ALGORITHMS

1) TIME AND SPACE COMPLEXITY

- Every statement: 1 unit of time.
  Every var: 1 unit of space

Algo(A, B, n)

eg → Pg 27 | cond^n check → n+1 (imp)

Time Comp & Space Complexity.

- for(i=0; i<n; i++) → O(n)
- for(i=0; i<n; i=i+2) → O(n/2) (imp)

eg → Pg 230 | T = O(n^2)

~~eg~~ → Pg 231 | T = O(log n)

eg → Pg 232 (cond^n check) [table, n=6] ([log n]+1)

- for(i=1; i<n; i=i*2) → O(log n)
- for(i=1; i<n; i=i*3) → O(log_3 n)
- for(i=1; i*i<n; i++) → O(√n)
- for(i=1; i<log n; i=i*2) ⇒ O(log log n)

- for(i=1, j=n; i<=j; i=i*2, j=j/2) → O(log_2 n) (imp)

How many times Stmt gets executed? [given n]

eg → Pg 28 [Rev Copy] (Vimp)

* ASYMPTOTIC NOTATIONS

$[1 < \log n < n < n\log n < n^2 < n^3 \dots < 2^n < 3^n < \dots < n^n]$.

(imp)

- $\sqrt{n}$ → LB = $\Omega(\log n)$, UB = $O(n)$
- $n!$ → LB = $\Omega(1)$, UB = $O(n^n)$
- $\log n!$ → LB = $\Omega(1)$, UB = $O(n \log n)$

* If $f(n) = 2n+3$

UB → $O(n)$, $O(n^2)$, $O(n^3) \dots O(n^n)$, $O(\log n)$ X.

LB → $\Omega(n)$, $\Omega(\log n)$, $\Omega(1) \dots \Omega(n^2)$ X.

TLB = TUB = $O(n) = \Omega(n)$

if, TUB & TLB is same, avg bound exists (Θ exists) (imp)

eg → Pg 239 [Gate]

① Big O

Upper Bound of f(n)

$f(n) = O(g(n))$ iff : $f(n) \le c * g(n)$

- $g(n) >> f(n)$

② Omega

Lower Bound of f(n)

$f(n) = \Omega(g(n))$

eg: $f(n) = 2n+3$ → $\Omega(n)$, $\Omega(1)$, $\Omega(\log n)$

$f(n) >> g(n)$

without log: const don't matter
with log: const matters

③ Average

eg: $f(n) = 2n+3 \longrightarrow \Theta(n)$

(imp) → n! ka Θ dne; ∵ TUB = n^n, TLB = 1 & TUB ≠ TLB

* Properties of Asymptotic Notn:

1. if $f(n) = O(g(n))$, then $g(n) = \Omega(f(n))$
2. if $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, then $f(n) = \Theta(g(n))$
3. $f(n) = O(g(n))$; $g(n) = O(h(n))$ (imp) $f(n) = O(h(n))$.
4. If, $f(n) = O(g(n))$; $d(n) = O(e(n))$

(P4) → (i) $f(n) + d(n) = O(\max(g(n), e(n))$

(ii) $f(n) * d(n) = O(g(n) \cdot e(n))$.

* Comparing of functns

Q $f(n) = 2^{\sqrt{n}}$, $g(n) = n^{\log n}$.

$\log 2^{\sqrt{n}}$; $\log n^{\log n}$

⇒ $\sqrt{n}$, $\log^2 n$

⇒ $\frac{1}{2}\log n > 2\log(\log n)$

Q $f(n) = 3n^{\sqrt{n}}$, $g(n) = 2^{\sqrt{n}\log n} = 2^{n^{\sqrt{n}}}$.

(imp) → they are similar.

[constants don't matter, ∵ no log]

eg → Pg 234 [Q4] → check bigger values.

eg → Pg 33 [Rev Copy] (Q1)
Jay both order, after that constant do matter. [imp]

* No rel^n b/w case & asymptotic order:
Linear Search
- worst case = f(n) = n → Ω(n), Θ(n), O(n) — any order
[found at last]
- Best case: f(n) = 1 → Ω(1), Θ(1), O(1)
case.

