Project Presentation

(Morse-code Translator)

by Richard, Vickel and Timo

What Did We Do?


2
3
4
5
6
30
25
20

Background

What is Morse Code?

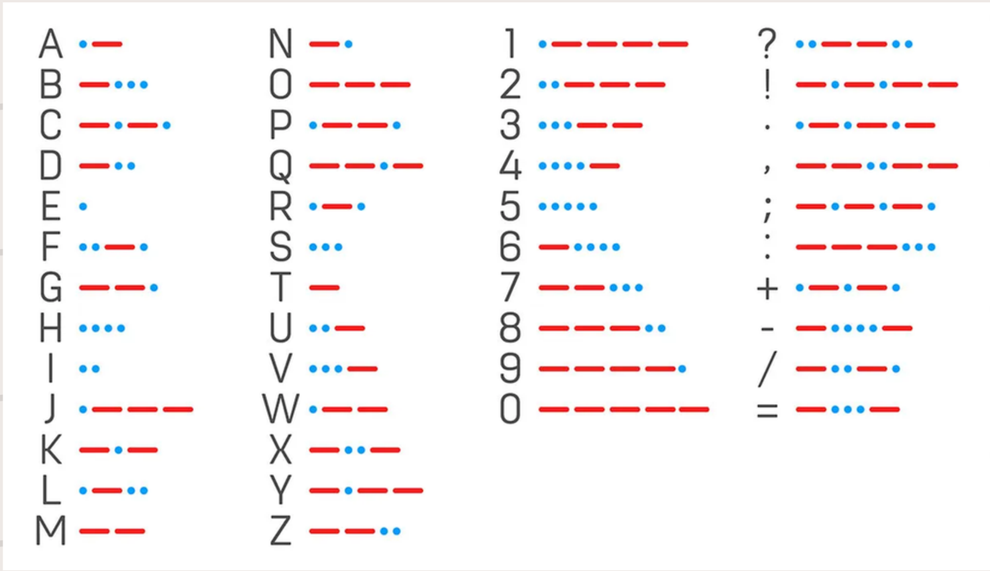
- Developed by Samuel Morse & Alfred Vail (1830s)
- Encodes text using a combination of dots (.) and dashes (-)
- Originally used for telegraphy but later adopted in various fields

Why is it important?

- Historically significant in military, aviation, and emergency communication
- Still used today in radio, survival situations, and accessibility technology

Why does it still matter?

- Unlike modern communication systems, Morse code works with minimal resources
- Can be transmitted through light, sound, or even touch (e.g., tapping on surfaces)



Problem Description

Manual Morse Code translation is inefficient:

- Requires memorization and practice
- Encoding and decoding by hand is slow and prone to errors

