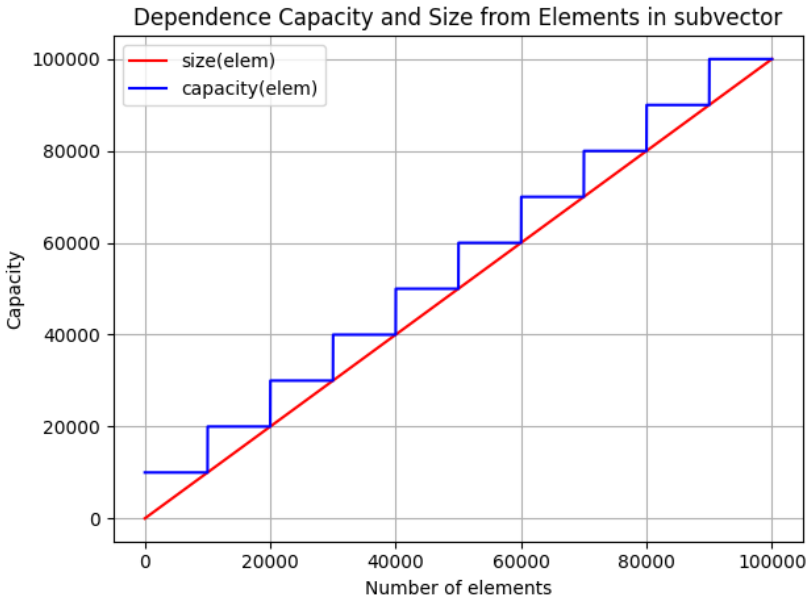
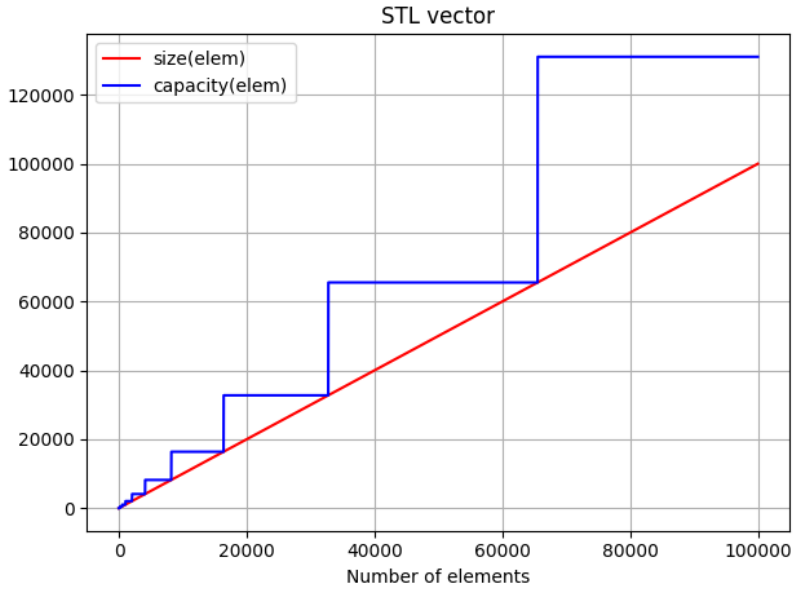
Лабораторная работа по информатике

