GATE CSE NOTES

by

UseMyNotes

Algorithms

- **Rate of growth of functions:**

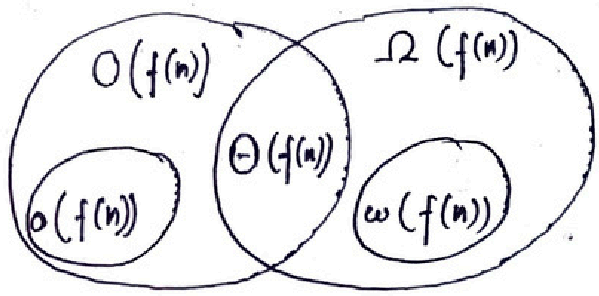
$1, \lg\lg n, \sqrt{\lg n}, \lg n, (\lg n)^c \; c>1, \; n^c \; 0<c<1, \; n = 2^{\lg n},$

$n \lg^* n, \; n\lg n = \lg n!, \; n^2, \; n^c \; c>2, \; c^n \; c>1, \; n!, \; 2^{2^n}$

Note. $\lg^* n = \begin{cases} 0 & , n \le 1 \\ 1 + \lg^*(\lg n) & , n>1 \end{cases}$

- **Asymptotic Notation.**

	Analogy	
$f(n) = O(g(n))$	$\le$	f grows slower than some multiple of g
$f(n) = o(g(n))$	$<$	f grows slower than any multiple of g
$f(n) = \Omega(g(n))$	$\geq$	f grows faster than some multiple of g
$f(n) = \omega(g(n))$	$>$	f grows faster than any multiple of g
$f(n) = \Theta(g(n))$	$=$	f grows at same rate of g



→ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in R^+$, $f(n) = \Theta(g(n))$ | → $\lim_{n\to\infty} \frac{f(n)}{g(n)} \le c \in R$, $f(n) = O(g(n))$ | → $\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c \in R$, $f(n) = \Omega(g(n))$

→ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) = O(g(n))$ & $f(n) \neq \Theta(g(n))$ & $g(n) \neq O(f(n))$

$\Rightarrow f(n) = o(g(n))$

→ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) = \Omega(g(n))$ & $g(n) \neq \Omega(f(n))$ & $f(n) \neq \Theta(g(n))$.

$\Rightarrow f(n) = \omega(g(n))$

→ $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$ | → $\Theta(f+g) = \Theta(\max(f,g))$

→ $\max(f(x), g(x)) = \Theta(f(x) + g(x))$. | → $f(n) = O(g(n))$ iff $g(n) = \Omega(f(n))$

→ $f(n) = O(g(n)), g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$. (for Θ & Ω also)

→ $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$ | $f = \Theta(g) \underset{\text{iff}}{\Rightarrow} f = O(g)$, $f = \Omega(g)$

- $a^{\lg_b x} = x^{\lg_b a}$ | $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} \; (x \neq 1)$ | $\sum_{k=1}^{n} \log k = n\lg n$

$\sum_{k=1}^{n} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \lg n$ | $\sum_{k=1}^{n} k^p = 1 + 2^p + 3^p + \dots + n^p = \frac{1}{p+1} n^{p+1}$

→ $f = O(g)$ iff $g = \Omega(f)$ | $f = o(g)$ iff $g = \omega(f)$

→ $f = O(g)$ & $g = O(h)$, then $f = O(h)$. If either (or both) big-oh is a little-oh, then $f = \underset{\text{little}}{o}(h)$. (Similar for Ω, ω)

→ If $f = \Theta(g)$, $g = \Theta(h)$, then $f = \Theta(h)$. If any one (but not both) of the Θ is replaced by another notation, then the conclusion uses that same (the replaced one) notation.

eg. $f = O(g)$, $\boxed{\cancel{g \neq \Theta(h)}}$ $g = \Theta(h) \Rightarrow f = O(h)$

→ $f = o(g) \Rightarrow f = O(g)$ | → $f = \omega(g) \Rightarrow \cancel{f}$ $f = \Omega(g)$

→ $f = O(f)$, $f = \Omega(f)$, $f = \Theta(f)$ but $f \neq o(f)$, $f \neq \omega(f)$.

