The International Olympiad in Informatics Syllabus

1 Version and status information

This is the official Syllabus version for **IOI 2017** in Iran.

All additions since the 2016 version were presented at IOI 2016. All changes are **in red**.

The Syllabus is an official document related to the IOI. For each IOI, an up-to-date version of the Syllabus is produced by the ISC, as described in the IOI Regulations, Statue 3.13.

2 Authors and Contact Information

The original proposal of the IOI Syllabus was co-authored by Tom Verhoeff[1], Gyula Horváth[2], Krzysztof Diks[3], and Gordon Cormack[4].

Since 2007, the following people have maintained the syllabus and made significant contributions: Michal Forišek[5], Jakub Łącki[6], and Richard Peng[7].

The most recent batch of revisions to the Syllabus was made by the ISC between February and July 2016.

You are welcome to send any feedback on the Syllabus to the current maintainer's e-mail address (`forisek@dcs.fmph.uniba.sk`).

For people interested in contributing to the quality of the Syllabus, some additional background on the Syllabus and other miscellaneous information can be found at `http://ksp.sk/~misof/ioi-syllabus/`.

[1]TU Eindhoven, The Netherlands, `t.verhoeff@tue.nl`
[2]University of Szeged, Hungary, `horvath@inf.u-szeged.hu`
[3]Warsaw University, Poland, `diks@mimuw.edu.pl`
[4]University of Waterloo, Canada, `gvcormac@uwaterloo.ca`
[5]Comenius University, Slovakia, `forisek@dcs.fmph.uniba.sk`
[6]Warsaw University, Poland, `j.lacki@mimuw.edu.pl`
[7]Georgia Tech, USA, `richard.pengatgmail.com`

3 Introduction

During the years, the Syllabus has evolved. Currently it has the following purposes:

- It specifies a small set of required prerequisite knowledge. Below, this is given in the category "Included, unlimited" and to some extent also in "Included, to be clarified".
- It serves as 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.
- As a consequence of the previous item, another purpose of the Syllabus is to help the organizers of national olympiads prepare their students for the IOI.

The Syllabus aims to achieve these goals by providing a classification of topics and concepts from mathematics and computer science. More precisely, this Syllabus classifies each topic into one of five categories:

♡ **Included, unlimited**

Topics in this category are considered to be prerequisite knowledge. Contestants are expected to know them. These topics can appear in task descriptions without further clarification.

Example: *Integer* in §4.1

△ **Included, 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 §4.2 DS2

⊖ **Included, not for task description**

Topics that belong to this category should not appear in tasks descriptions. However, developing solutions and understanding model solutions may require the knowledge of these topics.

Example: *Asymptotic analysis of upper complexity bounds* in §5.2 AL1

Note: This is the main category that should be of interest when preparing contestants for the IOI. It should be noted that this set of topics contains a wide range of difficulties, starting from simple concepts and ending with topics that can appear in problems that aim to distinguish among the gold medallists. It is **not** expected that all contestants should know everything listed in this category.

Outside of focus
Any topic that is not explicitly addressed by the Syllabus should be considered to belong to this category.

Contestants are not expected to have knowledge of these topics. Most competition tasks will not be related to any topics from this category.

However, this does not prevent the inclusion of a competition task that is related to a particular topic from this category. The ISC may wish to include such a competition task in order to broaden the scope of the IOI.

If such a task is considered for the IOI, the ISC will make sure that the task can reasonably be solved without prior knowledge of the particular topic, and that the task can be stated in terms of ♡ and △ concepts in a precise, concise, and clear way.

Examples of such tasks being used at recent IOIs include :

- Languages (a.k.a. Wikipedia) from IOI 2010 in Canada, and
- Odometer (a.k.a. robot with pebbles) from IOI 2012 in Italy, and
- Art class from IOI 2013 in Australia.

Excluded Some of the harder algorithmic topics are explicitly marked as excluded. It is guaranteed that there will not be a competition task that *requires* the contestants to know these areas.

Furthermore, the tasks will be set with the goal that knowledge of Excluded topics should not help in obtaining simpler solutions / solutions worth more points.

This category mainly contains hard textbook algorithms and advanced areas in mathematics.

Still, 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.

Due to the the common occurrence of tasks that can benefit from knowing excluded topics since the introduction of the syllabus, and the differing treatments of these topics in various national/regional competitions, this set is further divided into two parts in this version.

- **Excluded, but open to discussion**
  This contains mostly topics that are natural extensions of topics in ♡ and △ , or ones where the boundary is difficult to draw.

  Examples: *Rabin-Karp string hashing* in §5.2

- **Explicitly excluded**
  This contains mostly topics whose inclusion will result in categories of problems that are unaccessible to a significant portion of IOI participants.

