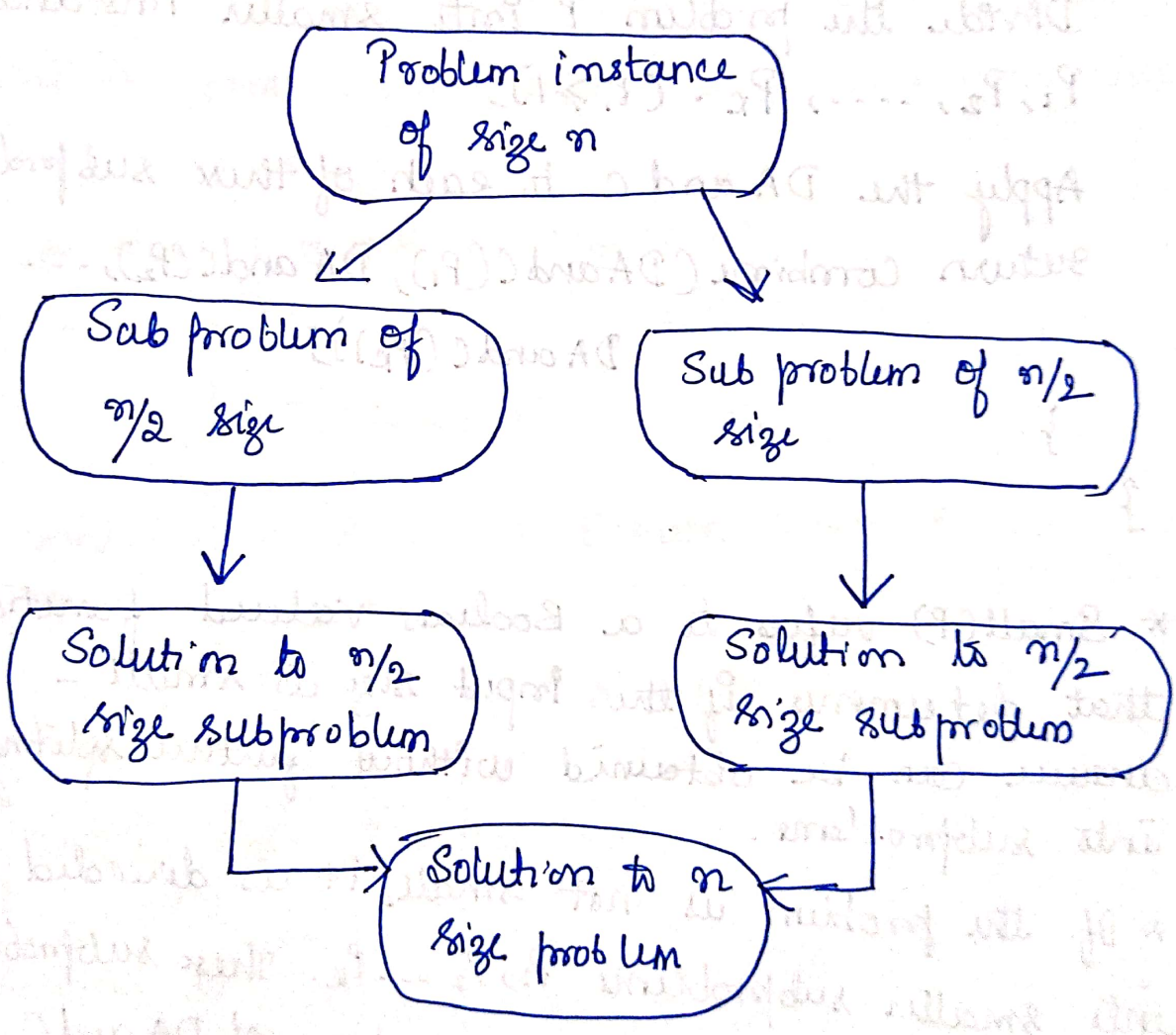
Module - 2

General Method :- Divide and Conquer.

