COMPSCI220 Algorithms and Data Structures

SCHOOL OF COMPUTER SCIENCE · UNIVERSITY OF AUCKLAND · UNDERGRADUATE STAGE 2

Full guide + ask Sia about COMPSCI220

SIDE 1/2 DEFINITIONS · FORMULAS · DECISION RULES · COMPARISONS

EXAM REVISION SHEET

S2 2026 · v2 · 2026-10-25

0 · Exam Map

COURSE.JSON · BRIEF

15% | Tests 35% | Final 50%

COMPONENT	WEIGHT	FORMAT / ROLE
Practical Component	15%	quiz, written/programming assignments, tutorial presentations; must be passed
Tests	35%	two midterm tests; theory component
Final Examination	**50%**	comprehensive theory assessment covering the full course

Permitted materials: not confirmed for this offering in the course materials we hold — check the exam instructions on the LMS. `Source: BRIEF.md`

The pass structure has **three checks**: pass the practical component, pass the theory component formed by tests and examination, and pass overall. Treat the final as a course-wide test of moving between *representation* → *trace* → *correctness* → *growth*.

input form → trace → why correct → count + qualify

correct output alone ≠ proof and does not establish scale

Course-wide answer route: representation, operation trace, correctness condition, counted growth.

PRIORITY GRID — WHAT A COMPLETE RESPONSE CONTAINS

CLUSTER	OBSERVABLE WORK	BOUNDARY TO SAY ALOUD
Asymptotics	define n; count; exhibit c, n_0; verify inequality	upper bound need not be tight
Correctness	state invariant; show state before/after a step	examples do not prove correctness
Sorting	trace subproblems and merge; justify recurrence	check worst case, memory, stability
Trees	execute insertion/removal; show affected path	running time follows actual height
Hashing	calculate location and α; trace each collision	expected cost needs assumptions

SIX-LINE ANSWER AUDIT

1 INPUT	Have I named the representation and input-size variable?
2 STATE	Can a marker see the relevant state before the operation?
3 CONDITION	Is the comparison, invariant or ordering rule explicit?
4 UPDATE	Did I show exactly what changes and what stays fixed?
5 COST	Does the growth claim follow from counted work or height?
6 LIMIT	Did I name the assumption that prevents overclaiming?

1 · Asymptotic Analysis

C1 · FINAL THEORY

Asymptotic analysis studies how time or space grows with input size while ignoring fixed implementation constants. Name the input variable n and the counted operation before naming a class.

BIG-O DEFINITION · BOUND PROOF

$T(n) \in O(g(n)) \Leftrightarrow \exists\ c, n_0 > 0:$
$0 \le T(n) \le c \cdot g(n)$ for every $n \ge n_0$

CLAIM	WHAT MUST APPEAR	BOUNDARY
$T \in O(g)$	exhibit c and n_0; verify the inequality	eventual upper bound
Correct loop	state invariant; trace state change; connect to result	tests alone are not proof
Scales as claimed	operation count + input variable	constants abstracted away

count T(n) choose g(n) → find c, n_0 prove ≤ → state upper bound only

upper bound need not be tight

Source figure redrawn: asymptotic object → Big-O evidence → qualified growth claim.

CORRECTNESS VIA A LOOP INVARIANT

A *loop invariant* is true before and after each iteration and supports the result. Before executing, record the relevant state; for each operation show the branch or iteration, state before/after, and what output depends on it. Check a normal, boundary and invalid input, but keep the logical distinction: matching examples do not constitute proof.

TRACE FIELD	QUESTION IT ANSWERS	WHAT NOT TO OMIT
Object	What value, structure or cost is analysed?	input variable and representation
Control	Which branch, iteration or call follows?	the condition selecting it
Before / after	What state transition occurred?	unchanged state that preserves the rule
Invariant	What proposition survives every iteration?	connection from preserved fact to result
Operation count	How does work grow with n?	the operation actually being counted

SAFE CONCLUSION LANGUAGE

Bound: "For the exhibited positive constants, the inequality holds from n_0 onward, so T(n) has the stated eventual upper bound." **Correctness:** "The stated proposition is preserved by the iteration and supports the required result." These are different claims; proving either one does not silently prove the other.

2 · Sorting + Recurrences

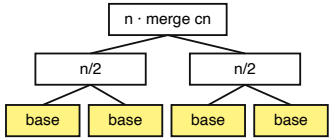
C2 · TRACE + COST

A *comparison sort* bases ordering decisions on comparisons between keys. Trace the subproblems and merge, then justify the cost. Keep correctness/order properties separate from time and memory claims.

