apsara

Date: __.__.__

ⓒ DIVIDE AND CONQUER

one of the many strategies used to solve a problem.

strategy → approach (or) design for solving a problem. [computational problem]

P of size = n given

$P_1 \quad P_2 \quad P_3 \quad \cdots \quad P_k$ [broken into subproblems]

$\downarrow \quad \downarrow \quad \downarrow \quad \quad \downarrow$

$S_1 \quad S_2 \quad S_3 \quad \cdots \quad S_k$ (solutions)

S (solution to main problem)

subproblems must be of the same type but of smaller size than the original problem.

these are recursive in nature.

subproblems should be able to combine themselves to generate solution of the original parent problem

↳ else don't use.

General Divide and conquer Approach

```
DAC (P)
{
    if ( small (P))
        S(P); → solve directly
    else
    {
        divide P into P1, P2, ..., Pk
        apply DAC (P1), DAC (P2) ...
        combine DAC (P1), DAC (P2) ...
    }
}
```


Topics using Divide and conquer strategy

1. Binary search
2. Finding maximum and minimum
3. Merge sort
4. Quick sort
5. Strassen's Matrix Multiplication

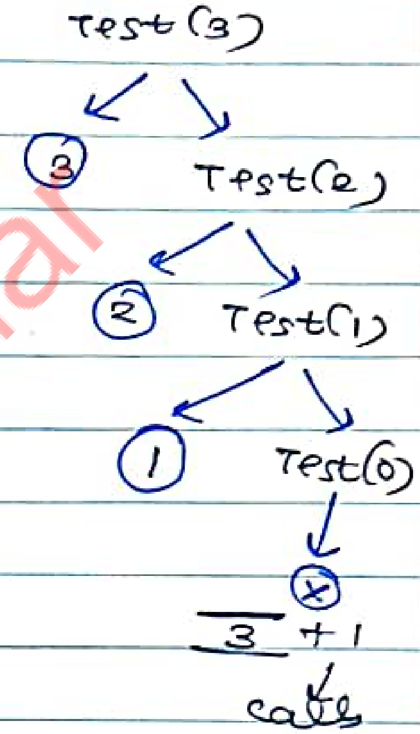
Recurrance Relations

Question → Get Reccurrance Relation → solve it to get time complexity

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

T(n) ←
```
void Test (int n)
{
    if (n>0)
    {
        printf("%d", n);
        Test (n-1);
    }
}
```
① ← printf("%d", n);
T(n-1) ← Test (n-1);

Tracing Tree



printf → n times
calls → n+1 times

$f(n) = n+1$ calls $\Rightarrow$ $O(n)$

Preparing recurrence relation :

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

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

Solving by back substitution

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

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

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

$\vdots$

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

$T(\cancel{n-n}) = 1 \quad (T(0))$

↓ adding all eqn

$T(n) = n+1$

(or) For k times

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

$k = n \rightarrow$ k steps processed

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

$T(n) = T(0) + n$

$T(n) = n+1 \rightarrow \Theta(n)$

✱ $T(n) = T(n-1) + n$

```
void Test (int n) → T(n)
{
  1 ← if (n>0)
      {
          for (int i=0; i<n; i++)  → n+1
              printf ("%d", n);  → n
          Test (n-1);  → T(n-1)
      }
}
```

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

taking asymptotic form

$T(n) = T(n-1) + n \rightarrow [O(n)]$

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

solving using recursion Tree

$T(n)$

$\underbrace{n}$ $\quad T(n-1)$

$\underbrace{n-1}$ $\quad T(n-2)$

Adding all times

$\underbrace{n-2}$ $\quad T(n-3)$

$\vdots$

$T(1)$

$T(n) = n + (n-1) + \dots + 1 + 0$ $\quad \underbrace{1}$ $\quad T(0)$

$\boxed{T(n) = \frac{n(n+1)}{2} \hookrightarrow \Theta(n^2)}$ $\qquad \cancel{0}\ 0$

