Outlines

- Introduction
- Recurrence and different methods to solve recurrence
- Multiplying large Integers Problem
- Problem Solving using divide and conquer algorithm
  - Binary Search
  - Max-Min problem,
  - Sorting (Merge Sort, Quick Sort)
  - Matrix Multiplication
  - Exponential.

Marwadi University

Department of Computer Engineering

Unit no: **3**
Unit title: **Divide and Conquer**
Subject name and code: Design and Analysis of Algorithms (01CE0503)

Prof. Aastha Masrani

Introduction

- Recursive in structure.
- A typical Divide and Conquer algorithm solves a problem using following three steps.

1.*Divide:* Break the given problem into subproblems of same type.
2. *Conquer:* Recursively solve these subproblems
3. *Combine:* Appropriately combine the answers

Recurrence and different methods to solve recurrence

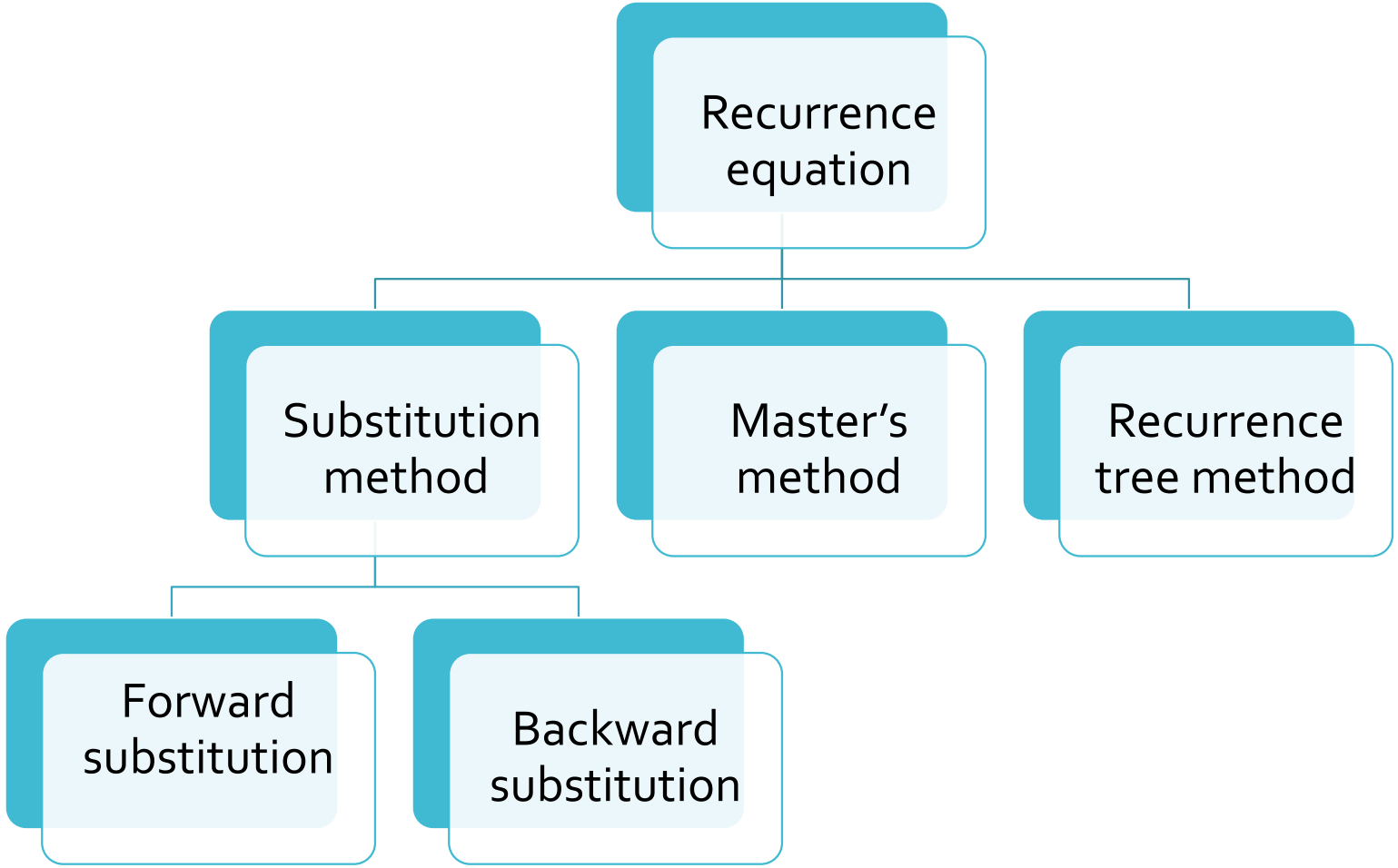
- **Recurrence equation:**
- **Equation that defines a sequence recursively.**

T(n) = T(n-1) + n, n>0 -------(1 (recurrence relation))

T(0) = 0 --------(2 (initial condition))

3 methods to solve recurrence equation.

Recurrence equation



Substitution method

- **1 forward substitution**
- Use initial conditional in initial term, use this value to find next term.
- Recurrence relation

T(n) = T(n-1) + n **Initial condition T(0)**=0

Sol.: T(n) = T(n-1) + n -----(1)

If n=1

T(1) = T(0)+1

= 0+1

T(1) = 1 ----(2)

Substitution method

If n=2

$T(2) = T(1)+2$

$= 1+2$

$T(2) = 3$ ----(3)

If n=3

$T(3) = T(2)+3$

$= 3+3$

$T(3) = 6$ ----(4)

- By observing,
- $T(n) = n(n+1)/2$

$= n^2+n/2$

$= O(n^2)$

It is difficult to find pattern so generally not used.

