GITAM
(DEEMED TO BE UNIVERSITY)
(Estd. u/s 3 of the UGC Act, 1956)
VISAKHAPATNAM • HYDERABAD • BENGALURU
Accredited by NAAC with A+ Grade

Design and Analysis of Algorithms

Module I: Part B: Divide and Conquer

Dr. PV Nageswara Rao

Professor, Dept. of CSE

GIT, GU, VSP

Divide and Conquer(DandC)

- **The general method, Control Abstraction**
- **Binary search**
- Finding maximum and minimum
- Merge sort
- Quick sort selection
- Strassen's matrix multiplication.

Divide and Conquer(DAndC): Introduction

- Split the inputs into k distinct subsets, $1<k\leq n$, yielding k sub problems.
- If the sub Problems are still large, then apply divide and conquer strategy again.
- If the sub Problems are still large, apply divide and conquer until it reduces to small enough to carryout the task.
- The sub Problems must be solved & then a method must be found to combine sub solutions into a solution of the whole.
- The sub Problems resulting from a divide and conquer are of the same type as the original Problem.
- The reapplication of the divide and conquer principle is expressed by a recursive algorithm.

Control Abstraction of Divide and Conquer

```
Algorithm DAndC(P)
{
   if small(P) then return  S(P);
   else
    {
      Divide P into small instances P1,  P2, P3,…………Pk  k>1;
      Apply DAndC to each of these sub Problems
      return combine (DAndC(P1), DAndC(P2), DAndC(P3),…DAndC(Pk));
     }  // end of if
}   // end of algorithm DAndC
```

Computing Time:

$$T(n)= \begin{cases} g(n) & n \text{ small} \\ T(n1)+T(n2)+\ldots\ldots +T(nk) + f(n) & \text{otherwise} \end{cases}$$

$$T(n)= \begin{cases} T(1) & n=1 \\ a\,T(n/b) + f(n) & n>1 \end{cases}$$

Where a & b are known constants   and n is a Power of b i.e., $n=b^k$

Example:

Substitution method for solving the recurrence relation:

Consider the case in which a=2 and b=2

Let T(1)=2 and f(n)=n .

we have..

$T(n) = 2T(n/2) + n$

$=2[2T(n/4) + n/2] + n$

$=4T(n/4) + 2n$

$=4[2T (n / 8) + n / 4] + 2n$

$=8T(n / 8) + 3n$

$=2^3 T(N / 2^3) + 3n$

In general, we see that $T(n) = 2^i T(n/2^i) + in$, for any $\log_2 n \geq i \geq 1$. In particular, then, $T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + n \log_2 n$, corresponding to the choice of $i = \log_2 n$. Thus, $T(n) = nT(1) + n \log_2 n = n \log_2 n + 2n$. $\square$

Search

- Linear Search
  - The data need not be in sorted form
- Binary Search
  - The data should be in sorted form on the search key

- Ascending order / Increasing order
- Descending order / Decreasing order
- Non decreasing order
- Non increasing order

Binary Search

- Let a[ ] $1 \le i \le n$, be a list of elements that are sorted in non-decreasing order.
- Problem statement: determine whether a given element x is present in the list.
  - If x is present, determine a value j such that a[j] = x.
  - If x is not in the list, then j is to be set to zero.
- Let $P = (n, a_i, a_{i+1}, a_{i+2}, \ldots., a_l, x)$ denote an arbitrary instance of this search problem.
- n is the number of elements in the list
- a[i : l] is the list of elements
- x is the element to be searched

Binary Search

- Let Small(P)be true if n = 1.
- In this case, S(P)will take the value i if x = a[i], otherwise it will take the value 0.
- Then g(1) = ɵ(1).
- If P has more than one element, it can be divided (or reduced) into a new sub-problem.
- Pick an index q (in the range [i, l])and compare x with a[q].

1. x = a[q]: In this case the problem P is immediately solved.
2. x < a[q]: In this case x has to be searched for only in the sub list a[i], a[i+1],….a[q-1].
   Therefore, P reduces to (q-i, a[i], a[i+1], …,a[q-1], x).
3. x > a[q]: In this case the sub list to be searched is a[q+1], a[q+2], …, a[l].