solving using Back substitution

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

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

$\cancel{T(n-2)} = \cancel{T(n-3)} + (n-2)$

$\vdots$

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

$\vdots$

$\cancel{T(1)} = T(0) + 1$

$T(n) = 0 + 1 + 2 + \dots + n = 1 + [n(n+1)/2] \rightarrow$ same ans

☆ $T(n) = T(n-1) + \log n$

```
void test (int n)      → T(n)
{
    if (n>0)
    {
        for (i=1; i<n; i=i*2)
        {   printf ("%d", i);   → log n
        }
        Test (n-1);   → T(n-1)
    }
}
```

$$T(n) = \begin{cases} 1 & \forall \; n=0 \\ T(n-1) + \log n & \forall \; n>0 \end{cases}$$

solving using recursion tree

$T(n)$ → $\log n$, $T(n-1)$

$T(n-1)$ → $\log(n-1)$, $T(n-2)$

⋮

$T(2)$ → $\log 2$, $T(1)$

$T(1)$ → $\log 1$, $T(0)$

$T(0)$ → 1

Total :

$T(n) = \log n$
$+ \log(n-1)$
$+ \log(n-2)$
$+ \ldots + \log 2$
$+ \log 1$

$T(n) = \log[(n)(n-1) \cdots (2)(1)]$

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

$\log(n!) < \log(n^n) \rightarrow n \log n$

$\boxed{O(n \log n)}$

$T(n) = T(n-1) + 1 \longrightarrow O(n)$

$T(n) = T(n-1) + n \longrightarrow O(n^2)$

$T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$

similarly

$T(n) = T(n-1) + n^2 \longrightarrow O(n^3)$

$T(n) = T(n-2) + 1 \longrightarrow n/2,\ O(n)$

$T(n) = T(n-100) + n \longrightarrow O(n^2)$

But coefficient appears

$T(n) = 2T(n-1) + 1 \longrightarrow$ answer changes

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

Algorithm Test (int n) $\rightarrow T(n)$

```
{     if (n>0)
      {
            printf("%d", n);   → 1
            Test(n-1);         → T(n-1)
            Test(n-1);         → T(n-1)
}     }
```

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

Recurrance relation : $T(n) = \begin{cases} 1 & \forall\ n = 0 \\ 2T(n-1) + 1 & \forall\ n > 0 \end{cases}$

solving using Recursion Tree :

$T(n)$ → ①

1 $\quad T(n-1) \quad T(n-1)$ → ②

1 $\quad T(n-2) \quad T(n-2) \quad$ 1 $\quad T(n-2) \quad T(n-2)$ → ④

⋮

2^k time

Total time

$= 1 + 2 + 4 + \dots + 2^k$

$= 2^{k+1} - 1$ [sum of terms of a Geometric Progression]

Assume $n - k = 0$, $n = k$

$T(n) = 2^{n+1} - 1 \rightarrow \boxed{O(2^n)}$

solving using Back Substitution

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

$\cancel{2T(n-1)} = 4\cancel{T(n-2)} + 2$

$\cancel{4T(n-2)} = 8\cancel{T(n-3)} + 4$

⋮

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

Assuming $n-k = -1$ $\quad k = n-1$

$$2^{n-2} T(1) = 2^{n-1} \cancel{T(0)} + 2^{n-1}$$

Adding all equations,

$$T(n) = 1 + 2 + 4 + \dots + 2^{n-1} + 2^{n}$$

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

$$\hookrightarrow O(2^n)$$

MASTER'S THEOREM FOR DECREASING FUNCTIONS

$$T(n) = T(n-1) + 1 \rightarrow O(n)$$

$$T(n) = T(n-1) + n \rightarrow O(n^2)$$

$$T(n) = T(n-1) + \log n \rightarrow O(n \log n)$$

$$T(n) = 2T(n-1) + 1 \rightarrow O(2^n)$$

$$T(n) = 3T(n-1) + 1 \rightarrow O(3^n)$$

$$T(n) = 2T(n-1) + n \rightarrow O(n 2^n)$$

$$\boxed{\begin{array}{l} T(n) = aT(n-b) + f(n) \\ a > 0, \; b > 0, \; f(n) = O(n^k) \\ \text{where } k \geq 0 \end{array}}$$

general form of a recurrence relation

apsara

Date: __.__.__

For $a = 1$

Time complexity = $\boxed{O(n \cdot f(n))}$

For $a > 1$

Time complexity = $\boxed{O(a^{n/b} f(n))}$

For $a < 1$

Time complexity = $\boxed{O(f(n))}$

Dividing Functions

```
Algorithm test (int n)      → T(n)
{
    if (n>1)
    {   printf ("%d", n);    → 1
        test (n/2);          → T(n/2)
    }
}
```

