European Junior Olympiad in Informatics Syllabus

The purpose of this document is to serve as a set of guidelines to help decide whether a task is appropriate for the European Junior Olympiad in Informatics (EJOI), but it cannot serve as a strong limitation regarding topics not explicitly or implicitly mentioned. 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 junior national olympiads prepare their students for the EJOI.

The Syllabus presented below has been compiled using the International Olympiad in Informatics (IOI) Syllabus and the topics excluded from the IOI Syllabus are noted with *.

1. Mathematics

1.1. Arithmetics

- ✅ Integers, operations (including exponentiation), comparison
- ✅ Fractions, percentages
- ✅ Basic properties of integers (sign, parity, divisibility)
- ✅ Prime numbers
- ✅ GCD and LCM
- ✅ Basic modular arithmetic: addition, subtraction, multiplication
- ✅ Representations of integers in different bases

- ❌ Additional topics from number theory
- ❌ Advanced modular arithmetic: division and inverse elements
- ❌ Complex analysis for increasing precision of floating-point computations
- ❌ Complex numbers

1.2. Geometry

- ✅ Line, line segment, angle, triangle, rectangle, square, circle
- ✅ Point, vector, coordinates in the plane
- ✅ Polygon (vertex, side/edge, simple, convex, inside, area)
- ✅ Euclidean distance
- ✅ Pythagorean theorem

- ❌ Geometry in 3D or higher dimensional spaces
- ❌ General conics (parabolas, hyperbolas, ellipses)
- ❌ Trigonometric functions

1.3. Discrete Structures (DS)

DS1. Sets, relations and functions

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

- ❌ Cardinality and countability (of infinite sets)

DS2. Basic logic

- ✓ First-order logic
- ✓ Logical connectives (incl. their basic properties)
- ✓ Truth tables
- ✓ Universal and existential quantification
- ✓ Using and applying basic rules for implication

- ✗ 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 (including mutually recursive definitions)

DS4. Basics of counting

- ✓ Counting arguments (sum and product rule, arithmetic and geometric progressions, Fibonacci numbers)
- ✓ Permutations, variations and combinations with or without repetition (basic definitions and applications)
- ✓ 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)
- ✓ Directed graphs (in-degree, out-degree) and directed acyclic graphs (DAG)
- ✓ Multigraphs, graphs with self-loops
- ✓ Bipartite graphs
- ✓ 'Decorated' graphs with edge/node labels, weights, colors
- ✓ Paths in graphs (undirected and directed path, cycle, Euler tour/trail, Hamilton path/cycle)
- ✓ Reachability (connected component, shortest distance)
- ✓ Trees (leaf, diameter, forest)
- ✓ Rooted trees (root, parent, child, ancestor, subtree)
- ✓ Spanning trees (subgraph)
- ✓ Traversal strategies
- ✓ Basic combinatorial properties of graphs[1]

[1] This item includes various relationships between the numbers of vertices, edges and connected components in graphs as well as vertex degrees and other similar properties. One example is the Handshaking lemma.

- ❌ Planar graphs
- ❌ Hypergraphs
- ❌ Specific graph classes such as perfect graphs

DS6. Discrete probability – ❌

1.4. Other Areas

- ✅ Matrices (definition)

- ❌ Theory of combinatorial games, e.g., NIM game, Sprague-Grundy theorem
- ❌ Linear algebra, including (but not limited to):
  - Matrix multiplication/exponentiation/inversion, and Gaussian elimination
  - Fast Fourier transform
- ❌ Calculus
- ❌ Statistics

2. Computer Science

2.1. Programming Fundamentals (PF)

PF1. Basic programming constructs

- ✅ Basic syntax and semantics of C++
- ✅ Variables, types, expressions, and assignment
- ✅ Simple I/O
- ✅ Conditional and iterative control structures
- ✅ Functions and parameter passing
- ✅ Recursion
- ✅ Bitwise operations
- ✅ Structured decomposition

PF2. Fundamental data structures

- ✅ Primitive types (boolean, signed/unsigned integer, character)
- ✅ Arrays
- ✅ Strings and string processing
- ✅ Static and stack allocation (elementary automatic memory management)
- ✅ Linked structures
- ✅ Implementation strategies for graphs and trees
- ✅ Elementary use of real numbers in numerically stable tasks
- ✅ The floating-point representation of real numbers, the existence of precision issues[2]
- ✅ Pointers and references
- ✅ Data representation in memory

- ❌ Heap allocation
- ❌ Runtime storage management
- ❌ Non-trivial calculations on floating-point numbers, manipulating precision errors

[2] Whenever possible, avoiding floating point calculations completely is the preferred solution, but it is not expected the need for extensive use of fractions to perform exact calculations.

PF3. Event-driven programming

Some competition tasks may involve a dialog with a reactive environment. This will include implementation of such an interaction with the provided environment.

