What is Computer Science?

- Overview
    - Computer Science encompasses theory, systems, and applications.
    - Theory: Studies what problems computers can solve.
    - Systems: Focuses on building and managing computer software and hardware.
    - Applications: Uses computers to solve real-world problems, often beyond core CS.
- Thinking Computationally
    - Involves breaking down problems into precise, logical steps that a computer can execute.
    - Goes beyond just writing code — it's about structured problem-solving.
- Systems in Computer Science
    - Includes the design and management of operating systems (e.g., UNIX).
    - Manages memory, data transfer, and allows multi-tasking across programs.
- Programming Languages
    - Serve as a way to communicate with computers through precise syntax and semantics.
    - Examples: Python, C++, Java, Perl.
    - Unlike natural language, they are strict and formal.
- Software Engineering
    - Involves writing efficient, reliable, and maintainable code.
    - Requires thorough testing to ensure programs work correctly in all scenarios.
    - Good software shouldn't crash or hang — user experience matters.
- Hardware and Embedded Systems
    - Software must interact with physical devices (laptops, phones, robots, etc.).
    - Many everyday devices are now program-controlled.
    - Understanding how they work benefits from computer science thinking.
- Conclusion
    - Computer Science is foundational to many aspects of modern life.
    - It’s not just about programming, but about problem-solving, engineering, systems thinking, and understanding the interaction between software and hardware.

Algorithms

- Algorithm: step-by-step, precise set of instructions for solving a problem or performing a task.
- Algorithms do not require computers; they are general procedures (e.g., a cooking recipe).
- Example: A brownie recipe is an algorithm (preheat oven, melt ingredients, mix, bake, etc.)
- In computer science, algorithms are used to describe computational procedures.
- Algorithms are essential in fields like optimization, sorting, and computational biology.
- Optimization example: Finding a maximum point in a landscape.
    - You look around, find the steepest upward slope, and take a step in that direction.
    - Repeat until you reach a point where no further upward steps are possible.
    - This is known as hill climbing—it may find a local maximum, but not always the global maximum.
- Sorting example: Bubble sort algorithm
    - Task: Sort a list of numbers (e.g., 0 through 9) in ascending order.
    - Steps:
        - Place the numbers in a list.
        - Compare the first two items; if out of order, swap them.
        - Continue comparing and swapping adjacent pairs throughout the list.
        - After one pass, return to the beginning and repeat until the list is fully sorted.
    - This algorithm always works and eventually sorts the list correctly.
    - However, bubble sort is inefficient—requires many passes, especially with disordered data.
- Algorithms must be both correct (produce the right result) and efficient (run quickly for large data).
- Not all correct algorithms are efficient—better algorithms exist for tasks like sorting.
- Understanding and designing efficient algorithms is a core goal of computer science.

Memory and Data Structures

- Purpose of Data Structures

    - Enable efficient storage and retrieval of data.
    - Especially critical for large datasets (e.g. genomics data).
- Genomics Example
    - Genomic sequences come as long strings of characters (A, C, G, T).
    - Large volumes of aligned sequences need optimized storage.
    - Storing only differences between similar sequences saves memory.
- Efficient Retrieval
    - DNA sequences are long and similar-looking; searching requires efficient structures.
    - Example: Finding a 100-nucleotide sequence within a 3-billion base pair genome.
    - Store location metadata like chromosome and position for quick access.
- Common Data Structures
    - Trees, lists, linked lists, and their variants.
    - Allow navigation between related data points using references (or pointers).
- Pointers and Memory Addresses
    - Every memory block has an address (pointer).
    - Pointers enable access to specific pieces of data quickly.
    - Efficient data structures use pointers to link related elements (e.g., neighboring DNA sequences).
- Efficiency Considerations
    - Important to balance space and speed.
    - DNA is naturally represented as letters (A, C, G, T), which are text characters.
- Bytes and Bits
    - Text characters are typically stored as 1 byte (8 bits).
    - A byte can represent values from 0 to 127 using ASCII encoding.
- Optimized DNA Storage
    - Only 4 DNA letters → can be represented using 2 bits:
        - A = 00, C = 01, G = 10, T = 11
    - This provides 4x compression vs. 1-byte-per-letter storage.
    - Critical for managing gigabytes or terabytes of sequence data.

