The Ultimate AP® Computer Science Principles Big Idea 2 Cheat Sheet

2.1 Bits, Bytes & Abstraction

Digital Data

Bit = binary digit, 0 or 1. Lowest-level piece of every value a computer stores; all data is digital (made of bits).

Byte = 8 bits → 2^8 = 256 patterns (00000000 to 11111111).
Abstraction: bits grouped to stand for numbers, text characters, colors, sound, images, even instructions.
Context: same bit sequence can mean different things, e.g., 01000001 = 65 as a number, or 'A' as a character.
Data values: stored in variables, lists or constants; passed as input to (or output from) procedures + programs.

How Many Values?

Bits	Patterns	Range (unsigned)
1	2	0–1
3	8	0–7
4	16	0–15
8	256	0–255
n	2^n	0 to 2^n−1

+1 bit → doubles the number of values (2^n → 2^{n+1}). **+n bits** → ×2^n, not +n: 8 → 10 bits = 4× as many values (256 → 1,024).

Bits needed for k values: smallest n w/ $2^n \geq k$. 100 values → 7 bits, since 2^7 = 128 ≥ 100 but 2^6 = 64 is too few.

Common Encodings

Color (RGB): 8 bits each for red, green, blue → 24 bits/pixel, each channel 0–255 → $2^{24} \approx 16.7$ million colors.
Text: each character mapped to a number (ASCII, Unicode) → stored as bits; a string = a sequence of these codes.

⚠ Same bits, different meaning: the program's context (number, char, color) decides how a bit pattern is read.

✓ "How many bits for N items?" → find the 1st power of 2 ≥ N. Never round down: 2^n must cover all N.

2.1 Binary ↔ Decimal (DAT-1.C)

Place Values

Decimal (base 10): digits 0–9, place = power of 10. **Binary** (base 2): digits 0–1, place = power of 2.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Binary → Decimal

Method: write place values under each bit, add the places holding a 1. Rightmost bit = 1s place, next = 2s, then 4s.
Examples: 101101 = 32 + 8 + 4 + 1 = **45**; 1100100 = 64 + 32 + 4 = **100**; 1000 = 8 (only the 8s place is on).

Max value: n bits → largest = 2^n−1 (all 1s): 1111 = 15, 11111111 = 255. 2^n needs n+1 bits: 16 = 10000 (5 bits).
Odd/even: last bit 1 → odd, last bit 0 → even. Quick way to rule out answer choices before converting the whole number.

Decimal → Binary

Subtract method: take biggest power of 2 ≤ number, write 1, subtract, repeat; 0 in every unused place.
Example: 77 → 64 (r 13) → 8 (r 5) → 4 (r 1) → 1 → **1001101**. Check: 64+8+4+1 = 77; the 32, 16 + 2 places get 0s.
Divide method: ÷2 repeatedly, record remainders, read them bottom → top: 13 → 6 r1 → 3 r0 → 1 r1 → 0 r1 → 1101.

Compare & Order

Ignore leading 0s, then longer number = bigger. Same length → 1st differing bit decides (the 1 wins).
Append a 0 on the right → value ×2 (101 = 5 → 1010 = 10). Append a 1 → ×2 + 1 (1011 = 11). Drop last bit → ÷2.
Ordering example: 0111 (7) < 1010 (10) < 1100 (12) < 10000 (16); a leading 0 adds no value, so 0111 = 111.

⚠ Read place values right → left. Starting at the left w/ 1 gives wrong answers on every conversion.

✓ MCQ ordering: convert each to decimal on scratch paper; mixed base 2/base 10 choices are a common trap.

2.1 Overflow, Roundoff & Analog Data

Fixed Bits → Limits (DAT-1.B)

Fixed-size integers: many languages use a fixed # of bits → limits range of values + math on them.
Overflow error: result needs more bits than are available. 4-bit unsigned: 15 + 1 → 10000 → keeps 0000 = 0.
Real-world bugs: counters, odometers, game scores that wrap to 0 or go negative after hitting their max.
Roundoff error: fixed bits for real (decimal) numbers → some values only stored as approximations, e.g., 1/3, 0.1.
Effect: tiny errors add up over many operations; equality tests on reals (0.1 + 0.2 = 0.3) can fail.

