UCLA COMPUTER SCIENCE FIELD NOTES

CS 32

Data Structures and Object-Oriented Design

Complexity, containers, trees, graphs, searching, sorting, and resource-safe C++

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	**Object-oriented design and RAII**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
2	**Asymptotic and amortized analysis**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
3	**Linked lists and iterator invariants**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
4	**Stacks, queues, and frontiers**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
5	**Trees and binary search trees**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
6	**Heaps, priority queues, and hashing**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
7	**Graphs and representations**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
8	**Sorting and binary search**	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/COMSCI32. 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 Object-oriented design and RAII

The idea

Inheritance models is-a, composition has-a, and polymorphism dispatches one interface to several implementations. RAII ties resource cleanup to object lifetime; virtual destructors protect polymorphic deletion.

Rebuild the chapter from first principles

The claim I need to own is this: Inheritance models is-a, composition has-a, and polymorphism dispatches one interface to several implementations. RAII ties resource cleanup to object lifetime; virtual destructors protect polymorphic deletion. 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 object-oriented design and raii. 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 object-oriented design and raii 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 asymptotic and amortized analysis. 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.** Inheritance models is-a, composition has-a, and polymorphism dispatches one interface to several implementations.
- **Mechanism.** RAII ties resource cleanup to object lifetime; virtual destructors protect polymorphic deletion.
- **Boundary.** The useful test is whether I can apply object-oriented design and raii to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when object-oriented design and raii is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I use composition first and inherit only when substitution is genuinely true.

The easy version

Objects own responsibilities; RAII makes cleanup happen automatically when the owner leaves scope.

Worked example

Compose a game Actor with inventory and behavior objects instead of building a hierarchy for every ability combination. The worked question below makes that reasoning concrete: Why should a polymorphic base have a virtual destructor? The conclusion is Deleting through a base pointer must invoke the complete derived destructor or derived resources may leak.

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 should a polymorphic base have a virtual destructor?
Solution. Deleting through a base pointer must invoke the complete derived destructor or derived resources may leak.

Practice 2. Give a short oral explanation of object-oriented design and raii 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 object-oriented design and raii 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 object-oriented design and raii 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 Asymptotic and amortized analysis

The idea

Big-O bounds growth above, Omega below, and Theta matches both. Worst, average, and amortized costs answer different questions. Every claim must define input size and counted operation.

Rebuild the chapter from first principles

The claim I need to own is this: Big-O bounds growth above, Omega below, and Theta matches both. Worst, average, and amortized costs answer different questions. Every claim must define input size and counted operation. 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 asymptotic and amortized analysis. 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 asymptotic and amortized analysis 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 object-oriented design and raii to linked lists and iterator invariants. 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.** Big-O bounds growth above, Omega below, and Theta matches both.
- **Mechanism.** Worst, average, and amortized costs answer different questions.
- **Boundary.** Every claim must define input size and counted operation.
- **Decision test.** I should be able to say when asymptotic and amortized analysis is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I write what n means before I write O anything.

The easy version

Complexity predicts scaling, not which program wins one small stopwatch race.

Worked example

Dynamic-array append is occasionally O(n) during resize but O(1) amortized because copied capacities form a geometric series. The worked question below makes that reasoning concrete: Analyze loops with inner bound j<i. The conclusion is The body runs 0+1+...+(n-1)=n(n-1)/2 times, so the cost is Theta(n^2).

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. Analyze loops with inner bound j<i.
Solution. The body runs 0+1+...+(n-1)=n(n-1)/2 times, so the cost is Theta(n^2).

Practice 2. Give a short oral explanation of asymptotic and amortized analysis 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 asymptotic and amortized analysis 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 asymptotic and amortized analysis 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 Linked lists and iterator invariants

The idea

Lists trade random access for cheap local insertion when a node position is known. Mutations must preserve neighbor links, sentinels, size, and iterator-validity rules.

Rebuild the chapter from first principles

The claim I need to own is this: Lists trade random access for cheap local insertion when a node position is known. Mutations must preserve neighbor links, sentinels, size, and iterator-validity rules. 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 linked lists and iterator invariants. 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 linked lists and iterator invariants 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 asymptotic and amortized analysis to stacks, queues, and frontiers. 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.** Lists trade random access for cheap local insertion when a node position is known.
- **Mechanism.** Mutations must preserve neighbor links, sentinels, size, and iterator-validity rules.
- **Boundary.** The useful test is whether I can apply linked lists and iterator invariants to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when linked lists and iterator invariants is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I sketch links before assignments and save every pointer I am about to disconnect.

