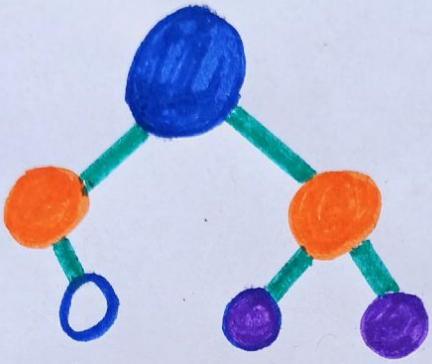
SEARCHING IN DATA STRUCTURE



Data Structure

ATUL KUMAR (LINKEDIN).
NOTES GALLERY (TELEGRAM).

What is Searching?

- Searching is the 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 the algorithmic process of finding a particular item in a collection of items.
- It can be done on internal data structure or on external data structure.

Searching Techniques:

1. Sequential Search / Linear Search
2. Binary Search

Sequential search –

→ Sequential search is also called as Linear search.
→ Sequential search starts at the beginning of the first and checks every element of the list.
→ It is a basic and simple 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.

Algorithm –

LSEARCH (ARR, N, ITEM, LOC) Here ARR is the array of numbers of elements, item holds the value we need to search in the array and algorithm return LOC, the location where ITEM is present in the ARR. Intially we have to set LOC = -1

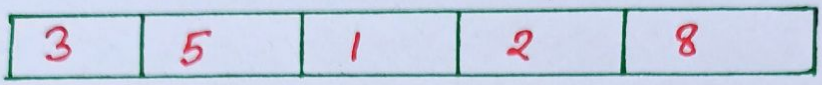
1. Set LOC = -1, i = 1
2. Repeat while DATA[i] != ITEM i = i+1
3. if i = N+1, then set LOC = 0 Else LOC = N+1
4. Exit

Time complexity –

Case	Time complexity
Best case	O(1)
Average case	O(n)
Worst case	O(n)

How linear search works.

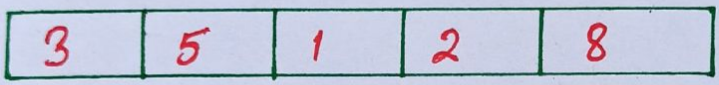
For an element K = 1 in the list below.



Array to be searched for

1. Start from the first element, compare K with each element X.

K = 1



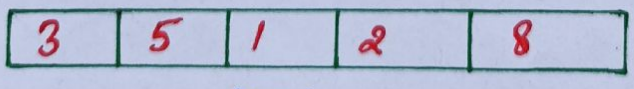
K ≠ 3

K ≠ 5

K = 1

Just like this compare with each and every element.

2. If X == K, return the index.



K = 1

Element found.

Continue ⟶

3. Else, return not found.

- Program :

```
#include <stdio.h>
int main( )
{
   int arr[50], search, c, n;

   Printf("Enter number of elements in array \n");
   scanf("%d", &n );
   Printf("Enter %d integer \n", n );

   for ( c=0; c<n; c++ )
    {
      if (array[c] == search)
      {
         printf("%d is present at location %d .\n", search,
                                                   c+1);
      }
    }
     if ( c ==n )
     printf("%d isn't present in array In" search);

     return 0;
}
```


- Binary Search

→ Binary search is used for searching an element.
→ It is a fast search algorithm with run-time complexity of $O(\log n)$
→ In works on divide and conquer rule.

Algorithm

1. Set BEG = LB , END = UB and MID = INT [(BEG + END)/2].
2. Repeat step 3 and 4 while BEG <= END ARR [MID] != ITEM.
3. IF ITEM < ARR [MID] then :
   Set END = MID - 1
   Else :
   SET BEG = MID + 1
4. SET MID = INT (BEG + END) /2
5. IF ARR [MID] = ITEM then;
   SET LOC = MID
   else
   Set LOC = NULL
6. Exit.

ATUL KUMAR (LINKEDIN).
NOTES GALLERY (TELEGRAM).

Time Complexity.

Case	Time complexity
Worst case	O(nlogn)
Best case	O(1)
Average case	O(nlogn).

HOW Binary search works.

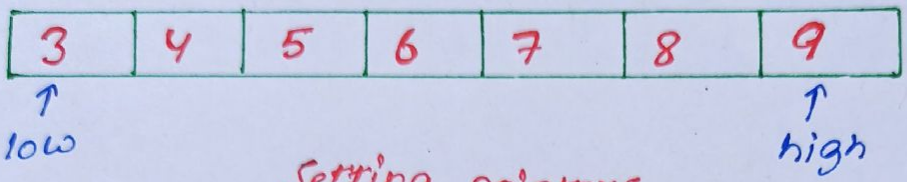
1. The array in which searching is to be performed is

3	4	5	6	7	8	9

Initial Array.

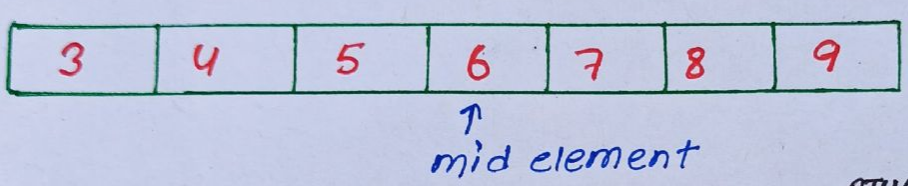
Let X = 4 be the element to be searched.

2. Set two pointers low and high at the lowest and highest positions respectively.



Setting pointers

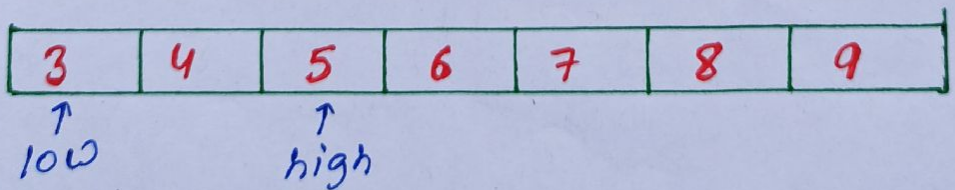
3. Find the middle element mid of the array ie. arr [low + high] / 2] = 6



4. If x == mid, then return mid. else compare element to be search with m.

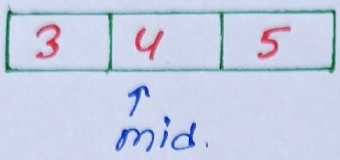
5. If x > mid, compare x with the middle element on the right side of mid. This is done by low = mid + 1.

6. Else, compare x with middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.



Find the mid element.

7. Repeat step 3 to 6 until low meets high.



8. $x = 4$ is found.

Program.

ATUL KUMAR (LINKEDIN).
NOTES GALLERY (TELEGRAM).

```
# include <stdio.h>
int binarysearch(int array[], int x, int low, int high]
 {
   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, 8, 10 };
   int n = size of(array) / size of(array[0])
    int x = 4;
  int result = binarysearch(array, x, 0, n-1).
```