Computer Science Curriculum

Sundar STEM School

Prepared by: Amir Kafshdar Goharshady

April - June 2021

Foreword

This curriculum consists of two parts. The first (Normal, N) part is designed to be taught to every student of the Sundar STEM School. The second (Elite, E) part targets only the students who are particularly talented in computing (top 5-10%) or would like to enter the Informatics Olympiads, as well as those who aspire for an academic career in computing. Both curricula cover the same basic core concepts. However, the normal curriculum focuses more on applications and examples, whereas the elite curriculum gets progressively deeper and more theoretical at a much faster pace. Both curricula are meant to be taught on top of a normal high-school education.

The Main Challenge. A common issue in the current educational system in Pakistan, both at high-school level and in the university, is that it does not prepare its graduates for algorithmic problem-solving or logical (mathematical) thinking. Pakistani talent is consequently being channeled into less demanding aspects of computing, such as software development, or in the case of universities, purely empirical research.

Goals. The ultimate goal of this curriculum is to break this vicious cycle and provide our students with an early introduction and firm basis in theoretical computer science. We aim to reverse the situation and make sure that our students have a visible advantage over their peers in other countries. The specific learning outcomes are as follows:

1. Logical reasoning ability and mastery of basic set theory, propositional and first-order logic, as well as reasoning strategies such as proof by induction and contradiction
2. Basic familiarity with discrete mathematics, especially when it comes to enumerative combinatorics and graph theory
3. Familiarity with classical algorithms
4. End-to-end algorithmic problem-solving ability, including:
    a. Mathematical modeling of a real-world problem
    b. Reasoning about its complexity (e.g. NP-hardness) -- only in E
    c. Designing an algorithm to solve it
    d. Formally proving the algorithm's correctness, and
    e. Analyzing the asymptotic runtime of the algorithm
5. Algorithmic programming experience and fluency in C/C++

Assumption. We assume that the students have already passed a basic coding course (e.g. in Python) before they begin with this curriculum. Sundar STEM school provides such courses over the summer.

Structure. The elite curriculum is divided into 80 lessons. The goal is to cover 30 lessons in each of the first two years and the remaining 20 lessons in the third year. The normal curriculum is divided into 60 lessons, covering 20 lessons per year.

The third-year elite lessons are much more advanced and resemble the first few sessions of undergraduate courses. The idea is to make sure the students obtain a very strong foundation in the first two years, and then get a taste of the various modern directions in TCS in their final year. While the latter part is broad, it is not deep. It only covers the most basic ideas.

The teachers will have significant flexibility with respect to the amount of time that should be spent on each lesson (based on the students' progress). However, each lesson has clearly-defined objectives and learning outcomes and the teachers shall try their best to make sure every student achieves these outcomes.

There are two fundamental guiding principles in the way this curriculum should be taught:

1. **Emphasis on Proofs:** The students should obtain the mindset of not believing anything unless they have a formal proof for it. Every statement is assumed to be untrustworthy and every algorithm is assumed to be incorrect unless and until one can provide a detailed mathematical proof of correctness. Unless otherwise noted, in every lesson, convincing proofs of correctness should be derived in an active discussion with the students. The teacher's role is to guide these discussions. The proofs should never be spoon-fed to the students. The students are the ones who should come up with the proof, even if it takes much longer.
2. **Emphasis on Implementation:** While most of the lessons are mainly theoretical in nature, it is extremely important that the students implement every single one of the algorithms that are covered in this curriculum.

   For the elite curriculum, we only use C and C++ for implementation lessons and homeworks. These languages give the programmer a very high degree of freedom. This is precisely what we are aiming for, since we would like to cover a lot of different data structures. For the normal curriculum, any programming language is fine and it is especially recommended that the students use Python.

Operating System. All students are required to use Ubuntu for every programming task. We use gcc and g++ for compilation.

List of Lessons

(1st Year, Elite)

Key:

A	Algorithms and Complexity
C	Coding and Programming Languages
D	Discrete mathematics (excluding graph theory)
G	Graph theory
L	Logic and Set Theory

1. **D** Introduction to binary numbers
2. **L** Basic set theory
3. **C** Data-types in C/C++ and the importance of the type system
4. **C** Reading input and writing output in C/C++
5. **C** Arrays, conditionals and loops in C/C++
6. **C** Representing numbers and characters in a computer
7. **L** Relations and functions
8. **C** Functions and scopes
9. **A** Sorting (bubble sort, selection sort, insertion sort)
10. **L** Introduction to propositional logic
11. **A** Divide and conquer
12. **A** Merge sort
13. **D** Basic counting (principles of addition and multiplication)
14. **G** Introduction to graphs and digraphs
15. **L** Proof by contradiction (and contrapositive)
16. **D** Permutations and nPk

17. **G** Trees and forests
18. **AG** Breadth-first search and depth-first search
19. **L** Introduction to mathematical induction
20. **G** Connected components and strongly-connected components
21. **C** Big integers
22. **G** Eulerian and Hamiltonian circuits/trails
23. **D** Choices and nCk
24. **G** Degrees and bijections
25. **L** Quantifiers and first-order logic
26. **A** Asymptotic runtime analysis
27. **A** Introduction to greedy algorithms
28. **A** Heaps
29. **G** Tournaments
30. **A** Sorting (counting sort, bucket sort, radix sort)

List of Lessons

(2nd Year, Elite)