MERGE-SORT RECURRENCE · COST RESPONSE

$T(n) = 2T(n/2) + c \cdot n \in \theta(n \log n)$

TERM / PROPERTY	MEANING IN THE ANSWER
2T(n/2)	two half-size recursive subproblems
c·n	linear merge work; not another recursive call
Θ(log n) levels	repeated halving to base cases
Θ(n) per level	linear work across a complete level
stability	equal keys retain their relative order



Source recurrence figure redrawn: logarithmically many levels, linear work across each level.

DECISION RULE

If the prompt asks for a trace, show subproblem states and the merge result. If it asks for cost, identify both recursive and non-recursive terms, then connect levels × work per level. If it asks whether the sort is suitable, check **worst-case input, memory cost and required stability**; asymptotic or average performance does not erase them.

LINE OF WORKING	WHAT TO WRITE	SELF-CHECK
Input	the sequence and comparison key	are equal keys distinguishable as records?
Divide	each half-size subproblem	does the trace match 2T(n/2)?
Combine	linear merge work across the level	is cn kept outside the recursive term?
Levels	repeated halving gives logarithmic depth	is work per whole level linear?
Order property	relative order of equal-key records	is the required stability preserved?
Conclusion	Θ(n log n) for this relationship	have worst case and memory been ignored?

Stability test: attach identity to records sharing a key, then inspect their relative order after the merge. A sorted sequence can still fail a stability requirement. Keep the stability observation alongside, not inside, the recurrence calculation.

3 · Heaps + Search Trees

C3 · OPERATION TRACE

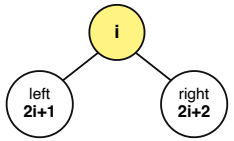
A *heap* is a complete-tree structure with parent–child ordering for extreme-priority operations. A *binary search tree* orders every node against keys in its left and right subtrees. These structures answer different questions: a heap exposes an extreme; a search tree organises ordered search.

ZERO-BASED ARRAY HEAP · INDEX TRACE

`left(i) = 2i + 1` `right(i) = 2i + 2`

`candidate index only` → verify index `< array length`

STRUCTURE	SHAPE / ORDER EVIDENCE	COST CONTROL
Heap	complete-tree shape + parent–child order	path through its height
BST	left/right subtree key relation	actual tree height
Degenerate BST	insertion order stretches the tree	do not claim logarithmic height



Source heap figure redrawn: formulas locate candidates; the array bound decides whether nodes exist.

TRACE INSERTION OR REMOVAL

1. State the representation and ordering rule.
2. Show the affected index/node before and after each move.
3. Stop only when the required relation is restored.
4. Relate the number of moves to structural height.

OPERATION EVIDENCE	HEAP	BST
Starting state	array length, index, parent–child order	root and left/right key relation
Next move	compare only existing candidate children	follow comparison into one subtree
Changed state	show each moved entry and index	show link/node affected by insert/remove
Stop rule	required parent–child relation restored	target found or insertion position reached



Tree height is the mechanism: logarithmic-operation claims require logarithmic height.

4 · Hashing + Collisions

C4 · TABLE-STATE TRACE

A *hash function* deterministically maps a key to an integer used to choose a table location. A collision means multiple keys map to the same location; a *collision strategy* must determine how insertion and retrieval proceed.

LOAD FACTOR · OCCUPANCY RESPONSE

$\alpha = n / m$
n = stored entries · m = table capacity

STAGE	RECORD EXPLICITLY	REASON
Hash	key → computed location	shows deterministic placement
Collision	occupied location + competing keys	activates the stated strategy
Resolve	every location examined or link followed	makes expected access cost traceable
Load	stored entries / capacity	connects occupancy to collision pressure

key → hash location → collision trace → place / find

α describes occupancy; it is not a probe count

Source hash-table figure redrawn: location, collision and resolution must be shown as one trace.

WHAT THE CONCLUSION MAY SAY

Expected collision and lookup behaviour depends on the collision method and hash-quality assumptions. Constant expected lookup is conditional: it does not mean every operation is constant in the worst case. Chaining and open addressing deteriorate differently, so do not interpret α without the stated collision method.

