Binary Search in C++

Jaydeep Shah (Email: radhey04ec@gmail.com)

Time complexity: In theoretical computer science, the time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm.

Linear search example:

```
//Linear search in C++
//Linear search involves comparing each element in an array with a given key. While this method is straightforward and easy to implement, it can be time-consuming when dealing with larger arrays.
//Created by : Jaydeep Shah (radhey04ec@gmail.com)


#include <iostream>
using namespace std;

//Function for finding key from array
//It will retrun position / index from array if key value is present in Array,otherwise return -1
int find_key(int my_array[],int size,int key_details)
{
    for(int j = 0; j < size; j++)
    {
        if(my_array[j] == key_details)
        {
```

```
            return j;
        }
    }

    return -1;   //If nothing found
}

int main()
{
    int total_array_elements;

    cout << "Enetr How many number you want to store into Array ? : " << endl;
    cin >> total_array_elements;

    int array[total_array_elements];  //Create Array
    cout << "Enter array elements : "<< endl;
    for(int k = 0; k < total_array_elements; k++)
    {
        cin >> array[k];            //store elements
    }

    //Store key (Or value which we want to find from Array)
    int key;
    cout <<endl<<"Enetr Key : " << endl;
    cin >> key;
```

```
    cout<<endl<<"Result : "<<find_key(array,total_array_elements,key);
    return 0;
}
```

So in linear searching method we are comparing "Key" with each element of array. Because of that this method is time and resource consuming.

Binary search:

Suppose You have answerkeys of 70 students, and you want to find anserkey of student whoes rollnumber is 35. you can optimize the process by selecting a subset of papers to check. Let's say you randomly pick an answer key belonging to roll number 30. This means that you only need to go 5 numbers ahead to find the answer key for roll number 35. By employing this approach, you avoid the need to compare roll number 35 with all 70 elements, and instead, narrow down the search by comparing it to a smaller set of numbers. This technique is based on the concept of the binary search algorithm.

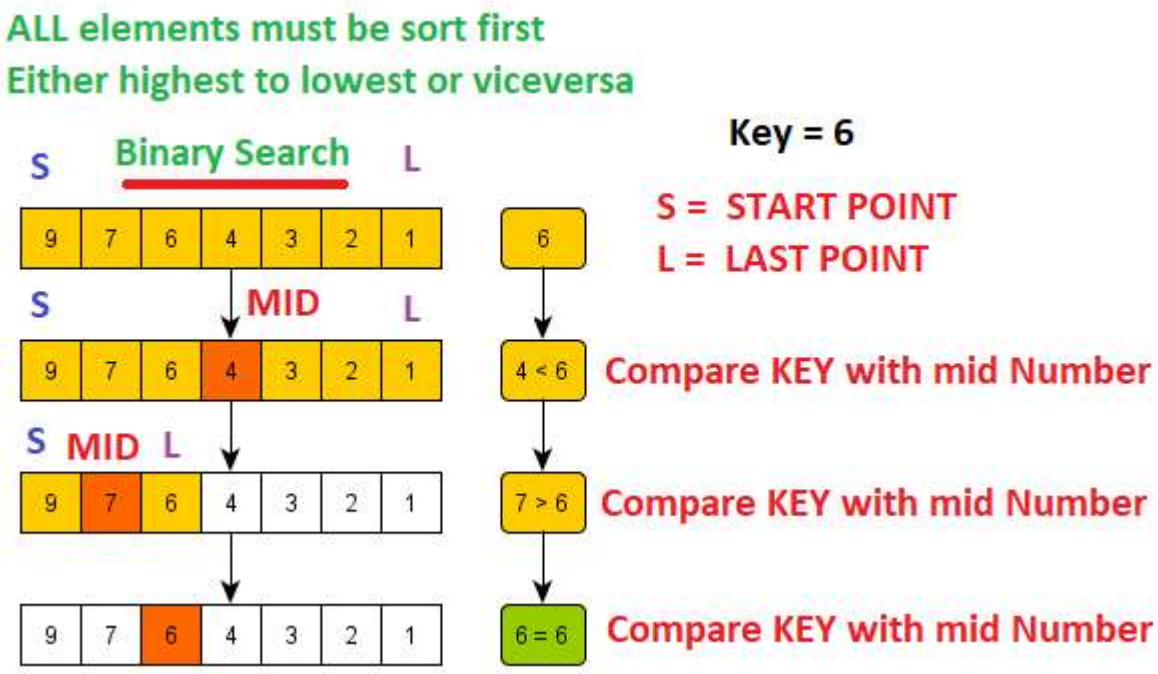
Algorithm flow:

1)All elements of the array must be sorted (either highest to lowest or vice versa)

2)Instead of picking random number, we will use middle point of the elements as a decision point. Initially start point is 0^{th} element and end_point is last element of array.

