A Proposal for an IOI Syllabus

Tom Verhoeff* Gyula Horváth† Krzysztof Diks‡
Gordon Cormack§

June 2006 – Version 1.1– Appeared in
Teaching Mathematics and Computer Science, Vol. IV, Issue I (2006)

Abstract

The International Olympiad in Informatics (IOI) is the premier competition in computing science for secondary education. The competition problems are algorithmic in nature, but the IOI Regulations do not clearly define the scope of the competition. The international olympiads in physics, chemistry, and biology do have an official syllabus, whereas the International Mathematical Olympiad has made the deliberate decision not to have an official syllabus. We argue that the benefits of having an official IOI Syllabus outweigh the disadvantages. Guided by a set of general principles we present a proposal for an IOI Syllabus, divided into four main areas: mathematics, computing science, software engineering, and computer literacy.

Keywords and phrases: informatics competitions, mathematics, computing science, software engineering, computer literacy, computing science education, syllabus.

ZDM Subject Classification: B50, B60, B70, D50, M50.

1 Introduction

The International Olympiad in Informatics, abbreviated IOI, is an annual competition in the field of computing science [12]. It was modeled after the International Mathematical Olympiad and first held in 1989. The IOI is targeted at students in

*Faculty of Mathematics and Computing Science, Technische Universiteit Eindhoven, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands, `T.Verhoeff@TUE.NL`

†University of Szeged, Hungary, `horvath@inf.u-szeged.hu`

‡Warsaw University, Poland, `diks@mimuw.edu.pl`

§University of Waterloo, Canada, `gvcormac@uwaterloo.ca`

secondary education, especially those talented in computing science. It has become the premier competition in this area.

The current *IOI Regulations* [12] describe the IOI as *an annual international informatics competition*. These regulations constrain the form and organization, but they do not restrict the actual content of the competition. From the very beginning [14], the IOI competition has focused on problems of an *algorithmic nature*, see for example [4]. This much is stated in the annual Competition Rules.

In this article, we present a syllabus that better defines the actual scope of the IOI competition.

1.1 Overview

Section 2 takes a look at other international science olympiads and their approach to a syllabus. In Section 3, we explain the various benefits that an official IOI Syllabus could have. We also give some reasons for not having an official IOI Syllabus. Section 4 concerns the various roles of mathematics in computing science. We present some general principles to guide the design of an IOI Syllabus in Section 5. Section 6 contains our proposal for an IOI Syllabus. Section 7 concludes the article with a discussion.

2 Other International Science Olympiads

The international olympiads in physics (IPhO [13]), chemistry (IChO [8]), and biology (IBO [7]) have officially defined syllabi, somehow connected to their regulations. On the other hand, the International Mathematical Olympiad (IMO [11]) does not have an official syllabus, and this has been a deliberate decision. It appears that the younger international olympiads in astronomy (IAO [6]), geography (IGeO [9]), and linguistics (ILO [10]) do not have an official syllabus.

The "general Regulations for an IMO" state in C1:

> "The problems should, if possible, cover various fields of pre-university mathematics and be of different degrees of difficulty."

In current IMO practice, only four topics occur, but this is not an official policy: Geometry, Number Theory, Algebra, Combinatorics. Note that these topics are often not covered extensively (if at all) by mathematics curricula for secondary education in many countries.

The IPhO Statutes state in §5:

> "The theoretical problems should involve at least four areas of physics taught at secondary school level, (see Syllabus). Secondary school

> students should be able to solve the competition problems with standard high school mathematics and without extensive numerical calculation."

And the IPhO Regulations to §5 state:

> "The examination topics should require creative thinking and knowledge contained within the Syllabus. Factual knowledge from outside the Syllabus may be introduced provided it is explained using concepts within the Syllabus."

The IPhO Syllabus seems to cover all physics generally taught in secondary education. The *Theoretical Part* is divided into 11 subfields: Mechanics, Mechanics of Rigid Bodies, Hydromechanics, Thermodynamics and Molecular Physics, Oscillations and Waves, Electric Charge and Electric Field, Current and Magnetic Field, Electromagnetic Waves, Quantum Physics, Relativity, Matter. The *Practical Part* elaborates on measurement, instruments, errors, approximation and curve fitting, graphing, and safety in laboratory work.

The IChO Syllabus classifies topics on three levels:

Level 1 These topics are included in the overwhelming majority of secondary school chemistry programs and need not to be mentioned in the preparatory problems.

Level 2 These topics are included in a substantial number of secondary school programs and maybe used without exemplification in the preparatory problems.

Level 3 These topics are not included in the majority of secondary school programs and can only be used in the competition if examples are given in the preparatory problems.