WRONG SHORTCUT	REPAIR
α = occupied buckets / entries	α = stored entries / capacity
"hashing is O(1)"	state expected claim + hashing/resizing assumptions
show final table only	show collision path and resulting state

TRACE LEDGER — ONE ROW PER INSERTED KEY

COLUMN	RECORD	WHY IT MATTERS
Key	the current key only	preserves insertion order
Hash result	initial table location	separates mapping from resolution
Observed state	empty, matching key, or collision	shows why the next rule fires
Resolution path	every slot examined or link followed	makes access work visible
Resulting table	state after this insertion	becomes the next row's input

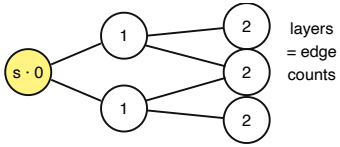
5 · Traversal + Components

C5 · REACHABILITY TRACE

A graph representation records vertices and edges, for example with adjacency lists or a matrix. *Breadth-first search* explores vertices in nondecreasing *unweighted* distance from a start vertex.

FIRST BFS DISCOVERY · DISTANCE TRACE

`d(v) = d(u) + 1 when v is first discovered from u`
`record predecessor u · enqueue v once`



Source BFS figure redrawn: first discovery fixes the shortest unweighted edge count.

NEED	DO	QUALIFICATION
Traversal	show queue/discovery order from given adjacency order	order can change with adjacency order
Distance	assign predecessor distance + 1 on first discovery	unweighted edge count
Component	collect all vertices reachable under direction rules	maximal reachable set

Connected component: a maximal set of vertices mutually reachable by paths under the graph's direction rules. A different adjacency order may change the discovery tree or predecessor, but not the shortest unweighted edge counts.

6 · Weighted Optimisation

C6 · COMPARE OBJECTIVES

RELAXATION · SHORTEST-PATH TRACE

`d(v) ← min{d(v), d(u)+w(u,v)}`
`update predecessor only when the candidate is cheaper`

shortest-path tree
paths from a source minimise path weight

minimum spanning tree
connect every vertex minimise total weight

different objectives are not interchangeable

Source weighted-graph figure redrawn: shortest-path and spanning-tree objectives differ.

OBJECT	DEFINITION / CONDITION
Shortest path	minimum total edge weight among paths allowed by the model and algorithm assumptions
MST	minimum-total-weight acyclic set connecting every vertex in a connected undirected graph
Dijkstra condition	non-negative edge weights for its finality claim

COMPSCI220 Algorithms and Data Structures · Self-Test Drills

ANSWER PATH · JUDGEMENT CARDS · CLASSIC TRAPS · EXAM-MORNING PLAN

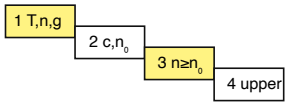
Full guide + ask Sia about COMPSCI220

SIDE 2/2 · 5 WORKED DRILLS · IF X → DO Y · TRAPS · FAST RECALL · EXAM REVISION SHEET · S2 2026 · answer before revealing the path

7 · Drill 1 – Big-O Proof

C1 · 5-MARK METHOD

Prompt: Use the Big-O definition while deriving an asymptotic bound from an operation count. Define quantities, transform, check and qualify.



Answer ladder: definition → constants → inequality → qualified conclusion.

1. Write `T(n) ∈ O(g(n))` and define the input variable and counted work.
2. Exhibit positive `c, n₀`; show `0 ≤ T(n) ≤ c·g(n)` for every `n ≥ n₀`.
3. Check the claimed inequality or a limiting case independently.
4. Conclude only an eventual upper bound. Do not claim equality or tightness without evidence.

ANSWER LINE

```
Therefore T(n) ∈ O(g(n)) for the displayed c,n₀; this establishes an eventual upper bound, not necessarily the tightest class.
```

MARK-BEARING MOVE	SELF-CHECK
relationship + symbols	is every quantity defined?
substitution / transformation	are sign and comparison visible?
constraint	does the claim stay inside the definition?

CLOSED-LOOP SELF-MARK

DEFINE	Cover the notes and say what T, n, g, c and n_0 mean.
TRANSFORM	Point to the exact inequality that establishes the bound.
CHECK	Test the stated range $n \geq n_0$, not only a convenient small input.
INTERPRET	Say "eventual upper bound"; avoid equality or automatic tightness.

Repair drill: replace "the algorithm is efficient" with the counted operation, input variable, inequality and qualified bound.

8 · Drill 2 – Merge Cost

C2 · 5-MARK METHOD

Θ(log n) × Θ(n) = Θ(n log n)

check stability + memory

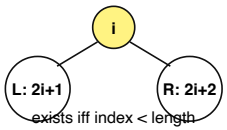
Worked recurrence: height of the recursion × work across each level.