3)If middle pouint is less compare to key number, our new start point is middle_point + 1

4)If middle point is greater to key number, our stop_point is middle_point – 1.



Example:

```
//Binary search algorithm
//Here we have assumed that the array is already sorted in ascending order.
//Jaydeep Shah (radhey04ec@gmail.com)
```

```
#include<iostream>
using namespace std;
```

```
//Binary search algorithm function
/*
1)Find middle point of the Array,Initially start point = 0, end_point = size of array - 1
2)If middle poit is greater than key,  end point = middle_point - 1
3)If middle poit is less than key,  start point = middle_point + 1
4)If middle point == key, return middle point position
```

```
5)If no data found return -1
*/
int binary_search(int data[],int array_size,int key)
{
     int middle_pos = 0;
     int start_pos  = 0;
     int end_pos    = array_size - 1;

while(start_pos <= end_pos)     //OR (end_pos > start_pos)
{
     middle_pos = (start_pos + end_pos) / 2;

     if(data[middle_pos] == key)
     {
          return(middle_pos);
     }
     else if(data[middle_pos] > key)
     {
          end_pos = middle_pos - 1;
     }
     else
     {
          start_pos = middle_pos + 1;
     }
}
```

```
    //No data found
    return -1;
}

int main()
{

//User input for total elements of Array
int size_of_array;
cout << "How many elements you want to store inside array ? :"<< endl;
cin >> size_of_array;

//Store element of array
int ARRAY_DATA[size_of_array];         //Create Array for storing details of elements
cout << endl << "Please eneter elements (Ascending order) : "<< endl;
for(int  m = 0; m < size_of_array; m++)
{
    cin >> ARRAY_DATA[m];
}

//Key elemnt details
    int my_key = 0;
    cout << endl << "Please eneter key number : "<< endl;
    cin >> my_key;

    //Function call
    cout << endl << "Result (0th index system)= " <<
binary_search(ARRAY_DATA,size_of_array,my_key);
```

```
    return 0;
}
```

```
How many elements you want to store inside array ? :
7

Please eneter elements (Ascending order) :
12 23 26 37 48 69 90

Please eneter key number :
48

Result (0th index system)= 4
--------------------------------
```

Time complexity of this algorith:

First itteration : Array length = n

Second itteration : Arary length = n/2

Third itteration : Array length = ((n/2)/2) = n/4

kth itteration : Array length = (n / (2 ^k))

So, Array size reduced with each itteration.

Array sorting (Ascending / Descending):

Basic flow:

- Find minimum number and swap with first position of the number.

- With every itteration, first position will be increented by 1.

Example:

```
//Array sorting (Ascending / Descending) Algorithm
//Jaydeep Shah - radhey04ec@gmail.com
#include <iostream>
using namespace std;
//Function for sorting the Array
//-----------------------------------------
//Algorithm flow
//Initially start location = oth pos, Find minimum number from start location to last
element and swap it with start location
//Start location ++
//Repeat untill start location become your array size
//-----------------------------------------
void array_sort(int *data,int size)
{
     //First for loop : Start location update with each itteration
     for(int i = 0; i < size; i++)
     {
          //Find Minimum value from start to last location of array
          for(int j = i; j < size; j++)
          {
               //Compare with start location if value is minimum then swap
               if(data[j] < data[i])        //For Descending condition is data[i] < data[j]
```

```
{
     //Two varaible swapping
     int swap = data[i];
     data[i]  = data[j];
     data[j]  = swap;
}
          }
     }

}

int main()
{
     //Total elements inside Array ??
     int array_size = 0;
     cout << "Please provides number of elements inside array : " << endl;
     cin >> array_size;

     //Create Array of given size
     int our_array[array_size];
     cout << "Enter element details : " << endl;
     for(int k =0; k < array_size; k++)
     {
          cin >> our_array[k];
     }
```

```
 //Array sorting function call
 array_sort(our_array,array_size);

 //print data
 cout << endl << "After sorting : " << endl;
for(int k =0; k < array_size; k++)
     {
          cout << our_array[k] << " , ";
     }

     return 0;
}
```