UCLA COMPUTER SCIENCE FIELD NOTES

CS C122

Algorithms in Computational Genomics

Sequence algorithms, indexes, alignment, assembly, variation, genome graphs, and scalable pipelines

How these notes are written. I wrote each chapter the way I wish the material had been explained the first time: the real idea, the point where I usually get stuck, the simplest mental model that still stays honest, and a worked decision instead of a polished answer appearing from nowhere. Every chapter closes with practice and a fully written solution. If a sentence sounds impressive but does not help me solve or explain something, it does not belong here.

Daniel Mastick

Curriculum map, working notes, practice, and solutions

Course map

Week	Core thread	What should be solid by the end
1	**Genomic strings, scale, and algorithmic formulation**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
2	**Exact matching, suffix structures, and compressed indexes**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
3	**Pairwise alignment and dynamic programming**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
4	**Read mapping and seed-and-extend search**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
5	**Genome assembly and de Bruijn graphs**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
6	**Variant calling, haplotypes, and local reconstruction**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
7	**Genome graphs, pangenomes, and structural variation**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
8	**Scalable pipelines, benchmarking, and reproducibility**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
9	**Integration workshop**	Connect at least three chapters in one end-to-end problem and explain where their contracts meet.
10	**Synthesis and project review**	Audit assumptions and explain a complete solution from interface to failure handling.

Scope anchor: https://catalog.registrar.ucla.edu/course/2025/COMSCIC122. The sequence is aligned to the official catalog description and expanded into a practical ten-week study plan.

How I would use this packet

Treat each numbered section as one chapter. Read the formal explanation, pause at the student interpretation, and see whether the easy version still says the same thing. Rework the example without looking, then cover the solution in the chapter practice box. The cumulative set at the end is deliberately mixed: knowledge becomes durable when I have to choose the tool instead of being told which chapter I am in.

Contents

CHAPTER 1 Genomic strings, scale, and algorithmic formulation

The idea

Computational genomics turns biological questions into problems over strings, reads, coordinates, graphs, and noisy observations. Input scale, alphabet, error model, memory layout, and desired output define the actual algorithmic problem.

Rebuild the chapter from first principles

The claim I need to own is this: Computational genomics turns biological questions into problems over strings, reads, coordinates, graphs, and noisy observations. Input scale, alphabet, error model, memory layout, and desired output define the actual algorithmic problem. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of genomic strings, scale, and algorithmic formulation. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach genomic strings, scale, and algorithmic formulation on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from the course foundations to exact matching, suffix structures, and compressed indexes. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Computational genomics turns biological questions into problems over strings, reads, coordinates, graphs, and noisy observations.
- **Mechanism.** Input scale, alphabet, error model, memory layout, and desired output define the actual algorithmic problem.
- **Boundary.** The useful test is whether I can apply genomic strings, scale, and algorithmic formulation to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when genomic strings, scale, and algorithmic formulation is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I do not start with a famous algorithm; I first decide what a symbol, match, and error mean in the experiment.

The easy version

Computational genomics turns biological questions into problems over strings, reads, coordinates, graphs, and noisy observations.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why is exact string equality usually

insufficient for read analysis? The conclusion is Sequencing error, mutation, indels, repeats, and strand orientation make corresponding strings differ. The formulation needs an explicit tolerance and scoring or probabilistic model.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why is exact string equality usually insufficient for read analysis?
Solution. Sequencing error, mutation, indels, repeats, and strand orientation make corresponding strings differ. The formulation needs an explicit tolerance and scoring or probabilistic model.

Practice 2. Give a short oral explanation of genomic strings, scale, and algorithmic formulation that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct genomic strings, scale, and algorithmic formulation from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a genomic strings, scale, and algorithmic formulation problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 2 Exact matching, suffix structures, and compressed indexes

The idea

Tries, suffix trees, suffix arrays, the Burrows-Wheeler transform, and FM indexes trade construction time, query time, and memory. Backward search narrows a suffix interval one query symbol at a time using rank information.