Backward substitution method

2. Backward substitution

$T(n) = T(n-1) + n$ -----(1)

If n=n-1

$T(n-1) = T(n-1-1)+(n-1)$

$= T(n-2) + (n-1)$ ----(2)

Put eq^n(2) in eq^n (1)

$T(n) = T(n-2) + (n-1)+ n$ ------(3)

let n=n-2

$T(n-2) = T(n-1-2)+(n-2)$

$= T(n-3) + (n-2)$ ----(4)

Backward substitution method

Put eq^n(4) in eq^n (3)

$$T(n) = T(n-3) + (n-2) + (n-1) + n \quad \text{------(3)}$$

$$\vdots$$

$$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \text{------} + n$$

Let k=n then,

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

$$T(n) = T(0) + 1 + 2 + 3 + \text{------} + n$$

$$T(n) = 0 + 1 + 2 + 3 + \text{------} + n$$

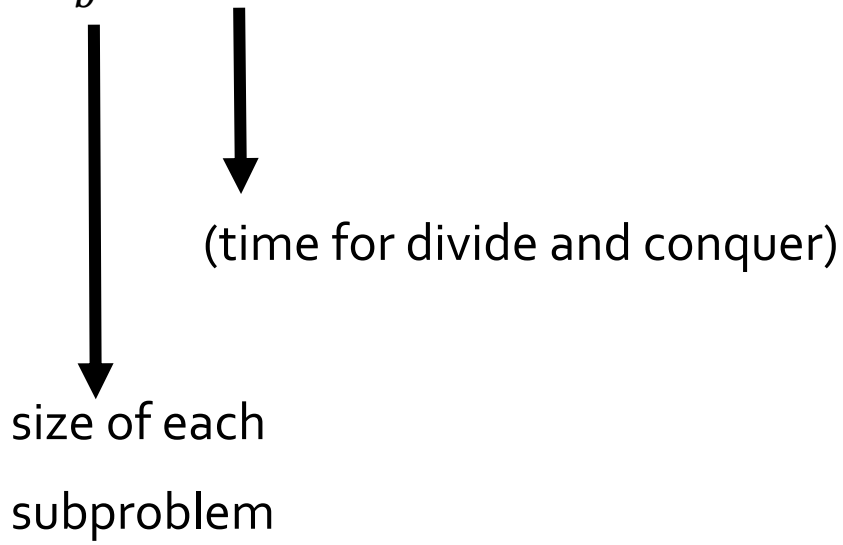
- $T(n) = n(n+1)/2$

$$= n^2 + n/2$$

$$= O(n^2)$$

Master's Method

- $T(n) = a\,T(\frac{n}{b}) + f(n)$ where n>= d and d is constant(ε > o)



Master's Method

- $T(n) = a\,T(\frac{n}{b}) + f(n)$ $a >= 1, b > 1$
- $f(n) = \Theta(n^k \log^p n)$
- Find two values **1)** $\log_b a$ **2) k**
- Total 3 cases:
- **Case 1:**
- **If** $\log_b a > k$ **then** $\Theta(n^{\log_b a})$

- **Case 2:**
- **If** $\log_b a = k$ **then**
  - **If** $p > -1$ **then** $\Theta(n^k \log^{p+1} n)$
  - **If** $p = -1$ **then** $\Theta(n^k \log \log n)$
  - **If** $p < -1$ **then** $\Theta(n^k)$

Master's Method

- **Case 3:**
- **If $\log_b a < k$ then**
  - **If p >= 0 then $\Theta(n^k \log^p n)$**
  - **If p < 0 then $O(n^k)$**

Ex1 . $T(n) = 9T(n/3) + n$

$a = 9, b = 3, f(n) = \theta(n^1 log^0 n), k = 1, p = 0$

$\log_b a = 2$

CASE 1: $\log_b a > k$

$T(n) = \Theta(n^2)$.

Master's Method

Ex2 . $T(n) = 2T(n/3) + 1$

$a = 2, b = 3 , f(n) = \theta(n^0 log^0 n), k = 0 , p = 0$ *and* $p > -1$

$\log_b a = \log_3 2 = n^0 = 1$

CASE 2: $\log_b a = k$,

If p > -1 then $\Theta(n^k \log^{p+1} n)$

$\therefore T(n) = \Theta(\log n)$.

Ex 3 $T(n) = 4T(n/2) + n^3$

$a = 4, b = 2 , f(n) = \theta(n^3 log^0 n), k=3, p=0$

$\log_b a = 2$

CASE 3: $\log_b a < k$

If p >= 0 then $\Theta(n^k \log^p n)$

$T(n) = \Theta(n^3)$.

Recursion tree method

- $T(n) = 4\,T(n/2) + n^3$
- *Where* n^3 = root node
  - $T(n/2)$ = size of sub-problem
  - 4 = number of sub-problem
- **Step 1: find cost of each level**
- **Step 2: find depth of tree**
- **Step 3: find number of leaves**

There are total 3 cases to solve examples. But case 1 and case 2 are solved by above 3 steps.

Case 1: cost of root node is maximum

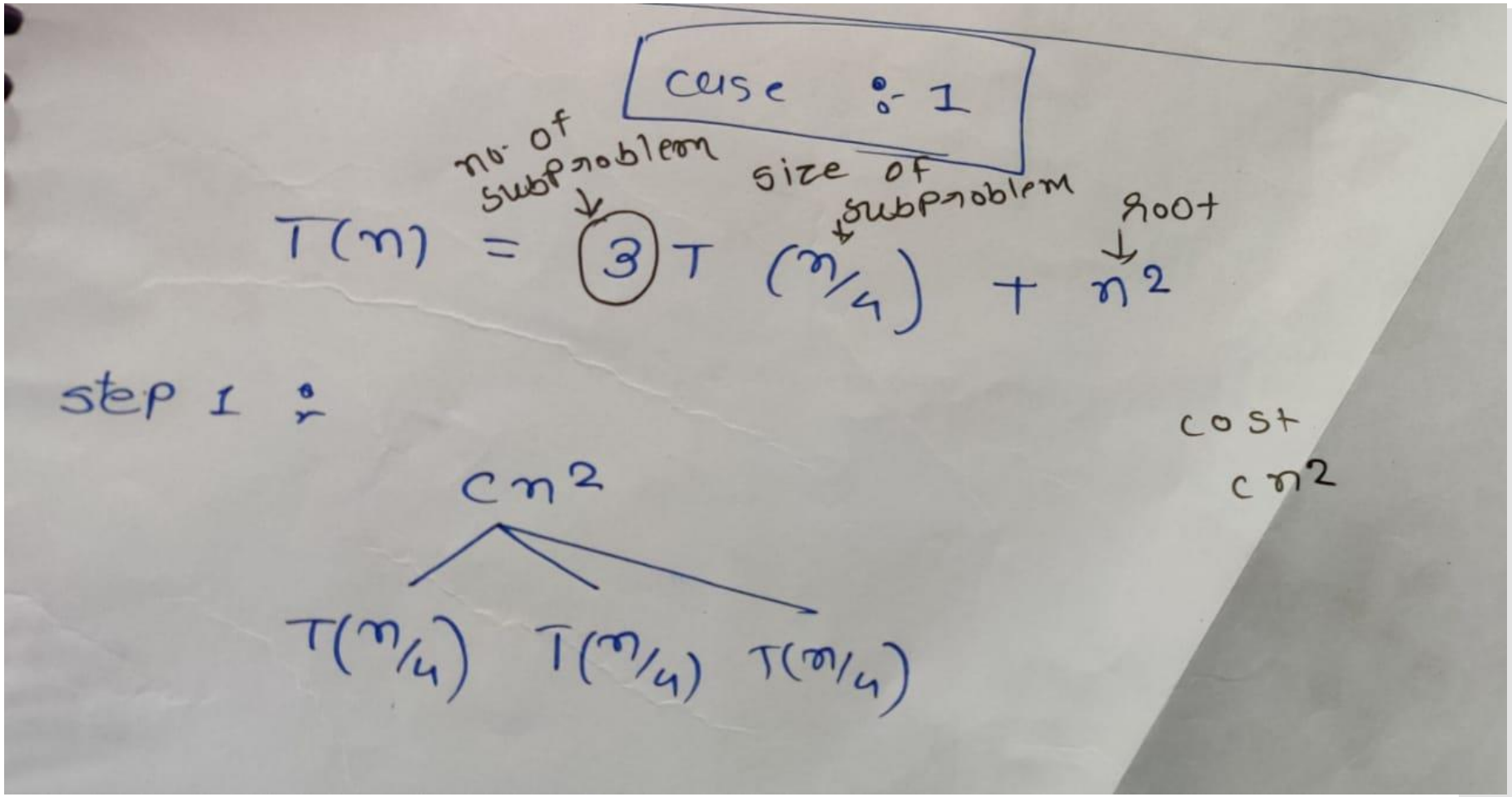
Case 2: cost of leaf node is maximum.

