The Ultimate AP® Computer Science A Unit 2 Cheat Sheet

2.1–2.2 Algorithms & Boolean Expressions

Algorithms

Algorithm = step-by-step process; built from **sequencing** (in order), **selection** (if) & **repetition** (loops).

Selection chooses which code runs based on a condition; **iteration** repeats code while a condition holds.

Relational Operators

Operator	Meaning	Ex (x = 5)
==	equal to	x == 5 → true
!=	not equal	x != 5 → false
< , >	less / greater	x > 3 → true
<= , >=	or equal	x <= 4 → false

Boolean expression evaluates to true or false; store it in a boolean: boolean pass = score >= 70;

Primitive compare: use == for int, double, boolean, char.
Objects: == checks the same reference only.

Comparing Objects

Strings: use s1.equals(s2) for same characters; s1.compareTo(s2) < 0 means s1 comes 1st alphabetically.

Null check: obj == null is fine; calling obj.equals(...) when obj is null throws a NullPointerException at run time.

Doubles

Avoid == on doubles: 0.1 + 0.2 == 0.3 is false due to rounding; compare w/ Math.abs(a − b) < 0.0001 instead.

Assignment ≠ equality: = stores a value; == compares. if (x = 5) won't compile in Java for an int x.

Boolean methods: s.equals(t), s.isEmpty() & custom methods like isEven(n) return true/false for if tests.

⚠ s1 == s2 can be false even when both Strings hold "hi"; it compares references, not characters.

✓ Trace a Boolean by plugging in values: x = 5 → (x > 3) is true; write T or F above each part of it.

2.3–2.4 if, else & else if

if & if-else

if (condition) { ... } runs the block only when the condition is true; otherwise the whole block is skipped.

if-else: exactly 1 of the 2 blocks runs. **else if** chain: checks top to bottom; the 1st true branch runs.

score	Chain result
95	A (1st true)
85	B
65	else → F

Chain: if (s >= 90) A; else if (s >= 80) B; else F. A 95 gets only A; the later branches never even run.

Order Matters

Wrong order: if (s >= 80) B; else if (s >= 90) A → a 95 gets B b/c the 1st true test wins; put strictest 1st.

Separate ifs (no else) are each checked, so several can run. An else-if chain runs at most 1 branch total.

Nested if

Nested: an if inside another if; the inner test runs only when the outer condition was true 1st, so trace outer → inner.

Dangling else: an else pairs w/ the NEAREST unmatched if, regardless of indentation; use braces.

Braces

No braces: only the 1st statement after the if belongs to it; Java ignores your indentation completely.

Ex: if (x > 0) count++; total++; → total++ ALWAYS runs; only count++ depends on the condition being true.

Ex: if (a > b) max = a; else max = b; → 1 runs.

⚠ A stray semicolon, if (x > 0); { ... }, ends the if statement, so the block below it runs every single time.

✓ When tracing a chain, stop at the 1st true condition & cross out every branch below it, including else.

2.5–2.6 Compound Booleans & De Morgan

Logical Operators

A	B	A && B	A || B	!A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Precedence: ! first, then &&, then ||. a || b && c means a || (b && c); add parentheses to be safe & clear.

Short-Circuit Evaluation

&&: if the left side is false, the right side is NOT evaluated. **||**: if the left is true, the right side is skipped.

Guard: if (x != 0 && 10 / x > 2) → safe; when x = 0, the division never runs, so no ArithmeticException.

De Morgan's Laws

!(a && b) ≡ !a || !b. **!(a || b)** ≡ !a && !b. Flip each part & swap && ↔ ||; flip the relational operators too.

Ex: !(x > 5 && y == 3) ≡ x <= 5 || y != 3.

Negation table: ! (==) → !=; ! (<) → >=; ! (>) → <=; ! (<=) → >; ! (>=) → <. Always include the boundary.

Equivalent Expressions

Test w/ values: plug in 2–3 cases (true/true, true/false, false/false) to check if 2 expressions match.

Range check: x >= 1 && x <= 10 means 1–10 inclusive; 1 <= x <= 10 is NOT valid Java & won't compile.

