CS617 | Chapter 02 Sorting | NOTES

Comparisons, instruction checks, and data movement

Xandrian Labs

Teaching edition R002 | 12 November 2026

CS617 | Chapter 02 Sorting

A practical experiment in comparisons, instructions, and data movement

Student NOTES - teaching edition R002 (12 November 2026). This edition describes the current Chapter 02 Sorting application (`sort_extra2`, x0.31a-dev) while preserving the scientific question of the original *Apple Polisher* extension. The older Apple Polisher WHAT and its accepted work remain historical, **voluntary and separate** from the frozen required Chapter 02 assignment. Nothing in this teaching edition adds a retroactive grading requirement.

1. The question: what does a sort cost?

A sorting program can produce the right order using very different amounts of work. Comparing two keys is not the same as moving the keys. The experiment distinguishes three measurements:

Counter	Meaning	What it does not measure
KEY-CMP	Logical key-comparison requests made by the sort	The number of data bytes moved
INST-CMP	Executed key-data comparison/check instructions	Overall elapsed time or total instructions
DATA-WR (shown as **WR**)	Bytes written into the active DATA array by the measured sort	Workspace writes, data generation, viewer output, validation

The counters apply to **sorting only**. Validation is performed afterward without charging the sorting counters. A correct `ORDER=PASS` is necessary evidence, but it is not a stopwatch reading and does not prove every possible property of the output.

> Key idea: two algorithms may have equal comparison counts but very different DATA-WR counts. The experiment asks us to explain why, not simply declare one algorithm universally best.

2. The four-option application menu

The four-line LCD presents a single main menu:

```
[setup ][sort  ][data  ][report]
```

Use the physical **FK1-FK4** keys, or their corresponding **numeric 1-4 shortcuts**, to choose the displayed slot. Numeric shortcuts apply to menus; they do **not** replace digits while editing numeric values or entering a VT command.

Slot	Main menu	What happens
FK1 / `1`	**setup**	Change `n`, `m`, or input pattern, then finish configuration
FK2 / `2`	**sort**	Choose Bubble, Shell, or Selection
FK3 / `3`	**data**	Open a read-only DATA document in VT
FK4 / `4`	**report**	Open the latest sort report in VT

The sort menu is `[bubble][shell ][select][back  ]`; the setup menu is `[n     ][m     ][input ][done  ]`. Each slot follows the same FK/digit convention. In the DATA pagination menu, slot 3 is intentionally empty and does nothing.

Configuration and reproducibility

The dataset contains **`n` records of `m` bytes each**, with total active storage at most **8,192 bytes**. The application checks `1 <= n <= 8192`, `1 <= m <= 8192`, and `n*m <= 8192`; the current generator also requires `n <= 255` when `m = 1`. Invalid settings produce an on-screen explanation.

Choose **ORDERED**, **REVERSE**, or **SAME**. These are deterministic test populations, not random samples. Before each new sort the application regenerates the configured input, clears its counters, and invalidates the previous last-run label. This makes comparisons between algorithms fairer: each algorithm receives the same original population.

3. Run a controlled comparison

1. Open **setup**, select `n=16`, `m=2`, and **REVERSE**, then choose **done**.
2. Open **sort** and run **Bubble**. Wait for the final four-line result. Use **MENU** to dismiss it and return home (EXE is an unadvertised compatibility alternative).
3. Open **report**. Confirm the algorithm name and the three counters; use `:q` to return.
4. Repeat for **Shell** and **Selection** without changing the configuration. A new sort regenerates the original REVERSE population.
5. Compare `KEY`, `INST`, and `WR`. Inspect **data** when useful to verify what was sorted.

The report has one truthful summary: **the most recently completed, validated sort**. `Algorithm: Not run` means no current completed sort is recorded, such as after regenerating DATA. The application does **not** maintain an independent history of all three algorithms; record each result externally if you want a comparison table.

Prediction-first reference case

The following values are **historical model predictions** for the controlled REVERSE generator, not a claim that this publication measured all three values again on x0.31a:

Algorithm	KEY-CMP	INST-CMP	DATA-WR
Bubble	120	240	480
Shell	66	132	128
Selection	120	240	32

Bubble and Selection request the same number of logical comparisons and execute the same number of counted key checks here, yet Bubble writes **15 times** as many DATA bytes (`480 / 32`). Explain the mechanics of the algorithms before drawing conclusions about performance.

