Homework 3: Vector vs. List

Yi Qi, UNI: yq2211

2/5/2017

• The initial random sequence is generated by calling **std::shuffle()** on a std::vector<int> of specified size n. For fair comparison, the random deletion sequence is also generated beforehand.

• To construct both std::vector and std::list data, traverse the initial random vector and linear-search for the insertion point in the target container for each integer, then call the **std::insert()** function of the container to insert.

• To delete elements one at a time from the std::vector and the std::list, traverse the pre-generated deletion sequence and call the **std::erase()** function on each index.

• The **std::set** data structure is used for comparison. No algorithm is needed for insertion as the nature of std::set is to form an ordered sequence of keys, which are the unique integers from the initial sequence. For deletion it uses the same approach as others.

• To access a specific index in the container, first initialize an iterator variable to **std::begin**, then use the **std::move_iterator::operator**++ in a for loop instead of calling the std::advance() or std::next() functions.

Results

The execution time of insertion and deletion operations are illustrated below.

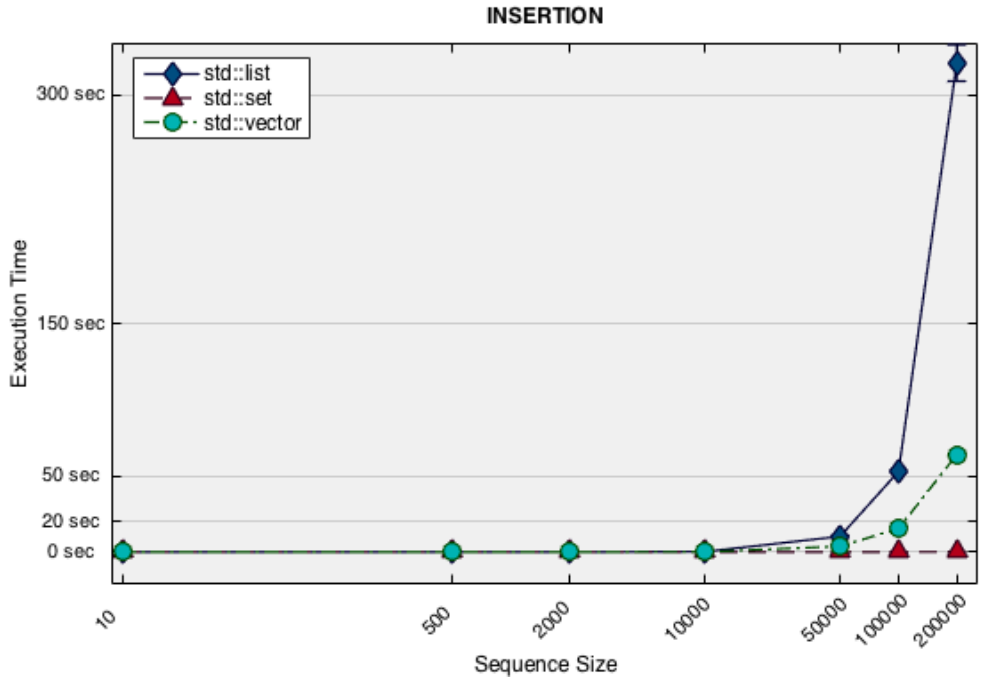

Figure 1: Execution Time of Insertion vs. Size of Input

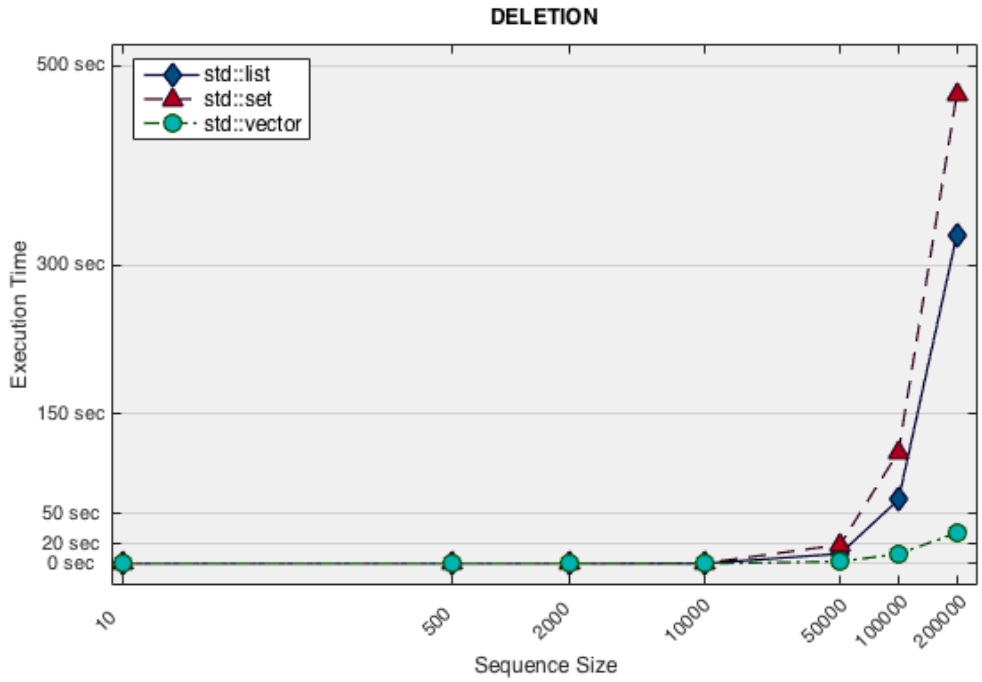

Figure 2: Execution Time of Deletion vs. Size of Input

• **std::vector**
Both insertion and deletion operations are fast, in almost linear time, although in theory it should of $O(n^2)$ time. Although inserting and deleting an element involves shifting an expected 50% members in the container, the elements are stored as an array in memory, which makes it much easier to copy them with caches.

• **std::list**
Both insertion and deletion operations perform with $O(n^2)$ time as expected. Inserting and deleting element at any given index cost constant time for std::list, however traversing to that index takes much more time than traversing in std::vector, because of the scattered storage of std::list members in memory.

• **std::set**
The std::set is the perfect choice for the insertion operation, which almost takes constant time. The deletion, on the other hand, is the slowest because the deletion is by iterator. The std::set is simply not designed for such usage. It could be equally fast as insertion if erasing elements by their keys.

Experiments

• Preallocation undermines the performance of insertion for both std::vector and std::list. Dynamic allocation has been made very efficient for these containers and should be adopted instead of preallocation.

• For std::vector the insertion time can be further improved with binary-search, while for std::list linear-search is faster due to less traversing. Using std::find_if() with function object or Lambda expression does not improve the performance of insertion, probably because of the overhead of function calls.

• Using std::advance() improves the deletion performance of std::vector significantly, however it is not the case for std::list and std::set.