Olympiads in Informatics, 2026, Vol. 20, 109–122

DOI: https://doi.org/10.15388/ioi.2026.07

On the Recurrent Themes of Constructives and Interactives in Informatics Olympiads

Troy Dylan SERAPIO[1], EdRonn James PANTI[2], Farrell Eldrian WU[3]
[1]Philippine Science High School Main Campus, Quezon City, National Capital Region, Philippines
[2]Philippine Science High School Main Campus, Quezon City, National Capital Region, Philippines
[3]Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America
e-mail: b2027tdserapio@pshs.edu.ph, b2026ejtpanti@pshs.edu.ph, farrellw@mit.edu

Abstract. Constructive and interactive tasks appear regularly in major informatics olympiads. Yet, they are often treated as ad hoc extensions of standard paradigms rather than as domains with recurring reasoning structures. This paper studies a curated set of constructive and interactive problems drawn from TOKI (Tim Olimpiade Komputer Indonesia), SGNOI (Singapore National Olympiad of Informatics), and EGOI (European Girl's Olympiad in Informatics) archives. For each selected task, we prepared an abridged solution write-up and extracted the underlying theme (central conceptual mechanism) and heuristic (meta-level reasoning pattern that guides discovery). Synthesizing across the curated set yields three recurring themes (i.e., Balanced Queries, Decomposition of Graphs, and Iterative Candidate Refinement) and five problem-solving heuristics. To concretize such themes, we present representative problems illustrating them and discuss how these themes characterize constructives and interactives beyond surface-level algorithms and data structures.

Keywords: competitive programming; olympiads; interactive problems; constructive problems; thematic analysis; problem-solving heuristics

1. Introduction and Background

1.1 Background on Constructive and Interactive Problems in Informatics Olympiads

Competitive programming has long served as a rigorous domain for algorithmic reasoning and computational problem solving. Over time, the diversity of problems encountered in contests has led to the formalization of well-known categories such as dynamic programming, graph theory, and greedy algorithms — each extensively analyzed and systematized in both educational and research contexts. Yet, not all kinds of problems have received the same degree of theoretical attention. Among these are constructive and interactive problems, categories that appear regularly in contests but remain largely dismissed as extensions of existing paradigms.

A constructive problem is focused on the explicit construction of an object, structure, or sequence that satisfies a given set of constraints. This includes designing a valid graph, permutation, or configuration. To be classified as a "constructive," the main challenge of the task lies in synthesizing a structure that fulfills the required conditions. In contrast, an interactive problem involves dynamic information exchange between the contestant's program and the judge, often requiring the program to adapt as new data is revealed by the judge.

However, despite their prevalence in major olympiads such as the IOI and ICPC, both remain underexplored, especially as unified domains of study. Thus, this paper examines curated problems from TOKI, SGNOI, and EGOI to identify recurring themes and heuristics that concretize the reasoning frameworks of these two conceptually-rich yet underexplored classes of problems.

1.2 Research Questions

This paper presents an exploratory qualitative analysis of constructive and interactive problems in competitive programming, particularly identifying recurring themes and problem-solving heuristics that characterize this class of problems. In this context, a theme refers to the central conceptual structure underlying a problem's solution, while a heuristic denotes the meta-strategies that guide the solver toward that idea.

For the identification of themes, we first create concise solution write-ups for each problem to ensure full understanding of the solutions. We then apply inductive thematic analysis (Braun & Clark, 2022) to these write-ups together with the official editorials of representative problems curated from TOKI, SGNOI, and EGOI archives. The objective of this analysis is to systematically examine how solutions are structured, identify recurring conceptual patterns, and synthesize these patterns into a small set of overarching themes that capture the nature of constructive and interactive problems.

For the identification of heuristics, we use heuristic inquiry (Moustakas, 1990) grounded in the lived experiences of the authors as competitive programmers. This process involves reflective examination of the authors' problem-solving experiences in the curated problem archive to propose practical heuristics that align with the identified themes. The goal is not to claim universal strategies, but to exhibit meta-strategies that may guide solvers when approaching constructive and interactive tasks.