The easy version

Arrays find by arithmetic; lists find by following arrows but make local surgery cheap.

Worked example

Insert x between a and b by connecting x to both before repairing a.next and b.prev. The worked question below makes that reasoning concrete: State the invariant after doubly linked deletion. The

conclusion is Every adjacent u,v satisfies u.next=v and v.prev=u; boundaries remain connected to sentinels and size decreases once.

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. State the invariant after doubly linked deletion.
Solution. Every adjacent u,v satisfies u.next=v and v.prev=u; boundaries remain connected to sentinels and size decreases once.

Practice 2. Give a short oral explanation of linked lists and iterator invariants 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 linked lists and iterator invariants 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 linked lists and iterator invariants 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 Stacks, queues, and frontiers

The idea

Stacks resume newest unfinished work; queues resume oldest. The policy shapes algorithms: DFS uses a stack and BFS uses a queue.

Rebuild the chapter from first principles

The claim I need to own is this: Stacks resume newest unfinished work; queues resume oldest. The policy shapes algorithms: DFS uses a stack and BFS uses a queue. 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 stacks, queues, and frontiers. 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 stacks, queues, and frontiers 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 linked lists and iterator invariants to trees and binary search trees. 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.** Stacks resume newest unfinished work; queues resume oldest.
- **Mechanism.** The policy shapes algorithms: DFS uses a stack and BFS uses a queue.
- **Boundary.** The useful test is whether I can apply stacks, queues, and frontiers to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when stacks, queues, and frontiers is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I ask what order pending work should resume instead of memorizing structure names.

The easy version

They are waiting-room policies: newest-first versus oldest-first.

Worked example

Balanced-parenthesis checking pushes opens and pops matching closes; BFS assigns unweighted distances layer by layer. The worked question below makes that reasoning concrete: Why is first BFS discovery a shortest path? The conclusion is The queue processes all distance d vertices before distance d+1, so a later route cannot use fewer edges.

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 first BFS discovery a shortest path?
Solution. The queue processes all distance d vertices before distance d+1, so a later route cannot use fewer edges.

Practice 2. Give a short oral explanation of stacks, queues, and frontiers 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 stacks, queues, and frontiers 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 stacks, queues, and frontiers 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 Trees and binary search trees

The idea

Trees encode hierarchy. Preorder visits parent first, postorder parent last, and inorder exposes BST sorted order. BST operations cost O(height), so unbalanced insertion can degrade to a list.

Rebuild the chapter from first principles

The claim I need to own is this: Trees encode hierarchy. Preorder visits parent first, postorder parent last, and inorder exposes BST sorted order. BST operations cost O(height), so unbalanced insertion can degrade to a list. 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 trees and binary search trees. 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 trees and binary search trees 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 stacks, queues, and frontiers to heaps, priority queues, and hashing. 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.** Trees encode hierarchy.
- **Mechanism.** Preorder visits parent first, postorder parent last, and inorder exposes BST sorted order.
- **Boundary.** BST operations cost O(height), so unbalanced insertion can degrade to a list.
- **Decision test.** I should be able to say when trees and binary search trees is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I choose traversal by asking when the parent has enough information.

The easy version

A tree is one node plus smaller trees; traversal decides when to open each nested box.

Worked example

Two-child BST deletion swaps with the inorder successor, reducing to a zero- or one-child case. The worked question below makes that reasoning concrete: What is the cost of visiting every node? The conclusion is Theta(n): each node contributes constant local work once. Balance affects root-to-leaf operations, not a full traversal.

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 is the cost of visiting every node?
Solution. Theta(n): each node contributes constant local work once. Balance affects root-to-leaf operations, not a full traversal.

Practice 2. Give a short oral explanation of trees and binary search trees 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 trees and binary search trees 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 trees and binary search trees 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 Heaps, priority queues, and hashing

The idea

A heap maintains parent-child order in a complete array-backed tree; only the root is globally guaranteed. Hash tables map keys to buckets and manage inevitable collisions through chaining or probing.