1. Problem is divided into sub-problems [until no further division is required] → Divide.
2. Solve the sub-problem with known method [Simple]
3. If required, combine the solution of the sub-problem instances to find the solution of overall problem.

Conquer.



Algorithm :-

```
Algorithm DA and C (P)
{
  if Small (P) then return S(P);
  else
  {
  Divide the problem P into smaller instances
  P1, P2, ...., Pk . (k >= 1);
  Apply the DA and C to each of these subproblems;
  return Combine (DA and C (P1), DA and C (P2), ...
                  DA and C (Pk));
  }
}
```

* Small(P) refers to a Boolean valued function that determines if the input size is small - answer can be obtained without further splitting into subproblems.
* If the problem is not small, it is divided into smaller subproblems $P_1, P_2 \ldots P_k$. These subproblems are solved by recursive applications of DA and C
* Combine is a function that determines the solution to P using the solutions to the k subproblems.

Recurrence Equation for Divide and Conquer

→ Given a problem P of size n and the sizes of the k subproblems are $n_1, n_2 \ldots n_k$ respectively, then the computing time of DA and C is indicated by the recurrence relation

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1), T(n_2) \ldots, T(n_k) + f(n) & \text{otherwise} \end{cases}$$

$T(n)$ → time for DA and C on any input of size n

$g(n)$ → time to compute the answer for small inputs.

$f(n)$ → time for dividing P and combining the solutions to subproblems.

→ Complexity recurrence relation:

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

where - a & b are known constants

$T(1)$ is known

n is a power of b ($n = b^k$)

Defective Coin Problem:-

Method 1:

1 - 1 → 1

1 - 1 → 2

⋮

1 - 1 → 16

} 2 coins must be kept if its balanced no defut. If it do not balanced then is a defut in coin.

complexity is $\lceil n/2 \rceil$

Method 2:

16 - (16) → 1
↙
8 - (8) → 2
↙
(4) - 4 → 3
↘
(2) - 2 → 4
↘
1 - 1 → 5

Here we split 32 coins into 16 & 16 that time which side shows less weight we will take & that portion and again split 16 coins into 8 & 8 this process continuous till we reach 1-1 finally we get that defective coin

Time complexity $\boxed{\log(n)}$

By comparing method 1 and method 2.
Method 2 is best.

Solution for recurrence relation :-

The complexity of many divide & conquer algorithms is given by the recurrence of the form

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

where a and b are known constants. we assume that T(1) is known and n is a power of b (i.e $n = b^k$)

- One of the methods for solving any such recurrence relation is called the substitution method.
- This method repeatedly makes substitution for each occurance of the function T in the right-hand side until all such occurances disappear.

Master's theorm :-

The efficiency analysis of many divide and conquer algorithms is greatly simplified by master theorm

It states that, in recurrence equation

$T(n) = aT(n/b) + f(n)$ ~~if $f(n) \in \Theta(n^d)$ where $d \geq 0$.~~

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$a \geq 1 \quad b > 1$

$f(n)$

$n^{\log_b a}$

Where $a \geq 1$, $b > 1$ and $f(n)$ should +ve always.

Cases :-

① $f(n) < n^{\log_b a}$

$T(n) = \Theta(n^{\log_b a})$

② $f(n) = n^{\log_b a}$

$T(n) = \Theta(n^{\log_b a} \cdot \log n)$

③ $f(n) > n^{\log_b a}$

$T(n) = \Theta(f(n))$

① $T(n) = 4T(n/2) + n$

Given $a = 4$

$b = 2$

$f(n) = n$

① $f(n) < n^{\log_b a}$

$n^{\log_b a} = n^{\log_4 2} = n^2$

$f(n) < n^2$

$n < n^2$ ✓

$T(n) = \Theta(n^{\log_b a})$

$\boxed{T(n) = \Theta(n^2)}$

② $T(n) = 2T(n/2) + n$

Given = $a = 2$

$b = 2$

$f(n) = n$

$\therefore n^{\log_b a} = n^{\log_2 2} = n$

$\therefore f(n) < n$

$n < n$ ✗

Case (2)