The IChO Regulations state in §10 item (3):

> "The organizer cannot give theoretical problems of level 3 (Appendix C) from more than 3 fields and a minimum of 6 tasks should be presented in the set of preparatory problems from each field. Subjects assigned to level 3 can be classified as level 2 if sufficient background is included in the formulation of the problem (e.g. formulas, graphs, structures, equations)."

Also the IChO Syllabus seems to cover all chemistry generally taught in secondary education. The general part of the (new) syllabus is divided into 12 subfields

(10 pages in total): The atom, Chemical bonding, Chemical calculations, Periodic trends, Inorganic chemistry, Physical chemistry, Chemical kinetics, Spectroscopy, Organic chemistry, Polymers, Biochemistry, Analytical chemistry. A major part of the syllabus is devoted to safety and the handling and disposal of chemicals. The *Syllabus for the experimental part of the IChO competition* covers: Synthesis of inorganic and organic compounds, Identification of inorganic and organic compounds (general principles), Determination of some inorganic and organic compounds (general principles), Special measurements and procedures, Evaluation of results.

The IBO Rules state in §4.1, concerning the selection of topics for the competition:

> "All disciplines of biology are acceptable for the IBO."

In Appendix I, it is stated that "the Theoretical test [. . .] should cover the following 7 topics in the indicated proportions": Cell biology (20%), Plant anatomy and physiology (15%), Animal anatomy and physiology (25%), Ethology (5%), Genetics and Evolution (20%), Ecology (10%), Biosystematics (5%). Each of these topic areas is described in more detail (8 pages in total). The section on *Basic Skills for the Practical Part of the IBO* covers such things as science process skills, basic biological skills, biological methods, physical and chemical methods, microbiological methods, statistical methods, and handling of equipment.

3 Purpose and Motivation for an IOI Syllabus

One of the main objectives of the IOI is "to bring the discipline of informatics to the attention of young people" (Statute S1.7 from the IOI Regulations). The olympiads in mathematics, physics, chemistry, biology, and geography are in the fortunate position that these sciences are regular exam topics in secondary education in most countries. Informatics, however, is not in this position. And even if it is, the topics of algorithmics and programming often receive minimal attention.

Algorithmics and programming were chosen as the main topics for the IOI competition; these were the only areas for which a sufficient number of self-taught contestants could be found. The wide availability of personal computers with easily accessible programming tools has helped create this situation. These areas are, in addition, fundamental to computing science.

Over the years, the difficulty level of IOI competition tasks has increased considerably. Among the best contestants, performance has certainly improved to warrant some increase in difficulty. But this improvement is not so much manifested by the 'average' contestant. In part, this disparity may be attributed to the lack of systematic education in computing science.

An IOI Syllabus would benefit:

- candidate IOI contestants,
- coaches of IOI contestants,
- teachers of computing science in secondary education,
- developers of curricula in computing science for secondary education,
- authors of computing science textbooks for secondary education,
- creators of competition tasks for the IOI and similar competitions,
- organizers of computing competitions,
- interested outsiders.

We must also point out possible dangers in having an official IOI Syllabus. Such a syllabus could easily obtain the status of a dogmatic standard. Any attempt to step outside the scope of the syllabus could be blindly suppressed, thereby stifling innovation. The process of deciding about the appropriateness of candidate competition tasks could be paralyzed by time-consuming discussions about the syllabus. Coaches and contestants could be misled to believe that knowing what is in the syllabus will guarantee some success in the competition.

In view of these dangers, the syllabus must contain information concerning its proper use, must be flexible, and must include a mechanism for its ongoing revision.

Finally, there are a number of challenges in composing an IOI Syllabus, such as obtaining sufficient consensus on form and content, ensuring clarity and precision (a single interpretation), and ensuring sufficient completeness. We hope that this article will help address these challenges.

4 Roles of Mathematics in Computing Science

It is important to understand the various roles that mathematics plays in computing science. These roles are often confused, especially in secondary education. One can distinguish the following roles of mathematics in computing science:

1. *As a problem domain.* For example, design an algorithm to compute the greatest common divisor of positive integers A and B.

2. *As a language to express formalized models*, both in the problem analysis and the solution domain. For example, a street network can be modeled as a directed graph.

3. *As a language to reason about models*. For example, if a graph has no cycles, then one can draw the conclusion that it has fewer edges than vertices.

4. *As a language to reason about computations, algorithms and data structures, and their implementation*; in particular, to reason about functional correctness, termination, and efficiency. For example, the binary search algorithm applied to an array of N elements, terminates in $\mathcal{O}(\log N)$ steps.

Note that the second and third roles (concerning the formulation of and reasoning about models) are present in most branches of science. The type of mathematical models depends to some extent on the science in question. For instance, group theory (to study symmetry), differential equations, numerical analysis, probability theory and statistics are more relevant in the natural sciences. Logic and discrete models involving combinatorial structures are more relevant in computing science.

