Finding Minimum and Maximum Algorithm and Analysis

- **Problem:** To find the maximum and minimum elements in a set of n elements.

- For analyzing time complexity, ~~cost~~ Number of element comparison is used, because if elements are polynomial, vector, large number, strings then cost of comparison is much higher than other operations

- **Straightforward Maximum & Minimum algorithm:**

Algorithm:

```
StraightMaxMin (a, n, max, min)
{
1.      max = min = a[1];
2.      for i = 2 to n
        {
3.          if (a[i] > max) then max = a[i];
4.          if (a[i] < min) then min = a[i];
        }

}
```

In straightMaxMin, total number of comparison is $2(n-1)$.

Improvement: Replacing content in for loop, i.e

```
1 2 3 4 |   if (a[i] > max) then max = a[i];
4 3 2 1 |   elseif (a[i] < min) then min = a[i].
```

For this, if best case number of comparisons will be $(n-1)$ & worst case number of comparisons will be $2(n-1)$.

- Divide And Conquer Approach

- $P(n, a[i] \dots a[j])$ - denotes problem instance with n elements in the array $a[]$.

- Base condition - is when $n \leq 2$.

if $n = 2$, problem can be solved with one comparison

if $n = 1$, then single element is max as well as min.

- Divide : 'n' elements array is divided into two arrays of size $n/2$ by calculating mid.

i.e $P = (n, a[1] \dots a[n])$ is divided into

$P_1 = (n_1, a[1] \dots a[n/2])$ and

$P_2 = (n_2, a[n/2+1] \dots a[n])$

Conquer: Find minimum and maximum on two subsequence recursively.

Combine: Two maxima and minima of subproblems are compared to achieve the solution for the entire set.

Algorithm:

i/p : i, j, max, min
i, j → Indices in array
max & min → will be smallest & largest in a[i ... j]
a[] is global array.

o/p : setting min & max in a[i:j]

```
MaxMin (i, j, min, max)
{
1.  if (i == j) then max = min = a[i];
                    // First case Base condition ie n=1
2.  else if (i == j-1) // case 2 : n = 2
    {
3.      if (a[i] < a[j]) then
        {
4.          max = a[j]
5.          min = a[i]
        }
6.      else
```

```
        {
7.          max = a[i]
8.          min = a[j]
        }
    }
9.  else                                    // case 3: n > 2
    {
10.         mid = ⌊(i+j)/2⌋                 // Divide

11.         MaxMin (i, mid, max, min);      } // conquer.
12.         MaxMin (mid+1, j, max1, min1);  }

13.         if (max < max1) then max = max1;    } // combine
14.         if (min > min1) then min = min1     }
    }
}
```

— Analysis :

Divide : computing mid takes constant time $\Theta(1)$

Conquer : Recursively solves 2 subproblem,
$\therefore$ $2T(n/2)$ running time.

Combine : line 13 & 14 ie 2 comparison,
hence constant time ie 2

Resulting recurrence relation is,

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

∴ Number of comparison are,

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

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

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

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

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

$$\vdots$$

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

$$= 2^{k-1} + \frac{2^{(k-1+1)} - 1}{2-1} - 1 \qquad \left[\because \sum_{i=0}^{n} 2^i = 2^{n+1} - 1\right]$$

$$2^k + \sum_{i=1}^{k-1} 2^i$$

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

$$n + 2n - 1$$

$$3n - 1$$

$$\left[\because \sum_{i=1}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}\right]$$