1.3 Data Processing Procedures

With the objectives outlined above, we examined problem archives from the following sources: TOKI (Tim Olimpiade Komputer Indonesia), SGNOI (Singapore National Olympiad in Informatics) from 2017 to the present, and EGOI (European Girls' Olympiad in Informatics). Problems were initially identified through contest archives and platform tags indicating constructive or interactive problem types. TOKI problems were filtered directly

using the constructive and interactive tags on the TLX platform, while SGNOI and EGOI problems were manually screened from their respective archives on codebreaker.xyz and QOJ.ac.

From this initial pool, we applied a second set of criteria: interactive problems must involve adaptive interaction with a judge; constructive problems must require producing a valid configuration where the main difficulty lies in devising the construction rather than optimizing bounds; problems exhibiting fundamentally different or heterogeneous solution paradigms were excluded to maintain consistency in analysis; problems had to be non-trivial, typically containing multiple subtasks and requiring more than a single isolated trick; and problems exceeding an estimated difficulty of around 2000 on the Codeforces scale were excluded unless accompanied by a clear editorial. This process resulted in a final curated set of sixteen TOKI problems, three SGNOI problems, and three EGOI problems.

For every selected problem, we prepared a concise write-up summarizing the core solution and reducing it to its essential conceptual structure. These write-ups were primarily developed by the lead author, with additional contributions from co-authors on selected problems, and served as the primary data for our inductive thematic analysis. Instead of applying predefined categories, we first distilled each solution into its conceptual "barebones," grouped solutions that shared similar reasoning structures, and iteratively refined these groupings into three overarching themes that capture recurring patterns in constructive and interactive problem solving. Additional problems, beyond what will be thoroughly discussed in this paper, may be found in Appendix B.

In parallel, we conducted a heuristic inquiry grounded in our lived experiences as competitive programmers. Through reflective examination of the reasoning processes involved in solving and analyzing the curated problems, we synthesized five heuristics aligned with the identified themes. Each pairing of Theme and Heuristic was supported by a brief analytical justification explaining its relevance. Our goal is to emphasize conceptual reasoning and solver decision-making rather than implementation details, guiding the reader through the structure and motivation behind constructive and interactive solutions while maintaining an analytical and accessible presentation style. Some common terminologies in competitive programming are used for brevity and familiarity; concise definitions are provided in Appendix A.

2. Balanced Queries

In many interactive problems, the judge's responses are drawn from a small, discrete set, with binary feedback such as {yes, no} being a common example. The central idea behind balanced querying is to design each query so that all possible responses yield comparable amounts of information about the hidden state. By ensuring that no outcome is significantly more informative than another, each interaction contributes meaningfully toward narrowing the search space, reducing the worst-case total number of queries needed to reach a solution. Problems employing this theme often rely on reasoning analogous to

binary search or information-theoretic decision making, where the objective is to partition the remaining candidate space into subsets of roughly equal size in order to maximize expected information gain.

2.1 Maybe Guess the Number (OSN Informatika 2015, Day 1C)

The first problem that we will tackle is called "Maybe Guess The Number" from the TOKI archive. To summarize this problem, Kwek must determine an unknown integer x between 1 and n by asking at most k questions. In each question, Kwek submits a set of integers S, where S is a subset of the numbers from 1 to n, and Kwak responds with one of three possible answers: YES if x is in S, MAYBE if x is not in S but either $x - 1$ or $x + 1$ is in S, and NO otherwise. After at most k queries, Kwek must correctly identify the value of x. It is implicitly stated that a solution that runs with query complexity$O(k) \leq O(\log N)$ is required to obtain full points.

The key observation lies in analyzing the behavior of an arbitrary queried set S, shown in the list below:

- If $x \in S$, the response will be YES. This indicates that one of the elements inside the set is definitely x. Thus, we can call all of the elements "YES numbers" (as they collectively triggered the YES response).
- If $x \pm 1 \in S$, the response will be MAYBE. This means that one of the elements adjacent to those in the set (but not in the set itself) is definitely x. Thus, we can call all of the elements "MAYBE numbers."
- Otherwise, the response will be NO, signifying that none of the elements in or adjacent to S can be x. Thus, we can call all of the elements "NO numbers."