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

Recurrance Relation $T(n) = \begin{cases} 1 & \forall \; n = 1 \\ T(n/2) + 1 & \forall \; n > 1 \end{cases}$

solving using recursion tree

$T(n)$

$1 \quad T(n/2^1)$

$1 \quad T(n/2^2)$

$\vdots$

$T(n/2^k)$

Assuming $\frac{n}{2^k} = 1$

$2^k = n$

$k = \log_2 n$

Adding all steps

$T(n) = 1 + 1 + \ldots \ldots \; k$ times

$T(n) = k \qquad T(n) = \log_2 n$

$\boxed{O(\log_2 n)}$

Algorithm Test (int n)

```
{
    if (n > 1)
    {
        for (int i = 1; i <= n; i++)
            printf ("%d", i);
        Test (n/2);
    }
}
```

Recurrence Relation : $T(n) = \begin{cases} 1 & \forall \; n = 1 \\ T(n/2) + n & \forall \; n > 1 \end{cases}$

Solving using recursion tree

$T(n)$

$T(n/2) \quad n$

$T(n/4) \quad n/2$

$\vdots$

$T(n/2^k)$

Assuming

$$\frac{n}{2^k} = 1$$

$$2^k = n$$

$$k = \log_2 n$$

Solving all steps

$$T(n) = n + \frac{n}{2} + \dots \frac{n}{2^{k-1}}$$

$$T(n) = n \cdot \left[\frac{1 - (1/2)^k}{1 - (1/2)}\right]$$

$$T(n) = 2n\left[1 - \frac{1}{2^k}\right] = 2n\left[1 - \frac{1}{n}\right]$$

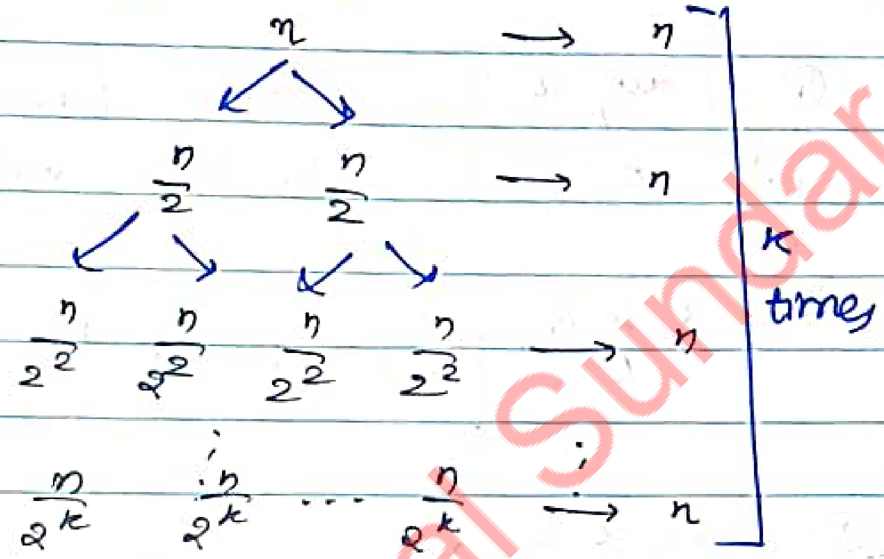
$$T(n) = 2n - 1 \longrightarrow \boxed{O(n)}$$

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

```
void test (int n) → T(n)
{
    if (n > 1)
    {
        for (i = 0; i < n; i++)
            stmt; → n
        test (n/2); → T(n/2)
        test (n/2); → T(n/2)
    }
}
```

Recurrance Relation : $T(n) = \begin{cases} 1 & \forall \; n = 1 \\ 2T(n/2) + n & \forall \; n > 1 \end{cases}$

solving using Recursion tree



nk

$\downarrow$

$O(nk)$

$\frac{n}{2^k} = 1$ is the last limiting case

$2^k = n \rightarrow k = \log_2 n$

$O(nk) = \boxed{O(n \log_2 n)}$

time complexity.

solving using Equations.