$$= 2^{k-1} + 2^k - 2 \qquad [\text{put } n = 2^k]$$

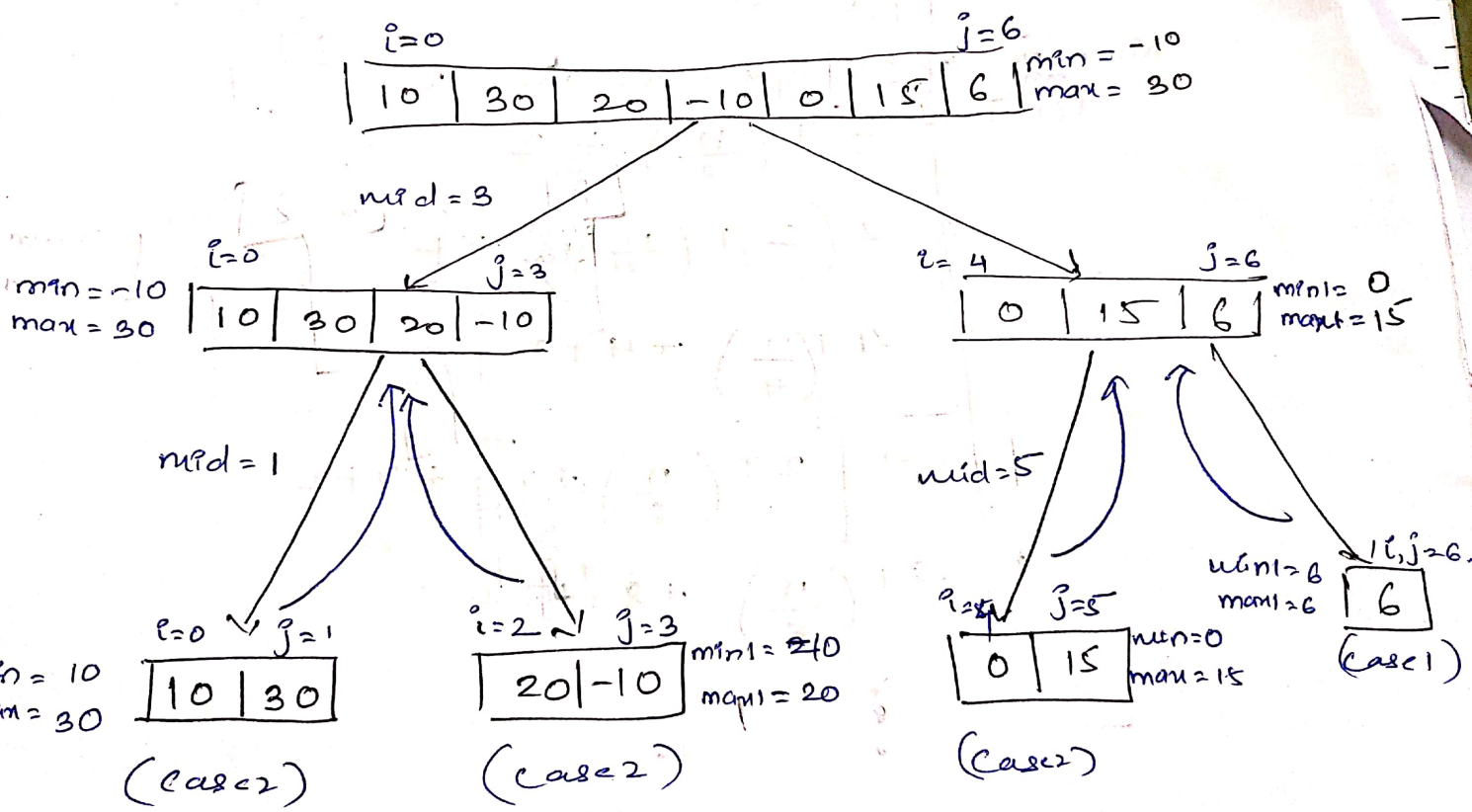
$$= \frac{n}{2} + n - 2$$

$$\sum_{i=1}^{n-1} = \sum_{i=1}^{n} - n$$

$$\boxed{T(n) = \frac{3n}{2} - 2}$$

