Algorithmic Foundations (AL)

Preamble

Algorithms and data structures are fundamental to computer science and software engineering since every theoretical computation and real-world program consists of algorithms that operate on data elements possessing an underlying structure. Selecting appropriate computational solutions to real-world problems benefits from understanding the theoretical and practical capabilities, and limitations, of available algorithms and paradigms, including their impact on the environment and society. Moreover, this understanding provides insight into the intrinsic nature of computation, computational problems, and computational problem-solving as well as possible solution techniques independent of programming language, programming paradigm, computer hardware, or other implementation aspects.

This knowledge area focuses on the nature of algorithmic computation including the concepts and skills required to design and analyze algorithms for solving real-world computational problems. It complements the implementation algorithms and data structures found in the Software Development Foundations (SDF) knowledge area. As algorithms and data structures are essential in all advanced areas of computer science, this area provides the algorithmic foundations that every computer science graduate is expected to know. Exposure to the breadth of these foundational AL topics is designed to provide students with the basis for studying additional topics in algorithmic computation in more depth and for learning advanced algorithms across a variety of knowledge areas and disciplines.

Changes since CS 2013

This area has been renamed to better reflect its foundational scope since topics in this area focus on the theoretical foundations of complexity and computability. They also provide the foundational prerequisites for advanced study in computer science. Additionally, topics focused on complexity and computability have been clearly separated into their respective knowledge units.To reinforce the important impact of computation on society, an Algorithms and Society unit has been added with the expectation that Societal, Ethical, and Professional (SEP) implications be addressed in some way during every lecture hour in the AL knowledge area.

The increase of four CS core hours acknowledges the importance of this foundational area in the computer science curriculum and returns it to the 2001 level. Despite this increase, there is a significant overlap in hours with the Software Development Fundamentals (SDF) and Mathematical Foundations (MSF) areas. There is also a complementary nature of the units in this area since, for example, linear search of an array covers topics in AL-Fundamentals and can be used to simultaneously explain AL-Complexity, e.g., O(n), and AL-Strategies, e.g. Brute-Force.

The KA hours primarily reflect topics studied in a stand-alone computational theory course and the availability of additional hours when such a course is included in the curriculum.

Core Hours

Knowledge Unit	CS Core	KA Core
Foundational Data Structures and Algorithms	14	6
Algorithmic Strategies	6	
Complexity Analysis	6	3
Computational Models and Formal Languages	6	23
Algorithms and Society	Included in SEP hours	
Total	**32**	**32**

Knowledge Units

AL-Foundational: Foundational Data Structures and Algorithms

CS Core:

1. Abstract Data Types (See also: SDF-ADT, FPL-Types: 1)
    a. Dictionary Operations (insert, delete, find)
    b. Objects (See also: FPL-OO: 2a)
2. Arrays (See also: SDF-Fundamentals, SDF-ADT)
    a. Numeric vs. Non-numeric, Character Strings
    b. Single (Vector) vs. Multidimensional (Matrix)
3. Records/Structs/Tuples (See also: FPL-Types: 1)
4. Linked Lists (See also: SDF-ADT)
    a. Single vs. double and Linear vs. Circular
5. Stacks (See also: SDF-ADT, AL-Models)
6. Queues and Dequeues (See also: SDF-ADT)
7. Hash Tables / Maps (See also: SDF-ADT)
    a. Collision Resolution and Complexity (e.g., probing, chaining, rehash)
8. Graphs (e.g., [un]directed, [a]cyclic, [un]connected, and [un]weighted) (See also: MSF-Discrete: 7)
    a. Adjacency list vs. matrix representations
9. Trees (See also: MSF-Discrete: 7)
    a. Binary, n-ary, and search trees

    b. Balanced (e.g., AVL, Red-Black)
10. Sets (See also: MSF-Discrete 1)
11. Search Algorithms (See also: SDF-Algorithms)
    a. O(n) (e.g., linear/sequential search)
    b. O($\log_2 n$) (e.g., binary search)
    c. O($\log_b n$) (e.g. depth/breadth-first tree)
12. Sorting Algorithms (e.g., stable, unstable) (See also: SDF-Algorithms)
    a. $O(n^2)$ complexity (e.g., insertion, selection),
    b. $O(n \log n)$ complexity (e.g., quicksort, merge, timsort)
13. Graph Algorithms
    a. Shortest Path (e.g., Dijkstra's, Floyd's)
    b. Minimal spanning tree (e.g., Prim's, Kruskal's)

