REPORT

Abstract:

Design and implement a C library for dynamic table with insert and delete functionalities. Analyse the time taken for each operation by varying the growth factor and arrive at an optimal number.

Objectives

The following functions to be implemented for dynamic table.

a) Initialize a dynamic table.
b) Insert element into end of the array
c) Delete last element of the array.

Analyse the following by plotting time vs elements for:

a) Inserting 1,000,000 elements by varying resizing factor.
b) Deleting 1,000,000 elements by varying resizing factor.

Write a demo program to demonstrate the functionalities of the library.

Approach:

The following steps were followed:

a) Creation of a C structure.
b) Design the structure of the class.
c) Implement the functions (insert, delete and resize).
d) Make a main file to test the functionalities of the structure.
e) Creating a makefile to execute the program.
f) Make a python script to automate the testing process.

Assumptions:

1. The table supports only integer values.
2. The maximum size supported by the table must not exceed the size of the systems long int.

Git Link: https://github.com/DarkFate13/Dynamic-Table

Observations:

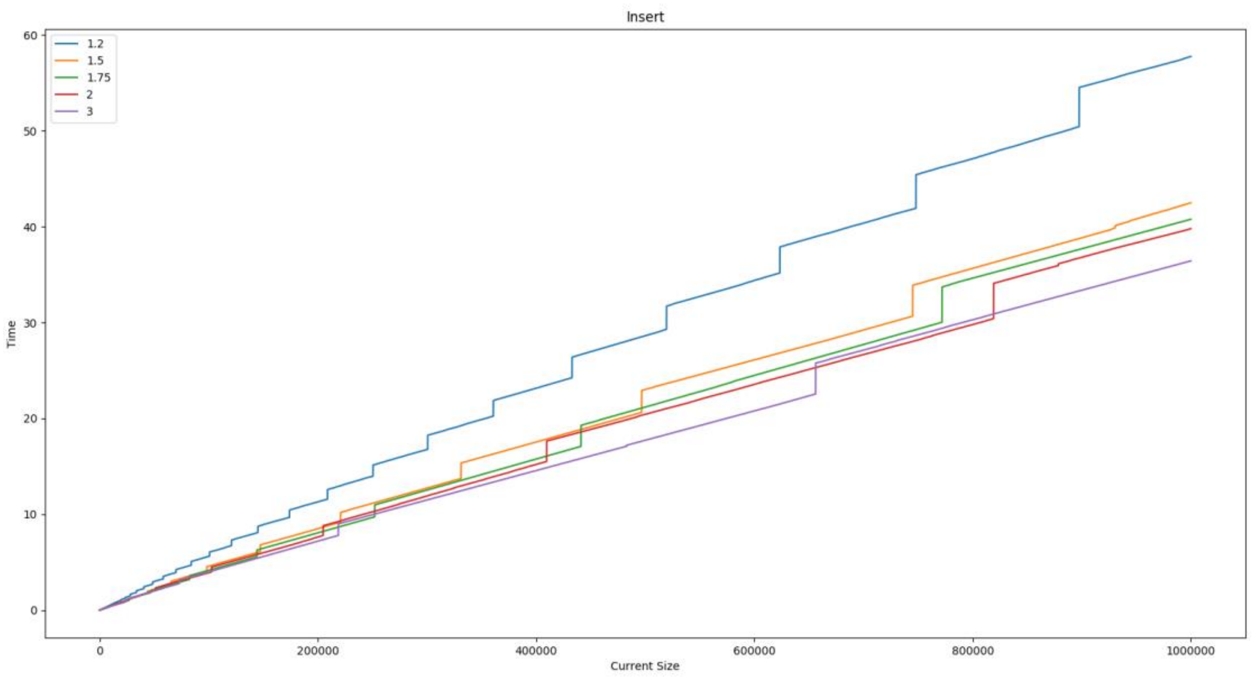

Figure 1 One million Insertions

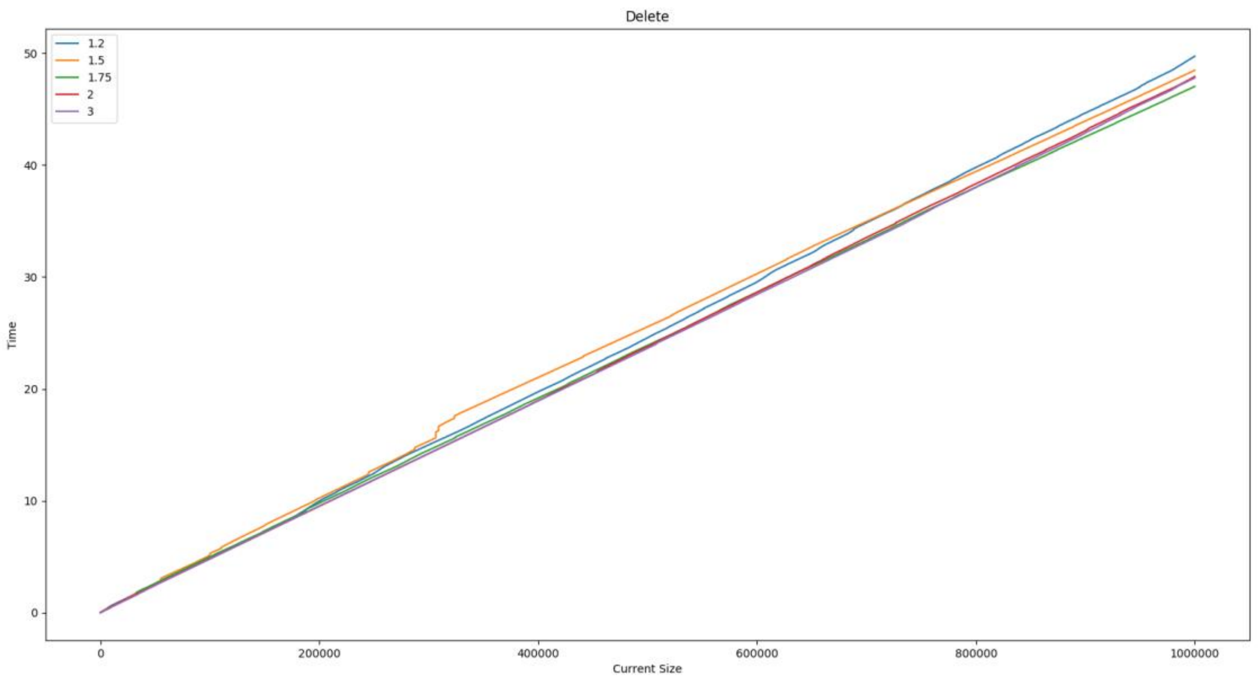

Figure 2 One million Deletions

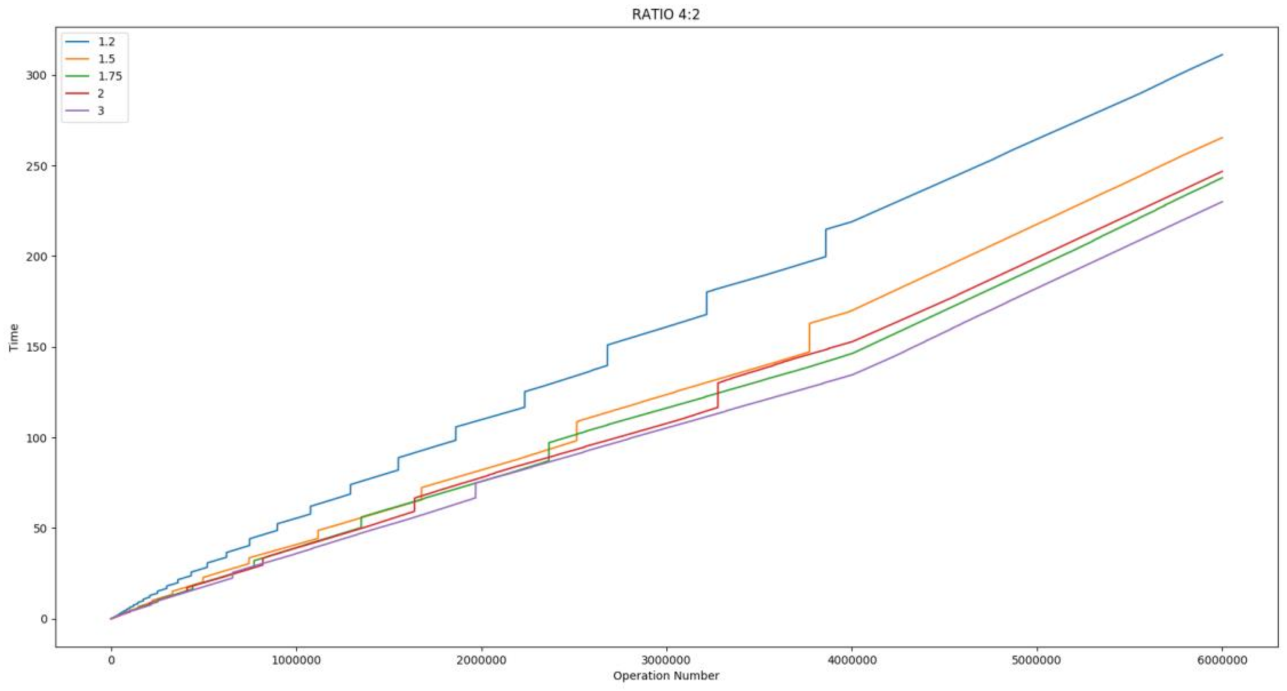

Figure 3 Four Million Insertions followed by two Million deletions

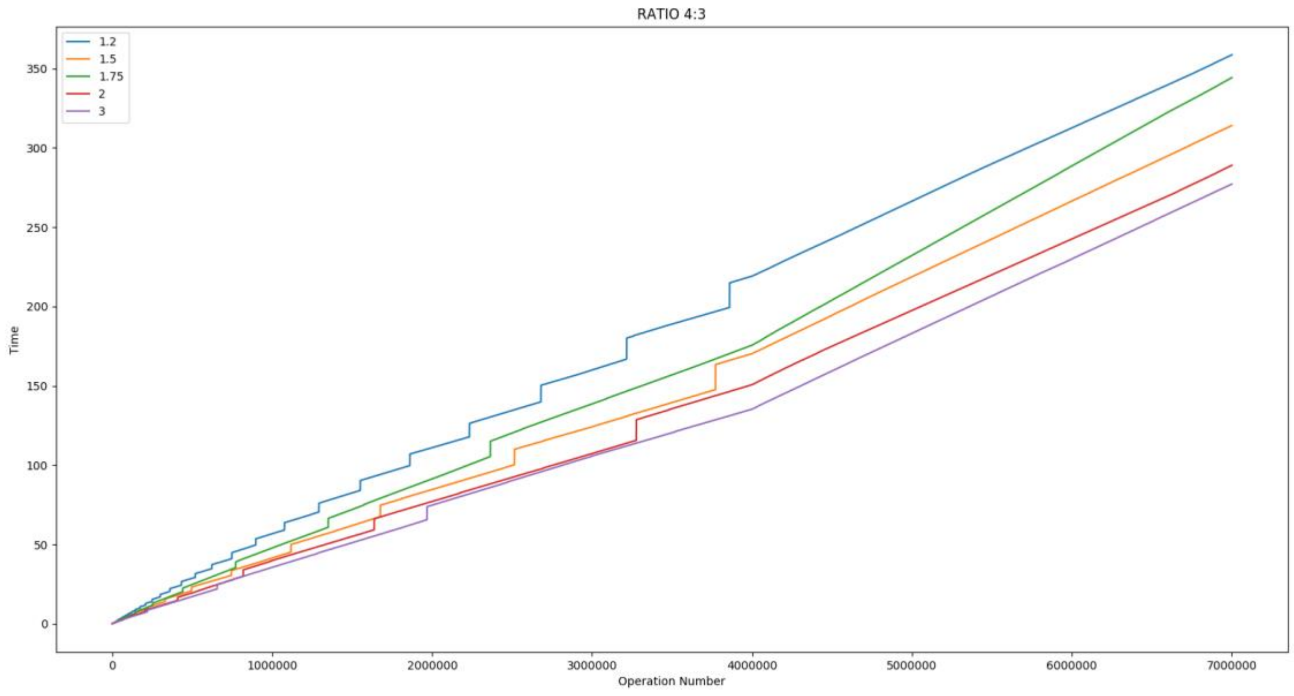

Figure 4 Four Million Insertions followed by three Million deletions

Result Analysis:

Maximum Time table (ms)

Operation	Factor – 1.2	Factor – 1.5	Factor – 1.75	Factor – 2	Factor – 3
Insertions	4.075924	3.221704	3.661241	3.674268	**3.215994**
Deletion	0.502145	**0.052824**	0.177969	0.062437	0.065113
4:2	14.97553	9.948075	12.935813	**8.140377**	12.935813
4:3	15.438233	9.623128	15.768057	**8.200089**	12.935813

From the above table we can conclude that:

1. Factor of 3 is optimal for large number of insertions but is mainly due to low number of element copies as an enormous amount of space is used when resizing.
2. Factor of 1.5 is optimal for large number of deletions.
3. Factor of 2 is optimal for combinations of these operations as it efficiently manages resizing operations by not allocating too much or too less space and resizing not too often or too frequently.

Average Time table (ms)

Operation	Factor – 1.2	Factor – 1.5	Factor – 1.75	Factor – 2	Factor – 3
Insertions	0.000564046	0.000417237	0.000390093	**0.000372383**	0.000395681
Deletion	0.000469895	0.000475060	**0.000453177**	0.000473454	0.000476701
4:2	0.000518011	0.000438505	0.000401997	**0.000377867**	0.000399869
4:3	0.000514205	0.000433867	0.000410443	**0.000392373**	0.000407926

From the above table we can conclude that:

1. Factor of 2 is optimal for large number of insertions as well as for the combination of the operations as it efficiently manages resizing operations by not allocating too much or too little space and resizing not too often or too frequently.
2. Factor 1.75 is optimal for deletions.

Conclusion:

The optimal resizing factor for this implementation of dynamic table is 2. Though a very big factor (bigger than 2) can be faster at certain cases, a lot space is consumed and remains unused for large periods of time. At the same time, a low resizing factor tends to call resize() multiple number of times which leads to slowing down of the program. Thus, 2 provides the best of both worlds and proves to be optimal.