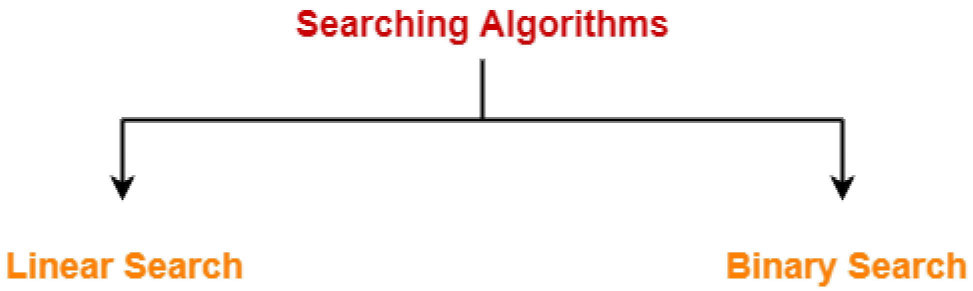
Searching

Searching-

- Searching is a process of finding a particular element among several given elements.
- The search is successful if the required element is found.
- Otherwise, the search is unsuccessful.

Searching Algorithms-

The searching of an element in the given array may be carried out in the following two ways-



1. Linear Search
2. Binary Search

Linear Search:

- Linear Search is the simplest searching algorithm.
- It traverses the array sequentially to locate the required element.
- It searches for an element by comparing it with each element of the array one by one.
- So, it is also called as **Sequential Search**.

Linear Search Algorithm is applied when-

- No information is given about the array.
- The given array is unsorted or the elements are unordered.
- The list of data items is smaller.

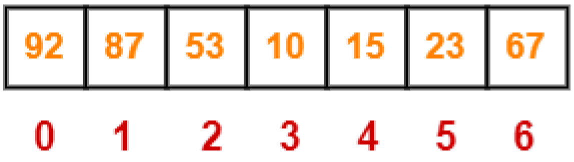
A simple approach to implement a linear search is

- Begin with the leftmost element of arr[] and one by one compare x with each element.
- If x matches with an element then return the index.
- If x does not match with any of the elements then return -1.

Linear Search Example-

Consider-

- We are given the following linear array.
- Element 15 has to be searched in it using Linear Search Algorithm.



92	87	53	10	15	23	67
0	1	2	3	4	5	6

LInear Search Example

Now,

- Linear Search algorithm compares element 15 with all the elements of the array one by one.
- It continues searching until either the element 15 is found or all the elements are searched.

Linear Search Algorithm works in the following steps-

Step-01:

- It compares element 15 with the 1st element 92.
- Since 15 ≠ 92, so required element is not found.
- So, it moves to the next element.

Step-02:

- It compares element 15 with the 2nd element 87.
- Since 15 ≠ 87, so required element is not found.
- So, it moves to the next element.

Step-03:

- It compares element 15 with the 3rd element 53.
- Since 15 ≠ 53, so required element is not found.

- So, it moves to the next element.

