Algorithm Design Techniques

Algorithms- procedure to solve a problem in steps

Why algorithms?

* Organization – to understand and compare different algos
* Problem solving – different problems require different algos
* Performance comparison – to compare time and space complexity
* Reusability – similar problems can use similar or same algos
* R&D

Classifications of algorithms based on-

1. Implementation method- recursion or iteration, exact or approximate, serial/parallel/distributed algos
2. Design Method- dynamic prog, greedy algo, divide and conquer, reduction (transform and conquer), backtracking, linear prog, branch and bound
3. Design Approach- top-down, bottom up
4. Others- randomized, based on complexity, research area

Time and Space Complexity

Why? To compare and find optimal solution

Time complexity – amt of time taken by the algorithm to run on the input

To estimate- cost of each fundamental instruction and no. of times it was executed

Space Complexity – amt of space taken by the algorithm to run on the input

To estimate – 2 parts

independent part – code, variables, constants, etc

variable part – input given, stack, etc

Auxiliary space complexity – amt of EXTRA space taken apart from the input.

C++ STL Containers

Container – object that stores – collection of objects – of same types.

Example- to store list of names – use vector

Types-

1. Sequential – allow to store elements – accessed in sequential manner – internally as arrays or linked lists

Eg:- Array, list, vector, deque

1. Associative – allow to store elements in sorted order – internally as binary tree

Eg:- set, map, multiset, multimap

1. Unordered Associative – doesn’t sort – internally as hash table

Eg:- unordered set, unordered map, unordered multiset, unordered multimap

Container Adapters – existing STL container – restricted to behave differently

Eg:- stack, queue, priority queue

stack(container adapter) – uses sequential container deque – restricted interface to – provide push and pop only

Bit manipulation Basics

1. Odd or even – n & 1, returns 0 if its even and 1 if its odd
2. Positive or negative – x ^ y – leftmost bit is 0 - positive

– leftmost bit is 1 – negative

1. Adding 1 - -~x will add 1 [-ve of a number is invert the bits of x and add 1]
2. Swap two numbers without using a third variable – use XOR since x ^ x = 0

x = x ^ y;

y = x ^ y;

x = x ^ y;

Bit manipulation problems

Pigeonhole principle

If there are n+1 objects[pigeons] and n boxes[pigeonholes], atleast one box contains two or more objects.

10 red, 10 white, 10 blue marbles. minimum no. of marbles you have to choose randomly from the bag to ensure that we get 4 marbles of same color?

Pigeons = no of marbles to choose = 4 (n)

Pigeonhole = colour = 3 (K)

Pigeon hole principle = Kn + 1

ceil[Average] is [Kn+1/n] = 4 [Kn+1/3] = 4 Kn+1 = 10 i.e., 3 red + 3 white + 3 blue + 1(red or white or blue) = 10

Mathematical Expectation

Inclusion And Exclusion Principle

Prime Factorization

Euclidean’s Algorithm

Theorems In Number Theory

Combinatorics

Recursion

Backtracking

Binary Search

Divide And Conquer

Divide

Conquer

Combine

Greedy Algorithms

Segment Trees

Lazy Propogation

Fenwick Trees

Sqrt Decomposition

Graph Traversal

Classic Dp

Dp Problems

Pattern/String Matching

Geometric Algorithms