Case 3: cost of each level is same.

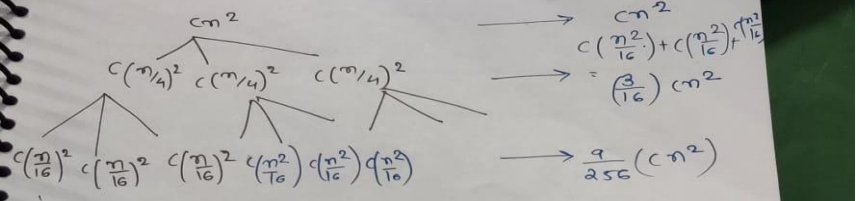
Recursion tree method

- **Case 3: cost of each level is same.**
- Step 1: find cost of each level
- Step 2: find depth of tree
- Step 3: find level of tree
- Step 4: total cost = cost of each level * number of level

CASE 1: recursion tree



Step 2 :-



cost of each level = $(\frac{3}{16})^i cn^2$

depth of each tree :-

$\boxed{\frac{n}{4^i} = 1}$ — formula

$n = 4^i$

take log both side

$\log n = \log 4^i$

$\log n = i \log 4$

$i = \log_4 n$

$n^{\log_4 3}$

$3^i = 3$ increase equal node at $i = \log_4 n$

$3^{\log_4 n}$

$n^{\log_4 3}$

no. of leaves

$3^i = 3^{\log_4 n}$

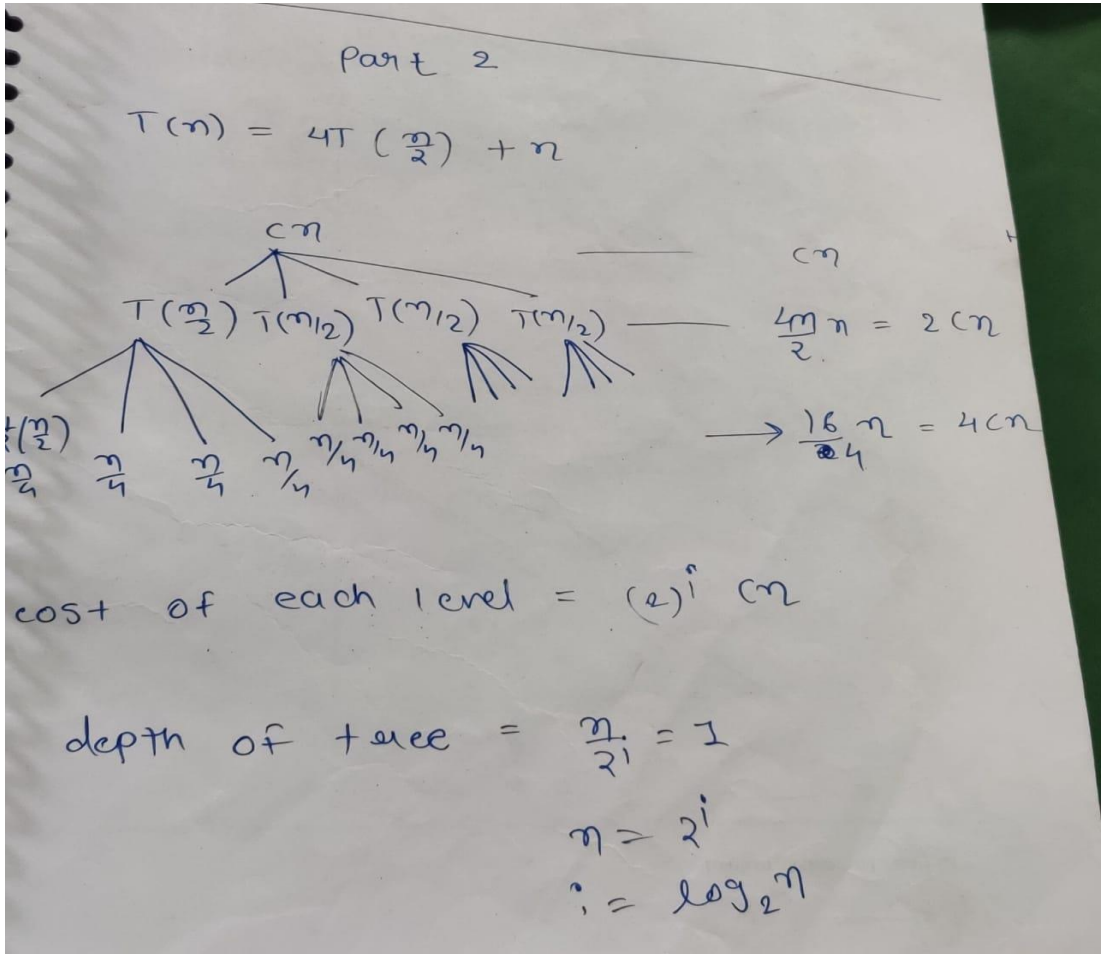
$= n^{\log_4 3}$

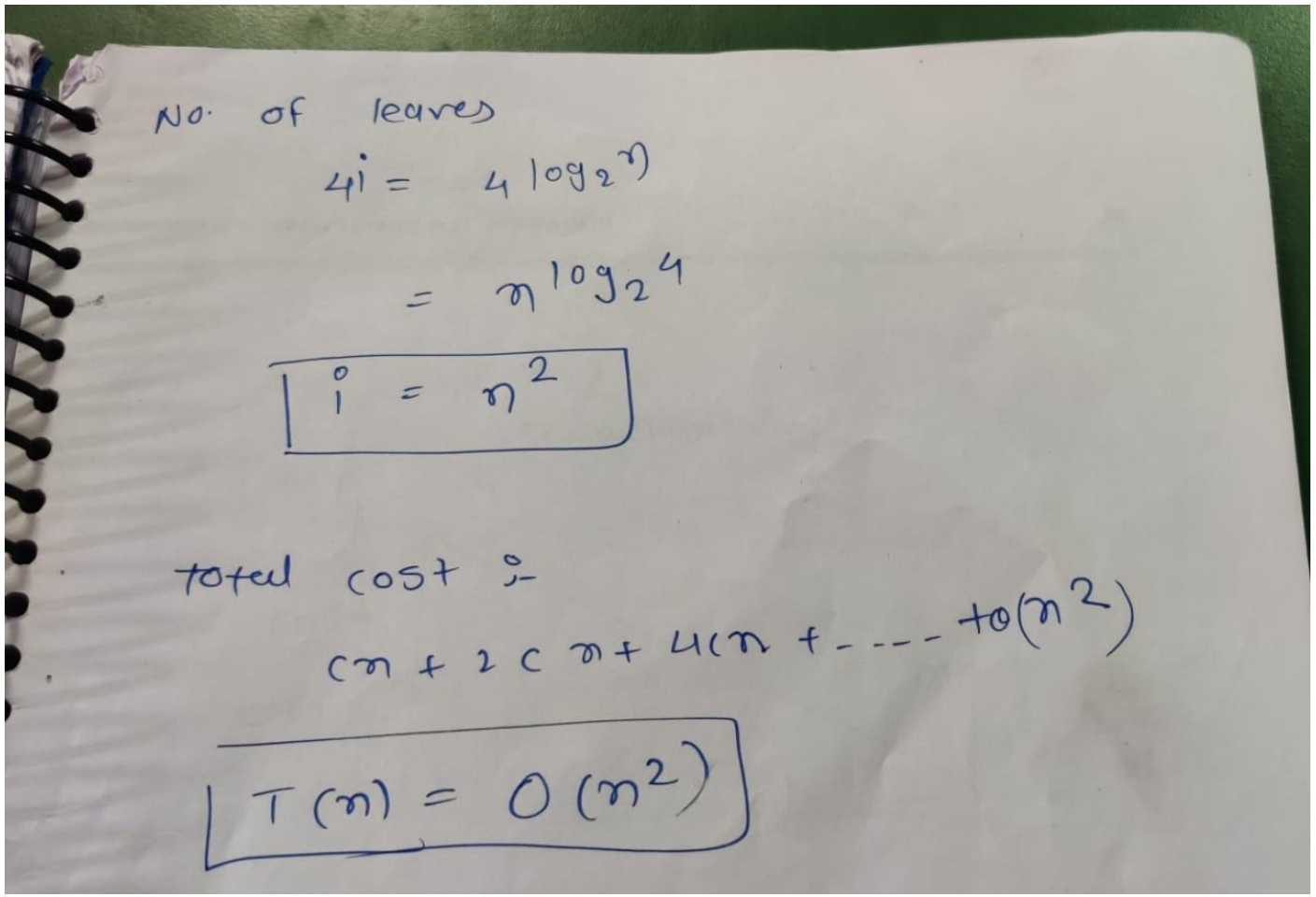
Total cost =

$$T(n) = cn^2 + \frac{3}{16}(n^2 + \cdots + \left(\frac{3}{16}\right)^{d-1} cn^2 + \Theta(3^{\log_4 n})$$

