Algorithms

Functions

- **Logarithmic** — eg: $\log n$, $\log\log n$.
- **Polynomial** — n^x, $x \geq 0$. eg: $n, n^2, n^3 \ldots$
- **Exponential** (higher rate of growth) — a^n, $a > 1$. eg: $2^n, 4^n, n!, n^n \ldots$

* Time Complexity → Worst Case
* Smaller functions are in order of bigger functions.
* Bigger functions are omega of smaller functions.
* Rate of growth of both functions equal then theta of each other.

1. $\log x^y = y \log x$
2. $\log x.y = \log x + \log y$.
3. $\log(\log n) = \log\log n$.
4. $a^{\log_b x} = x^{\log_b a}$.
5. $\log^k n = (\log n)^k$
6. $\log \frac{x}{y} = \log x - \log y$
7. $\log_b a = \frac{1}{\log_a b}$.
8. $\log_b \frac{1}{a} = -\log_b a$

Notations

1. **Big-O**: $f(n)$ is $O(g(n))$
   → $f(n) \leq c \times g(n)$ (any constant).
2. **Big-Omega**: $f(n)$ is $\Omega(g(n))$
   → $f(n) \geq c \times g(n)$
3. **Theta "Θ"**: $f(n)$ is $\Theta(g(n))$.
   → $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
4. **Small-O**: $f(n)$ is $o(g(n))$
   → $f(n) < c.g(n)$
5. **Small-Omega**: $f(n)$ is $\omega(g(n))$
   → $f(n) > c.g(n)$

Summation Series

* $\sum_{k=1}^{n} = 1+2+\ldots n = \frac{n(n+1)}{2}$.
* $\sum_{k=0}^{n} x^k = 1+x+x^2 \ldots x^n = \frac{x^{n+1}-1}{x-1}$ $(x \neq 1)$
* $\sum_{k=1}^{n} \frac{1}{k} = 1+\frac{1}{2}+\ldots\frac{1}{n} \simeq \log n$

Geometric Sum formula (finite terms)

→ if $r=1$, $S_n = n*a$. (first term)

→ if $|r|<1$, $S_n = \frac{a(1-r^n)}{1-r}$

→ if $|r|>1$, $S_n = \frac{a(r^n-1)}{r-1}$

(common ratio)

for Infinite terms

→ if $|r|<1$, $S_\infty = \frac{a}{1-r}$

Dominance Relation

Constants < logarithms < polynomials < exponentials.

eg: $10 < \log n < \sqrt{n} < n < n\log n < n^2 < 2^n < n! < n^n$.

1. $f(n) = \sum_{i=1}^{n} 1 = n = \underline{O(n)}$

2. $f(n) = \sum_{i=1}^{n} n = n \cdot \sum_{i=1}^{n} 1 = \underline{O(n^2)}$

3. $f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \underline{O(n^2)}$

4. $f(n) = \sum_{i=1}^{n} i2 = \frac{n(n+1)(2n+1)}{6}$

$= \underline{O(n^3)}$

5. $f(n) = \sum_{i=1}^{n} i3 = \left[\frac{n(n+1)}{2}\right]^2 = \underline{O(n^4)}$

6. $f(n) = \sum_{i=1}^{n} 2^i = (2^{n+1}-2) = \underline{O(2^n)}$

7. $f(n) = \sum_{i=1}^{n} 1/2^i = \left(1-\frac{1}{2^n}\right) = \underline{O(1)}$

8. $f(n) = \sum_{i=1}^{n} 1*2^i = (n-1)2^{n+1}+2$

$= \underline{O(2^n)}$

9. $f(n) = \prod_{i=1}^{n} 1 = \underline{O(1)}$

10. $f(n) = \prod_{i=1}^{n} i = (1*2*3*\ldots*n)$

$= n!$

$= O(n^n)$

(loose bound)

→ $\log n! = \Theta(n \log n)$

General properties of Big-O notations.

1. if $d(n)$ is $O(f(n))$ then $a \times d(n)$ is $O(f(n))$, $a > 0$.
2. if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$
3. if $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$ then $d(n) \times e(n)$ is $O(f(n) \times g(n))$
4. if $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$
5. if $f(n)$ is polynomial of degree d ($f(n) = a_0 + a_1 n + a_2 n^2 \ldots a_d n^d$) then $f(n)$ is $O(n^d)$
6. n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$
7. $\log n^x$ is $O(\log n)$ for any fixed $x > 0$
8. $\log^x n$ is $O(n^y)$ for any fixed constant $x > 0$ and $y > 0$.