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

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

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

$\vdots$

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

Adding $T(n) = 2^k + n + \dots + n = (k+1)n$ times

MASTER'S THEOREM FOR DIVIDING FUNCTIONS

$$\boxed{\begin{array}{l} T(n) = a\,T(n/b) + f(n) \\ a \geq 1,\ b > 1,\ f(n) = O(n^k \log^p n) \end{array}}$$

① $\log_b a$
② k

} 3 cases based on these values

case ① $\log_b a > k$

$\hookrightarrow$ $\boxed{O(n^{\log_b a})}$

case ② $\log_b a = k$

$\hookrightarrow$ 3 cases again

$p > -1 \rightarrow O(n^k \log^{p+1} n)$

$p = -1 \rightarrow O(n^k \log \log n)$

$p < -1 \rightarrow O(n^k)$

case ③ $\log_b a < k$

$\hookrightarrow$ 2 cases here

$p \geq 0 \rightarrow O(n^k \log^p n)$

$p < 0 \rightarrow O(n^k)$

[1 + 3 + 2
$\hookrightarrow$ total of 6 cases]

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

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

$k = 0$, $p = 0$

$\log_b a = 1$, $k = 0$, $1 > 0$

$O(n^1) \Rightarrow O(n)$

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

$a = 4$, $b = 2$, $f(n) = O(n) = O(n^1 \log^0 n)$

$k = 1$, $p = 0$

$\log_b a = 2$, $k = 1$, $2 > 1$

$O(n^2) \Rightarrow O(n^2)$

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

$a = 8$, $b = 2$, $f(n) = O(n^2) = O(n^2 \log^0 n)$

$k = 2$, $p = 0$

$\log_b a = 3$, $k = 2$, $3 > 2$

$O(n^3) \Rightarrow O(n^3)$

$T(n) = 9T(n/3) + 1$

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

$K = 0$, $p = 0$

$\log_b a = 2$, $k = 0$, $2 > 0$

$O(n^2) \rightarrow O(n^2)$

apsara

Date:

$T(n) = 9T(n/3) + n^2$

$a = 9,\ b = 3,\ f(n) = O(n^2) = O(n^2 \log^0 n)$

$k = 2,\ p = 0$

$\log_b a = 2,\ k = 2 \quad 2 = 2$

Now checking $p = 0,\ 0 > -1$

$\hookrightarrow O(n^2 \log^{0+1} n) = O(n^2 \log n)$

$T(n) = 8T(n/2) + n \log n$

$f(n) = O(n \log n) = O(n^1 \log^1 n)$

$a = 8,\ b = 2 \qquad k = 1,\ p = 1$

$\log_b a = 3 \qquad 3 > 1 \rightarrow$ case 1

$O(n^3)$

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

$f(n) = O(n) = O(n^1 \log^0 n)$

$k = 1,\ p = 0$

$a = 2,\ b = 2$

$\log_b a = 1,\ k = 1,\ 1 = 1$

Now checking $p = 0,\ 0 > -1$

$\hookrightarrow O(n^1 \log^{0+1} n) = O(n \log n)$

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

$\log_2 4 = 2$, $K = 2$ equal

and $p = 0 > -1$

$\downarrow$

$\boxed{O(n^2 \log n)}$

$T(n) = 4T(n/2) + n^2 \log n$

$\hookrightarrow$ $\boxed{O(n^2 \log^2 n)}$

$T(n) = 4T(n/2) + n^2 \log^2 n$

$\hookrightarrow$ $\boxed{O(n^2 \log^3 n)}$

$T(n) = 8T(n/2) + n^3$

$\log_2 8 = 3$, n^3 equal

$\downarrow$

$\boxed{O(n^3 \log n)}$

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

$\log_2 2 = 1$, $K = 1 \rightarrow$ case 2

$p = -1 \longrightarrow$ case (ii)