(imp) eg → Pg 246 [BST]

* TIME COMPLEXITY OF RECURSIVE FUNCTIONS

2 types.

(i) Decreasing functions → dividing functions.

```
void fun(int n)
{ if(n>0)                    — 1
  { for(i=0; i<n; i++)       — n+1
    { printf(i);             — n.
    }
  } fun(n-1);                — T(n-1)
}
```

T(n) = T(n-1) + 2n + 2.
∴ T(n) = T(n-1) + n

Solve using back Substitution.

eg → Pg 35 [Sain] (imp)

Note: Take Stopping cond^n as 0, in case of decreasing funct^n: n - K = 0; n = K. T(0) = 1

T(n) = T(n-1) + 1 → O(n) (imp)
T(n) = T(n-1) + n² → O(n³)
T(n) = T(n-50) + n log n → O(n² log n)

eg → Pg 36, T(n) = 2T(n-1) + 1 → Θ(2^n)

(imp)
T(n) = 3T(n-1) + 1 → Θ(3^n)
T(n) = 2T(n-1) + n → Θ(n·2^n)
T(n) = T(n-1) + T(n-2) + 1 → Θ(2^n) [not proved]

* Master Theorem for decreasing funct^n
T(n) = aT(n-b) + O(f(n))
case 1: if a = 1, then O(n * f(n))
case 2: if a > 1, then O(a^n * f(n)).

(ii) Dividing Functions

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

Note: Take SC as 1 in case of divid. funct.
n/2^K = 1; K = log n, T(1) = 1

eg → Pg 38 [T(n) = 2T(n/2) + n]
(Rev Copy)

eg → Pg 35 $T(n) = T(n/2) + n$ → $O(n)$

* MASTERS THEOREM (VV. imp)

$T(n) = aT(n/b) + f(n)$; $f(n) = n^K \log^p n$

Case 1: if $\log_b a > K$; then $O(n^{\log_b a})$

Case 2: if $\log_b a = K$; then:

(i) if $p \geq 0$; $T(n) = O(f(n) \times \log n)$

(ii) if $p = -1$; $T(n) = O(n^{\log_b a} * \log \log n)$

(iii) if $p < -1$; $T(n) = O(n^{\log_b a})$

Case 3: if $\log_b a < K$; then $O(f(n))$

eg → Pg 39 [Q4, Q5, Q6, Q8, Q9] → $O(f(n) \log n)$

(C-3)

(i) $T(n) = T(n/2) + T(n/3) + 1$

$\Rightarrow 2T(n/2) + 1$

$\log_b a = 1$; 0; $1 > 0$.

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

eg → Pg 263 [Fixed Prac.] (imp)

(R) $T(n) = T(\sqrt{n}) + 1 \rightarrow O(\log \log n)$

$T(n) = T(\sqrt{n}) + n \rightarrow O(\log n)$

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

(imp) [Quick Sort]

2) SORTING

- Criteria for Analysis of Sorting Algo.

(i) # comparisons

(ii) # Swaps

(iii) Adaptive: if list sorted / partially sorted → takes less time.

(iv) Stable: if duplicate elements present, order must be preserved. (imp)

(v) Extra mem

(vi) in Place: no extra array used.

(1) Bubble Sort

- Heaviest element gets settled down after Pass 1. (imp)

eg → Pg 42 (Working of BS)

No. of pass = $n-1$; n = no. of elem.

- Pass 1: $n-1$ comp, max $n-1$ swaps
- Pass 2: $n-2$ comp, max $n-2$ Swap.
- Last Pass: 1 comp.

(imp) → Comparisons = $n(n-1)/2 = \Theta(n^2)$ [avg]

Swaps = $O(n^2)$

eg → Pg 268 [Adaptive BS] (made)

eg → Pg 269 [QS] (if heaviest elm, perform BS)

② INSERTION SORT

- insert an element in Sorted list in the correct position.

Note: Shifting process tells us the position of element to be placed; even if we predict pos^n b4 hand, same time.

eg → Pg 413 [Working of IS] (imp)

No. of passes = n-1

Pass 1: 1 comp.
Pass 2: 2 comp (max).
Pass (n-1): (n-1) comp max.