Rebuild the chapter from first principles

The claim I need to own is this: A heap maintains parent-child order in a complete array-backed tree; only the root is globally guaranteed. Hash tables map keys to buckets and manage inevitable collisions through chaining or probing. 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 heaps, priority queues, and hashing. 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 heaps, priority queues, and hashing 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 trees and binary search trees to graphs and representations. 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 heap maintains parent-child order in a complete array-backed tree; only the root is globally guaranteed.
- **Mechanism.** Hash tables map keys to buckets and manage inevitable collisions through chaining or probing.
- **Boundary.** The useful test is whether I can apply heaps, priority queues, and hashing to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when heaps, priority queues, and hashing is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I never call a heap sorted, and I expect hash collisions instead of treating them as bugs.

The easy version

A heap keeps the next priority accessible; a hash table jumps near where a key should live.

Worked example

Removing a heap root moves the last value to index zero and sifts toward the better child. The worked question below makes that reasoning concrete: Why can hashing degrade to O(n)? The conclusion is A poor distribution can place many keys in one bucket or cluster. A sound hash and controlled load factor preserve expected constant time.

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 hashing degrade to O(n)?
Solution. A poor distribution can place many keys in one bucket or cluster. A sound hash and controlled load factor preserve expected constant time.

Practice 2. Give a short oral explanation of heaps, priority queues, and hashing 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 heaps, priority queues, and hashing 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 heaps, priority queues, and hashing 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 Graphs and representations

The idea

Graphs preserve vertices and relationships; direction and weight are modeling choices. Lists use O(V+E) space and suit sparse neighbor scans; matrices use O(V^2) but test one edge in O(1).

Rebuild the chapter from first principles

The claim I need to own is this: Graphs preserve vertices and relationships; direction and weight are modeling choices. Lists use O(V+E) space and suit sparse neighbor scans; matrices use O(V^2) but test one edge in O(1). 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 graphs and representations. 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 graphs and representations 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 heaps, priority queues, and hashing to sorting and binary 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.** Graphs preserve vertices and relationships; direction and weight are modeling choices.
- **Mechanism.** Lists use O(V+E) space and suit sparse neighbor scans; matrices use O(V^2) but test one edge in O(1).
- **Boundary.** The useful test is whether I can apply graphs and representations to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when graphs and representations is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

The drawing is not the graph. I name what vertices and edges mean before choosing storage.

The easy version

Graphs keep relationships and discard coordinates.

Worked example

A prerequisite DAG can be topologically sorted; a detected cycle means the requirements are inconsistent. The worked question below makes that reasoning concrete: Compare neighbor iteration in lists and matrices. The conclusion is A list costs Theta(degree(u)); a matrix scans Theta(V). The matrix trades space for constant-time edge lookup.

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. Compare neighbor iteration in lists and matrices.
Solution. A list costs Theta(degree(u)); a matrix scans Theta(V). The matrix trades space for constant-time edge lookup.

Practice 2. Give a short oral explanation of graphs and representations 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 graphs and representations 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 graphs and representations 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 Sorting and binary search

The idea

Binary search maintains a candidate-interval invariant over a monotone predicate. Stable sorting preserves equal-key order. Comparison sorting has an Omega(n log n) worst-case lower bound, while key-structured methods use extra assumptions.

Rebuild the chapter from first principles

The claim I need to own is this: Binary search maintains a candidate-interval invariant over a monotone predicate. Stable sorting preserves equal-key order. Comparison sorting has an Omega(n log n) worst-case lower bound, while key-structured methods use extra assumptions. 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 sorting and binary 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 sorting and binary 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 graphs and representations 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.** Binary search maintains a candidate-interval invariant over a monotone predicate.
- **Mechanism.** Stable sorting preserves equal-key order.
- **Boundary.** Comparison sorting has an Omega(n log n) worst-case lower bound, while key-structured methods use extra assumptions.
- **Decision test.** I should be able to say when sorting and binary search is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I write binary search as an interval proof, not a memorized midpoint loop.

The easy version

Search repeatedly discards impossible regions; sorting invests work now to make later questions cheaper.

Worked example

Merge sort guarantees Theta(n log n) with extra storage; quicksort is in-place and usually fast but pivot-sensitive. The worked question below makes that reasoning concrete: State the lower_bound invariant. The conclusion is Below lo values are less than target; at or above hi values are at least target. Shrinking [lo,hi) until empty leaves the first legal index.

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. State the lower_bound invariant.
Solution. Below lo values are less than target; at or above hi values are at least target. Shrinking [lo,hi) until empty leaves the first legal index.

Practice 2. Give a short oral explanation of sorting and binary 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 sorting and binary 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 sorting and binary 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.

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.