Design Document

Chosen Algorithm: Radix Sort

The algorithm chosen for this assignment is **Radix Sort**. The reason for choosing this algorithm is efficiency, Radix Sort has a sort efficiency of O(d.n) for n keys which have d or fewer digits. When d is a constant Radix Sort is as efficient as the best comparison-based sorts, however when the values differ largely Radix is reduced to O(n log n) which still exceeds or equals other sorts such as Insertion, Bubble and Quicksort.
First created basic radix sort based of the diagrams located in the resource on moodle (https://moodle.uowplatform.edu.au/mod/resource/view.php?id=190315). Once completed got times of around 4*10^10 nanoseconds.
By getting rid of obvious inefficiencies such as copying over the vector supplied to the function and using a referenced vector instead times improved by a factor of around 2.5.
First round of optimisation: { 1287081781, 1300494267, 1324498438, …, 1302825054 }.

Optimised further by removing keys that are already sorted from the next pass. Improved times by a factor greater than 6.
Second round of optimisation: { 170806448, 173493628, 173593741, ..., 175326088 }

Using pointers to sort instead of swapping the string values gets another improvement by a factor of greater than 2, now in the 10^8 nanosecond range. Also replacing all at(index) calls on my vectors with operator[index] increased times by magnitude of 0.25.
Third round of optimisation: { 50589586, 51604252, 51806891, …, 52630799 }

Moving the sort function to the main function and not converting the resulting sorted vector of string pointers to string objects makes the algorithm 2 times quicker.
Final round of optimisation: { 22908204, 23129124, 23144789, .., 23153981 }

Adding the -Ofast flag to the make file increase the efficiency by a factor of 3.
Post-final round of optimisation: { 8631110, 8673872, 8674469, …, 8681141 }

Future optimisations, its possible that using character arrays instead of strings could possibly make this sort function considerably more efficient.

Quick Sort

Quick sort is O(n log n). First created a recursive quick sort using a separate compare function.
Times for first implementation: { 3132462626, 3133389060, ..., 3136525117 }
Inlining the comparisons for sort kept times in 10 digit range but improved them slightly. I also tried an iterative approach to quick sort seeing again only a slight increase.
Final round of optimisation: { 2773608062, 2774170559, 2774492667, ..., 2778301047 }
At this point radix seemed like the best option since any further optimisations could be applied to both algorithms while Radix was exceeding Quick Sort by several magnitudes.