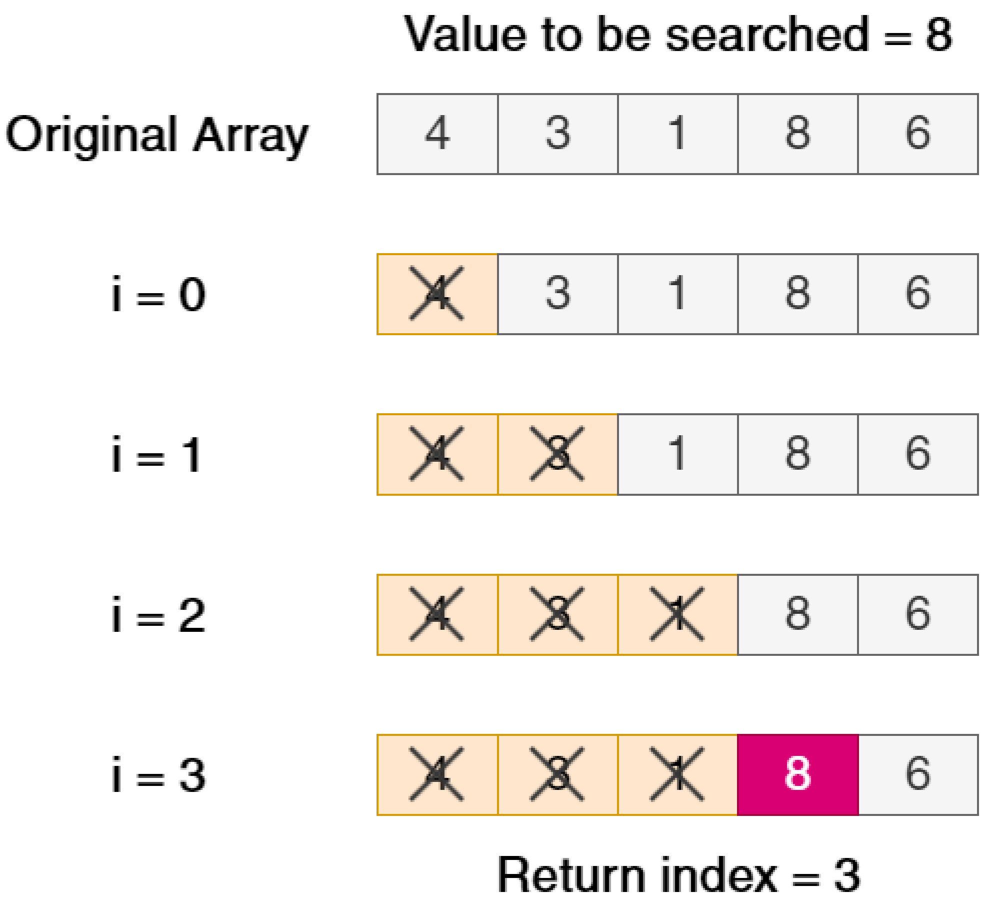
Step-04:

- It compares element 15 with the 4th element 10.
- Since 15 ≠ 10, so required element is not found.
- So, it moves to the next element.

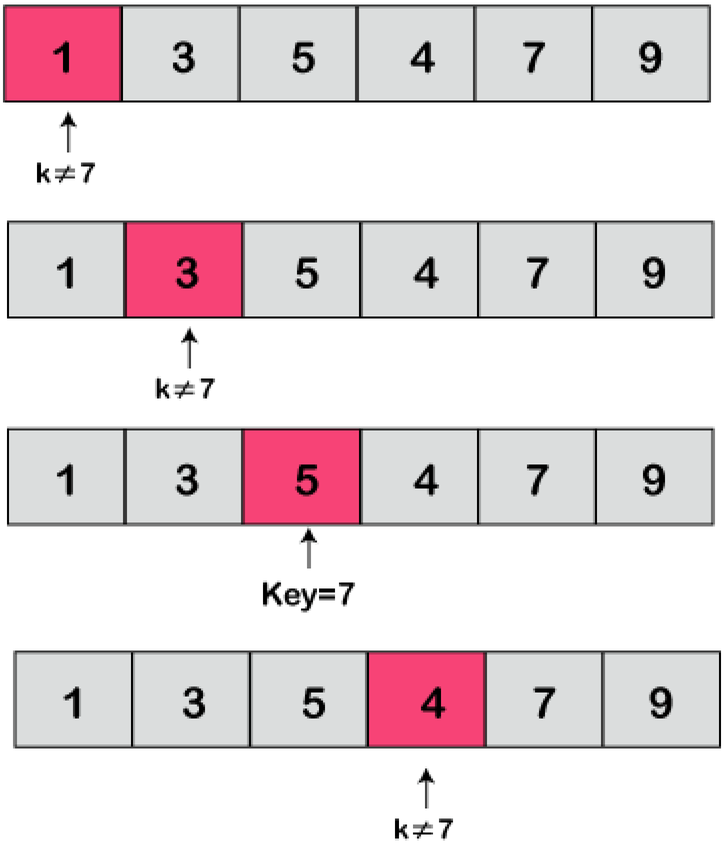
Step-05:

- It compares element 15 with the 5th element 15.
- Since 15 = 15, so required element is found.
- Now, it stops the comparison and returns index 4 at which element 15 is present.

Example2:



Example3:



Time Complexity Analysis-

Linear Search time complexity analysis is done below-

Best case-

In the best possible case,

- The element being searched may be found at the first position.
- In this case, the search terminates in success with just one comparison.
- Thus in best case, linear search algorithm takes O(1) operations.

