Introduction to Competitive Programming

by Dr. Arnab Chakraborty

Objective:

- The purpose of this micro-course is to give you a thorough introduction to competitive programming. It is assumed that you already know the basics of programming, but no previous background in competitive programming is needed.

Objective (Contd.):

- This micro-course is especially intended for students who want to learn algorithms and possibly participate in the International Olympiad in Informatics (IOI) or in the International Collegiate Programming Contest (ICPC). Of course, this micro-course is also suitable for anybody else interested in competitive programming.

Objective (Contd.):

- It takes a long time to become a good competitive programmer, but it is also an opportunity to learn a lot. You can be sure that you will get a good general understanding of algorithms if you complete this micro-course, solving problems and taking part in contests.

Introduction to Competitive Coding:

- Competitive programming combines two topics:
    - ➢ the design of algorithms and
    - ➢ the implementation of algorithms.
- The **design of algorithms** consists of problem solving and mathematical thinking. Skills for analysing problems and solving them creatively are needed.
  An algorithm for solving a problem must be both correct and efficient, and the core of the problem is often about inventing an efficient algorithm.

Introduction to Competitive Coding (Contd.):

- Competitive programming combines two topics:
    - ➢ the design of algorithms and
    - ➢ the implementation of algorithms.
- The **implementation of algorithms** requires good programming skills. In competitive programming, the solutions are graded by testing an implemented algorithm using a set of test cases.
- Idea and implementation both should be correct.

Programming languages:

At the moment, the most popular programming languages used in contests are C++, Python and Java.

For example, in recent Google Code Jam contest, among the best 3,000 participants, 79 % used C++, 16 % used Python and rest % used Java. Some participants also used several languages.

Many people think that C++ is the best choice for a competitive programmer, and C++ is nearly always available in contest systems. The benefits of using C++ are that it is a very efficient language and its standard library contains a large collection of data structures and algorithms. C++ has powerful STL (Standard Template Library).

Programming languages (Contd.):

On the other hand, if large integers are needed in the problem, Python can be a good choice, because it contains built-in operations for calculating with 3 large integers. Still, most problems in programming contests are set so that using a specific programming language is not an unfair advantage.

Time complexity:

- The efficiency of algorithms is important in competitive programming. Usually, it is easy to design an algorithm that solves the problem slowly, but the real challenge is to invent a fast algorithm. If the algorithm is too slow, it will get only partial points or no points at all.
- The **time complexity** of an algorithm estimates how much time the algorithm will use for some input. The idea is to represent the efficiency as a function whose parameter is the size of the input. By calculating the time complexity, we can find out whether the algorithm is fast enough without implementing it.

Sorting:

- **Sorting** is a fundamental algorithm design problem. Many efficient algorithms use sorting as a subroutine because it is often easier to process data if the elements are in a sorted order.
- There are many algorithms for sorting, and they are also good examples of how to apply different algorithm design techniques. The efficient general sorting algorithms work in O(n log n) time, and many algorithms that use sorting as a subroutine also have this time complexity.

Data Structures:

- A **data structure** is a way to store data in the memory of a computer. It is important to choose an appropriate data structure for a problem because each data structure has its own advantages and disadvantages. The crucial question is: which operations are efficient in the chosen data structure?
- It is a good idea to use the standard library available in the programming language whenever possible, because it will save a lot of time.

Complete Search:

- **Complete search** is a general method that can be used to solve almost any algorithm problem. The idea is to generate all possible solutions to the problem using brute force, and then select the best solution or count the number of solutions, depending on the problem.
- Complete search is a good technique if there is enough time to go through all the solutions, because the search is usually easy to implement and it always gives the correct answer. If complete search is too slow, other techniques, such as greedy algorithms or dynamic programming, may be needed.

Algorithm Types:

- There are multiple different types of algorithms are possible. Some of the important types are –
    - Simple recursive,
    - Backtracking,
    - Divide and conquer,
    - Dynamic programming,
    - Greedy,
    - Branch and bound,
    - Brute force,
    - Randomized etc.

Diff. categories of Competitive Coding problems:

- Array, Hash Table, Linked List, Math, Two Pointers, String
- Binary Search, Divide and Conquer, Dynamic Programming
- Backtracking, Stack, Heap, Greedy, Sort, Bit Manipulation, Tree
- Depth-first Search, Breadth-first Search, Union Find, Graph, Design
- Topological Sort, Trie, Binary Indexed Tree, Segment Tree
- Binary Search Tree, Recursion, Brainteaser, Memoization, Queue
- Minimax, Reservoir Sampling, Ordered Map, Geometry, Random
- Rejection Sampling, Sliding Window, Line Sweep, Rolling Hash
- Suffix Array etc.

