

The International Olympiad in Informatics Syllabus

August 19, 2008

1 Authors and Contact Information

The original proposal of the IOI Syllabus was co-authored by:

- Tom Verhoeff
(TU Eindhoven, The Netherlands, t.verhoeff at tue.nl)
- Gyula Horváth
(University of Szeged, Hungary, horvath at inf.u-szeged.hu)
- Krzysztof Diks
(Warsaw University, Poland, diks at mimuw.edu.pl)
- Gordon Cormack
(University of Waterloo, Canada, gvcormac at uwaterloo.ca)

The current maintainer of the Syllabus is

- Michal Forišek
(Comenius University, Slovakia, forisek at dcs.fmph.uniba.sk)

You are welcome to send any feedback on the Syllabus to the current maintainer's e-mail address.

2 Introduction

This Syllabus has two main purposes.

The first purpose is to provide a set of guidelines that help decide whether a task is suitable for the International Olympiad in Informatics (IOI). Based on this document, the International Scientific Committee (ISC) evaluates the task proposals when selecting the competition tasks.

The second and closely related purpose is to help the organizers of national olympiads prepare their students for the IOI.

The goal of this Syllabus is to give a classification of topics and concepts from mathematics and computer science. The Syllabus specifies which of these concepts are suitable to appear in the competition tasks and/or their solutions.

Issues related to the usage of suitable terminology and notations in competition tasks are beyond the scope of this document. (These issues are discussed in [6].)

This 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** The topic concerns prerequisite knowledge, and can appear in task descriptions without further clarification¹.

Example: *Integer* in §3.1

△ **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 §3.2 DS2

⊖ **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* in §4.2 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 §3.2 DS4

¹Danger of confusion (e.g. Fibonacci numbers) must always be avoided by further clarification.

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

Example: *Calculus* in §3.3

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* \triangle , provided that they require no special knowledge and will be defined in terms of included non- \ominus 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.

3 Mathematics

3.1 Arithmetics 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)
- \triangle Euclidean distance, Pythagoras' Theorem
- ♡ Line segment, intersection properties
- \triangle Angle
- ♡ Triangle, rectangle, square, circle
- ♡ Polygon (vertex, side/edge, simple, convex, inside/outside, area)

Excluded: Real and complex numbers, general conics (parabolas, hyperbolas, ellipses), trigonometric functions

3.2 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 quantification²
- ⊖ 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, Normal forms

Excluded: 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

²In case the definition of a term in the problem statement contains more than one quantifier (and especially if the quantifiers are of both types), it is advised to split the definition into parts, each containing a single quantifier. E.g., don't define the diameter of a graph directly, use eccentricity of a vertex as an intermediate step.

- ♡ 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
- ⊖ Pascal's identity, Binomial theorem

Excluded: Solving of recurrence relations

DS5. Graphs and trees

- △ Trees (and their basic properties)
- △ 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*

3.3 Other Areas in Mathematics

Not needed: Polynomials, Matrices and operations, Geometry in 3D space

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

4 Computing Science

4.1 Programming Fundamentals (PF)

PF1. Fundamental programming constructs (for abstract machines)

- ♡ Basic syntax and semantics of a higher-level language (the specific languages available at an IOI will be 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. [5])
- ⊖ The role of algorithms in the problem-solving process
- ⊖ Implementation strategies for algorithms (also see §5 SE1)
- ⊖ Debugging strategies (also see §5 SE3)
- △ The concept and properties of algorithms (correctness, efficiency)

PF3. Fundamental data structures

- ♡ Primitive types (boolean, signed/unsigned 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 references³

Excluded: Floating-point numbers (see [3]), Implementation strategies for hash tables

PF4. Recursion

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

³The inessential advantage of scalable memory efficiency is outweighed by the increased complexity in reasoning. Static memory implementations should suffice to solve IOI tasks.