(imp) → Max Swaps = $O(n^2)$
Max Comp = $O(n^2)$

⇒ If already Sorted: n-1 comp: $\theta(n)$ (1 comp in each Pass)

IS :: adaptive by nature
BS: made adaptive

eg → Pg 272 [208]

IS is more suitable in Linked list than arrays

③ SELECTION SORT

Select i=1, Find min elem from [i+1 to n] as j; Swap a[i] & a[j].

No. of Passes = n-1

Pass 1: n-1 comp; 1 swap.
Pass 2: n-2 comp; 1 swap.
⋮
Pass (n-1): 1 comp; 1 swap.

comparisons → $\theta(n^2)$
Swaps → $O(n)$

* ~~Not~~ Stable Sort. (imp)

SS → After K Pass: K min elements are Selected at the beginning of list.
IS → After K Pass: K elements will be in sorted manner at the beginning
BS → After K Pass: K max elements will be present at the end of the list.

④ MERGE SORT

* Merging: combining 2 Sorted lists into Single Sorted List. (imp)

Time: $\theta(m+n)$

[Two way Merging]

Merging 2 Sorted lists (in same array) into another array & copying back to A.

eg → Pg 276

(imp)

* (ii) 4 lists merged: 4 way merge together)

* Merge Sort → Iterative MS
→ recursive MS.

1. Iterative Merge Sort

eg → Pg 47 [2 way merging used].

imp → No. of Passes = $\log n$
⇒ In each Pass, n elements are merged.
∴ MS = $O(n \log n)$

2. Recursive Merge Sort

- Similar like iterative MS, just the order in which elements are merged changes (not level wise)

eg → Pg 48 [Code & working]

* Size of stack = $\log n$ [depth of tree]
imp → No. of levels = $\log n$.
No. of elem merged at each level = n
∴ $T(MS) = n \log n$

$MS \rightarrow 2T\left(\frac{n}{2}\right) + n$, $S = O(n)$
(mergeSort() called 2 times; merge())

* For small size lists, MS too slow, ∵ recursion stack
⇒ MS is fast for lot of elements.
* for less than 15 elem, IS is used, else MS.
imp

	elem	level
eg → Pg 50	$2n/\log n$	1.
	$4n/\log n$	2.
	$8n/\log n$	3.
	:	
	$n\log n/\log n$	$\log(\log n)$ level.

Total time req to merge = $O(n \log \log n)$ [n merge in each level]

* Time taken to compare 2 strings of length n = $O(n)$
eg → Pg 51 [String merge] imp

5. QUICK SORT

Element in sorted posn : if elements left is smaller & elem right is greater.

imp → i → if greater, then Stop.
j → if less, than Stop

eg → Pg 282 (working) [Comparisons]

Sorted Position.
(20, 40, 30, 10, 50, 80, 90, 70, 60)
Partitioning posn [Partitioning at middle]

eg → Pg 52 [Code] imp
(Choosing middle element as pivot).
when LHS of list is taken as Pivot

Best case: Partitioning at middle : $O(n \log n)$
Worst Case: Partitioning at last : $O(n^2)$
→ Sorted list

eg → Pg 284 (Best & Worst case).
[No. of comparisons at each level deciding partitioning posn = O(n)]

* When Selecting middle elem as Pivot:
  - Best Case = O(nlogn) (Part. at middle)
    eg: Sorted Array
  - Worst Case = O(n²) (Part. at left)
    eg: dont know the order.
    [When mid elem is smallest/largest]

* Randomized Quick Sort: θ(nlogn)

If pivot → median; (imp) [best case]
partitioning will be done at middle.
∴ Best case : O(nlogn)
[ht of tree → logn]

eg → Pg 286 [Vimp] (alpa = O(n)).

3) GREEDY METHOD

* No of Spanning trees in a complete Graph → $n^{(n-2)}$ [K_n] (imp)

1) PRIM'S ALGORITHM (MST)

Prims
Kruskal

- To find minimum weight Spanning tree. (connects every vertex).
- Start at some Point & grow into a tree. [From visited, check nbrs & pick least].

eg → Pg 2 (Q 2) [edge Wt not distinct].
(many MSTs possible)

∴ [Shouldnt form a cycle].

