

**PA4 Report**

By

**Louis Vinton**

**Wes Beard**

**CSI 281**

**Data Structures & Algorithms**

**Fall 2019-20**

1. **Introduction**

In this experiment we are determining which list structure: arrays, linked lists, and doubly linked lists, is the most efficient in terms of memory usage and speed.

1. **Background**

**Array:** Can be randomly accessed, no way to change size after compilation. Does not inherently sort data as it is added[1].

**Linked List:** Cannot be randomly accessed, size can change after compilation, only can be traversed forwards. It is generally more flexible than an array but at the cost of memory and code size[1].

**Doubly Linked List:** The same as linked list but has the ability to point both forward and backward within the list, this has added efficiency and speed at the cost of memory[1].

1. **Experiment Plan**

* We are primarily trying to determine which data structure is the fastest, but also gauge each structure on the memory it uses to complete each task.
* Three different list sizes will be tested: 1000, 10000, 100000.
* The program will test each data structure using the following tests: adding random numbers to the list, deleting data from the list, and finding data in a list. Each test will be run three times to create an average time for each.
* The time it takes for each process to complete will be determined by a built-in timer.
* The memory size will be determined by a rough calculation of the data structure. For example: array size(in bytes) = (size of data \* length of array)

linked list size(in bytes) = (size of data + pointer) \* (length of list)

doubly linked list size(in bytes) = [size of data + (pointer \* 2)] \* (length of list)

1. **Experimentation Detail**
   1. 16Gb DDR4 @ 2133Ghz
   2. i7 6700k 4 Core/8 Thread
   3. 4.2Ghz
   4. Windows 10 64 bit

**Summarized Data**

|  |  |  |  |
| --- | --- | --- | --- |
| Array | 1000 | 10000 | 100000 |
| Add | 2.43e-04 | 1.54e-03 | 1.26e-02 |
| Find | 2.81e-02 | 3.13e-02 | 3.55e-02 |
| Delete | N/A | N/A | N/A |

|  |  |  |  |
| --- | --- | --- | --- |
| Linked List | 1000 | 10000 | 100000 |
| Add | 3.93e-04 | 3.35e-03 | 3.26e-02 |
| Find | 5.02e-02 | 5.36e-02 | 5.63e-02 |
| Delete | 8.20e-02 | 8.46e-02 | 8.68e-02 |

|  |  |  |  |
| --- | --- | --- | --- |
| Double Linked List | 1000 | 10000 | 100000 |
| Add | 4.93e-02 | 5.14e-02 | 5.41e-02 |
| Find | 4.93e-02 | 5.14e-02 | 5.41e-02 |
| Delete | 7.72e-02 | 7.99e-02 | 8.20e-02 |

1. **Discussion and Conclusion**

From our results we can see that it is much faster to add data to an Array as the memory is already reserved and the random access makes it much quicker to input the data into the list. Surprisingly Doubly Linked list was the second fastest. Array also was the fastest for finding numbers in the list. Binary search is significantly faster for the size of the lists that were used in the experiment. Surprisingly Doubly linked list was only minimally faster that Linked list even though Doubly Linked list implemented a binary search. Array did not have any results for deletion since in the experiment the array was not being resized. This would make the speed for deletion essentially the same as the find results. Doubly Linked list was faster at deletion than Linked list. This is interesting as they are implemented in virtually the same way.

Not surprisingly Array was faster at every task. Doubly Linked list was slightly faster than Linked list. In terms of memory Array was also the winner with Linked list in second and Doubly Linked list in third.

1. **References**

[1]GeeksforGeeks. (2019). *Linked List vs Array - GeeksforGeeks*. [online] Available at: https://www.geeksforgeeks.org/linked-list-vs-array/ [Accessed 26 Oct. 2019].

1. **Appendix**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array Add | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 2.42e-04 | 2.58e-04 | 2.29e-04 | 2.43e-04 |
| 10000 | 1.43e-03 | 1.68e-03 | 1.51e-03 | 1.54e-03 |
| 100000 | 1.30e-02 | 1.15e-02 | 1.33e-02 | 1.26e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array Find | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 1.57e-02 | 2.81e-02 | 4.07e-02 | 2.81e-02 |
| 10000 | 1.77e-02 | 3.11e-02 | 4.51e-02 | 3.13e-02 |
| 100000 | 2.21e-02 | 3.62e-02 | 4.83e-02 | 3.55e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linked List Add | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 5.36e-04 | 2.60e-04 | 3.85e-04 | 3.93e-04 |
| 10000 | 4.00e-03 | 3.21e-03 | 2.84e-03 | 3.35e-03 |
| 100000 | 3.45e-02 | 3.10e-02 | 3.23e-02 | 3.26e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linked List Find | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 3.95e-02 | 5.05e-02 | 6.07e-02 | 5.02e-02 |
| 10000 | 4.14e-02 | 5.40e-02 | 6.56e-02 | 5.36e-02 |
| 100000 | 4.46e-02 | 5.64e-02 | 6.80e-02 | 5.63e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Linked List Delete | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 7.29e-02 | 8.25e-02 | 9.07e-02 | 8.20e-02 |
| 10000 | 7.54e-02 | 8.48e-02 | 9.36e-02 | 8.46e-02 |
| 100000 | 7.77e-02 | 8.68e-02 | 9.61e-02 | 8.68e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Doubly Linked List Add | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 3.41e-04 | 3.58e-04 | 2.60e-04 | 3.19e-04 |
| 10000 | 1.86e-03 | 2.11e-03 | 2.17e-03 | e 2.04e-03 |
| 100000 | 2.23e-02 | 2.19e-02 | 1.95e-02 | 2.12e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Doubly Linked List Find | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 4.03e-02 | 4.94e-02 | 5.84e-02 | 4.93e-02 |
| 10000 | 4.29e-02 | 5.12e-02 | 6.03e-02 | 5.14e-02 |
| 100000 | 4.51e-02 | 5.45e-02 | 6.28e-02 | 5.41e-02 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Doubly Linked List Delete | Experiment 1 | Experiment 2 | Experiment 3 | Average |
| 1000 | 6.83e-02 | 7.76e-02 | 8.58e-02 | 7.72e-02 |
| 10000 | 7.14e-02 | 7.97e-02 | 8.88e-02 | 7.99e-02 |
| 100000 | 7.34e-02 | 8.15e-02 | 9.12e-02 | 8.20e-02 |