KA Core:

1. Heaps and Priority Queues
2. Sorting Algorithms
    a. $O(n \log n)$ heapsort
    b. Pseudo O(n) complexity (e.g., bucket, counting, radix)
3. Graph Algorithms
    a. Transitive closure (e.g., Warshall's Algorithm)
    b. Topological sort
4. Matching
    a. Efficient String Matching (e.g., Boyer-Moore, Knuth-Morris-Pratt)
    b. Longest common subsequence matching
    c. Regular expression matching

Non Core:

5. Cryptography Algorithms (e.g., SHA-256) (See also: SE-Cryptography, MSF-Discrete: 5)
6. Parallel Algorithms (See also: PDC-Algorithms, FPL-Parallel)
7. Consensus algorithms (e.g., Blockchain) (See also: SE-Cryptography: 14)
    a. Proof of work vs. proof of stake (See also: SEP-Sustainability: 3)
8. Quantum computing algorithms (See also: AR-Quantum: 6)
    a. Oracle-based (e.g. Deutsch-Jozsa, Bernstein-Vazirani, Simn)
    b. Superpolynomial speed-up via QFT (e.g., Shor's algorithm)
    c. Polynomial speed-up via amplitude amplification (e.g., Grover's algorithm)

Illustrative Learning Outcomes:

CS Core:

1. For each Fundamental ADT/Data Structure in this unit:
    a. Articulate its definition, properties, representation(s), and associated ADT operations,
    b. Using a real-world example, explain step-by-step how the ADT operations associated with the data structure transform it.
2. For each of the algorithmic in this unit:
    a. Use a real-world example, show step-by-step how the algorithm operates.

3. For each of the algorithmic approach in this unit:
    a. Give a prototypical example of the approach (e.g., Quicksort for Sorting)
4. Given requirements for a real-world application, create multiple design solutions using various data structures and algorithms. Subsequently, evaluate the suitability, strengths, and weaknesses of the selected approach for satisfying the requirements.
5. Explain how collision avoidance and collision resolution is handled in hash tables.
6. Discuss factors other than computational efficiency that influence the choice of algorithms, such as, programming time, maintainability, and the use of application-specific patterns in the input data.

KA Core:

7. Describe the heap property and the use of heaps as an implementation of a priority queue.
8. For each of the algorithms and algorithmic approaches in the KA core topics:
    a. Give a prototypical example of the algorithm,
    b. Use a real-world example, show step-by-step how the algorithm operates.

AL-Strategies: Algorithmic Strategies

CS Core:

1. Paradigms
    a. Brute-Force (e.g., linear search, selection sort, traveling salesperson, knapsack)
    b. Decrease-and-Conquer
        i. By a Constant (e.g., insertion sort, topological sort),
        ii. By a Constant Factor (e.g., binary search),
        iii. By a Variable Size (e.g., Euclid's algorithm)
    c. Divide-and-Conquer (e.g., Binary Search, Quicksort, Mergesort, Strassen's)
    d. Greedy (e.g., Dijkstra's, Kruskal's)
    e. Transform-and-Conquer
        i. Instance simplification (e.g. find duplicates via list presort)
        ii. Representation change (e.g., heapsort)
        iii. Problem reduction (e.g., least-common-multiple, linear programming)
        iv. Dynamic Programming (e.g., Floyd's)
    f. Space vs. Time Tradeoffs (e.g., hashing) (See also: AL-Fundamentals)
2. Handling Exponential Growth (e.g., heuristics, A*, branch-and-bound, backtracking)
3. Iteration vs. Recursion (e.g., factorial) (See also: MSF-Discrete: 2)

KA Core:

4. Paradiams
    a. Approximation Algorithms
    b. Iterative improvement (e.g., Ford-Fulkerson, Simplex)
    c. Randomized/Stochastic Algorithms (e.g., Max-Cut, Balls and Bins)

Non Core:

5. Quantum computing (See also AL-Fundamentals: 8, AL-Models: 8)

Illustrative Learning Outcomes

CS Core:

1. For each of the paradigms in this unit
    a. Articulate its definitional characteristics,
    b. Give an example that demonstrates the paradigm including explaining how this example satisfies the paradigm's characteristics
2. For each of the algorithms in the AL-Fundamentals unit:
    a. Describe the paradigm used by the algorithm and how it exemplifies this paradigm
3. Given an algorithm, describe the paradigm used by the algorithm and how it exemplifies this paradigm
4. Give a real-world problem, determine appropriate algorithmic paradigms and algorithms from these paradigms that address the problem including considering the tradeoffs among the paradigms and algorithms selected.
5. Give an example of an iterative and a recursive algorithm that solves the same problem including explaining the benefits and disadvantages of each approach.
6. Determine if a greedy approach leads to an optimal solution.
7. Explain at least one approach for addressing a computational problem whose algorithmic solution is exponential.

AL-Complexity: Complexity

CS Core:

1. Complexity Analysis Framework
    a. Best, average, and worst case performance of an algorithm
    b. Empirical and Relative (Order of Growth) Measurements
    c. Input Size and Primitive Operations
    d. Time and Space Efficiency
2. Asymptotic complexity analysis (average and worst case bounds)
    a. Big-O, Big-Omega, and Big-Theta formal notations
    b. Foundational complexity classes and representative examples/problems
        i. $O(1)$ *Constant* (e.g., Array Access)
        ii. $O(\log_2 n)$ *Logarithmic* (e.g., Binary Search)
        iii. $O(n)$ *Linear* (e.g., Linear Search)
        iv. $O(n \log_2 n)$ *Log Linear* (e.g., Mergesort
        v. $O(n^2)$ *Quadratic* (e.g., Selection Sort)
        vi. $O(n^3)$ *Cubic* (e.g., Gaussian Elimination)
        vii. $O(2^n)$ *Exponential* (e.g., Knapsack, SAT, TSP, All Subsets)
        viii. $O(n!)$ *Factorial* (e.g., Hamiltonian Circuit, All Permutations)
3. Empirical measurements of performance
4. Tractability and Intractability

    a. P, NP and NP-Complete complexity classes
    b. NP-Complete problems (e.g., SAT, Knapsack, TSP)
    c. Reductions
5. Time and space trade-offs in algorithms.

KA Core:

6. Little-o and Little-Omega notations
7. Recursive Analysis: (e.g., recurrence relations, Master theorem, substitution)
8. Amortized Analysis
9. Turing Machine-Based Models of Complexity
    a. Time complexity (See also: AL-Models)
        i. P, NP, NP-C, and EXP classes
        ii. Cook-Levin Theorem
    b. Space Complexity
        i. NSpace and PSpace
        ii. Savitch's Theorem

Illustrative Learning Outcomes

CS Core:

1. Explain what is meant by "best", "average", and "worst" case behavior of an algorithm..
2. State and explain the formal definitions of Big-O, Big-Omega, and Big-Theta notations and how they are used to describe the amount of work done by an algorithm.
3. Compare and contrast each of the foundational complexity classes listed in this unit.
4. For each foundational complexity class in this unit:
    a. Give an algorithm that demonstrates the associated runtime complexity.
5. For each algorithm in the AL-Fundamentals unit:
    a. Give its runtime complexity class and explain why it belongs to this class.
6. Determine informally the foundational complexity class of simple algorithms.
7. Perform empirical studies to determine and validate hypotheses about the runtime complexity of various algorithms by running algorithms on input of various sizes and comparing actual performance to the theoretical analysis.
8. Give examples that illustrate time-space trade-offs of algorithms.
9. Explain how tree balance affects the efficiency of various binary search tree operations.
10. Explain to a non-technical audience the significance of tractable versus intractable algorithms using an intuitive explanation of Big-O complexity.
11. Explain the significance of NP-Completeness.
12. Describe NP-Hard as a lower bound and NP as an upper bound for NP-Completeness.
13. Provide examples of NP-complete problems.

KA Core:

14. Use recurrence relations to determine the time complexity of recursively defined algorithms.
15. Solve elementary recurrence relations using some form of the Master Theorem.
16. Use Big-O notation to give upper case bounds on time/space complexity of algorithms.
17. Explain the Cook-Levin Theorem and the NP-Completeness of SAT.

