Report of the SAE S1.02 project

Algorithm explanation

std::sort

The `std::sort` function is a sorting algorithm that is part of the C++ Standard Library. It is a comparison-based sorting algorithm that uses a hybrid sorting algorithm: a mix of quicksort, heapsort, and insertion sort. The algorithm has a complexity of O(n log n) on average, where n is the number of elements to sort.

std::stable_sort

The `std::stable_sort` function is a sorting algorithm that is part of the C++ Standard Library. It is a comparison-based sorting algorithm that is a stable version of `std::sort`. The algorithm has a complexity of O(n log n) on average, where n is the number of elements to sort.

qsort

The `qsort` function is a sorting algorithm that is part of the C Standard Library. It is a comparison-based sorting algorithm that uses the quicksort algorithm. The algorithm has a complexity of O(n log n) on average, where n is the number of elements to sort.

quicksortrnd

The `quicksortrnd` function is a sorting algorithm that is a random version of the quicksort algorithm. The algorithm has a complexity of O(n log n) on average, where n is the number of elements to sort.

quicksortdet

The `quicksortdet` function is a sorting algorithm that is a deterministic version of the quicksort algorithm. The algorithm has a complexity of O(n log n) on average, where n is the number of elements to sort.

bubble sort

The `bubble sort` function is a sorting algorithm that is a comparison-based sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The algorithm has a complexity of O(n^2) on average, where n is the number of elements to sort.

insertion sort

The `insertion sort` function is a sorting algorithm that is a comparison-based sorting algorithm that builds the final sorted array one item at a time. The algorithm has a complexity of O(n^2) on average, where n is the number of elements to sort.

selection sort

The `selection sort` function is a sorting algorithm that is a comparison-based sorting algorithm that divides the input list into two parts: the sublist of items already sorted and the sublist of items remaining to be sorted. The algorithm has a complexity of O(n^2) on average, where n is the number of elements to sort.

ICan'tBelieveItCanSort

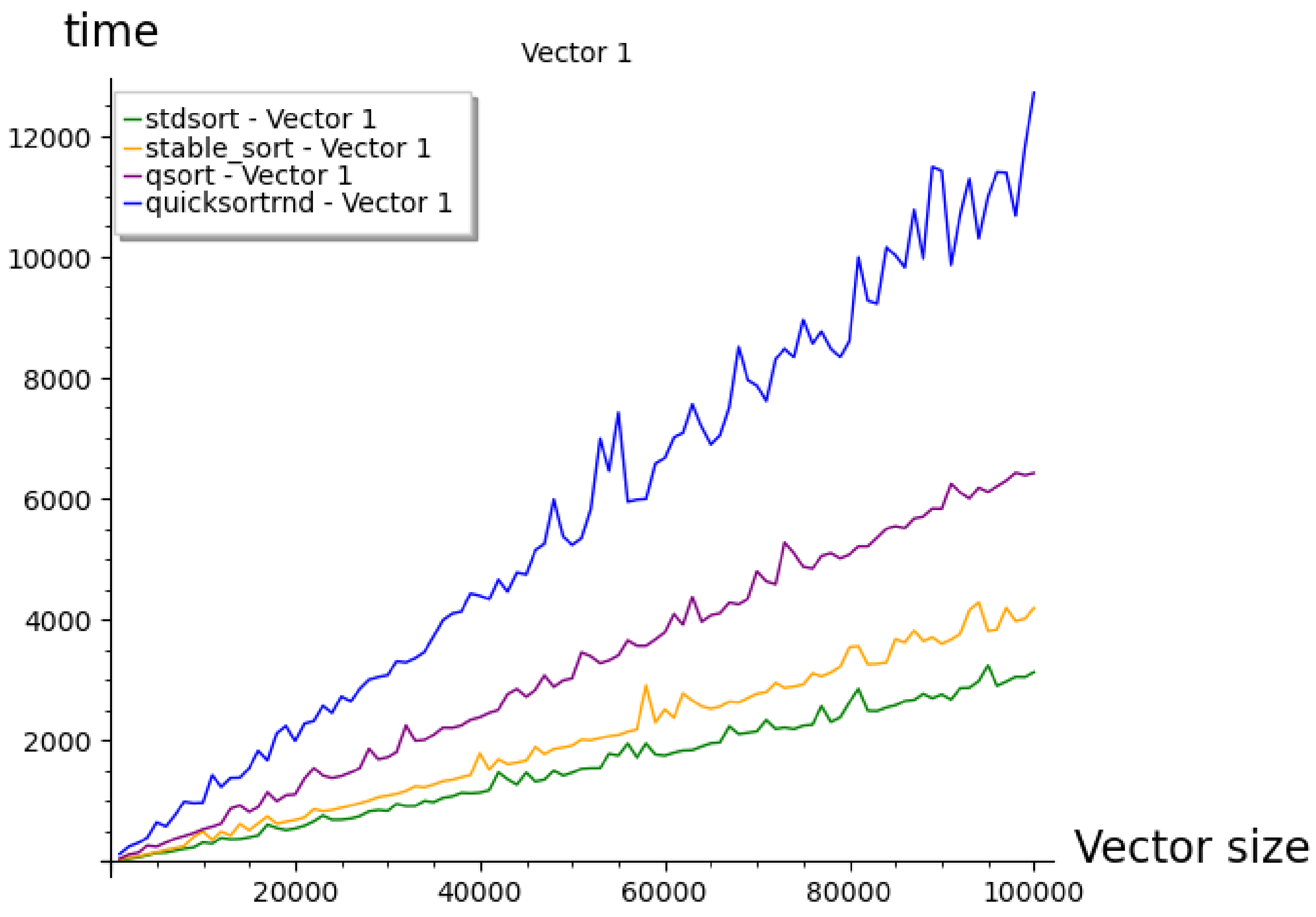
The `ICan'tBelieveItCanSort` function is a sorting algorithm that is a comparison-based sorting algorithm that is a joke algorithm. The algorithm has a complexity of O(n^2) on average, where n is the number of elements to sort.

