

CS -08

DATA STRUCTURE USING
C LANGUAGE

By Rachel

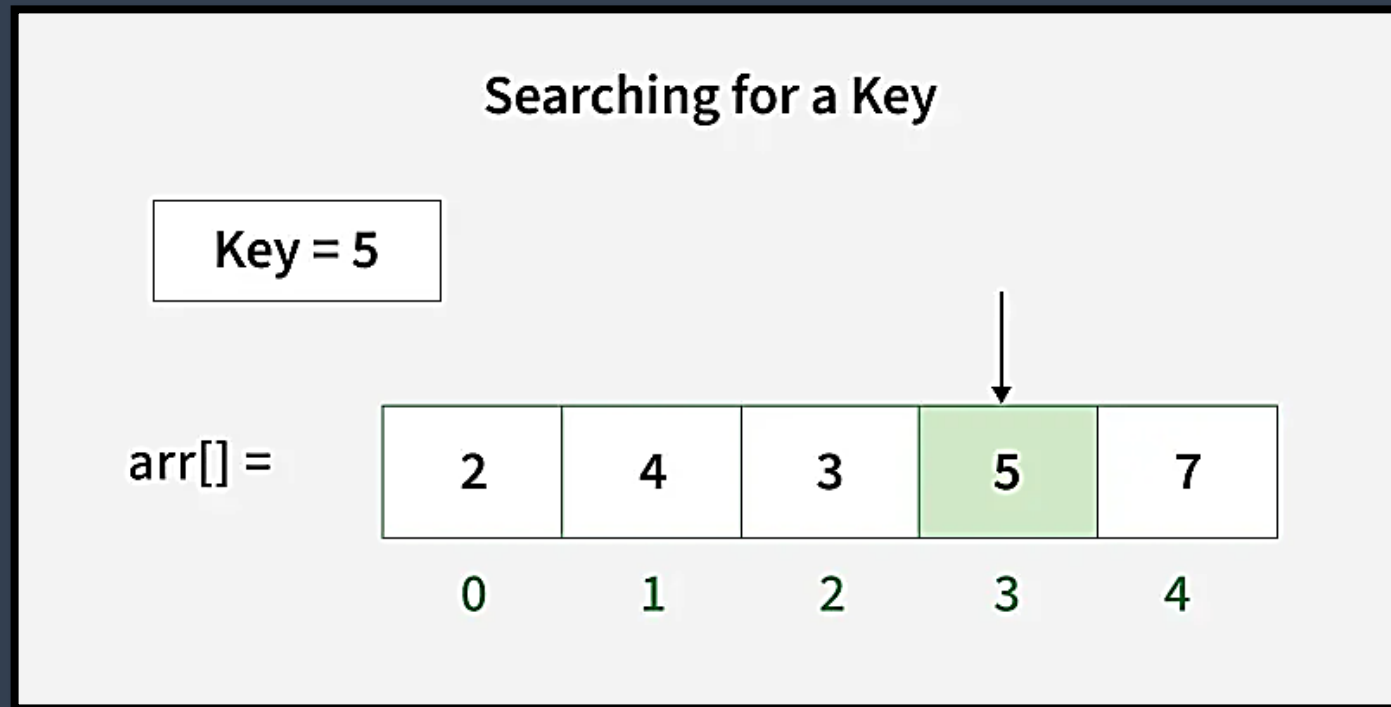
UNIT 2

SORTING AND SEARCHING

BASIC SEARCHING TECHNIQUES

What is Searching?

- Searching is the fundamental process of locating a specific element or item within a collection of data.
- This collection of data can take various forms, such as arrays, lists, trees, or other structured representations.

