Binary Search Working

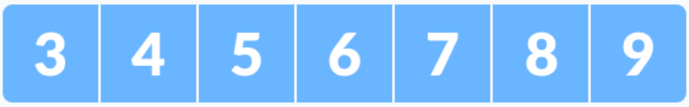
Binary Search Algorithm can be implemented in two ways which are discussed below.

1. Iterative Method
2. Recursive Method

The recursive method follows the divide and conquer approach.

The general steps for both methods are discussed below.

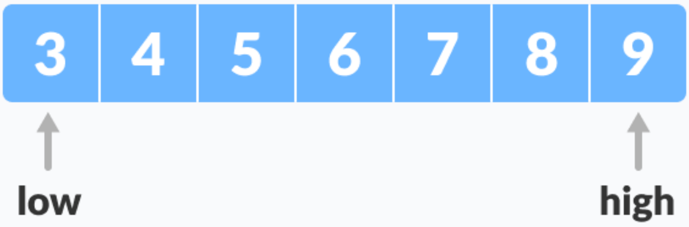
1. The array in which searching is to be performed is:



Initial array

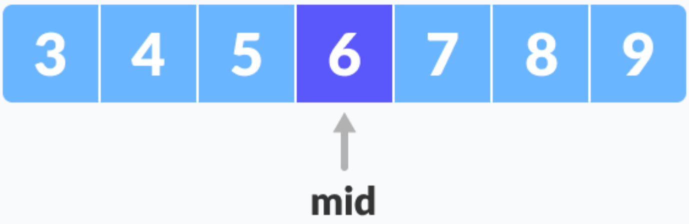
Let x = 4 be the element to be searched.

2.Set two pointers low and high at the lowest and the highest positions respectively



Setting pointers

3.Find the middle element `mid` of the array ie. `arr[(low + high)/2] = 6`.

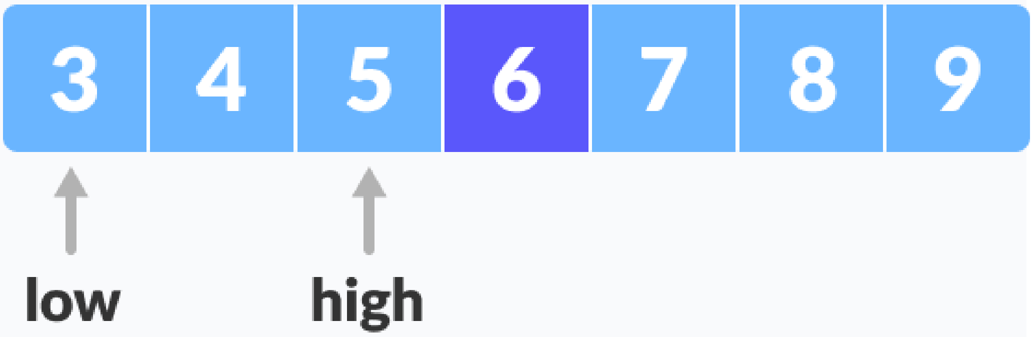


Mid element

4.If x == mid, then return mid.Else, compare the element to be searched with m.

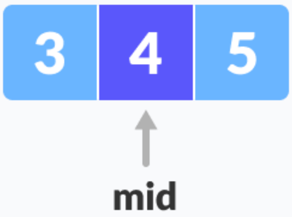
5.If `x > mid`, compare `x` with the middle element of the elements on the right side of `mid`. This is done by setting `low` to `low = mid + 1`.

6.Else, compare `x` with the middle element of the elements on the left side of `mid`. This is done by setting `high` to `high = mid - 1`.