1. **G** Bipartite graphs
2. **C** Playing with strings
3. **D** Basic probability
4. **A** Master’s Theorem
5. **D** Permutations and choices with repetitions
6. **A** Quick sort and quick select
7. **D** Partitions
8. **DL** Principle of invariance
9. **A** Topological sorting and strongly connected components
10. **C** Structs and classes
11. **A** Dijkstra’s algorithm
12. **C** Pointers and memory management (+ linked lists)
13. **A** Introduction to dynamic programming (+knapsack)
14. **A** Floyd-Warshall and Bellman-Ford algorithms
15. **A** String matching (Rabin-Karp)
16. **A** String matching (KMP)
17. **A** Binary search trees
18. **D** The pigeonhole principle
19. **A** Vectors and their amortized runtime analysis
20. **C** Vectors and sets
21. **C** Operators
22. **C** Further data structures in the C++ standard library
23. **A** Introduction to number theoretic algorithms

24. **G** Matchings
25. **AG** Maximum bipartite matching
26. **C** Iterators and “auto”
27. **C** Macros
28. **A** Data-structures for disjoint sets
29. **A** Tries
30. **A** Hashing

List of Lessons

(3rd Year, Elite)

1. **D** The inclusion-exclusion principle
2. **G** Introduction to network flow
3. **A** Ford-Fulkerson algorithm
4. **L** Countable vs uncountable
5. **A** Memoization
6. **A** Lowest common ancestor (LCA)
7. **C** Templates
8. **A** Randomized binary search trees and treaps
9. **D** Combinatorial games (winning and losing positions and Nim)
10. **D** Combinatorial games (Sprague-Grundy)
11. **A** Balanced binary search trees (AVL)
12. **A** Segment trees
13. **A** Fenwick trees
14. **A** Incomputability and the halting problem
15. **A** P and NP
16. **A** Reductions and NP-hardness
17. **A** Introduction to randomized algorithms (2-3 examples)
18. **A** Introduction to parameterized algorithms (2-3 examples)
19. **A** Introduction to kernelization (2-3 examples)
20. **A** Introduction to approximation algorithms (2-3 examples)

List of Lessons

(1st Year, Normal)

1. **D** Introduction to binary numbers
2. **L** Basic set theory (intersection, union, equality, subsets and supersets)
3. **L** Relations (pairs and tuples, equivalence relations)
4. **L** Functions (one-to-one, onto)
5. **C** Recursion
6. **A** Bubble sort
7. **A** Insertion sort
8. **A** Binary search
9. **A** Warmer-colder
10. **A** Two-dimensional search
11. **L** Propositional logic (truth values and operators)
12. **L** Propositional logic (proof by truth tables)
13. **D** Basic counting (principles of addition and multiplication)
14. **G** Introduction to graphs and digraphs
15. **L** Proof by contradiction
16. **L** Gentle introduction to quantifiers
17. **L** Counterexamples
18. **A** Complete search (brute force)
19. **A** Returning change
20. **A** The knapsack problem

List of Lessons

(2nd Year, Normal)

1. **A** Number of paths in a grid
2. **G** Trees
3. **A** Depth-first search
4. **A** Breadth-first search
5. **A** Counting sort
6. **A** Radix sort
7. **A** Primality checking
8. **A** Palindromes
9. **D** Permutations and factorial
10. **D** nPk
11. **A** Merge sort
12. **G** Connected components and strongly-connected components
13. **D** Choices and nCk
14. **C** Playing with strings
15. **D** Introduction to mathematical induction I (introduction)
16. **D** Introduction to mathematical induction II (weak induction)
17. **D** Introduction to mathematical induction III (strong induction)
18. **D** Introduction to mathematical induction IV (problem solving)
19. **A** Quick sort
20. **A** Queues and stacks

List of Lessons

(3rd Year, Normal)

1. **D** Basic Probability
2. **AC** Singly-linked lists (Pointers via arrays)
3. **G** Tournaments
4. **A** Doubly-linked lists and the text editor problem
5. **C** Implementing binary trees
6. **A** Tree traversals
7. **D** Gentle introduction to the pigeonhole principle
8. **A** Asymptotic runtime analysis
9. **A** Chess puzzle: How to check
10. **A** Knight on a chessboard
11. **G** Bipartite graphs
12. **D** Combinatorial games (winning and losing positions)
13. **D** Nim with two heaps
14. **A** Introduction to greedy algorithms
15. **A** Median via quick select
16. **A** Topological sort
17. **A** Dijkstra's algorithm
18. **AD** Greatest common divisor
19. **G** Eulerian and Hamiltonian cycles/paths
20. **A** Tries

Details of Lessons and Learning Outcomes

Notes:

- The homeworks for each lesson will also include problems from Codeforces.com or train.usaco.org. These are not included in the homework plan below and will be chosen on a case-by-case basis. When no other homework plan is given, one should only focus on codeforces problems and have the students solve 5-10 of them until they master the topic at hand.
- We should have a meeting every week to discuss and update the upcoming lessons based on the students' progress.
- It is fine if we end up covering some courses faster/more slowly. The curriculum is flexible.
- There is a huge emphasis on coding early and coding a lot.
- The students must use Ubuntu and a standard compiler (gcc/g++).

Introduction to Binary Numbers (1N, 1E)

Covered Topics

- Representing numbers in various bases
- Integers in base 2 (binary)
- Addition, subtraction and multiplication in binary
- Fractional binary numbers (decimal point -> binary point)

Presentation Notes

The fractional part is only presented in the elite curriculum.

Learning Outcomes

- Ability to convert numbers between bases (especially decimal to binary and back)
- Ability to perform basic mathematical operations on binary numbers

Homework Plan

- Both coding and pen-and-paper homeworks are required. In the pen-and-paper homeworks, the students are asked to perform mathematical operations in binary and to convert numbers between bases. In the coding homework, the students have to write a program that converts both negative and non-negative integers between decimal and binary.