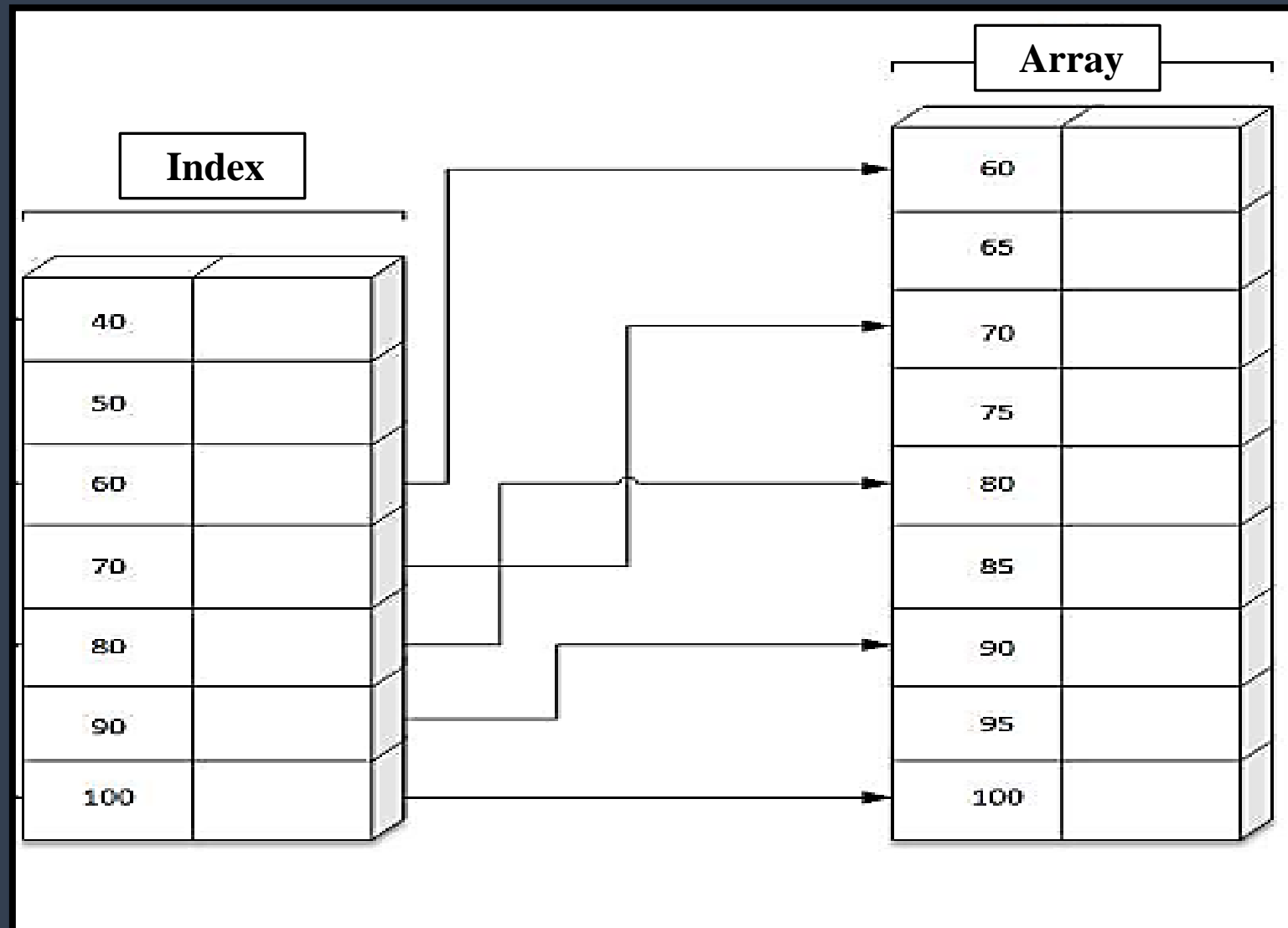


TOPIC 8

INDEX SEARCHING

What is Index Searching?

- Index Searching is a data retrieval method for sorted data that speeds up searches by using an index to jump to probable data blocks, then performing a linear (sequential) search within that smaller block.
- It is ideal for large datasets but requires extra memory for the index.
- It works by creating an index of starting points, finding the right block using the index, and then scanning just that block to find the item, making it faster than linear search but slower than binary search.



Program

```
// C program to implement Index Searching
#include <stdio.h>

int IndexSearch(int arr[], int n, int k)
{
    int elements[50], indices[50], i;
    int j = 0, ind = 0, start, end;
    for (i = 0; i < n; i += 3)
    {
        // Storing element
        elements[ind] = arr[i];
        // Storing the index
        indices[ind] = i;
        ind++;
    }
}
```

```
if (k < elements[0])
{
    printf("Not found");
    exit(0);
}
else
{
    for (i = 1; i <= ind; i++)
        if (k <= elements[i])
        {
            start = indices[i - 1];
            end = indices[i];
            break;
        }
}
```

```
for (i = start; i <= end; i++)
{
    if (k == arr[i])
    {
        j = 1;
        break;
    }
}
if (j == 1)
    printf("Found at index %d", i);
else
    printf("Not found");
}
```

```
int main()
{

    int arr[] = { 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Element to search
    int k = 14;
    IndexSearch(arr, n, k);
}
```

Output

Found at index 8

Advantages of Index Searching

- More efficient.
- Time required is less.