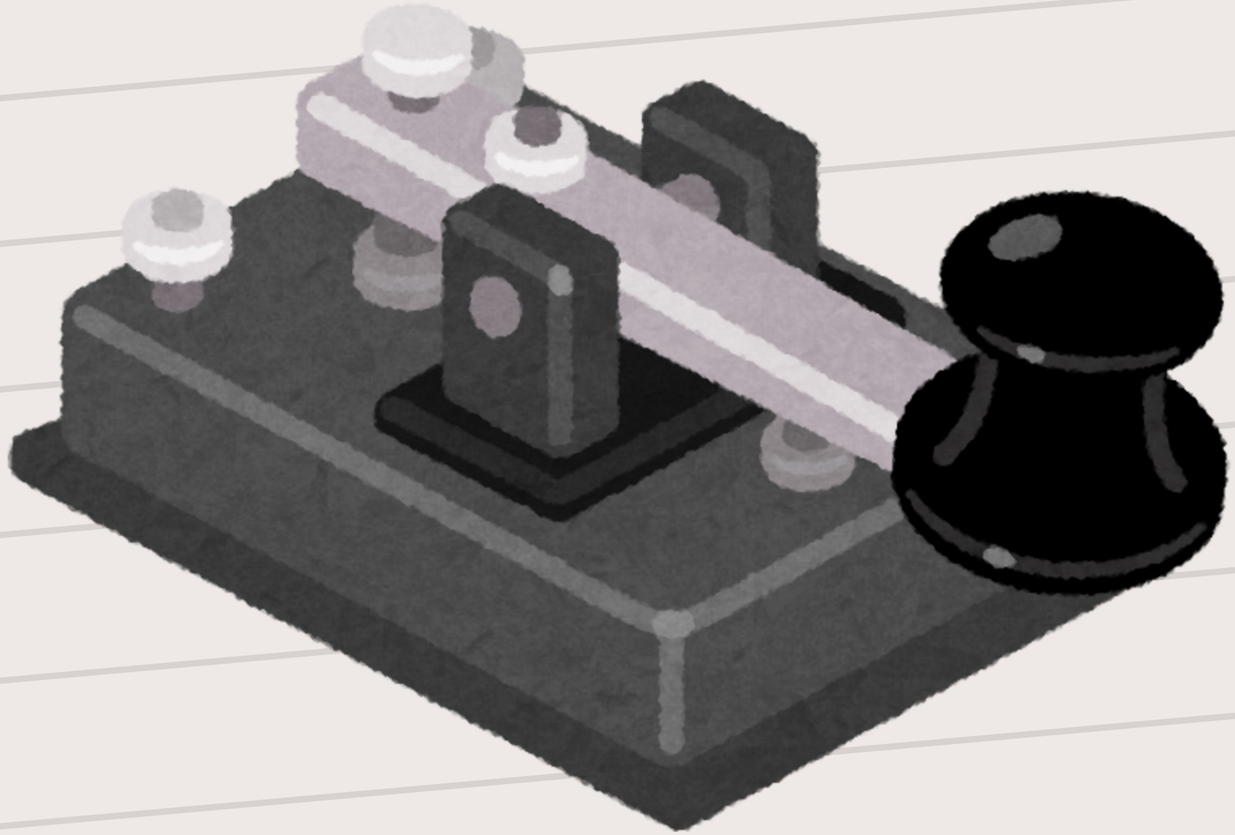
Existing digital Morse code translators have limitations:

- Some are not optimized for speed or large text inputs
- Lack of efficiency in handling special characters and real-time translation
- lacks engaging features, such as: sound and visuals

Goal:

- Develop a fast and accurate Morse Code Translator
- Utilize efficient data structures to optimize encoding and decoding
- Introduce features such as sound and light mediums as well as complementary features.

DEMO



Morse Code Translator

Feature

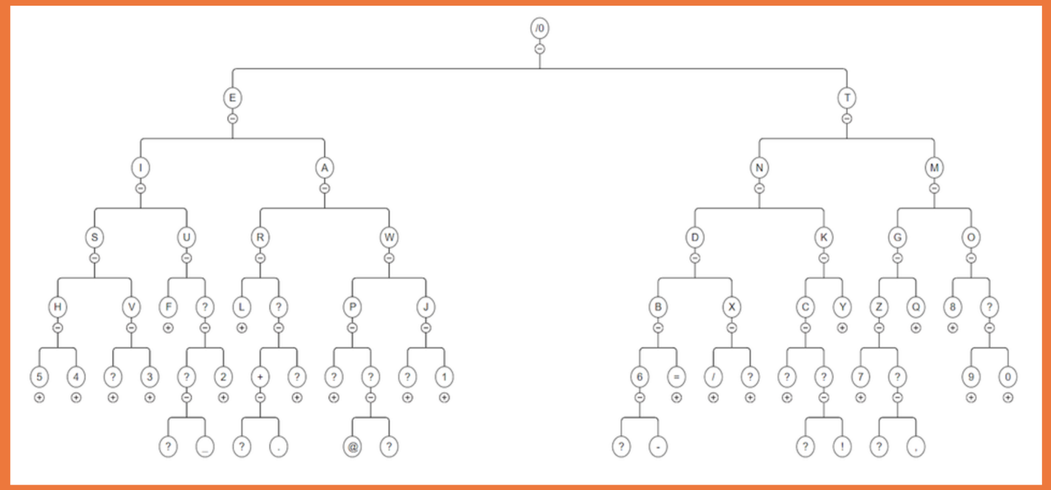
- Text to Morse: Converts text to morse code (Hashmap or Binary Tree)
- Light Medium Translation: Translates Morse code into flashing light signals.
- Sound Output: Plays Morse code as audible beeps for better recognition.
- History: For user to see the previous translation (JList & Listmodel)
- Import: Users can import files to directly translate it to morse code. (JFileChooser)

Data Structure

- Hashmap (Dictionary)
- Binary Tree
- ListModel

Data Structures (Translation)

