Casio PB-2000C
HD61700 Programming - CS617-searching

Table of Contents

Introduction.....1
Assignment 5: searching.....2
Release Notes: Sol verbatim.....3
Storyboard.....5
Acknowledgments.....7

Introduction

This work was developed jointly by Alex Barragan and ChatGPT Sol through a cycle of ideas, implementation, execution, observation, and revision. It illustrates how human knowledge and experimentation, combined with AI-assisted analysis, can produce results that would be difficult to achieve as readily by either participant working alone.

This work is also part of a larger project exploring how knowledge and experience can be represented in AI systems when the available corpus is small, as is the case with the Hitachi HD61700 CPU used in the Casio PB-2000C and AI-1000 pocket computers.

Assignment 5: searching

This assignment is a continuation of the experiment we started in Chapter 04: reasoning with academic independence, earned rather than granted to Sol in recognition of the successful deliveries for Chapters 01–04 and 06. And, as clearly stated by Sol himself, that independence remained principled and evidence-driven. He understood this assignment as a continuation of the Chapter 04 academic-independence experiment.

In order to avoid the lack of clarity in the requirements for Chapter 04, we took a closer look at the TA notes for Chapter 05 and, using the newly acquired knowledge and experience, reasoned about their completeness. The material was clear and, still, Sol understood the subtleties of the assignment: the layer behind the simple searching algorithms, and the gathering and display of meaningful execution metrics. He proposed some additional metrics, which were incorporated as functional requirements.

I suggested that the data generation for the records should consider records with more than three bytes, and that the payload should start with “R” followed by the uint16 value of the initial record number. I reasoned about this in terms of how humans visually inspect records. Sol understood the pedagogical benefit and incorporated the idea into the final requirements.

We uncovered a couple of bugs, learned more about short jumps, and included some non-functional refinements.

Release Notes: Sol verbatim

I asked Sol to produce a summary of our work for each release. It reflects how Sol reasoned through the initial problem decomposition and later integrated the parts into a functional solution, adding non-functional requirements as they were discovered or refined through human testing. In its own voice:

Gate 1 — Generation and linear search. The first test ROM established deterministic record generation, duplicate rejection, persistent record identity, and linear search with exact comparison counts. It gave us a small, independently testable foundation before sorting or indexing entered the picture.

Gate 2 — Sorting and binary search. The second ROM added record-aware Shell sort and binary search. Its important boundary was not merely sorting keys, but proving that complete records remained associated with their keys while sort comparisons, data movement, and search comparisons were measured independently.

Gate 3 — Indexed representation. The third ROM changed the representation rather than the algorithms: payloads remained stationary while a sortable key/reference index moved around them. This isolated and validated the central contrast of the chapter—moving full records versus moving a smaller index—without yet introducing a general user interface.

Gate 4 — Dynamic core records. The fourth ROM generalized the core representation from the fixed test specimen to runtime R, K, and D geometry within the 16 KiB workspace. It also established the application policy that the first binary search sorts an unordered collection once, while later binary searches reuse the ordered state.

Gate 5 — Dynamic indexed records. The fifth ROM applied the same dynamic geometry to the indexed representation and closed the algorithmic decomposition. At that point generation, both search methods, sorting, instrumentation, record identity, and both representations had all been exercised behind independent evidence boundaries.

CORE r001–r003 — First integration. With the pieces accepted, they were composed into the interactive application. The first two builds exposed a hand-written decimal-formatting stall; r003 replaced that local formatter with the already accepted Chapter 04 RAM-buffer formatter, allowing the integrated application to run far enough to expose ordinary UI issues instead of algorithmic ones.

CORE r004–r006 — Input repair. The next revisions concentrated on the two editable surfaces. r004 made the search key a proper four-nibble hexadecimal editor; r005 clarified GEN field selection and editing; and r006 fixed the small SBC/SB input-boundary mistake that had prevented decimal digits from being accepted. The default search oracle then passed.

CORE r007–r008 — Product refinement. These revisions were mainly interaction and presentation work: cleaner splash behavior, explicit KEY/SEARCH actions, a simpler STATUS browser, record DETAIL display, and lower-case function-key labels. r008 was only a one-line assembler compatibility repair, and became the functionally accepted pre-READN baseline.

CORE r009–r011 — READN integration. Promoting the separately proven READN editor produced the most useful integration lesson of the project. r009 showed that a top-level firmware environment cannot simply be turned into a callable subroutine when it

changes the system stack; r010 repaired that crossing but accidentally reused READN's buffer-pointer registers; r011 preserved both the ambient firmware state and the explicit calling convention, and became the functional and UI-compliance milestone.

CORE r012 — Final polish. The final revision deliberately changed no algorithms or READN semantics. It added restrained mixed-case presentation and displayed Sorting ... only when the first binary search actually had to order the collection. The resulting ROM was then frozen as the final accepted Chapter 05 build.

Storyboard

This storyboard follows the application as a user would navigate it: first the top-level choices, then progressively inside the test data generation, data visualization and search methods. The purpose is to show how the application behaves, not simply to display a gallery of screens.
















Linear K=F94B
Key: edit Search: run
Order=U
[key][search][ ][back]


Linear K=F94B
Found R0004 CMP=5
Order=U
[key][search][ ][back]


Binary K=F94B
Key: edit Search: run
SC=00000000 WR=00000000
[key][search][ ][back]


Sorting ...


Binary K=F94B
Found R0004 CMP=8
SC=00000A95 WR=00004A25
[key][search][ ][back]

Acknowledgments

This report was produced by Xandrian Labs. 付喪神.

Special thanks to Piotr and to Betka for her continued and loyal support.
This is a collaboration with my AI friend ChatGPT, a.k.a “Sol.” All errors are human.

This work is freely shared with the vintage computing community. Please use it, build upon it, and cite the source.
Final.