Disadvantages of Index Searching

- Increased storage space
- Overhead of maintenance.

TOPIC 9

SEQUENTIAL SEARCHING

What is Sequential Searching?

- In Sequential Search, we iterate over all the elements of the array and check if the current element is equal to the target element.
 - If we find any element to be equal to the target element, then return the index of the current element.
 - Otherwise, if no element is equal to the target element, then return -1 as the element is not found.
- sequential search is also known as linear search.

01

Step

Compare the key with each element one by one starting from the 1st element.

Key

30

Not Equal

0

1

2

3

4

5

6

7

8

10

50

30

70

80

60

20

90

40

Cur

02

Step

Compare the key with 2nd element which is not equal to the key, so move to the next element

Key

30

Not Equal

0	1	2	3	4	5	6	7	8
10	50	30	70	80	60	20	90	40

Cur

03

Step

Compare the key with the 3rd element. Key is found so stop the search.

Key
30
Equal

0	1	2	3	4	5	6	7	8
10	50	30	70	80	60	20	90	40

Cur

Program

```
//C program to implement Sequential Searching
#include <stdio.h>

int SequentialSearch(int arr[], int n, int x)
{
    int i;
    for(i=0;i<n;i++)
    {
        if(arr[i] == x)
        {
            return i;
        }
    }
    return -1;
}
```

```
int main()
{
    int arr[] = {2,3,4,10,40};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\nGiven Array :\n");
    int i;
    for (i=0;i<n;i++)
    {
        printf("%d ",arr[i]);
    }
    int x;
    printf("\nEnter the number to be searched in the array :");
    scanf("%d",&x);
    int result = SequentialSearch(arr,n,x);
    if(result == -1)
        printf("\nThe number %d is not found in the array",x);
    else
        printf("\nThe number %d is present at Index %d",x,result);
    return 0;
}
```

Output

```
Given Array :
```

```
2 3 4 10 40
```

```
Enter the number to be searched in the array :10
```

```
The number 10 is present at Index 3
```

Advantages of Sequential Searching

- Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Disadvantages of Sequential Searching

- Linear search has a time complexity of $O(n)$, which in turn makes it slow for large datasets.
- Not suitable for large arrays.

TOPIC 10

BINARY SEARCHING

What is Binary Searching?

Binary Search is a searching algorithm that operates on a sorted search space, repeatedly dividing it into halves to find a target value or optimal answer in logarithmic time $O(\log n)$.

Step-by-step algorithm for Binary Search:

- Divide the search space into two halves by finding the middle index "mid".
- Compare the middle element of the search space with the key.
- If the key is found at middle element, the process is terminated.
- If the key is not found at middle element, choose which half will be used as the next search space.
- If the key is smaller than the middle element, then the left side is used for next search.
- If the key is larger than the middle element, then the right side is used for next search.
- This process is continued until the key is found or the total search space is exhausted.