$\downarrow$

$\boxed{O(n \log \log n)}$

apsara

Date:

$T(n) = 2T(n/2) + n/\log^2 n$

$\log_2 2 = 1 \quad n^1 \Rightarrow 1$ equal

$p = -2 < -1 \longrightarrow$ too small ignore

$\boxed{O(n)}$

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

$\log_2 1 = 0 < 2 \rightarrow$ case 3

$O(n^2)$

$T(n) = T(n/2) + n^2 / \log^2 n$

$\hookrightarrow O(n^2 / \log^2 n)$

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

$\log_2 4 = 2 < 3$

and $p = -1 < 0$

ignore

$O(n^3)$

MORE EXAMPLES:

case ①

$T(n) = 2T(n/2) + 1 \longrightarrow O(n)$

$T(n) = 4T(n/2) + 1 \longrightarrow O(n^2)$

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

apsara

Date: __.__.__

$T(n) = 8T(n/2) + n^2 \rightarrow O(n^3)$

$T(n) = 16T(n/2) + n^2 \rightarrow O(n^4)$

case ③

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

$T(n) = 2T(n/2) + n^2 \rightarrow O(n^2)$

$T(n) = 2T(n/2) + n^2 \log n \rightarrow O(n^2 \log n)$

$T(n) = 4T(n/2) + n^3 \log^2 n \rightarrow O(n^3 \log^2 n)$

$T(n) = 2T(n/2) + n^2/\log n \rightarrow O(n^2)$

case ②

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

$T(n) = 2T(n/2) + n \rightarrow O(n \log n)$

$T(n) = 2T(n/2) + n \log n \rightarrow O(n \log^2 n)$

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

$T(n) = 4T(n/2) + (n \log n)^2 \rightarrow O(n^2 \log^3 n)$

$T(n) = 2T(n/2) + n/\log n \rightarrow O(n \log\log n)$

$T(n) = 2T(n/2) + n/\log^2 n \rightarrow O(n)$

Root Function

```
void test (int n)      → T(n)
{    if (n > 2)
     {      stmt;       → 1
            test (√n);  → T(√n)
     }
}
```

Recurrance Relation $T(n) = \begin{cases} 1 & \forall \; n = 2 \\ T(\sqrt{n}) + 1 & \forall \; n > 2 \end{cases}$

solving using substitution & Equations

$$T(n) = \cancel{T(n^{1/2})} + 1$$
$$\cancel{T(n^{1/2})} = \cancel{T(n^{1/4})} + 1$$
$$\cancel{T(n^{1/4})} = \cancel{T(n^{1/8})} + 1$$
$$\vdots$$
$$\cancel{T(n^{1/2^{k-1}})} = T(n^{1/2^k}) + 1$$
$$\vdots$$

Adding all equations,

$$T(n) = T(n^{1/2^k}) + k$$

Assuming $n = 2^m$

$$T(2^m) = T(2^{m/2^k}) + k$$

Assuming $T(2^{m/2^k}) = T(2^1)$

$$\frac{m}{2^k} = 1, \quad m = 2^k \text{ and } k = \log_2 m$$

since $n = 2^m$ and $m = \log_2 n$

$O(k) \rightarrow O(\log_2 \log_2 n)$

STRASSEN'S MATRIX MULTIPLICATION

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

2 × 2   2 × 2   2 × 2

$$c_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

```
for ( i = 0; i < n; i++)
{
    for ( j = 0; j < n; j++)
    {
        c[i, j] = 0;
        for ( k = 0; k < n; k++ )
        {
            c[i, j] = A[i, k] * B[k, j];
        }
    }
}
```

$\boxed{O(n^3)}$

small problem → (dim ≤ 2)

$$\left.\begin{aligned} c_{11} &= a_{11} b_{11} + a_{12} b_{21} \\ c_{12} &= a_{11} b_{12} + a_{12} b_{22} \\ c_{21} &= a_{21} b_{11} + a_{22} b_{21} \\ c_{22} &= a_{21} b_{12} + a_{22} b_{22} \end{aligned}\right\}$$ matrix multiplication

Direct formulae

8 units of time = $O(1)$

Date: __.__.__

For dimensions ≥ 3

Apply divide and conquer and divide into smaller subproblems.

