Finding Minimum and Maximum

- **Session Learning Outcome-SLO:** Able to Solve Finding Minimum and Maximum using divide and conquer approaches

- **Motivation of the topic:** The problem is to find the ‘maximum’ and ‘minimum’ items in a set of ‘n’ elements

- Algorithm:
- *Algorithm MaxMin(A, n, max, min)*// **DIRECT APPROACH**

```
// Set max to the maximum and min to 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];
}
}
```

- The above algorithm requires 2(n-1) element comparisons in the best, average and worst cases.

- A *divide-and-conquer* algorithm for this problem would proceed as follows: Let P = (n,a[i],….,a[j]) denote an arbitrary instance of the problem. Here n is the number of elements in the list a[i],….,a[j] and we are interested in finding the maximum and minimum of this list. Let small(P) be true when n ≤ 2. In this case, the maximum and minimum are a[i] if n = 1. If n = 2, the problem can be solved by making one comparison.

- If the list has more than two elements, P has to be divided into smaller instances. For example, we might divide P into the two instances P1 = (n/2,a[1],….,a[n/2]) and P2 = (n - n/2,a[n/2 + 1],….,a[n]). After having divided P into two smaller sub problems, we can solve them by recursively invoking the same divide and conquer algorithm.

- Now the question is How can we combine the Solutions for P1 and P2 to obtain the solution for P?
- If MAX(P) and MIN(P) are the maximum and minimum of the elements of P, then MAX(P) is the larger of MAX(P1) and MAX(P2) also MIN(P) is the smaller of MIN(P1) and MIN(P2).

- MaxMin is a recursive algorithm that finds the maximum and minimum of the set of elements {a(i),a(i+1),…,a(j)}. The situation of set sizes one (i=j) and two (i=j-1) are handled separately.

- For sets containing more than two elements, the midpoint is determined and two new sub problems are generated. When the maxima and minima of this sub problems are determined, the two maxima are compared and the two minima are compared to achieve the solution for the entire set.

Algorithm for maximum and minimum using divide-and-conquer

```
MaxMin(i, j, max, min)
// a[1:n] is a global array. Parameters i and j are integers,   // 1≤i≤j≤n. The effect is to set max and min to the largest and  // smallest
values in a[i:j].
{
    if (i=j) then max := min := a[i]; //Small(P)
    else if (i=j-1) then // Another case of Small(P)
        {
            if (a[i] < a[j]) then max := a[j]; min := a[i];
            else max := a[i]; min := a[j];
        }
    else
    {
        // if P is not small, divide P into sub-problems.
        // Find where to split the set.
        mid := ( i + j )/2;
        // Solve the sub-problems.
        MaxMin( i, mid, max, min );
        MaxMin( mid+1, j, max1, min1 );
        // Combine the solutions.
        if (max < max1) then max := max1;
        if (min > min1) then min := min1;
    }
```

Analysis

what is the number of element comparisons needed for MaxMin? If T(n) represents this number, then the resulting recurrence relation is

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

When n is a power of two, $n = 2^k$

-for some positive integer k, then

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \\ &= 4T(n/4) + 4 + 2 \\ &\;\;. \\ &\;\;. \\ &= 2^{k-1} T(2) + \sum(1 \le i \le k-1)\, 2^k \\ &= 2^{k-1} + 2^k - 2 \\ &= 3n/2 - 2 = O(n) \end{aligned}$$

Note that 3n/2 – 2 is the best, average, worst case number of comparison when n is a power of two.

Comparisons with Straight Forward Method:

Compared with the 2n – 2 comparisons for the Straight Forward method, this is a saving of 25% in comparisons. It can be shown that no algorithm based on comparisons uses less than 3n/2 – 2 comparisons.

- Activity:
- The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is ________.