Comparison of the algorithms

- We separated the algorithms into two categories: fast and slow algorithms. The fast algorithms are `std::sort`, `std::stable_sort`, `qsort`, `quicksortrnd` because they have a complexity of O(n log n) on average.
- The slow algorithms are `quicksortdet`, `bubble sort`, `insertion sort`, `selection sort`, and `ICan'tBelieveItCanSort` because they have a complexity of O(n^2) on average.

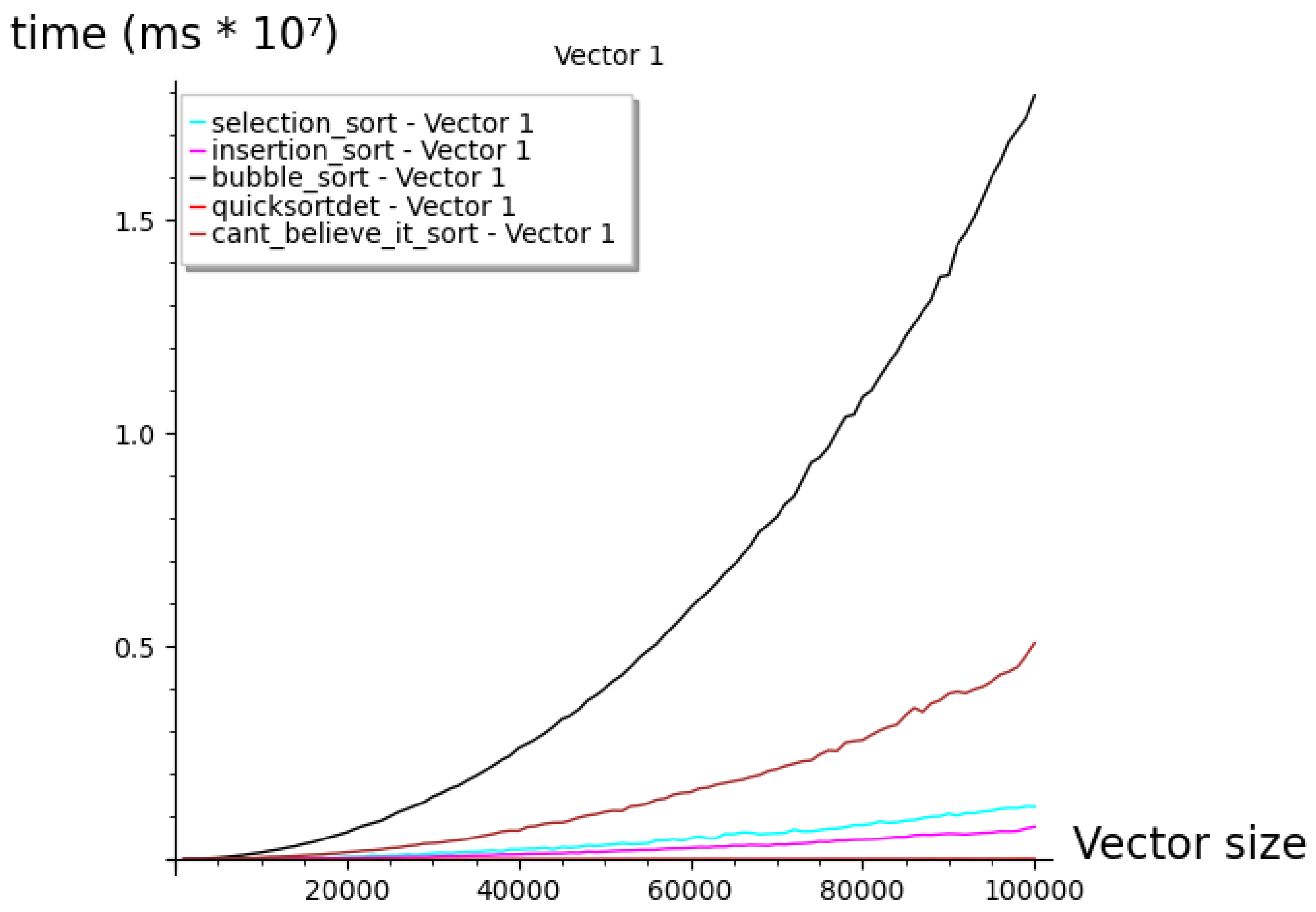
Random vector

Fast algorithms



We can see that the std::sort algorithm is the fastest algorithm for the random vector, followed by the std::stable_sort algorithm. The quicksortrnd algorithm is the slowest of the fast algorithms. The qsort algorithm is faster than the quicksortrnd algorithm, we can see that the quicksortrnd take less then thirteen hundred milliseconds to sort the vector, for the other algorithms, they take less than seven hundred milliseconds to sort the vector.