Basic set theory (1N, 1E)

Covered Topics

- An intuitive definition of the notion of “set”
- Elements
- Subsets and Supersets
- Equality of sets
- Power sets
- Set operations (intersection, union)
- Venn diagrams

Presentation Notes

- Mention that we assume the existence of a universal set U and only talk about its subsets
- Do not get into Zermelo-Fraenkel. We want to provide a very informal and intuitive introduction to sets.

Learning Outcomes

- A basic knowledge of set theory and fluency with set-theory notation
- Ability to reason using Venn diagrams
- Ability to formally prove the equality of two sets (by showing that each set is a subset of the other)

Homework Plan

- Pen-and-paper: Ask the students to prove several identities with set operations

- Coding: Writing a code that gets a set of integers and prints its power set

Relations (1N)

Covered Topics

- The product of two or more sets (pairs and tuples)
- Equality between pairs and tuples
- The notion of a relation between two sets
- The various notations used to specify a relation
- Reflexive, Symmetric and Transitive properties
- Equivalence relations and partitions

Presentation Notes

- It is useful to rely on the students' understanding of the notion of tuple in programming in order to present pairs and tuples in a set-theoretic sense.
- Do not get into the difference between ((x,y),z) and (x, (y,z)) or other such formal distinctions.

Learning Outcomes

- Ability to reason about products of sets and relations
- Fluency in identifying properties such as reflexivity, symmetry and transitivity
- A conceptual understanding of the relationship between equivalence relations and partitions

Homework Plan

- Pen-and-paper: Compute the product of given sets / Decide if given relations are reflexive, symmetric, or transitive / Convert between partitions and equivalence relations
- Coding: Write a code that gets two sets of integers as input and outputs their product. The code should handle the case where the input sets have repetitive items.

Functions (1N)

Covered Topics

- Formal definition of a function as a special type of relation
- Composition of functions
- One-to-one functions
- Onto functions
- Bijections

Presentation Notes

- Make sure to stress the fact that if f: A -> B is one-to-one then B is, in a sense, at least as large as A. Similarly, if f is onto, then A is at least as large as B. Finally, if f is a bijection, then both A and B have the same size.
- Provide examples of sets with various types of functions between them. Make sure to use both finite and infinite sets.
- Show bijections between sets of integers, positive integers and even numbers.
- For many students, this is their first encounter with infinite sets. Make sure they get enough support to form a solid intuitive understanding of these sets.

Learning Outcomes

- Ability to formally reason about which relations are functions.
- A solid intuition about how one can compare the “size” of various sets by finding functions between them.
- Ability to obtain bijections by composing other bijections.

Homework Plan

- Pen-and-paper: Ask the students to find bijections between various sets, including infinite ones. A good example is to have them find a bijection between the set of odd numbers and the set of prime numbers.

Data-types in C/C++ and the importance of the type system (1E)

Covered Topics

- Fundamental data types in C/C++
- Conversion between types
- Overflow and underflow
- The importance of static typing in C/C++
- u and ll suffixes for integers

Presentation Notes

- Emphasize the differences in typing in C/C++ and Python.
- Emphasize that the type of every variable should be known as soon as it is defined.
- Show examples of automatic and manual type casting (including by an operator and by assignment)
- Show examples of overflow and underflow
- Talk about the difference between unsigned and signed integers and their ranges
- Do not get into the details of floating point numbers.

Learning Outcomes

- Understanding the C/C++ type system
- Understanding how memory is allocated based on type
- Ability to reason about type errors and type casting
- Understanding the importance of being vigilant about overflow errors

Homework Plan

- Give the students 4-5 examples of programs with typing or overflow bugs and ask them to identify the problems.

Reading input and writing output in C/C++ (1E)

Covered Topics

- stdin, stdout and stderr
- Difference between stdout and stderr
- cin, cout and cerr
- Specifying stdout and stderr in the linux command line
- Handling files (ifstream, ofstream)
- printf and scanf
- Special characters: \n, \t

Presentation Notes

- Talk about the header file names in C vs C++ (e.g. stdio.h vs cstdio)
- Emphasize that printf/scanf and cin/cout do not mix

Learning Outcomes

- Ability to read input and write output of any type using both the C functions and the C++ streams

Homework Plan

- Coding: Ask the students to write various programs that read different types of input from different sources and write their output to other destinations. Make sure the students are tested for reading input using cin/scanf/ifstream and writing output using cout/printf/ofstream/cerr.

Recursion (1N)

Note: In the E curriculum, this course is covered under “divide and conquer”.

Covered Topics

- Recursive functions (functions that call each other)
  - In Python for the N curriculum
  - In C/C++ for the E curriculum
- More complicated recursions (e.g. those consisting of two or more functions)
- Ensuring the termination of a recursive function
- Examples of using recursion to compute recurrences, such as Fibonacci
- Basic introduction to recursion depth

Presentation Notes

- At this point, we do not really care about formal runtime analysis. So, do not talk about asymptotics. However, it is a good idea to ask the students to implement the Fibonacci sequence using recursion and see how slow their programs get.

Learning Outcomes

- Ability to implement both simple and multi-function recursive algorithms in Python (N) and C/C++ (E)
- Ability to prove the termination of a recursive function

Homework Plan

- Coding: Write a code that gets an input n and computes the n-th element of the Fibonacci sequence.
- Pen-and-paper: Ask the students to identify why their code is so slow. [Due to repeated spurious calculations]

Arrays, conditionals and loops in C/C++ (1E)

Covered Topics

- if, while, for, do ... while, switch
- break and continue
- arrays
- ?:

Presentation Notes

This is a pretty straightforward lesson, especially given that the students are already familiar with Python.