The first and last role (as problem domain and for reasoning about computations) are more specific to computing science. Mathematical knowledge concerning the first role (as problem domain) could be avoided by allowing only competition tasks that involve non-mathematical problem domains. However, such a restriction would be unrealistic:

- There are few non-mathematical domains that all IOI contestants can be expected to know well enough. Such knowledge is often difficult to present succinctly and clearly as part of the problem statement in an attempt to compensate for deficiencies.

- Mathematics as a problem domain has the advantage that it allows very compact and precise problem statements. Furthermore, a rich problem domain is available through elementary mathematics, which should be well within reach of students from secondary education.

- Mathematics in the second role (to express models) is indispensable for the development of algorithmic solutions, even when dealing with algorithmic problems from a non-mathematical domain. There is a significant overlap between mathematics in the first and second role.

Mathematical knowledge in the first and second role mostly concerns concepts, terminology, and notations. Verhoeff [17] presents a classification of elementary concepts, terminology, and notations with respect to their usability in IOI competition tasks. This classification could be used in an IOI Syllabus. It should, however, be augmented with some relevant methods and techniques to combine and apply these elementary notions.

Mathematical knowledge in both the third and fourth roles both concerns *reasoning*, viz. about models and about computations. This skill demands some familiarity with mathematical logic. Contestants need the ability

- to express conjectures and theorems (even if only informally), and
- to construct and understand logical deductions, applying theorems.

They are also expected to know certain (elementary) mathematical theorems.

Note that the mathematics in the fourth role (reasoning about computations) is most specific to computing science, and is often not encountered in other branches of science. Unfortunately, it tends to be underexposed in secondary education.

Any IOI Syllabus must clarify what mathematical knowledge is important in what roles. However, it should always be kept in mind that the IOI is an informatics competition and not a math contest.

5 General Principles behind the Syllabus

Our first principle is that we intend to capture the current accepted IOI practice in the syllabus. There may be reasons for changing the IOI and they may become even clearer by writing a syllabus, but it is not our goal to reform the IOI through the syllabus.

Because it does not seem to be documented anywhere, we feel that a brief explanation is in order as to why the IOI competition focuses on algorithmic problem solving and programming. This narrow focus is similar to the IMO, and contrasts sharply with the IBO, which strives for comprehensive coverage of the discipline.

The IOI competition is not meant to be an ordinary exam that tests whether the participants have learned their lessons. It is aimed at discovering and challenging talented pupils. For this, it was decided that depth rather than breadth, and innovative problems rather than standard exercises are of primary importance. To minimize the advantage of having special prior knowledge, the topics should be elementary and fundamental.

The reasons for requiring programmed implementations of algorithms are[1]:

- There is no standardized abstract algorithm notation suitable for use in the IOI. Programming languages have a well-defined syntax and semantics[2], and can stand in to express algorithms.

[1] Also see §7.2 of [1]: “Where does programming fit in the introductory curriculum”

[2] We refer to a semantics in terms of an abstract machine, rather than via a specific compiler.

- Pupils interested in computing typically know a popular programming language. It is easy to learn the basics of programming.
- Implementing an algorithm as a computer program requires you to fill in all the gaps. You cannot afford to be vague anywhere.
- Programs can be executed and thereby facilitate automated evaluation to some extent.
- Producing a working program is a satisfactory experience. This is common to all engineering disciplines.
- Programming is one of the fundamental topics of computing science.

Nevertheless, the overwhelming details of modern computing platforms (programming languages, programming tools, operating systems, system and processor architecture) should not become an obstacle to success.

Currently, imperative programming languages are used in the IOI, but the authors wish to point out that functional languages would also be suitable for this purpose.

5.1 The Broader Context of Algorithms

It is tempting simply to list various algorithms and data structures, and underlying design techniques that are relevant to the IOI competition. Rather than collecting a large number of topics for an IOI Syllabus, we find it more valuable to state some general principles to guide the inclusion and exclusion of specific topics.

The decision about the relevance of a particular algorithm should be based not only on how complicated the algorithm is when written in pseudocode. It is important to distinguish

- how complicated an algorithm is (in a static sense) and
- how advanced the reasoning behind the algorithm's design is, concerning correctness and/or efficiency[3].

Here are some typical examples:

- The pseudocode for the *Knuth-Morris-Pratt string search algorithm* is not complicated, but the reasoning behind its correctness is.

[3]In current IOI practice, algorithms are only assessed for functional correctness and time/space efficiency, and not for other qualities.