Odd/even ex: n % 2 == 1 misses negative odd numbers (−3 % 2 = −1); n % 2 != 0 catches every odd n.

Redundant tests: x > 5 && x > 3 simplifies to x > 5, while x > 5 || x > 3 simplifies to x > 3 (the looser test).

⚠ The opposite of x > 5 is x <= 5; writing x < 5 drops the case x == 5, a classic MCQ trap to watch for.

✓ For "which is equivalent" MCQs, build a quick truth table or test boundary values like 5 & 6 in both.

2.7 while Loops

Structure

while (condition) { body }: checks the condition BEFORE each pass; if it's false at the start, the body runs 0 times.

3 parts: initialize a variable before, test it in the condition & update it in the body so the loop can end.

Ex: int i = 1; while (i <= 5) { sum += i; i++; } → sum = 15 after 5 passes, & i ends the loop holding 6.

Common Patterns

Sentinel loop: repeat until a special value appears (e.g., −1) that signals "stop reading input" to the loop.

Condition-driven: while (n > 0) { n /= 10; count++; } counts digits; the # of passes isn't known up front.

Ex digits: n = 472 → 472 → 47 → 4 → 0 → 3 passes, so count = 3; the loop stops once n reaches 0.

Infinite Loops & Errors

Infinite loop: the update never makes the condition false (you forgot i++, or wrote i-- when it needed i++).

Off-by-one: < vs. <= changes the pass count by 1; trace the 1st & last pass to confirm the bounds.

Loop	Passes
i = 0; i < n	n
i = 1; i <= n	n
i = 0; i <= n	n + 1
i = 1; i < n	n − 1

Final Values

After the loop, the control variable holds the 1st value that failed the test (i = 6 above, not 5), so check it.

⚠ A variable declared inside the loop body can't be used after the loop; declare it before the loop.

✓ MCQ "what is printed" → make a trace table: 1 column per variable, 1 row per pass, plus the test.

2.8 for Loops

Structure

for (init; condition; update) { body }: init runs once; condition is checked before each pass; update after.

Ex: for (int i = 0; i < 5; i++) prints 0 1 2 3 4; i is declared in the header, so it's out of scope after the loop.

Same as while: int i = 0; while (i < 5) { ...; i++; } → any for loop can be rewritten as a while loop.

Counting Passes

Formula: for (i = a; i < b; i++) runs b − a times; for (i = a; i <= b; i++) runs b − a + 1 times in total, always.

Step size: for (i = 0; i < 20; i += 5) → 0, 5, 10, 15 → 4 passes. List the values; don't just divide & guess.

Counting down: for (int i = 10; i > 0; i -= 2) → 10, 8, 6, 4, 2 → 5 passes, since the value 0 fails the test i > 0.

Ex: for (int i = 1; i <= 100; i *= 2) → 1, 2, 4, 8, 16, 32, 64 → 7 passes; the next value, 128, fails the test.

Choosing a Loop

Use	When
for	Known # of passes
while	Stop on a condition
for over a String	Visit each index

Common Errors

Changing i inside a for body (i++ in the body too) skips values; MCQs test exactly this kind of change.

Accumulate: int sum = 0; for (int i = 1; i <= n; i++) sum += i; → sum = 1 + 2 + ... + n = n(n + 1)/2 at the end.

Semicolon trap: for (int i = 0; i < 5; i++); → an empty loop body, so the block after it runs only once.

⚠ for (int i = 0; i <= str.length(); i++) goes 1 past the last index → StringIndexOutOfBoundsException.

✓ Count passes w/ the formula, then confirm by tracing the 1st 2 values & the last value of the loop variable.

2.9–2.10 Standard Algorithms

Number Algorithms

Sum / average: total += value each pass; average = (double) total / count; cast to avoid int division.

Min / max: start w/ the 1st value (or Integer.MAX_VALUE for min), then update when a better one appears.

Count matches: if (condition) count++; e.g., count the evens w/ if (n % 2 == 0) count++ in the loop.

Prime check: test divisors from 2 up to n − 1 (or √n); if any n % d == 0, n is not prime, so stop early.

