COMPSCI130 Introduction to Software Fundamentals

SCHOOL OF COMPUTER SCIENCE · THE UNIVERSITY OF AUCKLAND · EXAM REVISION SHEET · SIDE 1 OF 2

Full guide + ask Sia about COMPSCI130

SIDE 1/2 DEFINITIONS → INVARIANTS → OPERATION COST → TRACE → BOUNDARY TEST

S2 2026 · 50% EXAMINATION

Methods and rules · check each precondition

0 · Exam Map

COURSE.JSON · BRIEF

Labs + Assign. 30% | Test 20% | Examination 50%

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

COMPONENT	FORMAT / TARGET
Labs + Assignments	Individual practical coursework
Test	Individual CodeRunner; programming
Examination	Individual CodeRunner; data-structure concepts + programming

★ The final rewards a repeated method: **name the required operations → state the invariant → choose a representation → trace state → justify cost → test normal, boundary and failure paths.**

1 · Program State

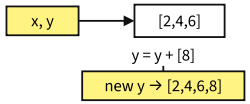
C1 · FINAL TRACING

Program state = current values plus control position. *Assignment* binds a name/location to a value; it is not mathematical equality. *Control flow* is the executed order of statements, branches, loops and calls.

STATE TRANSITION

$s_{t+1} = F(s_t, x_t)$

Executed statement + current input transform current state.



Alias trace: mutation changes the shared object; concatenation creates a new list and rebinds y.

LINE KIND	TRACE QUESTION
Assignment	Which name now refers to which object?
Mutation	Which aliases observe the changed object?
Branch / loop	Which predicate values select the path?
Call	What frame, arguments and return value exist?

CHECK → *A trace is valid only for stated inputs, types and evaluation order; never generalise past an untested branch.*

2 · Modules, Tests, Exceptions

C2 · CODE QUALITY

Module: named unit grouping related definitions behind an interface. *Test case*: input + expected behaviour + observation, aimed at one rule.

Exception: runtime event interrupting ordinary control flow; propagate or handle explicitly.

OBSERVED PASS RATE

$r = n_{pass} / n_{run}$ · says nothing about untested behaviour or test quality.

3 · Complexity: Count First

C3 · EXPLAIN EFFICIENCY

Input size n must be defined. *Time complexity* tracks growth of relevant operations. *Big O* is an asymptotic upper bound.

UPPER-BOUND RULE

$T(n) \in O(g(n))$ when constants c, n_0 exist such that $T(n) \le c \cdot g(n)$ for $n \ge n_0$.

exact count → bound → compare

define n → count ops → growth term

keep preconditions and scale visible

Source redraw: input size leads through operation count to a defensible complexity class.

CODE SHAPE	COUNT / RECURRENCE	GROWTH
Fixed work	c	O(1)
One full pass	an+b	O(n)
Repeated halving	T(n)=T(n/2)+c	O(log n)
Two independent full loops	n+n	O(n)
Nested full loops	n·n	$O(n^2)$
Triangular nesting	$\sum_{i=0}^{n-1} i = n(n-1)/2$	$\Theta(n^2)$
One-step recursion	T(n)=T(n−1)+c	O(n)

HOW TO JUSTIFY, NOT GUESS

1. Choose the operation being counted.
2. Write each loop's actual iteration count; sequential costs add, nested costs multiply only when independent.
3. Solve the sum/recurrence; then discard constants and lower terms for the asymptotic class.
4. Name worst/average/best case if the path varies.

LIMIT → *Big O suppresses constants and lower-order terms: it does not predict exact runtime, hardware effects or every small input.*

4 · Search/Sort Trade-off

C4 · CHOOSE BY WORKLOAD

BINARY-SEARCH RECURRENCE

$T(n)=T(n/2)+c$ · each comparison discards about half the ordered interval.

REPRESENTATION	INSERT	MEMBERSHIP	USE WHEN
Unsorted array	O(1) append	O(n)	few queries, many appends
Sorted array	O(n) movement	O(log n)	read-heavy, order maintained

Binary search needs ordered data and random access. Its search cost excludes creating/maintaining order. Fix one interval convention: inclusive bounds use `while low ≤ high`; failure is `low > high`.

5 · ADT Before Class

C5 · DESIGN QUESTIONS

Abstract data type defines values and permitted operations independently of implementation. *Abstraction* separates essential behaviour from representation. A Python *class* stores object state and supplies methods, but is only an implementation mechanism.

OPERATION CONTRACT

$S' = op(S,x)$, $I(S) \Rightarrow I(S')$

Every public operation, including rejection, must preserve invariant I.

client sees: values + operations + errors

contract preserves invariant I

hidden representation

Source redraw: define the ADT contract before selecting class storage.