Binary Tree



Hashmap

```
public void morse_dict() {
   // letters
   morse_map.put('A', ".-");
   morse_map.put('B', "-...");
   morse_map.put('C', "-.-.");
   morse_map.put('D', "-..");
   morse_map.put('E', ".");
   morse_map.put('F', "..-.");
   morse_map.put('G', "--.");
   morse_map.put('H', "....");
   morse_map.put('I', "..");
   morse_map.put('J', ".---");
   morse_map.put('K', "-.-");
   morse_map.put('L', ".-..");
   morse_map.put('M', "--");
   morse_map.put('N', "-.");
   morse_map.put('O', "---");
   morse_map.put('P', ".--.");
   morse_map.put('Q', "--.-");
   morse_map.put('R', ".-.");
   morse_map.put('S', "...");
   morse_map.put('T', "-");
   morse_map.put('U', "..-");
   morse_map.put('V', "...-");
   morse_map.put('W', ".--");
   morse_map.put('X', "-..-");
   morse_map.put('Y', "-.--");
   morse_map.put('Z', "--..");

   // numbers
   morse_map.put('0', "-----");
   morse_map.put('1', ".----");
   morse_map.put('2', "..---");
   morse_map.put('3', "...--");
   morse_map.put('4', "....-");
   morse_map.put('5', ".....");
   morse_map.put('6', "-....");
   morse_map.put('7', "--...");
   morse_map.put('8', "---..");
   morse_map.put('9', "----.");

   //special chars
   morse_map.put('.', ".-.-.-");
   morse_map.put(',', "--..--");
   morse_map.put('?', "..--..");
   morse_map.put('!', "-.-.--");
   morse_map.put('-', "-....-");
   morse_map.put('/', "-..-.");
   morse_map.put('@', ".--.-.");
   morse_map.put('=', "-...-");
   morse_map.put(' ', "/"); //slash
   as space
   morse_map.put('[', "-.--.");
  morse_map.put(']', "-.--.-");
   morse_map.put('{', "-.--.");
   morse_map.put('}', "-.--.-");
   morse_map.put('+', ".-.-.");
   morse_map.put('_', "..--.-");
```

Data Structures (Translation)

Binary Tree

Decoding Algorithm:

Path Traversal: The Morse Code acts as directions, dots to the left and dash to the right.

Encoding Algorithm:

Depth-First Search(DFS): Because we can't use English character to navigate tree path, we search the entire tree to find it. Start it from the root, then search the entire left subtree, and if you still haven't found it yet search the entire right subtree

Hashmap

Decoding Algorithm:

Hashing: Direct lookup, works as a dictionary. When we put "A" it pass it throught a hash function. This function instantly converts the key to the specific memory address or index. After that, it'll store the value ".-" to that specific location.

Encoding Algorithm:

The hash function takes the key and then restore it's value

Data Structures (History)