Rebuild the chapter from first principles

The claim I need to own is this: Tries, suffix trees, suffix arrays, the Burrows-Wheeler transform, and FM indexes trade construction time, query time, and memory. Backward search narrows a suffix interval one query symbol at a time using rank information. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of exact matching, suffix structures, and compressed indexes. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach exact matching, suffix structures, and compressed indexes on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from genomic strings, scale, and algorithmic formulation to pairwise alignment and dynamic programming. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Tries, suffix trees, suffix arrays, the Burrows-Wheeler transform, and FM indexes trade construction time, query time, and memory.
- **Mechanism.** Backward search narrows a suffix interval one query symbol at a time using rank information.
- **Boundary.** The useful test is whether I can apply exact matching, suffix structures, and compressed indexes to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when exact matching, suffix structures, and compressed indexes is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I track the interval invariant: every suffix in the current range matches the query suffix processed so far.

The easy version

Tries, suffix trees, suffix arrays, the Burrows-Wheeler transform, and FM indexes trade construction time, query time, and memory.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: What does an FM-index backward-search

interval represent? The conclusion is It is the contiguous suffix-array range whose prefixes equal the processed pattern suffix. Extending by one character uses cumulative counts and rank to produce the next valid range.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. What does an FM-index backward-search interval represent?
Solution. It is the contiguous suffix-array range whose prefixes equal the processed pattern suffix. Extending by one character uses cumulative counts and rank to produce the next valid range.

Practice 2. Give a short oral explanation of exact matching, suffix structures, and compressed indexes that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct exact matching, suffix structures, and compressed indexes from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a exact matching, suffix structures, and compressed indexes problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 3 Pairwise alignment and dynamic programming

The idea

Global and local alignment define states over sequence prefixes, recurrence choices for substitutions and gaps, boundary conditions, traceback, and scoring. Affine gaps use separate states for opening and extending a gap.

Rebuild the chapter from first principles

The claim I need to own is this: Global and local alignment define states over sequence prefixes, recurrence choices for substitutions and gaps, boundary conditions, traceback, and scoring. Affine gaps use separate states for opening and extending a gap. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of pairwise alignment and dynamic programming. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach pairwise alignment and dynamic programming on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from exact matching, suffix structures, and compressed indexes to read mapping and seed-and-extend search. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Global and local alignment define states over sequence prefixes, recurrence choices for substitutions and gaps, boundary conditions, traceback, and scoring.
- **Mechanism.** Affine gaps use separate states for opening and extending a gap.
- **Boundary.** The useful test is whether I can apply pairwise alignment and dynamic programming to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when pairwise alignment and dynamic programming is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I write the state sentence and boundaries before the recurrence; most alignment mistakes begin there.

The easy version

Global and local alignment define states over sequence prefixes, recurrence choices for substitutions and gaps, boundary conditions, traceback, and scoring.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why does affine-gap alignment use multiple DP matrices? The conclusion is One matrix represents alignments ending in a match

or substitution, and separate matrices represent gaps in either sequence. Transitions charge a gap-open cost once and an extension cost thereafter.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why does affine-gap alignment use multiple DP matrices?
Solution. One matrix represents alignments ending in a match or substitution, and separate matrices represent gaps in either sequence. Transitions charge a gap-open cost once and an extension cost thereafter.

Practice 2. Give a short oral explanation of pairwise alignment and dynamic programming that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct pairwise alignment and dynamic programming from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a pairwise alignment and dynamic programming problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 4 Read mapping and seed-and-extend search

The idea

A read mapper uses exact or approximate seeds to find candidate loci, then extends and scores candidates while accounting for strand, indels, repetitive matches, paired-end constraints, and mapping uncertainty.

Rebuild the chapter from first principles

The claim I need to own is this: A read mapper uses exact or approximate seeds to find candidate loci, then extends and scores candidates while accounting for strand, indels, repetitive matches, paired-end constraints, and mapping uncertainty. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of read mapping and seed-and-extend search. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach read mapping and seed-and-extend search on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from pairwise alignment and dynamic programming to genome assembly and de bruijn graphs. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A read mapper uses exact or approximate seeds to find candidate loci, then extends and scores candidates while accounting for strand, indels, repetitive matches, paired-end constraints, and mapping uncertainty.
- **Mechanism.** The useful test is whether I can apply read mapping and seed-and-extend search to a new case and explain the assumption that makes the answer valid.
- **Boundary.** The useful test is whether I can apply read mapping and seed-and-extend search to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when read mapping and seed-and-extend search is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

My shorthand for this chapter is: a reported coordinate is a hypothesis with a confidence score, not proof that the read came from one unique location.