Изучение асимптотик структур данных

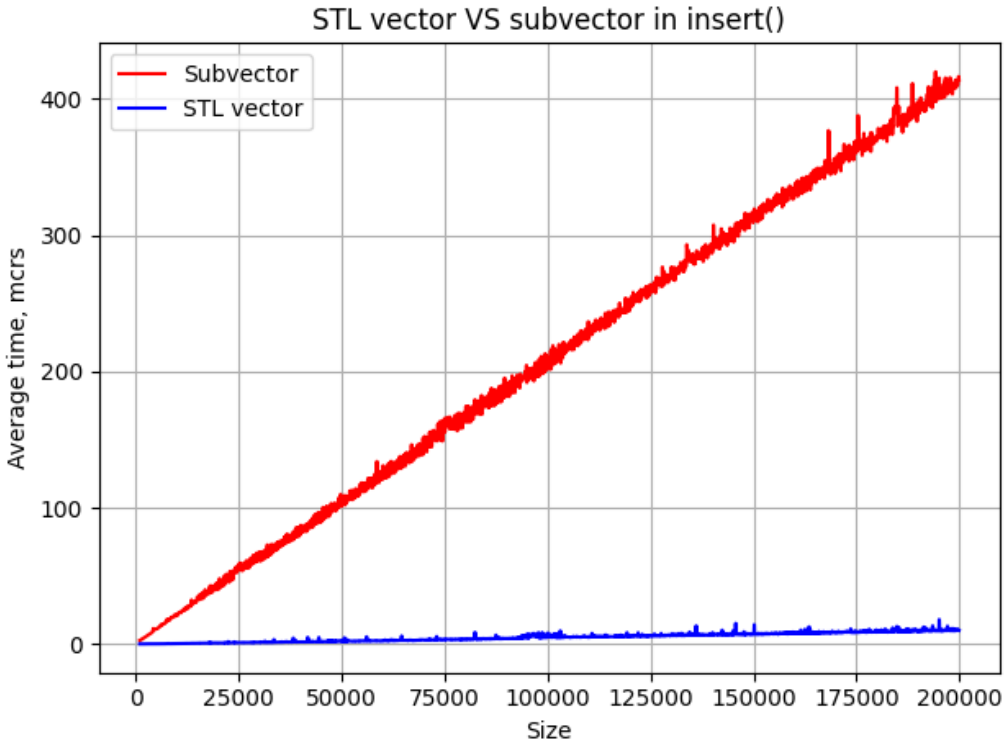
1. **Push_back** для **subvector** и **STL vector**. Графики **capacity** и **size** от номера итерации **i**.



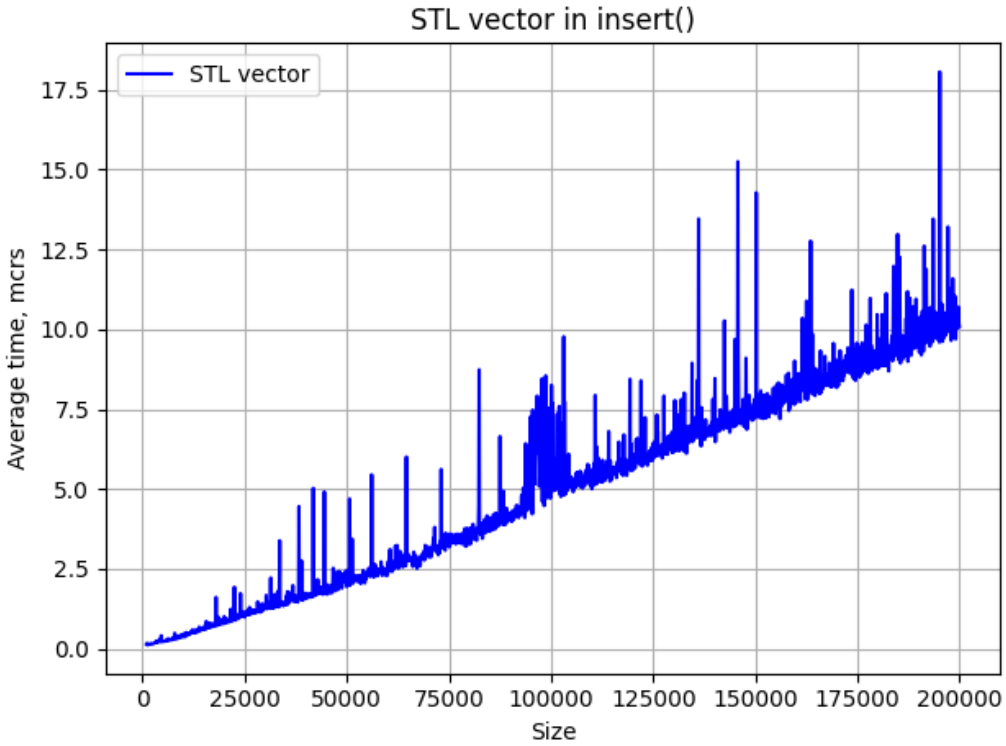


Сверху представлен график для **subvector**. **Resize()** увеличивает **capacity** на 10000 элементов. Снизу же видно, что **vector** из **STL** увеличивает **capacity** в 2 раза.