Worst Case-

In the worst possible case,

- The element being searched may be present at the last position or not present in the array at all.
- In the former case, the search terminates in success with n comparisons.
- In the later case, the search terminates in failure with n comparisons.
- Thus in worst case, linear search algorithm takes O(n) operations.

Thus, we have-

Time Complexity of Linear Search Algorithm is O(n).

Here, n is the number of elements in the linear array.

Linear Search Efficiency-

- Linear Search is less efficient when compared with other algorithms like Binary Search & Hash tables.
- The other algorithms allow significantly faster searching.

Implementing Linear Search in C

```c
#include<stdio.h>


int main()

{

    int a[20],i,x,n;

    printf("How many elements?");

    scanf("%d",&n);


    printf("Enter array elements:n");

    for(i=0;i<n;++i)

        scanf("%d",&a[i]);


    printf("nEnter element to search:");

    scanf("%d",&x);


    for(i=0;i<n;++i)

        if(a[i]==x)

            break;


    if(i<n)

        printf("Element found at index %d",i);

    else

        printf("Element not found");


    return 0;

}
```

Binary Search-

- Binary Search is one of the fastest searching algorithms.
- It is used for finding the location of an element in a linear array.
- It works on the principle of divide and conquer technique.

Binary Search Algorithm can be applied only on **Sorted arrays**.

So, the elements must be arranged in-

- Either ascending order if the elements are numbers.
- Or dictionary order if the elements are strings.

To apply binary search on an unsorted array,

- First, sort the array using some sorting technique.
- Then, use binary search algorithm.

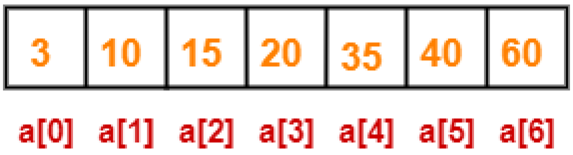
Binary search algorithm:

```
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
```

Binary Search Example-

Consider-

- We are given the following sorted linear array.
- Element 15 has to be searched in it using Binary Search Algorithm.



3	10	15	20	35	40	60
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

Binary Search Example

Binary Search Algorithm works in the following steps-

Step-01:

- To begin with, we take beg=0 and end=6.
- We compute location of the middle element as-

$$\begin{aligned} mid &= (beg + end) / 2 \\ &= (0 + 6) / 2 \\ &= 3 \end{aligned}$$

- Here, a[mid] = a[3] = 20 ≠ 15 and beg < end.
- So, we start next iteration.

Step-02:

- Since a[mid] = 20 > 15, so we take end = mid – 1 = 3 – 1 = 2 whereas beg remains unchanged.
- We compute location of the middle element as-

$$\begin{aligned} mid &= (beg + end) / 2 \\ &= (0 + 2) / 2 \\ &= 1 \end{aligned}$$

- Here, a[mid] = a[1] = 10 ≠ 15 and beg < end.

- So, we start next iteration.

Step-03:

- Since a[mid] = 10 < 15, so we take beg = mid + 1 = 1 + 1 = 2 whereas end remains unchanged.
- We compute location of the middle element as-

