SEARCHING

Searching :-

- Searching is a process of finding a given value position in a list of values.
- It decides whether a search key is present in the data or not.
- It is a algorithmic process of finding particular item in a collection of items.

Searching Techniques :-

To search an element in given array, it can be done in following ways:

(1) Sequential Search | Linear Search

(2) Interval Search | Binary Search.

SEQUENTIAL SEARCH

- Sequential search starts at the beginning of the list and checks every element of the list.
- It is very simple and basic search algorithm
- Sequential search compares the element with all the other elements given in the list. If the element is matched, it returns the value index, else it returns -1.

Ex :-

0	1	2	3	4	5	6	7
15	5	20	32	2	42	67	17

$n = 8$

data = 42

There will be 2 cases :-

(1) Element is present in the list

You want to print the location or index

(2) Element not present.

You want to print element not present.

	0	1	2	3	4	5	6	7
a:	15	5	20	35	2	42	67	17

$\boxed{42}$

→ print the index - 5

STOPPING CONDITION → You find an data in array.

	0	1	2	3	4	5	6	7
a:	15	5	20	35	2	42	67	17

$\boxed{41}$

STOPPING CONDITION : → you reach end of the array, data is not present in array.

I When element is present in array.

```
for(i=0; i<n; i++)
{
    if(a[i]==data)
    {
        printf("Element found at index :%d", i);
        printf("Element found at location : %d", i+1);
    }
}
```

i=0   0<7 $i = \cancel{0}, \cancel{1}, \cancel{2}, \cancel{3}, \cancel{4}, 5, 6, 7$

a[0] == 42 ✗

i=5   5<7   a[5]==42 ⟶ printf.

DRAWBACK

Even we have found element at index 5 still it will search for 6th and 7th index,
So what it is the point for searching the data after 5th index, so you have to add one more condition in the code.

```
for (i=0; i<n; i++)
  {
      if (a[i] == data)
      {
         printf("Element found at index: %d", i);
         break;
      }
  }
```

II When Element is not present in array.

```
for (i=0; i<n; i++)
  {
      if (a[i] == data)
      {
      printf("Element found at index: %d", i);
         break;
      }
  }

  if (i == n)
  {
  printf("Element not found");
  }
```

TIME COMPLEXITY

Best Case :- $O(1)$
Worst case :- $O(n)$

Average Case :- $\frac{\Sigma \text{ all cases}}{\text{no of cases}}$

$$\frac{1+2+3+\cdots+n}{n}$$

$$\frac{n(n+1)}{2 \cdot \cancel{n}}$$

$$= O\left(\frac{n+1}{2}\right)$$

BINARY SEARCH

- These algorithms are specifically designed for searching in sorted data.
- These types of searching are much more efficient than linear search as they repeatedly target the center of the search structure and divide the search space in half.

$\boxed{n=10}$

	0	1	2	3	4	5	6	7	8	9
a :	5	9	17	23	25	45	59	63	71	89

→ Array should be sorted if you want to apply binary search but in linear search this is not the condition, you can have either sorted or unsorted list.

⇒ Divide and conquer technique - means divide the array into two halfs → recursively divide the array

Now, we will see how it recursively divide the array, ie we are going to find the middle value

	0	1	2	3	4	5	6	7	8	9
a	5	9	17	23	25	45	59	63	71	89
	l				mid					r

n = 10

data = 59

l	r	mid
0	9	$(l+r)/2$ $\frac{0+9}{2} = 4.5 = 4$

3 CASES :-

CASE I :- data == a[mid]

CASE II :- data < a[mid]

CASE III :- data > a[mid]

data = 59

59 > 25

CASE III

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
				mid	l				r

l	r	mid
0	9	4
5	9	$\frac{5+9}{2} = 14/2 = 6+1 = 7$

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					l	r	mid		~~r~~

l	r	mid
0	9	4
5	9	7
5	6	$\frac{5+6}{2} = 11/2 = 5$

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
					l mid				

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
						mid l r			~~r~~

l	r	mid
6	6	$\frac{6+6}{2} = 6$

data == a[mid]

I Stopping condition

II If element is not present

data = 60

l	r	mid
0	9	4
5	9	7
5	6	5
6	6	6
X → 7	6	

0	1	2	3	4	5	6	7	8	9
5	9	17	23	25	45	59	63	71	89
l				mid	mid l r	r	mid		r

if (l > r) → data is not present.

```
int BinarySearch(a, n, data)
{
    l=0 , r=n-1;
while(l<=r)
{   mid = (l+r)/2;
    if (data == a[mid])
        return mid;
    elseif (data < a[mid])
        r = mid-1;
    else
        l = mid+1;
}
return -1; }
```

TIME COMPLEXITY.

WORST CASE :- $O(\log n)$

BEST CASE :- $O(1)$