  Examples: *Calculus* in §4.3

It is the hope that a consensus can be reached on most topics in the **Excluded, but open to discussion** category in the next few years.

Obviously, none of the categories can ever be exhaustively enumerated. Instead, the list given in the following Sections should serve as examples that map out the boundary: anything easier or similar to *Included* topics is to be considered Included as well, and anything similar or harder than the *Explicitly excluded* topics is Excluded, too.

If there is a particular topic for which you are not sure how it should be classified, we invite you to submit a clarification request to the current Syllabus maintainer.

Note that issues related to the usage of suitable terminology and notations in competition tasks are beyond the scope of this document.[8]

The rest of this document contains the classification of topics.

4 Mathematics

4.1 Arithmetics and Geometry

♡ Integers, operations (incl. exponentiation), comparison
♡ Basic properties of integers (sign, parity, divisibility)
♡ Basic modular arithmetic: addition, subtraction, multiplication

[8] See T. Verhoeff: *Concepts, Terminology, and Notations for IOI Competition Tasks*, `http://scienceolympiads.org/ioi/sc/documents/terminology.pdf`

- ⊖ Prime numbers
- ♡ Fractions, percentages
- ♡ Line, line segment, angle, triangle, rectangle, square, circle
- ♡ Point, vector, coordinates in the plane
- ♡ Polygon (vertex, side/edge, simple, convex, inside, area)
- △ Euclidean distances
- ⊖ Pythagorean theorem

Excluded, but open to discussion

- Additional topics from number theory.

Explicitly excluded:

- geometry in 3D or higher dimensional spaces
- analyzing and increasing precision of floating-point computations
- modular division and inverse elements
- complex numbers,
- general conics (parabolas, hyperbolas, ellipses)
- trigonometric functions

4.2 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 (inclusion/exclusion, complements, Cartesian products, power sets)

Explicitly excluded:

- Cardinality and countability (of infinite sets)

DS2. Basic logic

- ♡ First-order logic
- ♡ Logical connectives (incl. their basic properties)

- ♡ Truth tables
- ♡ Universal and existential quantification (Note: statements should avoid definitions with nested quantifiers whenever possible).
- ⊖ 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 $\wedge$, $\vee$, $\forall$, $\exists$.

Out of focus:

- Normal forms

Explicitly excluded:

- Validity
- 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)

DS4. Basics of counting

- ♡ Counting arguments (sum and product rule, arithmetic and geometric progressions, Fibonacci numbers)
- △ Permutations and combinations (basic definitions)
- △ Factorial function, binomial coefficients
- ⊖ Inclusion-exclusion principle
- ⊖ Pigeonhole principle
- ⊖ Pascal's identity, Binomial theorem

Explicitly excluded:

- Solving of recurrence relations
- Burnside lemma

DS5. Graphs and trees

△ Trees and their basic properties, rooted trees
△ 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
△ 'Decorated' graphs with edge/node labels, weights, colors
△ Multigraphs, graphs with self-loops
△ Bipartite graphs
⊖ Planar graphs

Explicitly Excluded

- Hypergraphs
- Specific graph classes such as perfect graphs
- Structural parameters such as treewidth and expansion
- Planarity testing
- Finding separators for planar graphs

DS6. Discrete probability

Applications where everything is finite (and thus arguments about probability can be easily turned into combinatorial arguments) are *Out of focus*, everything more complicated is *Explicitly excluded.*

4.3 Other Areas in Mathematics

Explicitly excluded:

- Geometry in three or more dimensions.
- Linear algebra, including (but not limited to):
  - Matrix multiplication, exponentiation, inversion, and Gaussian elimination

    - Fast Fourier transform
- Calculus,
- Statistics

5 Computing Science

5.1 Programming Fundamentals (PF)

PF1. Fundamental programming constructs (for abstract machines)

- ♡ Basic syntax and semantics of a higher-level language (at least one of the specific languages available at an IOI, as 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, etc.)[9]
- ⊖ The role of algorithms in the problem-solving process
- ⊖ Implementation strategies for algorithms (also see §6 SE1)
- ⊖ Debugging strategies (also see §6 SE3)
- △ The concept and properties of algorithms (correctness, efficiency)

PF3. Fundamental data structures

- ♡ Primitive types (boolean, signed/unsigned integer, character)
- ♡ Arrays (incl. multicolumn dimensional arrays)
- ♡ Strings and string processing
- △ Static and stack allocation (elementary automatic memory management)

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

- △ Linked structures
- △ Implementation strategies for graphs and trees
- △ Strategies for choosing the right data structure
- ⊖ Elementary use of real numbers in numerically stable tasks
- ⊖ The floating-point representation of real numbers, the existence of precision issues.[10]
- ⊖ Pointers and references

Out of focus:

- Data representation in memory,
- Heap allocation,
- Runtime storage management,
- Using fractions to perform exact calculations.

