- Report

I started with a function to generate N random integers and insert them into a sequence and a applied a function that generates a random number + change range A for loop is needed to insert numbers in its proper positions using lower_bound

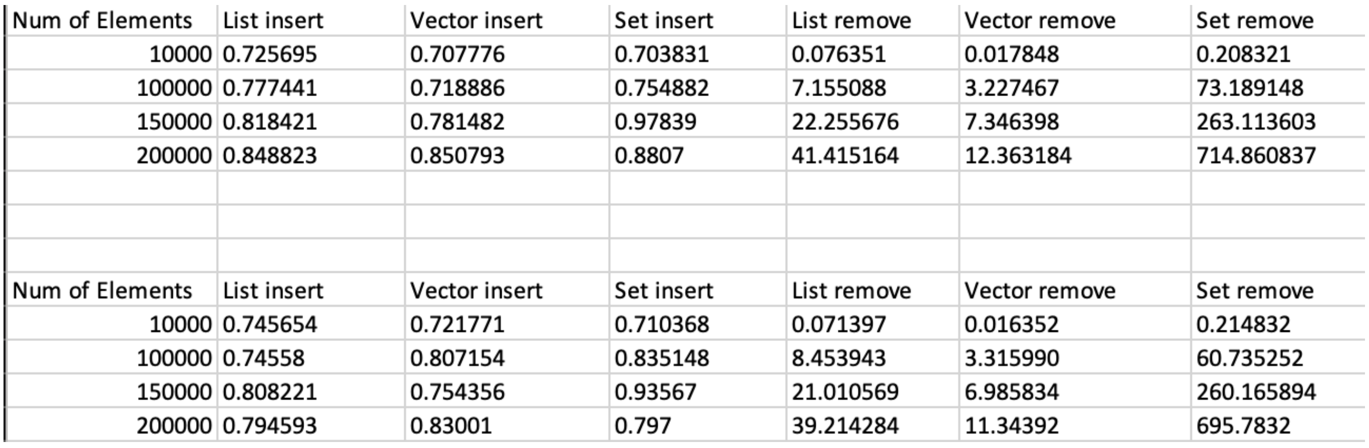
Then I implemented a function to remove elements one at a time based on random positions also using a random number generator

In case of list and set I had a lot of problems with that implementation, since it was giving me results similar to:

```
ist(9249,0x1178a3600) malloc: *** error for object 0x7ff7b117e310: pointer
being freed was not allocated
list(9249,0x1178a3600) malloc: *** set a breakpoint in malloc_error_break
to debug
[1]    9249 abort
```

at the end of every remove. So I thought that it is likely due to the iterator position not being checked for validity before erasing the element from the list, so I used I started checking if the iterator it is valid (not equal to sequence.end()). Same with set, basically I used a custom iterator approach to remove elements from the set. I don't know how smart this solution is, but it didn't work without it. With set implementation I used a specific erase method to remove elements from the set while iterating through it, which randomly selects an element from the set and erases it

Results of each data structure operations:



Num of Elements	List insert	Vector insert	Set insert	List remove	Vector remove	Set remove
10000	0.725695	0.707776	0.703831	0.076351	0.017848	0.208321
100000	0.777441	0.718886	0.754882	7.155088	3.227467	73.189148
150000	0.818421	0.781482	0.97839	22.255676	7.346398	263.113603
200000	0.848823	0.850793	0.8807	41.415164	12.363184	714.860837
Num of Elements	List insert	Vector insert	Set insert	List remove	Vector remove	Set remove
10000	0.745654	0.721771	0.710368	0.071397	0.016352	0.214832
100000	0.74558	0.807154	0.835148	8.453943	3.315990	60.735252
150000	0.808221	0.754356	0.93567	21.010569	6.985834	260.165894
200000	0.794593	0.83001	0.797	39.214284	11.34392	695.7832