9 · Drill 3 – Heap Indices

C3 · 5-MARK METHOD

Prompt: Explain zero-based child lookup while executing a heap operation.

`left(i)=2i+1 · right(i)=2i+2`



Formula gives candidate positions; a bounds check determines existence.

1. Name zero-based representation.
2. Substitute the current index `i` into both formulas.
3. Compare each candidate with array length.
4. Use only existing children in the ordering comparison.
5. Trace each move until the heap relation is restored; relate moves to height.

CLAIM	ALLOWED ONLY WHEN
child exists	candidate index < length
operation follows a short path	the complete-tree height controls moves
BST operation is logarithmic	BST height remains logarithmic

HEAP TRACE LEDGER

STEP	WRITE BESIDE THE ARRAY/TREE
Locate	current index, candidate child indices, array length
Validate	which candidates are inside the array
Compare	the parent–child ordering test actually used
Move	entries and indices before/after the change
Stop	the restored heap relation
Cost	number of moves as a path through tree height

BST contrast: use key comparisons and links; the path length is the actual height. If insertion order degenerates the tree, qualify the cost.

TRAP *A formula can point outside the stored array. Computing an index does not create a node.*

10 · Hashing Quick Test

C4 · CHANGED SCENARIO

Before looking back, write: `α = ____ / ____`. Then name three dependencies behind an expected constant-lookup claim.

α = n/m | trace rule | E[T] cost

not worst-case O(1)

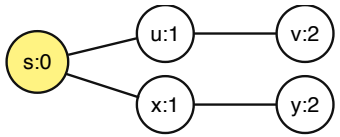
Hash response: occupancy calculation, collision trace, conditional performance claim.

11 · Drill 4 – BFS Discovery

C5 · 5-MARK METHOD

Prompt: Trace a traversal, record discovery structure and identify reachable components.

```
On first discovery of v from u:
d(v)=d(u)+1 · pred(v)=u · enqueue v once
```



queue order: nondecreasing distance

adjacency order may change predecessor

BFS drill: label distance and predecessor at first discovery, not at later encounters.

1. State the representation and start vertex.
2. Initialise start distance, then process neighbours in the given adjacency order.
3. On first discovery only, set distance/predecessor and enqueue once.
4. The discovered set gives the reachable component under the graph's direction rules.
5. Qualify: the stored tree/order may vary with adjacency order; shortest unweighted edge counts do not.

IF THE PROMPT SAYS…	RECORD…
trace BFS	queue, discovery order, distance, predecessor
component	maximal reachable set under direction rules
shortest route with weights	do not reuse the unweighted BFS distance rule

FIVE-COLUMN TRACE YOU CAN SELF-MARK

DEQUEUED	ADJACENCY EXAMINED	NEWLY DISCOVERED	DISTANCE	PREDECESSOR
current vertex	given order	first encounters only	parent + 1	discovering vertex
next queued	continue in order	never enqueue twice	unchanged later	unchanged later

Completion test: when the queue is empty, the discovered set is exactly what the traversal reached from the start under the direction rules. To identify all components, the same traversal logic must restart from an undiscovered vertex until none remain.

Order qualification: if adjacency order changes, write the new queue and predecessor choices; do not change the shortest unweighted distances. This distinction is the quickest check on whether a trace confuses representation order with graph distance.

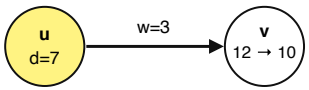
FAST-RECALL TERMS

REPRESENTATION	structure recording vertices and edges, such as adjacency lists or a matrix
BFS	visits vertices in nondecreasing unweighted distance
COMPONENT	maximal mutually reachable set under direction rules
DISCOVERY TREE	stored predecessor structure; may vary with adjacency order

12 · Drill 5 – Relaxation

COURSE WORKED EXAMPLE · 4 MARKS

Given: `d(u)=7`, `d(v)=12`, and `w(u,v)=3`. Apply one relaxation and explain the condition.



candidate 7+3=10 < 12

therefore update

The guide's worked values carried through: candidate, comparison, update, qualification.

1. **1:** candidate through u = 7+3=10.
2. **1:** compare candidate 10 with current 12.
3. **1:** set `d(v)=10` and predecessor of v to u because the candidate is smaller.
4. **1:** a non-improving candidate changes neither record; algorithm assumptions on edge weights still matter.