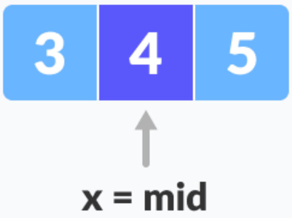


low meets high. Mid element

7.Repeat steps 3 to 6 until



8. x = 4 is found. Found



Binary Search Algorithm

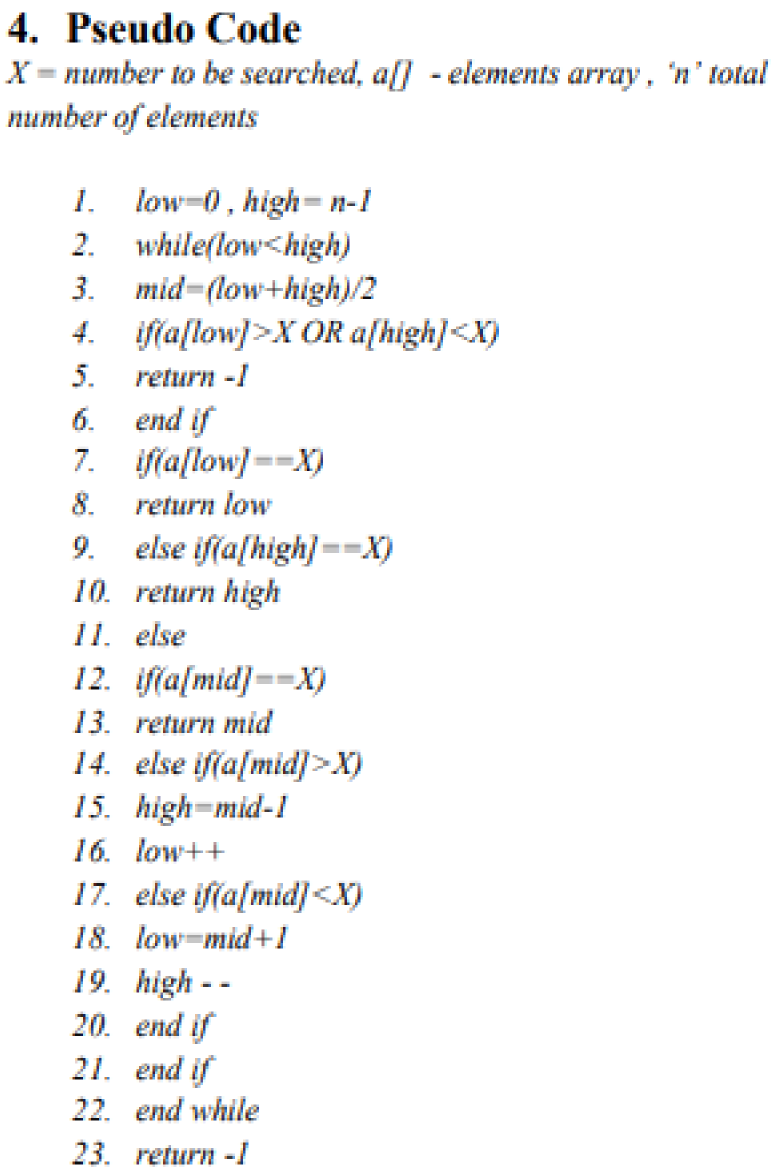
Iteration Method

```
do until the pointers low and high meet each other.

    mid = (low + high)/2

    if (x == arr[mid])

        return mid

    else if (x > arr[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 > arr[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)
```