The easy version

A read mapper uses exact or approximate seeds to find candidate loci, then extends and scores candidates while accounting for strand, indels, repetitive matches, paired-end constraints, and mapping uncertainty.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why can a longer seed improve speed but reduce sensitivity? The conclusion is Long seeds occur at fewer candidate loci, reducing extension work, but one sequencing error inside the seed can eliminate the true location. Multiple or spaced seeds balance the tradeoff.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why can a longer seed improve speed but reduce sensitivity?
Solution. Long seeds occur at fewer candidate loci, reducing extension work, but one sequencing error inside the seed can eliminate the true location. Multiple or spaced seeds balance the tradeoff.

Practice 2. Give a short oral explanation of read mapping and seed-and-extend search that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct read mapping and seed-and-extend search from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a read mapping and seed-and-extend search problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 5 Genome assembly and de Bruijn graphs

The idea

Assembly reconstructs sequence from overlapping reads. De Bruijn graphs use k-mers as vertices or edges and expose tips, bubbles, repeats, coverage variation, and Eulerian traversal questions.

Rebuild the chapter from first principles

The claim I need to own is this: Assembly reconstructs sequence from overlapping reads. De Bruijn graphs use k-mers as vertices or edges and expose tips, bubbles, repeats, coverage variation, and Eulerian traversal questions. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of genome assembly and de bruijn graphs. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach genome assembly and de bruijn graphs on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from read mapping and seed-and-extend search to variant calling, haplotypes, and local reconstruction. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Assembly reconstructs sequence from overlapping reads.
- **Mechanism.** De Bruijn graphs use k-mers as vertices or edges and expose tips, bubbles, repeats, coverage variation, and Eulerian traversal questions.
- **Boundary.** The useful test is whether I can apply genome assembly and de bruijn graphs to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when genome assembly and de bruijn graphs is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I treat k as a resolution choice: too small collapses repeats, while too large fragments coverage.

The easy version

Assembly reconstructs sequence from overlapping reads.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: How does a heterozygous variant appear in a de Bruijn graph? The conclusion is It can create a bubble: paths share flanking sequence, diverge around alternative alleles, and then rejoin. Errors can also create bubbles, so coverage and consistency matter.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. How does a heterozygous variant appear in a de Bruijn graph?
Solution. It can create a bubble: paths share flanking sequence, diverge around alternative alleles, and then rejoin. Errors can also create bubbles, so coverage and consistency matter.

Practice 2. Give a short oral explanation of genome assembly and de bruijn graphs that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct genome assembly and de bruijn graphs from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a genome assembly and de bruijn graphs problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 6 Variant calling, haplotypes, and local reconstruction

The idea

Variant calling compares read evidence with reference and alternative haplotypes while modeling base quality, mapping quality, ploidy, and local alignment. Phasing assigns alleles to chromosome copies.

Rebuild the chapter from first principles

The claim I need to own is this: Variant calling compares read evidence with reference and alternative haplotypes while modeling base quality, mapping quality, ploidy, and local alignment. Phasing assigns alleles to chromosome copies. I do not count that as learned just because the sentence looks familiar. I

should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of variant calling, haplotypes, and local reconstruction. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach variant calling, haplotypes, and local reconstruction on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from genome assembly and de bruijn graphs to genome graphs, pangenomes, and structural variation. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Variant calling compares read evidence with reference and alternative haplotypes while modeling base quality, mapping quality, ploidy, and local alignment.
- **Mechanism.** Phasing assigns alleles to chromosome copies.
- **Boundary.** The useful test is whether I can apply variant calling, haplotypes, and local reconstruction to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when variant calling, haplotypes, and local reconstruction is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I separate 'an alternative base was observed' from 'the individual carries this genotype.' The second needs an evidence model.

The easy version

Variant calling compares read evidence with reference and alternative haplotypes while modeling base quality, mapping quality, ploidy, and local alignment.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why is naive mismatch counting a weak genotype caller? The conclusion is It ignores error probabilities, mapping ambiguity, strand and position bias, allele balance, ploidy, and competing haplotypes. A likelihood-based caller weighs those factors.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why is naive mismatch counting a weak genotype caller?
Solution. It ignores error probabilities, mapping ambiguity, strand and position bias, allele balance, ploidy, and competing haplotypes. A likelihood-based caller weighs those factors.