18. Define the classes P and NP.
19. Prove that a problem is NP-Complete by reducing a classic known NP-C problem to it (e.g., 3SAT and Clique)
20. Define the P-space class and its relation to the EXP class.

AL-Models: Computational Models and Formal Languages

CS Core:

1. Formal Automata
    a. Finite State
    b. Pushdown (See also: AL-Fundamentals: 5, SDF-ADT)
    c. Linear Bounded
    d. Turing Machine
2. Formal Languages, Grammars and Chomsky Hierarchy (See also: FPL-H Translation, FPL-J Syntax)
    a. Regular (Type-3)
        i. Regular Expressions
    b. Context-Free (Type-2)
    c. Context-Sensitive (Type-1)
    d. Recursively Enumerable (Type-0)
3. Relations among formal automata, languages, and grammars
4. Decidability, (un)computability, and halting
5. The Church-Turing Thesis
6. Algorithmic Correctness
    a. Invariants (e.g., in: iteration, recursion, tree search)

KA Core:

1. Deterministic and nondeterministic automata
2. Pumping Lemma Proofs (See also: MSF-Discrete: 3)
    a. Finite State/Regular
    b. Pushdown Automata/Context-Free
3. Decidability
    a. Arithmetization and Diagonalization (See also: MSF-Discrete: 1)
4. Reducibility and reductions
5. Time Complexity based on Turing Machine
6. Space Complexity (e.g., PSPACE, Savitch's Theorem)
7. Equivalent Models of Algorithmic Computation
    a. Turing Machines and Variations (e.g., multi-tape, non-deterministic)
    b. Lambda Calculus (See also: FPL-Functional)
    c. Mu-Recursive Functions

Non Core:

8. Quantum Computation (See also: AR-Quantum)

a. Postulates of quantum mechanics
   i. State Space
   ii. State Evolution
   iii. State Composition
   iv. State Measurement
b. Column vector representations of Qubits
c. Matrix representations of quantum operations
d. Quantum Gates (e.g., XNOT, CNOT)

Illustrative Learning Outcomes

CS Core:

1. For each formal automata in this unit:
   a. Articulate its definition comparing its characteristics with this unit's other automata,
   b. Using an example, explain step-by-step how the automata operates on input including whether it accepts the associated input,
   c. Give an example of inputs that can and cannot be accepted by the automata.
2. Given a real-world problem, design an appropriate automaton that addresses the problem.
3. Design a Regular Expression to accept a sentence from a Regular language.
4. Explain the difference between Regular Expressions (Type-3 acceptors) and Re-Ex pattern matchers (Type-2 acceptors) used in programming languages.
5. For each formal language/grammar in this unit
   a. Articulate its definition comparing its characteristics with the others in this unit,
   b. Give an example of inputs that can and cannot be accepted by the language/grammar.
6. Describe a univTuring Machine.
7. Explain how decidability and computability for various automata are related.
8. Explain the Halting problem, why it has no algorithmic solution, and its significance for real-world algorithmic computation.
9. Give examples of classic uncomputable problems.
10. Explain the Church-Turing Thesis and its significance for algorithmic computation.
11. Explain how invariants assist in proving the correctness of an algorithm as a formal model.

Illustrative Learning Outcomes

KA Core:

1. For each formal automata in this unit
   a. Compare/contrast its deterministic and nondeterministic capabilities.
2. Use a pumping lemma to prove the limitations of Finite State and Pushdown automata.
3. Use arithmetization and diagonalization to prove Turing Machine Halting/Undecidability.
4. Explain a reduction such as between Halting and Undecidability of the language accepted by a Turing Machine, where one has been previously proven by diagonalization.
5. Convert among equivalently powerful notations for a language, including among DFAs, NFAs, and regular expressions, and between PDAs and CFGs
6. Explain Rice's Theorem and its significance.