CONTRACT ITEM	QUESTION TO ANSWER
Domain	Which values/states are valid?
Operations	Inputs, outputs and state changes?
Failure	Which exception; is state unchanged?
Invariant	What must be true before/after every call?
Ownership	Can callers alias internal storage?

Bounded stack example: owned storage contains exactly the logical items in order; $0 \le len(items) \le capacity$; positive immutable capacity; no public alias. Overflow checks occur *before* mutation.

TRAP → *A class does not itself prove useful abstraction, encapsulation or invariant preservation.*

6 · Stack vs Queue

C6 · CHOOSE REMOVAL RULE

STACK · LIFO: A · B · C ↑ top → pop → C

QUEUE · FIFO: front A · B · C rear → dequeue → A

Removal discipline decides the ADT: latest item for stack; earliest item for queue.

NEED	CHOOSE	INVARIANT / WARNING
Undo / reverse arrival	Stack	Most recent item at top
Arrival order	Queue	Oldest item at front
Smallest key first	Min-priority queue	Minimum pending key removed
Largest key first	Max-priority queue	Maximum pending key removed

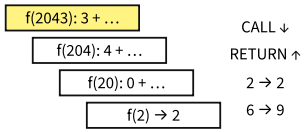
SUCCESSFUL SIZE UPDATE

$|S'| = |S|+1$ for insert; $|S'| = |S|-1$ for removal, with ordering discipline preserved.

7 · Recursion: Proof + Trace

C7 · CALLS AND RETURNS

Base case solves a terminating input directly. *Recursive case* reduces the instance and combines returned results. Correct small outputs are insufficient: prove progress.



Source redraw: expand one frame per call; combine values only while unwinding returns.

PROOF OBLIGATION	WHAT TO WRITE
Domain	Inputs for which the definition applies
Base	Condition and direct return
Progress	Recursive argument is smaller
Lower bound	Measure cannot decrease forever
Combine	How returned value solves caller

ONE-STEP RECURRENCE

$T(n)=T(n-1)+c$ → linear work, provided one smaller call and constant local work.

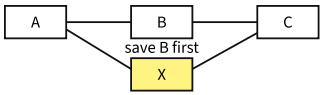
Digit-sum termination: for integers $n \ge 10$, $0 \le n//10 < n$; non-negative n strictly decreases until a single digit.

TRACE → *One row per frame. Do not add local work until the deeper call has returned to that frame.*

8 · Linked Lists

C8 · POINTER SAFETY

Node stores an element and link(s). The *link invariant* keeps the structure connected and well formed. Relinking may be O(1) *after* the target is known; finding it may require O(n) traversal.



Source redraw: set X.next to the saved successor before overwriting A.next.

OPERATION	SAFE POINTER ORDER
Insert X after A	`old=A.next; X.next=old; A.next=X`
Delete B after X	`target=X.next; X.next=target.next; target.next=None`

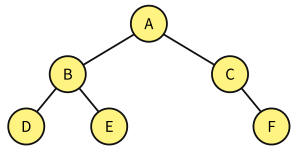
TRAVERSAL

$T(n)=T(n-1)+c$: following one next link per node is linear.

9 · Trees + Traversal

C9 · OUTPUT ORDER

A *tree* is hierarchical and acyclic from a root; a *subtree* is one node plus all descendants. Choose traversal by *when the root is visited*.



Worked tree: A root; B has D,E; C has right child F.

TRAVERSAL	RULE	RESULT ABOVE
Preorder	root-left-right	A B D E C F
Inorder	left-root-right	D B E A C F
Postorder	left-right-root	D E B F C A
Level order	breadth by depth	A B C D E F

BINARY-TREE WIDTH BOUND

$w_d \le 2^d$; sparse trees may contain fewer nodes at depth d.

Inorder is sorted only if the tree is also a BST. A traversal order does not itself provide search efficiency, balancing or key ordering.

10 · Binary Search Trees

C10 · PRESERVE ORDER

Invariant: for every node, smaller keys occupy the designated left subtree and larger keys the right subtree (under the chosen duplicate policy). Search follows one branch per comparison.

0 children: detach leaf

1 child: splice child

2 children: successor → right-subtree min

BST deletion: classify structural case before changing links.

OPERATION	METHOD	COST
Search	compare, take one child	O(h)
Insert	search to empty link	O(h)
Delete leaf	detach	O(h)
Delete one child	splice child upward	O(h)
Delete two children	copy successor; delete its old node	O(h)

HEIGHT-DEPENDENT SEARCH

$T(h)=T(h-1)+c \Rightarrow O(h)$. Balanced height is logarithmic; a highly unbalanced insertion order can make h linear.