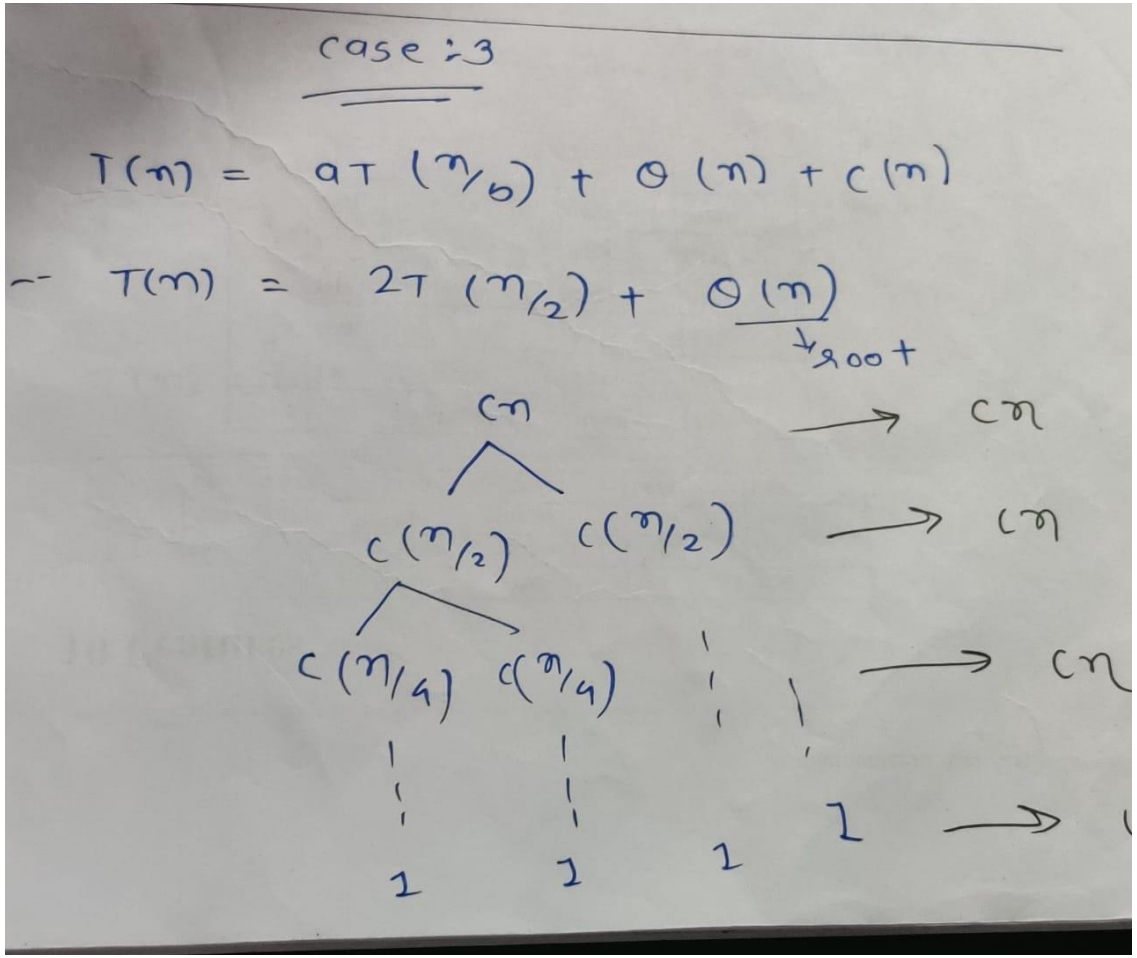
$$= \Theta(n^2)$$

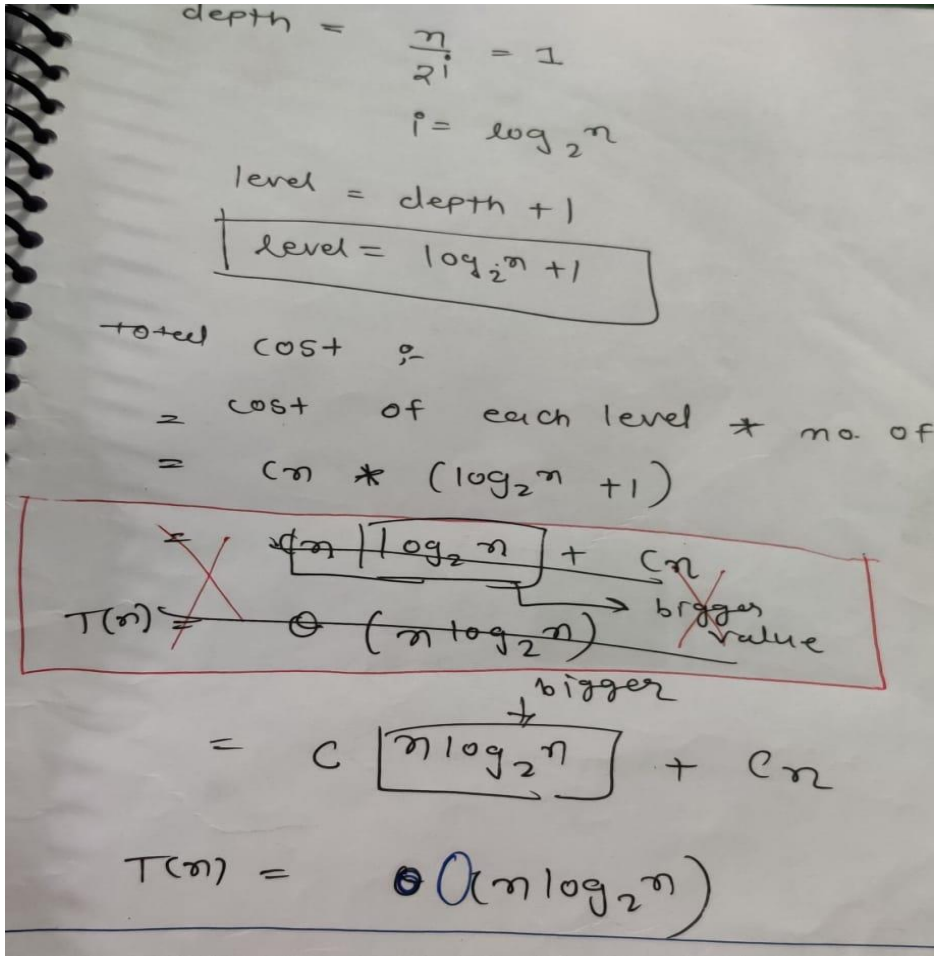
CASE 2: recursion tree




No. of leaves
$4^i = 4^{\log_2 n}$
$= n^{\log_2 4}$
$i = n^2$
total cost :-
$cn + 2cn + 4cn + \cdots + \Theta(n^2)$
$T(n) = O(n^2)$

CASE 3: recursion tree




depth = $\frac{n}{2^i} = 1$
$i = \log_2 n$
level = depth + 1
level = $\log_2 n + 1$
total cost :-
= cost of each level * no. of
= $cn * (\log_2 n + 1)$
$cn \log_2 n + cn$
bigger value
$T(n) = \Theta(n \log_2 n)$
bigger
$= c\,n \log_2 n + cn$
$T(n) = \Theta(n \log_2 n)$

How to find

- *For equation* $T(n) = 4T(n/2) + n^3$
- **Cost of each level:** find , in multiple of which number cost of each level is increasing
- If it increasing in multiple of 2 then
- $2^i cn$ (i=0,1,2,---,n-1)
- **Depth of tree:** Size of sub-problem in recurrence equation and division of sub-problem is stopped when tree size reaches to 1.
- $\frac{n}{2^i} = 1$
- **Number of leaves:** number of sub-problem
- 4^i

Multiplying large integers problem

- X = 1234
- Y= 2345
- Divide integer into 2 portion X = a b
  where a= 12, b = 34 (X= 1234)
- Y = c d
  Where c= 23, d = 45 (Y= 2345)

  XY = (a * $10^{n/2}$ + b) (c * $10^{n/2}$ + d)
  = ac * 10^{n} + $(a*d * 10^{n/2})$+ $(b*c * 10^{n/2})$+ bd
  = ac * 10^{n} + $(ad+ bc)\ 10^{n/2}$ + bd
- The above procedure still needs four multiplications; ac,ad,bc and bd