Practice 2. Give a short oral explanation of variant calling, haplotypes, and local reconstruction

that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct variant calling, haplotypes, and local reconstruction from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a variant calling, haplotypes, and local reconstruction problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 7 Genome graphs, pangenomes, and structural variation

The idea

Linear references bias representation when populations contain diverse haplotypes and structural variants. Pangenome graphs encode alternative paths, but indexing, mapping, coordinate systems, and interpretation become more complex.

Rebuild the chapter from first principles

The claim I need to own is this: Linear references bias representation when populations contain diverse haplotypes and structural variants. Pangenome graphs encode alternative paths, but indexing, mapping, coordinate systems, and interpretation become more complex. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of genome graphs, pangenomes, and structural variation. List the known quantities, the unknown, the units or types, and any hidden constraint.

2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach genome graphs, pangenomes, and structural variation on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from variant calling, haplotypes, and local reconstruction to scalable pipelines, benchmarking, and reproducibility. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Linear references bias representation when populations contain diverse haplotypes and structural variants.
- **Mechanism.** Pangenome graphs encode alternative paths, but indexing, mapping, coordinate systems, and interpretation become more complex.
- **Boundary.** The useful test is whether I can apply genome graphs, pangenomes, and structural variation to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when genome graphs, pangenomes, and structural variation is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

My shorthand for this chapter is: a richer reference reduces one kind of bias while creating a harder naming and search problem.

The easy version

Linear references bias representation when populations contain diverse haplotypes and structural variants.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: What problem does a pangenome graph address? The conclusion is It represents multiple common sequences and structural alternatives instead of forcing every sample against one linear path, improving representation of diverse haplotypes and some variant types.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. What problem does a pangenome graph address?
Solution. It represents multiple common sequences and structural alternatives instead of forcing every sample against one linear path, improving representation of diverse haplotypes and some variant types.

Practice 2. Give a short oral explanation of genome graphs, pangenomes, and structural variation that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a

four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct genome graphs, pangenomes, and structural variation from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a genome graphs, pangenomes, and structural variation problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 8 Scalable pipelines, benchmarking, and reproducibility

The idea

Real genomic algorithms need streaming, parallelism, locality, compressed formats, provenance, deterministic parameters, workflow recovery, and careful benchmarks. Accuracy must be measured without leaking benchmark structure into development.

Rebuild the chapter from first principles

The claim I need to own is this: Real genomic algorithms need streaming, parallelism, locality, compressed formats, provenance, deterministic parameters, workflow recovery, and careful benchmarks. Accuracy must be measured without leaking benchmark structure into development. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of scalable pipelines, benchmarking, and reproducibility. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.

4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach scalable pipelines, benchmarking, and reproducibility on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from genome graphs, pangenomes, and structural variation to the cumulative course model. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Real genomic algorithms need streaming, parallelism, locality, compressed formats, provenance, deterministic parameters, workflow recovery, and careful benchmarks.
- **Mechanism.** Accuracy must be measured without leaking benchmark structure into development.
- **Boundary.** The useful test is whether I can apply scalable pipelines, benchmarking, and reproducibility to a new case and explain the assumption that makes the answer valid.

- **Decision test.** I should be able to say when scalable pipelines, benchmarking, and reproducibility is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I count wall time, memory, accuracy, and failure recovery together; a fast pipeline that silently drops samples is not scalable.

The easy version

Real genomic algorithms need streaming, parallelism, locality, compressed formats, provenance, deterministic parameters, workflow recovery, and careful benchmarks.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: How should two variant pipelines be compared? The conclusion is Use identical inputs and regions, a documented truth set, precision and recall by variant type and difficult region, resource measurements, repeated runs, error logs, and independent data not used to tune either pipeline.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. How should two variant pipelines be compared?
Solution. Use identical inputs and regions, a documented truth set, precision and recall by variant type and difficult region, resource measurements, repeated runs, error logs, and independent data not used to tune either pipeline.

Practice 2. Give a short oral explanation of scalable pipelines, benchmarking, and reproducibility that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct scalable pipelines, benchmarking, and reproducibility from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a scalable pipelines, benchmarking, and reproducibility problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

Final study check

I should be able to close this packet and reconstruct the core models, not merely recognize their names. My final check is to explain one complete problem aloud: what the inputs mean, which invariant or contract controls the work, why the method is legal, what it costs, how it can fail, and what evidence would convince another engineer.