2.2. Algorithms and Complexity (AL)

AL1. Algorithmic analysis

- ✅ Algorithm specification, precondition, postcondition, correctness, invariants
- ✅ Asymptotic analysis of upper complexity bounds
- ✅ Amortized analysis
- ✅ Big O notation
- ✅ Standard complexity classes: constant, logarithmic, linear, $O(n \log n)$, quadratic, cubic, exponential, etc.
- ✅ Time and space trade-offs in algorithms
- ✅ Empirical performance measurements
- ✅ Identifying differences among best, average, and worst-case behaviors
- ✅ 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 (except the simple recurrent relation used to analyze merge sort)

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, including (but not limited to):
  - basic and classical DP
  - DP with bitmasks
  - digit DP
  - DP on tree and DAG

- ❌ Meet in the middle
- ❌ Square root decomposition
- ❌ Heuristics
- ❌ Discrete approximation algorithms
- ❌ Randomized algorithms
- ❌ Finding good features for machine learning tasks
- ❌ Clustering algorithms (e.g. k-means, k-nearest neighbor)
- ❌ Minimizing multi-variate functions using numerical approaches

AL3a. Basic 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), fast exponentiation
- ✓ Simple operations on arbitrary precision integers (addition, subtraction, multiplication)
- ✓ Simple array manipulation (filling, shifting, rotating, reversal, resizing, minimum/maximum, prefix sums, histogram, count sort)
- ✓ Sliding window and two pointers
- ✓ Simple string algorithms (e.g. naive substring search)
- ✓ Sequential processing/search and binary search (also binary search the answer)

AL3b. Advanced algorithms

- ✓ Bucket sort and radix sort
- ✓ Quicksort and Quickselect to find the k-th smallest element
- ✓ $O(n \log_2 n)$ worst-case sorting algorithms (heap sort, merge sort)
- ✓ Traversals of ordered trees (pre-, in-, and post-order)
- ✓ Basics of combinatorial game theory, winning and losing positions, minimax algorithm for optimal game playing
- ✓ Binary lifting

- ✗ Extended Euclid's algorithm
- ✗ Arithmetical expressions and calculating the value using shunting yard algorithm
- ✗ 2-SAT
- ✗ Hashing
- ✗ Advanced string algorithms such as Rabin-Karp, KMP, Z-algorithm, Aho-Corasick
- ✗ Complex dynamic programming optimizations such as divide and conquer, convex hull trick

AL3c. Graph algorithms

- ✓ Depth- and breadth-first traversals
- ✓ Applications of the depth-first search, such as topological ordering and Euler tour/trail
- ✓ Finding connected components and transitive closures
- ✓ Shortest-path algorithms (Dijkstra, Bellman-Ford, Floyd-Warshall)
- ✓ Minimum spanning tree (Jarnik-Prim and Kruskal algorithms)
- ✓ Graph extension[3]

- ✗ Lexicographical BFS, maximum adjacency search and their properties
- ✗ Biconnectivity in undirected graphs (bridges, articulation points)*
- ✗ Connectivity in directed graphs (strongly connected components)*
- ✗ Maximum bipartite matching*
- ✗ Maximum flow. Flow/cut duality theorem

[3] Technique that modifies the graph and adds additional information to each vertex and in that sense 'extends' the initial information for every vertex. For example, if we have a graph that is a road network and every road has a travel time, we can add the current time information to every vertex and look at the vertices as pairs – (initial number, current time).

AL3d. 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
- ✓ Coordinate compression
- ✓ Sweeping line method

- ✗ Checking for collinear points, parallel/orthogonal vectors and clockwise turns (for example, by using determinant evaluation or cross product and dot product of two-dimensional vectors)*
- ✗ Computing the area of a polygon from the coordinates of its vertices*
- ✗ Intersection of two lines*
- ✗ Checking whether a general polygon contains a point*
- ✗ Advanced algorithms for finding convex hull*
- ✗ 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

AL4. Data structures

- ✓ Stacks, queues and double-ended queues
- ✓ STL data structures: pair, vector, priority queue, (multi)set, (multi)map and unordered structures
- ✓ Representations of graphs (adjacency lists, adjacency matrix, edge list)
- ✓ Binary heap data structures
- ✓ Representation of disjoint sets: the Union-Find data structure
- ✓ Statically balanced binary search trees. Instances of this include binary indexed trees (also known as Fenwick trees) and segment trees (also known as interval trees and tournament trees).
- ✓ Sparse table for LCA, RMQ queries
- ✓ Nesting of data structures, such as having a sequence of sets

- ✗ Lazy propagation technique
- ✗ Merge-sort tree
- ✗ Persistent data structures*
- ✗ Balanced binary search trees*
- ✗ Augmented binary search trees*
- ✗ Cartesian tree
- ✗ Two-dimensional tree-like data structures (such as a 2D statically balanced binary tree or a treap of treaps) used for 2D queries
- ✗ Complex heap variants such as binomial and Fibonacci heaps

- ❌ Trie*
- ❌ String data structures such as suffix array/tree/automata
- ❌ Decomposition of static trees (heavy-light decomposition, separator structures such as centroid decomposition)*
- ❌ Euler tour technique
- ❌ Data structures for dynamically changing trees and their use in graph algorithms
- ❌ Using and implementing hash tables (incl. strategies to resolve collisions) but one is expected to know and use STL unordered data structures

AL5. Distributed algorithms – ❌

AL6. Cryptographic algorithms – ❌

AL7. Parallel algorithms – ❌

2.3. Other Areas

- Basic computability – ❌
- The complexity classes of P and NP – ❌
- Automata and grammars – ❌