Slow algorithms

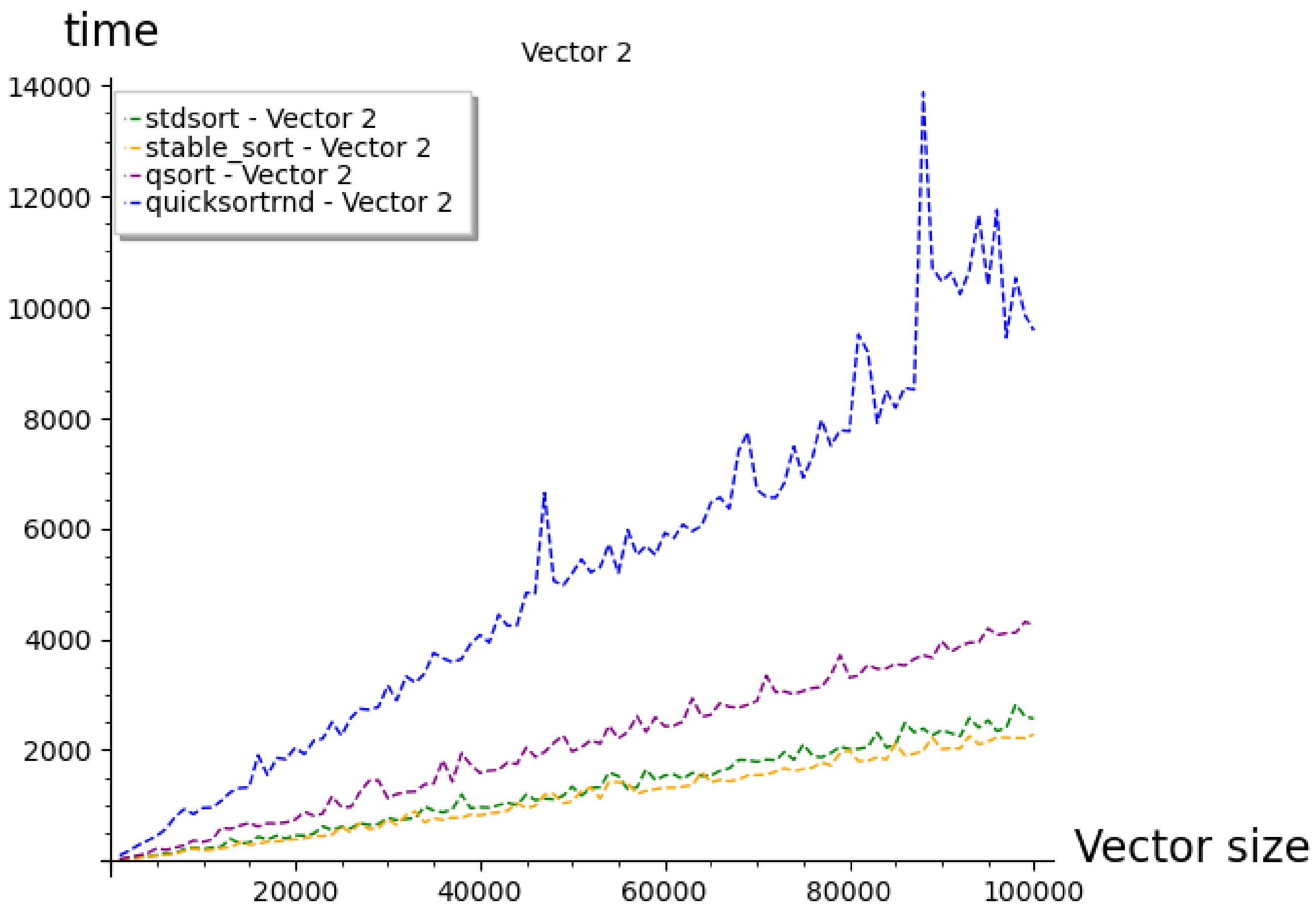


We can see that the quicksortdet algorithm is the fastest algorithm for the random vector, followed by insertion sort, the selection sort algorithm, the ICan'tBelieveItCanSort algorithm and the bubble sort algorithm. The bubble sort algorithm is the slowest of the slow algorithms with a lot of difference with the other algorithms because it just go take million of milliseconds to sort the vector compared to the other algorithms.

To conclude, the std::sort algorithm is the fastest algorithm of the fast algorithm for the random vector and the insertion sort algorithm is the fastest algorithm of the slow algorithm for the random vector.

