3.4.2 Max-Min Problem

Max min problem is to find maximum and minimum number from give array. Algorithm needs to scan an array once to find minimum or maximu element. Running time of algorithm linearly grows with input size. Simp approach to find minimum and maximum element from array is shown below :

Algorithm :

MAX_MIN(A) *// A is the array of size n*	**Number of steps**
min ← A[1]	1
max ← A[1]	1
for i ← 2 to n **do**	n
if A[i] < min **then**	n − 1
min ← A[i]	O(n − 1)
end	–
if A[i] > max **then**	n − 1
max ← A[i]	O(n − 1)
End	–
End	–
Complexity :	3n + 2.O(n − 1) = O(n)

- This is easy to conclude that running time of algorithm is O(n). Algorit performs number of comparison in order of n.
- Let us discuss the divide and conquer approach to solve the Max-M problem. This approach for Max.

Min problem works in three stages.

- If a_1 is the only one element in array, a_1 is the maximum a minimum.

- If a_1 and a_2 are only two elements in the array, then simple comparison between two elements can identify minimum and maximum of them.
- If there are more than two elements, algorithm divides the array from middle and creates two sub problems. Both sub problems are treated as independent problem and same recursive call is applied on them. Subdivision continues until sub problem size becomes one or two.

- After solving two sub problems, their minimum and maximum numbers are compared to build the solution of large problem. This process continues in bottom up fashion to build solution of even larger and larger problem.

```
Algorithm DC_MAXMIN (A, i, j, max, min)
// A is the array of size n, 1 ≤ i ≤ j ≤ n
// i is the lower index of array passed to function
// j is the upper index of array passed to function

if i == j then                          // Problem of size 1
    min ← max ← A[i]
else if i == j - 1 then                 // Problem of size 2
    if A[i] < A[j] then
        min ← A[i]
        max ← A[j]
    else
        min ← A[j]
        max ← A[i]
else
    mid ← (i + j) / 2
    DC_MAXMIN (i, mid, max, min)              // Divide problem if it is large
    DC_MAXMIN (mid + 1, j, max1, min1)        // Solve sub problem 1
                                              // Solve sub problem 2
    if max < max1 then                        // Combine solution
        max ← max1
    end
    if min < min1 then                        // Combine solution
        min ← min1
    end
end
```

Complexity analysis :

Conventional algorithm takes 2(n – 1) comparisons in worst, best and average case.

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

Let us solve this equation using interactive approach.

$$T(n) = 2T\left(\frac{n}{2}\right) + 2$$

By substituting n by (n / 2) in above equation

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + 2\right] + 2$$

$$= 4T\left(\frac{n}{4}\right) + 4 + 2$$

By substituting n by $\frac{n}{2}$ in original recurrence,

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + 2$$

$$\therefore\ T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2 = 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

$$= 2^3\,T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2^1$$

$$\vdots$$

After k – 1 iterations

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

$$= \frac{2^k}{2} + 2^k - 2 = \frac{n}{2} + n - 2, \text{ since } n = 2^k = \frac{3n}{2} - 2$$

It can be observed that, divide and conquer approach does only $\left(\frac{3n}{2} - 2\right)$ comparisons compared to 2(n – 1) comparisons of conventional approach.