A_{11}, A_{12}, A_{21}, A_{22}

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right]$$

4×4

B_{11}, B_{12}, B_{21}, B_{22}

$$B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right]$$

4×4

Algorithm:

```
Algorithm MM (A, B, n)
{
    if (n ≤ 2)
    {   c = 4 formulae
    }
    else
    {
        mid -- n/2
same 4      { MM( A11, B11, n/2) + MM(A12, B21, n/2)
formulae    { MM( A11, B12, n/2) + MM(A12, B22, n/2)
    }           :
}
```

Recurrance Relation

$$T(n) = \begin{cases} 1 & \forall \ n \le 2 \\ 8T(n/2) + n^2 & \forall \ n > 2 \end{cases}$$

$$T(n) = \underbrace{8\,T(n/2)}_{\text{recursion}} + \underbrace{n^2}_{\text{matrix addition}}$$

$$\log_2 8 = 3, \quad 3 > 2 \rightarrow \boxed{O(n^3)}$$

Strassen's Approach

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22})\,B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12})\,B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{21} + B_{22})$

And finally

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

7 multiplications only instead of 8 times

Recurrance Relation

$$T(n) = \begin{cases} 1 & \forall \quad n \le 2 \\ 7T(n/2) + n^2 & \forall \quad n > 2 \end{cases}$$

$$\log_2 7 = 2.81 > 2$$

$$\longrightarrow \boxed{O(n^{2.81})}$$

reduced time complexity

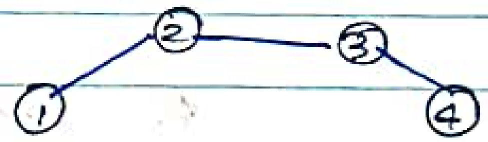
DISJOINT SETS

1. Disjoint sets and operations
2. Detecting a cycle
3. Graphical Representation
4. Array Representation
5. weighted union and collapsing Find

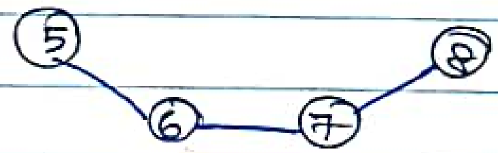
(1) Find (2) Union are basic operations

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{5, 6, 7, 8\}$



Disjoint : $S_1 \cap S_2 = \phi$



$S_3 = S_1 \cup S_2 = \{1, 2, 3, 4, 5, 6, 7, 8\}$

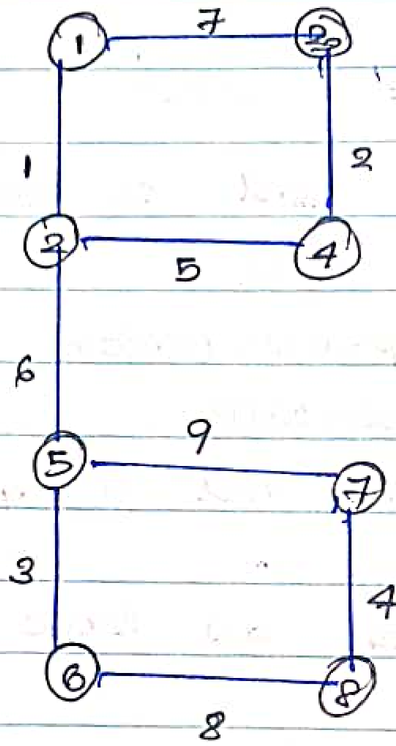
Find $(u, v) = (4, 8)$ connect it

Find $(1, 5)$: both belong to same set

If connected → cycle is formed.