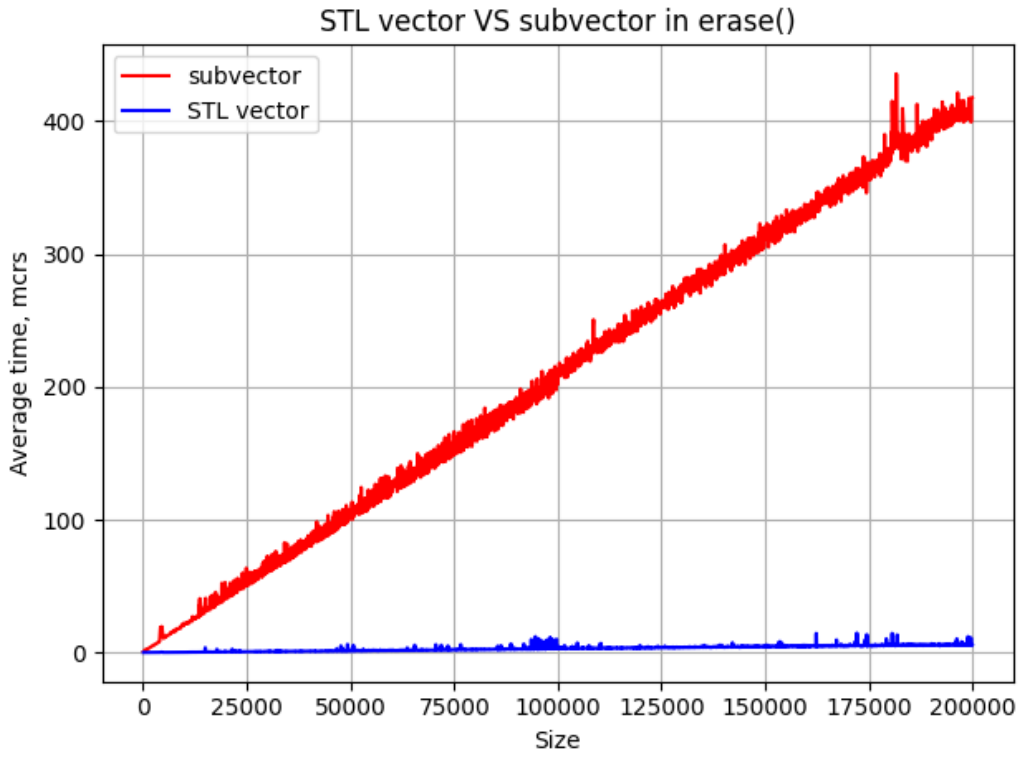
2. **Insert()** для **subvector** и для **STL vector**. Графики зависимости среднего времени вставки в произвольное место от размера вектора **size**.



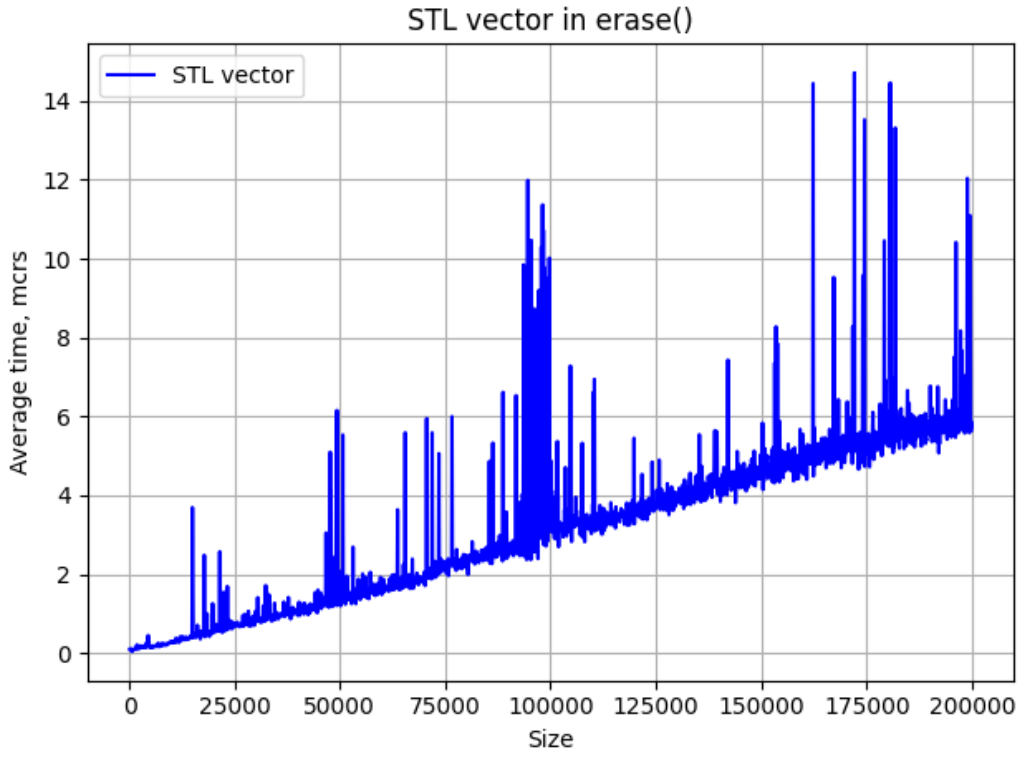
Написанный вручную **subvector** работает примерно в 40 раз медленнее чем **vector** из **STL**. Ниже представлен график только вектора из **STL**. Видно, что асимптотика — O(N).



