# Binary Search

## Introduction:

Suppose there is an Array of numbers given. E.g : {7,14,17,21,35,60,92,121,155}

We need to find if number x is on the Array or not.

Let's say, if x=92. Then the straight forward approach could be to run a for loop, and we will compare each number of the list/array with x. And if found, then we would return true.

Algorithm of the straight forward approach would be:

```
Find_number(x)
{
        for(i=0;i<array.length();i++)
        {
                if(array[i]==x)
                {
                        return true;
                }
        }
        return  false;
}
```

*(Find_number is a function to check if the number is in the array or not)*

**The worst case time complexity of this approach is 0(n)**

So if we need to check a particular number in an array list of millions or billions of numbers, then it would require lots of times.

We can reduce this time complexity by using Binary_Search algorithm.

But there are some requirements before using binary search. These are:

i. Array should be sorted.
ii. And if the Array is not sorted, then the programmer need to sort the array to reduce the time complexity of searching the element.

# How Binary Search Works:

Suppose,

There are *n* numbers in an array.

e.g: 7,14,17,21,35,60,92,121,155

Here, n=9. (As there are 9 numbers in the array)

Here the value of middle element is 35. (In an array of 9 numbers, the middle element is the $5^{th}$ index if we consider first element as $1^{st}$ index).

So, starting index = $1^{st}$ index (having value 7).

Ending index = $9^{th}$ index (having value 155).

Middle index = (Start index + End index)/2 = (1+9)/2 = $5^{th}$ index (having value 35).

[ We know that, mid= (low + high)/2 ]

As the arrays are sorted, so suppose we are searching for 92. If we know the value of the middle element, then we can determine whether the value is less than the value of middle element or greater than middle element.

## Example: 1

Suppose, we are searching for **92.**

7,14,17,21,35,60,92,121,155

Here 92 is greater than the value of middle 35. So we don't need to check the left part of the middle element. We can say we only need to search on the right part of the middle element.

On the right part of the middle element, there are 4 numbers. We only need to check 92 on that part.

So now we will consider starting index as the next index after the middle elements index.

Now, Starting index = middle index + 1 = $5^{th}$ index + 1 = $6^{th}$ index

End index = $9^{th}$ index (Did not change)

Middle index = (6+9)/2 = $7^{th}$ index (Integer Division, avoided floating point)

Here, $7^{th}$ index element value is 92. Which is equal 92. So 92 is found!

## Example: 2

# 7,14,17,21,35,60,92,121,155

**(Those who have already understood binary search, don't need to read example 2 or 3)**

Suppose, we are searching for **17.**

So at the beginning, starting index = 1st index (having value 7).

Ending index = 9th index (having value 155).

Middle index = (1+9)/2 = 5th index (having value 35).

So 17 is less than 35. As Array is sorted, it should be on the left half of the array.

So we, only need to search in between index 1 and index 4 for the value 17.

now starting index is still 1st index.

Ending index is now the index before the middle element which is 4th index.

Ending Index = Middle Index – 1 = 5th index – 1 = 4th index.

Middle index = (1+4)/2 = 2nd index.

Value of middle element is 14. 17 is greater than 14, so it is on the right side of 14.

Now starting index should be the next index of middle index.

So, now starting index = middle index+1= 2nd index + 1 = 3rd index

End Index = 4th index (Did not change)

Middle index = (3+4)/2 = 3rd index

Value of middle index is now 17. So 17 is found. We would now return found.

## Example: 3

# 7,14,17,21,35,60,92,121,155

**(Those who have already understood binary search, don't need to read example 3)**

Suppose we are searching for the value **51.**

Start index=1st index

End index=9th index

Middle index=(1+9)/2=5th index.

Value of middle index = 35. 51 is greater than 35. So 51 should be on the right side of middle element.

Now Starting index = 1 + Middle Index = 6th index

Ending Index = 9th index (Did not change)

Middle Index = (6+9)/2 = 7th index

Middle index is now 7 and Value of middle index is now 92. 92 > 51 (We are searching 51). So searching value should be on left of middle index.

So we would reduce value of Ending index now.

Start index = 6th index (Did not change)

End index = middle index – 1 = 6th index

Middle index = (6+6)/2 = 6 index

Middle index is now 6 and Value of middle index is now 60. 60 > 51. So searching value should be on left of Middle index.

Start index = 6th index (Did not change)

End index = middle index – 1 = 5th index

But wait, our end index has become less than start index! So when end index becomes less than start index, it means the searching value is not found in the array. We would return ***not_found*** !

## Algorithm:

```
While(Start_Index<End_Index){
        mid_index = ( start_index + end_index ) /2;
        if(array[mid index]==searching_Value) return Found;
        else If (searching_value > array[mid_index])
        {
                Start_Index=mid_index+1;
        }
        else If (searching_value < array[mid_index])
        {
                End_Index=mid_index-1;
        }
}
```

**Worst Case Complexity of Binary Search: 0(log(n))**

Points to be needed:

1. When the searching value is less than middle elements value, then starting index will not be changed. Ending index would become the index 1 before the middle element.
2. When the searching value is greater than middle elements value, then ending index will not be changed. starting index would become the index 1 after the middle element.
3. We would repeat this approach until we have found the value.
4. If the ending index become greater than starting index, it means that the value is not found.
5. Time Complexity in Binary Search is O(log(n)). Which is much less than 0(n). So it is a faster approach to avoid time limit.
6. When you write this algorithm in your program, the starting index at the beginning will be 0 and ending index will be *total_number_of_elements - 1*. But here, at the beginning we consider starting index as 1 and ending index as *total_number_of_elements* for better understanding.

- **Md. Tahmid Rahman Laskar**

**IUT '11**