   Therefore, P reduces to (l-q, a[q+1], a[q+2], …. a[l], x).

Algorithm : Binary Search -- Recursive Method

```
Algorithm BinSrch(a, i, l, x)
// Given an array a[i:l] of elements in non-decreasing order, 1<=i<=l,
//determine whether x is present and if so, return j such that x=a[j]; else return 0
{  if (i==l) then // If Small(P)
      { if x==a[i] then return l;
         else return 0; }
    else
       { // Reduce P into 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)
        }  // end of if(i==l)
} // end of Algorithm
```

Algorithm Binary Search : Iterative Method

```
Algorithm BinSearch(a, n, x)
// Given an array a[1:n] of elements in non decreasing order, n>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)
       {  mid = └(i+l)/2┘
             if (x<a[mid] then high = mid -1;
             else if (x>a[mid] then low = mid +1;
                    else return mid;
          } // end of while loop
      return 0;
   } // end of Algorithm
```

Binary Search : Example

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

Example 1: X = 151

Low	high	mid
1	14	7
8	14	11
12	14	13
14	14	14
		found

Example 2: x = -14

Low	high	mid
1	14	7
1	6	3
1	2	1
2	2	2
2	1	not found

Example 3: x = 9

Low	high	mid
1	14	7
1	6	3
4	6	5
		found

Analysis

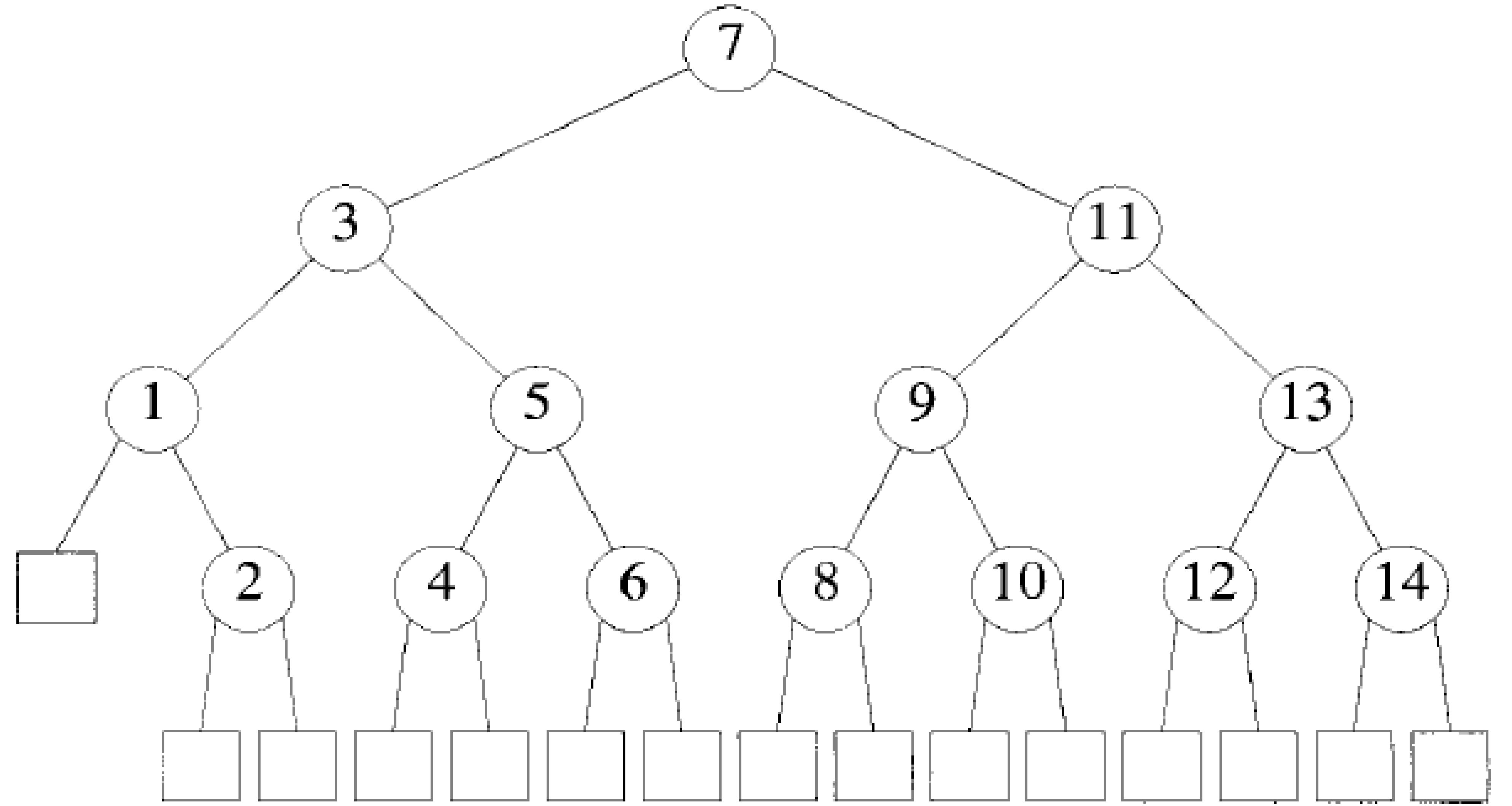
- Is BinSearch an algorithm?
- Does all the operations such as comparisons between x and a[mid] are well defined?
  - The relational operators carry out the comparisons among elements of 'a' correctly if these operators are appropriately defined.
- Does BinSearch terminate?
  - It is observed that low and high are integer variable such that each time through the loop either x is found or low is increased by at least one or high is decreased by at least one. Thus two sequences of integers approaching each other an eventually low becomes greater than high and causes termination in a finite number of steps if x is not present.