- Extended Master's $T(n) = aT(n/b) + \Theta(n^k \lg^p n)$; $a \geq 1$; $b > 1$; $k \geq 0$
  p real number

1. If $a > b^k$, $T(n) = \Theta(n^{\lg_b a})$
2. If $a = b^k$, i) If $p > -1$, $T(n) = \Theta(n^{\lg_b a} \lg^{p+1} n)$
   ii) If $p = -1$, $T(n) = \Theta(n^{\lg_b a} \lg\lg n)$
   iii) If $p < -1$, $T(n) = \Theta(n^{\lg_b a})$
3. If $a < b^k$, i) If $p \geq 0$, $T(n) = \Theta(n^k \lg^p n)$
   ii) If $p < 0$, $T(n) = O(n^k)$.

- Master's for subtract & conquer $T(n) = aT(n-b) + O(n^k)$
  $a > 0$, $b \geq 1$, $k \geq 0$

1. $a > 1$ $O(n^k a^{n/b})$
2. $a = 1$ $O(n^{k+1})$ 3. $a < 1$ $O(n^k)$.

- Sorting i) Insertion sort insert keys one by one into the sorted subarray so that the key is placed at the correct place in the sorted subarray. ii) Merge Sort divides the array into 2 parts until 1 elem in array is remaining & then merge them recursively to get a sorted array. iii) Quick sort takes an elem as pivot, places the pivot at its correct position in sorted array & places all smaller elems (than pivot) to left of pivot & greater to the right. Then recursively call QS on the left & right parts.

- Searching (Linear, Binary)
  $O(n)$ $O(\lg_2 n)$

Algorithms

- **Divide & Conquer** : Min-max, Strassen's matrix mult^n, Merge sort, Quick sort, Binary Search

- **Greedy Algorithm** : Fractional KS $O(n\lg n)$, Huffman coding $O(n\lg n)$, Job sequencing with deadlines $O(n\lg n)$,

Kirchoff's matrix tree theorem (-find # ~~mat~~ spanning trees for a connected graph), Prim's Algo (MST) (pick min weight edge everytime)

*Finding MST from adj matrix., Kruskal's algorithm (finding MST) - add connecting edges at last, first include all least weight edges in the MST (Disjoint Set - Kruskal's Algo),

Finding connected components using Disjoint Set DS (if for each edge, the adj. vertices are not in the same set, then union)

- **Shortest Path Algo.** Single source shortest path : Relaxation (Shortest path estimate $d[v]$), Dijkstra's algo - greedy

(using min prio queue Q, set of vertices S whose final shortest path from source is known) $d[v]$ $O(|V|^2)$, Bellman Ford Algorithm (relax each edge $|V|$ times) $O(|E||V|)$ [relaxing edges $|V|-1$ times means we are finding shortest path weights when the path length is at max $|V|-1$, $|V|$ th relaxation is to detect -ve cycle), Topological sort (-for DAGs).

- **Dynamic Programming** 1. Matrix chain multiplication (# of parenthesizations $= C_{n-1}$, $m[i,j]$ be min # scalar multiplications needed compute matrix $A_{i..j}$;

A_i $p_{i-1} \times p_i$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1}p_k p_j \} & i < j \end{cases}$$

| TC $O(n^3)$ | SC $O(n^2)$

#unique subproblems possible for $m[1;n] =$

$$\underset{\text{size }1}{n} + \underset{\text{size }2}{(n-1)} + \cdots + \underset{\text{size }n}{1} = \frac{n(n+1)}{2}.$$

2. Longest common subsequence (Brute force $\theta(n2^m)$, X_m, Y_n,

DP: $lcs(i,j) = \begin{cases} 0 & i=0 \text{ or } j=0 \\ 1 + lcs(i-1, j-1) & i,j>0 \ \& \ x_i = y_j \\ \max(lcs(i-1,j), \ lcs(i,j-1)) & i,j>0 \ \& \ x_i \neq y_j \end{cases}$

TC $O(nm)$

SC $O(nm)$

3. Shortest path in multistage graph.

$$cost(i,j) = \min_{\substack{l \in V_{i+1} \\ \langle j,l\rangle \in E}} \{ c(j,l) + cost(i+1,l) \}$$

Vertex j in V_i to vertex t (sink)

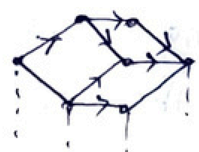
weight of edge

Vertices partitioned into 'V_i's. $1 \le i \le K$

edge (u,v) s.t.

$u \in V_i$ | $|V_1| = 1$

$v \in V_{i+1}$ | $|V_K| = 1$



$i=1$ 2 3 4 V_i

TC $O(E)$ or $O(V^2)$.