Multiplying large integers problem

- The key observation is that there is no need to compute both ad and bc; all we really need is the sum of these two terms.
- So there is one algorithm which takes only 3 multiplication.

begin

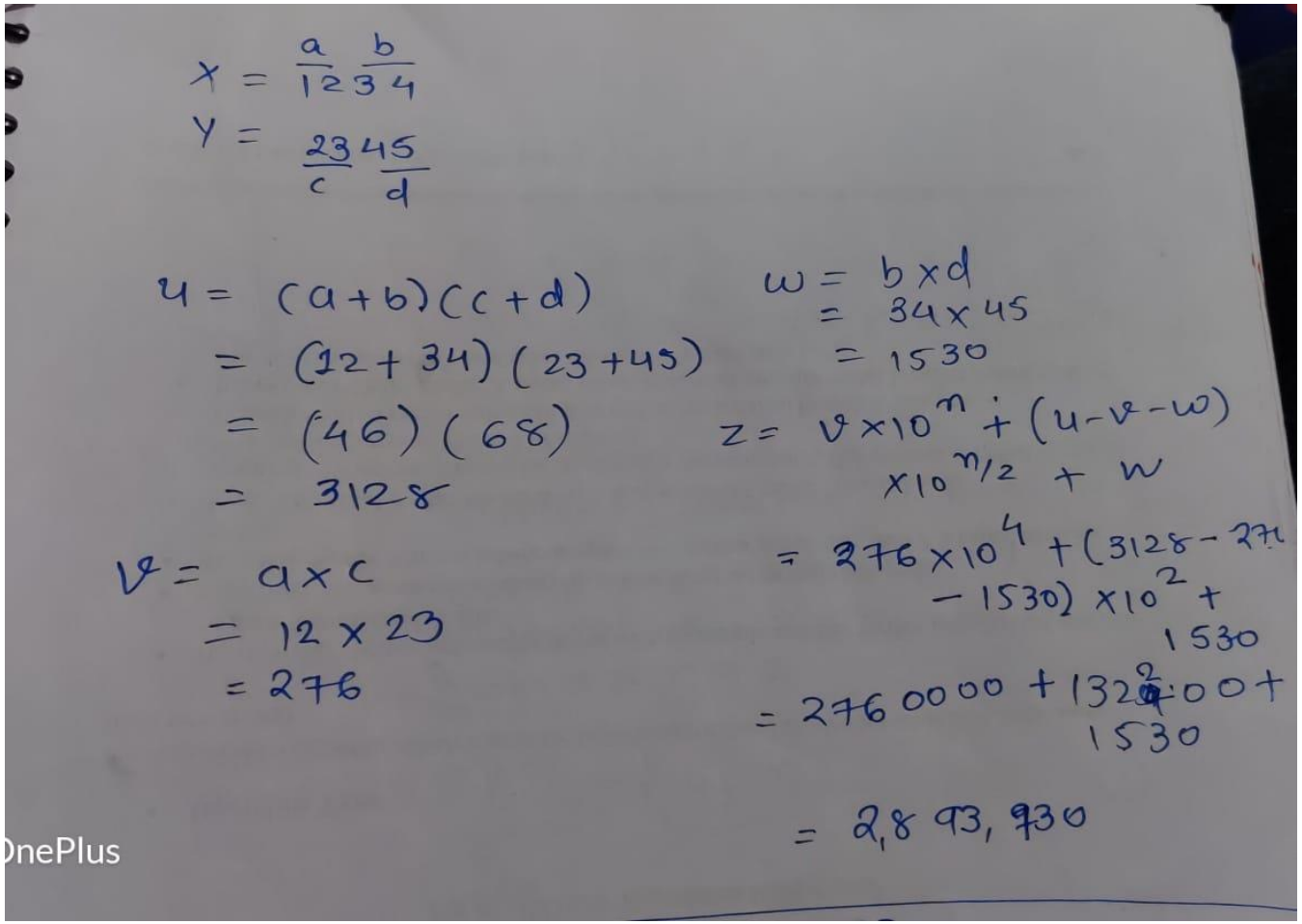
u= (a+b)(c+d)