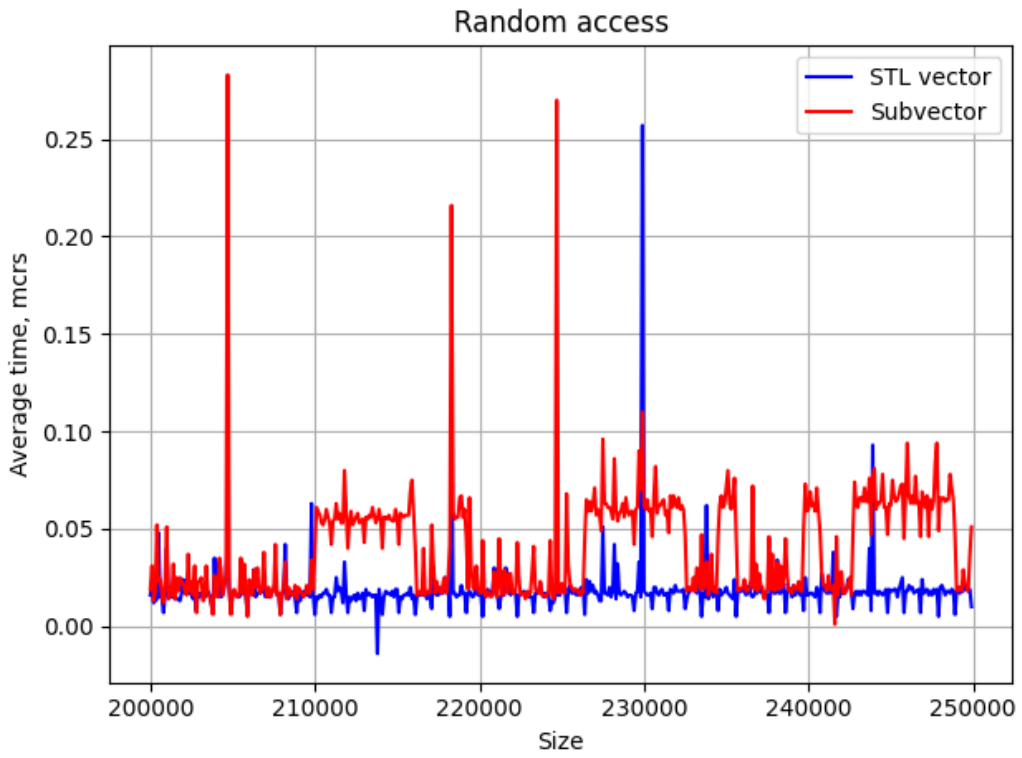
3. **Erase()** для **subvector** и для **STL vector**. Графики зависимости среднего времени удаления произвольного элемента от размера вектора **size**.



Как и в случае **insert()**, **subvector** работает гораздо дольше. Асимптотики так же O(N). График только вектора из **STL**:

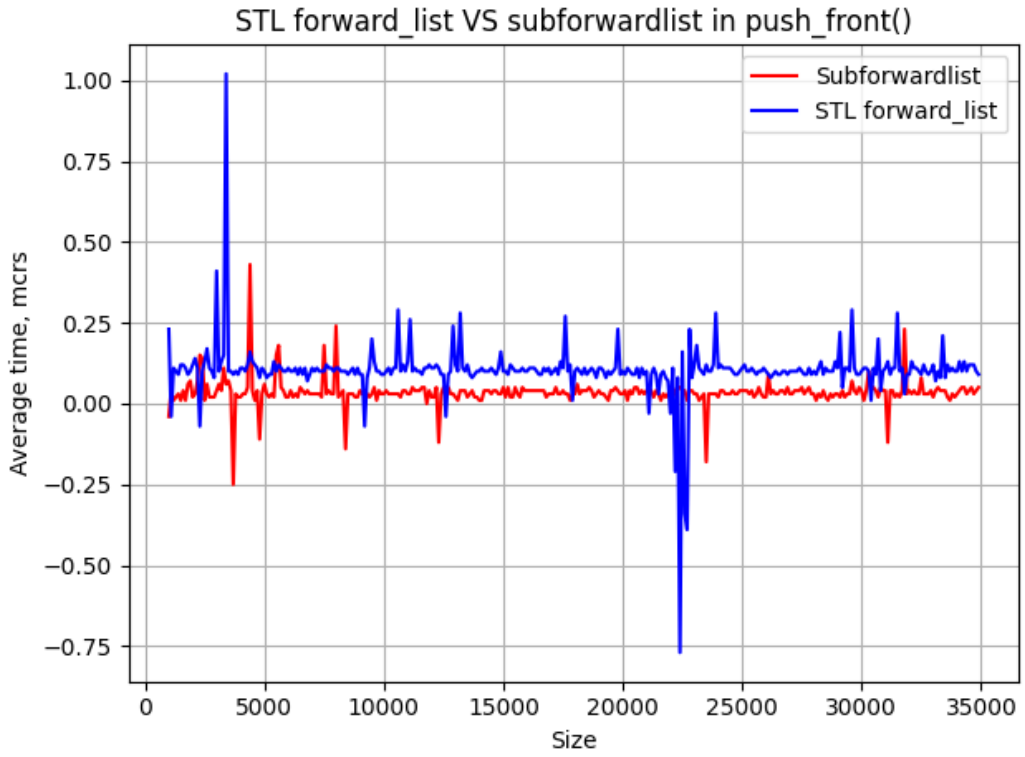


4. Графики зависимости среднего времени доступа к произвольному элементу вектора.

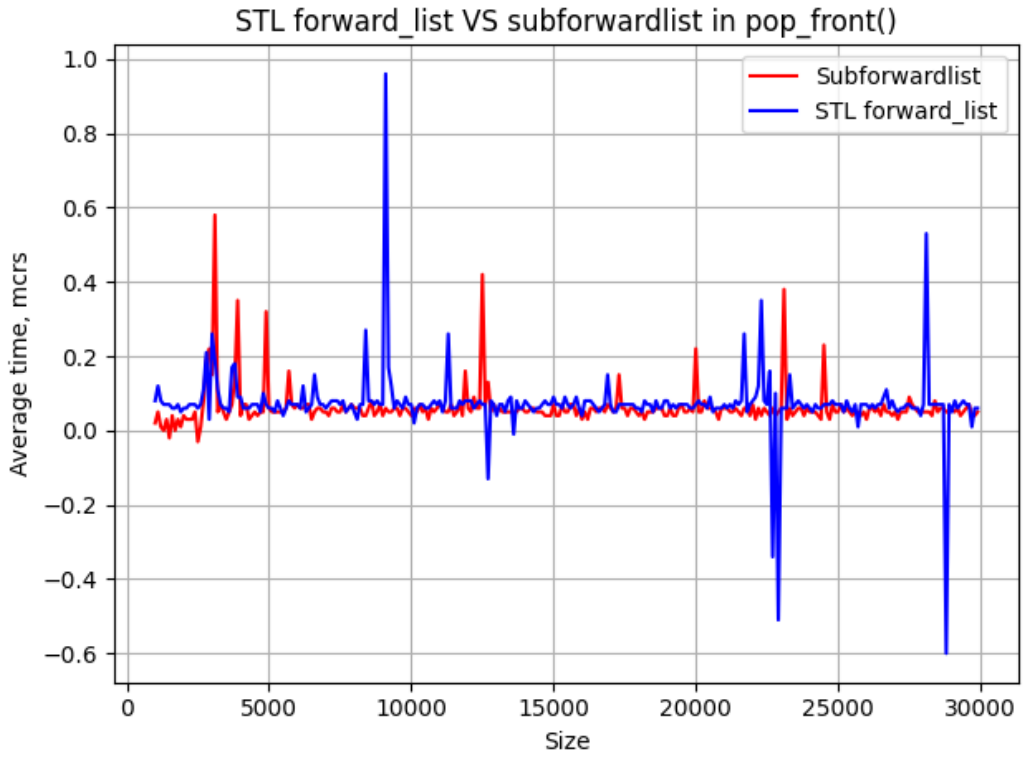


Прослеживается асимптотика O(1). На данном графике видны скачки среднего времени доступа к **subvector**, это я пытался определить кэш-память процессора. Но время доступа иногда понижается до начального значения, так что непонятно, кэш это или что-то другое повлияло на него.