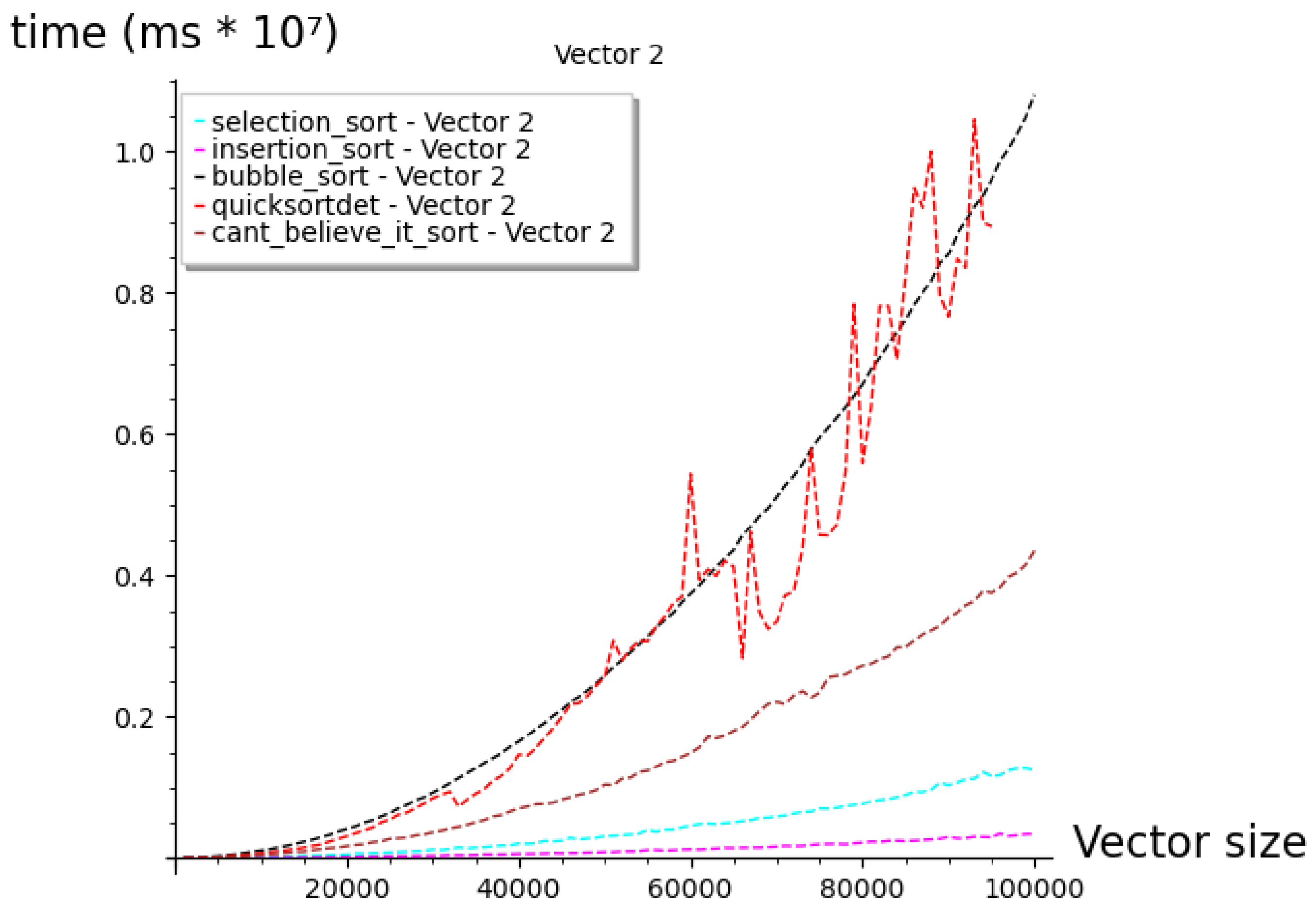
Half sorted vector

Fast algorithms



We can see that the stable_sort is faster than the std::sort for this one followed by the qsort, we can see that they are faster than with the random vector. The quicksortrnd, him, is slower than with the random vector.