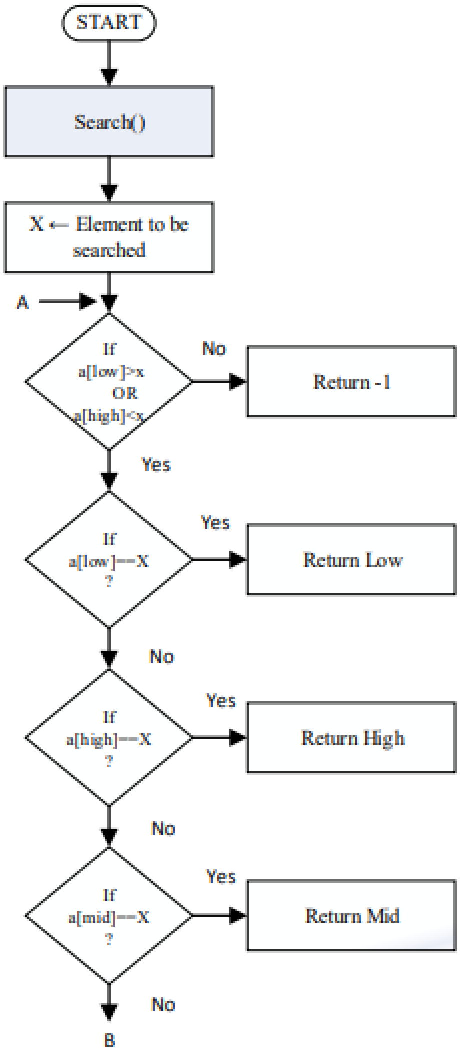
4. Pseudo Code

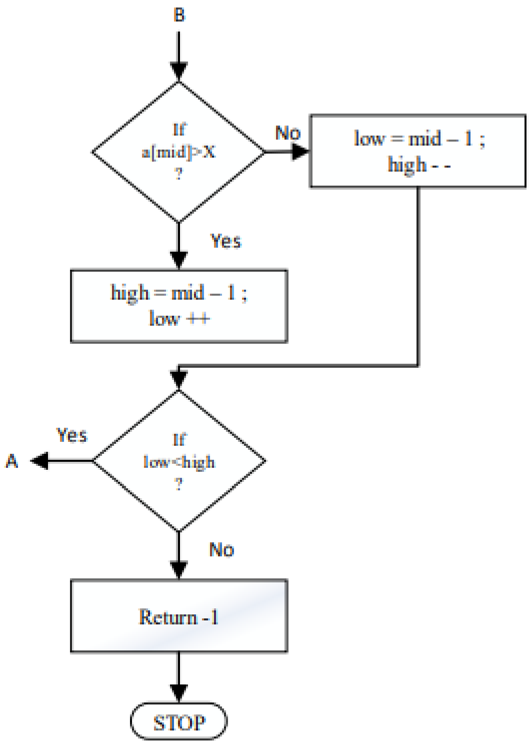
X = number to be searched, a[] - elements array , 'n' total number of elements

```
low=0 , high= n-1
while(low<high)
    mid=(low+high)/2
    if(a[low]>X OR a[high]<X)
        return -1
    end if
    if(a[low]==X)
        return low
    else if(a[high]==X)
        return high
    else
        if(a[mid]==X)
            return mid
        else if(a[mid]>X)
            high=mid-1
            low++
        else if(a[mid]<X)
            low=mid+1
            high - -
        end if
    end if
end while
return -1
```

