SEARCHING.

① Binary Search.

o) We already know linear Search.

a) → I/p: arr[] = [10, 20, 30, 40, 50]

$x = 20$

O/p: 1.

b) → Normally we can use linear Search, but it takes $O(n)$ time in worst case (element not there in array).

c) → Corner cases in Binary Search.

→ I/p: arr[] = {10, 15}
$x = 20$
O/p: −1

I/p: arr[] = {10, 10}
O/p: 0 (or) 1

d) → Idea of binary search is to use the fact that array is sorted. and cut the array by half every time

Code:

```
int BSearch (int arr[], int n, int x)
{ int low = 0, high = n-1;
  while( low <= high)
  {
      int mid = (low+high)/2;
      if (arr[mid] == x)
          { return mid; }
      else if (arr[mid] > x)
          { high = mid-1; }
      else { low = mid+1; }
  }
  return -1;
}
```

Note: 2 elem
$(0+1/2) = 0$
is midpoint.

(2) Recursive Code for Binary Search.

(1) →

```
int bSearch (int arr[], int low, int high, int x)
{
    if (low > high) return -1;

    int mid = (low + high)/2 ;

    if (arr[mid] == x){return mid;}

    else if (arr[mid] > x)
      { return bSearch(arr, low, mid-1, x); }

    else
      { return bSearch (arr, mid+1, high, x); }

}
```

---

(3) Analysis of Binary Search

→ In general iterative and recursive req $O(\log N)$ time.

recursive also req → $O(\log N)$ extra space.

→ till 16 ele → 4 iterations. → loop is divided by constant amount i.e 2

17 to 32 ele → 5 iterations.

33 to 64 ele → 6 iterations.
