

The International Olympiad in Informatics Syllabus

1 Version and status information

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 International Scientific Committee (ISC) as described in the IOI Regulations, Statute 3.13.

This is the official Syllabus version used for IOI 2025.

The most recent changes in this version of the Syllabus were made in February 2024 and published as a draft in March 2024. At the end of IOI 2024 this version of the document became the official Syllabus.

This Syllabus version contains multiple changes since the official 2024 version. You can find a summary of these changes in the draft version of this Syllabus, which is available at <https://algo.sk/ioi-syllabus/ioi-syllabus-2025-draft.pdf>. The draft version of this Syllabus also has all changes since the 2024 version highlighted.

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 also maintained the syllabus and made significant contributions: Michal Forišek,⁵ Jakub Łacki,⁶ and Richard

¹TU Eindhoven, The Netherlands, t.verhoeff@tue.nl

²University of Szeged, Hungary, horvath@inf.u-szeged.hu

³Warsaw University, Poland, diks@mimuw.edu.pl

⁴University of Waterloo, Canada, gvcormac@uwaterloo.ca

⁵Comenius University, Slovakia, misof@algo.sk

⁶Warsaw University, Poland, j.lacki@mimuw.edu.pl

Peng.⁷

You are welcome to send any feedback on the Syllabus to the current maintainer's e-mail address (misof@algo.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 <https://algo.sk/ioi-syllabus/>.

⁷Georgia Tech, USA, richard.peng@gmail.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 defined”.
- 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 six categories. Ordered by topic suitability, these are:

- ✓ Included, unlimited
- ✓📄 Included, to be defined
- ✓📄 Included, not for task description
- ? Outside of focus
- ✗🗑️ Excluded, but open to discussion
- ✗ Explicitly excluded

In the next section we explain the scope of each category.

4 Categories

This Syllabus classifies a selection of topics into six different categories. Obviously, such a set of topics can never be exhaustive. Instead, the list given in this Syllabus should serve as examples that map out the boundary. Topics not explicitly mentioned in the Syllabus should be classified as follows:

- Anything that is a prerequisite of an Included topic is also Included.
- Anything that is an extension of an Excluded topic or similar to an Excluded topic is also Excluded.
- Anything else that is not mentioned in the Syllabus is considered Outside of focus.

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

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.

Here are the definitions of the six possible classifications:

✓ **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 §5.1

✓📖 **Included, to be defined**

Contestants should know this topic, but when it appears in a task description, the statement should contain a sufficient definition. This category is usually applied in situations where a general concept that would be ✓ has many different “flavors” and a formal definition is needed to distinguish among those.

Example: *Directed graph* in §5.2 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.

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

Example: *Asymptotic analysis of upper complexity bounds* in §6.2 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 IOIs include:

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

x Explicitly 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 contains topics whose inclusion will result in problems that are unaccessible to a significant portion of IOI participants. This includes but is not limited to 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.

Examples: *Calculus* in §5.3

✖ Excluded, but open to discussion

As the Syllabus is a living document, there can be cases when we consider bringing in some of the Excluded topics. Usually, the topics in question are natural extensions of Included topics, or ones where drawing an exact boundary is difficult. Should such topics appear, they will be temporarily classified as “Excluded, but open to discussion”, and by doing so we encourage all members of the IOI community to give us feedback on these topics.

The rest of this document contains the classification of topics.

5 Mathematics

5.1 Arithmetics and Geometry

- ✓ Integers, operations (incl. exponentiation), comparison
- ✓ Basic properties of integers (sign, parity, divisibility)
- ✓ Basic modular arithmetic: addition, subtraction, multiplication
- ✓📄 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
- ✗ Geometry in three or more dimensions
- ✗ Analyzing and increasing precision of floating-point computations
- ✗ Modular division and inverse elements
- ✗ Complex numbers
- ✗ General conics (parabolas, hyperbolas, ellipses)
- ✗ Trigonometric functions

5.2 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)

- ✗ 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

- ? Normal forms

- ✗ 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
- ✗ Solving of recurrence relations
- ✗ Burnside lemma

DS5. Graphs and trees

- ✓☞ Undirected graphs (vertex/node, edge, degree, adjacency, vertex and edge labels)
- ✓☞ Directed graphs (in-degree, out-degree)
- ✓☞ Multigraphs, graphs with self-loops
- ✓☞ Paths in graphs (undirected and directed path, cycle, tour, walk; Euler tour; Hamiltonian path/cycle)
- ✓☞ Reachability (connected component, shortest distance)
- ✓☞ Trees (leaf, diameter, forest)
- ✓☞ Rooted trees (root, parent, child, ancestor, subtree, binary tree)
- ✓☞ Spanning trees (and general subgraphs)
- ✓☞ Traversal strategies
- ✓☞ Bipartite graphs
- ✓☞ Directed acyclic graphs
- ✓☞ Simple specific graph shapes (complete, complete bipartite, star)
- ✓☞ More general connectivity (biconnected undirected and strongly connected directed graphs)
- ✓☞ Structure of trees (center, centroid)
- ✓☞ Planar graphs
- ✓☞ Basic combinatorial properties of graphs⁹

⁹This item includes various relationships between the numbers of vertices, edges and connected components in graphs as well as vertex degrees and other similar properties. Examples include the Handshaking lemma and Euler characteristic for planar graphs.