- Sequence Patterns
    - Certain DNA regions (e.g., introns) have common patterns.
    - Introns typically start with GT and end with AG.
    - Instead of storing each intron sequence: use probabilistic models to capture sequence features.
- Sequence Logos
    - Visual representation of base probabilities at specific positions.
    - Letter height indicates the likelihood of each nucleotide.
    - Captures meaningful biological patterns in a compact format.
- Benefits of Probabilistic Models
    - Condense information from thousands of sequences.
    - Useful for pattern recognition in new genomic data.

Efficiency in Algorithms

- Definition: Computational or algorithmic efficiency refers to how quickly and resource-effectively an algorithm completes a task.
- Importance
    - Central to computer science, especially when handling big data (e.g., in genomics).
    - Efficient algorithms save time, memory, and computational power.
- Mail Delivery Analogy
    - Inefficient approach: Mail truck picks up and delivers mail for one house at a time, returning to the warehouse each time.
        - Correct but slow—excessive back-and-forth wastes time and fuel.
    - Efficient approach: Truck collects mail for multiple houses in one trip, delivering sequentially without unnecessary returns.
        - Minimizes distance, trips, and delivery time.
- Traveling Salesman Problem (TSP)
    - A classic example of an efficiency-focused problem.
    - Goal: find the shortest route that visits a list of locations once and returns to the start.
    - TSP illustrates how efficiency involves route optimization and resource use.

- Look for ways to: Minimize number of operations, Reduce memory usage, Limit redundant steps (e.g., unnecessary data access or loops).
- Efficiency Analysis
    - Evaluate how an algorithm scales with increasing data sizes.
    - Optimize how frequently a computer accesses memory or performs repetitive tasks.

Software Engineering in Genomic Data Science

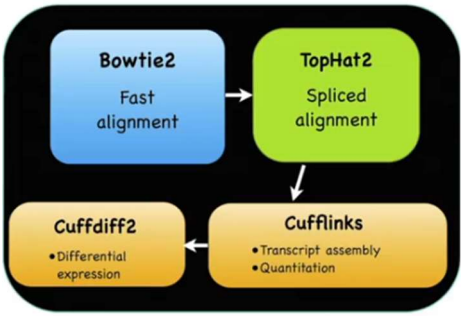
- Software engineering is about more than writing code—it ensures reliability, robustness, and correct handling of edge cases.
- Especially critical in computational biology, where we deal with large, complex datasets.
- Beyond Equations
    - Programs often include simple equations (e.g., z = x / y), but software must also handle unexpected inputs (e.g., division by zero).
    - Error handling is a key part of robust software engineering.
- Why It Matters in Genomics
    - Big datasets = more rare edge cases.
    - Even rare bugs or incorrect assumptions can lead to hundreds or thousands of false results.
- RNA Editing Example
    - Biological background: RNA editing is a rare, real phenomenon where RNA sequences differ slightly from the corresponding DNA (e.g., A→I editing).
    - Computational analysis: Researchers tried to identify new RNA editing sites by aligning DNA and RNA sequences and looking for mismatches. They used widely accepted alignment software and found thousands of novel RNA edits.
    - What went wrong
        - The alignment software had small inaccuracies—tiny error rates magnified by massive datasets.
        - Researchers misunderstood or overlooked how the software actually worked.
        - False positives were interpreted as novel biological findings, leading to controversy and confusion.
- Key Lessons

    - Understanding software internals is essential; don’t treat tools as black boxes.
    - Software that runs and gives results isn't necessarily correct.
    - Reliable outputs require reliable engineering: handling all edge cases, especially with large-scale data.
    - Even rare bugs (e.g., 1 in a million) can cause hundreds of errors in datasets with billions of entries.
- Best Practices
    - Always verify unexpected results.
    - Question software behavior, especially when outcomes are surprising.
    - Prioritize defensive programming—assume that things can go wrong and plan for it.
    - Be skeptical of results that seem too good to be true without thorough validation.