- Emphasize that the size of the array should be known at the time of definition (due to the type system and memory allocation)
- Show many examples of break/continue
- Make sure the students understand the new constructs that were absent in Python

Representing numbers and characters in a computer (1E)

Covered Topics

- The size of various types (int, long long, etc) and its inherent limitations
- Assigning binary sequences to unsigned integers
- Signed integers (one’s complement --with two zeros-- and two’s complement)
- Addition in two’s complement
- What happens when an int overflows?
- -1u/2
- ASCII and representation of characters
- Saving space by using char for storing small numbers
- Unsigned char
- Why doesn’t Python have the same limitations when it comes to integers?

Presentation Notes

The students should see two’s complement and ASCII in action by type casting between int/unsigned and char/int.

Learning Outcomes

- An understanding of how data is stored in binary format on memory
- An intuition about the problem with signed numbers and the methods to avoid it

Relations and Functions (1E)

This lesson covers everything in Relations (1N) and Functions (1N).

Bubble Sort (1N)

Covered Topics

- The bubble sort algorithm
- Proof of the algorithm’s correctness
- Implementation of the algorithm
- Counting the number of comparisons needed in the worst case

Presentation Notes

- Show the students an animation of bubble sort in action.
- Make sure the correctness proof is presented in detail and that all students have a good intuitive understanding of why the algorithm works.
- Explicitly consider the case where the array has repeated elements and make sure the students understand why the algorithm still works.
- When counting the number of comparisons, do not ignore constant factors or use the O notation. The students have not seen that yet.

Learning Outcomes

- Familiarity with the most basic sorting algorithm (bubble sort)
- First experience in coding a real algorithm that uses loops and arrays

Homework Plan

- Coding: Implement bubble sort for an array of integers

Insertion Sort (1N)

Covered Topics

- The insertion sort algorithm
- Proof of its correctness
- Implementation of the algorithm
- Counting the number of comparisons in the worst case

Presentation Notes

- See Presentation notes for “bubble sort”.
- Compute the exact number of comparisons in the worst-case and compare this with bubble sort

Learning Outcomes

- Learning another quadratic sorting algorithm
- More experience in coding basic algorithms with loops and arrays

Homework Plan

- Coding: Implement insertion sort on an array of integers
- Compare the real-world runtime of your insertion sort and bubble sort implementations. Do they correspond to your expectations?

Sorting (bubble sort, selection sort, insertion sort, 1E)

Covered Topics

- The three basic sorting algorithms (bubble, selection, and insertion)
- Proofs of their correctness
- Their implementation
- Counting the number of comparisons performed in the worst case in each algorithm

Presentation Notes

See “Bubble sort (1N)” and “Insertion Sort (1N)” above

Learning Outcomes

- Same as “Bubble sort (1N)” and “Insertion Sort (1N)”

Homework Plan

- Coding: the students must implement all three algorithms.
- Ask the students to run their codes on huge randomly-generated instances. Is one algorithm always faster? Why (not)?

Functions and Scopes (1E)

Covered Topics

- How to write functions in C/C++
- Global vs local variables
- Scopes of variables and their relationship to lifetimes
- Variables with the same name

Presentation Notes

- Emphasize the difference in memory management for global vs local variables
- Emphasize the role of scope in automated memory management
- Explain the rules when several variables have the same name but urge the students to use different names in their codes

Learning Outcomes

- Ability to write programs consisting of several functions in C/C++
- Understanding the difference between local and global variables
- Understanding the scope and lifetime of a variable
- Understanding how memory is managed automatically in C/C++

Binary Search (1N)

Covered Topics

- The linear search algorithm
- The binary search algorithm
- Its proof of correctness
- A detailed analysis of the maximum number of comparisons needed (logarithmic vs linear)
- Implementations of binary search
  - by recursion
  - by a while loop

Presentation Notes

There are several important takeaways in this lesson. Especially, this is the first time our students are seeing algorithms with vastly different runtimes for the same problem. Make sure this point is well understood by everyone. Have them run their codes of both search algorithms over large arrays, compare the runtimes, and make a graph.

Learning Outcomes

- Familiarity with the binary search algorithm and its runtime
- Ability to implement binary search using both recursion and loops

Homework Plan

- Coding: the students must implement binary search by both recursion and a while loop.

- Pen-and-paper: They should formally reason about the correctness of their code and provide proofs of correctness *for the implementation*.

Warmer-colder (1N)

Covered Topics and Presentation Notes

- In this lesson, we consider the following game: Alice chooses an integer between 0 and n. Bob should guess her number. After each guess, Alice answers with one of the following responses:
  - "bingo": This was the correct guess.
  - "first": This was Bob's first guess but it was incorrect.
  - "warm": The guess was incorrect, but it was closer to Alice's number than the previous guess.
  - "cold": The guess was incorrect, but it was farther from Alice's number than the previous guess (or had the same distance).
- The students should first write a program that gets the number n as input and plays as Alice. The program should choose an integer between 0 and n and then accept guesses and play. It should terminate only after "bingo".
- The students should find a strategy for Bob in order to figure out Alice's number as soon as possible. This strategy need not be optimal. Try to steer the discussion towards binary search and stop as soon as a strategy with O(lg n) guesses is suggested. Do not try to improve the constants!
- Have the students code their strategy.