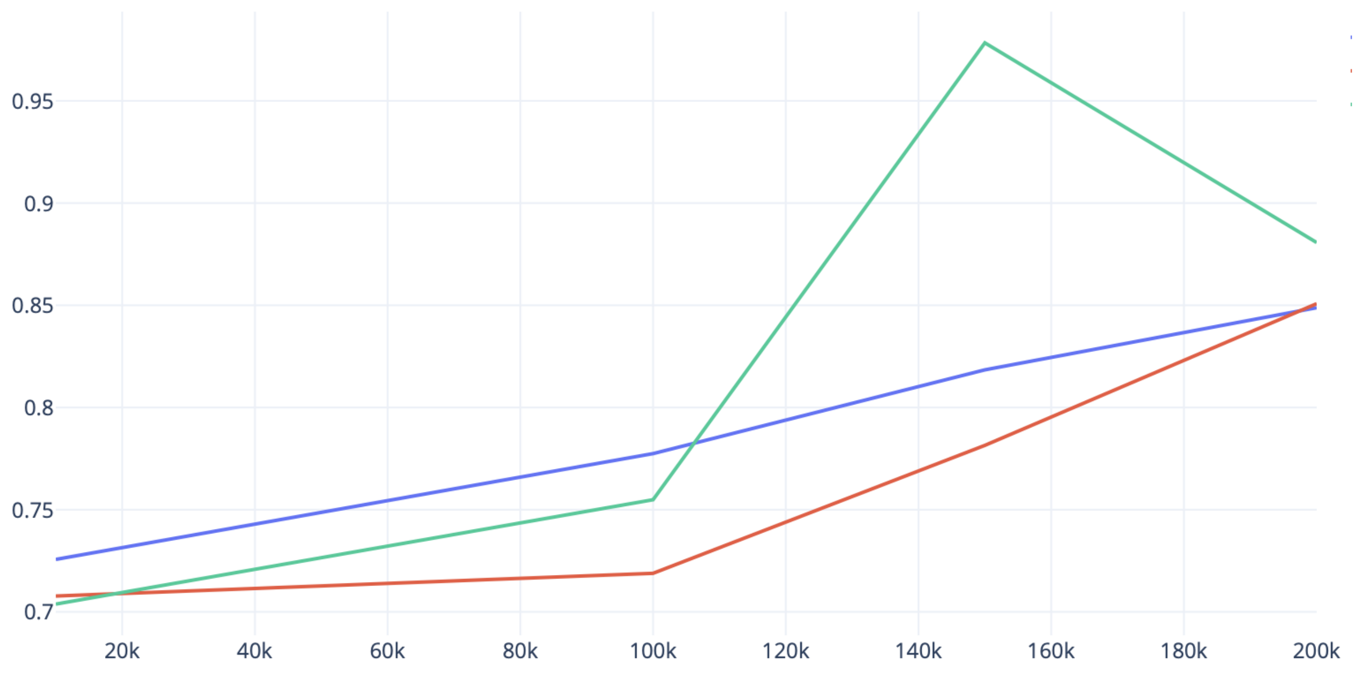
Graph Analyze

blue - List;