mid

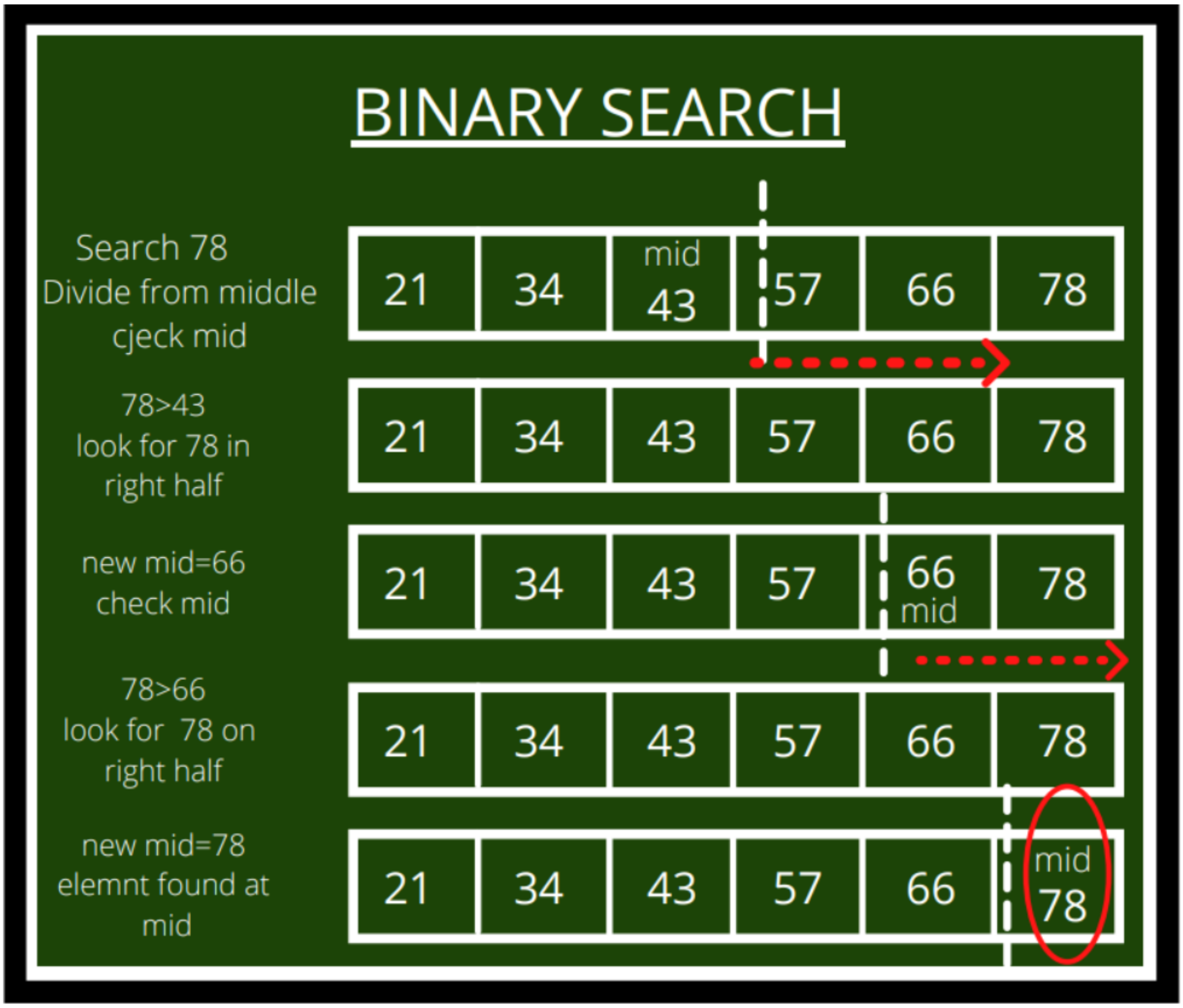
= (beg + end) / 2

= (2 + 2) / 2

= 2

- Here, a[mid] = a[2] = 15 which matches to the element being searched.
- So, our search terminates in success and index 2 is returned.

Example2:



Time Complexity Analysis-

Binary Search time complexity analysis is done below-

- In each iteration or in each recursive call, the search gets reduced to half of the array.
- So for n elements in the array, there are $\log_2 n$ iterations or recursive calls.

Thus, we have-

Time Complexity of Binary Search Algorithm is $O(\log_2 n)$.

Here, n is the number of elements in the sorted linear array.

Binary Search Algorithm Advantages-

The advantages of binary search algorithm are-

- It eliminates half of the list from further searching by using the result of each comparison.
- It indicates whether the element being searched is before or after the current position in the list.
- This information is used to narrow the search.
- For large lists of data, it works significantly better than linear search.

Binary Search Algorithm Disadvantages-

The disadvantages of binary search algorithm are-

- It employs recursive approach which requires more stack space.
- Programming binary search algorithm is error prone and difficult.
- The interaction of binary search with memory hierarchy i.e. caching is poor. (because of its random access nature)

Implementation:

```c
// Binary Search in C

#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
  // Repeat until the pointers low and high meet each other
  while (low <= high) {
    int mid = low + (high - low) / 2;

    if (array[mid] == x)
      return mid;

    if (array[mid] < x)
      low = mid + 1;

    else
      high = mid - 1;
  }

  return -1;
}

int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
  return 0;
}
```