Learning Outcomes

- Ability to write interactive programs
- Ability to design non-standard algorithms (in this case based on binary search)

Two-dimensional Search (1N)

Covered Topics and Presentation Notes

- We consider the following game: We are given an n x n grid whose points have coordinates from (0, 0) at the bottom left to (n, n) at the top right. Alice chooses a point (a, b) with integral coordinates 0 <= a, b <= n. Bob should find this point. At each turn, Bob can do one of the following:
    - Choose a horizontal line y = c for 0 <= c <= n, and ask Alice if the point is above, below or on this line. c should be an integer.
    - Choose a vertical line x = c for 0 <= c <= n, and ask Alice if the point is to the left, to the right, or on this line. c should be an integer.
- Similar to the previous lesson, the students should first write an interactive program that simulates Alice.
- The students should then find a strategy for Bob to find Alice's point within O(lg n) steps. Note that our students do not know anything about the O notation at this point. Try to steer the discussion towards binary searching and stop as soon as there is a logarithmic algorithm.
- Ask the students to implement this algorithm.

Learning Outcomes

This is a more advanced version of the previous lesson. The learning outcomes are very similar. Specifically, we are interested in making sure that the students can use binary search in unfamiliar situations.

Propositional logic (truth values and operators, 1N)

Covered Topics

- Propositions
- Assigning truth values (T, F) to propositions
- logical negation (not)
- logical conjunction (and)
- logical disjunction (or)
- logical exclusive disjunction (xor)
- logical implication (->)
- logical two-sided implication (if and only if, <->)

Presentation Notes

- Students often struggle with the difference between OR and XOR. This should be clarified by providing several examples from the real world.
- Many students struggle with P -> Q when P is false. A good way of providing them with the right intuition is to talk of P -> Q as a promise that if P holds, then Q will hold, too. The promise is of course fulfilled by default if P is false.
- Emphasize that we will always use the logical meaning of OR when talking about mathematics or algorithms

Learning Outcomes

- Basic understanding of logical operators
- Ability to talk formally and exactly when formulating mathematical sentences

Propositional logic (proof by truth tables, 1N)

Covered Topics

- Truth tables and how to form them
- Proving that a formula is a tautology using truth tables
- Proving implication using truth tables
- Proving "if and only if" statements using truth tables
- Prove the following logical identities
    - p->q == q OR ~p
    - ~(p OR q) == ~p AND ~q
    - ~(p AND q) == ~p OR ~q
    - p <-> q == (p->q) AND (q->p)

Presentation Notes

The concept being covered here (truth tables) is quite simple, but the students will need a lot of practice. Provide many examples and make sure every student contributes to filling the truth tables.

Learning Outcomes

- Proving tautologies and implications by relying on truth tables
- An intuitive understanding of basic identities in propositional logic

Homework Plan

- Pen-and-paper: The students should prove at least one tautology, one implication, and one two-sided implication using truth tables.

Introduction to propositional logic (1E)

Covered Topics

This lesson covers everything in “Propositional logic (truth values and operators, 1N)” and “Propositional logic (proof by truth tables, 1N)”. Additionally, it covers the following ideas:

- Formal proof/argument: a sequence of formulas such that each formula is either an assumption or a consequence of the ones coming before it
- Modus ponens

Presentation Notes

We cover this rather informally. For example, in the proofs, we do not require the use of modus ponens as the only rule. We also allow the students to rely on truth tables in order to add axioms.

Divide and conquer (1E)

This lessons also covers everything in “Binary Search (1N)” and “Recursion (1N)”

Covered Topics

- The basic idea behind divide-and-conquer
- Binary search and its two implementations
- Implementing divide-and-conquer algorithms using recursion
- Post office routing problem

Presentation Notes

The main idea that should be conveyed in this course is the mindset that breaking down a problem into simpler subproblems is often a good idea. The entirety of the lessons on binary search and recursion (1N) should be covered here as well. Warmer-colder (1N) and two-dimensional search (1N) can also be used as additional examples. Towards the end of this lesson (and in the homeworks), ask the students to think about applying the divide-and-conquer approach to the sorting problem. This ties nicely into the next lesson (merge sort). Make sure the students code every example.

Note: the students should definitely implement the Fibonacci by recursion example and do its related exercise. We will later build upon this in the dynamic programming lesson.

Learning Outcomes

- Familiarity with the general notion of divide-and-conquer
- Ability to code divide-and-conquer algorithms using recursion

Homework Plan

- Coding: Give the students a homework that essentially guides them to merge sort.

Merge Sort (1E)

Covered Topics

- The merge sort algorithm
- Its proof of correctness
- Implementing merge sort using recursion
- Implementing merge sort using loops (painful, but necessary)
- Analysis of the number of comparisons in merge sort
- Counting the number of inversions in an array

Presentation Notes

- Make sure the students code merge sort using both recursion and loops.
- When presenting the runtime analysis (number of comparisons), rely on a binary tree. Let the students figure out the number of vertices in a binary tree of depth k.

Learning Outcomes

- Familiarity with the most classical O(n lg n) sorting algorithm, namely merge sort
- Ability to code involved divide-and-conquer algorithms
- Ability to analyze the runtime of an algorithm based on a binary tree
- Ability to solve unseen algorithmic problems (inversions using merge sort)

Homework Plan

- Coding: Implement an algorithm that counts the number of inversions in a given array of integers by relying on merge sort

Basic Counting (principles of addition and multiplication, 1N, 1E)

Covered Topics

- Rule of sum
- Importance of disjointness in the rule of sum
- $|A \cup B| = |A| + |B| - |A \cap B|$
- Rule of product

Presentation Notes

Follow the presentations in the links below:

- https://brilliant.org/wiki/rule-of-sum-and-rule-of-product-problem-solving/
- https://www.youtube.com/watch?v=zG9Y8hnXZOc

Learning Outcomes

- Familiarity with the basic counting rules
- Ability to solve unseen problems by combining the two rules

Introduction to graphs and digraphs (1N, 1E)

Covered Topics

- What is a graph? G = (V, E)
- Directed and undirected edges
- Walks, trails and paths
- Cycles
- Degrees and neighborhood

Presentation Notes

It is useful to explain graphs as a set of cities with roads between them, where directed edges are one-way roads. However, make sure to emphasize the lack of geometric information. Otherwise, just follow the presentation in Chapter 1 of “Introduction to Graph Theory” by West. The models in the book are especially great examples.