- The *linear median selection algorithm* by Blum et al. is not complicated, but the proof of its worst-case linear time efficiency is.
- Hoare's *Quicksort* is not a complicated algorithm, but the analysis of its average-case running time involves advanced mathematics.

Whether or not a specific algorithm is to be considered prerequisite IOI knowledge should also be based on the (mathematical) techniques relevant for the underlying reasoning.

It is also important to distinguish

- how seemingly straightforward an algorithm is and
- how demanding it can be to implement it as an actual correct and efficient computer program.

Again, an example may help:

- *Kruskal's algorithm* for computing a minimum spanning tree is simple when expressed in terms of an abstract data type that maintains a partition of the nodes, but an efficient implementation of this data type is much more involved.
- *Algorithms for determining whether two line segments intersect* involve a careful case distinction, which can easily lead to hard-to-spot implementation errors.

It seems inappropriate to ask contestants to develop algorithms that require advanced reasoning techniques to understand their correctness and/or efficiency, even if the algorithms themselves are not complicated.

It may also not be a good idea to ask contestants to develop algorithms whose implementation is intrinsically troublesome, even if the abstract algorithm is not so complicated. Keep in mind that the IOI is not just an implementation contest.

Thus, the required reasoning techniques and implementation techniques play a fundamental role in selecting topics for the IOI Syllabus.

5.2 Main Topic Areas for an IOI Syllabus

To organize the relevant topics we have consulted the ACM curriculum models for college [1] and K–12 [2]. The latter refers to, and is in part derived from, the former, but the K–12 curriculum is too restrictive for our purposes. We have chosen to use the Computing Curricula (CC2001) topics of [1] as a basis for our syllabus proposal. Because CC2001 is aimed at university-level education, it offers the required depth. It also contains many topics that will not occur in the IOI competition. However, we think that it is good to be able to point out explicitly what further topics exist in computing and are not relevant for the IOI.

Taking a top-down view, we arrive at the following main areas:

- Mathematics, in particular, Discrete Structures (DS), but with small additions from number theory and geometry;
- Computing Science, in particular, Programming Fundamentals (PF), and Algorithms and Complexity (AL);
- Software Engineering, in particular, its application "in the small";
- Computer Literacy, in particular, use of a computer for program development and other competition-related purposes (e.g. submitting files via a web browser, printing).

Some basic science and engineering skills and methods will be included under Computing Science and under Software Engineering respectively.

6 Proposed IOI Syllabus

The proposed IOI Syllabus focuses on topics in the form of concepts. For further details on terminology and notations, we refer to [17]. An updated version of that document could be incorporated into a final IOI Syllabus.

The proposed IOI Syllabus classifies each topic into one of three categories:

Included This is the default category. It means that the topic is relevant for the IOI competition, that is, it could play a role in the description of a competition task, in the contestant's process of solving the task, or in the model solution. Included topics are further qualified as:

♡ **Unlimited** It concerns prerequisite knowledge, and can appear in task descriptions without further clarification[4].

Example: *Integer* in §6.1.1

△ **To be clarified** Contestants should know this topic, but when it appears in a task description, the author must always clarify it sufficiently.

Example: *Directed graph* in §6.1.2 DS2

[4]Danger of confusion (e.g. Fibonacci numbers) must always be avoided by further clarification.

⊖ **Not for task description** It will not appear in tasks descriptions, but may be needed for developing solutions or understanding model solutions.

Example: *Asymptotic analysis of upper complexity bounds*, §6.2.2 AL1

Not needed This means that although the topic may be of interest, it will not appear in task descriptions or model solutions, and that it will not be needed to arrive at a solution. However, see also the note below about possible promotion to *Included*.

Example: *Binomial theorem* in §6.1.2 DS4

Excluded This means that the topic falls outside the scope of the IOI competition.

Example: *Calculus* in 6.1.3

The classifications under *Not needed* and *Excluded* are not intended to be exhaustive, but rather serve as examples that map out the boundary. Topics not mentioned in the syllabus are to be treated as *Excluded*. However, topics not classified for use in task descriptions, including topics not mentioned, could be promoted to *Included* △ , provided that they require no special knowledge and will be defined in terms of included non-⊖ concepts, in a precise, concise, and clear way. Special cases of non-included topics can be good candidates for such promotion. For example, planar graphs are excluded, but trees (a special case) are in fact included.

Note that the syllabus must not be interpreted to restrict in any way the techniques that contestants are allowed to apply in solving the competition tasks. Of course, each task or the Competition Rules can impose binding restrictions, which are to be considered as part of the problem statement (e.g. that no threads or auxiliary files are to be used).

Topics literally copied from [1] are typeset in sans serif font.