Exam Reference Language

AP pseudocode: integer size limited only by computer memory → no overflow in exam-sheet code.
Other languages (fixed bits) → overflow possible. Read the question for which kind of language it describes first.
Overflow fix: use more bits (e.g., 32 → 64-bit) or a language w/ unlimited-size integers like AP pseudocode.

Analog vs Digital

	Analog	Digital
Values	Change smoothly	Discrete steps
Examples	Pitch, volume	Bits in a file
Stored as	Can't be exact	Exact bits

Analog examples: pitch + volume of music, colors of a painting, position of a sprinter → all sampled to store.
Sampling: measure analog signal at regular intervals (samples) → each sample stored as bits.
More samples or more bits/sample → closer approximation, but bigger file. Still never 100% exact.

⚠ Overflow = too BIG a number for the bits. Roundoff = too PRECISE a real number. Don't swap them.

✓ Digital data represents analog data via abstraction: an approximation, not a perfect copy of the original.

2.2 Lossless vs Lossy Compression

Why Compress? (DAT-1.D)

Compression: reduce # of bits stored or transmitted → saves storage space, sends faster, uses less bandwidth.
Fewer bits ≠ less info: redundant data fits in fewer bits w/ nothing lost, e.g., "100×white" vs 100 entries.
Size reduction depends on 2 things: amount of **redundancy** in the data + the **algorithm** applied to it.

The Two Types

	Lossless	Lossy
Original?	Fully rebuilt	Approximation
Size cut	Usually less	Usually more
Use when	Quality key	Size/speed key
Examples	Text, code, ZIP	Photos, music

Lossless: guarantees complete reconstruction of the original data. Bank records, source code, legal docs.
Lossy: drops details people barely notice → big cuts; original can't be recovered from compressed file.
Lossy trade-off: ↑ compression → ↓ quality (e.g., blocky low-quality JPEG); the lost detail can never be restored.

Run-Length Encoding (Lossless)

RLE: replace repeats w/ count + value: AAAABBBCC → 4A3B2C (9 chars → 6). Decode back exactly.
Weak case: little repetition, e.g., ABCD → 1A1B1C1D gets **longer**. Redundancy drives the savings.
Dictionary method: swap repeated words/patterns for short codes + store a key → also lossless, fully reversible.
Typical: images, audio, video → lossy; text, spreadsheets, code → lossless (1 wrong bit breaks them).

⚠ Lossy ≠ "bad": streaming video + phone photos use lossy b/c small size matters more than perfect detail.

✓ Scenario MCQ: "must restore exactly" → lossless. "Minimize size / transmission time" → lossy.

2.3 Data → Information & Metadata

Information (DAT-2.A)

Information: collection of facts + patterns extracted from data. Data → trends, connections, solutions.
Trends: data → spot trends, make connections, address problems, e.g., ride data → where to add bus routes.
Correlation ≠ causation: data may show 2 variables move together; more research needed to show 1 causes the other.
3rd factor: ice cream sales + drownings rise together; hot weather drives both → not cause + effect.
Combine sources: one data set often lacks what's needed for a conclusion → merge several sources.
Size matters: dataset size affects how much info can be extracted; bigger → more patterns visible.

Metadata (DAT-2.B)

Metadata = data about data. Photo = data; date, location, file size, camera model = its metadata (stored w/ the file).

Data	Its metadata
Photo	Date, GPS, size
Song file	Artist, length
Email	Sender, time
Web page	Author, keywords

Uses: find, organize + manage info; adds structure; ↑ effective use of data sets (e.g., search by date).
Independence: editing or deleting metadata does NOT change the primary data (renaming a tag ≠ editing the photo).
Organize: sort photos by date, filter songs by artist, search email by sender → all use metadata, not content.
Privacy: metadata alone can reveal a lot, e.g., GPS + timestamps on photos → where you live.

⚠ "Which can be found using ONLY metadata?" → match only fields listed (date, size, tags), never the content itself.

✓ Answer pattern: "need more data / further research" is the right call before claiming 1 thing causes another.