Learning Outcomes

- This lesson is our students’ first contact with graph theory. The most important point is that they understand what can and cannot be modeled by a graph.

Proof by Contradiction (1N, 1E)

Covered Topics

- The basic idea behind proof by contradiction
  - We want to prove P -> Q
  - We assume P is true
  - We assume Q is false (~Q is true)
  - We obtain a contradiction from (P and ~Q)
- Sqrt(2) is not rational
- There are infinitely many prime numbers
- There are no integer solutions to 3 x + 9 y = 2
- For all positive integers n, if n^3 + 9 is odd, then n is even

For 1E only:

- Proof by contrapositive

Presentation Notes

- Explain that the idea behind proof by contradiction is basically this identity:
  - (P -> Q) == (P and ~Q)->false
- Ask the students to prove this identity using the tools we have already seen
- Similarly, for 1E students, show that proof by contrapositive is based on this identity:
  - P -> Q == ~Q -> ~P
- Ask them to prove this identity
- Follow this presentation for proof by contrapositive: https://www.youtube.com/watch?v=0YqZIHFmVzg

Learning Outcomes

- Familiarity and significant experience in proof by contradiction (both 1N and 1E) and proof by contrapositive (only 1E)

Gentle introduction to quantifiers (1N)

Covered Topics

- Predicates
- Existential quantification (∃)
- Universal quantification (∀)
- Negating statements with quantifiers

Presentation Notes

- Avoid nested quantifiers and definitely avoid quantifier alternation
- Use the examples in https://www.csm.ornl.gov/~sheldon/ds/sec1.6.html
- Follow a presentation similar to https://www.youtube.com/watch?v=GJpezCUMOxA

Learning Outcomes

- Understanding basic quantifiers (without alternation)
- Reasoning about quantified statements

Counterexamples (1N)

Covered Topics

- Basic idea behind counterexamples
  - Suppose that we want to refute ∀x f(x)
  - This is equivalent to proving ∃x ~f(x)
  - It is therefore enough to find a single x such that ~f(x)
  - This x is called a counterexample

Presentation Notes

- Present the examples in https://www.youtube.com/watch?v=mxcGpNji4ik
- Present at least 4-5 examples from graph theory
- Emphasize the differences in proof by counterexample and proof by contradiction and make sure the students understand that these are not the same method

Learning Outcomes

- The students should be able to disprove universally-quantified statements by providing counterexamples
- They should be able to solve unseen problems based on this technique

Homework Plan

- As homework, ask the students to disprove at least 5 different statements using a counterexample. Make sure there are at least 2 graph statements and 1 algorithmic statement in the mix.

Complete search (brute force, 1N)

Covered Topics

- Complete search (trying all the possibilities by brute force)

Presentation Notes

- Ask the students to write a simple for loop in order to estimate (roughly) how many operations per second they can manage to execute on their machine.
- Explain that they can rely on the computational power of their machine and try all the possibilities in a problem, as long as the number of possibilities is small enough and each of them can be checked fast enough.
- It is extremely important that one avoid checking too many spurious cases.
- Follow the examples on https://train.usaco.org/usacotext2?a=IhhvfuQpHFq&S=comp Drop the more involved examples.

Learning Outcomes

- Ability to perform (smart) brute-force search

Homework Plan

The first two problems of Section 1.3 of train.usaco.org

Returning Change (1N)

Covered Topics

- In this lesson, we consider the following problem: You are a cashier at a supermarket and should give back change to a customer. The change is n rupees. You can pay it back using 1, 2, 5, 10, 50 and 100 rupee coins. Your goal is to minimize the number of coins that you give out.
- Greedy algorithm for the problem
- Proof that the greedy algorithm works
- Adding an 8-rupee coin and showing (by counterexample) that the greedy algorithm does not work anymore

Presentation Notes

- Make sure the students code the algorithm
    - If a student uses a linear-time implementation, try to steer them towards a constant-time variant
- Let the students figure out the counterexample. Do not simply present it to them.

Learning Outcomes

- First interaction with greedy algorithms and proving their correctness
- Gaining more experience with counterexamples

The knapsack problem (1N)

Covered Topics

- 0-1 Knapsack and its pseudo-polynomial algorithm
- Multi-item Knapsack and its pseudo-polynomial algorithm
- Solving the general case of the “returning change” problem
- Proof of correctness of the algorithms

Presentation Notes

- If time allows, provide the following variant and ask the students to solve it:
    - Consider the 0-1 Knapsack problem, except that we have 10 “special” items. The special items have dependency and conflict relations among themselves (which are given in the input). We want to fill our knapsack and maximize the value while respecting these constraints.