FULL CONCLUSION

```
Relaxation changes d(v) from 12 to 10 and records u. This is a tentative improvement; the step alone does not prove 10 globally final.
```

SHORTEST-PATH TREE OR MST?

QUESTION	TARGET	DO NOT CLAIM
cheapest route from source	shortest-path distances/tree	minimum total connecting tree
connect all vertices cheaply	MST of connected undirected graph	every pairwise path is shortest

candidate improves? | update tentative d | algorithm permits final?

local relaxation ≠ global finality

Finality is supplied by the surrounding algorithm, not by the relaxation inequality alone.

RELAXATION DECISION CARD

COMPARISON	DISTANCE	PREDECESSOR	CLAIM
d(u)+w(u,v) < d(v)	replace with candidate	set to u	tentative improvement
d(u)+w(u,v) is not smaller	unchanged	unchanged	no improvement through this edge

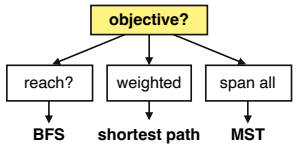
Before executing the arithmetic, define whether the graph is directed or undirected as the representation requires and identify the edge being examined. After the arithmetic, keep the local update separate from the algorithm's later proof of finality.

Objective check: shortest-path work accumulates edge weights along routes from a source. MST work selects an acyclic connecting set of minimum total weight in a connected undirected graph. Similar-looking trees do not make the objectives equivalent.

13 · Method Chooser

ALL CHAPTERS · IF X → Y

IF YOU SEE…	DO THIS	MUST QUALIFY
growth / scaling	count operations; name n; prove a bound with c,n_0	upper bound may not be tight
why loop works	state invariant; trace state before/after iteration	tests are not proof
divide-and-conquer cost	identify subcalls + combine work + levels	worst case, memory, stability
heap index	use 2i+1, 2i+2; bounds-check	candidate ≠ existing child
BST operation	trace comparisons down the actual tree	cost follows height
hash collision	show computed slot and complete resolution path	expected cost assumptions
unweighted reachability	BFS queue + first-discovery distance/pred	adjacency-order effect
weighted distance	relax only an improving candidate	edge-weight / algorithm conditions
connect all vertices cheaply	identify MST objective	connected, undirected; not all-pairs paths



Judgement card: read the objective before selecting a graph method.

UNIVERSAL TRACE FRAME

USE FOR EVERY ALGORITHM RESPONSE

```
INPUT / representation → STATE before → operation + condition → STATE after → invariant / objective → cost from counted work → assumption / limit
```

This frame separates **what changed** from **why the change is valid** and from **how much work it costs**. When stuck, write the current state and the condition controlling the next legal move.

OUTPUT ANATOMY — TURN WORKING INTO A CONCLUSION

SENTENCE JOB	REQUIRED CONTENT
Define	precise course term and represented object
Select	why this method matches the requested objective

14 · Classic Traps

FAQ LIMITS · SELF-MARK

TEMPTING ANSWER	WHY IT FAILS	REPAIR SENTENCE
"It passed the examples, so it is correct."	examples are not proof	give the invariant and state transition
"O(g) means equal to g."	Big-O is an eventual upper bound	exhibit c,n_0; avoid equality/tightness claim
"Merge contributes a third recursive call."	merge is additive linear work	2T(n/2)+cn
"Every BST operation is logarithmic."	insertion order may degenerate height	condition the cost on actual height
"2i+1 is always a child."	index may be out of bounds	check against array length
"α counts occupied buckets."	wrong numerator/denominator	stored entries / capacity
"Hash lookup is always constant."	expected claim needs assumptions	name hash, collision and resizing conditions

correct? invariant | exists? bounds | final? algorithm | fast? limits

qualify the conclusion, not just the calculation

Four recurring checks: proof, existence, finality and performance assumptions.

FAST-RECALL TERMS

INVARIANT	true before and after each iteration; supports correctness
STABILITY	equal-key records retain relative order
TREE HEIGHT	maximum edge count on a downward root-to-leaf path
COMPONENT	maximal mutually reachable set under direction rules
SHORTEST PATH	minimum allowed path weight under method assumptions
MST	minimum-total-weight acyclic connector of all vertices

15 · Exam-Morning Plan

BRIEF · FINAL REVIEW

retrieve formulas → trace one new case → say each boundary → LMS rules

recall → apply → qualify → verify operations

Final review control: retrieval and changed-case practice, then operational verification.