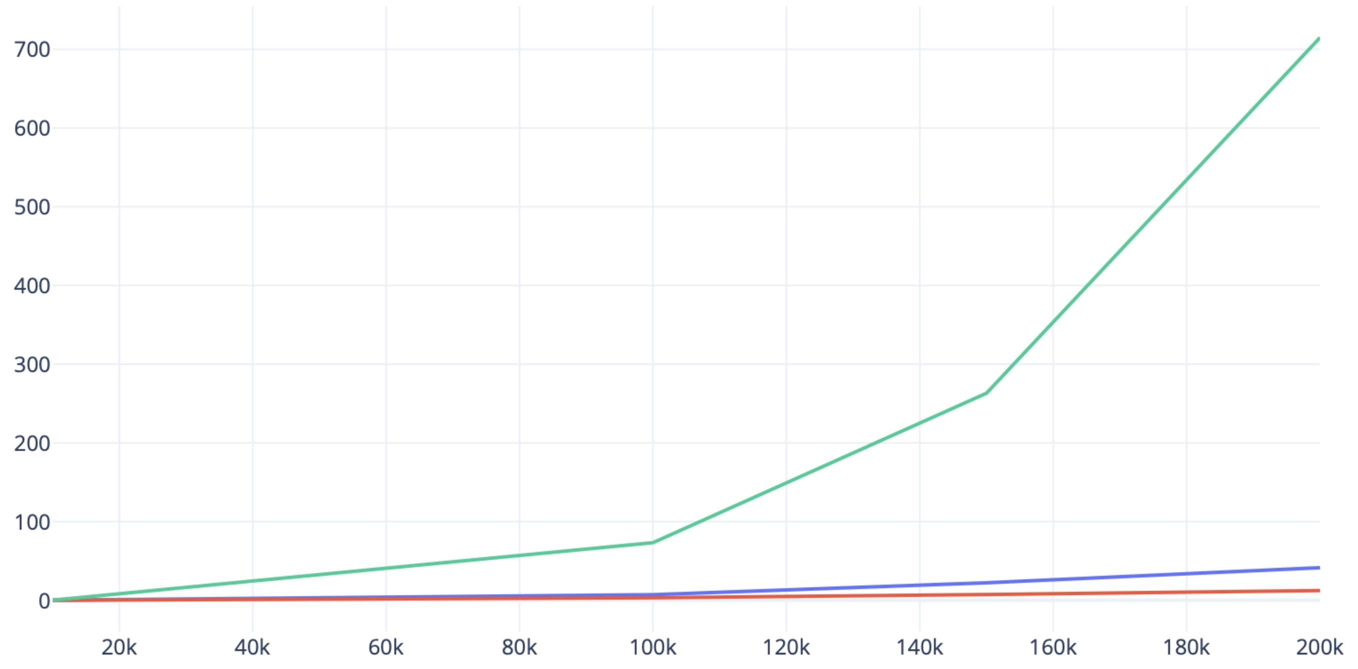
Red - Vector;

Green - Set;

The following picture represents insert operation for 3 data structures. We can analyze that vector is the most efficient data structure in both data sets



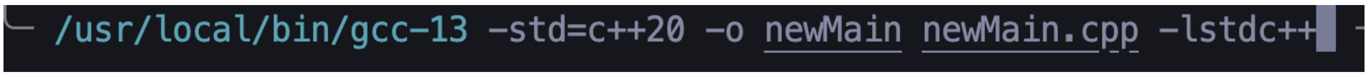
Now let's analyze remove operation for datastructures: List - The list removal time seems to be consistently efficient across the board. This is likely due to the fact that list has constant-time removal since it uses doubly linked lists, which means that removing elements at arbitrary positions is efficient. Vector - he vector removal time also seems fairly efficient and remains consistent across different container sizes. This might be due to the fact that vectors use contiguous memory and have efficient cache usage, which can lead to relatively faster removal times, especially for small to moderate container sizes. Set - The set removal time is significantly higher compared to the list and vector removal times, and it increases as the container size grows. This inefficiency can be attributed to the nature of the set data structure. In a set, elements are stored in a way that ensures they are ordered, which is why insertion and retrieval times are efficient. However, deletion times can be higher because removing an element from a balanced binary search tree (which is the underlying data structure of std::set) requires rebalancing the tree, which can be relatively expensive. Additionally, the set removal time might be influenced by factors such as cache inefficiency, especially for larger container sizes.



By analyzing those result we can see depending on the number of elements we can find the most suitable data structure for the insert and remove operations, however, in my case set showed himself as the least efficient data structure, which is a thing since by removing elements it needs to rebalance the whole tree In my personal case set took even more time than expected because of the problem with the implementation, described above

The source code of each file is in the zip file

You can compile the code by creating the executable first, in my case it is done by:



```
╰─ /usr/local/bin/gcc-13 -std=c++20 -o newMain newMain.cpp -lstdc++ -
```