- ✗ 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 **?**, everything more complicated is **✗**.

5.3 Other Areas in Mathematics

- ✗ Linear algebra, including (but not limited to):
 - Matrix multiplication, exponentiation, inversion, and Gaussian elimination
 - polynomial interpolation
 - Fast Fourier transform
- ✗ Calculus
- ✗ Theory of combinatorial games, e.g., bitwise xor solution to the general NIM game, Sprague-Grundy theory¹⁰
- ✗ Statistics

6 Computing Science

6.1 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

¹⁰Note that in AL3a some basic parts of combinatorial game theory are ✓📄, so tasks that involve a combinatorial game can still be acceptable if they have an ad-hoc solution that does not depend on prior knowledge of advanced general techniques.

- ✓ 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.)¹¹
- ✓📄 The role of algorithms in the problem-solving process
- ✓📄 Implementation strategies for algorithms (also see §7 SE1)
- ✓📄 Debugging strategies (also see §7 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)
- ✓📄 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.¹²
- ✓📄 Pointers and references

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

¹¹See G. Polya: *How to Solve It: A New Aspect of Mathematical Method*, Princeton Univ. Press, 1948

¹²Whenever possible, avoiding floating point computations completely is the preferred solution.

- ✗ 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.¹³ 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

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 ?.

6.2 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.

¹³See G. Horváth and T. Verhoeff: *Numerical Difficulties in Pre-University Education and Competitions*, Informatics in Education 2:21–38, 2003

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, etc.
- ✓📖 Time and space tradeoffs in algorithms
- ✓📖 Empirical performance measurements
- ? 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
- ✗ 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 programming¹⁴
- ? Heuristics
- ? Finding good features for machine learning tasks¹⁵
- ? Discrete approximation algorithms
- ? Randomized algorithms

¹⁴Common optimizations of dynamic programming such as convex hull, Knuth-Yao, or divide&conquer optimization are also ✓📖 but problems where a standard optimization is a mechanical last step are generally discouraged. Additionally, proof that the technique can be applied to that particular problem must also be within the scope of the IOI.

¹⁵E.g., finding a good way to classify images in the IOI 2013 Art class problem.

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

AL3a. Algorithms

- ✓📖 Simple algorithms on 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)¹⁶
 - ✓📖 Simple array manipulation (filling, shifting, rotating, reversal, resizing, minimum/maximum, prefix sums, histogram, bucket sort)
 - ✓📖 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 ordering and Euler paths/cycles
 - ✓📖 Finding connected components and transitive closures
 - ✓📖 Shortest-path algorithms (Dijkstra, Bellman-Ford, Floyd-Warshall)
 - ✓📖 Minimum spanning tree (Jarník-Prim and Kruskal algorithms)
 - ✓📖 Maximum bipartite matching ($\mathcal{O}(VE)$ algorithm based on augmenting paths)
 - ✓📖 Maximum network flow (Ford-Fulkerson $\mathcal{O}(|\text{flow}| \cdot E)$ algorithm based on augmenting paths)¹⁷
- Duality between maximum flows and minimum cuts

¹⁶The necessity to implement these operations should be obvious from the problem statement.

¹⁷Problems that require the computation of bipartite matchings or network flows should be designed in such a way that knowledge of more efficient algorithms or heuristic optimizations for these problems doesn’t give any significant advantage.

- ✓📖 Biconnectivity in undirected graphs (bridges, articulation points)
- ✓📖 Connectivity in directed graphs (strongly connected components)
- ✓📖 Basics of combinatorial game theory, winning and losing positions, minimax algorithm for optimal game playing
- ✗ Optimization problems that are easiest to analyze using matroid theory, problems based on matroid intersections (except for bipartite matching)
- ✗ Lexicographical BFS, maximum adjacency search and their properties
- ✗ More efficient implementations of bipartite matching and network flow algorithms.¹⁸
- ✗ Maximum matching in polynomial time on general graphs
- ✗ Minimum cost versions of maximum matchings and network flows
- ✗ Alpha-beta pruning

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).¹⁹
- ✓📖 Balanced binary search trees²⁰
- ✓📖 Augmented binary search trees

¹⁸Also note that linear programming is ✗ in AL10.