Once these notions of labels on numbers are established, the guessing game transforms into partitioning a search space (particularly, the number line). The central idea is to design each query so that the three possible labels of numbers (YES, MAYBE, NO) partition the numbers as evenly as possible. Thus, a possible balanced pattern (on some number line) is shown below:

$$[ABBA][ABAB \ldots AB][CC \ldots CC]$$

where A means "YES," B means "MAYBE," and C means "NO." The set queried to the judge is composed only of numbers with labels A, while the other labels follow as consequences of adjacency. After each query, all numbers inconsistent with the judge's response are eliminated, and the remaining candidates are recolored according to the same scheme. Since every round reduces the search space by approximately a constant factor, the total number of queries grows logarithmically with respect to n, yielding a query complexity of $O(\log_3 n) \sim O(\log n)$.

This reasoning illustrates the Balanced Queries theme: each query is designed to maximize guaranteed information by minimizing the disparity among possible outcomes (YES, MAYBE, NO). Specifically, the search space is partitioned into three equally sized subsets, each corresponding to one response. As a result, regardless of the outcome, each interac-

tion reduces the remaining candidates by roughly the same proportion, ensuring that the querying process remains balanced.

2.2 One-Day Delivery (OSN Informatika 2023, Day 1C)

An unknown shoe size s, where $1 \leq s \leq n$ must be determined by the night of the day t. Each morning, you may make exactly one shoe of an integer size at most n and ship it; the shoe arrives on the night of the following day. Upon arrival, you receive a message corresponding to the shipment from the previous day: KEKECILAN if the shoe is too small, KEBESARAN if the shoe is too big, or PAS if the shoe fits, in which case the program immediately ends with an accepted verdict. The task is to design a daily schedule of shoe sizes that guarantees the correct size will be identified by night t, despite the one-day delay in feedback due to the query complexity $O(\log N)$.

We need to identify a hidden integer $x \in [1,n]$ using ternary feedback: KEKECILAN (our guess is too small), KEBESARAN (too large), or PAS (correct). The twist is that feedback is delayed by one turn: after issuing a query q_t, we only receive the judge's response to $q_{(t-1)}$. This means we cannot immediately update the search interval based on the number we just asked, thus making a standard binary search break (this only gives 16/100 points in the original task).

To cope with the one-step delay, we structure each "decision" through two consecutive queries that effectively partition the current search space $[l,r]$ into three blocks:

$$[l, a-1]\ a\ [a+1, b-1]\ b\ [b+1, r]$$

where we ask a first and b next. Once the delayed response arrives, it tells us whether the secret is to the left, between, or to the right of these pivots (or equals one of them). The key is to choose a (and hence how the interval is split) so that no matter which delayed outcome occurs, the remaining uncertainty is as small as possible given that we will already have spent (or be spending) two queries.

The optimal way to place these pivots follows a Fibonacci-style partition: maintain interval lengths that decrease according to consecutive Fibonacci numbers. Intuitively, because each refinement consumes one new query while "processing'' the previous response, the worst-case remaining interval after a delayed step should match the recurrence of the Fibonacci search. Implement this via a helper *getNext(l,r)* that returns the next query position using the largest Fibonacci decomposition fitting $[l,r]$, ensuring the three resulting regions are sized so that the worst-case branch remains optimal.

Algorithmically, keep the current feasible range $[l,r]$ and a variable holding the previous response. Repeatedly compute the next query $q=getNext(l,r)$ and print q; then read the response to the previous query and shrink $[l,r]$ accordingly: if it is KEBESARAN, set $r=q_{prev}-1$; if KEKECILAN, set $l=q_{prev}+1$; if PAS, stop. The Fibonacci placement guarantees that, despite the one-turn lag, each new query is "pre-scheduled" so that every possible delayed outcome leaves an optimally small remaining search interval.