$f(n) = n$

$n = n$

$\therefore f(n) = n^{\log_b a} = n^{\log_2 2} = n$

$T(n) = \Theta(n^{\log_b a} \cdot \log n)$

Divide and conquer method to find min and max value :-

```
Algorithm MaxMin(i, j, Max, Min)
{
    if (i=j) then
    Max ← a[i];
    Min ← a[i];                    } One element
    else if (i=j-1) then
    {
        if (a[i]<a[j]) then
        {
            Max ← a[j];
            Min ← a[i];
        }
```

```
else
{
    Max ← a[i];
    Min ← a[j];
}
}
```

two elements

```
else
{
    Mid ← (i+j)/2
    MaxMin (i, Mid, Max, Min);
    MaxMin (Mid+1, j, Max1, Min1);
    if (Max < Max1) then
        Max ← Max1
    if (Min > Min1) then
        Min ← Min1
}
```

more elements.

Example:

5 7 3 4 9 12 6 2

Case 1: If one element

[5] → Min → 5
Max → 5

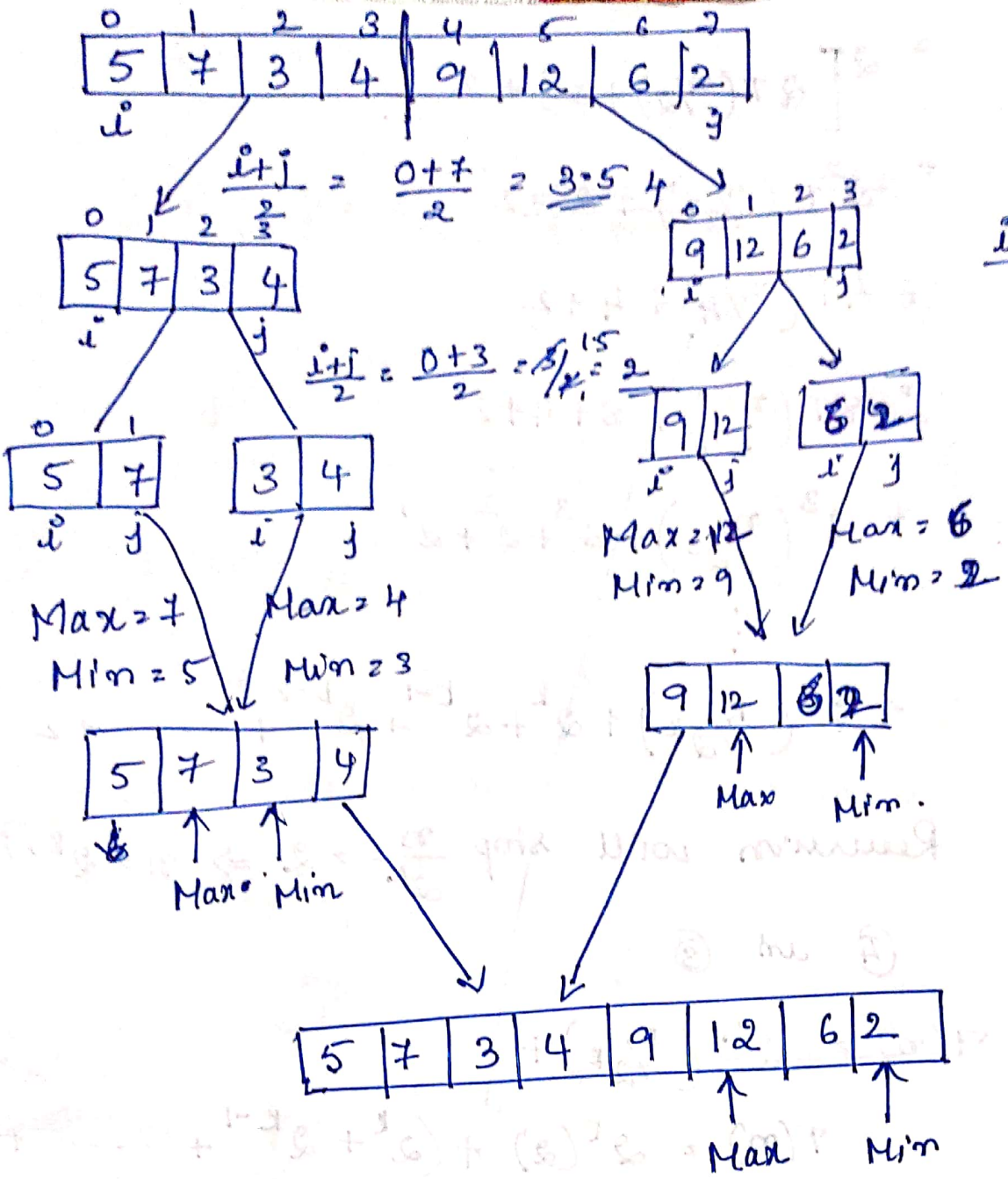
Case 2: If 2 elements

5	7
0	1
↑ i	

if a[i] < a[j]
5 < 7

i = j-1
i = 1-1
i = 0

Max = a[j] = 7
Min = a[i] = 5 //



∴ Max = 12

Min = 2.

Case 3:

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

Time Complexity

$T(n) = T(n/2) + T(n/2) + 2$ → Dividing, combining

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

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

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

$a^{m-n} = \frac{a^m}{a^n}$

$T(n) = 2T(n/2) + 2 \rightarrow (1)$

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

Sub (2) in eqn (1).

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

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

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

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

$= 2^K T(n/2^K) + 2^K + 2^{K-1} + 2^{K-2} + \ldots + 2^3 + 2^2 + 2$

$= 2^3 T(n/2^3) + 14$

$= 2^K T(n/2^K) + C$

$\therefore n/2^K = 2$