* If edge wts (weight) not distinct, more than 1 MST is possible (imp)

→ Apply Prim's Alg till (n−1) edges.
∴ MST = n−1 edges

eg → Pg 4 [Q 2] (Direct Method to draw MST from adjacency matrix).

eg → Pg 5 [Vimp Q].
[Construct MST in such a way, that 0 is a leaf] (imp)

* Edge of min cost will be found in: $O(E) + O(n) + O(n \times n)$.

$T = O(n^2)$ (without min Heap) (imp)

$T = O(E \log V)$ (with min heap)

(iii) KRUSKAL'S ALGORITHM (MST)

eg → Pg 6

→ May get a forest, whereas Prim's = never a forest.

* We can only ST of graph which are connected. (If distinct, MST always unique) (imp)

- Every MST contains min e.
- removal of max e disconnects Graph [worst case]. ∴ max e in MST

eg → Pg 8 [Q2]

e is not present in every MST possible [if repeated]

eg → Pg 9 [Gate 2006]

Adding edge weights one by one.

eg → Pg 10 [No of distinct MSTs pass] (2014 & next page) (imp)

eg → Pg 11 [Q2]

Paper conclusion [No RL].

a) wt of edge (u,v) is |u − v| ⇒ line Graph.

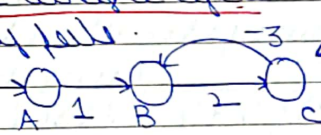
b) wt of edge (u,v) is (u+v) ⇒ Star Graph.

(iii) DIJKSTRA ALGORITHM [Single Source Shortest Path]

Don't apply dijkstra on graph with −ve edges. It may work/may not work. (imp)

eg → Pg 16 [Working-Table & Path]

Note: If −ve weight cycle present, dijkstra surely fails.



(imp)

eg → Pg 15 [Feb 2008] (Here the answer come out to be correct). → even find SP from a–d.

* ANALYSIS OF KRUSKALS

- Disjoint Set Method used.
- Create_Set (V) → O(V)
- Sort → O(E log E)

∵ $K_n \rightarrow n^2$ edges

* Now in a connected graph, E can never be greater than V^2.

∴ $T = O(E \log E)$ → $T = O(E \log V^2)$

$T = O(E \log V)$ } Time Complexity

imp

$S = O(V)$ Space Complexity

* ANALYSIS OF DIJKSTRA

- Heap implementation.

1. Initialize : $O(V)$
2. Build Heap : $O(V)$.
3. Extract min : $O(\log V)$ [repeated V times].
4. decrease Key : $O(\log V)$ [every edge relaxed once exactly (worst case = E).

imp [remember steps]

∴ $T = O(V + V + V \log V + E \log V)$

∴ $T = O(E \log V)$

$O((|E| + |V|) \log |V|)$ Gate

(iii) HUFFMAN CODING → $T = O(n \log n)$

- If frequency of characters vary in file, HC is used to compress size of data: [zip]

eg → Pg 23 [6 char] (Prefix code to be added to each character)

⇒ No of bits required to save file = external path length.

imp

* sort → $\log n$, create heap → $O(n)$
extract min → $\log n$
∴ Time Complexity = $O(n \log n)$
Space Complexity = $O(n)$

imp

Pull out node & put it back.

Q letters a, b, c, d, e, f have prob. 1/2, 1/4, 1/8, 1/16, 1/32, 1/32.
avg length = ? → No. of bits/char
↳ prob method / freq method

eg → Pg 24 [Gate Q]

eg → Pg 25 [a e i o u s t]
[we might even get a forest].

- Keep least 2 char. frequency.
L → small, R → greater.

O(n+m)
uses DFS

(V) TOPOLOGICAL SORT → O(n+m)

(imp)
- Graph should be directed and acyclic for topological ordering.

S-1: find vertex with indegree 0.
[delete it & the corr edges]
S-2: choose another vertex with indeg 0.

eg → Pg 28 [Q2]
(a Top. Ordering is possible]

(VI) BELLMAN-FORD ALGORITHM O(nm)

(imp)
- Single Source Shortest Path.
- relax all edges for (n-1) times.

