**ICPC Brainstorm**

**Concrete**

* Inequations for reading and finding solutions.
* About Online Judges (OJ) and problem reading.
* About possible interactions with OJs.
* Some types of correctness proofs: induction, contradiction hypothesis, etc.
* Topics:
  + Strategies: Brute Force1, Greedy2, Dynamic Programming (DP)3, Divide and Conquer3, Two Pointers2, Naïve1, Adhoc2.
  + Math: Number Theory1, Probability and Statistics3, Modular arithmetic2
  + (Pure) algorithms or groups of algorithms: Binary Search1, Union-Find2, Fast Pow2, Fermat Little Theorem2, Minimum Paths in graphs3 (Dijkstra, Floyd), minimum spanning trees3 (Kruskal, Prim), KMP3 (Knuth Morris Pratt), Rolling Hash3.
  + Data Structures: Iterable1 (Array, Vector, List, etc.), BIT2.5, Euclid’s sieve2, Segment Tree2, Stack3, Queue3, Deque2, Linked list3, Graphs3 (Adjacency Matrix, Adjacency List, etc.), HLD4 (heavy light decomposition), Suffix Array4.
* Books:
  + Any Discrete Math Book (especially if you are not so used to read/write formal math).
  + Competitive Programming *X* | X{1,2,3,4}
  + Cormen’s Introduction to Algorithms
  + Useful training mirrors: Codeforces, Codechef, SPOJ, COJ

**Tricks**

* The usage of *macros (defines) and typedefs*.
* The usage of *typedefs*.
* Own Local Judge or using *while (Testcases--)* trick.

**Philosophical**

* Talk about every topic just as specific idea and **NOT just as a plain tool.** combinations of ideas + construction/deconstruction are generally the path I consider the truth one because it’s more generic.
* Truth of current Mexico & Latin America Level in ICPC (IMO, probably cultural stuff, including but not limited to natural language)
* Learn maybe another language, besides English, which is native of top competitive programmers (interesting experiment)
* Don’t focus in being experts of the programming language(s) you use, can be beneficial knowing more advanced or newer features gradually to write code faster and more concise, but it’s not determinant to be better in competitive programming.
* Practice and Practice is very important but beware of adhoc holes.
* Understanding what programming paradigms are useful and to what extent (structured1, OOP2, Functional2).
* Some say knowing basic Calculus is beneficial but haven’t totally understand yet why 🤔 My hypothesis is that it may help understanding general patterns (concepts and tricks) in Math.