Detecting cycle in a graph

use of disjoint sets is best



$U = \{1, 2, 3, 4, 5, 6, 7, 8\}$

- Edge $(1, 2)$ $w = 1$

  $\boxed{1}$ $S_1 = \{1, 2\}$

- Edge $(3, 4)$ $w = 2$

  $\boxed{2}$ $S_2 = \{3, 4\}$

- Edge $(5, 6)$ $w = 3$

  $\boxed{3}$ $S_3 = \{5, 6\}$

- Edge $(7, 8)$ $w = 4$

  $\boxed{4}$ $S_4 = \{7, 8\}$

- Edge $(2, 4)$ $w = 5$

  $\boxed{5}$ $S_5 = S_1 \cup S_2$

  $S_5 = \{1, 2, 3, 4\}$

- Edge $(2, 5)$ $w = 6$

  $\boxed{6}$ $S_6 = S_5 \cup S_3$

  $S_6 = \{1, 2, 3, 4, 5, 6\}$

- Edge $(1, 3)$ $w = 7$

  Belong to same set → cycle (x)

- edge $(6, 8)$ $w = 8$

  $S_7 = S_6 \cup S_4$

  $\boxed{7}$ $S_7 = \{1, 2, 3, 4, 5, 6, 7, 8\}$

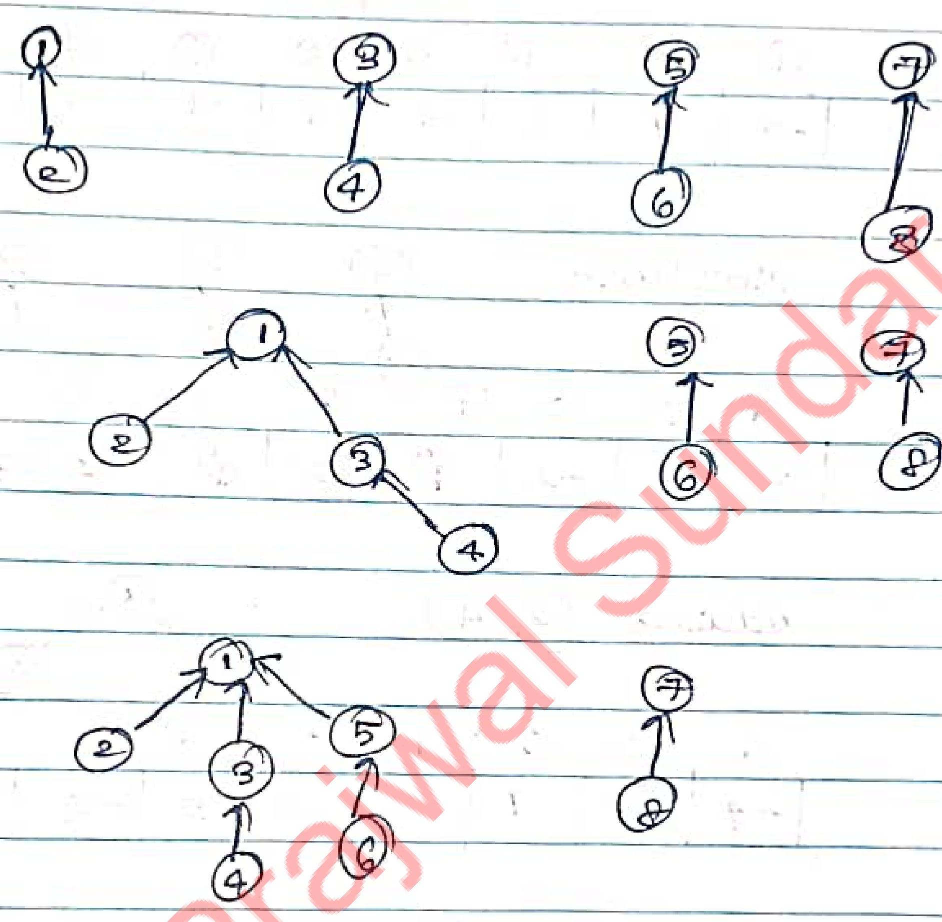
MST formed with 7 edges

- edge $(5, 7)$ $w = 9$

  Belong to same set → cycle (x)

