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

Module I: Part B: Divide and Conquer

CSEN3001: DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-I: Finding Maximum and Minimum Element

Finding Maximum and Minimum: Straight method

```
Algorithm Straightmaxmin(a, n, max, min)
//let max be the maximum and  min be the minimum of  a[1:n]
{
    max:=  min := a[1]
    for i:=2 to n do
    {
        if(a[i] > max) then max :=a[i];
        if  (a[i] < min) then min := a[i];
    }
}
```

Index	Element
1	22
2	25
3	20
4	47
5	37
6	25
7	10
8	45
9	66
10	55

Max : 66
Min : 10

Case1: if (a[i] > max) then max:=a[i];
if (a[i] < min) then min:= a[i];

- This requires 2(n-1) elements comparisons in the best , average and worst case.
- The comparison a[i]<min is necessary only when a[i]>max is false.

if (a[i] > max) then max:=a[i];
else if (a[i] < min) then min:= a[i];

- The **best case** occurs when the elements are in increasing order.
- Number of elements comparisons : (n-1).
- The **worst case** occurs when the elements are in decreasing order.
- The no. of elements comparisons is 2(n-1).

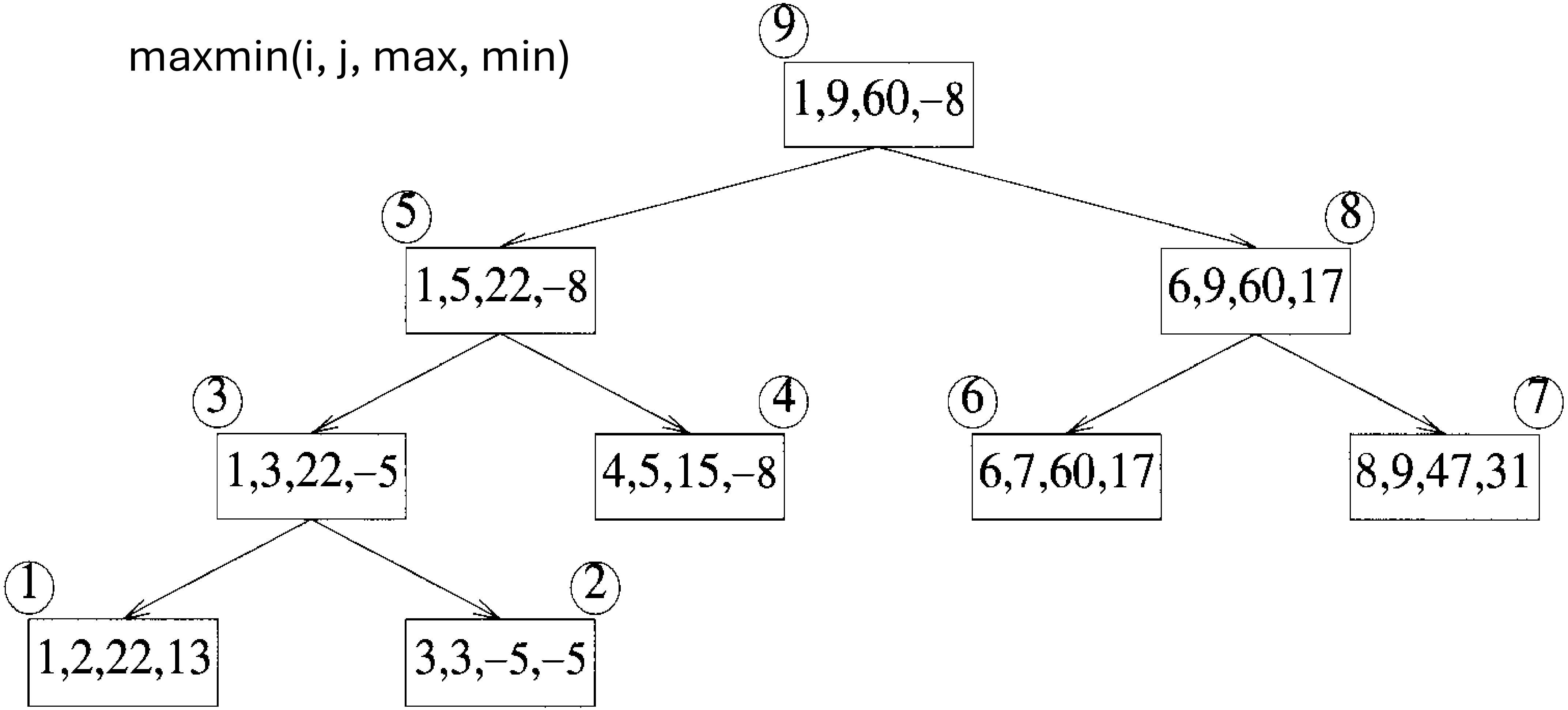
Index	Element
1	22
2	32
3	34
4	47
5	57
6	65
7	70
8	85
9	86
10	95

Finding Maximum and Minimum: DAndC

```
Algorithm  maxmin(i, j, max, min)
// a[1:n] is a global array, i and j integers 1 ≤ i  ≤ j ≤ n
{     if(i=j) then max := min := a[i];   //Small(P)
      else if  (i = j-1) then // Another case for small(P)
              {  if  (a[i] < a[j]) then  {  min :=a[i];   max :=a[j];
                 else {min := a[j]; max:=a[i]}
               }
          else {  // if  P is not small divide P into sub problems. Find where to split the set.
                      mid= (i+j)/2;
                     // solve the sub problem/
                      maxmin (i, mid, max, min);
                      maxmin (mid+1, j, max1, min1);
                 }
   // combine the solutions
                 if  (max< max1) then max:=max1;
                 if(min>min1)the min := min1;
   } // end of if-else
} // end of algorithm
```

Index	1	2	3	4	5	6	7	8	9
Value	22	13	-5	-8	15	60	17	31	47

maxmin(i, j, max, min)



Finding Maximum and Minimum: Introduction

No . of elements comparisons needed for max min is:

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2 & n > 2 \\ 1 & n = 2 \\ 0 & n = 1 \end{cases}$$

When n is a power of 2. i.e., $n=2^k$, where k is a positive integer.

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

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

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

$4(2T(n/8)+2)+2^2+2$

$8T(n/8)+2^3+2^2+2..$

$2^{k-1}T(2) + \sum_{1 \le i \le k-1} 2^i$

$2^{k-1} + 2^k - 2$

3n/2 -2 number of comparisons for best, average and worst case comparisons when n is a power of w.

Recursive calls of max min

- In terms of storage, max-min is worse than the straightforward algorithm. Because it requires stack space for i, j max, min, max1, min1.
- For n elements, there will be [$\log_2 n$] +1 levels of recursion needed to save 'n' values for the recursive call
- If comparisons among the elements of a[ ] are much more costly than comparisons of integer variables, then the divide and conquer technique has given a more efficient algorithm. If not, it yields a less efficient algorithm.
- DAndC strategy is only a guide to better algorithm design, which may not always succeed. Both maximum & straightmaxmin are O(n)