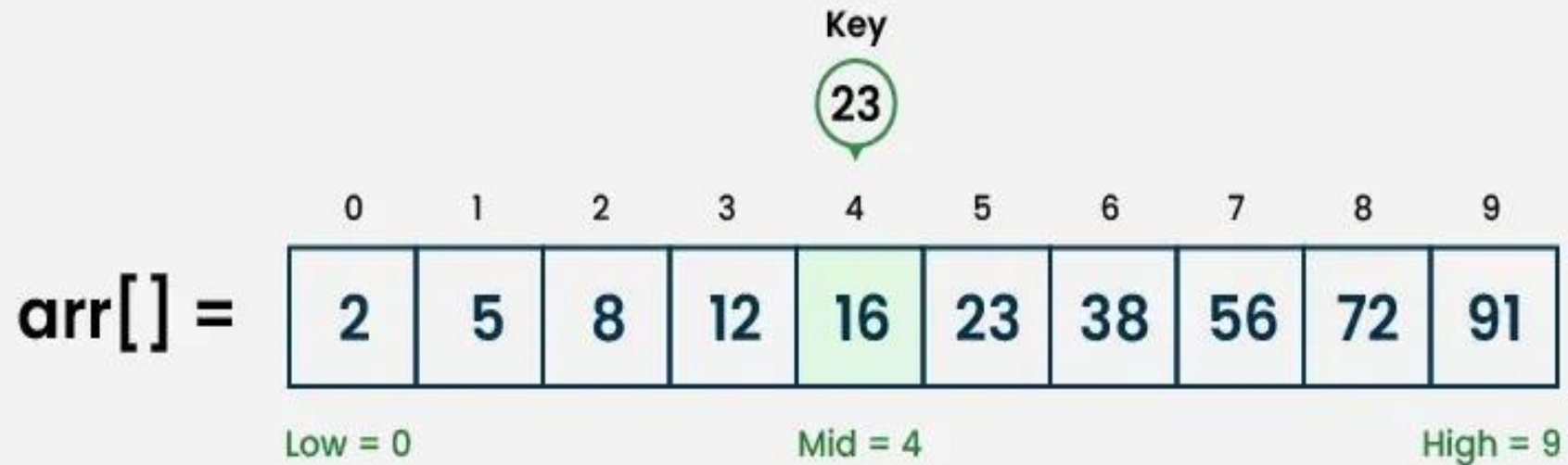
Initially

Find Key = 23 using Binary Search

	0	1	2	3	4	5	6	7	8	9
arr[] =	2	5	8	12	16	23	38	56	72	91

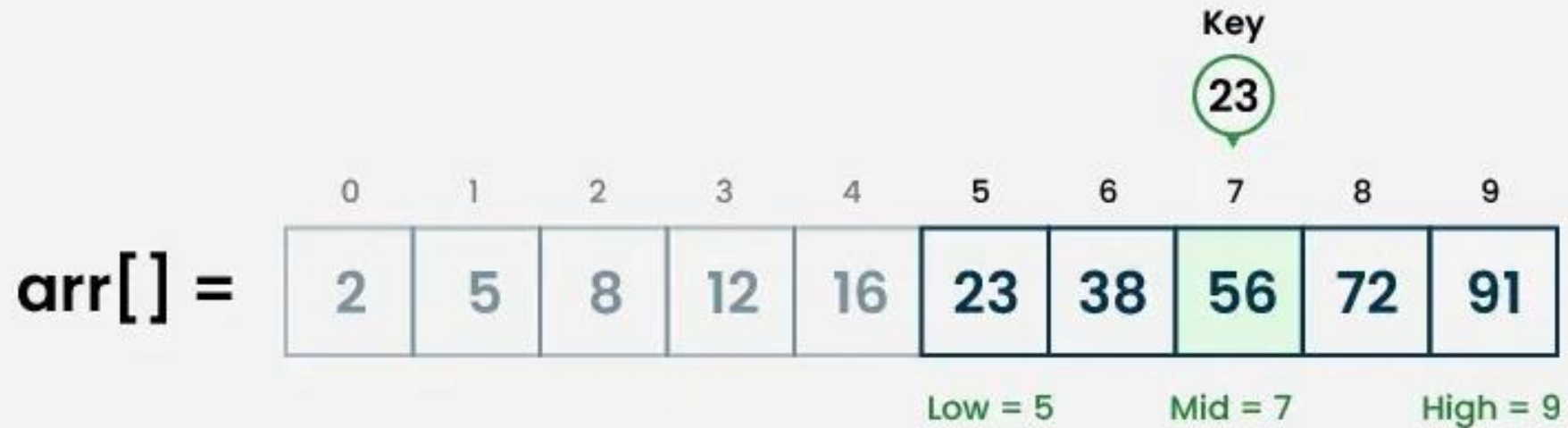
Step 1

Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.



