



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Experiment No. 4
Binary Search Algorithm
Date of Performance:
Date of Submission:



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Experiment No. 4

Title: Binary Search Algorithm

Aim: To study and implement Binary Search Algorithm

Objective: To introduce Divide and Conquer based algorithms

Theory:

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty

- Binary search is efficient than linear search. For binary search, the array must be sorted, which is not required in case of linear search.
- It is divide and conquer based search technique.
- In each step the algorithms divides the list into two halves and check if the element to be searched is on upper or lower half the array
- If the element is found, algorithm returns.



Binary Search

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2 nd half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 > 56 take 1 st half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

The idea of binary search is to use the information that the array is sorted and reduce the time complexity to $O(\log n)$.

- ☐ Compare x with the middle element.
- ☐ If x matches with the middle element, we return the mid index.
- ☐ Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we recur for the right half.
- ☐ Else (x is smaller) recur for the left half.
- ☐ Binary Search reduces search space by half in every iterations. In a linear search, search space was reduced by one only.
- ☐ n =elements in the array
- ☐ Binary Search would hit the bottom very quickly.



	Linear Search	Binary Search
2 nd iteration	$n-1$	$n/2$
3 rd iteration	$n-2$	$n/4$

Example:



Algorithm $BINARY_SEARCH(A, key)$

// Description: Perform BS on array A

// I/P : array A of size n & key element to be searched.

// O/P : Success/failure.

$low \leftarrow 1$

$high \leftarrow n$

while $low < high$ do

$mid \leftarrow (low + high) / 2$

 if $A[mid] == key$ then

 return mid

 else if $A[mid] < key$ then

$low \leftarrow mid + 1$

 else

$high \leftarrow mid - 1$

end

end

return 0

$A = \{11, 22, 33, 44, 55, 66, 77, 88\}$

$key = 33$

$low = 1$

$high = 8$

$mid = (1+8)/2 = 4$

$A[4] == 33 \times$

$A[4] < 33 \times$

44

$high = 4 - 1$

$high = 3$

$\{11, 22, 33\}$

1 2 3

$low = 1$

$high = 3$

$mid = (1+3)/2 = 2$

$A[2] == 33 \times$

$22 < 33$

$low = 3$

$\{33\}$ $mid = (3+3)/2 = 3$

$A[3] = 33$

$A[mid] = 33$

$key = A[3]$

Algorithm and Complexity:



The binary search

- Algorithm 3: the binary search algorithm

Procedure binary search (x : integer, a_1, a_2, \dots, a_n : increasing integers)

$i := 1$ { i is left endpoint of search interval }

$j := n$ { j is right endpoint of search interval }

While $i < j$

begin

$m := \lfloor (i + j) / 2 \rfloor$

if $x > a_m$ **then** $i := m + 1$

else $j := m$



end

If $x = a_i$ **then** $location := i$

else $location := 0$

{ $location$ is the subscript of the term equal to x , or 0 if x is not found }

2

BINARY SEARCH			 Array  Divide and Conquer
Best	Average	Worst	
$O(1)$	$O(\log n)$	$O(\log n)$	

search (A, t) 1. $low = 0$ 2. $high = n - 1$ 3. while ($low \leq high$) do 4. $ix = (low + high) / 2$ 5. if ($t = A[ix]$) then 6. return true 7. else if ($t < A[ix]$) then 8. $high = ix - 1$ 9. else $low = ix + 1$ 10. return false end	search ($A, 11$)		
	low	ix	$high$
	first pass		
	1	4	8 9 11 15 17
	second pass		
	1	4	8 9 11 15 17
	third pass		
	1	4	8 9 11 15 17
	explored elements		



Best Case:

Key is first compared with the middle element of the array.

The key is in the middle position of the array, the algorithm does only one comparison, irrespective of the size of the array.

$$T(n)=1$$

Worst Case:

In each iteration search space of BS is reduced by half, Maximum $\log n$ (base 2) array divisions are possible.

Recurrence relation is

$$T(n)=T(n/2) + 1$$

Running Time is $O(\log n)$.

Average Case:

Key element neither is in the middle nor at the leaf level of the search tree. It does half of the $\log n$ (base 2).

Base case= $O(1)$

Average and worst case= $O(\log n)$

Code :

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
// Function to perform Binary Search recursively
```

```
int binarySearch(int arr[], int low, int high, int key) {
```

```
    if (low <= high) {
```

```
        int mid = low + (high - low) / 2;
```

```
        if (arr[mid] == key) {
```

```
            return mid;
```

```
        } else if (arr[mid] < key) {
```

```
            return binarySearch(arr, mid + 1, high, key);
```

```
        } else {
```

```
            return binarySearch(arr, low, mid - 1, key);
```

```
        }
```

```
    }
```

```
    return -1; // Key not found
```

```
}
```

```
// Function to generate a sorted array of 'n' numbers
void generateSortedArray(int arr[], int n) {
    srand(time(NULL));
    arr[0] = rand() % 10; // Generating the first element randomly
    for (int i = 1; i < n; i++) {
        arr[i] = arr[i - 1] + (rand() % 10); // Generating subsequent elements by adding a
        random number between 0 and 9
    }
}
```

```
int main() {
    int n, key;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Generating sorted array
    generateSortedArray(arr, n);

    // Displaying the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Prompting the user to enter the key to search
    printf("Enter the key to search: ");
    scanf("%d", &key);

    // Performing binary search
    clock_t start = clock();
    int index = binarySearch(arr, 0, n - 1, key);
    clock_t end = clock();
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
```



```

// Displaying the result of binary search
if (index != -1) {
    printf("Key %d found at index %d.\n", key, index);
} else {
    printf("Key %d not found.\n", key);
}

// Displaying time taken for binary search
printf("Time taken for binary search: %f seconds\n", time_taken);

return 0;
}

```

Output :

```

PS E:\Testing_Lang> cd "e:\Testing_Lang\" ; if ($?) { gcc test.c -o test } ; if ($?) { .\test }
Enter the number of elements: 4
Sorted array: 1 10 13 19
Enter the key to search: 

```

Conclusion:

This program generates a sorted array of 'n' numbers and performs a binary search for a key entered by the user. It measures the time taken for the binary search operation and displays the result along with the time taken.

To perform an analysis for different values of 'n', you can modify the program to execute multiple times with increasing values of 'n' and measure the time taken for each execution. Additionally, you can compare the performance of the binary search algorithm with other searching algorithms to analyze their efficiency for different input sizes.