3. Decomposition of Graphs

Many constructive problems operate by decomposing a complex graph or structure into simpler subcomponents, solving each subcomponent individually, and then reassembling them into a complete solution. This approach often starts with identifying simpler fragments that can be independently handled. Once these components satisfy local constraints, they are combined to form a valid global construction. This theme captures the reasoning process of reducing a global construction problem into a family of simpler local constructions whose properties collectively imply the general case.

3.1 Autosynthesis (COMPFEST 15 Final CPC Junior 2A)

Chaneka is given an array A of n positive integers, initially with no elements circled. In each operation, she may circle any element of A, possibly circling the same index multiple times. After all operations, she constructs a sequence R consisting of all uncircled elements of A in increasing order of indices, and a sequence P where P_i denotes the index circled in the i-th operation. Determine whether it is possible to choose operations so that $R=P$, or report that it is impossible; if multiple solutions exist, output any one. From the problem statement, the required algorithmic time complexity to earn full points is $O(N)$.

We model the array as a functional graph with nodes $1,\ldots,n$, where each node i has a directed edge i to $A[i]$. For an edge u to v, we call u a parent of v and v a child of u. Thus, each node will be labeled either CIRCLED (belongs in P) or UNCIRCLED (belongs in R). This leads to the following set of rules by translating the array representation into the functional graph representation:

- If all parents of a node k are circled, then all array elements with value k are removed from R, so k they cannot appear in P; hence, k must be uncircled.
- If at least one parent of k is uncircled, then k appears in R, so it must also appear in P, implying that k must be circled.

These rules allow us to determine the coloring for all nodes reachable from any node with an in-degree 0. We start from such nodes and perform a BFS, propagating colors using the rules above. Any remaining unvisited nodes must lie in connected components that are purely cycles. In these components, nodes must be colored alternately CIRCLED and UNCIRCLED; this is possible if and only if the cycle length is even, while odd cycles make the condition $R=P$ impossible. The entire procedure runs in $O(N)$ time (BFS and iterating through the cycles).

This task follows the idea of graph decomposition by breaking the global requirement $R=P$ into simpler, locally checkable conditions on the functional graph defined by i to $A[i]$. The graph naturally decomposes into components reachable from nodes with in-degree 0 and components that are purely cycles. The former can be handled independently using a BFS that propagates forced colorings based only on parent-child relationships. At the same time, the latter reduces to standalone cycle problems whose validity depends solely on parity. Each component is solved in isolation using local rules, and the full construction is

obtained by combining these solutions, making the overall approach a direct application of decomposing a global construction into simpler subgraph-level constructions.

3.2 Social Engineering (EGOI 2022 Day 1(3))

A social network is modeled as a connected undirected graph with n vertices and m edges, where each edge may be used at most once. Starting from Maria at vertex 1, players alternately challenge a friend by traversing an unused edge, forming a walk that never repeats an edge. A player loses if they have no legal move on their turn. Maria always moves first and plays optimally whenever she has a winning strategy, while the remaining players cooperate to force her to lose if possible. Given the full graph in advance, the task is to determine whether Maria has a forced win, or otherwise to coordinate the friends' moves so that she is eventually left without a legal move. The algorithm must correctly handle all optimal Maria strategies and run in $O(n+m)$.

To solve this problem, the first step is to model the game directly as the given graph, with Maria fixed at vertex 1. The neighbors of Maria are the only vertices through which play can move between her position and the rest of the graph, so we call them entry vertices. Removing Maria partitions the graph into connected components, each containing some number of entry vertices. The key observation is that the outcome depends solely on the parity of these entry vertices within each component. If any component contains an odd number of entry vertices, then Maria has a winning strategy. Once she enters such a component, parity guarantees that the remaining players will eventually run out of moves first. In this case, no coordinated response can prevent her from winning.