6.1 Mathematics

6.1.1 Numbers and Geometry

♡ Integers, operations (incl. exponentiation), comparison
♡ Properties of integers (positive, negative, even, odd, divisible, prime)
♡ Fractions, percentages

♡ Point, vector, Cartesian coordinates (on a 2D integer grid)
△ Euclidean distance, Pythagoras' Theorem
♡ Line segment, intersection properties
△ Angle
♡ Triangle, rectangle, square
♡ Polygon (vertex, side/edge, simple, convex, inside/outside)

Not needed: Circle

Excluded: Real numbers, trigonometric functions

6.1.2 Discrete Structures (DS)

DS1. Functions, relations, and sets :

△ Functions (surjections, injections, inverses, composition)
△ Relations (reflexivity, symmetry, transitivity, equivalence relations, total/linear order relations, lexicographic order)
♡ Sets (Venn diagrams, complements, Cartesian products, power sets)
△ Pigeonhole principle

Excluded: Cardinality and countability (of infinite sets)

DS2. Basic logic :

♡ Propositional logic
♡ Logical connectives (incl. their basic properties)
♡ Truth tables
♡ Predicate logic
♡ Universal and existential quantification
⊖ Modus ponens and modus tollens

N.B. This article is not concerned with notation. In past task descriptions, logic has been expressed in natural language rather than mathematical symbols, such as $\wedge$, $\vee$, $\forall$, $\exists$.

Not needed: Validity

Excluded: Normal forms, Limitations of predicate logic

DS3. Proof techniques :

△ Notions of implication, converse, inverse, contrapositive, negation, and contradiction
⊖ Direct proofs, proofs by: counterexample, contraposition, contradiction
⊖ Mathematical induction
⊖ Strong induction (also known as complete induction)
♡ Recursive mathematical definitions (incl. mutually recursive definitions)

Not needed: The structure of formal proofs

Excluded: Well orderings

DS4. Basics of counting :

♡ Counting arguments (sums and product rule, inclusion-exclusion principle, arithmetic and geometric progressions, Fibonacci numbers)
△ Pigeonhole principle (to obtain bounds)
△ Permutations and combinations (basic definitions)
△ Factorial function, binomial coefficient

Not needed: Pascal's identity, Binomial theorem

Excluded: Solving of recurrence relations

DS5. Graphs and trees :

△ Trees (connected, no cycles, #nodes = #edges + 1; ordered/not-ordered)
△ Undirected graphs (degree, path, cycle, connectedness, Euler/Hamilton path/cycle, handshaking lemma)
△ Directed graphs (in-degree, out-degree, directed path/cycle, Euler/Hamilton path/cycle)
△ Spanning trees
△ Traversal strategies (defining the node order for ordered trees)
△ 'Decorated' graphs with edge/node labels, weights, colors
△ Multigraphs, graphs with self-loops

Not needed: Planar graphs, Bipartite graphs, Hypergraphs

DS6. Discrete probability : *Excluded*

6.1.3 Other Areas in Mathematics

Not needed: Polynomials, Matrices and operations, Solid geometry

Excluded: (Linear) Algebra, Calculus, Probability Theory, Statistics

6.2 Computing Science

6.2.1 Programming Fundamentals (PF)

PF1. Fundamental programming constructs : (for abstract machines)

♡ Basic syntax and semantics of a higher-level language (also see the following topics; the specific languages available at an IOI will announced in the *Competition Rules* for that IOI)
♡ Variables, types, expressions, and assignment
♡ Simple I/O
♡ Conditional and iterative control structures
♡ Functions and parameter passing
⊖ Structured decomposition

PF2. Algorithms and problem-solving :

⊖ Problem-solving strategies (understand–plan–do–check, separation of concerns, generalization, specialization, case distinction, working backwards; see e.g. [16])
⊖ The role of algorithms in the problem-solving process
⊖ Implementation strategies for algorithms (also see §6.3 SE1)
⊖ Debugging strategies (also see §6.3 SE3)
△ The concept and properties of algorithms (correctness, efficiency)

PF3. Fundamental data structures :

♡ Primitive types (Boolean, integer, character)
♡ Arrays (incl. multidimensional arrays)
♡ Records
♡ Strings and string processing
△ Static and stack allocation (elementary automatic memory management)
△ Linked structures (linear and branching)
△ Static memory implementation strategies for linked structures
△ Implementation strategies for stacks and queues
△ Implementation strategies for graphs and trees
△ Strategies for choosing the right data structure
△ Abstract data types, priority queue, dynamic set, dynamic map

Not needed: Data representation in memory, Heap allocation, Runtime storage management, Pointers and references[5], Implementation strategies for hash tables, Arbitrary-size integer arithmetic[6]

Excluded: Floating-point numbers (see [5])

PF4. Recursion :

♡ The concept of recursion
♡ Recursive mathematical functions
♡ Simple recursive procedures (incl. mutual recursion)
⊖ Divide-and-conquer strategies
⊖ Recursive backtracking

Not needed: Implementation of recursion

PF5. Event-driven programming : *Not needed*

[5]The inessential advantage of scalable memory efficiency is outweighed by the increased complexity in reasoning. Static memory implementations should suffice to solve IOI tasks.