Relaxation →
if [d(u) + cost(u,v) < d[v]]
d[v] = d[u] + cost(u,v)

S1: Source node=0, other nodes=∞
S2: write down the list of edges
S3: traverse edges (n-1) times.

E edges relaxed |V|-1 times.

T = O(|V||E|)
T = O(n²)

(imp)

eg → Pg 29

* If complete Graph:
$\frac{n(n-1)}{2} \times (n-1) \rightarrow O(n^3)$

* Drawback: (imp)
if -ve edge cycle: Graph cant be solved (Goto)
But, BF can detect -ve edge cycle;
if even after (n-1) itr^n, no nothing changes. [if reachable from src]

(VII) GREEDY KNAPSACK PROBLEM

- used for objects which can be divisible. [Fractional KS]

S-1: Select object highest P/W ratio.

x(1, 2/3, 1, 0, 1, 1, 1).
↳ 2kg out of 3kg can be pushed into a KS (fract^n)
or 4 out of 6.

eg → Pg 31 [Greedy KS]

Constraint: $\sum x_i w_i \leq m$
Objective: $\max \sum x_i p_i$
(imp)

→ Optimization Problem.

(VIII) OPTIMAL MERGE PATTERNS (O(n log n))

- Here, we need to merge files

(containing records) with min no. of record movements. (imp)

External Path Length = no. of record movements

Given n files:
create min Heap → O(n).
Take 2 min & insert back → O(log n)
(n-1) times.
∴ T = O(n log n) (like Huffman coding).

eg → Pg 34

(imp) JOB SEQUENCING WITH DEADLINES — O(n²)

- Schedule the jobs, get the profit; if you finish within deadline.
Task: Maximise the profit.

S-1: Sort the jobs (descending order of profits) → O[n log n]

S-2: Take the first job & place it as far as possible in deadline & subsequently other jobs → O[n²].

eg → Pg 36 [Q2]

(imp) T → O(n²) (Searching for job)

4) DYNAMIC PROGRAMMING

① LONGEST COMMON SUBSEQUENCE

S1 : a b d a c e
S2 : b a b c e ⇒ bace ⇒ abce
2 solutions.

matching must not intersect.

* Recursive Algo to find LCS (Gate)

```
int LCS(i,j)   → [can be computed either row/column major].
{ if(A[i] == '\0' || B[j] == '\0')
      return 0;
  else if(A[i] == B[j])
      return 1 + LCS(i+1, j+1);
  else
      return max(LCS(i+1,j),
                 LCS(i,j+1));
}
```

(imp)

eg → Pg 39 [Stone, Largest] (imp) n=5 m=7

Time → O(m*n) TS = (m+1)(n+1)

② SUBSET SUM PROBLEM

eg → Pg 40 [next pg] (imp)

```
if (j < input[i])
    T[i][j] = T[i-1][j]   // from Top
else
    T[i][j] = T[i-1][j] ||
              T[i-1][j - input[i]]
```

$T = \theta(n \times Sum)$

PREVIOUS YEARS INSIGHTS

① Divide & Conquer

Merge Sort = Stable (imp)

* QuickSort → not adaptive.

* Connected components of a graph can be computed in linear time by using bfs/dfs.

* Algo to determine if there are any 2 elements in Sorted array with sum less than 1000. → O(1)
[Take starting 2 elements]

* Recurs. eqns
  - Binary Search → $T(n/2) + 1$
  - Merge Sort → $2T(n/2) + n$
  - QuickSort → $T(n-k) + T(k) + n$
  - Tower of Hanoi → $2T(n-1) + 1$ (imp) → depends on Pivot.

ht of tree = $\log n$

* To find gcd(n, m); no. of recursive calls = $\theta(\log n)$ (imp)
  - Worst case → consecutive fibbonacci Numbers.

eg → Q.2.23 [Quick Sort → partitioning such that: Let 1 → 1/5th element]

eg → Q.2.24 → n/4 Smallest element taken as pivot. [Partition in some way]
$T(n) = T\left(\frac{n}{4}\right) + 3T\left(\frac{3n}{4}\right) + n$ (n/4, 3n/4)
(known result → $\theta(n \log n)$)

Comparing 2 elements = O(1) (imp)
Comparing 2 n-length string = O(n)

* The min no of comparisons required to find the minimum & maximum of n numbers is $\leq \frac{3n}{2} - 2$ [Gate 2014] (imp) (2 times)
[atmost]

* When worst case, think abt the situation like: worst case complexities of insertion & deletion in BST: O(n)
∴ Skewed
$T(n) = T(n-1) + T(n-2) + 1 \rightarrow O(2^n)$.