- ⊖ Recursive backtracking

Not needed: Implementation of recursion

PF5. Event-driven programming

Not needed

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

4.2 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, invariants
- ⊖ Asymptotic analysis of upper complexity bounds (informally if possible)
- ⊖ 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 is elementary)
- ⊖ Dynamic programming⁴
- ⊖ Discrete approximation algorithms⁵

Excluded: Heuristics, Numerical approximation algorithms

AL3. Fundamental computing algorithms

- ⊖ Simple numerical algorithms involving integers (radix conversion, Euclid’s algorithm, primality test by $\mathcal{O}(\sqrt{N})$ trial division, Sieve of Eratosthenes, factorization, efficient exponentiation)
- ⊖ Simple operations on arbitrary precision integers (addition, subtraction, simple multiplication)⁶
- ⊖ Simple array manipulation (filling, shifting, rotating, reversal, resizing, minimum/maximum, prefix sums, histogram, bucket sort)
- ⊖ Sequential processing, 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 heap data structure⁷
- ⊖ Binary search trees
- ⊖ Fenwick trees⁸

⁴[1] puts this under AL8, but we believe it belongs here.

⁵The purpose of this item is to include tasks such as Xor from IOI 2002, where the contestants have to devise an approximation algorithm, and achieve scores based on its performance relative to other submissions. Textbook approximation algorithms are *Excluded*.

⁶The necessity to implement these operations should be obvious from the problem statement.

⁷The more complex heap data structures, such as binomial and Fibonacci heaps, are *Excluded*.

⁸Introduced in [2], also known as binary indexed trees. A 2D version of a Fenwick tree was used in the IOI 2001 task Mobiles. This data structure is sometimes known as a segment/interval tree, but this name shall be avoided to prevent confusion with data structures that actually store segments or intervals.

- ⊖ Representations of graphs (adjacency list, adjacency matrix)
- ⊖ Traversals of ordered trees
- ⊖ Depth- and breadth-first traversals of graphs, determining connected components of an undirected graph
- ⊖ Shortest-path algorithms (Dijkstra, Bellman-Ford, Floyd-Warshall)
- ⊖ Transitive closure (Floyd's algorithm)
- ⊖ Minimum spanning tree (Jarník-Prim and Kruskal⁹ algorithms)
- ⊖ Topological sort
- ⊖ 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, Maximum flow algorithms, Bipartite matching algorithms, Strongly connected components in directed graphs

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

AL7. Automata and grammars

Excluded

However, Finite automata, Regular expressions, and rewriting systems could be considered in the future.

AL8. Advanced algorithmic analysis

- ⊖ Basics of Combinatorial game theory, Minimax algorithms for optimal game playing

⁹In terms of a disjoint-set ADT

Not needed: Online and offline algorithms, Combinatorial optimization

Excluded: Amortized analysis, Randomized algorithms, Alpha-beta pruning, Sprague-Grundy theory

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
- ⊖ Sweeping line method

AL11. Parallel algorithms

Excluded

4.3 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.

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, 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 §4.1.

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.

GV. Graphics and Visual Computing

Excluded

IS. Intelligent Systems

Excluded

IM. Information Management

Excluded

SP. Social and Professional Issues

Excluded

CN. Computational Science

Excluded

5 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 format¹⁰

Not needed: Software architecture, Design for reuse

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

SE2. Using APIs

- ⊖ API (Application Programming Interface) 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 (Integrated Development Environment)

In particular, contestants may be expected to

¹⁰Evaluation 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.

- 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

- 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 Computer Literacy

The text of this section is \ominus .

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)

References

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

- [2] P. Fenwick. “A New Data Structure for Cumulative Frequency Tables”, *Software – Practice And Experience*, **24**(3):327–336 (1994).
- [3] G. Horváth and T. Verhoeff. “Numerical Difficulties in Pre-University Education and Competitions”, *Informatics in Education*, **2**(1):21–38 (2003).
- [4] IOI, *International Olympiad in Informatics*, Internet WWW-site. <http://www.IOInformatics.org/> (accessed February 2006).
- [5] G. Polya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton Univ. Press, 1948.
- [6] 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>