	O	Ω	Θ	o	ω
→ Reflexive	✓	✓	✓	×	×
→ Symmetric	×	×	✓	×	×
→ Transitive.	✓	✓	✓	✓	✓

Common Recurrence Equations:

1. $T(n) = a + T(n-1) + b$, $n > 1$. $\Rightarrow O(n)$
   $a > 0$, $b > 0$ – constants.
2. $T(n) = a + T(n-1) + b \Rightarrow O(n^2)$
3. $T(n) = a + T(n-1) + \frac{1}{n}$, $n > 1 \Rightarrow O(1)$

4. $T(n) = 2T(n-1)+d, \ n>1 \rightarrow O(2^n)$

5. $T(n) = a + T(\sqrt{n}), \ n>2, \rightarrow O(\log\log n)$

6. $T(n) = a + 2T(\sqrt{n})+b, \ n>2 \rightarrow O(\log n)$

7. $T(n) = \sqrt{n} \times T(\sqrt{n}) + n, \ n>2 \Rightarrow O(n \log\log n)$

8. $T(n) = 2T(n/2)+d, \ n>1 \Rightarrow O(n)$ (Conv)

9. $T(n) = T(n/2)+d, \ n>1 \Rightarrow O(\log n)$ (Constant)

10. $T(n) = 8T(n/2)+c, \ n>1 \Rightarrow O(n^3)$

11. $T(n) = 2T(n/2) + n\log n \Rightarrow O(n \log^2 n)$

Loop Complexities.

* for(i=1; i≤n; i=i+a) $\rightarrow n/a$

* for(i=1; i≤n; i=i*a) $\rightarrow \log_a n$

Masters Method for Solving D and C Recurrences.

$T(n) = aT(n/b) + f(n)$ $a \geq 1$, $b \geq 1$ are constants

$f(n)$ is true

(I.) if $f(n)$ is $O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then,

$T(n) = \Theta(n^{\log_b a})$

(II) if $f(n) = \Theta(n^{\log_b a} \log^k n)$, for some k such that

(a). $k \geq 0$ then $T(n) = \Theta(n^{\log_b a} \times \log^{k+1} n)$

(b) $k < 0$ then $T(n) = \Theta(n^{\log_b a} \times \log\log n)$

(III) if $f(n) = \Omega(n^{\log_b a + \epsilon})$, $\epsilon > 0$ and $af(n/b) \leq c \cdot f(n)$

then $T(n) = \Theta(f(n))$

(any constant $c<1$)

Divide and Conquer Problems

1. **Max Min**: $T(n) = 2T\left(\frac{n}{2}\right) + 2,\ n > 2$.

$T(n) = \underbrace{\frac{3n}{2} - 2}_{\text{for all calls}} = O(n)$, Space : $O(\log n)$

2. **Binary Search**: $T(n) = T\left(\frac{n}{2}\right) + a,\ n > 1$, $TC = O(\log_2 n)$

↓ Max no. of comparisions $= \lceil \log_2 n \rceil$ (n = no of elements)

Space : $O(\log n)$ (for recursive)

3. **Merge Sort**: $T(n) = 2T\left(\frac{n}{2}\right) + bn,\ n > 1, b > 0$

$TC = O(n \log n)$, Space = $O(n)$.

→ No of comparisions to merge sorted list of size m and n

→ $\min(m, n) \leq$ no of comparisions $\leq (n + m - 1)$

→ No of recursive call $= 2(n-1)$.

4. **Quick Sort**: $T(n) = O(n) + 2T(n/2) = O(n \log n)$

↓ If elements are in ascending/descending order

↳ $T(n) = O(n) + T(n-1) = O(n^2)$

Space : Best $= O(\log n)$ Worst $= O(n)$

5. **Matrix multiplications**: $T(n) = 8T(n/2) + bn^2,\ n > 2, b > 0$

$TC = O(n^3)$, Space : $O(\log n)$.

→ Strassens method: $T(n) = 7T(n/2) + bn^2,\ n > 2, b > 0$

$T(n) < n^{2.81}$. Space = $\log_2 n + n^2 = O(n^2)$

(6.) Long Integer Multiplication : $T(n) = 4T(n/2) + b.n$, $n>1$, $b>0$

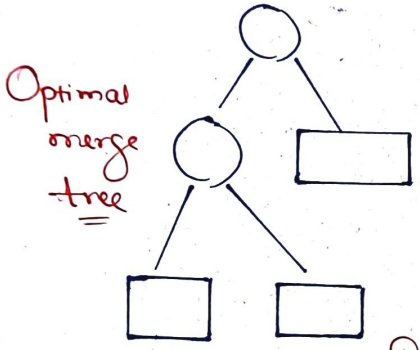
$T_C = O(n^2)$, Space $= O(\log n)$

Greedy Method Problems

(1.) Knapsack, $T_C = O(n \log n)$.

(2.) Job Sequencing with deadlines, $T_C = O(n^2)$

(3.) Optimal merge pattern



* Total record movement = weighted external path length

$\Rightarrow \sum_{i=1}^{n} d_i \times q_i$

d_i = distance from root to F_i

q_i = frequency/size of F_i.

* Construction starts from bottom (leaf) with pair of low frequency.

$T_C = O(n \log n)$

Space $= O(n)$

* Huffman Coding is an application of Optimal merge pattern and "Huffman tree" is constructed with starting from leaf node with character having "lowest probability".

(4.) Minimum Cost- Spanning tree:

Prim algorithm: Start with least cost edge, next edges to choose should be connected to existing vertices and adding new edge should not create a cycle.