Divisibility: a % b == 0 means b divides a evenly. **Digits**: n % 10 = the last digit; n / 10 drops the last digit.

String Algorithms

Task	Key code
Each char	s.substring(i, i + 1)
Reverse	rev = ch + rev
Count a letter	if (ch.equals("a"))
Find a word	s.indexOf("cat") != −1

Traverse: for (int i = 0; i < s.length(); i++) visits each index once; the last valid index is length − 1.

Ex reverse: "abc" → the loop builds "a", then "ba", then "cba" by putting each new char in front of rev.

Ex count: count "a" in "banana" → check each 1-letter substring; matches at 1, 3, 5 → count = 3.

Substring Search

Check each start: for (i = 0; i <= s.length() − k; i++) compare s.substring(i, i + k) to the k-letter target.

Early exit: a found-it loop can return true the moment it matches, but return false only AFTER the loop.

⚠ Integer division: 7 / 2 = 3 in Java; cast 1 operand to double for an average: (double) sum / n.

✓ FRQ: these patterns (count, sum, min/max, traverse) appear every year; write them from memory.

2.11–2.12 Nested Loops & Run Time

Nested Iteration

Inner loop runs ALL its passes for EACH pass of the outer loop; total = outer passes × inner passes.

Ex: for (i = 0; i < 3; i++) for (j = 0; j < 4; j++) → 3 × 4 = 12 executions of the inner loop's body in all.

Dependent inner: for (j = 0; j < i; j++) w/ i = 0...3 → 0 + 1 + 2 + 3 = 6 passes in total, not 4 × 4 = 16.

Ex: i = 1...4, j = i...4 → 4 + 3 + 2 + 1 = 10 runs; a triangle pattern sums to n(n + 1)/2 when n = 4.

Patterns

Code	Output
i 1..3, j 1..i print *	*, **, ***
i 0..2, j 0..2 print i+j	0 1 2 / 1 2 3 / 2 3 4

Rows & columns: the outer loop usually controls rows & the inner loop controls columns in a pattern.

println vs. print: print stays on 1 line; a println after the inner loop ends that row & starts the next one.

Grid traversal: nested loops over rows & columns visit every (row, col) pair; Unit 4 uses this for 2D arrays.

Informal Run-Time Analysis

Statement execution count: how many times a line runs; count passes of every loop that contains it.

Ex: for (k = 0; k < n; k += 2) → about n/2 passes; a nested n × n pair → n^2 runs of the innermost line.

Compare: a loop that halves n each pass (n /= 2) runs about $\log_2$ n times, far fewer than n passes.

Efficiency: fewer passes = faster; exiting early (break-free, w/ a found flag) can cut the count a lot.

⚠ An inner loop that depends on the outer variable doesn't give outer × inner; add up each row's count.

✓ For count questions, list the inner count for each outer value (0, 1, 2, ...) & then add them all up.

Tracing & Exam Strategy

Exam Format

Section I: 42 MCQs in 90 min (55%). **Section II**: 4 FRQs in 90 min (45%). Unit 2 = 25–35% of the exam.

Java Quick Reference lists String & Math methods; it does NOT explain if/loop syntax, so know those cold.

Tracing Method

Trace table: 1 column per variable + the condition; update after every line; mark T/F on each test.

Shortcut: for long loops, find the pattern after 3 passes, then jump to the last pass instead of all n passes.

MCQ pacing: ≈ 2 min per question; skip long traces, mark them & come back after the quick ones.

Top Unit 2 Traps

Trap	Fix
< vs. <=	Count passes
== on Strings	Use .equals
Missing braces	Only 1st line is in
else-if order	Strictest test 1st
int / int	Cast to double

FRQ Tips

Methods & control (FRQ 1): expect loops w/ if inside; return the right type & don't print unless told.

Partial credit: correct loop bounds & condition logic earn points even if 1 small detail is off; always attempt.

Edge cases: test your code mentally w/ an empty String, a single element & the very last index before moving on.

Read the header: parameters & return type.

⚠ Returning inside a loop on the 1st check (if ... return true; else return false;) ends the method after 1 pass.

✓ Write the loop header 1st, then the if inside; check bounds before writing anything else.