4. Binary Knapsack (pseudo-polynomial time)

$C[i,w]$ value of optimal profit for items $1,\dots,i$ and max weight w. | c - capacity

$$c[i,w] = \begin{cases} 0, & i=0 \text{ or } w=0 \\ c[i-1,w], & w_i > w \\ \max\{p_i + c[i-1, w-w_i], \ c[i-1,w]\} & , i>0 \ \& \ w_i \le w \end{cases}$$

#unique subproblems in recursion tree = (#objs) × (capacity) = $n \times c$

TC $O(nc)$

SC $O(nc)$

5. Subset Sum (pseudo-polynomial time)

$$isSS(n, sum) = \begin{cases} False, & n=0 \ \& \ sum>0 \\ True, & sum = 0 \\ isSS(n-1, sum) \,||\, isSS(n-1, sum - set[n]), & \text{otherwise} \end{cases}$$

TC $O(n \cdot sum)$

SC $O(n \cdot sum)$

6. Traveling Salesman Problem (Held-Karp Algo)

$$\begin{cases} g(i,\phi) = c_{i1}, & 1 \le i \le n, \ S=\phi \\ g(i,S) = \min_{k \in S} \{ c_{ik} + g(k, S-\{k\}) \} & S \neq \phi \end{cases}$$

TC $O(2^n n^2)$

SC $O(2^n \sqrt{n})$ by storing only subsets of size s-1 & s at any point of the algo.

7. Floyd-Warshall Algo. (APSP)

$d_{ij}^{(K)}$ be the weight of a shortest path from i to j for which all intermediate vertices are in the set $\{1, 2, \ldots, k\}$.

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & , k = 0 \\ \min \left(d_{ij}^{(k-1)}, \; d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right) & , k \geq 1. \end{cases}$$

TC $O(V^3)$ SC $O(V^2)$ using 2 matrix of $O(n^2)$ & reusing them

8. Bellman-ford Algo (SSSP)

$d(v, i)$ be the length of shortest 'source to v' path whose length is at most i.

$$d(v,i) = \begin{cases} 0 & i = 0 \;\&\; v = src \\ \infty & i = 0 \;\&\; v \neq src. \\ \min \left\{ d(v, i-1), \; \min\limits_{(u,v)\in E} \{ d(u, i-1) + w(u,v) \} \right\} & \text{otherwise} \end{cases}$$

TC $\Theta(VE)$.

SC $\Theta(V)$

# graph criteria	BFS $O(V+E)$	Dijkstra's $O((V+E)\lg V)$	Bellman-ford $O(VE)$	Floyd-Warshall $O(V^3)$
Max size	V, E ≤ 10M	V, E ≤ 300K	VE ≤ 10 M	V ≤ 400
Unweighted	Best	OK	Bad	Bad in general
weighted	WA	Best	OK	Bad in general
-ve weight	WA	OK	OK	"
-ve cycle	can't detect	Can't detect	Can detect	Can detect
Small graph	WA if weighted	Overkill	Overkill	Best

- Sorting algos logic: 1. Quick sort: Choose pivot element & place in correct position & continue until each subproblem has either 1 or 0 elems, 2. Merge sort: Divide 2 equal parts, recursively sort each subproblem & merge into single sorted list, 3. Heap sort: Build max heap, delete max place in last position (repeat $n-1$ times), 4. Bubble sort: Compare & exchange adjacent elems, repeat $n-1$ passes, 5. Selection sort: Find pos^n of min elem from $a[i \dots n]$ and swap with $a[i]$ where $i = 1, 2, \dots, n-1$ passes, 6. Insertion sort: Insert $a[i+1]$ into correct position into $a[1]$ to $a[i]$ sorted part of array, where $i = 1, 2, \dots, n-1$ passes.
- # comparisons for sorting algos: 1. Insertion sort ($\Theta(n^2)$ worst case, $O(kn)$ if $\leq k$ items out of order, 2. Merge sort ($\Theta(n \lg n)$ worst case), 3. Heap sort ($\Theta(n \lg n)$ worst case). 4. Quick sort ($\Theta(n^2)$ worst, Θ ($n \lg n$ average)).

AAA

* $f(n)$ polynomially bounded iff $\log(f(n)) = O(\log n)$

$f(n)$ exponentially " iff $\log(f(n)) \neq O(\log n)$

$> \log n$

* # BSTs for n distinct keys $= \frac{1}{n+1}\binom{2n}{n} = C_n$

Unlabeled Binary trees for n nodes $= \frac{1}{n+1}\binom{2n}{n} = C_n$

Labeled Binary trees for n nodes $= n! \cdot C_n$