Algorithm : Binary Search1

1. Algorithm BinSearch1(a,n,x)
2. // Same specification as BinSearch except n>0
3. {
4. low =1; high = n+1;
5. // high is one more than possible
6. While(low<(high-1) do
7. { mid = └(i+l)/2┘
8. if (x<a[mid] then high = mid;
9. // only one comparison in the loop
10. else low = mid;
11. }
12. if (x = a[low]) then return low;
13. else return 0;
14. } // end of algorithm

Binary Decision Tree for Binary Search, n=14

a:	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
Elements:	−15	−6	0	7	9	23	54	82	101	112	125	131	142	151
Comparisons:	3	4	2	4	3	4	1	4	3	4	2	4	3	4



Binary Decision Tree for Binary Search, n=14

a:	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
Elements:	−15	−6	0	7	9	23	54	82	101	112	125	131	142	151
Comparisons:	3	4	2	4	3	4	1	4	3	4	2	4	3	4

- NO element requires more than 4 comparisons to be found.
- Average for successful : (3+4+2+4+3+4+1+4+3+4+2+4+3+4)/14 = 45/14 = 3.21~
- Average for unsuccessful : (3 + 14 * 4)/15 = 59 /15 = 3.93~

Time Complexity of Binary Search

Successful searches			Unsuccessful searches
Best	**Average**	**Worst**	**Best, Average, Worst**
$\Theta(1)$	**$\Theta(\log n)$**	**$\Theta(\log n)$**	**$\Theta(\log n)$**

- If n is in the range $[2^{k-1}, 2^k)$ the BinSearch makes at most k element comparisons for a successful search and either k-1 or k comparisons for an unsuccessful search.
- The time for successful search is $O(\log n)$ and for unsuccessful search is $\Theta(\log n)$

Computing time of two binary search algorithms

Array sizes	5,000	10,000	15,000	20,000	25,000	30,000
successful searches						
BinSearch	51.30	67.95	67.72	73.85	76.77	73.40
BinSearch1	47.68	53.92	61.98	67.46	68.95	71.11
unsuccessful searches						
BinSearch	50.40	66.36	76.78	79.54	78.20	81.15
BinSearch1	41.93	52.65	63.33	66.86	69.22	72.26

Exercise Problem:

- In an infinite array, the first n cells contain integers in sorted order and the rest of cells are filled with infinity. Present an algorithm that takes x as input and finds the position of x in the array in $\Theta(\log n)$ time. You are not given the value of n.