5. Среднее время добавления в начало **STL forward_list** и **subforwardlist** (**push_front()**)



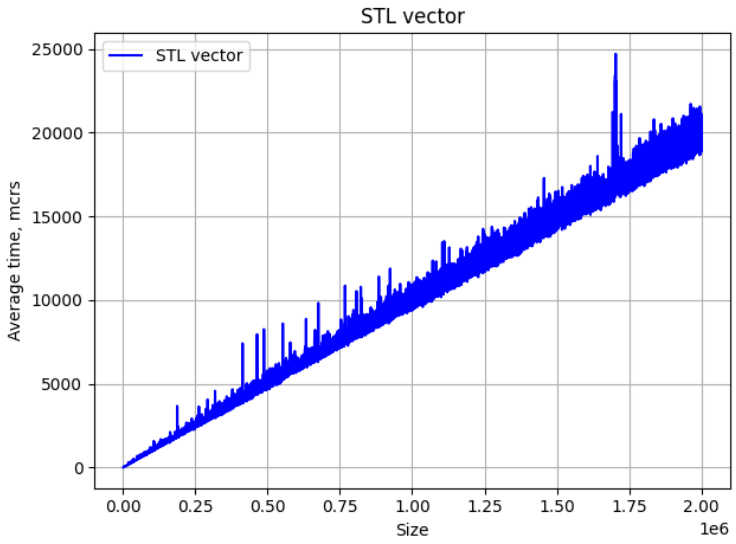
6. Среднее время удаления первого элемента **STL forward_list** и **subforwardlist** (**pop_front()**)