* Generating function G(z) for fibbonacci nos = $G(z) = \frac{z}{1 - z - z^2}$ (Gate)

Complexity → $O(2^n)$

② Algorithm Analysis & Asymptotic Notations

* $100n \log n = O(n \log n/100)$
∵ constants doesn't matter at all, they don't change growth rate of func
$2n = O(nK)$

X	1	2	3	.....
P(X)	$1/n$	$(n-1/n)(1/n)$	$(n-1/n)(n-1/n)(1/n)$	

expected no. of comparisons.
$\therefore E(X) = \sum x P(X)$. [Linear Search]
$\therefore$ Sum of AGP $= a/(1-r) + dr/(1-r)^2$

eg → Q 1.10 ✓ (imp)

In: $T(1) = 1$.

eg → Q 1.19 [Simply Substitute]

$(n+K)^m = \theta(n^m)$ ✓
$2^{n+1} = O(2^n)$. ✓ (imp)
$2^{2n+1} \neq O(2^n)$ X. → $\theta(2^{2n})$

Code to find e^x → Imp.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \infty$$

eg → Q 1.13 [Such approach for Math]

* Transitive closure [edi graph] is computed using Floyd Warshall Algorithm → $O(n^3)$ (imp)

[finds direct & indirect path from node to another]

```
void fun(int n)
{ if (n > 2)
  { pf(n);
    fun(√n) + n;
  }
}
```

→ $T(n) = T(\sqrt{n}) + 1$ ∴ $O(\log \log n)$ (imp) [don't repeat]

eg → Q 1.43 $K = n/2 + n/2 + \cdots$ ($n/2 * \log n$ times)
$\therefore \frac{n}{2} \times \frac{n}{2} \times \log n$. (imp)
$\Rightarrow \theta(n^2 \log n)$

$2T(n-1) + 1 = O(2^n)$
Towers of Hanoi with n disk = $\theta(2^n)$
Addition of 2 $n \times n$ matrix = $\theta(n^2)$

* $\sum_{i=0}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2 = \theta(n^4)$.

eg → Q 1.51 $f(n) = n$; $g(n) = n^{(1+\sin n)}$.
[Clear Concept]

eg → Q 1.52 $T(n) = 5T(n/2) + 1$. (imp) [Master]
$\theta = n^{\log_2 5}$

$n^{7/4}$; > $n \log^9 n$.
$\Rightarrow \frac{7}{4} \log n$; $\log n + \log(\log n)^9$

Sum of Harmonic = log n
Series
[Famous].

Page 26

$\Rightarrow \frac{7}{4}\log n > \log n + 9 \log\log n$

* The min no. of comparisons required to determine if integer appears more than $n/2$ times in a sorted array → $\theta(\log n)$

(imp)

1 3 4 4 4 4 4 6 7 9

check $(i + n/2) = a$

$\therefore \left[1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \ldots\ldots \frac{1}{n}\right] = \log n$ (imp)

eg → 1.56 [Sum of Harmonic Series]

eg → 1.46 (Here p becomes $\log n$)
next loop runs $O(\log\log n)$ times
(V.imp)
IMP [for analysing functions]

If balanced BST:
find L & H → $O(\log n)$ (imp)
find all n nos b/w them → $O(m)$.
∴ Alg that computes sum of all nos b/w L & H = $O(m + \log n)$
[due to back edge]

* Backtracking Search on Graph → DFS

* Time complexity of radix Sort → $\theta(Kn)$

Page 27

* Decomposition into strongly connected components.
{P, Q, R, S, T, V} {U}
[every vertex among them is reachable from any other vertex]
⇒ U can't reach P, Q ∴ U separate
eg → Q 6.21 (imp)