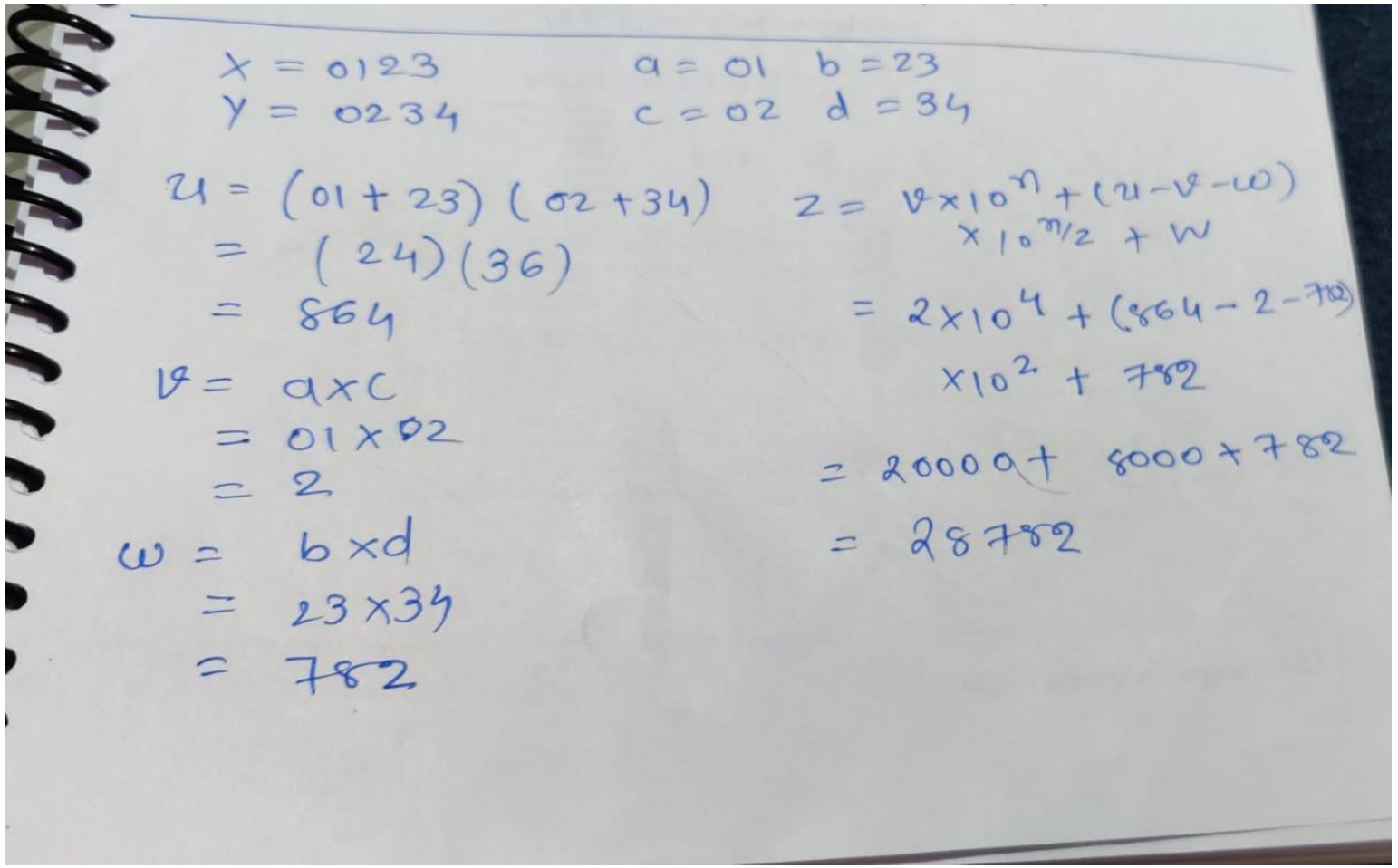
v=a*c

w=b*d

z= v* 10^n **+ (u-v-w) *** $10^{n/2}$ **+ w**

end


x = a b / 12 34
y = 23 45 / c d
u = (a+b)(c+d)
= (12+34)(23+45)
= (46)(68)
= 3128
v = a x c
= 12 x 23
= 276
w = b x d
= 34 x 45
= 1530
z = v x 10^n + (u-v-w) x 10^n/2 + w
= 276 x 10^4 + (3128 - 276 - 1530) x 10^2 + 1530
= 2760000 + 132800 + 1530
= 2,893,930
OnePlus


x = 0123
y = 0234
a = 01 b = 23
c = 02 d = 34
u = (01 + 23) (02 + 34)
= (24)(36)
= 864
v = a x c
= 01 x 02
= 2
w = b x d
= 23 x 34
= 782
z = v x 10^n + (u - v - w) x 10^n/2 + w
= 2 x 10^4 + (864 - 2 - 782) x 10^2 + 782
= 20000 + 8000 + 782
= 28782

Time complexity

begin

u= (a+b)(c+d)

v=a*c

w=b*d

z= v* 10^n + (u-v-w) * $10^{n/2}$ + w

End

divide part-
Total 3 multiplication –
number of sub-problem is 3

Shifting, addition, subtraction– conquer part

Divide 1 digit into 2 parts so size of sub problem is n/2

So our recurrence equation is

$T(n) = 3\,T(n/2) + k_1 n$ ------- eqn (1)

Time complexity

- Solve this equation by backward substitution

n=n/2 in eqn 1

$T(n/2) = 3T(n/4) + k_1 n/2$ ------(2)

Put eqn(2) in eq(1)

$T(n) = 3(3T(n/4) + k_1 n/2) + k_1 n$

$= 3^2T(n/4) + k_1 n (3/2) + k_1 n$ -------(3)

Put n=n/4 in eqn (1)

$T(n/4) = 3T(n/8) + k_1 n/4$ ------(4)

Put eqn(4) in eq(3)

- $= 3^3T(n/2^3) + k_1 n (3/2)^2 + k_1 n(3/2) + k_1 n$ -------(5)

(upto k we have to find)

$= 3^kT(n/2^k) + k_1 n [(3/2)^2 + (3/2) + 1]$

Put $n=2^k$

$= 3^kT(n/n) + k_1 n [(1 + (3/2) + (3/2)^2 + \text{----} + (3/2)^{k-1}]$