[6]This means that tasks are solvable with the appropriate primitive integer types, without concern for overflow.

However, competition tasks could involve a dialog with a reactive environment.

6.2.2 Algorithms and Complexity (AL)

We quote from [1]:

> Algorithms are fundamental to computer science and software engineering. The real-world performance of any software system depends only on two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. Good algorithm design is therefore crucial for the performance of all software systems. Moreover, the study of algorithms provides insight into the intrinsic nature of the problem as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or any other implementation aspect.

AL1. Basic algorithmic analysis :

△ Algorithm specification, precondition, postcondition, correctness, invariant
⊖ Asymptotic analysis of upper complexity bounds (i.e. worst-case; informally)
⊖ Big O notation
⊖ Standard complexity classes (constant, logarithmic, linear, $\mathcal{O}(N \log N)$, quadratic, cubic, exponential)
⊖ Time and space tradeoffs in algorithms

Not needed: Identifying differences among best, average, and worst case behaviors, Little o, omega, and theta notation, Empirical measurements of performance

Excluded: Asymptotic analysis of average complexity bounds, Using recurrence relations to analyze recursive algorithms

AL2. Algorithmic strategies :

⊖ Simple loop design strategies
⊖ Brute-force algorithms (exhaustive search)
⊖ Greedy algorithms (insofar that understanding correctness is elementary)
⊖ Divide-and-conquer (insofar that understanding efficiency is elementary)
⊖ Backtracking (recursive and non-recursive)
⊖ Branch-and-bound (insofar that understanding correctness and efficiency are elementary)
⊖ Pattern matching and string/text algorithms (insofar that understanding

correctness and efficiency are elementary)
⊖ Dynamic programming[7]

Excluded: Heuristics, Numerical approximation algorithms

AL3. Fundamental computing algorithms :

⊖ Simple numerical algorithms involving integers (Euclid's algorithm, primality test by $\mathcal{O}(\sqrt{N})$ trial division, Sieve of Eratosthenes, efficient exponentiation)
⊖ Simple iterative algorithms (min/max selection, histogram, bucket sort)
⊖ Sequential and binary search algorithms
⊖ Search by elimination, "slope" search
⊖ Quadratic sorting algorithms (selection, insertion)
⊖ Partitioning, order statistics by repeated partitioning, Quicksort
⊖ $\mathcal{O}(N \log N)$ worst-case sorting algorithms (heap sort, merge sort)
⊖ Binary search trees
⊖ Representations of graphs (adjacency list, adjacency matrix)
⊖ Traversals of ordered trees
⊖ Depth- and breadth-first traversals of graphs
⊖ Shortest-path algorithms (Dijkstra's and Floyd's algorithms)
⊖ Transitive closure (Floyd's algorithm)
⊖ Minimum spanning tree (Prim's and Kruskal's[8] algorithms)
⊖ Topological sort
⊖ Algorithms to determine connected components of an undirected graph
⊖ Algorithms to determine (existence of) an Euler path/cycle

Not needed: Hash tables (including collision-avoidance strategies)

Excluded: Simple numerical algorithms involving floating-point arithmetic, Max flow algorithms, Bipartite matching algorithms

AL4. Distributed algorithms : *Excluded*

AL5. Basic computability :

Not needed: Finite-state machines, Context-free grammars (could be considered in the future)

Excluded: Tractable and intractable problems, Uncomputable functions, The halting problem, Implications of uncomputability

AL6. The complexity classes P and NP : *Excluded*

[7] [1] puts this under AL8, but we believe it belongs here.
[8] In terms of a disjoint-set ADT

AL7. Automata theory : *Excluded*

However, Finite automata, Regular expressions, Turing machines could be considered in the future.

AL8. Advanced algorithmic analysis :

⊖ Minimax algorithms for optimal game playing

Not needed: Online and offline algorithms, Combinatorial optimization

Excluded: Amortized analysis, Randomized algorithms, Alpha-beta pruning

AL9. Cryptographic algorithms : *Excluded*

AL10. Geometric algorithms : (on 2D grids, i.e. integer (x, y)-coordinates)

⊖ Line segments: properties, intersections
⊖ Point location w.r.t. simple polygon
⊖ Convex hull finding algorithms

AL11. Parallel algorithms : *Excluded*

6.2.3 Other Areas in Computing Science

The following areas are all excluded.