Greedy Algorithm:

- A **greedy algorithm** constructs a solution to the problem by always making a choice that looks the best now. A greedy algorithm never takes back its choices, but directly constructs the final solution. For this reason, greedy algorithms are usually very efficient.
- The difficulty in designing greedy algorithms is to find a greedy strategy that always produces an optimal solution to the problem. The locally optimal choices in a greedy algorithm should also be globally optimal. (Functions are Selection, Optimality check, Feasibility check and Objective)

Greedy Algorithm (Contd.):

Where to use Greedy algorithm?

- A problem must comprise these two components for a greedy algorithm to work:
    1. It has optimal substructures. The optimal solution for the problem contains optimal solutions to the sub-problems.
    2. It has a greedy property (hard to prove its correctness!). If you make a choice that seems the best now and solve the remaining sub-problems later, you still reach an optimal solution. You will never have to reconsider your earlier choices.

Greedy Algorithms (Contd.):

Sample problems on Greedy algorithm?

- Activity Selection problem (Tasks and deadlines)
- Fractional Knapsack problem
- Scheduling problem
- Coin problem etc.

Dynamic Programming:

- Dynamic programming is a technique that combines the correctness of complete search and the efficiency of greedy algorithms. Dynamic programming can be applied if the problem can be divided into overlapping subproblems that can be solved independently.
  There are two uses for dynamic programming:
  - **Finding an optimal solution:** We want to find a solution that is as large as possible or as small as possible.
  - **Counting the number of solutions:** We want to calculate the total number of possible solutions.

Dynamic Programming (Contd.):

- We will first see how dynamic programming can be used to find an optimal solution, and then we will use the same idea for counting the solutions.
  Understanding dynamic programming is a milestone in every competitive programmer’s career. While the basic idea is simple, the challenge is how to apply dynamic programming to different problems.
- **Example:** Coin problem, Longest increasing subsequence, Path in a grid, Knapsack problems, Edit distance or Levenshtein distance (e.g. LOVE -> MOVIE)

Amortized Analysis:

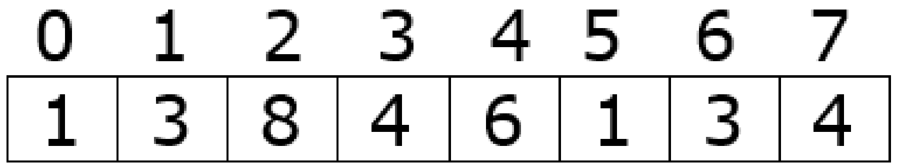
- **Amortized analysis** can be used to analyse algorithms that contain operations whose time complexity varies. The idea is to estimate the total time used to all such operations during the execution of the algorithm, instead of focusing on individual operations.

Range Queries:

- We discuss data structures that allow us to efficiently process range queries. In a range query, our task is to calculate a value based on a subarray of an array. Typical range queries are:
  - $\text{sum}_q(a, b)$: calculate the sum of values in range [a, b]
  - $\text{min}_q(a, b)$: find the minimum value in range [a, b]
  - $\text{max}_q(a, b)$: find the maximum value in range [a, b]

Range Queries (Contd.):

For example, consider the range [3, 6] in the following array:



0	1	2	3	4	5	6	7
1	3	8	4	6	1	3	4

In this case, $sum_q(3, 6) = 14$, $min_q(3, 6) = 1$ and $max_q(3, 6) = 6$. A simple way to process range queries is to use a loop that goes through all array values in the range.

Bit Manipulation:

- All data in computer programs is internally stored as bits, i.e., as numbers 0 and 1. This chapter discusses the bit representation of integers, and shows examples of how to use bit operations. It turns out that there are many uses for bit manipulation in algorithm programming.
- Multiple such Bit operations can be - AND, OR, NOT, XOR, Shifts, Complements.

Basics of Graphs:

- Many programming problems can be solved by modelling the problem as a graph problem and using an appropriate graph algorithm. A typical example of a graph is a network of roads and cities in a country. Sometimes, though, the graph is hidden in the problem and it may be difficult to detect it.
- All graph terminologies should remain properly defined in our knowledge.
- Different graph representation techniques should be well understood.

Graph Traversals:

- Different type of Graph algorithms, e.g. DFS, BFS, Kruskal, Prim, Dijkstra, Warshall, Floyd, Bellman Ford etc.
- Some of the related Graph problems are Connectivity check, Finding cycles, Bipartiteness check etc.