Date: __.__.__

Graphical Representation of a Disjoint set



Array Representation of a Disjoint set

U = {1, 2, 3, 4, 5, 6, 7, 8}

1, 2 3 4 5 6 7 8

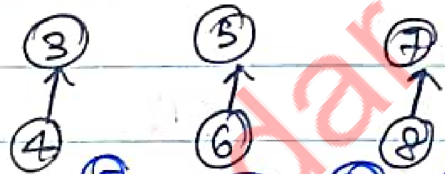
Parent							
-1	-1	-1	-1	-1	-1	-1	-1

Each node is its own parent

1 ∪ 2

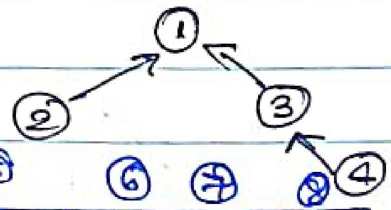
1	2	3	4	5	6	7	8
-2	1	-1	-1	-1	-1	-1	-1

similarly



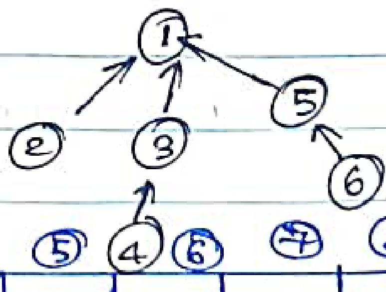
1	2	3	4	5	6	7	8
-2	1	-2	3	-2	5	-2	7

Adding (2, 4)



1	2	3	4	5	6	7	8
-4	1	1	3	-2	5	-2	7

Adding (2, 5)

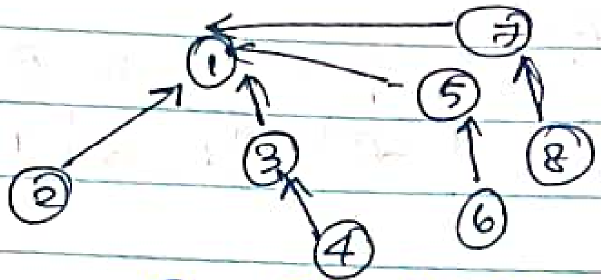


1	2	3	4	5	6	7	8
-6	1	1	3	1	5	-2	7

Adding (1, 3) → same parent

Including (1, 3) forms a cycle
↳ Don't include it

Adding (6, 8)



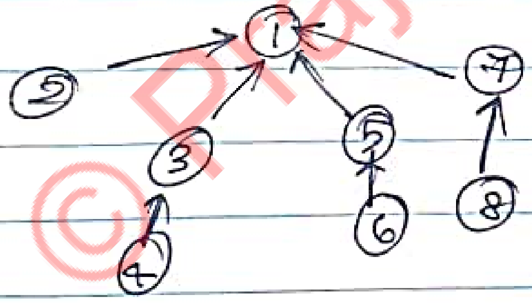
1	2	3	4	5	6	7	8
-8	1	1	3	1	5	1	7

Adding (5, 7) → same parent

Including (5, 7) forms a cycle

↳ don't include it.

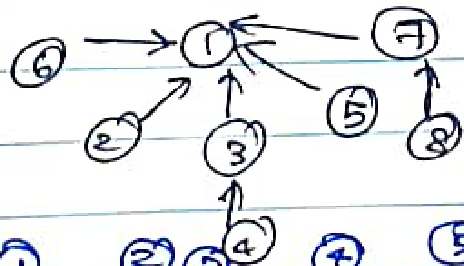
collapsing find :



while finding parent of 6 → goto 5 then 1.

Now we know that 1 is the parent of 6

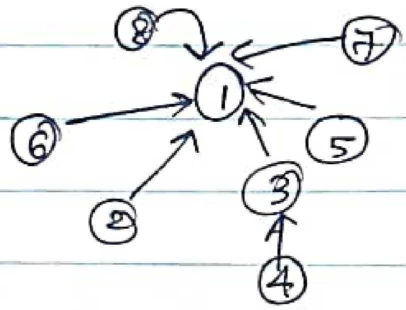
so



establish a direct connection

1	2	3	4	5	6	7	8
-8	1	1	3	1	1	1	7

find parent of 8 : 1



1	2	3	4	5	6	7	8
-8	1	1	3	1	1	1	1

Now the time needed to access the parent drastically reduces. almost to O(1).