2.3 Data Challenges: Clean, Bias, Scale

Messy Data (DAT-2.C)

Challenges at every size: need to clean data, **incomplete** data, **invalid** data, combining sources.
Non-uniform: open-text fields → users spell, abbreviate, capitalize differently (NY, N.Y., new york).
Cleaning: makes data uniform w/o changing its meaning, e.g., all variants → "New York"; trim spaces, fix capitals.
Duplicates: same record entered twice → double-counted; remove during cleaning, before counting or averaging.
Invalid: impossible values (age −4, 31 Feb). **Incomplete**: missing fields → skews counts + averages.

Bias

Bias often comes from the type or source of data collected, e.g., online-only survey misses people w/o internet.
More data ≠ less bias: collecting more from the same biased source keeps the same bias, just w/ more rows.
Fix: change how/where data is collected → sample from a wider, representative group of the population.

Scale & Tools

Issue	Response
Huge data set	Parallel systems
Growing data	Scalable storage
Limited tools	Less you can do
Many sources	Combine + clean

Scalability: system's computational capacity limits how data sets are processed + stored (more data → more machines).
Ability to process data depends on users' skills + their tools (spreadsheet vs custom program vs cloud cluster).

⚠ Trap: "collect a larger sample" is NOT the answer to a bias question. Change the source/method instead.

✓ Cleaning question: right answer keeps meaning the same; deleting rows or altering values is not cleaning.

2.4 Programs Process Data (DAT-2.D)

Four Core Operations

Process	Example
Transform	Double every item
Filter	Keep only band kids
Combine/compare	Sum list, max GPA
Visualize	Chart, histogram

Transform: change every element, e.g., add a parent's email to every student record, convert °F → °C.
Filter: keep only rows meeting a condition, e.g., positive numbers, orders > $50 → spot patterns.
Combine/compare: aggregate (sum, count, average) or find extremes (highest GPA, earliest date).
Visualize: tables, diagrams, charts, text → communicate insight + knowledge gained from data.

Tools

Search tools: find info efficiently in large data sets (by keyword, field, date range) instead of scanning every row.
Spreadsheets: organize data, sort, filter, formulas → find trends quickly w/o writing a full program.
Filtering systems: key tools to find info + recognize patterns, e.g., show only late orders → spot a slow warehouse.
Sort: order rows by a field (price, date) → top/bottom values + trends jump out; often paired w/ filter.

Can It Be Answered?

Method: list the columns given → does question need only those? Missing field → can't answer from data.
Example: table w/ store, item, price. "Most-sold item?" ✓ (count). "Buyer age?" × (no age field).

⚠ Filter first, then aggregate: count/average only the rows that meet the condition, or the result mixes groups.

✓ "Which can be determined?" MCQ: cross-check every needed field against the column list before choosing.

2.4 Insight from Data (DAT-2.E)

Iterative Analysis

Iterative + interactive: run a program, see result, refine question or filter, run again → deeper insight.
Filter + clean with programs → reliable data → insight + knowledge. Garbage in → garbage out.
Patterns emerge when data is transformed (sorted, grouped, graphed), e.g., sales by month → seasonal spike.

Combine, Cluster, Classify

Process	What it does
Combine	Merge data sources
Cluster	Group similar items
Classify	Label into categories
Translate	Change representation

Combining: join sets on a shared field (student ID) → answer new questions neither set could answer alone.
Translate: change representation, e.g., GPS points → map route, timestamps → hours per day → new insight.
Clustering: find natural groups, e.g., shoppers w/ similar habits. **Classifying**: sort into set labels (spam/not).

Exam Moves

Code + list MCQ: trace a loop that filters or sums a list; track the result variable each pass, then check its final value.
Insight → action: knowledge from data drives decisions, e.g., quiz data → schedule more tutoring before tests.
Order of steps: questions ask which sequence (filter → sort → count) gives the answer; test it on 2–3 rows.
Multi-select (pick 2): check both choices independently; a data question often has 2 valid methods.

⚠ Big Idea 2 = 17–22% of MCQs (~12–15 of 70): binary math + data scenarios are fast points; don't rush them.

✓ Create PT link: list data in your program = DAT-2 in action; explain how the list manages complexity.