Explicitly excluded:

- Non-trivial calculations on floating point numbers, manipulating precision errors

Regarding floating point numbers, there are well-known reasons why they should be, in general, avoided at the IOI.[11] However, the currently used interface removes some of those issues. In particular, it should now be safe to use floating point numbers in some types of tasks – e.g., to compute some Euclidean distances and return the smallest one.

PF4. Recursion

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

---

[10] Whenever possible, avoiding floating point computations completely is the preferred solution.

[11] See G. Horváth and T. Verhoeff: *Numerical Difficulties in Pre-University Education and Competitions*, Informatics in Education 2:21–38, 2003

PF5. Event-driven programming

Some competition tasks may involve a dialog with a reactive environment. Implementing such an interaction with the provided environment is △.

Everything not directly related to the implementation of reactive tasks is *Out of focus*

5.2 Algorithms and Complexity (AL)

We quote from the IEEE-CS Curriculum:

> 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
- ⊖ Empirical performance measurements.

Out of focus:

- Identifying differences among best, average, and worst case behaviors,
- Little o, Omega, and Theta notation,
- Tuning parameters to reduce running time**, memory consumption or other measures of performance**

Explicitly 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
- ⊖ Divide-and-conquer
- ⊖ Backtracking (recursive and non-recursive), Branch-and-bound
- ⊖ Dynamic programming[12]

Out of focus:

- Heuristics
- Finding good features for machine learning tasks[13]
- Discrete approximation algorithms
- Randomized algorithms.

Explicitly excluded:

- Clustering algorithms (e.g. k-means, k-nearest neighbor)
- Minimizing multi-variate functions using numerical approaches.

AL3a. Algorithms

- ⊖ Simple algorithms involving integers: radix conversion, Euclid's algorithm, primality test by $\mathcal{O}(\sqrt{n})$ trial division, Sieve of Eratosthenes, factorization (by trial division or a sieve), efficient exponentiation
- ⊖ Simple operations on arbitrary precision integers (addition, subtraction, simple multiplication)[14]
- ⊖ Simple array manipulation (filling, shifting, rotating, reversal, resizing, minimum/maximum, prefix sums, histogram, bucket sort)

[12]The IEEE-CS Curriculum puts this under AL8, but we believe it belongs here.
[13]E.g., finding a good way to classify images in the IOI 2013 Art class problem.
[14]The necessity to implement these operations should be obvious from the problem statement.

- ⊖ **Simple string algorithms (e.g., naive substring search)**
- ⊖ sequential processing/search and binary search
- ⊖ Quicksort and Quickselect to find the k-th smallest element.
- ⊖ $\mathcal{O}(n \log n)$ worst-case sorting algorithms (heap sort, merge sort)
- ⊖ Traversals of ordered trees (pre-, in-, and post-order)
- ⊖ Depth- and breadth-first traversals
- ⊖ Applications of the depth-first traversal tree, such as:
  - Topological sort
  - Algorithms to determine the (existence of an) Euler path/cycle
- ⊖ Finding connected components and transitive closures.
- ⊖ Shortest-path algorithms (Dijkstra, Bellman-Ford, Floyd-Warshall)
- ⊖ Minimum spanning tree (Jarník-Prim and Kruskal algorithms)
- ⊖ $O(VE)$ time algorithm for computing maximum bipartite matching.
- ⊖ **Biconnectivity in undirected graphs (bridges, articulation points).**
- ⊖ **Connectivity in directed graphs (strongly connected components).**

Excluded, but open to discussion

- Maximum flow. Flow/cut duality theorem.
- ~~**Strongly connected components, bridges and articulation points.**~~

Explicitly excluded:

- Primal-dual graph algorithms (e.g. minimum cost arborescence)
- Lexicographical BFS, maximum adjacency search and their properties

AL3b. Data structures

- △ Stacks and queues
- ⊖ Representations of graphs (adjacency lists, adjacency matrix)
- △ Binary heap data structures
- ⊖ Representation of disjoint sets: the Union-Find data structure.

- △ Statically balanced binary search trees. Instances of this include binary index trees (also known as Fenwick trees) and segment trees (also known as interval trees and tournament trees).[15]
- ⊖ Balanced binary search trees[16]
- △ Augmented binary search trees
- ⊖ $O(\log n)$ time algorithms for answering lowest common ancestor queries in a static rooted tree.[17]
- ⊖ Creating persistent data structures by path copying or using fat nodes.
- ⊖ Composition of data structures, e.g. 2-D statically balanced binary trees
- △ **Tries**

Excluded, but considering inclusion

- String algorithms: there is evidence that the inter-reducibility between KMP, Rabin-Karp hashing, suffix arrays/tree, suffix automaton, and Aho-Corasick makes them difficult to separate.
- Heavy-light decomposition and separator structures for static trees.
- Data structures for dynamically changing trees and their use in graph algorithms.