7. Give an example proof of a problem that is uncomputable by reducing a classic known uncomputable problem to it
8. Explain the Primitive and General Recursive functions (zero, successor, selection, primitive recursion, composition, and Mu), their significance, and Turing Machine implementations.
9. Explain how computation is performed in Lambda Calculus (e.g., Alpha Conversion and Beta-Reduction)

Non Core:

10. For a quantum system give examples that explain the following postulates:
    a. State Space: system state represented as a unit vector in Hilbert space,
    b. State Evolution: the use of unitary operators to evolve system state,
    c. State Composition: the use of tensor product to compose systems states,
    d. State Measurement: the probabilistic output of measuring a system state.
11. Explain the operation of a quantum XNOT or CNOT gate on a quantum bit represented as a matrix and column vector respectively

AL-SEP: Society, Ethics, and Professionalism

CS Core: (See also: SEP-Context, SEP-Sustainability)

1. Social, Ethical, and Secure Algorithms
2. Algorithmic Fairness (e.g., Differential Privacy)
3. Accountability/Transparency
4. Responsible algorithms
5. Economic and other impacts of inefficient algorithms
6. Sustainability

Illustrative Learning Outcomes

CS Core:

1. Devise algorithmic solutions to real-world societal problems, such as routing an ambulance to a hospital
2. Predict and explain the impact that an algorithm may have on the environment and society when used to solve real-world problems taking into account that it can affect different societal groups in different ways and the algorithm's sustainability.
3. Prepare a presentation that justifies the selection of appropriate data structures and/or algorithms to solve a given real-world industry problem.
4. Give an example that articulates how differential privacy protects knowledge of an individual's data.
5. Describe the environmental impacts of design choices that relate to algorithm design.
6. Discuss the tradeoffs involved in proof-of-work and proof-of-stake algorithms.

Professional Dispositions

- **Meticulous:** As an algorithm is a formal solution to a computational problem, attention to detail is important when developing and combining algorithms.
- **Persistent:** As developing algorithmic solutions to computational problems can be challenging, computer scientists must be resolute in pursuing such solutions
- **Inventive:** As computer scientists develop algorithmic solutions to real-world problems, they must be inventive in developing solutions to these problems.

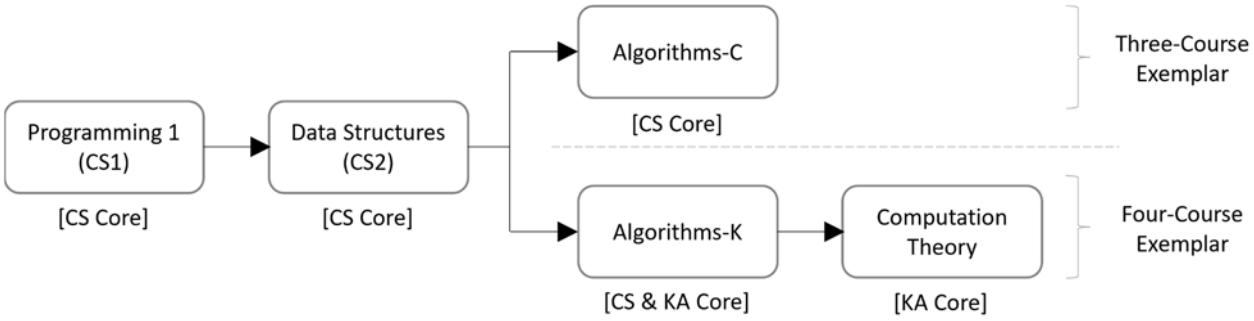
Math Requirements

Required:

- MSF-Discrete

Course Packaging Suggestions

As depicted in the following figure, the committee envisions two common approaches for addressing foundational AL topics in CS courses. Both approaches included required introductory Programming (CS1) and Data Structures (CS2) courses. In the three-course approach, all CS Core topics are covered. Alternatively, in the four-course approach, AL-Model unit CS and KA core topics are addressed in a Computational Theory focused course, which leaves room to address additional KA topics in the third Algorithms course. Both approaches assume Big-O analysis is introduced in the Data Structures (CS2) course and that graphs are taught in the third Algorithms course. The committee recognizes that there are many different approaches for packaging AL topics into courses including, for example, introducing graphs in CS2 Data Structures, Backtracking in an AI course, and AL-Model topics in a theory course which also addresses FPL topics. The given example is simply one way to cover the entire AL CS Core in three introductory courses.



