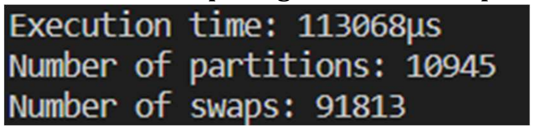
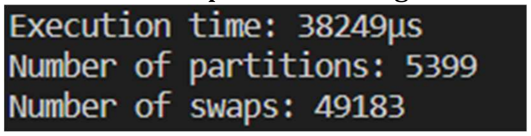
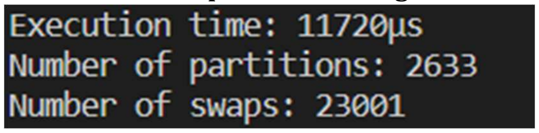
Adil Mahmudlu 150200915

BLG335 HW1 Part 2 Report

a. How2Run
   The code is compiled using VSCode SSH remote connection and C/C++ extensions. To compile the code, starting in H1 folder, writing "g++ src/main.cpp -o bin/main" on command line is required. To run the code, "./bin/main books.csv" need to be written on the command line. The name of the csv file is passed to the program via command line, and therefore it is essential to write it on command line when running the program.
   If there occurs any problem, such as the csv file is missing, or the name of the csv file is not passed through the command line, the according exception is thrown. Otherwise, the program runs normally.
b. Read, Sort, Write
   The program reads the csv file line by line (tuple by tuple), and then reads these lines word by word, and saves them in a vector of string vectors as a table. The table is sorted via quickSort function and written to a new sorted_books.csv file.
c. Sorting all of the table, the time of execution, the number of partitions, and the number of swaps is given as an output:
   
   ```
   Execution time: 113068µs
   Number of partitions: 10945
   Number of swaps: 91813
   ```
   Sorting half of the table, the time of execution, the number of partitions, and the number of swaps is on average:
   
   ```
   Execution time: 38249µs
   Number of partitions: 5399
   Number of swaps: 49183
   ```
   Sorting quarter of the table, the time of execution, the number of partitions, and the number of swaps is on average:
   
   ```
   Execution time: 11720µs
   Number of partitions: 2633
   Number of swaps: 23001
   ```
   The change in the number of data is directly responsible for the change in execution time. As the number of data halved, the execution time is more than halved. The reason behind this result arises is because the time complexity of the quicksort is O(nlogn), and therefore, when n is halved, the execution time decreases by more than 2.
d. Best time complexity of quicksort: O(nlogn)
   When the pivot is the median of the partition, the best time complexity is observed.
   $T(n) = 2T(n/2) + O(n)$
   Using Master Theorem of form $T(n) = aT(n/b) + f(n)$, we find that
   $T(n) = \Theta(n\log n) = O(n\log n)$

Average time complexity of quicksort: $O(n\log n)$
When the pivot is not the first or last element of the sorted data, the average time complexity is expected.
$T(n) = T(n/4) + T(3n/4) + O(n)$
$T(n/4) = T(n/16) + T(3n/16) + O(n/4)$
$T(3n/4) = T(3n/16) + T(9n/16) + O(3n/4)$
$T(n) = T(n/16) + 2T(3n/16) + T(9n/16) + O(n/4) + O(3n/4) + O(n)$
…
$T(n) = nT(1) + n\log n = O(n\log n)$

Worst time complexity of quicksort: $O(n^2)$
When the pivot is chosen to be first or the last element of the sorted data, the worst time complexity is anticipated.
$T(n) = T(n-1) + O(n)$
Using Master theorem of the form $T(n) = aT(n-b) + f(n)$, we find that
$T(n) = nO(n) = O(n^2)$

e. Worst case scenario happens when the pivot is chosen as the maximum or the minimum of the partition. This results in number of partitions being equal to the n instead of logn, and because in each partition the work done is $O(n)$, the time complexity of the quicksort becomes $O(n^2)$, rather than $O(n\log n)$. To prevent such issues, randomizing the pivot can be useful. This significantly lowers the chances of the pivot being chosen as the max or min of the partitioned data.