AR. Architecture and Organization : *Excluded*

This area is about digital systems, assembly language, instruction pipelining, cache memories, etc.The basic structure of a computer is covered in §6.4.

OS. Operating Systems : *Excluded*

This area is about the *design* of operating systems, covering concurrency, scheduling, memory management, security, file systems, real-time and embedded systems, fault tolerance, etc. The basics of *using* the high-level services of an operating system are covered in §6.4, but low-level system calls are specifically excluded.

NC. Net-Centric Computing : *Excluded*

PL. Programming Languages : *Excluded*

This area is about *analysis and design* of programming languages, covering classification, virtual machines, translation, object-orientation, functional programming, type systems, semantics, and language design. The basics of *using* a high-level programming language are in §6.2.1.

HC. Human-Computer Interaction : *Excluded*

This area is about the *design* of user interfaces, etc. The basics of *using* a (graphical) user interface are covered in §6.4.

GV. Graphics and Visual Computing : *Excluded*

IS. Intelligent Systems : *Excluded*

IM. Information Management : *Excluded*

SP. Social and Professional Issues : *Excluded*

CN. Computational Science : *Excluded*

6.3 Software Engineering (SE)

We quote from [1]:

> Software engineering is the discipline concerned with the application of theory, knowledge, and practice for effectively and efficiently building software systems that satisfy the requirements of users and customers.

In the IOI competition, the application of software engineering concerns the use of light-weight techniques for small, one-off, single-developer projects under time pressure. All included topics are $\ominus$.

SE1. Software design :

Fundamental design concepts and principles
Design patterns
Structured design

In particular, contestants may be expected to

Transform an abstract algorithm into a concrete, efficient program expressed in one of the allowed programming languages, possibly using standard or competition-specific libraries.
Make their programs read data from and write data to text files according to a prescribed simple format[9]

Not needed: Software architecture, Design for reuse

Excluded: Object-Oriented analysis and design, Component-level design

[9]Evaluation of submitted programs will only be based on input data that agrees with the prescribed input format. Submitted programs need not check input validity. However, when contestants offer input data of their own design, then obviously no such guarantees can be made.

SE2. Using APIs :

API[10] programming

In particular, contestants may be expected to

> Use competition-specific libraries according to the provided specification.

Not needed: Programming by example, Debugging in the API environment

Excluded: Class browsers and related tools, Introduction to component-based computing

SE3. Software tools and environments :

Programming environments (incl. IDE[11])

In particular, contestants may be expected to

> Write and edit program texts using one of the provided program editors.
> Compile and execute their own programs.
> Debug their own programs.

Not needed: Testing tools, Configuration management tools

Excluded: Requirements analysis and design modeling tools, Tool integration mechanisms

SE4. Software processes :

Software life-cycle and process models

In particular, contestants may be expected to

> Understand the various phases in the solution development process and select appropriate approaches.

Excluded: Process assessment models, Software process metrics

SE5. Software requirements and specification :

Functional and nonfunctional requirements
Basic concepts of formal specification techniques

In particular, contestants may be expected to

[10] Application Programming Interface
[11] Integrated Development Environment

> Transform a precise natural-language description (with or without mathematical formalism) into a problem in terms of a computational model, including an understanding of the efficiency requirements.

Not needed: Prototyping

Excluded: Requirements elicitation, Requirements analysis modeling techniques

SE6. Software validation :

Testing fundamentals, including test plan creation and test case generation
Black-box and white-box testing techniques
Unit, integration, validation, and system testing
Inspections

In particular, contestants may be expected to

> Apply techniques that maximize the the opportunity to detect common errors (e.g. through well-structured code, code review, built-in tests, test execution).
>
> Test (parts of) their own programs.

Not needed: Validation planning

Excluded: Object-oriented testing

SE7. Software evolution :

Not needed: Software maintenance, Characteristics of maintainable software, Re-engineering, Legacy systems, Software reuse

SE8. Software project management :

Project scheduling (especially time management)
Risk analysis
Software configuration management

In particular, contestants may be expected to

> Manage time spent on various activities.
>
> Weigh risks when choosing between alternative approaches.
>
> Keep track of various versions and their status while developing solutions.

Not needed: Software quality assurance

Excluded: Team management, Software measurement and estimation techniques, Project management tools

SE9. Component-based computing : *Excluded*

SE10. Formal methods :

Formal methods concepts (notion of correctness proof, invariant)
Pre and post assertions

In particular, contestants may be expected to

> Reason about the correctness and efficiency of algorithms and programs.