Courses Common to Three and Four Course Exemplars

Programming 1 (CS1)

- AL-Foundational: Fundamental Data Structures and Algorithms (2 hours)
  - Arrays and Strings
  - Linear Search

Note: the following AL topics are demonstrated in CS1, but not explicitly taught as such:

- AL-Strategies: Algorithmic Strategies
    - Brute Force (e.g., linear search)
    - Iteration (e.g., linear search)
- AL-Complexity: Complexity Analysis
    - O(1) and O(n) runtime complexities

Data Structures (CS2)

- AL-Foundational: Fundamental Data Structures and Algorithms (12 hours)
    - Abstract Data Types and Operations (ADTs)
    - Binary Search
    - Multi-dimensional Arrays
    - Linked Lists
    - Hash Tables/Maps including conflict resolution strategies
    - Stacks, Queues, and Dequeues
    - Trees: Binary, Ordered, Breadth- and Depth-first search
    - An $O(n^2)$ sort, (e.g., Selection Sort)
    - An $O(n \log n)$ sort (e.g., Quicksort, Mergesort)

- AL-Strategies: Algorithmic Strategies (3 hour)
    - Brute Force (e.g., selection sort)
    - Decrease-and-Conquer (e.g., depth/breadth tree search)
    - Divide-and-Conquer (e.g., mergesort, quicksort)
    - Recursive (e.g., depth/breadth-first tree/graph search, factorial)
    - Space vs. Time tradeoff (e.g., hashing)
- AL-Complexity: Complexity Analysis
    - Asymptotic complexity analysis
    - Empirical measurements of performance
    - Time-and-space tradeoffs (e.g., hashing)
- AL-SEP: Algorithms and Society Algorithms and Society (2 hours)

Three Course Exemplar Approach

Algorithms-C

- AL-Foundational: Fundamental Data Structures and Algorithms (3 hours)
- AL-Strategies: Algorithmic Strategies (1 hour)
    - Brute Force (e.g., traveling salesperson, knapsack)
    - Decrease-andConquer (e.g., topological sort
    - Divide-andConquer (e.g., Strassen's algorithm)
    - Dynamic Programming (e.g. Warshall's, Floyd's, Bellman-Ford)
    - Greedy (e.g., Dijkstra's, Kruskal's)
    - Heuristic (e.g., A*)
    - Transform-and-Conquer/Reduction (e.g., heapsort, trees (2-3, AVL, Red-Black))
- AL-Models: Computational Models (6 hours)
    - All CS core topics

Four Course Exemplar Approach

Algorithms-C

- All topics from Algorithms-C course plus AL-Foundational KA Core (6 hours)

Computation Theory

- AL-Complexity: Complexity Analysis (3 hours)
    - Turing Machine-based models of complexity (P, NP, and NP-C classes)
    - Space complexity (NSpace, PSpace Savitch' Theorem)
- AL-Models: Computational Models (29 hours)
    - All CS and KA Core topics

Committee

Chair: Richard Blumenthal, Regis University, Denver, Colorado, USA

Members:

- Cathy Bareiss, Bethel University, Mishawaka, Minnesota, USA
- Tom Blanchet, Hillman Companies Inc., Boulder, Colorado, USA
- Doug Lea, State University of New York at Oswego, Oswego, New York, USA
- Sara Miner More, John Hopkins University, Baltimore, Maryland, USA
- Mia Minnes, University of California San Diego, California, USA
- Atri Rudra, University at Buffalo, Buffalo, New York, USA
- Christian Servin, El Paso Community College, El Paso, Texas, USA

Appendix: Core Topics and Skill Levels

Many

KA	KU	Topic	Performance	CS/KA	Hours
AL	Foundational Complexity	2. Arrays (single & multi dimension, strings) 1a. ADT and Dictionary operations 2b i. Foundational complexity classes: Constant O(1)	Explain Apply Explain	CS	1
AL	Foundational Complexity Strategies	11a. Search O(n), (e.g., Linear search of an array) 2b iii. Foundational complexity classes: Linear O(n) 1a. Brute Force	Apply Evaluate Explain	CS	0.5
AL	Foundational Complexity Strategies	12a. Sorting O(n^2), (e.g., Selection sort of an array) 2b v. Foundational complexity classes: Quadratic O(n^2) 1a. Brute Force	Apply Evaluate Explain	CS	0.5
AL	Foundational Complexity Strategies	11b. Search O($\log_2 n$), (e.g., Binary search of an array) 2b ii. Foundational complexity classes: Logarithmic 1b ii. Decrease and Conquer	Apply Evaluate Explain		1
AL	Foundational Complexity	12b. Sorting O($n \log n$), (e.g., Quick, Merge, Tim: array) 2b iv. Foundational complexity classes: Log Linear	Apply Evaluate		1

