Exercises for 27 May

1. Implement the `insertionSort` function in sorting.h.

2. Analyze the worst-case run-time performance of the merge sort. **Hint:** use the recursive definition of the merge sort to describe the run time of the merge sort for N values (i.e., T(N)), then you may find a telescoping sum to be helpful. If you need some extra hints (**after** attempting it yourself), take a look at p306 of the textbook and/or try watching the video.

   Apparently, T(1) = 1

   $T(n) = n + 2T(\frac{n}{2})$

   We can find that:

   $T(n) = kn + 2^k T(\frac{n}{2^k})$

   $T(n) = nlog_2 n + nT(1)$

   so the worst-case run-time performance of the merge sort is $nlog_2 n + n$.

3. [optional] Analyze the worst-case run-time performance of your insertion sort. **Hint:** the answer should be $O(n^2)$, **even if** the way you designed your algorithm has a doubly-nested `for` loop.

```
template<typename Iter, typename Comparator>
void insertionSort(const Iter& begin, const Iter& end, Comparator compareFn)
{
    for(auto i = begin + 1; i != end; i++)
    {
        //begin with the last element
        auto j = i-1;
        // record the value needed to insert
        auto key = std::move(*i);
        /* Move the top (i-1)th elements, that are greater/less
         * than key, to one position ahead
         * of their current position */
        while(j >= begin && compareFn(key, *j))
        {
            *(j+1) = std::move(*j);
            // move to the previous element
            --j;
        }
        // insert the key to correct position
        *(j+1) = std::move(key);
    }
}
```

   For ith iteration of the first for loop, there will be 3 assignment operations outside the while loop, and
   $i * (2compareoperations + 1logicaloperation + 1assignmentoperation + arithmeticoperation)$ inside the while loop.

   $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$

   Thus, $T(n) = 3n + 4\frac{n(n+1)}{2} = O(n^2)$

Exercises for 29 May

1. Implement the radix sort for integers using a function template and *specialize* your template for strings.

   see function radixSort() and radixSortString() in "sorting.h"

2. [optional] Change the code above to simply `cout << uniqueNames << "\n";`. Implement whatever operator functions are required to make this work.

   skip

3. [optional] Implement the counting/bucket sort for integers.

   skip