INVARIANT CHECK → *Inorder traversal after an update must retain strict key order and contain exactly the intended keys.*

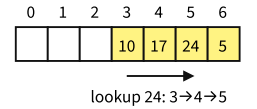
11 · Hashing + Collisions

C11 · TRACE PROBES

A *hash function* deterministically maps a key to an integer candidate position. A *collision* means distinct keys share a candidate; correctness therefore requires an explicit resolution strategy.

LOAD FACTOR

$\alpha = n/m$ · entries n divided by capacity m; interpret collision pressure with distribution, resizing and strategy.



Worked linear probing with h(k)=k mod 7 after inserting 10,17,24,5.

KEY	HOME	PROBE SEQUENCE	FINAL
10	3	3	3
17	3	3,4	4
24	3	3,4,5	5
5	5	5,6	6

Without deletion, unsuccessful lookup stops at the first *never-used* slot, not at the first different key. Average constant-time claims depend on key distribution, resizing and collisions; worst case is not guaranteed constant.

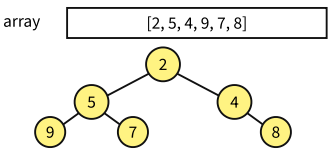
12 · Priority Queue + Heap

C12 · RESTORE INVARIANTS

Priority queue removes by priority, not arrival. A binary *heap* is complete (shape) and obeys parent-child order; it is *not* a fully sorted sequence.

ZERO-INDEXED HEAP

parent(i)=⌊(i-1)/2⌋ for every non-root index i.



Min-heap: complete shape; every parent is no greater than either child.

OPERATION	RESTORE ORDER	EXAM CHECK
Insert	append; sift up	compare with parent
Remove min	move last to root; sift down	compare both children; swap with smaller
Peek min	read root	does not imply sorted remainder

During sift-down, compare both existing children. Choosing the left automatically may leave a smaller right child above its parent. Equal priorities are not necessarily stable.

COMPSCI130 Worked Drills, Traps & Fast Recall

INTRODUCTION TO SOFTWARE FUNDAMENTALS · EXAM REVISION SHEET · SIDE 2 OF 2

Full guide + ask Sia about COMPSCI130

SIDE 2/2 FIVE WORKED DRILLS · JUDGEMENT CARDS · CLASSIC TRAPS · EXAM-MORNING RETRIEVAL

SHOW STATE + INVARIANT + COST

Answers include the reasoning path

Drill 1 · Aliases + Loop C1 · 6 MARKS

Prompt. `x=[2,4]; y=x; x.append(6); y=y+[8]; total=0`. For each `v in x`, add `v` only if `v%4==0`. Find x, y, total; name the alias-breaking line.

1. x,y → object [2,4]
2. append mutates → [2,4,6]
3. y+[8] creates new object
4. predicate: 2✕ 4✓ 6✕

x=[2,4,6] · y=[2,4,6,8] · total=4

Answer path: preserve object identity until concatenation constructs a new list.

STEP	STATE AFTER STEP	WHY
y=x	both names share one object	assignment does not copy
append(6)	x,y both see [2,4,6]	in-place mutation
y=y+[8]	y gets [2,4,6,8]	new list + rebinding
loop	total=4	only 4 has remainder 0

TRAP → *"Assignment copies the list" is false here. Draw names and objects; distinguish mutation from construction.*

Test Card · Boundary Parser C2

Contract: decimal strings for ages 0–130; invalid syntax or out-of-range values raise `ValueError`.

```
try: age = int(text)
except (TypeError, ValueError) as exc:
  raise ValueError("age must be a decimal integer") from exc
if age not in range(0,131): raise ValueError("age out of range")
return age
```

INPUT	PATH	RESULT
"20"	convert; range passes	20
"" / "twenty"	conversion fails	ValueError
"−1" / "131"	converts; range fails	ValueError

TEST EVIDENCE + EXCEPTION SCOPE

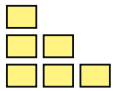
Test case = input, expected behaviour and observation checking one rule. State the expectation first; a mismatch then localises the defect.

