★ BFS ⇒ Queue Complexity ⇒ $O(V+E)$

- Produces Shortest Paths (in terms of the minimum number of edges) to all reachable vertex from the source s.

★ DFS ⇒ Stack Complexity ⇒ $O(V+E)$

Colors to keep track of progress

White : Undiscovered
Gray : Discovered
Black : Finished.

★ Shortest Path

$$\begin{cases} \min(w(p): u \overset{p}{\rightarrow} v\}; & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

Optimal Substructure property : subpath of shortest paths are also shortest paths.

$P_{ik} = \langle v_i, \ldots v_k \rangle$ v_i to v_k

$P_{ij} = \langle v_i, \ldots, v_j \rangle$ (subpath of P_{ik} v_i to v_j)

Proof : If subpath not shortest then can be replaced thus not shortest (contradiction)

Triangle Inequality

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

★ Directed Acyclic Graph (DAGs)
$O(V+E)$ by topological sort

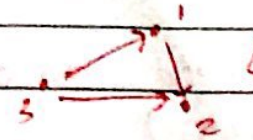
★ For shortest path with equal weight edge use BFS

★ Priority Queue

	→ dynamic unordered array
IsEmpty	→ dynamic unordered array with min index
add	→ Array in increasing order
peekmin	→ Array in decreasing order
removemin	→ Linkedlist in increasing order
	→ unordered linked list with reference to node with the minimum

★ Kosaraju's Algorithm

Strongly connected : Can reach to every point from every point



Can't reach from 2 to 3 not strongly connected.

⇒ Transpose of a graph : Reversing all the edges direction

Time Complexity Kosaraju Algorithm

Step 1 DFS // O(V+E)
Step 2 Create // O(V+E)
Computing G^T O(V+E)
Step 3 DFS // O(V+E)

★ Prims
Remove All loop
Remove parallel connections
Help moving selecting smallest without forming loop
(Greedy Algorithm)

Kruskal
Remove all loop.
Remove parallel connections
keep for connecting
Smallest without forming loop.

E log V + V log V
≈ E log V

★ 0/1 Knapsack Dynamic Programming (Dynamic Programming)
Have to take whole item or take none (can't take fractional.
Make a table while filling check
max (value of curr weight + T[pr row - 1][Total weight - curr wt.] ; value from top) (Please be careful while filling)
Now finall trace the items that are filled for that if value taken from above then not taken otherwise taken.
To check which element to check see from where the val came.
T[pr row - 1][tot wt - curr wt.].

★ Fractional Knapsack Problem (Greedy Algorithm)

Take $\frac{\text{Value}}{\text{weight}}$ in ascending order and at last fill the fraction left.

Total wt. = w

★ Integer Multiplication (Divide and Conquer (Karatsuba algo)

$T(n) = 3T(n/2) + O(n)$ $\quad O(n^{1.59})$

$XY = 2^n x_l y_l + 2^{n/2} \times [(x_l + x_r)(y_l + y_r)) - x_l y_l + x_r y_r$

★ Strassen's Matrix Multiplication method. (Div & Conq)

$T(n) = 7T(n/2) + O(N^2)$

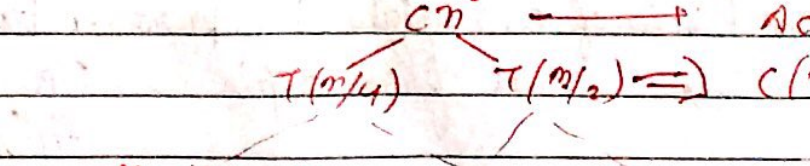
$O(N^{\log 7}) = O(N^{2.8074})$

(Not memorizing)

★ Algorithm analysis

⇒ Recurrence Tree method.

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



Master Method.

$T(n) = aT(n/b) + f(n) \quad a \geq 1 \quad b > 1$

⇒ $f(n) = \Theta(n^c) \quad c < \log_b^a \quad T(n) = \Theta(n^{\log_b^a})$

⇒ $f(n) = \Theta(n^c) \quad c = \log_b^a \quad T(n) = \Theta(n^c \log n)$

⇒ $f(n) = \Theta(n^c) \quad c > \log_b^a \quad T(n) = \Theta(f(n))$

★ Tromino Tiling (Divide & Conq)
$T(n) = 4T(n/2) + C$. $O(n^2)$

★ Randomized Quick Sort
Choose pivot element randomly.
and swap it with last element
Rest algo is same
Expected Worst Case $O(n \log n)$
Improves time complexity.

★ Huffman Coding
Sort frequency in increasing order
Select small 2 and make them
leaf node of a node with sum
as their value. Make tree
by this method
Less frequency left, High right.
0 to left 1 to right
Write in table.
$T(n) = n \log n$

★ matrix Chain Multiplication $O(n^3)$ (DP)

$q = m[i][k] + m[k+1][j] + p[i-1] * p[k] * p[j]$

$L = 0$ is chain length

Teacher's Signature

★ NP Hard and NP Complete Problem

P	NP
Deterministic	Nondeterministic
Polynomial	Polynomial
Algorithm	Algorithm

Complet Graph : All connected to all Clique : Subgraph of a graph is complete

Decision Problem : Require yes/no ans.

Optimization : What is the max-clique.

$(x_1, x_2) \wedge (\bar{x}_1, \bar{x}_2) \wedge (x_1, x_3)$

if 3 form then satisfiable formula.

★ Bellman Ford

Initialise every one with ∞

Start from node and ~~mark adj~~

change adjacent node distance if required

Switch to any other node. If path to that node is defined the repeat step 2 otherwise switch to other node.

Do this in any times

at last Shortest Distance found.

Teacher's Signature

DATE: / /
PAGE NO.:

★ Interval Scheduling (Greedy Algorithm)

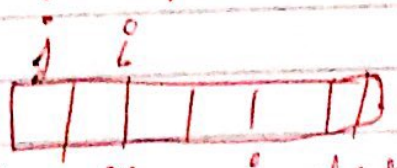
Sort all ascending order finishing time.
If activity compatible to A / then add that activity
$O(n \log n)$ → sort travel
$\log n$ n

★ Radix Sort — Sort one's place
ten's place
⋮

★ Longest Common Subsequence
write in up and side both string
If match diagonal mai jo hai + 1.
At last / If ai left or up same then taken from that

input [i] == input [j]
T[i][j] = T[i-1][j-1] + 1.
else max T[i-1][j], T[i][j-1]

★ Job Scheduling (DP)
finishing time in ascending order.



if no overlapp.
T[i] = max [T[i], T[j] + prof[i]]

for fix i check all j if overlap
if not add value in box to the value of present i if in box its more don't change otherwise change
Max value in arr is max value.

$O(n^2)$

Teacher's Signature

DATE : / /
PAGE NO :

★ Count inversion

while merging

~~temp[k++] =~~

if arr[i] ≥ arr[j] inv_count = inv_count + (mid - i);

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

★ closest pair of Point

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

Teacher's Signature