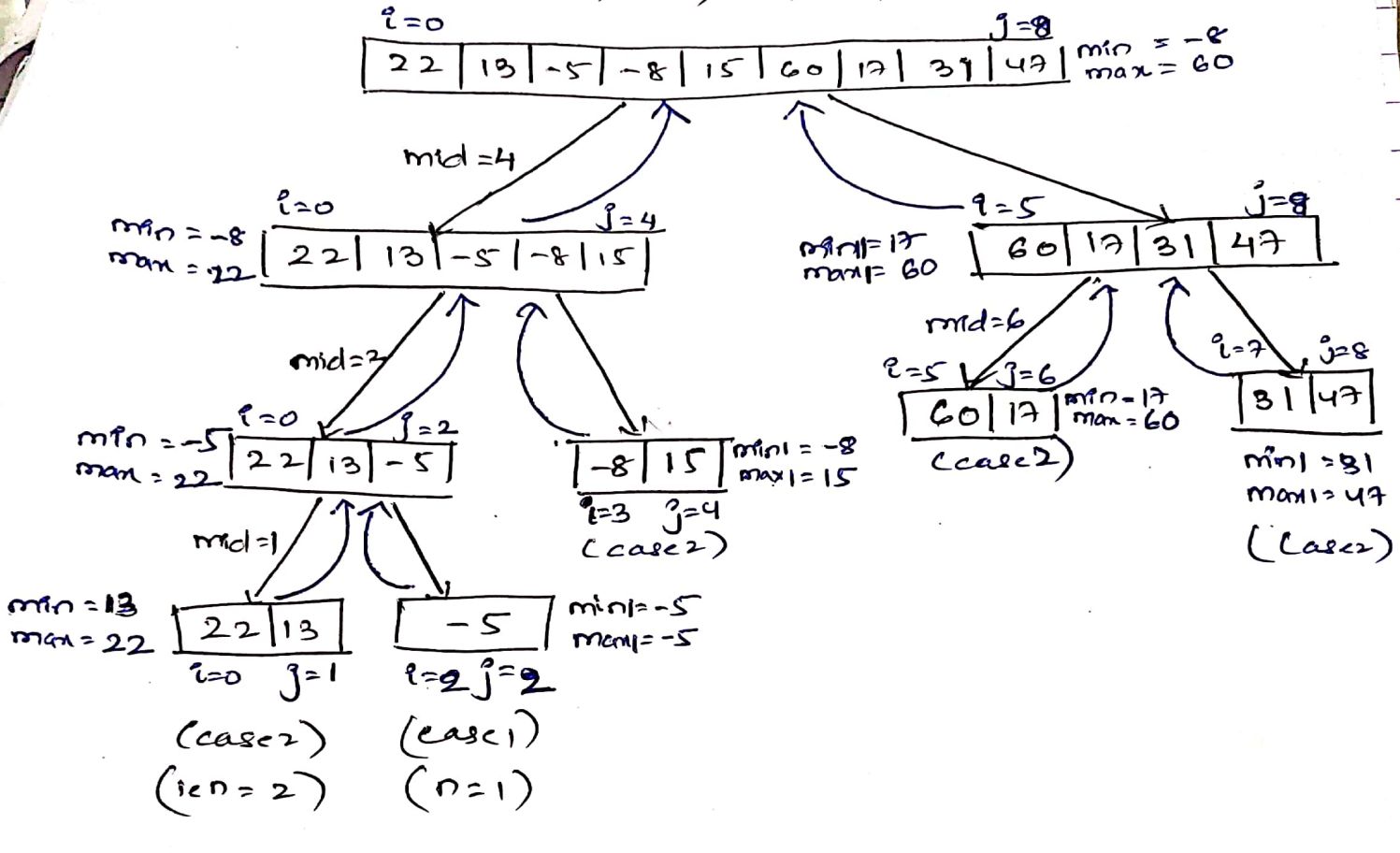
Example 1.

a[] = {10, 30, 20, -10, 0, 15, 6}



Example : 2

a[] = {22, 13, -5, -8, 15, 60, 17, 31, 47}



∴ min = -8

max = 60