$T_C = O(E+V \log V)$ using Fibonacci heap

$= O(E \log V)$ using Binary heap

→ Using Sorted Array and Adjacency list-
Prime algorithm TC = O(EV)

Kruskal Algorithm: Sort edges according to weights and add them without creating a cycle to get MST.

TC = O(ElogE), if edges sorted then O(E+V)

Dijkstra Algorithm (MST): add edges in any order, if loop formed then remove higher weight edge and continue. O(nxe)

Time Complexity: Dijkstra Single Source Shortest Paths.

(1) Min heap and adj. list = O(E + VlogV)

(2) Adj matrix and min heap = O(V² + ElogV)

(3) Adj list and unsorted array = O(V²)

(4) Adj list and sorted doubly linked list = O(EV)

Dynamic Programming

(1.) Fibonacci Series: TC = O(n), Space = O(n)

(2.) Travelling Salesperson problem: TC = O(n² 2ⁿ)

(3.) All pairs shortest paths: TC = O(n³), Space = O(n²)

Floyd Warshall's Algorithm:

$$A[i,j] = \min\{A[i,j], A[i,k] + A[k,j]\}.$$

A[i,j]: Cost of path from vertex i to j with k being intermediate vertex.

A[i,k]: Intermediate vertex.

(4.) 0/1 Knapsack problem: Tc = O(n×m), Space = O(n×m)

→ if w[i] ≤ j :

$$x[i,j] = \max\{x[i-1,j],\ x[i-1, j-w[i]] + p_i\}$$

x : dp table. w[i] : weight i. p_i : profit i

(5.) Longest Common Subsequence: Tc = O(nm), Space = O(nm)

→ Let x and y be 2 strings.

→ if x[i] = y[i], L(i,j) = 1 + L(i-1, j-1)

else $L(i,j) = \max\{L(i-1,j), L(i,j-1)\}$

L : dp table of lengths.

(6.) Matrix Chain Product: Tc = O(n³), Space = O(n²)

$$M[i,j] = \min\{M[i,k] + M[k+1,j] + P_{i-1} \times P_k \times P_j\}$$

M : denote no of scalar multiplications. k = splitting point.

(7.) Sum of Subsets: Tc = O(n×m), Space = O(n×m)

→ if A[i] > j : x[i,j] = x[i-1,j]

→ else : x[i,j] = x[i-1,j] || x[i-1, j-A[i]]

(|| : or)

Note Points :

* Dijkstra always works for graphs having +ve edges.
* If graphs have -ve edges and no negative weight cycle dijkstra algorithm may/may not work

* Belman Ford Algorithm always works if graph has -ve weight edges but not -ve weight cycle.

* If graph has -ve weight cycle reachable from source to destination then no algorithm works.

* TC of Bellman Ford = $O(VE)$ (V: vertices, E: edges), if $E = V^2$ (complete graph) then $O(V^3)$

Graph Traversals

1. Depth first search
2. Breadth first search

* TC of both DFS and BFS depend upon representation:
  (i) Adj matrix: $O(n^2)$
  (ii) Adj. list: $O(n+e)$

DFS leads to following edges

1. Tree edge: part of DFS spanning tree.
2. Forward edge: node to its non child descendent. in Spanning Tree.
3. Back edge: node to its ancestor (↑ edge between)
4. Cross edge: edge btw 2 nodes such that they do not have ancestor and a descendant relationship.

→ Both DFS and BFS can be used to detect presence of
→ cycle as well as to check if graph is connected or not.

→ Both DFS and BFS can be used to check if two vertices are connected or not.

→ DFS used to determine Connected, Strongly connected, biconnected and articulation points.

→ A graph is said biconnected if it does not contain articulation pairs.

Sorting Algorithms

Algorithm	Best Case	Avg Case	Worst C.	Stable	Inplace
Quick Sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n^2)$	✗	✓
Merge Sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	✓	✗
Heap Sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	✗	✓
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	✗	✓
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓

Binary Search Tree : Tree in which every node satisfies

* All keys in left subtree are smaller than the node
* All keys in right subtree are greater than the node.

Insert, Delete, Search } Avg : $O(\log n)$
Worst : $O(n)$ — skewed bst.

AVL Tree : Self balancing BST, where difference between height of left and right subtrees for any node cannot be more than 1. Balancing factor can be: "0, -1 or +1".

→ Max no. of nodes of height $h = 2^{h+1} - 1$.

→ Min no of nodes : $n(h) = 1 + n(h-2) + n(h-1)$

Heap : Heap is a complete binary tree with the property that value at each node is atleast as large as the value of children.

(i) Root. $\geq |L, R|$ → max heap

(ii) Root $\leq |L, R|$ → Min heap

* Max no of nodes at level 'i' of binary tree = $2^i - 1$.
* No of level comparisions for node getting inserted at level 'i'
  $= (i+1)$
* Total no of comparisions (max) for nodes at level 'i'
  $= (i+1) \times 2^{i-1}$
* $T(n)$ = no of comparisions for all nodes at all levels

$$= \sum_{i=1}^{k} (i-1) \times 2^{i-1} \Rightarrow TC = O(n \log n).$$

  * Insert in heap $\Rightarrow O(\log n)$
  * Build heap / heapify $\Rightarrow O(n)$.