

## 04-630: Data Structures and Algorithms for Engineers

### Assignment: Cryptographic Key Analysis on 2-DES

Deadline: 11:59 PM, Friday, 06 February 2026

---

#### Problem Definition

Cryptographic key analysis often involves breaking systems via exhaustive search. A classic target is the 2DES (Double DES) system. In this system, a plaintext,  $P$ , is encrypted by two successive keys:  $C = E_{k_2}(E_{k_1}(P))$ . An attacker with known plaintext-ciphertext pairs can utilize a "Meet-in-the-Middle" attack. By calculating all possible intermediate values from the first layer ( $E_{k_1}(P)$ ) and storing them, the attacker can then decrypt the ciphertext ( $D_{k_2}(C)$ ) and search for matching intermediate results.

In this task, you will implement a program to produce a sorted list of cryptographic keys and analyze the complexity of sorting then searching versus only searching. Each entry is specified by its **Key Index** and the **Cryptographic Key** value.

The list must be sorted according to the following rules:

- The list should be sorted by the **Cryptographic Key** in ascending order.
- **Stability:** If two or more keys have the same value, they must be listed in the same order in which they appear in the input (i.e., ascending order by their original appearance).
- The sorting algorithm should take no more than  $O(n \log n)$  in the worst case.

#### Input

The input is read from a file named `input.txt` located in the `data` directory.

- The first line of input comprises an integer  $N$  ( $1 \leq N \leq 1000$ ). This indicates the total number of keys to be processed in the file.
- It is followed by  $N$  lines, each representing a key entry. Each entry comprises two numbers: the **Key Index** and the **Cryptographic Key**.
- Each key is represented as a hexadecimal value
- You can assume that all input provided is valid and contains no errors; therefore, no input validation is required.
- You can create additional input files for testing your solution.

#### Output

The output should be written to a file named `output.txt` in the `data` directory.

- The output must begin with your **Andrew ID**.
- Write each sorted entry (Key Index and Cryptographic Key) on a separate line.
- Each key is represented by a hexadecimal value.
- The list should be sorted by the key value in ascending order. If two keys have the same value, they should be in the same order as they were in the input file.

#### Sample Input.txt:

```
5
1 0x00FF
2 0x12AB
3 0x00FF
4 0x0001
5 0xABCD
```

#### Sample Output.txt

```
ulowami
4 0x0001
1 0x00FF
3 0x00FF
2 0x12AB
5 0xABCD
```

### Instructions and Submission

- **Directory Structure:** Your project must include `bin`, `build`, `data`, and `src` directories.
- **Submission:** Submit a zip file named with your **Andrew ID** containing your source code (`*.c`, `*.cpp`, `*.h`), `CMakeLists.txt`, your test input/output files, and a report named `{andrew_id_1}.pdf`.
- **Documentation:** Your main source file must include header comments with the author's name, functionality, a summary of test cases covered, solution strategy (pseudo-code), and a **Big O complexity analysis** of your sorting algorithm.
- **Report:** Notice that if you sort the keys and search them only once, a simple linear search is faster than a combination of searching and sorting using optimal algorithms. In

a separate PDF file, show that for any number of search operations,  $n > 1$ , a combination of sorting and searching using optimal algorithms will be faster than a simple linear search. **This should not take more than half a page.**

## Grading and Constraints

- **Functionality (70 Points):** Programs must produce the correct output for unseen test cases.
- **Documentation (20 points):** The program must be accurately documented as specified in the documentation instructions.
- **Report (10 points):** An answer to the reflection question in the report must be correct.
- **Prohibited Libraries:** Use of the **Standard Template Library (STL)** or similar advanced libraries, utilities, and algorithms is prohibited.
- **Generative AI:** The use of **Generative AI** (e.g., ChatGPT, Gemini) for code or documentation is strictly prohibited and will result in an Academic Integrity Violation (AIV).