Matthew Laderoute

List vs Vector

Program Design

The challenge was to create a container that would be ordered when an element of a random integer value was inserted and then remove elements at a random index until the container emptied. Calling the STL `sort()` function was restricted. We could not use `std::advance()`. Nor could duplicate values be inserted into the container we would build. To overcome those challenges, I implemented several different functions to carry out these task.

Insertion

I created a function which handled the insertion and the ordering of the insertion. It was generic, and meant to handle Sequence Containers via usage of their iterators. I'm not sure if it's implementation is the most efficient --considering for sufficiently large N values (N > 200,000) , the program took in excess of 20 mins to handle those values. (Also, the deletion function may have slowed the program down, considering I coupled them in the timer function)

Deletion

I created a generic function which picks a random index value, and uses iterators to move to that index and delete the element at that location. It is able to handle all the types I tested. Vector, List, Map, Deque.

Experiment Design

Because of limitations with computing power in the Windows Subsystem for Linux, completing all the experiments with multiple trials on N >= 200,000 proved to be a challenge. Given the challenge to create one container of N>=200,000 the program can produce output, --however, for comparison's sake I capped the size of N at 100,000.

I chose to carry out 5 separate experiments with 3 trials per experiments for N values **N**={1000, 2000, 4000, 8000, 16000, 32000, 64000, 100000}

For the first 3 experiments I used the containers in the assignment instructions:

- Vector
- List
- Map

For the other 2 experiments I checked:

- Vector (reserving the memory needed prior to insertions)
- Deque

Predictions

Most Efficient

- Map

Least Efficient

- List

Results

Most Efficient

- Vector

Least Efficient

- List

Discussion

I was quite thrown off by the results presented in *Figure 1*. I predicted the most efficient container would be the Map with it's *O(Log N)* insertion and sort time. It did not, however, prove to be the best container to use for this sort of operation. Vector and deque were far superior in performance. I'm not sure why the Map data structure did not outperform the other two, when theoretically, it should have (in terms of on paper performance). It could be that generating they Key-value pair added some run-time costs to creating the Map. It could also be because of heap-memory allocations that the map makes slowed it down. Also, I could have just royally messed something up.

As far as the List data structure is concerned. the result makes sense because the all the nodes will not be stored contiguously in memory. Running over the nodes in the list will take longer because of the cache misses that occur with getting to a specific node.

Deque proved to be a contender in this experiment running slightly slower than both vector implementations. Like Vector, Deque stores its elements contiguously in memory and thus is efficient at insertions and traversals, which can explain why it was the second most efficient container.

Surprisingly, though, reserving the space for the Vector prior to adding elements to it did not yield a significant performance boost except at the max value N=100,000. Even then, it saved

roughly 14-20 ms on the entire insertion and removal process. Potentially, at even higher values of N, this may constitute a more significant performance advantage.

Across the experiments the trial times were pretty consistent, so I chose to plot the data from only 1 trial considering the general shapes of the data would remain unchanged. Included in the zip folder is the raw data output and the excel workbook containing the organized data values.

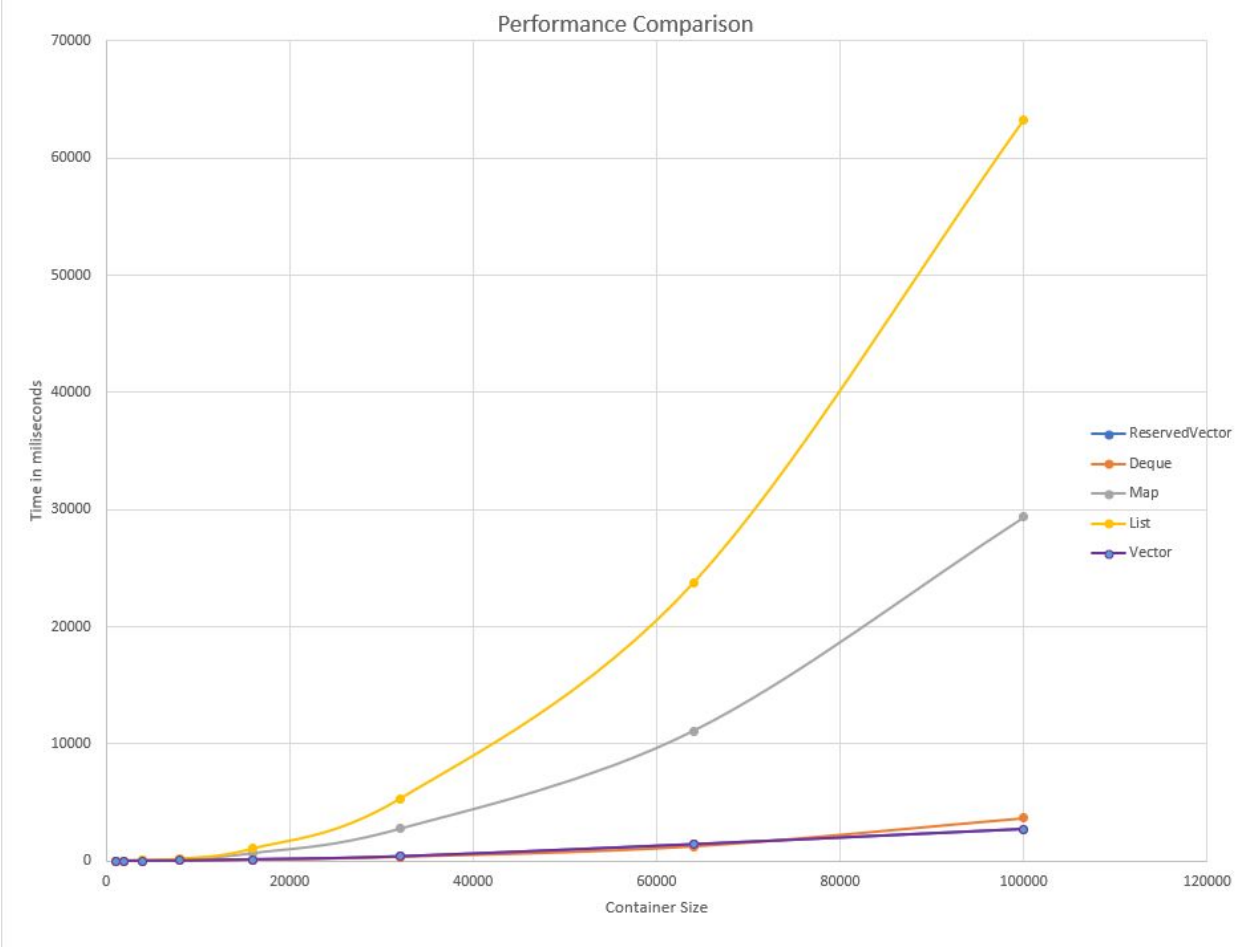

Figure 1