Slow algorithms

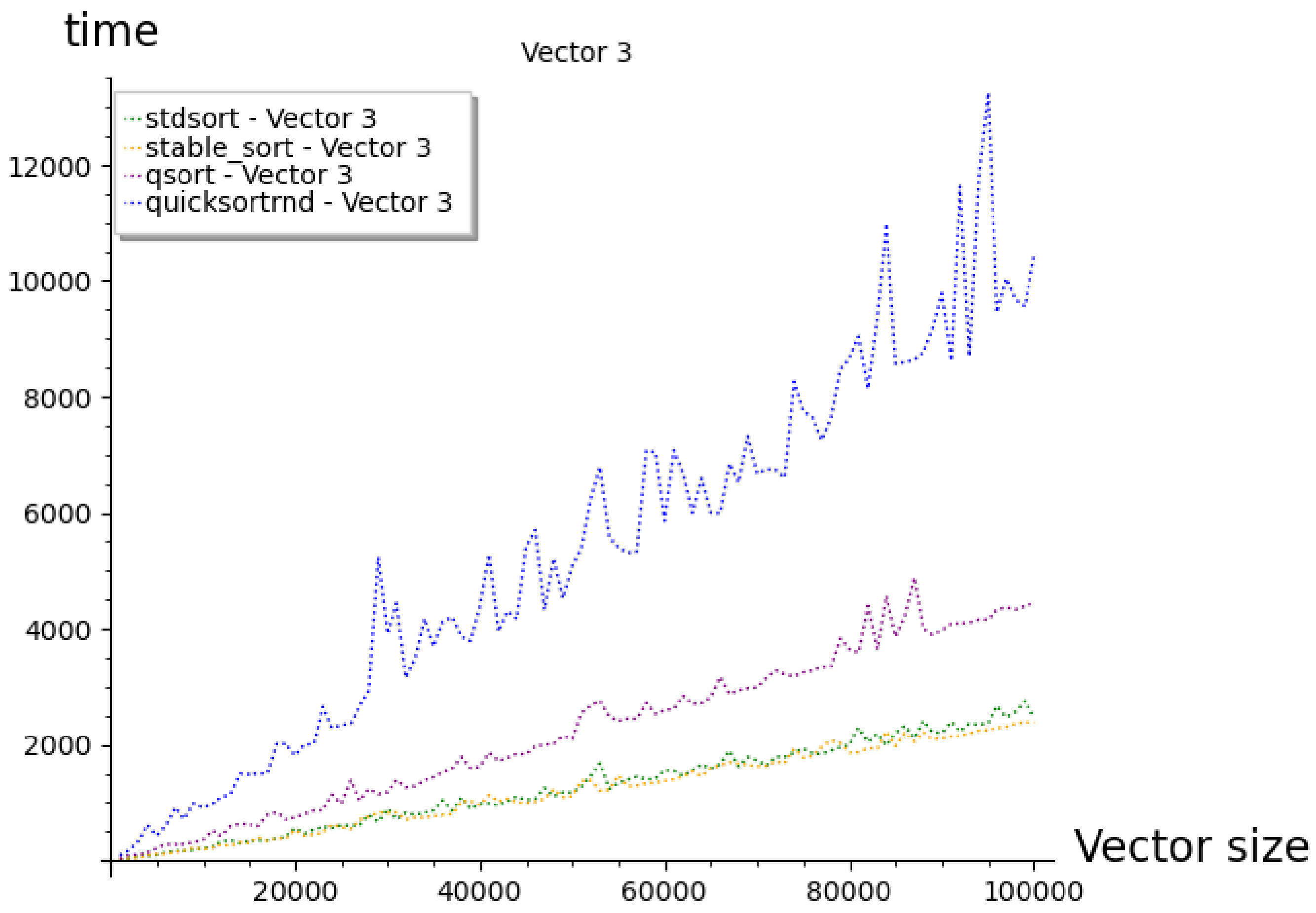


We can see that the quicksortdet algorithm is really more slower than with the random vector, it is mostly equal to the bubble sort algorithm because he was faster for this vector than for the random vector, the insertion sort algorithm is the fastest of the slow algorithms for the half sorted vector followed by the selection sort algorithm and the ICan'tBelieveItCanSort algorithm, we can see that the ICan'tBelieveItCanSort sorting time is pretty the same as the random vector.

To conclude, the std::stable_sort algorithm is the fastest algorithm of the fast algorithm for the half sorted vector and the insertion sort algorithm is the fastest algorithm of the slow algorithm for the half sorted vector.