¹⁹Not to be confused with similarly-named data structures used in computational geometry.

²⁰Problems will not be designed to distinguish between the implementation of BBSTs, such as treaps, splay trees, AVL trees, or scapegoat trees

- ✓📄 $O(\log n)$ time algorithms for answering lowest common ancestor queries in a static rooted tree.²¹
- ✓📄 Decomposition of static trees (heavy-light decomposition, separator structures such as centroid decomposition)
- ✓📄 Persistent data structures²²
- ✓📄 Nesting of data structures, such as having a sequence of sets
- ✓📄 Tries
- ✗ String algorithms and data structures (KMP, Rabin-Karp hashing, suffix arrays/trees, suffix automata, Aho-Corasick)
- ✗ Complex heap variants such as binomial and Fibonacci heaps
- ✗ Using and implementing hash tables (incl. strategies to resolve collisions)
- ✗ Two-dimensional tree-like data structures (such as a 2D statically balanced binary tree or a treap of treaps) used for 2D queries
- ✗ Fat nodes and other more complicated ways of implementing persistent data structures
- ✗ Data structures for dynamically changing trees (e.g., link/cut trees) and their use in graph algorithms

AL4. Distributed algorithms

This entire section is ?.

AL5. Basic computability

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

However, see AL7 for basic computational models.

²¹Once again, different implementations meeting this requirement will not be distinguished.

²²Problems should not attempt to distinguish between implementation techniques. In particular, implementation by path copying should be sufficient.

AL6. The complexity classes P and NP

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

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 **X** 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
- ? Formal definition and properties of finite-state machines
- ? Context-free grammars and related rewriting systems
- ? Regular expressions
- X Properties other than the fact that automata are graphs and that grammars have parse trees

AL8. Advanced algorithmic analysis

- ✓📖 Amortized analysis
- ? Analysis of online algorithms, including competitive analysis²³
- ? Analysis of randomized algorithms

AL9. Cryptographic algorithms

This entire section is **?**.

²³Here we use “online algorithms” in the more general CS sense, including e.g. stream sampling algorithms and competitive online algorithms like k-server scheduling and page replacement. Note that section PF5 specifies that interactions between contestants’ programs and a grader are ✓📖, and thus reactive tasks that can be described as “answering queries online” are acceptable as long as the algorithms they use to answer the queries are ✓📖, including their analysis.

AL10. Geometric algorithms

In general, the ISC has a strong preference towards problems that can be solved using integer arithmetics to avoid precision issues. This may include representing some computed values as exact fractions, but extensive use of such fractions in calculations is discouraged.

Additionally, if a problem uses two-dimensional objects, the ISC prefers problems in which such objects are rectilinear.

- ✓📄 Representing points, vectors, lines, line segments
- ✓📄 Checking for collinear points, parallel/orthogonal vectors and clockwise turns (for example, by using dot products and cross products)
- ✓📄 Intersection of two lines
- ✓📄 Computing the area of a polygon from the coordinates of its vertices.²⁴
- ✓📄 Checking whether a (general/convex) polygon contains a point
- ✓📄 Coordinate compression
- ✓📄 $\mathcal{O}(n \log n)$ time algorithms for convex hull
- ✓📄 Sweeping line method
- ✗ 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
- ✗ Center of mass of a 2D object
- ✗ Computing and representing the composition of geometric transformations if the knowledge of linear algebra gives an advantage

AL11. Parallel algorithms

This entire section is **?**.

²⁴The recommended way of doing so is to use cross products or an equivalent formula. See e.g. http://geomalgorithms.com/a01-_area.html

6.3 Other Areas in Computing Science

Except for GV (specified below), all areas are ✗.

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 ?, everything else (including the use of graphics libraries such as OpenGL) is ✗.

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 §8 and §6.1.

7 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
-
- ✗ 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
-
- ✗ 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

- ✗ 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

- ✗ 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

- ✗ 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
- ✗ Validation planning
 - ✗ Object-oriented testing

SE7. Software evolution

- ✗ 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
- ✗ Software quality assurance
 - ✗ Team management
 - ✗ Software measurement and estimation techniques
 - ✗ Project management tools

SE9. Component-based computing

This entire section is ✗.

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

- ✗ Formal verification
- ✗ Formal specification languages
- ✗ Executable and non-executable specifications

SE11. Software reliability

This entire section is ✗.

SE12. Specialized systems development

This entire section is ✗.

8 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.

The following topics are all **?**: Calculator, Word-processors, Spreadsheet applications, Database management systems, E-mail clients, Graphics tools (drawing, painting).