Learning Outcomes

- First experience with dynamic programming (though it is not named as such)
- Ability to solve new problems based on recently-learned algorithmic techniques
- Ability to combine algorithmic ideas (in this case knapsack and brute force)

Permutations and nPk (1E)

Covered Topics

- Number of permutations of n items (and the definition of factorial)
- nPk
- Viewing nPk as number of one-to-one functions
- Circular permutations

Presentation Notes

Follow Chapter 1 of “Principles and Techniques in Combinatorics”

Learning Outcomes

- Familiarity with factorials
- Ability to apply the multiplication principle repeatedly
- Ability to solve unseen problems related to permutations
- Understanding the idea of counting everything c times, and then dividing by c

Homework Plan

- Coding: Write a program that gets n and k as input and prints all k-permutations of {1,...,n} in lexicographic ordering.
    - Variant: prints circular k-permutations
- Pen-and-paper: Exercises from “Principles and Techniques in Combinatorics”

Trees and Forests (1E)

Covered Topics

- Definition of trees
    - The three fundamental characteristics of a tree
    - Any two of the three entail the remaining property
    - There is a unique path between any pair of vertices
- Rooted trees and the notions of child, parent, sibling, etc.
- Definition and number of edges in a forest
- How to implement a tree using arrays and vectors

Presentation Notes

- It is unavoidable to talk a bit about connected components when presenting forests. This is fine. However, do not overdo it. Connected components will be covered in more detail in a future lesson.
- Present Section 2.1 and “Trees in Computer Science” from Introduction to Graph Theory by D.B. West

Learning Outcomes

- Familiarity with trees and forests
- Ability to represent trees in C/C++

Homework Plan

- Pen-and-paper: Exercises from Section 2.1 of West

Breadth-first search and depth-first search (1E)

Covered Topics

- The DFS algorithm
- Its implementation
- DFS tree and its properties (including a discussion of back-edges)
- The BFS algorithm
- Its implementation
- BFS tree and its properties
- Proof that BFS finds shortest paths
- Informal complexity analysis of both algorithms
- Implementation of DFS using a loop and using recursion
- Implementation of BFS using a queue

Presentation Notes

- Motivate the algorithms using reachability
- Talk about the shortest-paths problem before presenting BFS
- Check every student's implementations in detail

Learning Outcomes

- Familiarity with DFS and BFS
- Ability to reason about the properties of DFS/BFS trees
- Ability to code the classical BFS and DFS algorithms and use them in problem-solving

Introduction to Mathematical Induction (1E)

This is the longest lesson of 1E and will probably take two or three weeks of time. It is of utmost importance that the students understand the idea of mathematical induction, but it is even more important that they solve roughly 50-60 problems using induction.

Covered Topics

- Weak induction
    - The three components of an inductive proof
- Strong induction
- Proof by infinite descent

Presentation Notes

- Follow the presentation in Chapter 1 of “Introduction to algorithms: a creative approach” by Udi Manber
- This is a very exercise-intensive lesson. We should show as many examples of induction as time allows.
- Make sure to demonstrate the fact that all three components of an inductive proof are absolutely necessary. Do this by providing faulty proofs of nonsensical statements, in which one of the components is dropped.
- The students should see examples in enumerative combinatorics, graph theory, and proving correctness of algorithms.
- Use examples from “Problem-Solving Strategies” by Engel and “102 Combinatorial Problems” by Andreescu and Feng
- Prove the correctness of previously-taught sorting and divide-and-conquer algorithms using induction

- Emphasize that the proofs of termination we gave for recursive functions were indeed proofs by induction and formalize them further

Learning Outcomes

- Ability to use the various forms of induction in a wide variety of situations, including proofs of correctness for algorithms

Connected components and strongly-connected components (1E)

Covered Topics

- Definition of CC
- Maximum and minimum number of edges in a graph with n vertices and c connected components
- Finding CCs by BFS and DFS
- Definition of SCC
- Uniqueness of maximal SCCs in a digraph
- Finding SCCs by DFS
- The relationship between SCCs and back-edges in a DFS tree

Presentation Notes

- Make sure the students code every single algorithm

Learning Outcomes

- Familiarity with connectivity in graphs
- Ability to compute the connected components and strongly-connected components of a graph in C/C++

Homework Plan

- Coding: implement the SCC by DFS algorithm

Big Integers (1E)

Covered Topics

- Basic introduction to vectors in C++ (push_back, pop_back)
- Using an array of digits to represent an integer
- Addition, subtraction and multiplication
- Multiplication by Karatsuba’s algorithm

Presentation Notes

- We do not want to get into how vectors actually work. At this point, just present them as resizable arrays. There will be a future lesson that goes into more detail.
- This is an implementation-heavy lesson. Most of the time will be spent on helping students code and debugging their codes.
- Present Karatsuba’s algorithm only after the students have successfully implemented the naive multiplication method.

Learning Outcomes

- Ability to work with integers of any length in C/C++
- Ability to convert strings to big integers
- Further familiarity with divide-and-conquer methods

Eulerian and Hamiltonian circuits/trails (E1)

Covered Topics

- Eulerian circuits and trails
- The conditions for a connected graph to be Eulerian
- Algorithms for finding Eulerian circuits and trails
- Hamiltonian paths and cycles
- Bit operations in C/C++
- Exponential-time algorithms for finding Hamiltonian paths and cycles

Presentation Notes

- The implementation of the algorithm for Hamiltonian paths/cycles will depend on bitwise operations. These should also be taught as part of this lesson.
- This lesson is our students' first experience with an NP-hard problem. At this point, we are not going to mention the theory of NP-hardness/completeness. However, we should make sure the students understand that finding a Hamiltonian cycle is inherently harder than finding an Eulerian circuit.