Not needed: Formal verification

Excluded: Formal specification languages, Executable and non-executable specifications

SE11. Software reliability : *Excluded*

SE12. Specialized systems development : *Excluded*

6.4 Computer Literacy ⊖

Contestants should know and understand the basic structure and operation of a computer (CPU, memory, I/O). They are expected to be able to use a standard computer with graphical user interface, its operating system with supporting applications, and the provided program development tools for the purpose of solving the competition tasks. In particular, some skill in file management is helpful (creating folders, copying and moving files).

Details of these facilities will be stated in the *Competition Rules* of the particular IOI. Typically, some services are available through a standard web browser. Possibly, some competition-specific tools are made available, with separate documentation.

It is often the case that a number of equivalent tools are made available. The contestants are not expected to know all the features of all these tools. They can make their own choice based on what they find most appropriate.

Not needed: Calculator

Excluded: Word-processors, Spreadsheet applications, Data base management systems, E-mail clients, Graphics tools (drawing, painting)

7 Conclusion

Contestants are well served when given clear, correct, and timely guidelines as to what they may expect in a competition. They are poorly served when unanticipated expectations are placed on them. While no syllabus should be construed to supplant reason or discretion, we advance the current proposal as a framework through which to make the expectations on IOI contestants more clear.

The proposed syllabus provides a mechanism for contestants, educators, and contest designers to achieve a better common understanding of the skills and knowledge assumed of contestants. Space does not permit a detailed rationale for every choice; the authors have relied on their experience with international competitions, an examination of past tasks, and an analysis of benefits and drawbacks arising from the inclusion or exclusion of certain concepts and techniques. No doubt some of these choices are controversial — we hope that the structure we have presented will help to guide and focus the ensuing debate.

It is desirable to cite text books and other training materials that contestants and educators might use in preparation for the IOI. None of which we are aware fits the syllabus perfectly; current materials would have to be used selectively so as to be consistent with the syllabus. A book covering a substantial fraction of the syllabus topics is [3].

Our immediate aim is that the IOI adopt a syllabus based on our proposal. In the medium term, we seek to identify specific resource materials supporting each of the included topics. In the long term, we believe it is appropriate to develop and distribute IOI-specific educational and training materials.

The authors wish to acknowledge the inspiration of the IOI Workshop [15].

References

[1] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 2001: Computer Science Volume*. December 2001.
`http://www.acm.org/sigcse/cc2001/`

[2] ACM K–12 Task Force Curriculum Committee. *A Model Curriculum for K–12 Computer Science: Final Report*. October 2003.
`http://www1.acm.org/education/k12/k12final1022.pdf`

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*, 2nd Ed. McGraw-Hill, 2001.

[4] G. Horváth and T. Verhoeff. “Finding the Median under IOI Conditions”, *Informatics in Education*, **1**(1):73–92 (2002).

[5] G. Horváth and T. Verhoeff. "Numerical Difficulties in Pre-University Education and Competitions", *Informatics in Education*, **2**(1):21–38 (2003).

[6] IAO, *International Astronomy Olympiad*, Internet WWW-site. `http://www.issp.ac.ru/iao/` (accessed February 2006).

[7] IBO, *International Biology Olympiad*, Internet WWW-site. `http://www.ibo-info.org/` (accessed February 2006).

[8] IChO, *International Chemistry Olympiad*, Internet WWW-site. `http://www.icho.sk/` (accessed February 2006).

[9] IGeO, *International Geography Olympiad*, Internet WWW-site. `http://www.geoolympiad.org/` (accessed February 2006).

[10] ILO, *International Linguistic Olympiad.* Wikipedia: `http://en.wikipedia.org/wiki/International_Linguistic_Olympiad` (accessed February 2006).

[11] IMO, *International Mathematical Olympiad.* Wikipedia: `http://en.wikipedia.org/wiki/International_Mathematical_Olympiad` (accessed February 2006).

[12] IOI, *International Olympiad in Informatics*, Internet WWW-site. `http://www.IOInformatics.org/` (accessed February 2006).

[13] IPhO, *International Physics Olympiad*, Internet WWW-site. `http://www.jyu.fi/tdk/kastdk/olympiads/` (accessed February 2006).

[14] P. S. Kenderov and M. N. Maneva (Eds.). *Proceedings of the International Olympiad in Informatics*, Pravetz, Bulgaria, May 16–19, 1989. Sofia: Union of the Mathematicians in Bulgaria, 1989.

[15] W. Pohl. "Foreword", *Informatics in Education*, **5**(1):3–4 (2006).

[16] G. Polya. *How to Solve It: A New Aspect of Mathematical Method.* Princeton Univ. Press, 1948.

[17] T. Verhoeff. *Concepts, Terminology, and Notations for IOI Competition Tasks*, document presented at IOI 2004 in Athens, 12 Sep. 2004. `http://scienceolympiads.org/ioi/sc/documents/terminology.pdf`