Tree algorithms & Spanning trees:

- A tree is a connected, acyclic graph that consists of n nodes and (n−1) edges. Removing any edge from a tree divides it into two components, and adding any edge to a tree creates a cycle. Moreover, there is always a unique path between any two nodes of a tree.
- General graph traversal algorithms can be used to traverse the nodes of a tree. However, the traversal of a tree is easier to implement than that of a general graph, because there are no cycles in the tree and it is not possible to reach a node from multiple directions.
- A spanning tree of a graph consists of all nodes of the graph and some of the edges of the graph so that there is a path between any two nodes. Like trees in general, spanning trees are connected and acyclic.

Some of the global contests company exams:

Some of the popular global contests are -

- ICPC (International Collegiate Programming Contest)
- IOI (International Olympiad in Informatics)
- Snackdown
- Google Codejam
- Google Kickstart
- Facebook Hacker cup etc

Some companies conducted Competitive Coding exams are Google (Google Kickstart and Google Codejam), Infosys (HackWithInfy, InfyTQ), TCS (TNQT, CodeVita) etc.

All Competitive Exam Notifications Available at: https://clist.by/

What are online judges?

An online judge is an **online system to test programs in programming contests**. They are also used to practice for such contests. Many of these systems organize their own contests. Some of the such coding portals are:

- **Codeforces, Codechef, HackerRank, Hackerearth,**
- **Atcoder, SPOJ, Leetcode, InterviewBit, binarysearch.io,**
- **Codewars, Coderbyte, Devskill, Codingame, Geeksforgeeks,**
- **Project Euler, Codecombat, Codingbat, Codility, Topcoder,**
- **Codingninjas, codingblocks.com etc.**

References of some Learning Resources:

- **Foundation Learning Resource:** https://certifications.codechef.com/data-structures-and-algorithms/prepare/foundation-level

- **Advanced Learning Resource:** https://certifications.codechef.com/data-structures-and-algorithms/prepare/advanced-level

Riddles
&
Logic Puzzles

With Arnab Chakraborty

Let’s learn Competitive Programming Together

Logic Puzzle – 1 (Hard Times)

Flynn has fallen on hard times, turned to drink and ended up a homeless tramp on the streets. He collects cigarette butts and uses the tobacco to roll his own cigarettes. For every 8 cigarette butts he finds, he rolls 1 full cigarette for himself. He has just collected 64 cigarette butts. How many cigarettes can he make?

Logic Puzzle – 1 (Hard Times)

Flynn has fallen on hard times, turned to drink and ended up a homeless tramp on the streets. He collects cigarette butts and uses the tobacco to roll his own cigarettes. For every 8 cigarette butts he finds, he rolls 1 full cigarette for himself. He has just collected 64 cigarette butts. How many cigarettes can he make?

Answer - 9. He can make 8 from the 64 he’s collected, and then an additional cigarette from those 8 once he’s smoked them.

Logic Puzzle – 2 (The Cork and Bottle)

A bottle costs a dollar more than a cork. Together they cost 110 cents. How much does the bottle cost and how much does the cork cost?

Note: (1 dollar = 100 US cents)

Logic Puzzle – 2 (The Cork and Bottle)

A bottle costs a dollar more than a cork. Together they cost 110 cents. How much does the bottle cost and how much does the cork cost?

Answer - The right answer is that the bottle costs 105 cents, and the cork costs 5 cents.
(1 dollar = 100 US cents)

Logic Puzzle – 3 (Bird Long I)

A bird has a head 9 cm long. The tail is equal to the size of the head plus a half of the size of the body. The body is the size of the head plus the tail.

How long is the bird?

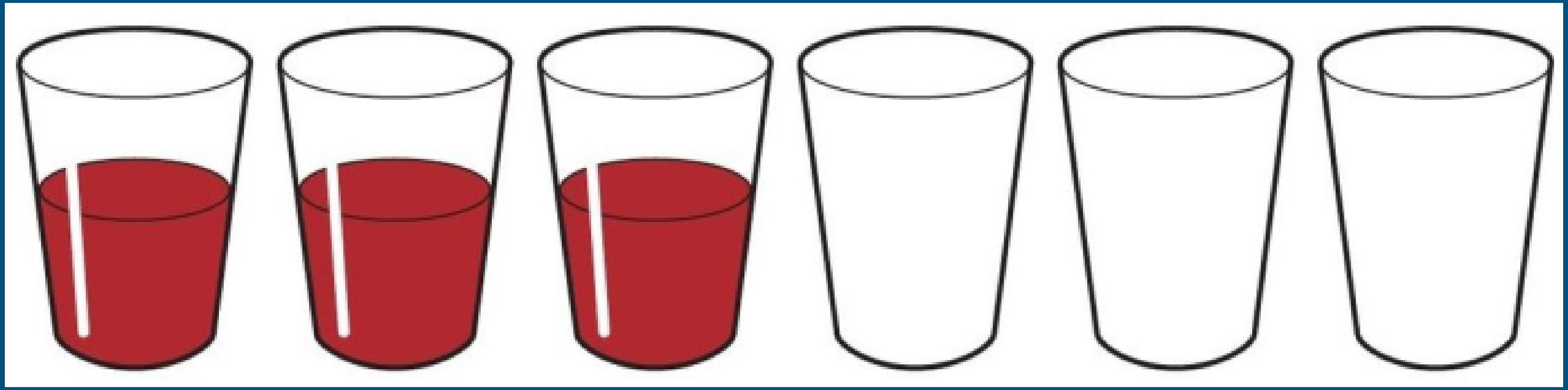
Logic Puzzle – 3 (Bird Long I)