For **ORDERED** or **SAME**, at `n=16,m=2`, the historical control predictions are Bubble `15 / 30 / 0`, Shell `49 / 98 / 0`, and Selection `120 / 240 / 0` in KEY/INST/WR order. Optimized Bubble can terminate early on these inputs; low movement alone is not proof of overall superiority.

Change the record width

For the same REVERSE population of 16 records, the original model predicts:

Width `m` (bytes)	Bubble WR	Shell WR	Selection WR
1	240	64	16
2	480	128	32
4	960	256	64
8	1,920	512	128
16	3,840	1,024	256

These predictions belong to the stated generator, implementations, and metric definition. They are not universal algorithm benchmarks or measurements of elapsed seconds.

4. Read DATA and REPORT through VT

VT (Virtual Terminal) presents a **24-row by 80-column** text document through a **four-row by 32-column** LCD window. The app constructs the document; VT provides the controls for browsing it. A DATA page and a REPORT are different documents, but their viewer controls are the same.

Key or command	Action inside the viewer
Arrow UP / DOWN	Move the selected virtual line by one
Arrow LEFT / RIGHT	Move the horizontal 32-column viewing window
`u` / `d`	Move **four** virtual lines up / down
`t` / `b`	Jump to the top / bottom
`:s`	Show viewer position/status
`:l`, row, EXE	Go to virtual row `1..24`; the numeric-entry underscore blinks
`:h`	Three-screen Help; EXE advances, Q returns to the document
`:i`	Toggle normal/inverse screen rendering
`:t`	Request transmission of the current 24x80 document, when supported
`:q`	Leave the viewer and return to the application

A leading **colon matters**: plain `t` moves to the top; `:t` requests transmission. Use the stand-alone **VT - Virtual Terminal User's Guide R003** for the complete guide. Inverse rendering is a display feature: the document's characters do not change. Serial availability is platform-dependent; a successful display does not certify serial support on every device.

How DATA uses the 80 columns

Each DATA line can show up to **24 bytes** as spaced hexadecimal pairs after a six-character record prefix. The first four virtual rows are headers; **20 data lines** remain on each logical document. Longer records continue on the next line of the *same* document. A continuation at a new document boundary is explicitly marked with `+` and a hexadecimal `OFFSET` in the header.

Two useful exercises:

- `n=1,m=32`: the one 32-byte record occupies **two successive lines (24 + 8 bytes)** in one document.
- `n=32,m=1`: the 32 records span **two documents (20 + 12 lines)**. After leaving VT, the application shows `[prev ][next ][ ][back ]`. 1/FK1 moves backward, 2/FK2 forward, and 4/FK4 returns home; movement clamps at the first and last document.

If DATA fits one document, `:q` returns directly home. If more documents exist, `:q` returns to the application-owned pagination menu. The menu is not a VT command.

What `:t` sends

In the qualified emulator workflow, the user has confirmed successful `:t` operations in both normal and inverse displays. The design exports the **current 24x80 character document**, not the visible 4x32 window and not the inverted pixels. Its raw format is **24 lines of 80 bytes, each followed by CR LF = 1,968 bytes**. An independent, byte-exact receiver capture has not yet been supplied; functional transmission and payload-length verification are different kinds of evidence.

5. Record an honest result

For each test, note the application version, `n`, `m`, input pattern, algorithm, `KEY`, `INST`, `WR`, and order result. Keep model predictions separate from values actually observed. A useful conclusion explains *how* comparisons and writes arise from the algorithm, *why* representation width matters, and *which* kinds of work were deliberately excluded.

Do not treat `ORDER=PASS` as proof of stability, preservation of the original multiset, portability across devices, or speed. More tests can strengthen those claims, but they are different claims.

6. Scope and reading

The frozen parent Chapter 02 and original voluntary Apple Polisher WHAT retain their historical authority. This teaching edition describes the **current experimental application**; it does not rewrite that original contract. The current working ROM has been byte-audited after an editorial comment cleanup; the physical device and raw serial capture remain separate qualifications. **TA and SOCRATIC remain instructor-only.**

- Application requirements: **Chapter 02 LIB2 Revamp Requirements** (engineering record, historical sections may retain former pending language).
- Reusable control reference: **VT - Virtual Terminal User's Guide R003** (application-agnostic).