Starting with my timing code, add the necessary lines so you can time appending to a dynamic array. Run the code, and note any interesting observations/patterns.

- I didn't require additional #include lines, not sure if my code could be affected by this
- It was interesting to see an apostrophe within an integer in 'int32_t const num_elems = 10'000;'. I have never seen that before and am assuming that it is a feature included in the 'int32_t' data type

Starting with my timing code, add the necessary lines so you can time prepending to a dynamic array. Run the code, and note any interesting observations/patterns.

- I used the .insert() feature within the dynamic array and .begin() as well
- As an interesting note to come back to, we discussed Big O notation on Sunday. I am curious to see if I were to not use .insert() and .begin(), would my code runtime be longer?

Starting with my timing code, add the necessary lines so you can time appending to a deque. Run the code, and note any interesting observations/patterns.

- To be honest, deque seems to behave similar to a vector (dynamic array) when actually using and operating it
- There was little change to my code when comparing vector<int32_t> and deque<int32_t>
- I had to use #include <deque>, which I nearly forgot about and produced an error in my code

Starting with my timing code, add the necessary lines so you can time prepending to a deque. Run the code, and note any interesting observations/patterns.

- Naturally, the code result in this file was different from me appending it. This is because the numbers are being moved in a different order instead of just being added to the back
- I used the .push_front() feature to add the numbers very easily in the deque

Are there any discrepancies between what you observed and what you expected? If so, try to find plausible explanations.

- Not really, at least initially. With the help of .push_front(), .push_back(), .insert(), and .begin(), the code was very simple and straightforward
- However, my biggest question is surrounding runtime. Do these methods make the code run quicker than manually doing the work? I will say no because I feel that these methods simply hide the manual work that is being done by the computer's end. The runtime is still the sam