	Strategies	1c. Divide-and-Conquer	Explain		
AL	Foundational Complexity Strategies	4. Linked Lists 1a. Dictionary Operations 11a. Search O(*n*), (e.g., Linear search of a linked list) 2b iii. Foundational complexity classes: Linear O(*n*) 1a. Brute Force	Explain Apply Apply Evaluate Explain		1
AL	Foundational Complexity	5. Stacks 1a. Dictionary Operations (push, pop) 2b iii. Foundational complexity classes: Constant O(*n*) 6. Queues and Deques	Explain Apply Explain		1
	Foundational Complexity Strategies	7. Hash tables / Maps 1a. Dictionary Operations (put, get) 7a. Collision resolution and complexity 2b iii. Foundational complexity classes: Constant O(*n*) 1f. Time vs. Space tradeoff	Explain Apply Explain Explain Explain		1 1
	Foundational" Strategies Foundational Strategies	9. Trees 1a. Dictionary operations (insert, delete) 11c. Search DFS/BFS 2b. Decrease and Conquer 9b. Balanced Trees (e.g., AVL, 2-3, Red-Black, Heaps) 1e ii. Transform and Conquer: Representation change	Explain Apply Apply Explain Apply Explain		1 2
	Foundational Foundational Strategies Foundational Strategies	8. Graphs (e.g., [un]directed, [a]cyclic, [un]connected, [un]weighted) 8a. Representation: adjacency list vs. matrix 13. Graph Algorithms 13a. Shortest Path (e.g., Dijkstra's, Floyd's) 1d. Greedy 1e iv. Dynamic Programming 13b. Minimal, spanning tree (e.g. Prim's, Kruskal's) 1d. Greedy	Explain		3
	Foundational	1. Abstract Data Types 3. Records/Structures/Tuples 10. Sets			1
AL	Strategies Strategies	1. Paradigms demonstrated in AL-Fundamentals algorithms: Brute-Force, Decrease-and-Conquer, Divide-and-Conquer, Iteration vs. Recursion, Time-Space Tradeoff, 1e. Transform-and-Conquer 1e i. Instance simplification (Find duplicates by pre-sorting) 1e iii. Problem reduction (Least-common-multiple)	Explain Explain		3 1

	Strategies	2. Handling Exponential Growth e.g., A*, Backtracking, Branch-and-Bound	Explain		1
	Strategies	1e iv. Dynamic Programming e.g., Bellman-Ford, Knapsack, Floyd-Warshall			1
AL	Complexity	1. Analysis Framework 2. Asymptotic complexity analysis Big O, Little O, Big Omega, and Big Theta	Explain	CS	1
	Complexity	2b. Foundational complexity classes demonstrated by AL-Fundamentals algorithms: Constant, Logarithmic, Linear, Log Linear, Quadratic, and Cubic			1
	Complexity	4. Tractability and Intractability Foundational Complexity Classes: Exponential $O(2^n)$ P, NP and NP-C complexity classes Reductions Problems Traveling Salesperson, Knapsack, SAT			4
	Strategies	1. Paradigms: Exhaustive brute force, Dynamic Programming			
	Complexity	2b viii. Factorial complexity classes: Factorial O(n!) All Permutations, Hamiltonian Circuit			
AL	Models	1a. Finite State Automata 2a. Regular language, grammar, and expressions	Apply Explain	CS	1
	Models	1b. Pushdown Automata 2b. Context-Free language and grammar	Apply Explain		1
	Models	1d. Turing Machine 2d. Recursively Enumerable language and grammar 1c. Linear-Bounded 2c. Context-Sensitive language and grammar	Explain		2
	Models	4. Decidability, Computability, Halting problem 5. The Church-Turing Thesis			1
	Models	6. Algorithmic Correctness Invariants (e.g., in: iteration, recursion, tree search)			1
AL	SEP	1. Social, Ethical, and Secure Algorithms 2. Algorithmic Fairness (e.g. differential privacy) 3. Accountability/Transparency 4. Responsible algorithms 5. Economic and other impacts of algorithms 6. Sustainability	Explain	CS	In SEP Hours