CASE	EXPECTED EVIDENCE
Normal	"20" returns 20
Boundary	0 / 130 return the boundary ages
Syntax failure	"" / "twenty" raise ValueError
Range failure	"−1" / "131" raise ValueError

TRAP → *Catch only conversion errors being translated; a blanket catch can hide defects. Passing tests shows observed behaviour, not absence of defects.*

Drill 2 · Exact Count → Θ C3 · 7 MARKS

Prompt. `for i in range(n): for j in range(i): count += 1`. Derive exact count; evaluate n=1000; state tight bound.



i=0 → 0 increments

i=1 → 1 increment

i=2 → 2 increments

$\Sigma_{i=0}^{n-1} i = n(n-1)/2$

n=1000 → 499,500 → $\Theta(n^2)$

Triangular nesting: varying inner bounds must be summed, not copied as n×n.

1. Inner loop executes exactly i times: 0,1,2,…,n−1.
2. Total = `0+1+…+(n-1)=n(n-1)/2`.
3. At n=1000: `1000×999/2=499,500`.
4. Leading term $n^2/2$ has upper and lower constant multiples of n^2, so $\Theta(n^2)$.

TRAP → *`range(i)` has i iterations ending at i−1. Using i+1 creates an off-by-one sum.*

Binary Search Trace C4 · WORKED VALUES

Find 42 in `[3,11,19,27,42,58,73]`, inclusive low/high.

(0,6,3): a[3]=27 → low=4

(4,6,5): a[5]=58 → high=4

(4,4,4): found 42

Inclusive interval: continue while low ≤ high; absence ends at low > high.

DESIGN	DAILY LEADING WORK FOR N=10,000
Unsorted	2 appends O(1) + 100 membership queries O(n)
Sorted array	2 insertions O(n) + 100 queries O(log n)

For the stated read-heavy day, maintaining order is the better of these two designs, while constants and batching should still be measured.

INCLUSIVE-BOUND AUDIT

OBSERVED COMPARISON	NEXT STATE
a[mid] < target	`low = mid + 1`
a[mid] > target	`high = mid - 1`
a[mid] = target	return mid
low > high	target absent

Preconditions: ordered data, random access and one consistent interval convention.

TRAP → *Do not mix half-open and inclusive bounds. Here high remains a candidate, so continue while `low ≤ high`.*

Drill 3 · Recursive Digit Sum C7 · 7 MARKS

Definition. Single digit → n; otherwise `n%10 + digit_sum(n//10)`. Trace `digit_sum(2043)` and prove termination.

expand 3 + f(204)

3 + 4 + f(20)

3 + 4 + 0 + f(2)

return f(2)=2 → f(20)=2

f(204)=6 → f(2043)=9

Unwind in caller order: 0+2=2, 4+2=6, 3+6=9.

FRAME N	LOCAL DIGIT	DEEPER CALL	RETURN
2	base	—	2
20	0	2	2
204	4	2	6
2043	3	6	9

Termination: on the stated non-negative integer domain, for n≥10, `0≤n//10≤n-1`. Thus n strictly decreases and is bounded below until the single-digit base case.

Pointer Drill · Insert + Delete C8 · 7 MARKS

Start A→B→C→None. Insert X after A; then delete B without losing C.

start A → B → C → None

insert A → X → B → C → None

delete A → X → C → None

Read or save each successor before detaching the only link that reaches it.

```
old_next=A.next; X.next=old_next;
A.next=X
target=X.next; X.next=target.next;
target.next=None
```

TRAP → *If A.next is overwritten before B is saved or linked from X, the old suffix can become unreachable.*

LINK-INVARIANT CHECK

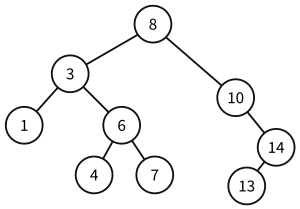
Link invariant keeps node references connected and well formed. Before overwriting a link, name every node reachable only through it.

CHECKPOINT	EVIDENCE
Before insert	save successor B
Before delete	read B.next first
After update	head still reaches the suffix

COST → *Relinking is O(1) only after the target position is known; a traversal needed to find it remains O(n).*

Drill 4 · BST Delete C10 · 9 MARKS

Insert `[8,3,10,1,6,14,4,7,13]`. Search 7; delete 3 with its inorder successor.



Constructed BST; search 7 follows 8→3→6→7.

1. Node 3 has two children. Its successor is the minimum in its right subtree: **4**.
2. Copy 4 into 3's position; delete the original 4 leaf.
3. Final inorder: `[1,4,6,7,8,10,13,14]`.

CHECK	EVIDENCE
Search path	one comparison and branch per level
Successor	leftmost node of right subtree
Key set	only 3 removed
Order	inorder remains strictly sorted

TRAP → *The successor is not automatically the immediate right child.*

Hash Drill · Linear Probing C11 · 8 MARKS

Size 7, `h(k)=k mod 7`; insert 10,17,24,5.