Time complexity

$= 3^k T(n/n) + k1n\,[(1 + (3/2) + (3/2)^2 + \text{----} + (3/2)^{k-1}]$

to solve series use this equation $a = \frac{r^n - 1}{r-1}$

r = 3/2 , n=k a =k1n

$= 3^k T(1) + k1n \frac{(3/2)^k - 1}{(3/2)-1}$ (put T(1)=1, $n = 2^k$)

$= 3^k\ k + 2k1n(3^k / 2^k) - 2k1n$

$= 3^k\ k + 2k1n(3^k /n) - 2k1n$

$= 3^k\ k + 2k1(3^k) - 2k1n$

$= 3^k\ (k + 2k1) - 2k1n$ ($n = 2^k$ hence k =log n)

$= 3^{\log n}\ (k + 2k1) - 2k1n$

Now , $3^{\log n} = n^{\log 3} = n^{1.58}$

$= n^{1.58}\,(k + 2k1) - 2k1n^1$

$T(n) = n^{1.58}$

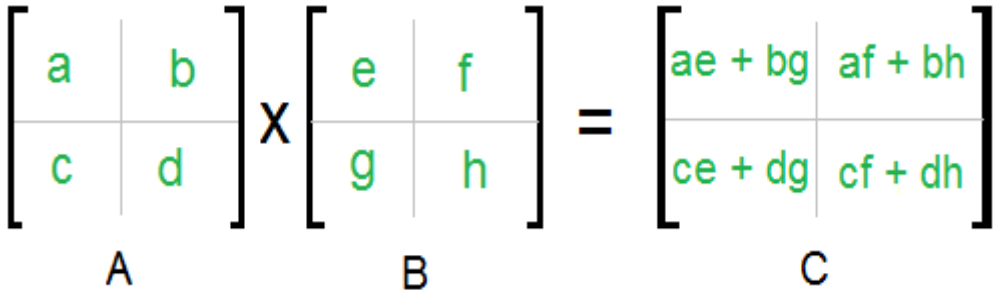
Max Min algorithm

```
Max_min(A, min, max, low, high)
        {
        if(low=high) then
                max= min= A[low];
        else if(low= high-1) then
        {
                if(A[low]<A[high]) then
                {
                        max = A[high];
                        min = A[low];
                }
        else
                {
                        max = A[low];
                        min = A[high];
                }
        }
```

Max Min algorithm

```
Else
{
        mid = (low+high)/2;

        Maxmin(low, mid, max, min);
        Maxmin(mid+1, max1, min1, high);

        if(max< max1) then max=max1;
        if(min> min1) then min=min1;
}
}
```

Matrix multiplication



A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

- In the above method, we do 8 multiplications for matrices of size N/2 x N/2 and 4 additions. Addition of two matrices takes $O(N^2)$ time.