* If we want to arrange n nos in array such that all -ve values occur before all +ve values. Min exchanges required in worst case → $O(n/2)$

* INSERTION SORT USING INVERSIONS (imp)
$O(n + f(n))$
↳ total no. of inversions.
$[0, \frac{n(n-1)}{2}]$ ↑ desc. dig.
$[O(n), O(n^2)]$ [Sorted]
Inversion is a pair (a_i, a_j) ; $i < j$, $a_i > a_j$

* If each set → linked list (imp)
Membership : $O(n1 + n2)$
Cardinality : $O(n1 + n2)$
Union : $O(n1 \times n2)$
Intersection : $O(n1 \times n2)$
} slowest.
if eqn = $O(n+n)$

eg → Q 6.15 [Grp Rhoda]
(V. imp)

* Even if edges are sorted, complexity of Kruskal → O(mlogn) (imp)
→ union-find

* In an unweighted, undirected connected Graph; the shortest path from a node S to every other node is computed in → BFS → T(O(m+n)). (Gate)
· using Queue data Structure.

eg → 3.11 [Imp]

(Graph eg)

Imp → edge weights not distinct
e → edge with max weight.

[Single Source SP]

* Bellman-Ford Shortest path algo always finds any negative-weighted cycle, which is reachable from the source. (imp)

eg → Q1.17 [Analysing time complexity of Algo] (Grp) (Tough)

T(n) = 2T(√n) + 1 → Recurrence eqn with square roots.
Substitute n = 2^m.
⇒ T(2^m) = 2T(2^{m/2}) + 1
⇒ T(log 2^m) = 2T(log 2^{m/2}) + 1
⇒ T(m) = 2T(m/2) + 1. Apply MT.
∴ θ(m) → θ(logn) (imp)

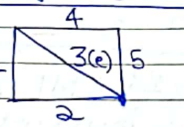
S(m) = 2S(m/2) + 1 → θ(m)

T = n(n+1)/2

(imp)

eg → 2.24 [Space Complexity is calculated using the depth of tree] → [no of times code runs]
If dynamic Programming used in the Q; Time Complexity [O(n^2)] but Space Complexity: O(n)

eg → 3.43

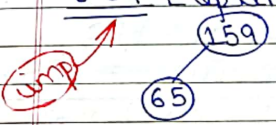


(Graph eg)

(imp) → edge weights are distinct. [Test Q]
e is lightest edge in cycle, still not included in MST.

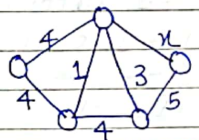
eg → 3.40 [Find the missing edge of Graph, if MST is given]

eg → 3.39 [Optimal Merge Patterns]



(imp) [ME Solution]

No. of comparisons = n − 1 ; if n output elements



x = 5 [will produce max no. of STs]
∴ 4 STs

* If Probability is given in Huffman coding:

Avg length / char = $\sum$ (No. of bits * freq. of occurrence of alphabet)

eg → 3.45 → EM Test

eg → 4.14 [ME Soln on Matrix chain multiplication Problem]

* An element in an array X is called a leader if its greater than all elem to the right of it. Algo that finds all leaders scan right to left in O(n)

TEST (2019) – ME.

* Level Order Traversal = BFS
Pre/Post/Inorder Trav. = DFS.

* Ternary Search: $T(n) = T(n/3) + 4$ (4 comp. [∵ mid1, mid2])
Binary Search: $T(n) = T(n/2) + 2$ (2 comp)
→ $2\log_2 n + 1$ comparison
→ $4\log_3 n + 1$ comp.
∴ Binary Search → efficient than ternary S.

* Divide & conquer: Strassen Matrix mult.
Backtracking: Graph colouring [DFS].

PQ

* Heap Q: (min-max Heap)
(constant time retrieval for both minimum & maximum elements)

Sorted both min max Heap

	minimum	max.
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

* St. Procedure to Solve $\sqrt{n}$

$T(n) = T(\sqrt{n}) + \log n$; $n = 2^m$

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

$\Rightarrow S(m) = S(m/2) + m$ (Master Theorem)

$\Rightarrow \theta(m) \Rightarrow \theta(\log n)$ (imp)

* Karatsuba Algo [Product of 2 integers (size n)] (imp)

$T(n) = 3T(n/2) + n$ — Fastest mul.

