BINARY SEARCH

Let a_i, $1 \leq i \leq n$ be a list of n elements that are sorted in nondecreasing order. Suppose the problem is to determining whether a given element x is present in the list. If x is present, then $a_j = x$. If x is not present in the list, then j is set to be zero.

Let P be a arbitary instance of this bin-search problem, $P = (n, a_i, \ldots, a_l, x)$ where n is the no. of elements lists, $a_i \ldots a_l$ is the list of elements, and x is the elements to be searched for.

Divide & Conquer strategy is used here to solve this problem. Let Small(P) be true if n = 1. In this case S(P) will take the value of i if $a_i = x$; otherwise it will take the value 0. Then $g(1) = \theta(1)$.

If P has more than one element it can be divided or reduced into a new subproblems as follows:

Suppose we pick an index q in the range $[i, l]$ and compare x with a_q. There are 3 possibilities —

(i) $x = a_q$ (in this case the problem is immediately solved)

(ii) $x < a_q$ (in this case x has to be searched for only in the sublist $a_i, a_{i+1}, \ldots a_{q-1}$.

Therefore, P reduces to $(q-i, a_i, \dots, a_{q-1}, x)$.

(3) $x > a_q$ (In this case the sublist to be searched is $a_{q+1}, \dots, a_l$ and P reduces to $(l-q, a_{q+1}, \dots, a_l, x)$

In this example, any given problem P gets divided (reduced) into one new subproblem. This division takes only $\Theta(1)$ time. After a comparison with a_q, the instance remaining to be solved (if any) can be solved by using this divide-and-conquer scheme again. If q is always chosen such that <u>a_q is the middle element.</u> (ie, $q = \lfloor (n+1)/2 \rfloor$), then the result-ing ~~searching~~ search algorithm is known as <u>binary search.</u>

<u>Note</u>:- The answer to the new subproblem is also the answer to the original problem P; there is no need for the combining.

Algo 1: describes this binary search method, where BinSrch has four i/ps a[ ], i, l and x. It is initially invoked as BinSrch (a, 1, n, x).

Algorithm 1 :-

Recursive Algorithm for Binary Search :-

```
Algorithm Binsrch(a, i, l, x)
// Given an array a[i : l] of elements in nondecreasing
order, 1 ≤ i ≤ l, determine whether x is present,
and if so, return 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;
    }
  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);
      else return Binsrch(a, mid+1, l, x);
  }
}
```

Algorithm BinSearch (a, n, x)

// Given an array a[1 : n] of elements in nondecreasing order, $n \geq 0$, determine whether x is present, and if so, return j such that x = a[j]; else return 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 low := mid+1;
             else return mid;
    }
    return 0;
}
```

Example :

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

Suppose we want to search $x = 151$.

$x = 151$

low	high	mid
1	14	7
8	14	11
12	14	13
14	14	14

found.

$x = -14$

low	high	mid
1	14	7
1	6	3
1	2	1
2	2	2
2	1	not found.

$x=9$

low	high	mid.
1	14	7
1	6	3
4	6	5
		found.

Complexity of Binary Search.

In binary search, after each iteration, the length of the array gets cut in half. Binary Search can be analyzed with the best, worst and average case number of comparisons. These analyses are dependent upon the length of the array. So let $N = |A|$ denote the length of the array A.

In recursive binary search, count each pass through the if-then-else block as one comparison. For iterative Binary Search, count each pass through the while block as one comparison.

Best case - $O(1)$ comparisons.

In the best case, the item X is the middle in the array A. A constant number of comparisons (actually just 1) are required.

Worst case - $O(\log n)$ comparisons.

In worst case for successful search or unsuccessful search of X, through each recursion or iteration of Binary Search the size of the admissible range is halved. This halving can be done ceiling $(\log n)$, $\lceil \log n \rceil$ times. Thus $\lceil \log n \rceil$ comparisons are required.

Scanned by CamScanner

Average case - $O(\log n)$ comparisons.

To find the average case, take the sum over all elements of the product of number of comparisons required to find each element, and the probability of searching for that element. To simplify the analysis, assume that no item which is not in A will be searched for, and that the probabilities of searching for each element are uniform.

The difference between $O(\log n)$ and $O(n)$ is extremely significant when n is large.

Successful Searches

$\Theta(1)$ → best, $\Theta(\log n)$ → average.

$\Theta(\log n)$ → worst

Unsuccessful Searches

$\Theta(\log n)$

best, average, worst.