Flow chart :-

Below is the flow chart for the Modified binary search algorithm



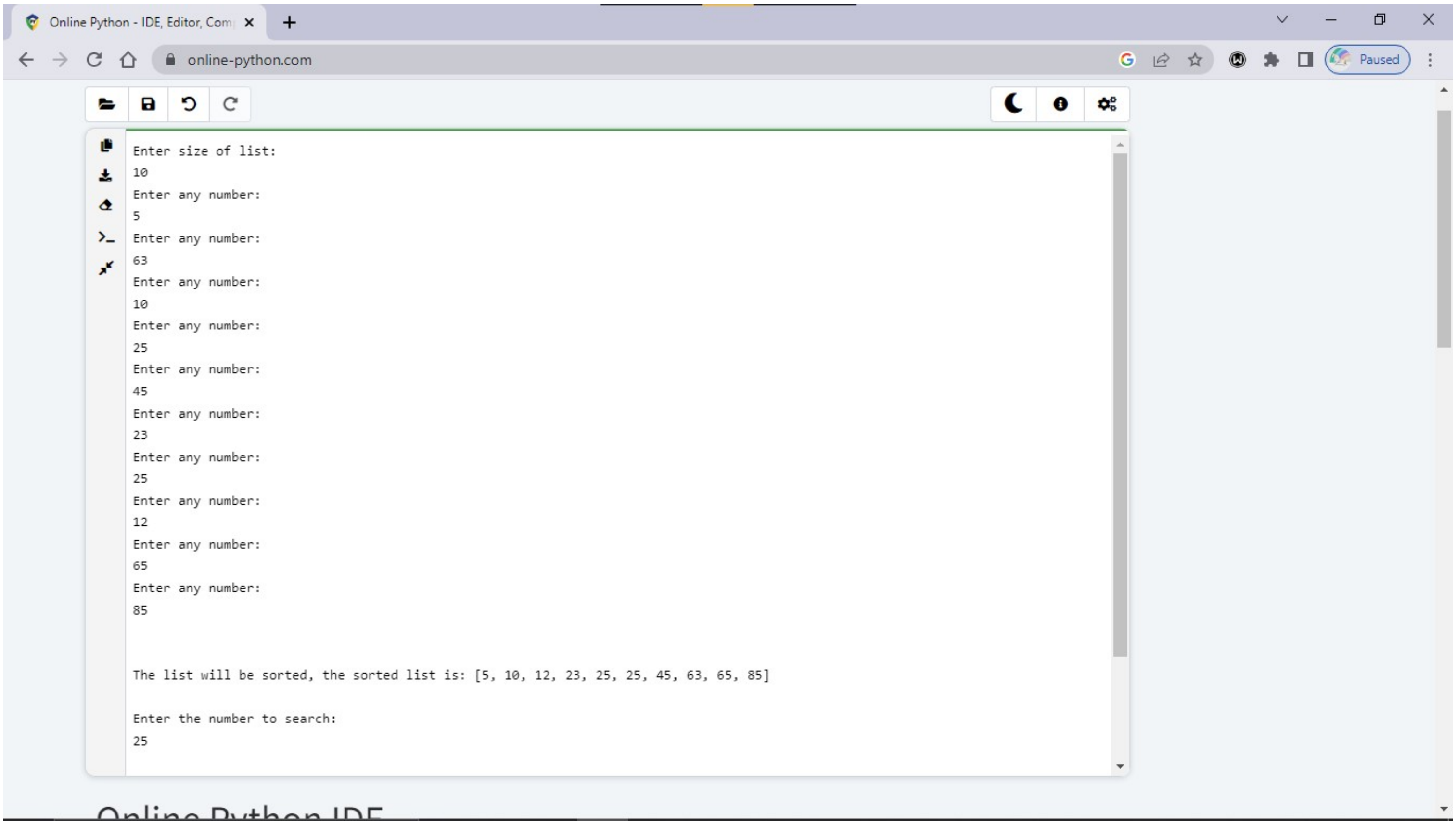


Terms used,
Search() – function call to check whether the given element is present or not

Low – the lowest index in the array
High- highest index in the array
Middle – middle element's index in the array
X – element to be searched

Coding:-(value from user in python)

```
def “indrjeet saini”
def binary_sort(sorted_list, length, key):
    start = 0
    end = length-1
    while start <= end:
        mid = int((start + end)/2)
        if key == sorted_list[mid]:
            print("\nEntered number %d is present at position: %d" % (key, mid))
            return -1
        elif key < sorted_list[mid]:
            end = mid - 1
        elif key > sorted_list[mid]:
            start = mid + 1
    print("\nElement not found!")
    return -1


lst = []


size = int(input("Enter size of list: \t"))


for n in range(size):
    numbers = int(input("Enter any number: \t"))
    lst.append(numbers)
lst.sort()

print('\n\nThe list will be sorted, the sorted list is:', lst)


x = int(input("\nEnter the number to search: "))


binary_sort(lst, size, x)
```



```
Enter size of list:
10
Enter any number:
5
Enter any number:
63
Enter any number:
10
Enter any number:
25
Enter any number:
45
Enter any number:
23
Enter any number:
25
Enter any number:
12
Enter any number:
65
Enter any number:
85


The list will be sorted, the sorted list is: [5, 10, 12, 23, 25, 25, 45, 63, 65, 85]

Enter the number to search:
25
```