Conversely, if every component contains an even number of entry vertices, then the remaining players can force Maria to lose. The strategy is to pair the entry vertices within each component so that each pair is connected by a path that does not conflict with other pairs. Whenever Maria enters a component through some entry vertex, the friends respond by following the precomputed path to its paired entry vertex and then returning the move to Maria. This enforces a controlled entry-and-exit structure within each component and prevents Maria from creating a position where the friends are trapped without a reply.

To construct these pairings efficiently, it suffices to work on a spanning tree of each component. For every component, select an arbitrary spanning tree and perform a depth-first search. Each subtree pairs its entry vertices locally and propagates at most one unmatched entry vertex to its parent; subtrees containing an even number of entry vertices propagate none. This process succeeds if and only if the total number of entry vertices in the component is even. The pairing construction and the resulting strategy can be implemented within a single DFS traversal per component, giving an overall time complexity of $O(n+m)$.

This solution illustrates graph decomposition as a unifying principle. Removing Maria separates the global game into independent components whose outcomes are determined solely by parity conditions. Each component is then reduced to a spanning tree, since additional edges do not affect the pairing argument. By solving each component independently

and combining the results, the complex interactive game reduces to a collection of simple structural checks and local constructions, leading to an elegant strategy for defeating Maria.

4. Iterative Candidate Refinement

Many constructive and interactive problems begin by proposing one or more candidate entities that could satisfy the constraints or goal of the problem. The solver then iteratively tests, eliminates, or refines these candidates until a correct or optimal one emerges. This theme often appears in problems where the search space is too large for full brute-force enumeration, but structured enough to allow progressive elimination. This idea closely parallels algorithmic search in artificial intelligence, where each piece of feedback guides systematic reasoning and adaptive decision-making toward the solution.

4.1 Colouring Balls (CKSN Informatika 2021 Day 1C)

You are given a sequence of balls, each having an unknown color represented by an integer. Your task is to construct an array *ans* such that balls with the same color are assigned the same material, while balls with different colors are assigned different materials. You may interact with the judge using a function *query(l,r)* that allows you to determine whether all balls in the interval [*l,r*] have the same color. The goal is to construct a valid *ans* while achieving a query complexity of $O(N \log M)$ and a time complexity of $O(N^2)$.

Thus, the solution is as follows. We first process the balls from left to right, assigning materials incrementally. Initially, we may arbitrarily assign $ans[1]=1$, since there is no prior information. Suppose we have already correctly assigned materials for positions 1 through k, and now want to determine the material of the ball $k+1$. The key observation is the following: If there exists an index $i \leq k$ such that balls at positions i and $k+1$ share the same color, then: $query(i,k)=query(i,k+1)$. Otherwise, if the color changes at the position $k+1$, the query result must differ.

To incorporate such an insight, we maintain an auxiliary array *lki*, where $lki[m]$ stores the last index that was assigned to material m. For each existing material m, we test whether $query(lki[m],k)=query(lki[m],k+1)$. If the equality holds, then the ball $k+1$ must share the same color as the ball at $lki[m]$, and we can assign $ans[k+1]=m$. Otherwise, if no such material satisfies this condition, then the ball $k+1$ represents a new color, and we assign it a new material. To optimize the identification of the correct material, we can binary-search over the materials using the monotonicity of query outcomes as we traverse the last-known indices in increasing order. This allows us to narrow down the matching material efficiently instead of checking all previous materials.

This problem exemplifies iterative candidate refinement: for each new ball, we start with the set of all previously seen colors as potential candidates and systematically eliminate the materials that are inconsistent with query results. The algorithm never attempts to guess a color outright, but it progressively refines the candidate set using information

derived from earlier assignments. Once all invalid candidates are ruled out, the remaining one (if any) must be correct.

4.2 Detecting Gold (OSN Informatika 2019 Day 2B)

You are given a grid and an unknown pair of points containing gold. You may issue queries of the form: choose a point *(x,y)*, and receive the Manhattan (taxicab) distance to the nearest gold point. Using 5 queries, you must determine the exact locations of the gold points.

