Exercises for 1 Jun 2020

1. Write the pseudocode for an algorithm that will remove all of the numbers from a list that are multiples of 10. Analyze its asymptotic complexity.

```
for(auto i = list.begin(); i != list.end(); i++)
{
     if (*i % 10 == 0) list.erase(i);
}
```

 asymptotic complexity should be O(n), assuming the time complexity of list.erase is O(1)

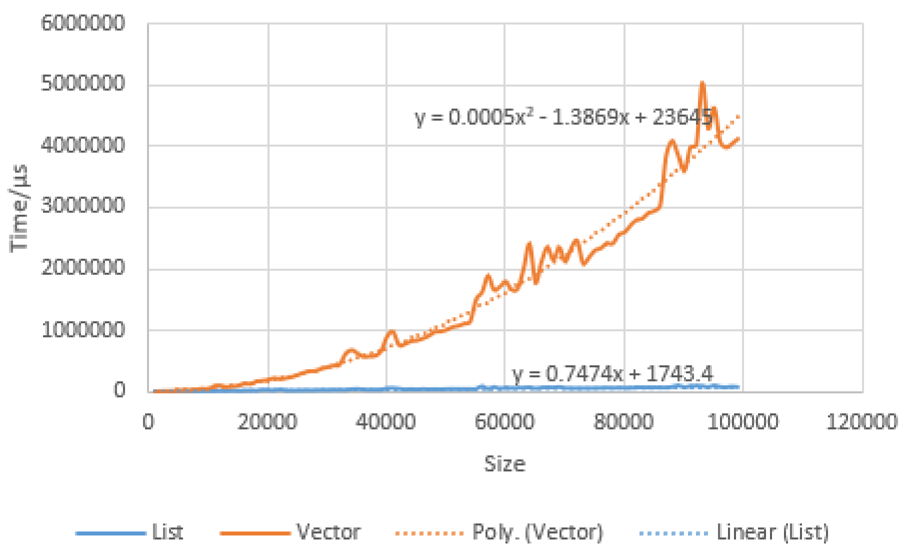
2. Modify the above algorithm to remove the same numbers from a vector instead of a list. Analyze *its* asymptotic complexity.

```
for(auto i = vector.begin(); i != vector.end(); i++)
{
     if (*i % 10 == 0) vector.erase(i);
}
```

 asymptotic complexity should be O(n^2), assuming the time complexity of vector.erase is O(n)

3. Implement your algorithms using the `std::list` and `std::vector` class templates. Time their execution for varying sizes of lists and vectors to conform your theoretical analysis.

 From the following figure, we can easily tell that for std::list, the time complexity is linear, while its quadratic for vector.



Exercises for 3 Jun 2020

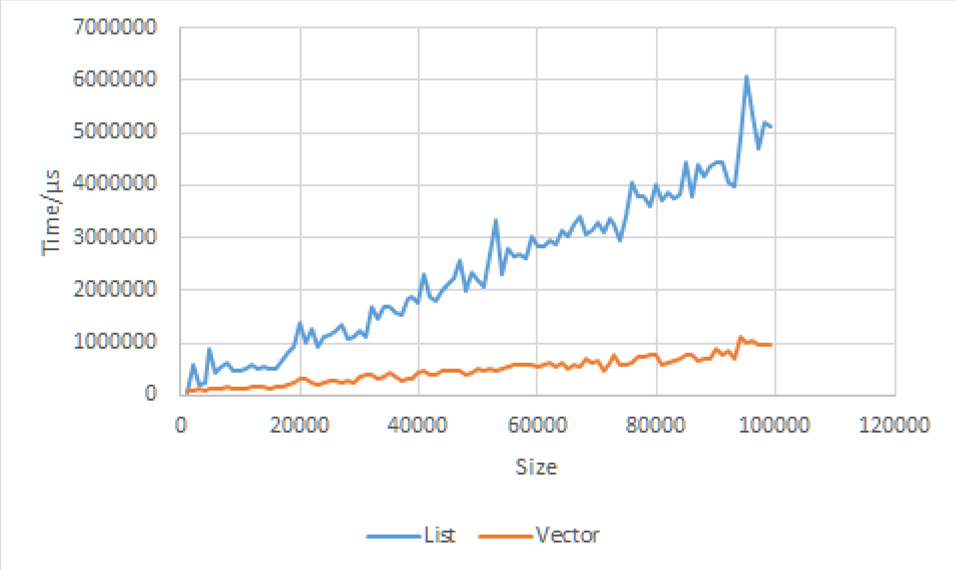
1. Write a naïve implementation of a quicksort function template that stores temporary values in `std::list<T>` objects, i.e., that constructs a `std::list<T>` for elements smaller than the pivot, another for values equal to the pivot and a third for values larger than the pivot. Use the `std::list::splice` method to join these lists together after the recursive sorting step.

see function naiveQuickSortList().

2. [optional] Convert your implementation to use `std::vector<T>` instead of `std::list<T>` for its temporary storage. Time both versions. What do you observe?

   see function naiveQuickSortVector().

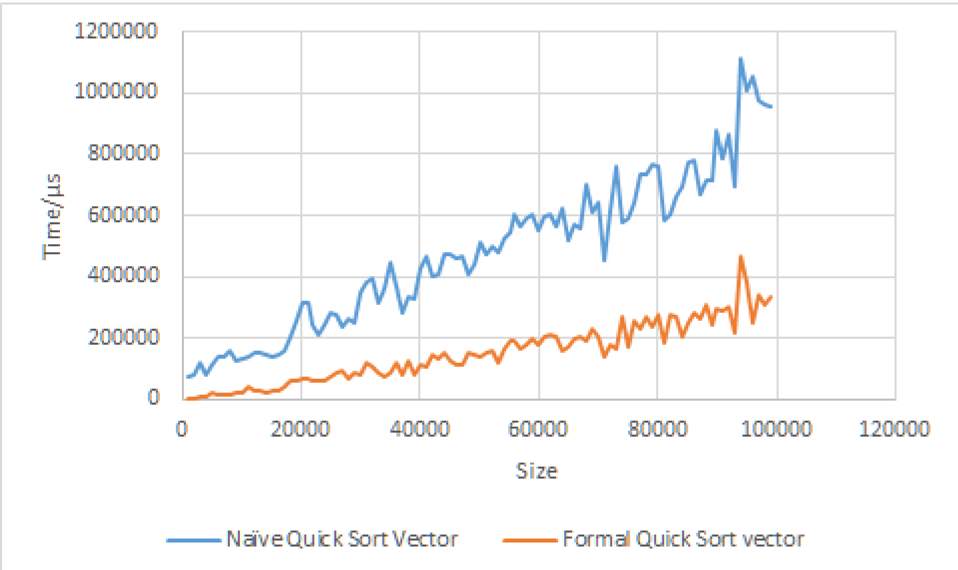
   As we can see from the following figure, the time used by list is much higher than vector.



3. [optional] Implement a better version of quicksort, i.e., one that does in-place swap operations to avoid superfluous memory allocation. **Design and sketch your algorithm with pseudocode before writing any C++ code!**

   See function quicksort().

   From the following figure, the better version of quicksort needs much less time than naive version of quicksort.



Exercises for 5 Jun 2020

1. Implement a class that stores `double` values in a linked list.
2. Turn your class into a class template. Test with types `int`, `double` and `std::string`.
3. [optional] Test your implementation with a non-copyable type such as `std::unique_ptr<double>`. Add a move constructor and move assignment operator to make your implementation work again.