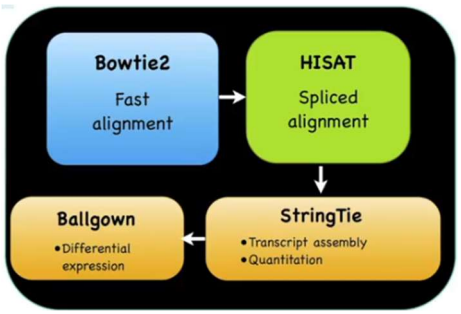
Computational Biology Software

- Purpose Software used to transform raw biological data into meaningful information to aid biological discoveries and guide experiments.
- Raw Data Input: Typically starts with large-scale genome data, such as DNA sequences (strings of A, C, G, T), which are initially uninterpretable.
- Data Processing: Software converts raw, large files into understandable formats (e.g., visualizations of DNA duplications affecting phenotypes).
- Analysis Pipelines
    - Pipelines are series of software tools that process raw data step-by-step.
    - Tasks include cleaning data, assembling sequences, comparing sequences to references, and condensing results for interpretation.
    - Hundreds to thousands of programs and pipelines exist, often integrating multiple steps.
- Visualization: Specialized software tools visualize processed data for easier interpretation by researchers.
- Discovery: Pipelines help identify differences between individual genomes and reference genomes.
- Example RNA-seq Pipeline
    - RNA-seq sequences RNA to identify gene expression in cells.

- The Tuxedo suite is a well-known RNA-seq pipeline consisting of:
  - **Bowtie**: Aligns short RNA reads to the genome.
  - **TopHat**: Handles RNA splicing by identifying reads spanning multiple exons.
  - **Cufflinks**: Assembles aligned reads into genes and estimates expression levels.
  - **Cuffdiff**: Compares gene expression between samples to identify differences.



  - Outputs tables listing gene expression changes that inform biological conclusions.
- Complexity in Gene Expression
  - Most human genes (>90%) have multiple splice variants (isoforms).
  - Software must distinguish and quantify expression of these isoforms.
- Software Evolution
  - Tools are regularly updated as sequencing technology evolves.
  - For example, newer tools replacing Tuxedo components:
    - **Bowtie2** (update to Bowtie)
    - **HISAT** (replacement engine for TopHat)
    - **StringTie** (faster and more accurate assembler than Cufflinks)
    - **Ballgown** (for differential expression analysis)



- Impact of Software Choice
  - Different programs produce varying results, even for basic tasks like read alignment.
  - Example: Bowtie2 and BWA align different subsets of reads, affecting downstream analysis.
  - Choice of software significantly affects accuracy and biological interpretations.
- Continuous Updates Required
  - Sequencing technology rapidly changes (e.g., longer reads, higher throughput).
  - Using outdated software on new data can produce misleading results.
  - Computational biologists must stay informed about new tools and technology to ensure accurate analysis.

Quiz

1. A computer algorithm is

   A description of the hardware and software capabilities of a computer

   A protocol for transmitting data over a network

   A precise specification of all the steps needed to compute a solution to a problem

   A description of the memory organization within a computer

2. You can make a program more efficient by

   Re-designing the algorithm to require fewer steps

   Recompiling it on a different computer

   Running it on the Amazon cloud

   Replacing your hard drives with faster solid state drives

3. DNA sequences can be represented efficiently using

   The SAM alignment format

   Two bits for each of the 4 possible bases

   One byte for each of the 4 possible bases

   One codon for each amino acid

4. A programming language is

   A formal language used to instruct computers what to do

   The way we describe high-level algorithms

   Anything written in Python

   A method for translating between languages such as English and French

5. Software engineering involves

Debugging computer code

Updating code so that it remains compatible with other software systems.

All of these options

Testing programs on a wide range of examples to see if they perform as expected

6. Bowtie, TopHat, and Cufflinks are

   A suite of software tools for alignment and analysis of next-generation sequence data

   A web-based system for browsing genome data

   Programs for determining the function of a gene

   Elements of men's formal evening wear

7. Sequence alignment refers to

   Lining up two DNA sequences so that positions with the same base match one another

   Making sure that all the DNA sequences in a file use the same format

   Shuffling the positions in a sequence to randomize them

   Determining the amino acid sequence produced by translating a DNA sequence

8. A software pipeline for RNA sequence (RNA-seq) analysis will

   Evaluate the best RNA-seq protocol for minimizing sequencing artifacts

   Process large raw sequence files into a summary table showing which genes were present

   Compare cases to controls and determine which genes were responsible for any differences in phenotype

   Create a database in which one can efficiently store and retrieve results

9. Which of the following is not a computer operating system?

   RedHat Linux

Google Drive

Unix

Mac OS X

10. A data set large enough to overwhelm the main memory of a computer would

Be larger than the available RAM

Contain more than 1000 genomes' worth of data

Use uncompressed files rather than compressed ones

Be at least 100 gigabytes in size