A key geometric observation is that the locus of points at taxicab distance *T* from a fixed center forms a square rotated by *45°*. If the center lies on a corner of the grid, then three sides of this square lie outside the grid, and the remaining feasible points form a straight line. We exploit this by querying the four corners of the grid. Each query produces a distance, which defines a line of candidate points that must contain at least one gold point.

After the four corner queries, we obtain four such lines. By the Pigeonhole Principle, each gold point must lie at the intersection of two of these lines. Moreover, valid gold positions must correspond to opposite intersections. We can then issue a fifth query at one of the intersection points to determine which opposite pair is valid. Simple boundary checks immediately discard any intersections that fall outside the grid. Thus, the time complexity and query complexity are both *O(1)*.

This solution demonstrates iterative refinement at a geometric level. Each query dramatically reduces the space of possible gold locations: from the entire grid, to lines, to intersection points. Rather than searching exhaustively, the algorithm repeatedly restricts the candidate set using the idea of the “diagonals”, until only four possibilities remain. The final query merely disambiguates among these candidates.

5. Heuristics

In addition to the three main themes that were derived from the thematic analysis, an additional five (5) heuristics were also derived from the heuristic inquiry done on the authors. These heuristics, similar to the themes, are also given names to encapsulate the purpose of such a heuristic. Thus, these heuristics include: (1) Fake Array, (2) Reduction Heuristics, (3) Wishful Thinking, (4) Subtask Decomposition, and (5) Brute Force Heuristics. Problems that best showcase these heuristics may be found in Appendix B.

5.1 Fake Array

Some problems appear to revolve around direct array manipulation, such as updating indices, simulating transitions, or repeatedly applying rules to positions, but the array is often only an encoding of a deeper structure. The Fake Array heuristic consists of deliberately reinterpreting the array as something else, most commonly a graph. A typical case is a functional graph (for example, in Autosynthesis), where each position maps to another

position according to a given rule. Recasting the problem this way often reveals underlying structure (e.g., cycles, trees rooted at cycles, or other recognizable graph forms) which then allows the task to be solved using standard graph techniques (e.g., cycle finding, DFS, and related algorithms).

5.2 Reduction Heuristics

If a problem features an odd query format or an unconventional output definition, this is often a hint that the query can be reformulated into something simpler. This heuristic emphasizes rewriting the queries into a representation that is easier to analyze, sometimes by adopting an entirely different viewpoint (for instance, in Maybe Guess The Number, the YES, MAYBE, and NO responses can be reinterpreted as a coloring problem). Once the superficial complexity is removed, many problems become manageable when the query is understood in terms of the actual quantity it computes. This pattern appears repeatedly across solved problems and reflects the broader Computer Science principle of reduction: in long-form contests, identifying the right reduction is often an essential part of the problem itself.

5.3 Wishful Thinking

Rather than immediately attempting to solve the entire problem, it can be highly effective to experiment with special cases, constraints, or imagined scenarios. By assuming certain properties hold and examining their consequences, one can uncover contradictions or extract useful insights. This intentional form of "wishful thinking" frequently leads to the discovery of invariants, monotonic behavior, or tight bounds that are not obvious at first glance. It is a broadly applicable technique in competitive programming and is especially powerful in constructive and interactive problems, where creative reasoning is often required.

5.4 Subtask Decomposition

Decomposing a problem using its subtasks is a common competitive programming strategy, but it should be applied with discretion. In some OI-style problems, subtasks may be designed around a particular intended approach that does not match how you naturally think about the problem. In these situations, overemphasizing subtasks can obscure clearer lines of reasoning. Nevertheless, subtasks remain valuable, as they frequently highlight constraints, corner cases, or key structural observations. This is particularly evident in problems like Social Engineering, where early subtasks clarify what constitutes a losing configuration, and later subtasks build upon this understanding to derive a guaranteed winning strategy.

5.5 Brute Force Heuristics

At times, the most effective initial approach is the simplest one. Writing a brute-force solution helps illuminate how the problem behaves, reveals recurring patterns, and provides a reference point for verifying more efficient methods. Even if the brute-force approach exceeds time or query limits, it can be incrementally improved, and having a solid grasp of the naive solution is often crucial during a contest.