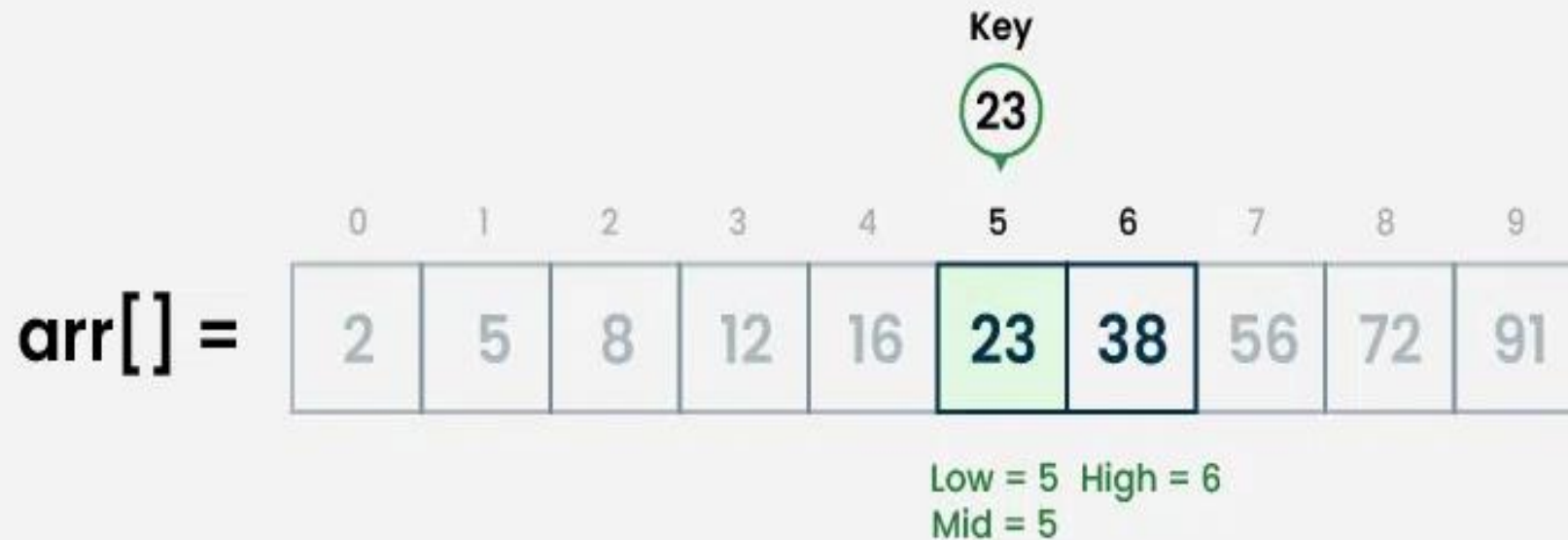
Step 2

Key is less than the current mid 56.
The search space moves to the left.



Step 3

If the key matches the value of the mid element, the element is found and stop search.



Program

```
//C Program to implement Binary Searching
#include <stdio.h>

int BinarySearch(int arr[],int n,int x)
{
    int low = 0;
    int high = n-1;
    while (low <= high)
    {
        int mid = low+(high-low)/2;
        // check if x is present at mid
        if(arr[mid]==x)
            return mid;

        //if x is greater than mid, then ignore left half
        if(arr[mid]<x)
            low=mid+1;

        //if x is smaller than mid, then ignore right half
        else
            high=mid-1;
    }
    return -1;
}
```

```
int main()
{
    int arr[] = {2,3,4,10,40};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("\nGiven Array :\n");
    int i;
    for (i=0;i<n;i++)
    {
        printf("%d ",arr[i]);
    }
    int x;
    printf("\nEnter the number to be searched in the array :");
    scanf("%d",&x);
    int result = BinarySearch(arr,n,x);
    if(result == -1)
        printf("\nThe number %d is not found in the array",x);
    else
        printf("\nThe number %d is present at Index %d",x,result);
    return 0;
}
```

Output

```
Given Array :
```

```
2 3 4 10 40
```

```
Enter the number to be searched in the array :40
```

```
The number 40 is present at Index 4
```

Advantages of Binary Searching

- Efficiency in Time Complexity: Binary search has a time complexity of $O(\log n)$, where n is the number of elements in the array.
- Optimal for Sorted Data: Binary search is specifically designed for sorted data.
- Low Space Complexity: Binary search has a space complexity of $O(1)$ when implemented iteratively. It does not require additional memory proportional to the input size, making it memory-efficient.

Disadvantages of Binary Searching

- Requires Sorted Data: Binary search only works on sorted arrays or lists.
- Inefficient for Small Datasets: For small datasets, binary search may not be significantly faster than a linear search.