JList

```
historyList = new JList<>(historyModel);
```

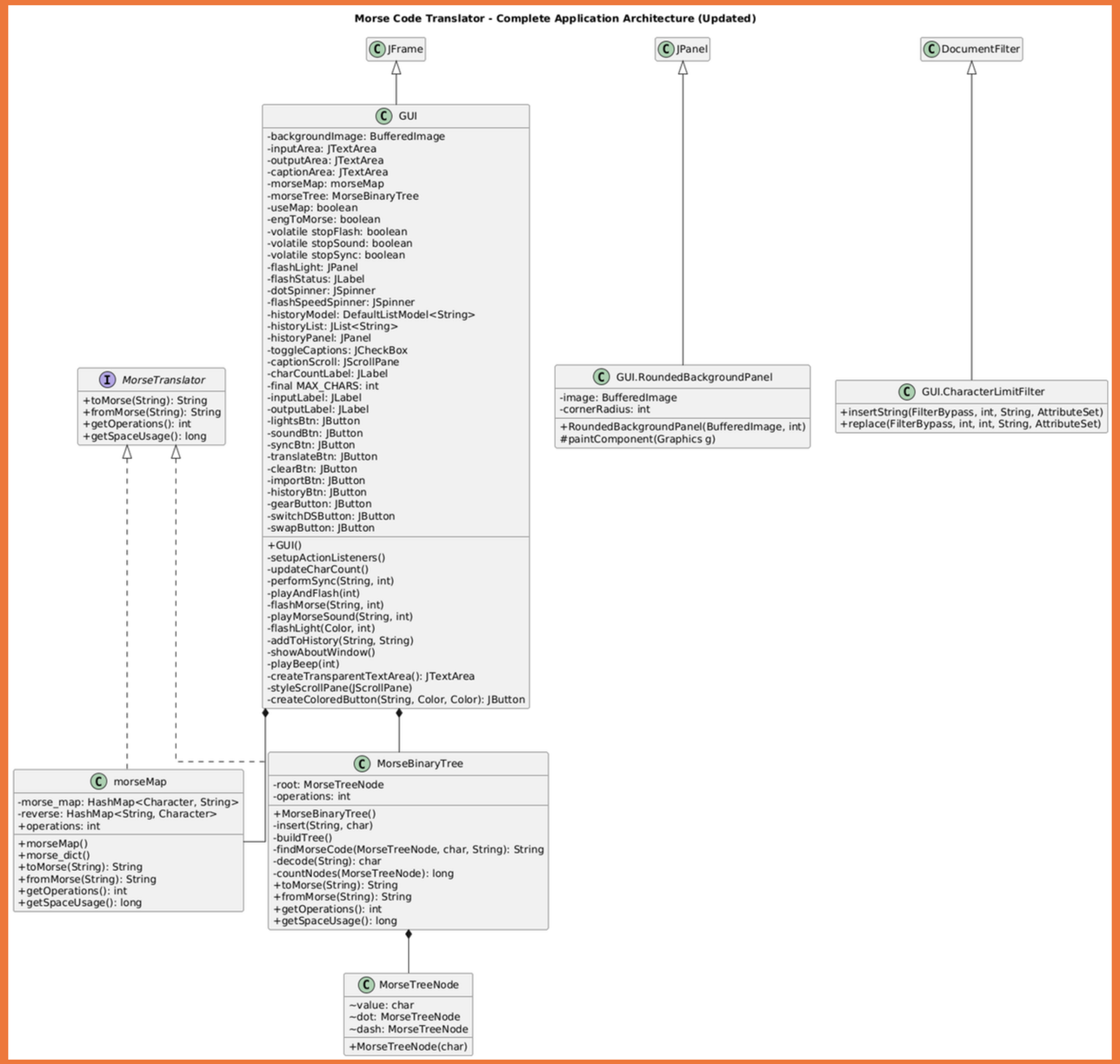
Allows the list to be shown in the GUI from using JList, using the data from the historyModel

ListModel

```
historyModel = new DefaultListModel<>();
```

creates an object historyModel from ListModel that uses a dynamic String list in order to store the actual History data. (The visualization will be handled by the JList)

Class Diagram



Time Complexity

Letter Inputs (A & B)

ENCODE

Table1

Input	Data Structure	Length	Operations	Runtime (Se
None	Both	0	0	0
A	Binary Tree	1	17	0.0012251 s
ABABA	Binary Tree	5	115	0.0019443 s
ABABABABAB	Binary Tree	10	245	0.0010857 s
ABABABABABABABABABAB	Binary Tree	20	490	0.0038853 s
ABABABABABABABABABABAE	Binary Tree	40	980	0.0027194 s
ABABABABABABABABABABAE	Binary Tree	80	1960	0.003701 s
ABABABABABABABABABABAE	Binary Tree	100	2450	0.004671 s

Table2

Input	Data Struct	Length	Operations	Runtime (S
None	Both	0	0	0
A	HashM...	1	1	0.000210 s
ABABA	HashM...	5	5	0.000026 s
ABABABABAB	HashM...	10	10	0.000054 s
ABABABABABABABABABAB	HashM...	20	20	0.000091 s
ABABABABABABABABABABA	HashM...	40	40	0.000076 s
ABABABABABABABABABABA	HashM...	80	80	0.000094 s
ABABABABABABABABABABA	HashM...	100	100	0.000117 s

DECODE

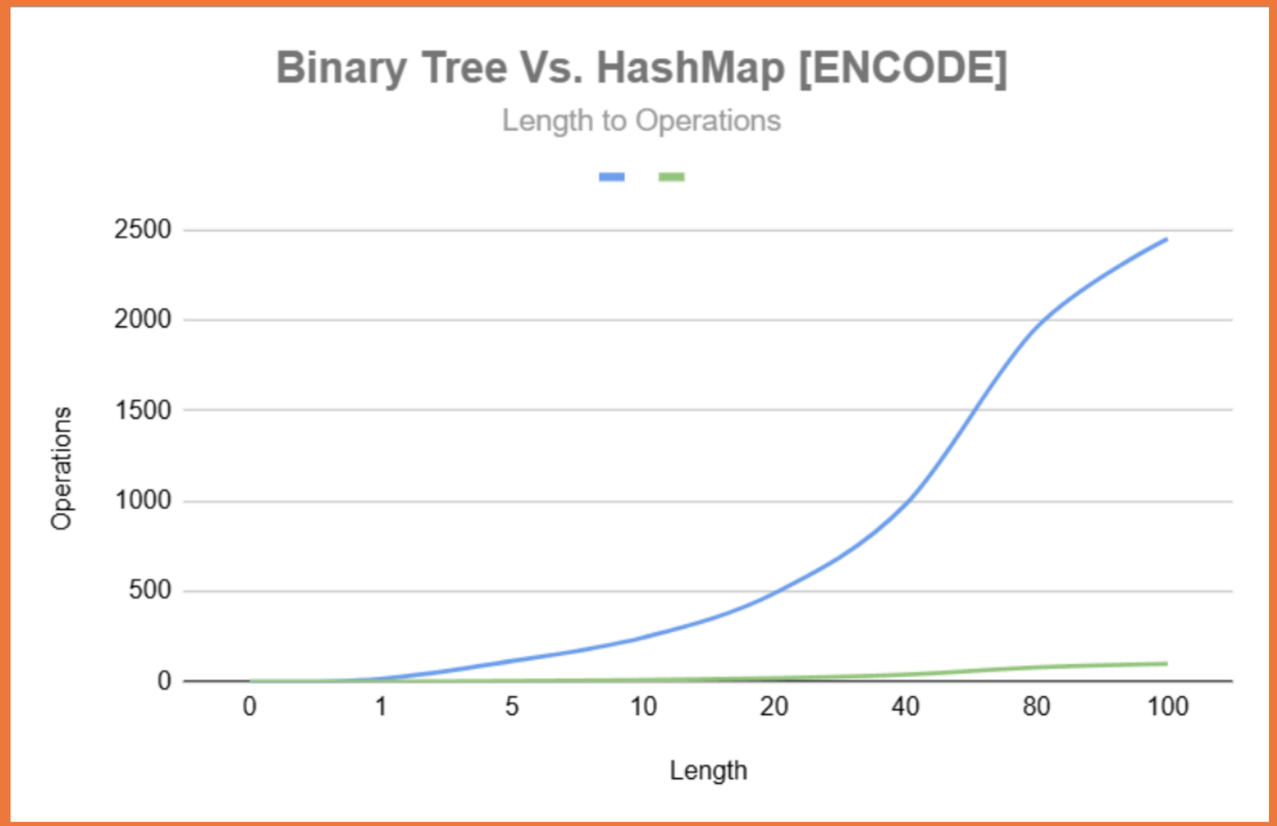
Table3

Input	Data Structure	Length	Operations	Runtime (Se
None	Both	0	0	0
.-	Binary Tree	1	2	0.0000092 s
.-...-...-... .-	Binary Tree	5	14	0.0000209 s
.-...-...-...-...-...-...-...	Binary Tree	10	30	0.0000287 s
.-...-...-...-...-...-...-...-...-...-.	Binary Tree	20	60	0.000045 s
.-...-...-...-...-...-...-...-...-...-.	Binary Tree	40	120	0.00006 s
.-...-...-...-...-...-...-...-...-...-.	Binary Tree	80	240	0.0002113 s
.-...-...-...-...-...-...-...-...-...-.	Binary Tree	100	300	0.0002039 s

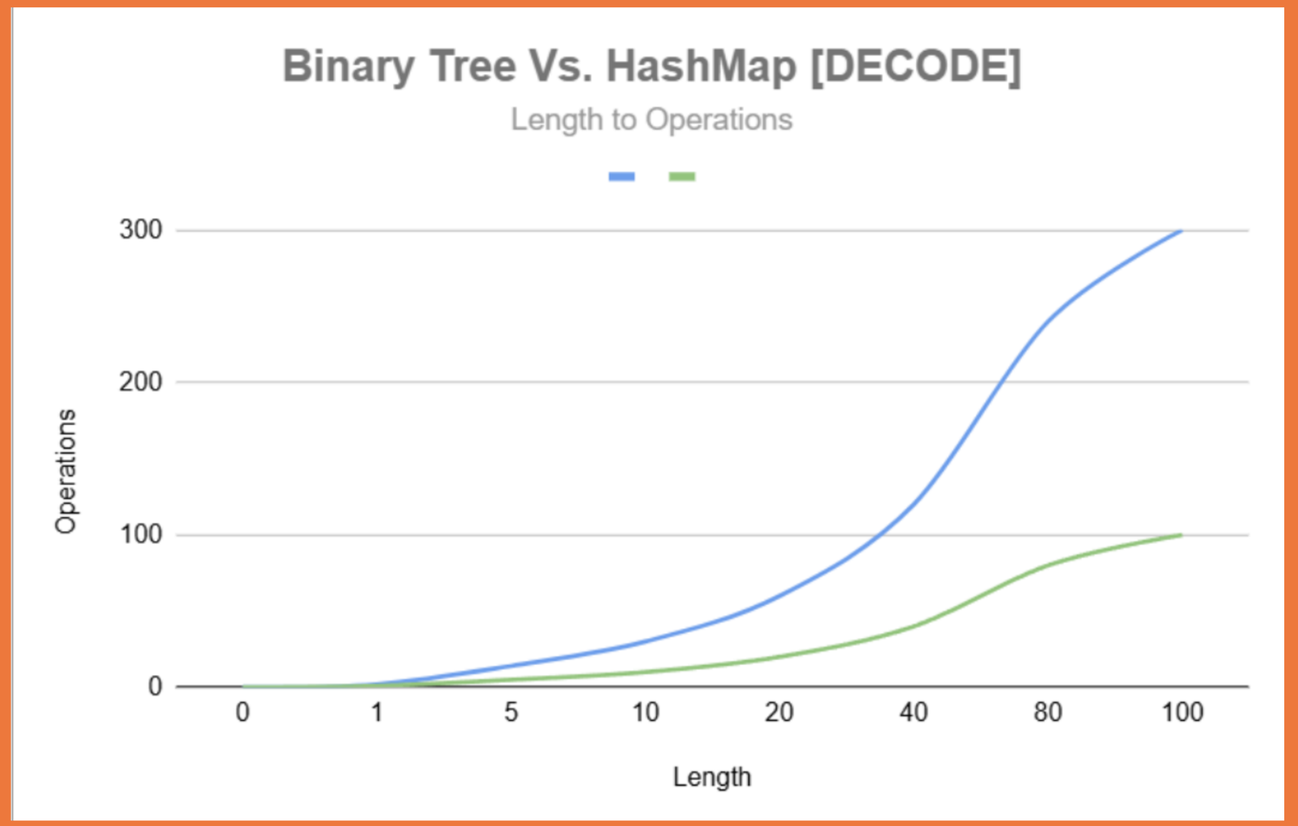
Table4

Input	Data Struct	Length	Operations	Runtime (S
None	Both	0	0	0
.-	HashM...	1	1	0,000072 s
.-...-...-... .-	HashM...	5	5	0,000175 s
.-...-...-...-...-...-...-...	HashM...	10	10	0,000216 s
.-...-...-...-...-...-...-...-...-...-.	HashM...	20	20	0,000269 s
.-...-...-...-...-...-...-...-...-...-.	HashM...	40	40	0,000310 s
.-...-...-...-...-...-...-...-...-...-.	HashM...	80	80	0,000425 s
.-...-...-...-...-...-...-...-...-...-.	HashM...	100	100	0,000518 s

Time Complexity



Time Complexity



Space Complexity

Letter Inputs (A & B)

ENCODE

Table5

Input	Data Structure	Length	Space
None	Both	0	55
A	Binary Tree	1	55
ABABA	Binary Tree	5	55
ABABABABAB	Binary Tree	10	55
ABABABABABABABABABAB	Binary Tree	20	55
ABABABABABABABABABABAE	Binary Tree	40	55
ABABABABABABABABABABAE	Binary Tree	80	55
ABABABABABABABABABABAE	Binary Tree	100	55

Table7

Input	Data Structure	Length	Space
None	Both	0	100
A	HashMap	1	100
ABABA	HashMap	5	100
ABABABABAB	HashMap	10	100
ABABABABABA	HashMap	20	100
ABABABABABA	HashMap	40	100
ABABABABABA	HashMap	80	100
ABABABABABA	HashMap	100	100

DECODE

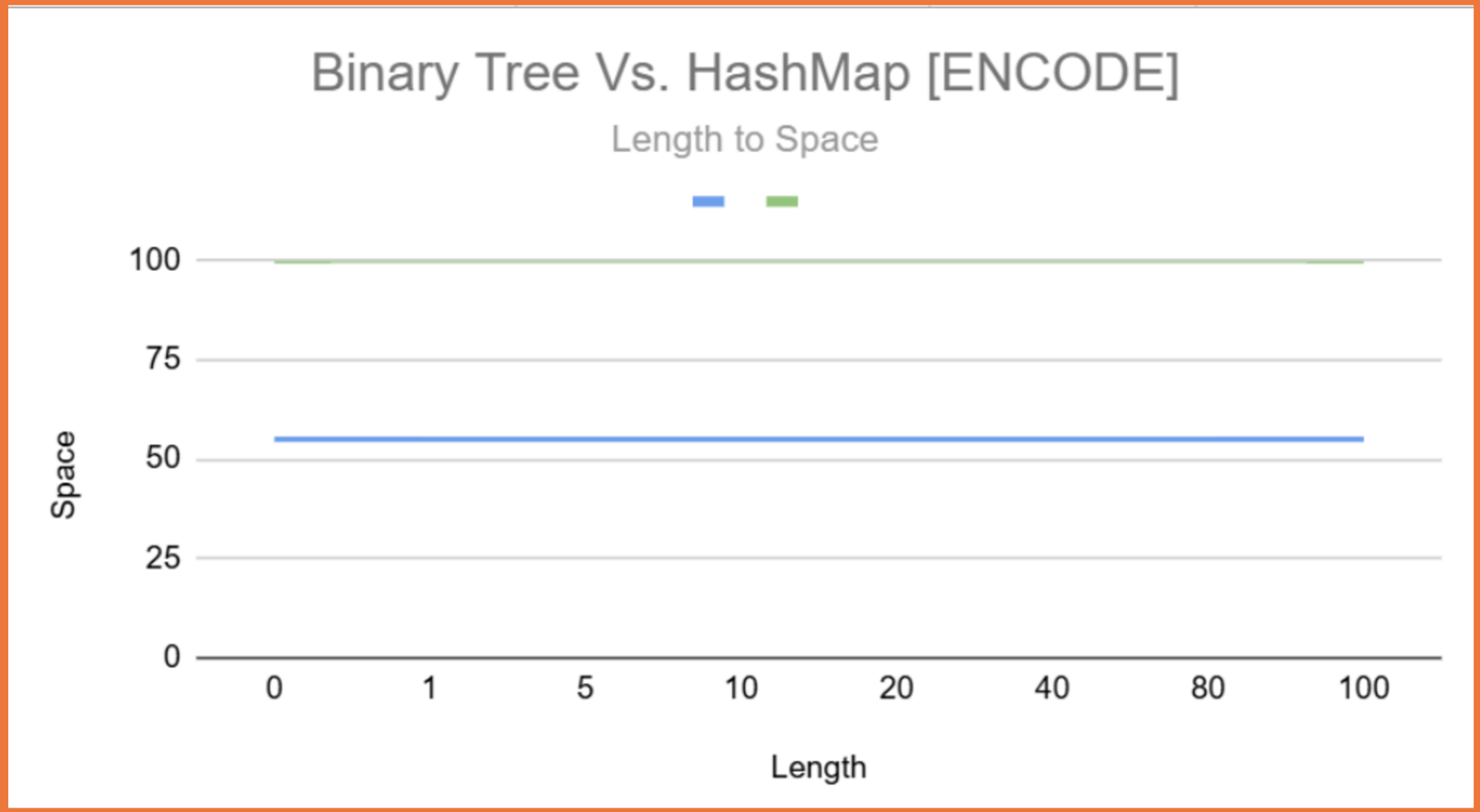
Table8

Input	Data Structure	Length	Space
None	Both	0	55
.-	Binary Tree	1	55
.--... .--... .-	Binary Tree	5	55
.--... .--... .--... .--... .--...	Binary Tree	10	55
.--... .--... .--... .--... .--... .--... .--.	Binary Tree	20	55
.--... .--... .--... .--... .--... .--... .--.	Binary Tree	40	55
.--... .--... .--... .--... .--... .--... .--.	Binary Tree	80	55
.--... .--... .--... .--... .--... .--... .--.	Binary Tree	100	55

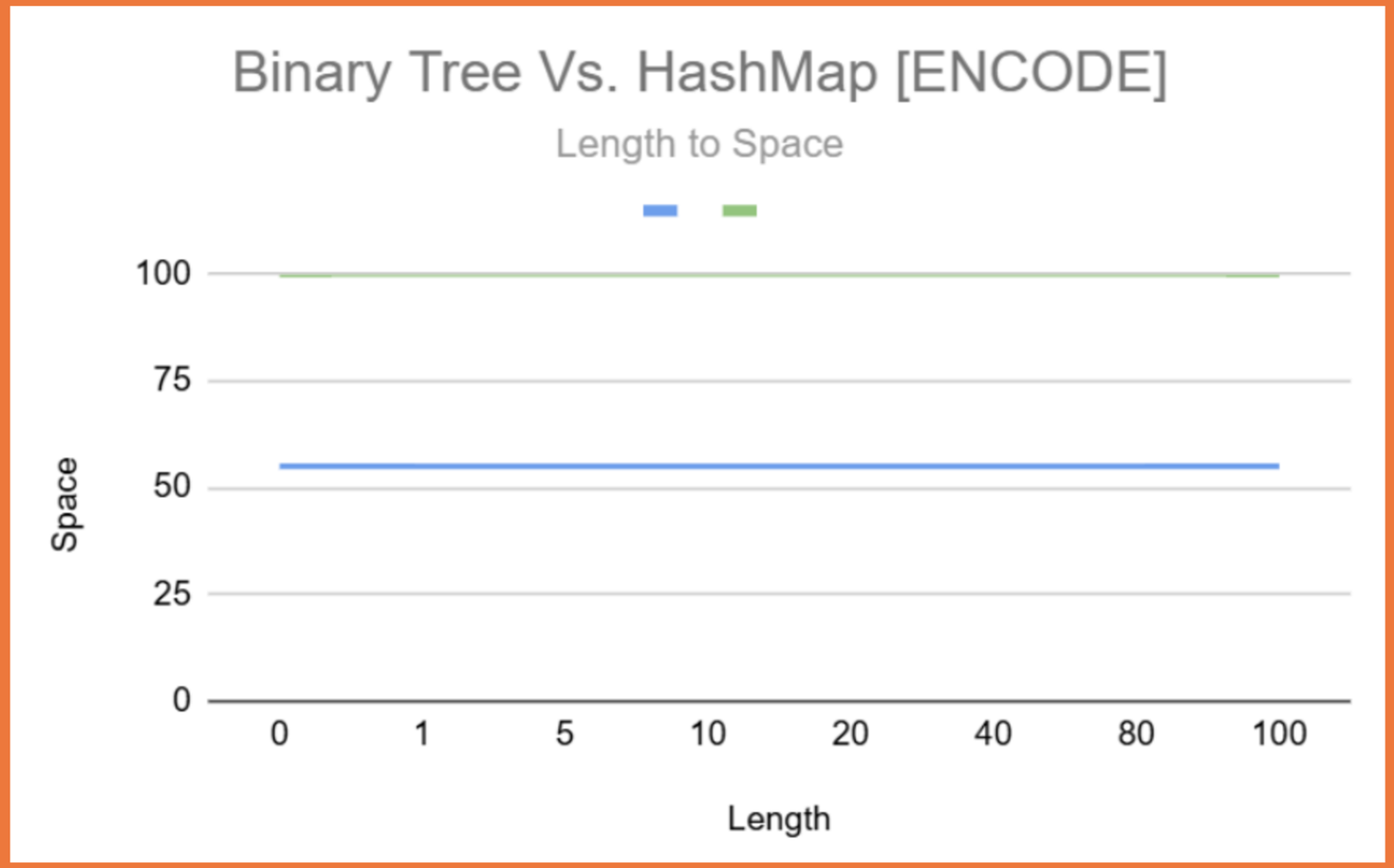
Table6

Input	Data Structure	Length	Space
None	Both	0	100
.-	HashMap	1	100
.--... .--... .-	HashMap	5	100
.--... .--... .--... .-	HashMap	10	100
.--... .--... .--... .-	HashMap	20	100
.--... .--... .--... .-	HashMap	40	100
.--... .--... .--... .-	HashMap	80	100
.--... .--... .--... .-	HashMap	100	100

Space Complexity



Space Complexity



Conclusion

Thank You