6. Conclusion and Further Work

This paper presents an exploratory qualitative analysis of constructive and interactive problems, with a particular focus on identifying recurring themes and solver heuristics. Overall, the study provides a structured perspective on patterns that are often recognized informally by experienced competitive programmers, while offering a clearer conceptual framework for understanding these classes of problems.

6.1 Summary of Findings

From the thematic analysis of curated problems, the primary observation is that many constructive and interactive tasks share a small number of recurring conceptual structures despite appearing diverse at the surface level. By distilling solution write-ups into their essential reasoning patterns, we identified three overarching themes that capture common modes of thinking across problems from TOKI, SGNOI, and EGOI. These themes highlight how problems frequently revolve around reframing queries, constructing valid configurations under implicit constraints, or exploiting structural guarantees provided by the problem statement.

Meanwhile, the heuristic inquiry grounded in our lived experiences as competitive programmers revealed five recurring heuristics that guide solver decision-making. Rather than focusing on specific algorithmic techniques, these heuristics describe higher-level reasoning patterns that help reduce complexity and expose the underlying structure of a problem. Across the analyzed problems, these strategies consistently appeared during both the initial exploration phase and the refinement of solutions, suggesting that they represent transferable modes of thought rather than isolated tricks.

6.2 Recommendations for Further Research

There are several directions in which this work can be extended to deepen the understanding of constructive and interactive problem-solving. As a first step, it would be valuable to expand the pool of reference problems by incorporating tasks from additional informatics olympiads, such as the IOI, as well as ICPC contests. These sources are known for their depth and diversity, and a broader dataset would help validate the generality of the proposed themes while potentially revealing new recurring patterns.

Another promising direction is to refine and extend the current thematic framework. While the three themes identified in this study already capture meaningful structural insights, further refinement or the introduction of additional categories may allow for a more precise characterization of different problem types. Any expansion, however, should preserve the generality of the themes so that they remain applicable across a wide range of contests rather than becoming overly specialized.

Finally, there is significant potential in developing a systematic classification system based on the identified themes and heuristics. A searchable index or database could be constructed to organize problems according to their underlying reasoning patterns, functioning similarly to existing problem tags but operating at a more conceptual level. Such a resource could support both learning and research by helping programmers identify problems that share similar modes of thinking.

Overall, we are satisfied with the outcomes of this work, as the process of conducting the analysis has already deepened our own understanding of constructive and interactive problems. We hope that this paper will likewise provide competitive programmers with a new perspective on these problem classes and encourage further exploration of their underlying structure.

References

Braun, V., & Clarke, V. (2022). Thematic analysis: A practical guide. SAGE Publications. European Girls' Olympiad in Informatics. (n.d.). Official website. https://egoi.org

Moustakas, C. (1990). Heuristic research: Design, methodology, and applications. SAGE Publications.

Singapore National Olympiad in Informatics. (n.d.). SGNOI archive [Computer software]. GitHub. https://github.com/noisg/sg_noi_archive

TLX TOKI Platform. (n.d.). TLX – Competitive Programming Training Gate. https://tlx.toki.id

Appendix A

For the purposes of this paper, the following terms are defined as follows:

- Program: The solution submitted by the participant to the online judge for evaluation.
- Judge: The automated system responsible for evaluating submitted programs by executing them on a predefined set of test cases and verifying the correctness of their outputs.
- Test Cases: Sets of input data provided to a program to produce corresponding outputs, which are then compared against the official correct outputs for validation.
- Query: An interaction in which the program requests additional information from the judge during the execution of an interactive problem.
- Query Complexity: The asymptotic behaviour (usually upper-bound) of the number of queries a program uses to solve the problem. This is expressed as a function of the input size, denoted through asymptotic notation (colloquially known as Big-O notation).
- Time Complexity: The asymptotic behaviour (usually upper-bound) of the number of operations a program uses to solve the problem. This is expressed as a function of the input size, denoted through asymptotic notation (colloquially known as Big-O notation).

