Binary Search

Let a_i, $1 \le i \le n$, be a list of elements that are sorted in non decreasing (increasing) order.

→ Determine whether a given element x is present in the list. If x is present then determine a value j such that $a_j == x$.

Search Problem $P = (n, a_i, a_{i+1}, \ldots a_l, x$

$a_i \downarrow a_1$, $a_l \downarrow a_n$

small(P) is true if $n=1$, in this case

S(P) will take value i if $x = a_i$

otherwise (if $x \neq a_i$) it will take the value 0

→ If P has more than one element, it can be divided into a new subproblem as follows.

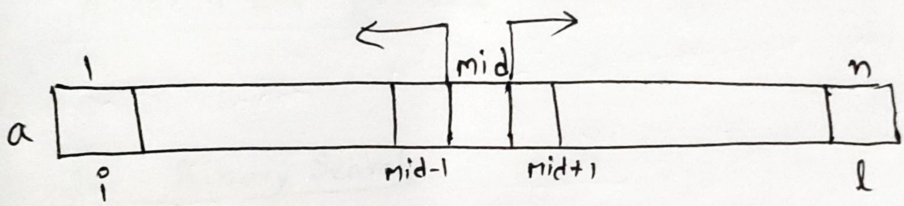
Pick an index q and compare x with a_q three possibilities

1) If $x == a_q$, P is immediately solved.
2) If $x < a_q$, search for x in the left subarray $a_i, a_{i+1}, \ldots a_{q-1}$
3) if $x > a_q$, search for x in the right subarray $a_{q+1}, a_{q+q} \ldots a_l$.

Division of array into 2 subarrays takes only O(1) time.

→ If q is always choosen such that a_q is the middle element that is $q = \lfloor (n+1)/2 \rfloor$

Then the resulting algorithm is known as binary search. There is no need to combine the solutions.



Algorithm BinSrch (a, i, l, x)

// Given an array $a[i:l]$ of elements in
// non decreasing order; $1 \le i \le l$, determine
// whether x is present, and if so, return
// array index j such that $x = a[j]$;
// else return 0

if $(l = i)$ then // If small (P)

{

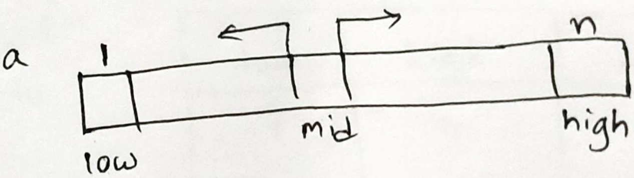
  if $(x = a[i])$ then return i;

  else return 0; // element x not found

}

```
else
{
    //Reduce P into a smaller subproblem
    mid = ⌊(i + l)/2⌋;
    if (x = a[mid]) then return mid;
    else if(x < a[mid]) then
    return BinSrch(a, i, mid-1, x); // search will
    //proceed in the left subarray
    else return BinSrch(a, mid+1, l, x);
    //search will proceed in the right subarray
  }
}
```

Recursive Binary Search




```
Algorithm BinSearch(a, n, x)
// a[1:n]  , n ≥ 0
{
    low := 1 ; high := n;
    while ( low ≤ high) do
    {
        mid := ⌊(low + high)/2⌋ ;
        if (x < a[mid]) then high := mid-1;
        else if (x > a[mid]) Then high := mid-1;
        else return mid;
    }
    return 0; // element x not found in array a
}
```

Iterative Binary Search:

Search for a given element x in the following array.

$a[1:14]$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-6	0	7	9	23	54	82	101	112	125	131	142	151

- Search for $x = 9$

low	high	$mid = \left\lfloor \frac{low+high}{2} \right\rfloor$
1	14	7 — $9 < a[7] = 54$
1	6	3 — $9 > a[3] = 0$
4	6	5 — $9 = a[5]$

Given element $x = 9$ is found at index 5.

- search for $x = -14$

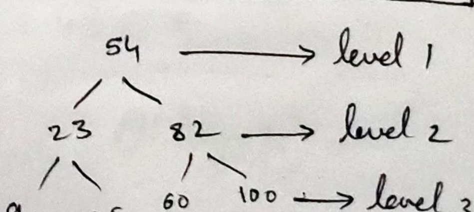
low	high	mid	
1	4	7	$-14 < a[7] = 54$
1	6	3	$-14 < a[3] = 0$
1	2	1	$-14 > a[1] = -15$
2	2	2	$-14 \neq a[2] = 6$

Given element $x = -14$ is not found in the array

Time complexity

	1	2	3	4	5	6	7
a	9	23	25	54	60	82	100

Binary search tree



No. of comparisons required in Binary search = level of that element in its Binary search Tree

(1) **Best case time complexity:-**

If given element $x = 54$ is matching with middle element [root] of the array then no. of comparisons required $= 1$.

$\therefore$ Time complexity $= O(1)$.

(2) **Worst Case:-**

Leaf element level $= \log_2(n+1)$

$$= \log_2(7+1)$$

$$= \log_2 8$$

If given element (eg $x = 9$) is matching with a leaf, then no. of comparisons required $= 3 = \log_2(n+1)$

$=$ level of x.

$\therefore$ Time complexity $= O(\log_2 n)$, we neglect the constant $+1$.

(3) **Average Case:-**

Avg no. of comparisons $= \frac{1+2+2+3+3+3+3}{7} = \frac{17}{7}$

$= 2.43 \approx 3 = \log_2 8 = \log_2 n$

$\therefore$ Time Complexity $= O(\log_2 n)$