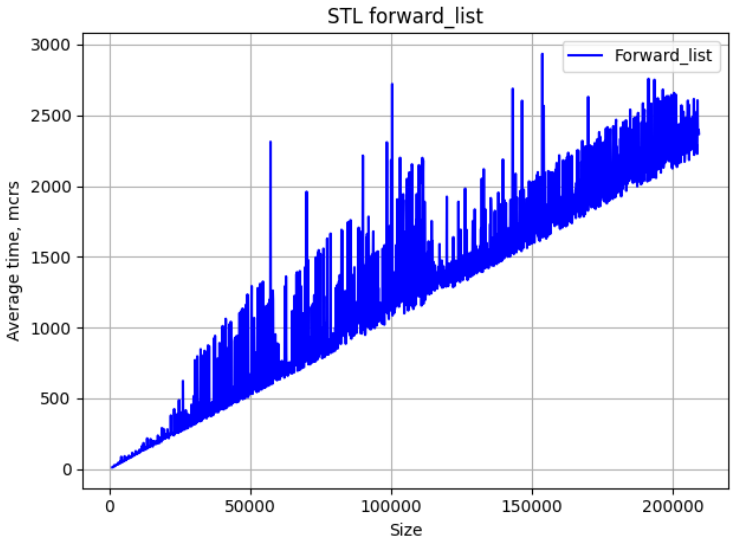


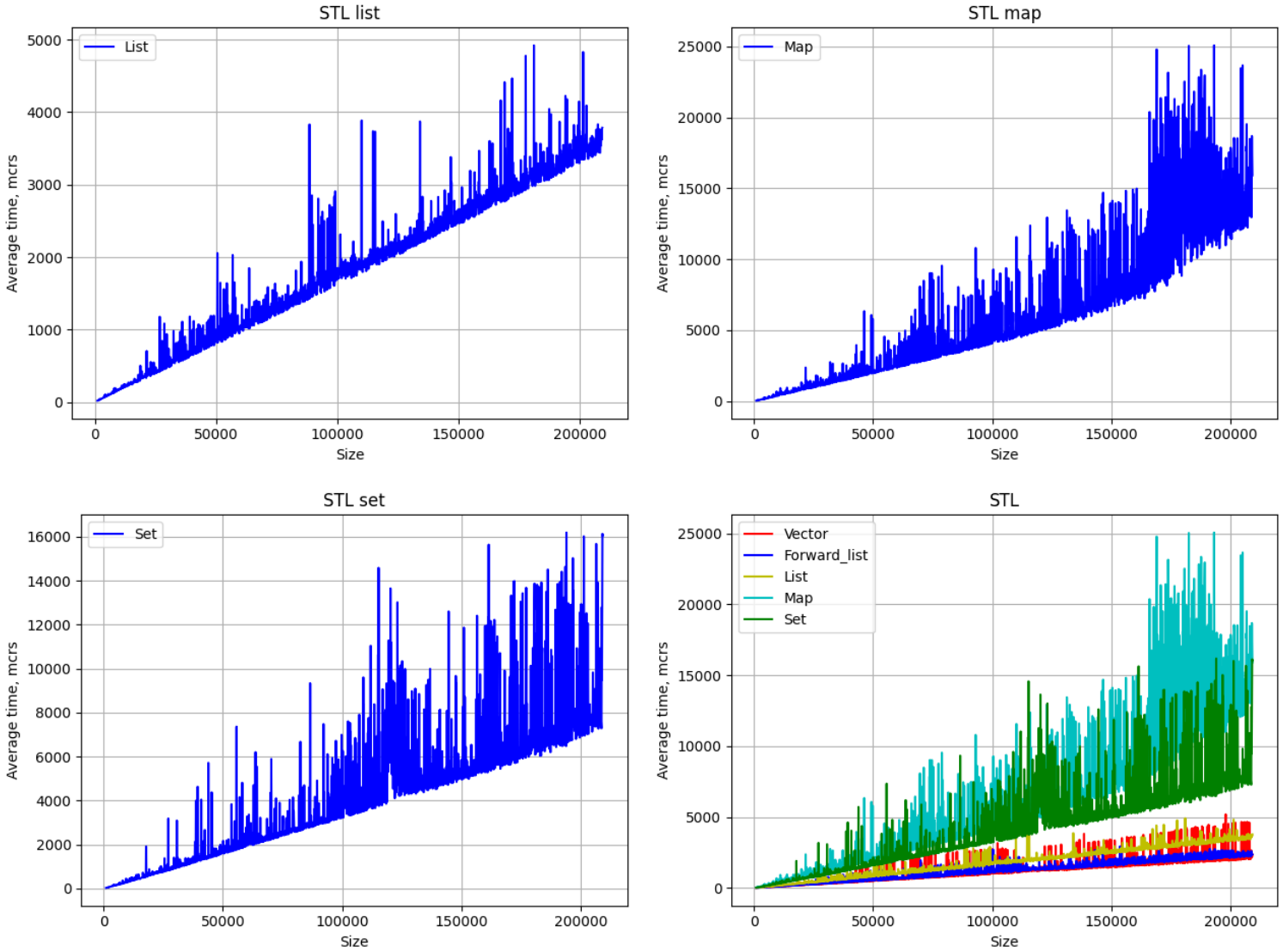
В пунктах 5 и 6, как и должно быть, асимптотика O(1). Функция **push_front()** у рукописного **subforwardlist** работает даже быстрее, чем у **STL**.

7. Среднее время обхода всего контейнера. Тестировались только классы из **STL**. К каждому элементу из **vector**, **forward_list**, **list** и **map** прибавлялась **1**. В случае **set** к переменной поочередно присваивались его элементы.

Снизу графики зависимости среднего времени обхода от размера контейнера сначала по отдельности, а потом на одном графике.