Appendix B

This is a list of more problems that fall under the themes and heuristics discussed in this paper.

Balanced Queries:

- Toxic Gene from SGNOI 2023 Finals (Task 5) | *Hint: consider a binary representation*
- Detecting the Naughty Village Thief from COMPFEST 13 Penyisihan CPC Junior | *Hint: think in 1D first*

Decomposition of Graphs:

- Sharing Lapis Talas from OSN Informatika 2023 | *Hint: solve the line graph, then what can you say about a tree?*
- Game from IOI 2014 Day 1-3 | *Hint: Solve the problem in a direct way, then consider the edge case of two components that are disconnected and try to work around that.*

Iterative Candidate Refinement:

- Mining Gold from TOKI Open Contest 2018 Day 1-1 | *Hint: what does isIntegerDistance mean in a number theory context?*
- Building Tunnels from OSN Informatika 2022 Day 1-B | *Hint: brute-force all of the possible A edges that you can find*

Fake Array:

- Lost Array from SGNOI NOI 2019 Prelims Task 2 | *Hint: do floodfill to assign a Ci for each element in the array*
- Magic from TOKI Open 2017 Day 2-1 | *Hint: there is "matching" with the deck*

Reduction Heuristics:

- Message from IOI 2024 Day 1-2 | *Hint: We need to deduce which bits aren't evil. Can we model this as a graph instead by looking at the first few sent bits of each cell?*
- Weird Chickens from OSN Informatika 2015 Day 2-C | *Hint: find the blocks of DNA first before finding the sequence of said blocks*

Wishful Thinking:

- Carnival General from EGOI 2023 Day 2-A | Hint: consider induction from 2 generals to N generals
- Square or Rectangle? from SGNOI 2015 Prelims Task 4 | *Hint: which cells must be filled for the shape to be a square?*

Subtask Decomposition:

- Building Bridges from OSN Informatika 2022 Day 2-A | *Hint: Ignore the bridges and first break the graph into connected components with limited "connection slots"*
- Split the Attractions from IOI 2019 Day 1 | *Hint: Think about what a single well-chosen DFS cut tells you about the sizes of the resulting components.*

Brute Force Heuristics:

- Scales from IOI 2015 Day 1-2 | *Hint: 6! < 3^6.*
- Lands and Glaciers from OSN Informatika 2017 | *Hint: what's an easy way to construct F*

Acknowledgements

We are deeply grateful for the help of Mateo Campos and Jaime Maniago for their invaluable help in curating, solving, and preparing the write-ups for the selected problems. We also extend our sincere appreciation to the entire NOI.PH community for their meaningful insights on constructive and interactive problems.

Author Information

Troy Serapio - is a Grade 11 student at the Philippine Science High School – Main Campus. He is an active participant in the National Olympiad of Informatics Philippines, placing Top 6 in the 2025 IOI Philippine Team Selection and serving as a Teaching Assistant for the NOI Year-Long Training Program (NOI-YTP). He is also deeply interested in artificial intelligence and founded Tomorrow, a student-led initiative dedicated to promoting accessible and responsible AI education.

EdRonn James Panti - is a Grade 12 student at the Philippine Science High School – Main Campus. He is an active participant in the National Olympiad of Informatics Philippines, placing Top 3 in the 2025 IOI Philippine Team Selection and serving as a Teaching Assistant for the NOI Year-Long Training Program (NOI-YTP). He also participated in the International Olympiad of Informatics 2025 in Bolivia.

Farrell Eldrian Wu – is a PhD student in Operations Research at the Massachusetts Institute of Technology, studying information-theoretic algorithms for reinforcement learning and decision making. In high school, he won a gold medal at the IMO and a bronze medal at the IOI. Prior to starting the PhD, he worked in a quantitative research position in industry and as an undergraduate was extensively involved in teaching probability and statistics. He also coaches the Philippines' IOI 2024 team.