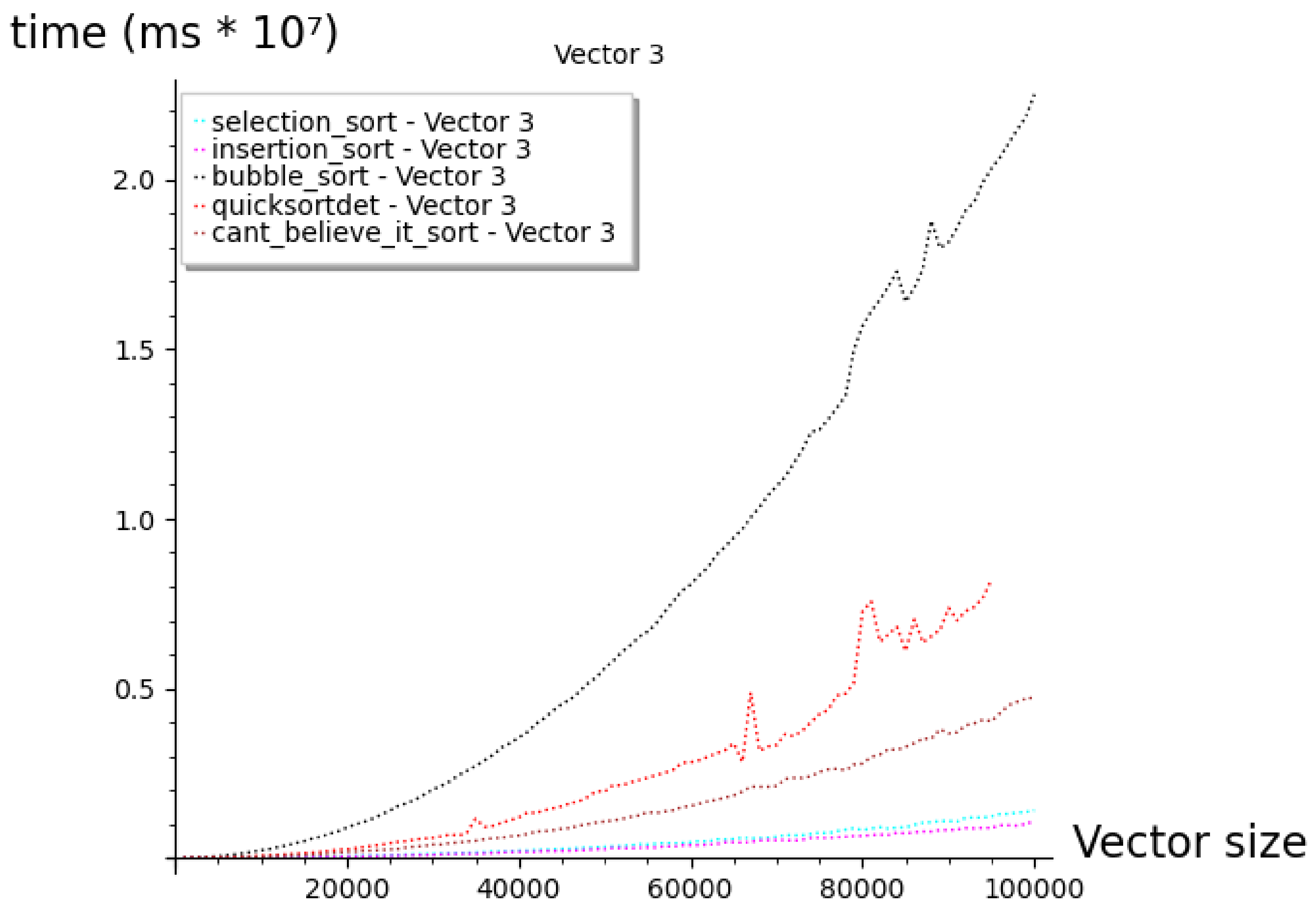
Half reverse sorted vector

Fast algorithms



We can see that the stable_sort is the faster algorithm for the half reverse sorted vector, followed by the std::sort algorithm, the qsort and the quicksortrnd, there is not a lot of difference with this vector compared to the half sorted vector.

Slow algorithms



We can see that the insertion sort is always the faster, followed by selection sort, the ICan'tBelieveItCanSort. The quicksortdet is much faster than with the half sorted vector, the bubble sort algorithm is always the slowest algorithm for this vector.

To conclude, the std::stable_sort algorithm is always the fastest algorithm of the fast algorithm for the half reverse sorted vector and the insertion sort algorithm is the fastest algorithm of the slow algorithm for the half reverse sorted vector.

Remarks

In all the plots, we can see some peaks or some variation, for most of the algorithms, this is because the computer is doing some other tasks at the same time, so the time is not really accurate, but we observe that the std::stable_sort is always the most stable because he has the less variation, this is why he is sometimes a bit slower than the std::sort algorithm.

Conclusion

In conclusion, we can see that the std::sort algorithm is the fastest algorithm for the random vector, the std::stable_sort algorithm is the fastest algorithm for the half sorted vector and the half reverse sorted vector. The insertion sort algorithm is the fastest algorithm of the slow algorithm for the random vector, the half sorted vector and the half reverse sorted vector. The bubble sort algorithm is always the slowest algorithm for the random vector, the half sorted vector and the half reverse sorted vector. The worst algorithm with millions of milliseconds is the bubble sort algorithm.

There is a big difference between the fast and the slow algorithm (O(n log n) and O(n^2)), the O(n log n) just take less than thirteen hundred milliseconds to sort the vector, for the O(n^2), they more than millions of milliseconds to sort a vector.