$O(n^{1.59}) \longrightarrow O(n^{\log_2 3})$

* If a balloon shot: 1 appears for n balloons?

1	1
2	3
3	5
n	2n−1 ∴ O(n)

Table Approach

* No of comparisons to find biggest elem. in array (size n)

$\Rightarrow n - 1$

8 4 3 ⇒ 2 comp

* Every no. in array is same [Sorted]
  Quick Sort: $O(n^2)$
  Insertion Sort: $O(n)$ [∵ adaptive] (imp)
  Selection: $O(n^2)$ [always]
  Merge: $O(n \log n)$ [always]

* If adjacency list rep. of undirected Graph is given:
  Find min. of length of Shortest dist. b/w all asked.
  Just find BFS: $O(V+e)$
  eg → Q23

* Merge Sort with → $O(n^2)$. (imp)
  No extra Space
  Find no. of inversion → $O(n \log n)$
  Pairs [done in MS]

* Determine Sink Node of a Graph.
  $O(n)$
  Code: 1st Part: (find the i^{th} row)
  2nd Part: (checks if
  $A[i][j] = 0$ && $A[j][i] = 1$; is it happening or not (Sink node cond^n).
  eg → Q25

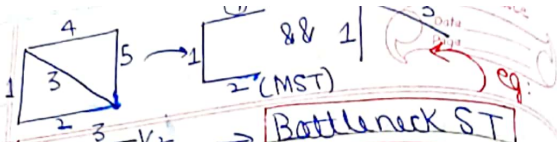
(imp) [Prev Years Gate Q.]

* Insert/delete/replace in Array.
  [if Unsorted & if at last (I/D)]
  ⇒ $O(1)$.

* Alg: find an element which is higher than its left & right element if any
  $O(n)$ [just traverse & compare]
  if (A[i] > A[i+1] && A[i] > A[i-1])
  print(A[i])

* Degree Sequence: Given a degree sequence $\{d_1, d_2, d_3 \cdots d_v\}$, determine if simple Graph is Possible (imp) (Graph Theory)
  'Havalle Hakimi Algorithm':
  - Sort deg. seq. first ($n \log n$)
  - decrement oper^n
  - Sort again (total $n-1$ times)
  ∴ $O(n^2 \log n)$

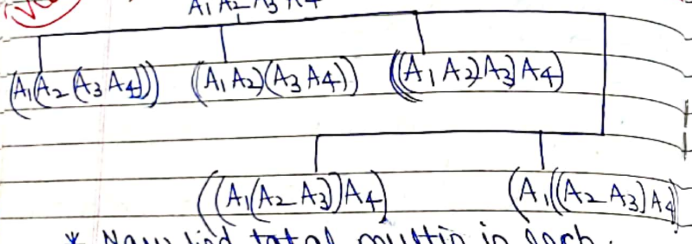
* Heap Q. (imp) [Mult Q in Heap].
  - delete middle element from min Heap of n elements.
  (i) find middle element = simply access $(n/2)^{th}$ elem: $O(1)$
  (ii) replace it with last elem = $O(1)$
  (iii) Heapify = $O(\log n)$
  $O(\log n)$



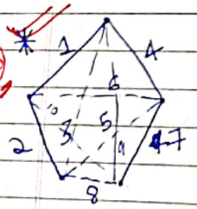
Every MST not ⟶ bottleneck min
Every min Bottleneck ⟶ not MST

* $A_1 A_2 A_3 A_4$ (4 Matrix)
Total Paranthesis = $C_3 = \frac{2n C_n}{n+1} = \frac{6C_3}{4}$ ⑤

Vimp



* Now find total multip in each.



(i,j+2)

Q Edge Wts : 1 2 3 4 5 6 7 8 9 10.

Find MST with max possible weight.
- Start with least edge [& grow from a tree].
if cycle made, make a cycle

* MultiStage Graph Algo.
[Greedy]
Tip: Start from backwards
eg → Q33

* Complexity of a recurrence relation
[Just count no. of function calls to calculate time complexity].
eg → Q16. [log n & n]