# 1. Mathematics

## 1.1 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)
4 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 oating-point computations

modular division and inverse elements complex numbers,

general conics (parabolas, hyperbolas, ellipses) trigonometric functions

## 1.2 Discrete Structures (DS)

DS1. Functions, relations, and sets

4 Functions (surjections, injections, inverses, composition)
4 Relations (re exivity, symmetry, transitivity, equivalence relations, total/linear order relations, lexicographic order)
4 Sets (inclusion/exclusion, complements, Cartesian products, power sets)

Explicitly excluded:

Cardinality and countability (of in nite sets)

DS2. Basic logic

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

~ Truth tables

~ Universal and existential quanti cation (Note: statements should avoid de nitions with nested quanti ers 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 ^, _, 8, 9.
Out of focus:

Normal forms

Explicitly excluded:

Validity

Limitations of predicate logic

DS3. Proof techniques

4 Notions of implication, converse, inverse, contrapositive, negation, and contradiction

Direct proofs, proofs by: counterexample, contraposition, contra-diction

Mathematical induction

Strong induction (also known as complete induction)

~ Recursive mathematical de nitions (incl. mutually recursive def-initions)

DS4. Basics of counting

~ Counting arguments (sum and product rule, arithmetic and geo-metric progressions, Fibonacci numbers)

4 Permutations and combinations (basic de nitions)

4 Factorial function, binomial coe cients

Inclusion-exclusion principle
Pigeonhole principle

Pascal's identity, Binomial theorem

Explicitly excluded:

Solving of recurrence relations Burnside lemma

DS5. Graphs and trees

4 Trees and their basic properties, rooted trees

4 Undirected graphs (degree, path, cycle, connectedness, Euler/Hamil-ton path/cycle, handshaking lemma)

4 Directed graphs (in-degree, out-degree, directed path/cycle, Eu-ler/Hamilton path/cycle)

4 Spanning trees

4 Traversal strategies

4 `Decorated' graphs with edge/node labels, weights, colors

4 Multigraphs, graphs with self-loops

4 Bipartite graphs

Planar graphs

Explicitly Excluded

Hypergraphs

Speci c 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 nite (and thus arguments about probability can be easily turned into combinatorial arguments) are Out of focus, everything more complicated is Explicitly excluded.

## 1.3 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

# 2 Computing Science

## 2.1 Programming Fundamentals (PF)

PF1. Fundamental programming constructs (for abstract machines)

~ Basic syntax and semantics of a higher-level language (at least one of the speci c 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, separa-tion of concerns, generalization, specialization, case distinction, working backwards, etc.)[7]

The role of algorithms in the problem-solving process

Implementation strategies for algorithms (also see x6 SE1) Debugging strategies (also see x6 SE3)

4 The concept and properties of algorithms (correctness, e ciency)

PF3. Fundamental data structures

~ Primitive types (boolean, signed/unsigned integer, character)

~ Arrays (incl. multicolumn dimensional arrays)

~ Strings and string processing

4 Static and stack allocation (elementary automatic memory man-agement)

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

4 Linked structures

4 Implementation strategies for graphs and trees

4 Strategies for choosing the right data structure

Elementary use of real numbers in numerically stable tasks

The oating-point representation of real numbers, the existence of precision issues.[8]

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 oating point numbers, manipulating precision errors

Regarding oating point numbers, there are well-known reasons why they should be, in general, avoided at the IOI.[9] However, the currently used interface removes some of those issues. In particular, it should now be safe to use oating 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

[8] ***Whenever possible, avoiding oating point computations completely is the preferred solution.***

[9] ***See G. Horvath and T. Verhoe :*** Numerical Di culties 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 environ-ment. Implementing such an interaction with the provided environ-ment is 4.

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

## 2.2 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 e ciency 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

4 Algorithm speci cation, precondition, postcondition, correctness, invariants

Asymptotic analysis of upper complexity bounds (informally if possible)

Big O notation

Standard complexity classes (constant, logarithmic, linear, O(n log n), quadratic, cubic, exponential)

Time and space tradeo s in algorithms

Empirical performance measurements.

Out of focus:

Identifying di erences among best, average, and worst case be-haviors,

Little o, Omega, and Theta notation,

Tuning parameters to reduce running time

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[10]

Out of focus:

Heuristics

Finding good features for machine learning tasks[11] 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 $O(\sqrt{n})$ trial division, Sieve of Eratosthenes, factorization (by trial division or a sieve), e cient exponentiation

Simple operations on arbitrary precision integers (addition, sub-traction, simple multiplication)[12]

Simple array manipulation ( lling, shifting, rotating, reversal, re-sizing, minimum/maximum, pre x sums, histogram, bucket sort)

sequential processing/search and binary search

[10] *The IEEE-CS Curriculum puts this under AL8, but we believe it belongs here.*

[11] *E.g., nding a good way to classify images in the IOI 2013 Art class problem.*

[12] *The necessity to implement these operations should be obvious from the problem statement.*

Quicksort and Quickselect to  nd the k-th smallest element.

O(n log n) worst-case sorting algorithms (heap sort, merge sort) Traversals of ordered trees (pre-, in-, and post-order)

Depth- and breadth- rst traversals

Applications of the depth- rst 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 (Jarn k-Prim and Kruskal algorithms)

O(V E) time algorithm for computing maximum bipartite matching.

Excluded, but open to discussion

Maximum  ow. 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

4 Stacks and queues

Representations of graphs (adjacency lists, adjacency matrix)

4 Binary heap data structures

Representation of disjoint sets: the Union-Find data structure.

4 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). ***[13]***

Balanced binary search trees ***[14]***

***[13]Note that in computational geometry there are di erent data structures with similar names.***

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

4 Augmented binary search trees

O(log n) time algorithms for answering lowest common ancestor queries in a static rooted tree.[15]

Creating persistent data structures by path copying or using fat nodes.

Composition of data structures, e.g. 2-D statically balanced binary trees

Excluded, but considering inclusion

String algorithms: there is evidence that the inter-reducibility between KMP, Rabin-Karp hashing, su x arrays/tree, su x automaton, and Aho-Corasick makes them di cult 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

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

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.

[15] ***Once again, di erent implementations meeting this requirement will not be distin-guished.***

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

AL7. Automata and grammars

4 Understanding a simple grammar in Backus-Naur form

Out of focus:

Formal de nition and properties of nite-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.

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

## 2.3 Other Areas in Computing Science

Except for GV (speci ed 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 x7 and x5.1.

## 3 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 e ectively and e ciently 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-o , 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, e - cient program expressed in one of the allowed program-ming languages, possibly using standard or competition-speci c libraries.

{ Make their programs read data from and write data to text les 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-speci c libraries according to the provided speci cation.

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, Con guration 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 met-rics

SE5. Software requirements and speci cation

Functional and nonfunctional requirements

Basic concepts of formal speci cation 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 e ciency 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 con guration 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 manage-ment 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 e ciency of algo-rithms and programs.

Explicitly excluded: Formal veri cation, Formal speci cation languages, Executable and non-executable speci cations

SE11. Software reliability

Explicitly excluded

SE12. Specialized systems development

Explicitly excluded

## 4 Computer Literacy

The text of this section is .

Contestants should know and understand the basic structure and oper-ation 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 le management is helpful (creating folders, copying and moving les).

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-speci c 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 nd most appropriate.

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