Explicitly excluded:

- Complex heap variants such as binomial and Fibonacci heaps,
- Using and implementing hash tables (incl. strategies to resolve collisions)

AL4. Distributed algorithms

Out of focus

AL5. Basic computability

[15] Note that in computational geometry there are different data structures with similar names.

[16] Problems will not be designed to distinguish between the implementation of BBSTs, such as treaps, splay trees, AVL trees, or scapegoat trees

[17] Once again, different implementations meeting this requirement will not be distinguished.

All topics related to computability are *Explicitly excluded*. This includes the following: Tractable and intractable problems; Uncomputable functions; The halting problem; Implications of uncomputability

However, see AL7 for basic computational models.

AL6. The complexity classes P and NP

Topics related to non-determinism, proofs of NP-hardness (reductions), and everything related is *Explicitly excluded*.

Note that this section only covers the results usually contained in undergraduate and graduate courses on formal languages and computational complexity. The classification of these topics as *Explicitly excluded* does not mean that an NP-hard problem cannot appear at an IOI.

AL7. Automata and grammars

△ Understanding a simple grammar in Backus-Naur form

Out of focus:

- Formal definition and properties of finite-state machines,
- Context-free grammars and related rewriting systems,
- Regular expressions

Explicitly excluded:

- Properties other than the fact that automata are graphs and that grammars have parse trees.

AL8. Advanced algorithmic analysis

⊖ Basics of combinatorial game theory, winning and losing positions, minimax algorithm for optimal game playing

⊖ Amortized analysis.

Out of focus:

- Online algorithms
- Randomized algorithms

Explicitly excluded:

- Alpha-beta pruning
- Theory of combinatorial games, e.g. NIM game, Sprague-Grundy theory

AL9. Cryptographic algorithms

Out of focus

AL10. Geometric algorithms ⊖ Representing points, vectors, lines, line segments.

- ⊖ Checking for colinear points, parallel/orthogonal vectors, clockwise turns.
- ⊖ Intersection of two lines.
- ⊖ Computing area of a polygon.
- ⊖ Checking whether a (general/convex) polygon contains a point.
- ⊖ Coordinate compression.
- ⊖ $\mathcal{O}(n \log n)$ time algorithms for convex hull
- ⊖ Sweeping line method

Explicitly excluded:

- Point-line duality
- Halfspace intersection, Voronoi diagrams, Delaunay triangulations.
- Computing coordinates of circle intersections against lines and circles.
- Linear programming in 3 or more dimensions and its geometric interpretations.

AL11. Parallel algorithms

Out of focus

5.3 Other Areas in Computing Science

Except for GV (specified below), all areas are *Explicitly excluded.*

AR. Architecture and Organization

OS. Operating Systems

NC. Net-Centric Computing (a.k.a. cloud computing)

PL. Programming Languages

HC. Human-Computer Interaction

GV. Graphics and Visual Computing

Basic aspects of processing graphical data are *Out of focus*, everything else (including the use of graphics libraries such as OpenGL) is *Explicitly excluded.*

IS. Intelligent Systems

IM. Information Management

SP. Social and Professional Issues

CN. Computational Science

Notes: AR is about digital systems, assembly language, instruction pipelining, cache memories, etc. OS is about the *design* of operating systems, not their usage. PL is about the *analysis and design* of programming languages, not their usage. HC is about the *design* of user interfaces.

Usage of the operating system, GUIs and programming languages is covered in §7 and §5.1.

6 Software Engineering (SE)

We quote from the IEEE-CS Curriculum:

> 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 ⊖.

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

(See also SE2 immediately below)

Explicitly excluded: Software architecture, Design for reuse, 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.

Explicitly excluded: Programming by example, Debugging in the API environment, 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

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

Explicitly excluded: Testing tools, Configuration management tools 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.

Explicitly 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.

Explicitly excluded: Prototyping, 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.

Explicitly excluded: Validation planning, Object-oriented testing

SE7. Software evolution

Explicitly excluded: 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.

Explicitly excluded: Software quality assurance, Team management, Software measurement and estimation techniques, Project management tools

SE9. Component-based computing

Explicitly 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.

Explicitly excluded: Formal verification, Formal specification languages, Executable and non-executable specifications

SE11. Software reliability

Explicitly excluded

SE12. Specialized systems development

Explicitly excluded

7 Computer Literacy

The text of this section is ⊖ .

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.

> *Out of focus*: Calculator, Word-processors, Spreadsheet applications, Database management systems, E-mail clients, Graphics tools (drawing, painting)