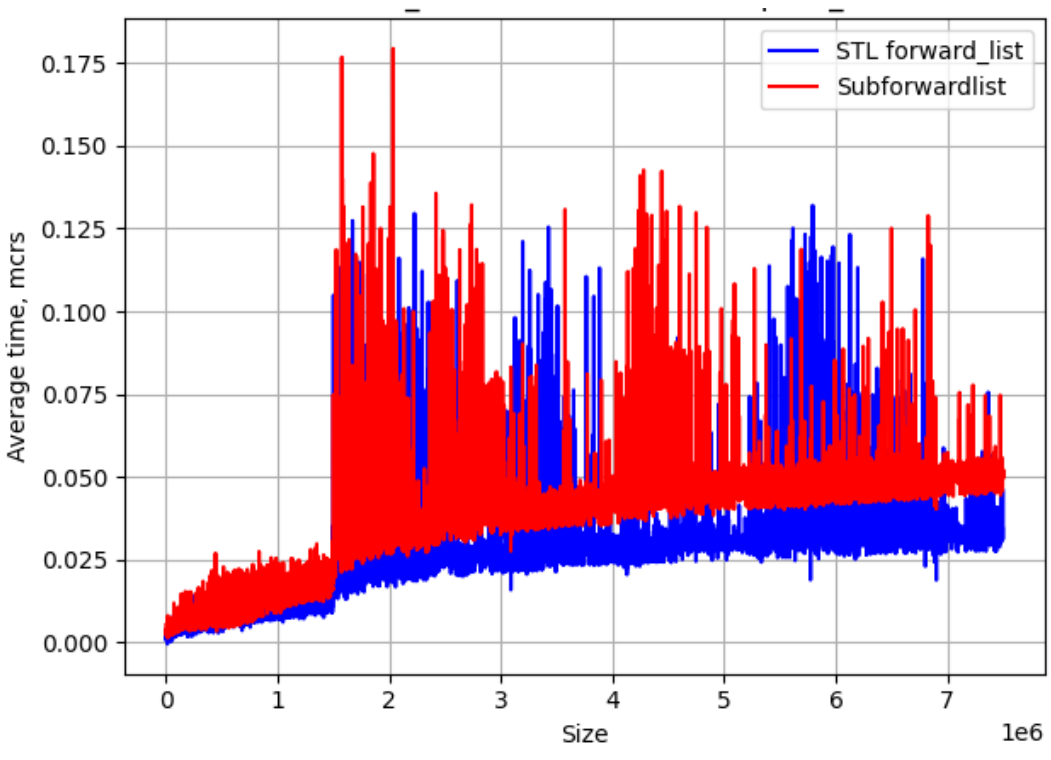




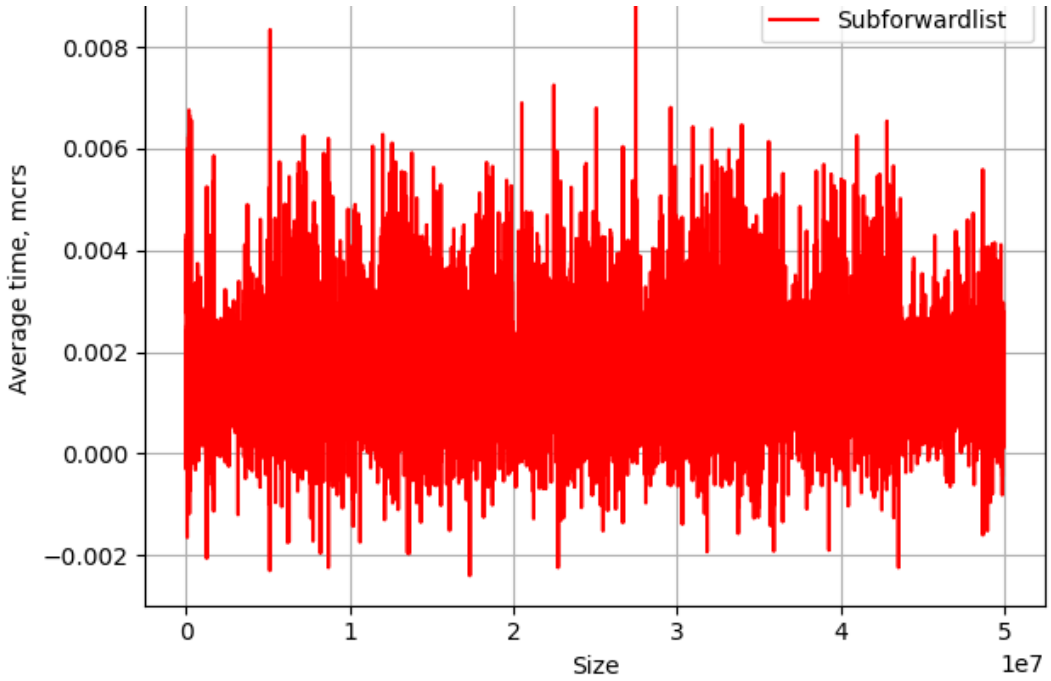


3. Ы. (К 4 пункту про **random access**)

Еще один график



Здесь виден резкий скачок в районе 1.5 миллионов элементов, и возможно это кэш. Кроме того зависимость кажется линейной, что обуславливается тем, что написанная функция **get_element** (доступ к элементу) возвращает не референс на элемент, а копию значения, из-за чего, как выяснилось, каждый раз копируется весь массив данных. График после того, как возвращаемое значение стало референсом:



Асимптотика стала O(1). 1.5 миллиона элементов типа **integer** занимает около 5.7 Мб памяти, что практически совпадает с размером кэша компьютера (6 Мб)