39

↳ Binary Search

Au: May.09, 11, Dec 12, 13
Mks : 16

Binary Search is an efficient searching method. While searching the elements using this method the most essential thing is that the ele. in the array should be sorted one

An ele. which is to be searched from the list of elements stored in array A[0 . . n-1] is called KEY ele

There are 3 conditions

A[0] . . . A[m-1] A[M] A[m+1] . . . A[n-1]

Search if Key < A(m)

Key (K)

Search if Key > A(m)

eg)

0	1	2	3	4	5	6	7	8	9	10	11
3	14	27	31	39	42	55	70	74	81	85	93

k = 70

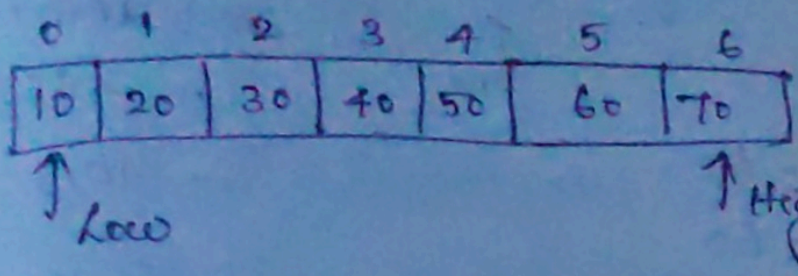
$\frac{0+12}{2} = 6$

$7+12/2 = 9.5$

$-14/2 = 7$

→ Example

Consider a list of elements stored in array A as



The key element to be searched is 60.

$m = (low + high)/2$

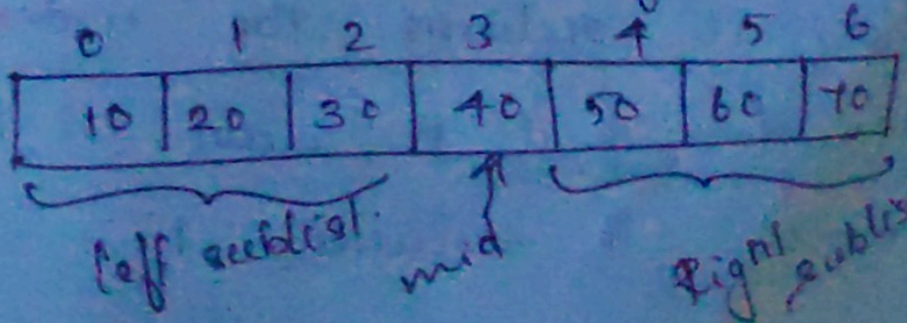
$= (0+6)/2$

$m = 3$

$A[3] = 40$. key value is 60.

$40 < 60$

∴ Search the right sublist.



30

The sorted sublist is

4	5	6
50	60	70

m

m = (low+high)/2
= 4+6/2
= 10/2 = 5

A[5] = 60
key = 60.

∴ the number is present in the list

→ Algorithm

Algorithm BinSearch (A[0 ... n-1], key)

// Problem description : searching the ele using binary search method.

// i/p : key k is to be searched.

// o/p : SI returns the index of an array ele if it is equal to key Otherwise it returns -1.

```
low ← 0
high ← n-1
while (low < high) do
{
    m ← (low + high)/2
    if (key = A[m]) then
    return m
    else if (key < A(m)) then
    high ← m-1          // Search the left sublist
    else
        low ← m+1.      // Search the right sublist
}
return -1
```

```
l ← 0, r ← n-1
while (l ≤ r) do
m ← ⌊(l+r)/2⌋
if (k = A[m])
return m
else
  if (k < A[m])
  r ← m-1
  else l ← m+1
return -1
```

↳ Analysis.

To analysis the efficiency of binary search we must count the no of times.

The comparison is also called a three way comparison to determine whether key is smaller, equal to or greater than A[m].

The worst case time complexity is given by

$C_{worst}(n) = C_{worst}(n/2) + 1$ for $n > 1$

1 — Time required to compare left or right sublist.

L — one comparison made with middle ele.

$C_{worst}(1) = 1$

When array gets divided the above eqns can be written as

$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1$ for $n > 1$

$C_{worst}(1) = 1$

Assume $n = 2^k$ it becomes.

$C_{worst}(2^k) = C_{worst}(2^k/2) + 1$

$= C_{worst}(2^{k-1}) + 1$

Using backward substn method, we can substitute

$C_{worst}(2^{k-1}) = C_{worst}(2^{k-2}) + 1$

Then Eqn becomes

$C_{worst}(2^k) = [C_{worst}(2^{k-2}) + 1] + 1$

$= C_{worst}(2^{k-2}) + 2$

⋮

$= C_{worst}(2^{k-k}) + k$

$= C_{worst}(2^0) + k$

$= C_{worst}(1) + k$

$C_{worst} = 1$ $C_{worst}(2^k) = 1 + k$

$n = 2^k$

$\log_2 n = \log_2 2^k$

$= k \log_2 2$

$= k(1)$

$k = \log_2 n$

$= 1 + k$

Cworst : $1 + \log_2 n$. for $n \geq 1$

∴ Worst case Complexity is $O(\log_2 n)$ & Avg case Best case $O(1)$

→ Another Method

Ans: Dec 10