- The idea of **Strassen's method** is to reduce the number of recursive calls to 7. Strassen's method is similar to above simple divide and conquer method in the sense that this method also divide matrices to sub-matrices of size N/2 x N/2 as shown in the above diagram, but in Strassen's method, the four sub-matrices of result are calculated using following formulae.

- Addition and Subtraction of two matrices takes $O(N^2)$ time. So time complexity can be written as

- **$T(n) = 7T(n/2) + O(n^2)$**
- From Master's Theorem, time complexity of above method is $O(n^{Log7})$ which is approximately $O(n^{2.8074})$

Merge sort

```
Mergesort(A[o,--,n-1], low, high)
If(low< high) then
        {
                mid = (low+high)/2
                mergesort(A, low, mid)
                mergesort(A, mid+1, high)
                combine(A , low, mid, high)

        }
Combine(A[o,---,n-1], low, mid, high)
{
        k = low       #new temp array
        i =low        #for left sub list
        j= mid+1      #for right sub list
```

```
While(i<= mid AND j <= high)do
        {
            if(A[i]= A[j]) then
            {
                temp[k] = A[i]
                i= i+1
                k = k+1
            }
            else
            {
                temp[k] = A[j]
                j= j+1
                k = k+1
            }
        }
```

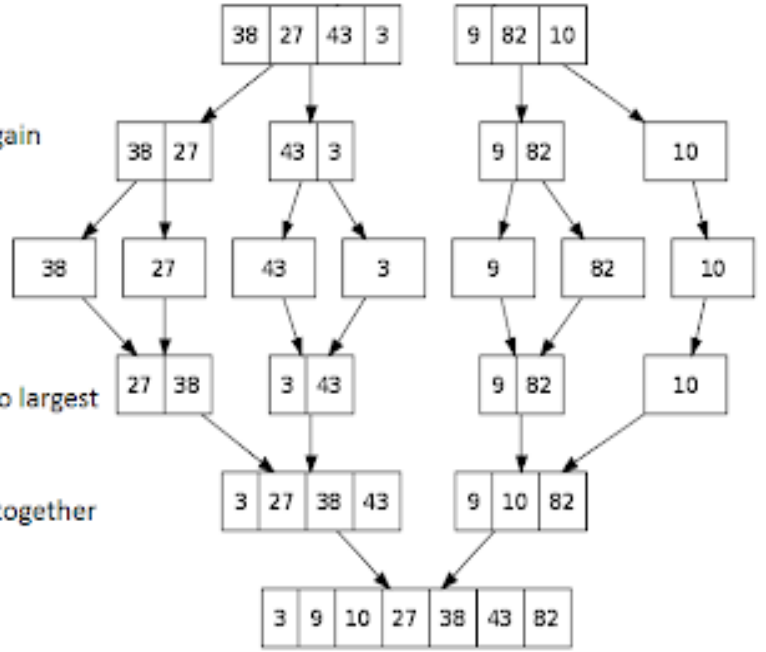
```
While(i<= mid) do
        {
                temp[k] = A[i]
                i= i+1
                k = k+1
        }

While(j<= high) do
        {
                temp[k] = A[j]
                j= j+1
                k = k+1

        }
}
```

When there is only one element left in left sub tree or right tree

Example of merge sort

How MergeSort Algorithm Works Internally

1. Divide the array into two parts
2. Divide the array into two parts again
3. Break each element into single parts
4. Sort the elements from smallest to largest
5. Merge the divided sorted arrays together
6. The array has been sorted



Quick sort Algortihm

Quicksort(**A[0,--,n-1], low, high)**

If(low< high) then

{

m = partition(A[low….high])
quicksort(A [low….m-1]))

quicksort(A [m+1….high]))

}

Quick sort Algortihm

Partition(A[low….high])

```
Pivot = A[low]
i= low
J= high
While(i<= j)do
{
        while(A[i]<= pivot)do
        i= i+1
        while(A[j>= pivot)do
        j= j+1
        if (i<= j) then
        swap(A[i], A[j])

}
Swap(A[low], A[j])    # (if j<i)
Return j
```

Quick sort Algortihm

- First of all we take first element and place it at its proper place. We call this element **Pivot** element.
- **Note:** We can take any element as **Pivot** element but for convenience the first element is taken as **Pivot.**
- There are two condition to place **Pivot** at its proper place.
- All the elements to the left of **Pivot** element should be smaller than
- All the elements to the right of **Pivot** element should be greater than
- Now we are having two sub part:
- **Quick sort Left sub tree**
- **Quick sort right sub tree**

Time complexity of quick sort and merge sort

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

  $a=2, b=2\ f(n)= n^1 \log^0 p$

  $\log_b a = \log_2 2 = 1 \quad k=1\ p=0$

  case : 2

  $T(n) = \theta(n \log n)$

Exponential

Let a and n be two integers.

We wish to compute $x = a^n$.

If n is small, the obvious algorithm is:

```
Function expoSeq(a, n) {
        r ← a
        for i ← 1 to n-1 do r ← a*r
        return r
}
```

$T(n) = O(n)$

Divide and conquer method

With divide and conquer:

a^n = return a if n = 1

return $(a^{n/2})^2$ if n is even

return $a * a^{n-1}$ otherwise

$a^{29} = a * a^{28} = a * (a^{14})^2$ …….

Which involves only three multiplications and four squaring instead of the 28 multiplication.

$a^{29} = a * a^{28}$

$= a * (a^{14})^2$

$= a * ((a^7)^2)^2$

$= a * (a * (a^6)^2)^2$

$= a * (a * ((a^3)^2)^2)^2$

$= a * (a * ((a * a^2)^2)^2)^2$

Which involves only three multiplications and four squaring instead of the 28 multiplication

PSEUDOCODE

```
Function expdc(a,n)
{
        if n = 1 then
                return a
        if n is even then
                return (expdc(a^(n/2))^2 )

      else
                return( a * expdc(a^(n-1)))
}
```

$T(n) = O(\log n)$