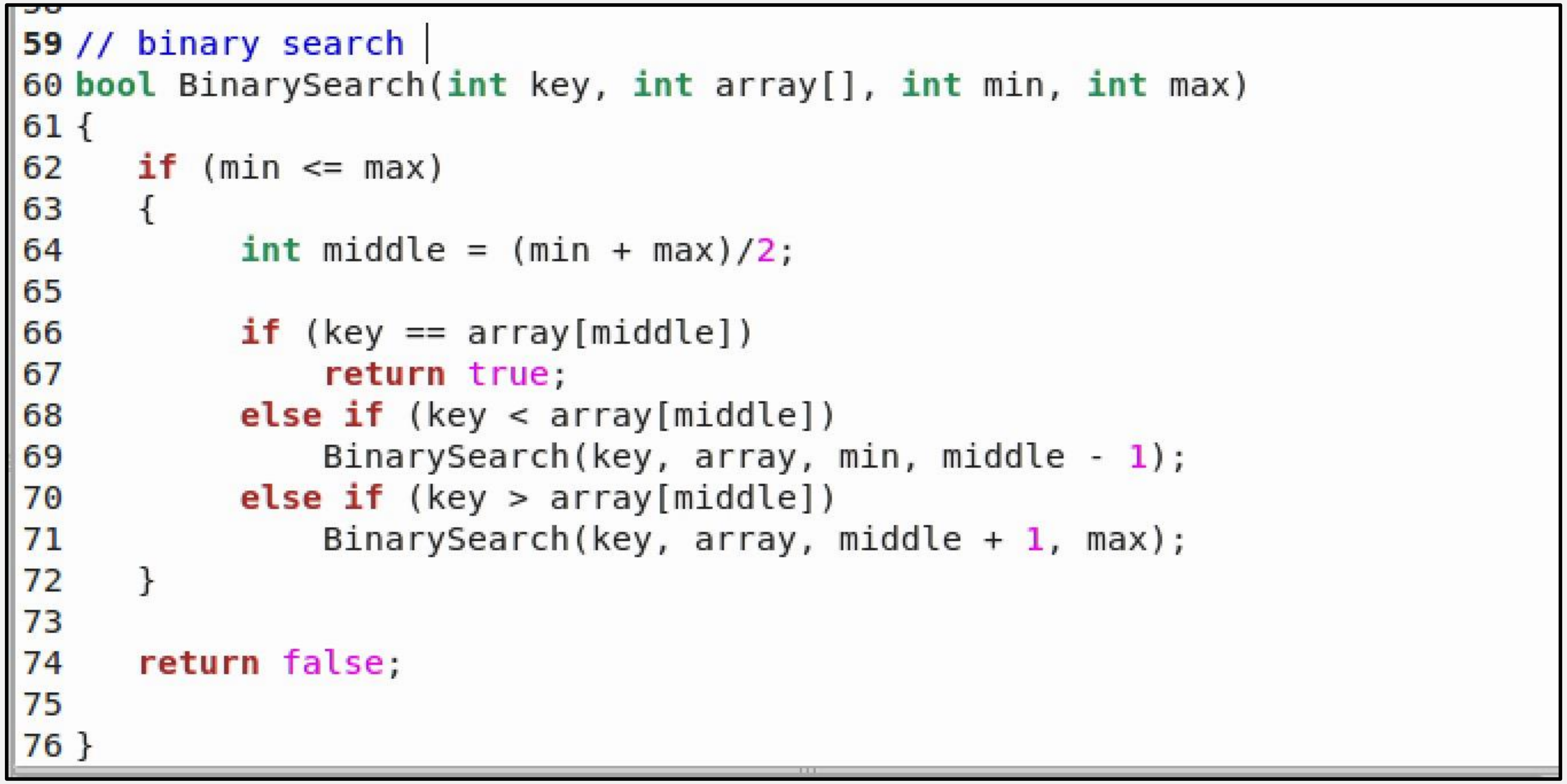


BINARY SEARCH CODE



```python
def BinarySearch(list,n):

    l = 0
    h = len(list)-1
    found = 0

    while i <= h:
        mid = (l + h) // 2

        if list[mid] == n:
            print("Element {} found at position {}".format(n,mid+1))
            found = 1
            return True
        if list[mid] > n:
            h = mid - 1
        if list[mid] < n:
             l = mid + 1
    if found !=1:
        print("Searching element {} not found in the array list".format(n))

    return

list = []
size = int(input("Enter the size of the array: "))

for i in range(size):
     x =  int(input("Enter the element at {} position in the array: ".format(i+1)))
     list.append(x)

list.sort()
print("Entered array elements are: ")
for lists in list:
    print(lists,end="\t")

se = int(input("\nEnter the array element to be searched: "))
BinarySearch(list, se)
```

BINARY SEARCH CODE USING RECURSION



```
// binary search
bool BinarySearch(int key, int array[], int min, int max)
{
    if (min <= max)
    {
        int middle = (min + max)/2;

        if (key == array[middle])
            return true;
        else if (key < array[middle])
            BinarySearch(key, array, min, middle - 1);
        else if (key > array[middle])
            BinarySearch(key, array, middle + 1, max);
    }

    return false;

}
```