Learning Outcomes

- Understanding the ideas behind Eulerian circuits and the importance of degrees in deciding whether a graph is Eulerian
- An intuitive understanding that finding Hamiltonian cycles is much harder than Eulerian circuits
- Further experience in implementing graph algorithms
- Familiarity with bit operations in C/C++

Homework Plan

- Coding: There are 4 coding tasks corresponding to finding Eulerian/Hamiltonian trails/circuits

Choices and nCk (E1)

Covered Topics

- Number of subsets of set vs number of permutations
- nCk (choosing an unordered team of k from a group of n people)
- Choosing more than one team
- Solving systems of linear equations over nonnegative integers (balls and walls)
- Pascal's identity
- Proving combinatorial identities by double counting
- The Binomial theorem

Presentation Notes

- Prove Pascal's identity both combinatorially and algebraically
- Follow "Principles and Techniques in Combinatorics"

Learning Outcomes

- Familiarity with nCk
- Ability to distinguish between ordered and unordered situations in counting
- Ability to combine nPk and nCk in counting problems
- Familiarity with Double-counting and ability to construct combinatorial proofs

Homework Plan

- Pen-and-paper: "102 problems in Combinatorics"

Degrees and Bijections (E1)

This lesson closely follows Section 1.3 of “Introduction to Graph Theory” by D.B. West. A huge emphasis is put on solving exercises and we will try to cover all of them.

Quantifiers and first-order logic (E1)

This is the elite version of “Gentle introduction to quantifiers (N1)” and “Counterexamples (N1)”. It covers everything in those lessons, as well as the following ideas:

Covered Topics

- Nested quantifiers (including those with quantifier alternations)
- The difference between ∀∃ and ∃∀
- Binding (free and bound variables)
- Logical identities (with quantifiers)
- Negating formulas with several quantifiers

Presentation Notes

- Use examples such as the ones in https://www.whitman.edu/mathematics/higher_math_online/section01.02.html
- There is a huge emphasis on the ability to translate from human language to formal quantified statements and vice versa

Learning Outcomes

- Fluency in quantifier logic
- Ability to handle nested quantifiers and quantifier alternations

Homework Plan

- Pen-and-paper: An extensive set of homeworks asking the students to:

- translate sentences from human language to quantified statements
- negating formulas with various quantifiers
- identifying free and bound variables

Asymptotic Runtime Analysis (E1)

Covered Topics

- Growth of functions (intuition)
- Irrelevance of constant factors
- The O notation and its formal definition
- o, Ω, ω, Θ and θ
- Runtime analysis of the quadratic sorting algorithms
- Runtime analysis of merge sort using a binary tree
- Runtime analysis of merge sort by writing its recurrence and applying induction
- Master theorem

Presentation Notes

- Follow the presentation in “Introduction to Algorithms” by Cormen et al (CLRS)
- Provide runtime analyses of various previously-taught algorithms as examples
- Remind the students that merge sort was so much faster than the other sorting algorithms we have seen
- There are many youtube videos on this subject, some with really nice graphics and animations. Use them when teaching this lesson
- Present the dynamic programming approach to computing Fibonacci numbers and compare the runtimes of the recursive and dp methods

Learning Outcomes

- Understanding the big O notation and its variants

- Ability to formally analyze the runtime of an algorithm
    - directly (by relying on counting methods)
    - by induction
    - by recurrence solving
    - by binary trees
    - by Master theorem

Homework Plan

- Pen-and-paper: Homeworks from CLRS

Introduction to Greedy Algorithms (E1)

This lesson covers "Returning change (N1)" as a special example.

Covered Topics

- Introduction to the idea of greedy algorithms
- Importance of proving correctness when applying greedy techniques
- Examples of cases when greedy algorithms fail (e.g. variants of returning change)
- Examples of greedy algorithms
  - Returning change
  - https://train.usaco.org/usacotext2?a=rgDjQ2qEhjb&S=greedy
  - Prim's algorithm for finding minimum spanning trees

Presentation Notes

- Follow https://train.usaco.org/usacotext2?a=rgDjQ2qEhjb&S=greedy and Chapters 16.1 and 16.2 of CLRS
- Do not present Kruskal

Learning Outcomes

- Ability to design greedy algorithms and prove/disprove their correctness and optimality
- Significant experience in coding greedy solutions
- Familiarity with minimum spanning trees and Prim's algorithm

Homework Plan

- Coding: USACO problems

- Coding: Implementing Prim’s algorithm
- Pen-and-paper: CLRS exercises

Heaps (E1)

Covered Topics

- Max-heaps and Min-heaps
- Storing a heap in an array
- Inserting new items in a heap (+correctness/runtime analysis)
- Removing items from a heap (+correctness/runtime analysis)
- Turning an array into a heap (+correctness/runtime analysis)
- Heap sort (+correctness/runtime analysis)
- Priority queues

Presentation Notes

- Sections 4.3.2 and 6.4.5 of "Introduction to algorithms: a creative approach"
- Chapter 6 of CLRS
- It is very important that the students implement both a max-heap and a min-heap

Learning Outcomes

- Understanding and implementing heaps (the first major data structure introduced to our students)

Tournaments (E1)

Covered Topics

- Definition of a tournament
- Every tournament has a Hamiltonian path (+algorithm)
- All remaining topics from Section 1.4 of “Introduction to graph theory” by West

Presentation Notes

Follow the West book.

Learning Outcomes

- Familiarity with tournaments
- More experience with mathematical induction (in showing the existence of a Hamiltonian path)
- Turning inductive arguments into (recursive) code

Sorting (counting sort, bucket sort, radix sort, E1)

Covered Topics

- Proof that comparison-based sorting needs at least Ω(n lg n) time
- Counting sort
- Bucket sort
- Radix sort
- Runtime analysis (linear)

Presentation Notes

- It is very important that the students understand the inherent limitations in linear-time sorting. Emphasize that these sorting methods are not solving the same problem as in our previous sorting algorithms.

Learning Outcomes

- Understanding that merge sort and heap sort are optimal
- Familiarity with linear-time sorting and ability to distinguish the cases where it is applicable

Homework Plan

- Coding: Implement all three linear-time sorting algorithms