Department of Computer Science
Memorial University

Algorithms and Software Development Comprehensive Exam Guidelines

Duration:
• 2-hour exam

Aim:
• Examine the candidates' foundations of Computer Science in the areas of algorithms and software development corresponding to the Core-Tier1 Topics of the ACM Knowledge Areas of Software Development Fundamentals (SDF) and Algorithms and Complexity (AL).

Topics:
Topics are described using the Knowledge Areas of the ACM Computer Science Curricula 2013. Percentage between brackets after a topic indicates corresponding weight in the exam for that topic.

AL/Basic analysis and SDF/Algorithms and Design (20%)
• Use of pseudocode to represent algorithms.
• Understanding “best”, “expected” and “worst” case behaviour of an algorithm, asymptotic analysis, O-notation, logarithmic/linear/quadratic/exponential running time.
• Problem-solving strategies:
  ○ Iterative and recursive mathematical functions
  ○ Iterative and recursive traversal of data structures
  ○ Identify the base case and the general case of a recursively-defined problem

SDF/Fundamental Programming Concepts (20%)
• Definition of abstraction, program decomposition, encapsulation, information hiding and recursion.
• Syntax and semantics of a higher-level language:
  ○ Design, implement, test and debug programs that use variables, expressions, assignments, simple I/O, file I/O, control structures (conditional and iterative), definition of functions, recursion and parameter passing.
  ○ Higher-level language shall be agreed upon by the examiner and examinees.

AL/Algorithmic Strategies (25%)
• Understand and be able to apply brute-force, greedy, divide-and-conquer, recursive backtracking and dynamic programming algorithmic strategies.
• Know examples of algorithms for each of the algorithmic strategies mentioned and be able to write pseudocode to represent these algorithms.
• Trace (show) the execution of algorithms on an example.
• Given a problem, solve it using an appropriate algorithmic approach, provide pseudocode for the solution and implement it in a higher-level language.
AL/Fundamental Data Structures and Algorithms, and SDF/Fundamental Data Structures (30%)

- Be able to state time complexity, represent with pseudocode, indicate which data structure should be used, and show the execution of the following standard algorithms.
  - Common search and sorting algorithms, including mergesort, heapsort, quicksort, insertion sort, selection sort, bubble sort, radix/bucket sort, binary search, linear search.
  - Simple numerical algorithms, such as finding average, min, max, and greatest common divisor.
  - Basic graph algorithms, including breadth first search, depth first search, minimum spanning tree and shortest path algorithms such as Dijkstra’s algorithm.

- Define and describe properties, common operations, and standard applications (uses) of the following data structures.
  - Strings, arrays, lists, sets, maps and linked lists
  - Stacks and queues
  - Rooted trees (general and binary)
  - Heaps and priority queues
  - Binary search trees, including balancing strategies
  - Hash tables, including collision resolution strategies and examples of hash functions.
  - Graphs, including concepts of adjacency list/adjacency matrix representation, directed/undirected graph, (free) tree, DAG

- Be able to write programs using the data structures listed in the previous item.
- Be able to evaluate algorithms in terms of their suitability to solve a given problem and select from a range of algorithms the most suitable for the given task providing justification for that selection. Be able to implement the chosen algorithm in a higher-level language and apply it to the given problem/task.

Extra material (5%)

- Specific algorithms using data structures or algorithmic strategies mentioned in this guide, and agreed upon by the examiner and examinees.

References:


Or

*Introduction to algorithms*, 3rd edition by Cormen, Leiserson, Rivest and Stein. Sections: I (1-4.1), II, III, IV (15-16) and VI, excluding starred subsections.

MUN Computer Science Courses Equivalences: