Task 07 - Spike Summary Report

Spike: Task_07
Title: Performance Measurement
Author: Thomas Horsley, 103071494

Goals & Deliverables

Aim: Develop the skillset to collect and analyze performance metrics programmatically.

Deliverables:

- Spike Summary Report
- Functional code
- Data and supporting conclusions

Technology, Tools and Resources

Tech and Tools

The project was scripted in C++ 17 using Visual Studio Community 2022.

UML's and charts are made with *www.Lucidchart.com*

Source control is handled using Git.

Resources

- Echo360 Lectures "*3.1 Code Performance*"
- Vector Performance Management: https://www.acodersjourney.com/6-tips-supercharge-cpp-11-vector-performance/

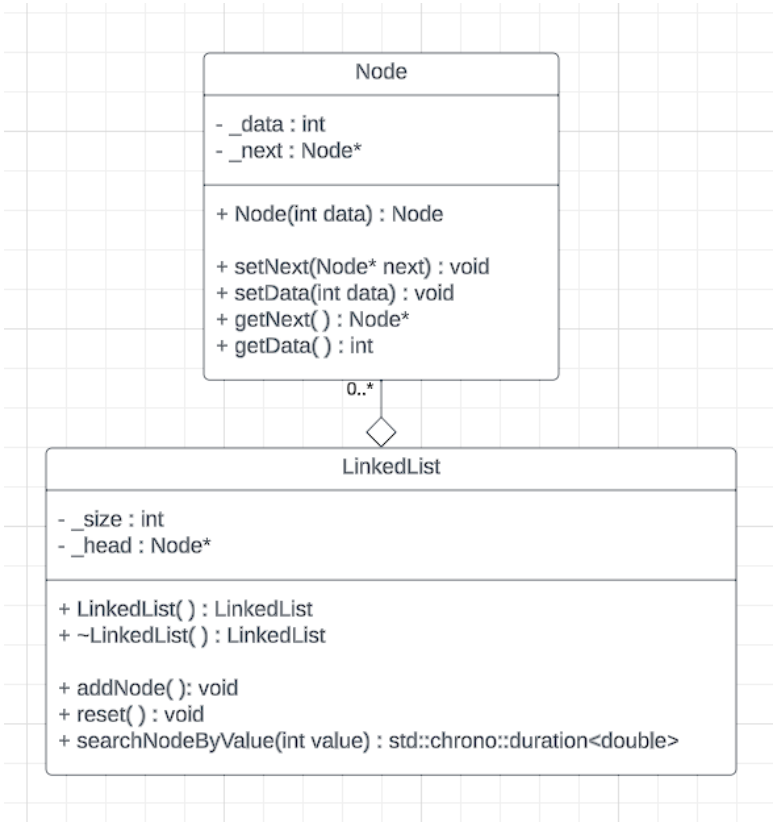
Task 07 - Spike Summary Report

Tasks Undertaken

Planning

> 💡 I wanted to take a closer look at the “overhead” referred to when discussing a standard template vector search. To measure this value and ensure the tests are consistent, tests will be ran on the standard template unordered_map and a custom linked list data structure.

Function Descriptions and Notes



Linked list UML

This Linked List structure contains a searchNodeByValue( ) method which returns a duration<double>, this is the time it took to complete the search.

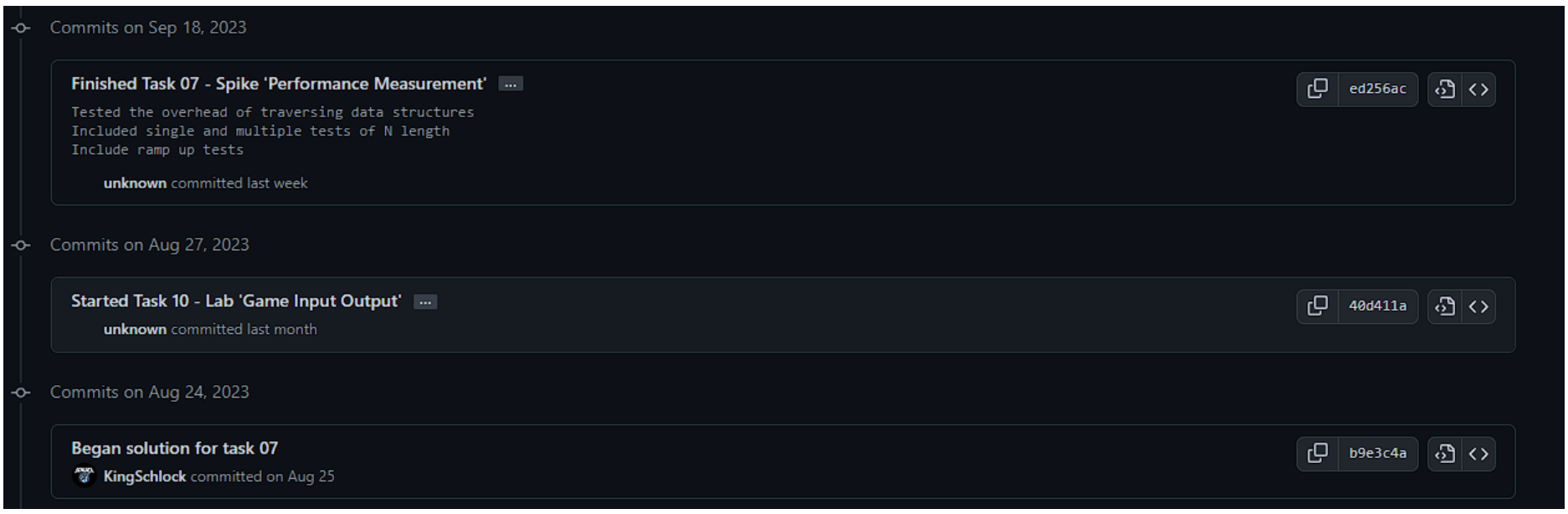
```
52    // This is the function which needs to be timed.
53    std::chrono::duration<double> searchNodeByValue(int value) {
54        int idx = 0;
55        int search_value = value + 1;
56
57        auto start = std::chrono::steady_clock::now();
58        Node* traversal_ptr = _head;
59
60        while (traversal_ptr->getNext() != nullptr) {
61            if (traversal_ptr->getData() == search_value) {
62                auto end = std::chrono::steady_clock::now();
63                return end - start;
64            }
65
66            idx++;
67            traversal_ptr = traversal_ptr->getNext();
68        }
69
70        if (traversal_ptr->getNext() == nullptr &&
71            traversal_ptr->getData() == search_value) {
72            auto end = std::chrono::steady_clock::now();
73            return end - start;
74        }
75
76        auto bad_end = std::chrono::steady_clock::now();
77        return start - bad_end; // Neg value for err
78    }
```

Test Name	Description	Implementation Plan
Single Structure Search	Tests each data structures search, records the results and returns a `vector<duration<double>>`. This is the only test method which returns a value as it’s required by other methods. Rendering can be toggled on or off.	Requires 3 functions, each of which tests a data structure search and returns a `duration<double>` (think `searchNodeByValue()`). whatever data these functions require must be taken by the `SingleStructureSearch()` and passed to test functions. return `vector<duration<double>>` and display the results.
Many Random Value	Calculates and displays the average time of multiple single test values. runs the `SingleStructureSearch()` multiple times, passing random search values at random positions to be found. Each set of test data is held within the a `vector<vector<duration<double>>>` structure.	Covered by description.
Ramp Up	Will run multiple SingleStructureSearch( ), with each methods search value being assigned to an element further from the structures iterators entry point upon each	Be sure to convert from scientific notation to nanoseconds for an easier read!

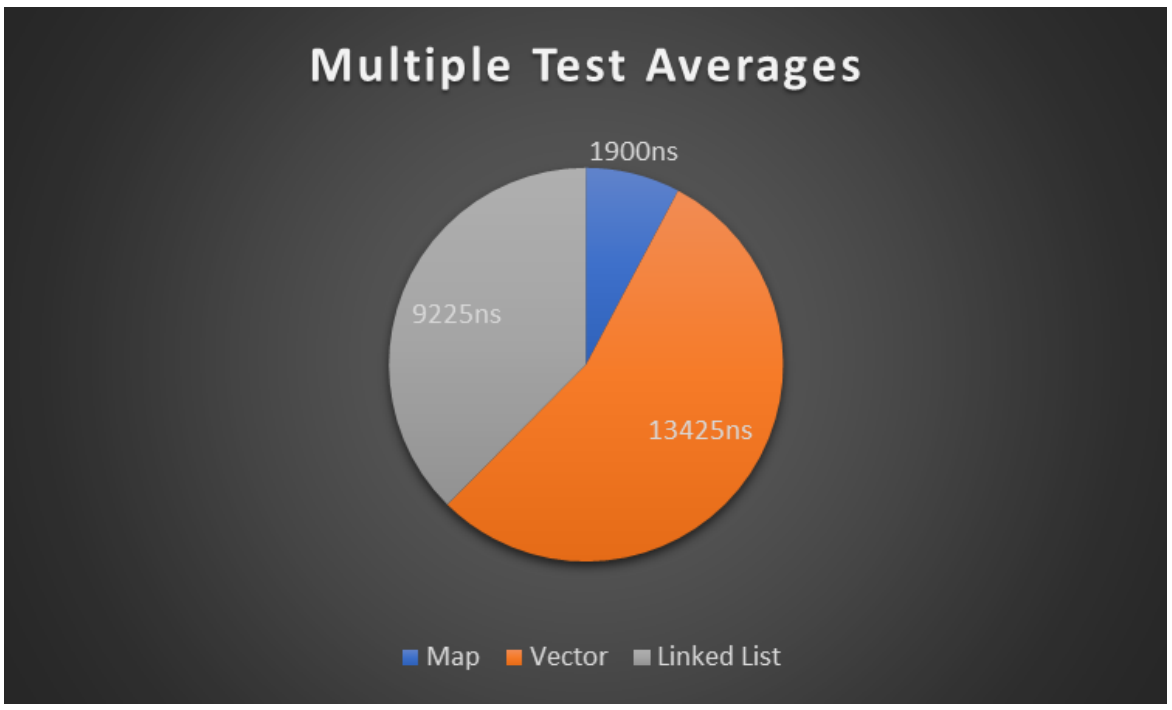
Test Name	Description	Implementation Plan
	ramp. These values will be recorded and displayed.	

Implementation

Git Commit History

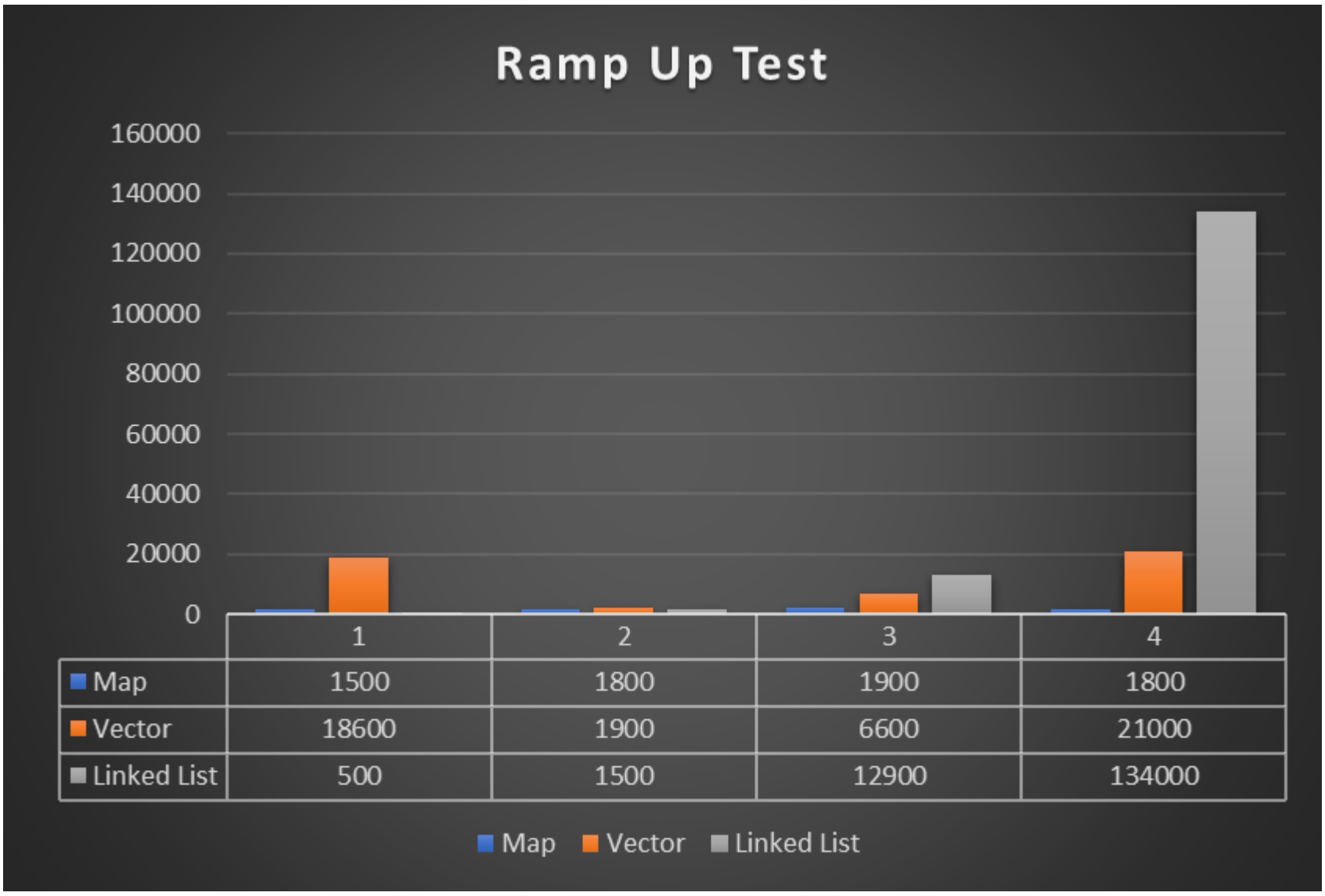


Results & Discussion



The the left, the average results of a many search test. This test was iterated 10 times with a maximum search depth of 1000 elements.

It is clear from the graph that the unordered map is the fastest for accessing elements of a large dataset.



Test 1

- Search for the 10th element

These results were curious to say the least, though the test environment was ran multiple times, Vector searches up to element 10 always came back exceptionally slow. The reason for this outlier is believed to be a measuring error given the incredibly short length of time observed.

Additionally, our Forward Linked List was found to be 3x faster than the standard map implementation for small searches. These results can be explained by the maps “search” overhead, for which the linked list has noticeably less of.

Test 2

- Search for the 100th element

Up to around 100 elements, all of the data structures had very similar search performances. The vector’s time to search was reduced by 89.8% and scaled as expected from here.

Test 3

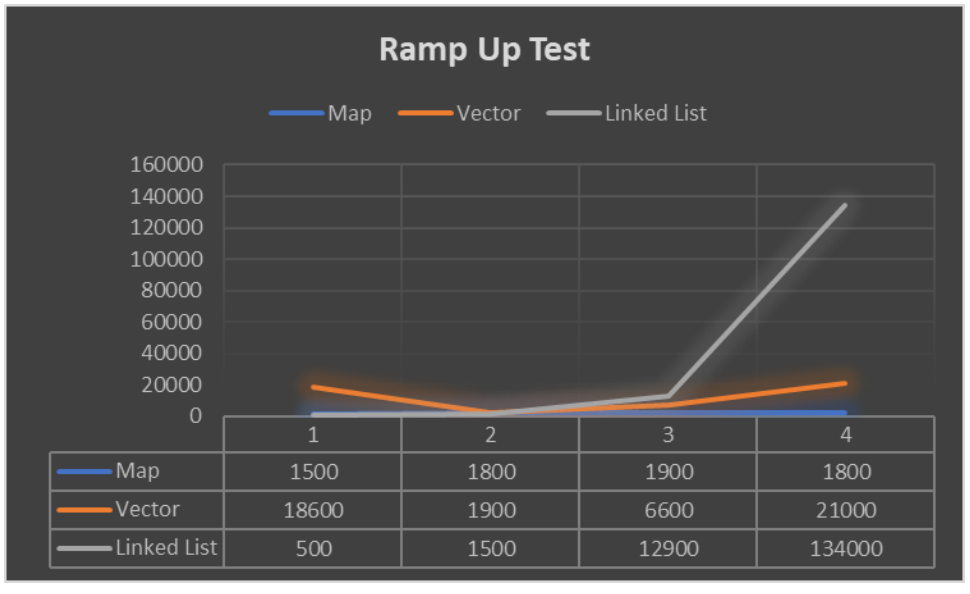
- Search for the 1,000th element

At sizes greater than 500 it becomes clear that the initial overhead involved with a map search is much faster than iterating through any structure to find a specific element. Therefore, for large datasets maps should always be considered first.

Test 4

- Search for the 10,000th element

Test 4 further compounded on the data provided by iteration 3. We saw Linked list search times bloat to over 6x that of the vector and nearly 75x slower than the unordered map (whose search time has been consistent since test 1).



Trend line perspective on data structure searches.

What was Learned?

💡 This spike taught me how to test data structures quickly, compile documentation on those tests and share findings with others. Additionally, I learnt about the overhead in different searches and at which point that overhead becomes negligible relative to the iterative search costs.