10: 3

17: 3→4

24: 3→4→5

5: 5→6

3:10 · 4:17

5:24 · 6:5

α=4/7

Lookup 24 probes indices 3,4,5; home-slot mismatch does not prove absence.

Successful lookup: compare 10 at 3, 17 at 4, find 24 at 5. With no deletion, unsuccessful lookup can stop only at a never-used slot.

PROBE + PERFORMANCE AUDIT

Collision = distinct keys sharing a candidate position; the stated resolution strategy gives the next probe. Home-slot mismatch does not prove absence.

CHECK	WHAT TO REPORT
Table state	n=4, m=7, α=n/m=4/7
Successful lookup	probe 3,4,5 until key 24
Absent, no deletion	stop at first never-used slot

LIMIT → *Average constant-time claims depend on key distribution, resizing and collision handling; they do not guarantee constant worst-case time.*

Drill 5 · Min-Heap C12 · 9 MARKS

Start `[2,5,4,9,7,8]`. Insert priority 1; then remove the minimum.

append [2,5,4,9,7,8,1]

swap ↑ [2,5,1,9,7,8,4]

[1,5,2,9,7,8,4]

remove root←4: [4,5,2,9,7,8]

sift ↓: [2,5,4,9,7,8]

Insertion restores order upward; removal restores order downward while shape stays complete.

1. Append 1 at i=6; parent is `(6-1)//2=2`. Swap with 4.
2. At i=2, parent is 0. Swap with 2 → `[1,5,2,9,7,8,4]`.
3. Remove 1; move last key 4 to root.
4. Compare children 5 and 2; swap with smaller 2 → `[2,5,4,9,7,8]`.

INVARIANT	AFTER EACH OPERATION
Shape	array represents a complete tree
Min-order	every parent ≤ each child
Removal	root is minimum; remainder need not be sorted

Integrated ADT Drill COURSE WORKED EXAMPLE

Commands arrive A,B,C; jobs arrive (draft,4),(exam,1),(notes,3).

SERVICE	ADT + INVARIANT	FIRST TWO REMOVALS
Undo	stack; LIFO	C, then B
Print service	min-priority queue; minimum key	(exam,1), then (notes,3)

Choose from the removal rule. Both services insert items, but reverse arrival demands a stack while smallest numeric priority demands a min-priority queue.

HEAP OPERATION AUDIT

Priority queue removes by priority; a *heap* is a complete tree with parent–child order.

ZERO-BASED PARENT

`parent(i)=⌊(i-1)/2⌋, for every non-root index i`

CHANGE	REQUIRED CHECK
Insert	append; restore order upward
Remove min	last to root; compare both children; restore downward
Audit	complete shape; every parent ≤ each child

TRAP → *Compare both children; choosing left automatically may miss a smaller right. A heap exposes the extreme, not a sorted or stable sequence.*

Judgement Cards C1–C12

IF YOU SEE X…	→ DO Y
two names, one mutable object	draw identities; classify mutation vs rebinding
nested loops with varying bound	write a sum before naming Big O
binary search	verify order + random access + bound convention
public class method	state contract; check invariant after success/failure
reverse arrival order	stack / LIFO
first arrival first	queue / FIFO
recursive function	base, smaller call, decreasing lower-bounded measure
link overwrite	save the successor first
tree output order	identify when root is visited
BST delete with two children	use successor = min(right subtree)
hash collision	follow the stated resolution strategy
heap sift-down	compare both existing children

What output / removal rule is required?

ordering discipline

key / shape rule

stack/queue

BST/hash/heap

Method chooser: derive structure from required behaviour, not surface syntax.

Classic Traps SOURCE LIMITS

WRONG SHORTCUT	REPAIR
Assignment always copies	track object identity
Tests prove correctness	claim only observed behaviours
$O(n^2)$ means exact n^2 work	separate exact count from growth
Binary search is always fastest	include ordering/maintenance cost
Class = abstraction	write ADT contract + invariant
List means queue or stack	state logical discipline and cost
Recursion works on sample, so done	prove termination; inspect repeated work
Relinking is always O(1)	include traversal to target
Inorder is always sorted	only under BST invariant
BST operations are always log n	cost depends on height
Hash lookup is guaranteed O(1)	state average-case assumptions
Heap array is sorted	only parent-child order is guaranteed

Exam-Morning Plan RETRIEVAL, NOT REREADING

1. Recite the 12 bold decisions in the judgement table without looking.
2. Trace one alias, one recursion and one structural update on paper.