A bird has a head 9 cm long. The tail is equal to the size of the head plus a half of the size of the body. The body is the size of the head plus the tail. How long is the bird?

Answer - 72 cm. The head is 9 cm. The tail is 18 + 9 = 27 cm. The body is 9 + 18 + 9 = 36 cm. 9 + 27 + 36 = 72 cm.

Logic Puzzle – 4 (Half Full, Half Empty)

There are six glasses in a row. The first three are full of wine, the last three are empty. By moving only one glass, how can you set them up so that full and empty glasses are lined up alternately?



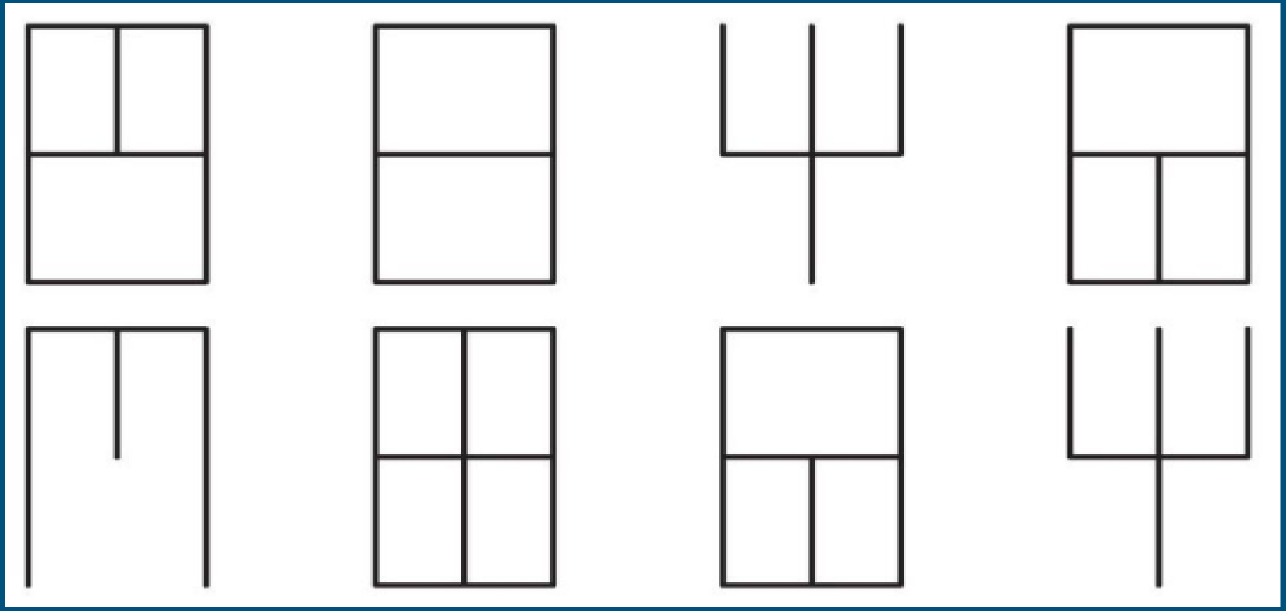
Logic Puzzle – 4 (Half Full, Half Empty)

There are six glasses in a row. The first three are full of wine, the last three are empty. By moving only one glass, how can you set them up so that full and empty glasses are lined up alternately?

Answer - Simply empty the contents of the second glass into the fifth glass.

Logic Puzzle – 5 (Hieroglyphs)

Look at these digital hieroglyphs, reflect on them a little, and then work out what the sixteen-digit number sequence is.



A hieroglyph (Greek for "sacred carvings") was a character of the ancient Egyptian writing system. Logographic scripts that are pictographic in form in a way reminder of ancient Egyptian are also sometimes called "hieroglyphs".

Logic Puzzle – 5 (Hieroglyphs)

Look at these digital hieroglyphs, reflect on them a little, and then work out what the sixteen-digit number sequence is.

Answer - The numbers are mirror images of themselves. The answer is 99 33 44 66 77 88 66 44.