$= 2^K T(2) + C$

$= 2^K (1) + C$

$= n(1) + C$

$= n$ $\quad \therefore O(n)$ //

$k = 4$

$= \sum_{i=1}^{3} 2^i$

$= 2^1 + 2^2 + 2^3$

last term $k-1$

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

$= \frac{2^k}{2} \cdot T\left(\frac{n/2^k}{2}\right) + \sum_{i=1}^{k-1} 2i$

$n = 2^k$

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

Assum $= 2^k = n$.

Algorithm for Merge Sort

```
Algorithm Merge Sort (A, lb, mid, ub)
// Input : Array element A[0...n-1]
// Output : Sorted elements sorted in array
 Merge (A, lb, mid, ub)
 {
     i = lb;
     j = mid+1;
     k = lb;
  while (i <= mid && j <= ub)
  {
      if (a[i] <= a[j])
       {
          b[k] = a[i];
           i++;
       }
      else
       {
          b[k] = a[j];
           j++;
       }
       k++;
  }
 if (i > mid)
   {  while (j <= ub)
```

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

$$= \sum_{i=1}^{k-1} 2^i \Rightarrow \frac{a(r^n-1)}{r-1}$$

$2+4+8$

$a=2$

$r=2$ [Common multiplier]

$$= \frac{2\left(2^{k-1}-1\right)}{2-1}$$

$$= 2\left(2^{k-1}-1\right)$$

$= 2 \cdot 2^{k-1} - 2$

$= 2^k - 2$

$= n-2$

```
{
  b[k]=a[j];
  j++;
  k++;
}
else
{
  while(i<=mid)
  {
    b[k]=a[i];
    i++;
    k++;
  }
}
for(k=lb; k<=ub; k++)
{
  a[k]=b[k];
}
```

$$= \frac{2^k}{2^k} \cdot T\left(\frac{2n}{2^k}\right) + n-2$$

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

$$= \frac{n}{2} \cdot T(2) + (n-2)$$

$$= \frac{n}{2} \cdot T(1) + (n-2)$$

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

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

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

$$= \frac{3n}{2} - 2$$

$$= \frac{3n}{2} - 2.$$

Straigh forward method = $O(n)$

$2n-2$

To reduce we use divide & conquer method