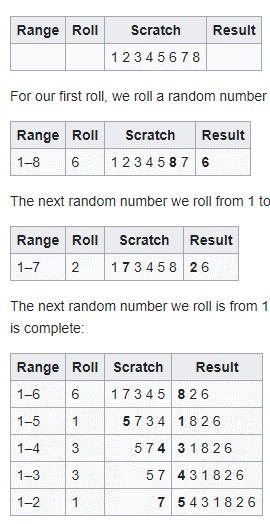
Data Structures & Algorithms Home Assignment

Reuben Xuereb MSD 6.1B

**Section 1**

Task 1:

Fisher-Yates shuffle was originally formed back in 1938 by Ronald Fisher and Frank Yates. They came up with a method called the “Pencil and Paper Method” which simply involved a table of random numbers. Here they select a random number for example from 1 till 8. The number selected then moves to a separate collection and removed from the source collection. This process goes on and on until the source collection is empty. Issues with this method was that when technology evolved and computers finally arrived, computer scientist noticed that the actual time complexity of the algorithm was very high: the big-O notation for the native algorithm was O(n2) in the worst case. Due to this Donald Knuth proposed a new algorithm which is now termed the Knuth shuffle or the “modern” shuffle method. In this new method the solution was instead of striking out the chosen numbers and copying them elsewhere, they are swapped with the last number not yet chosen. For example, we have the range from 1 till 8 as before. For the first roll we roll a 2, so we swap the 2nd and the 8th numbers in the list. The next random number we roll from 1 till 7 is 5, so we swap the 5th and the 7th numbers and continue doing this process until the permutation is complete. A clearer example is shown in the table below:



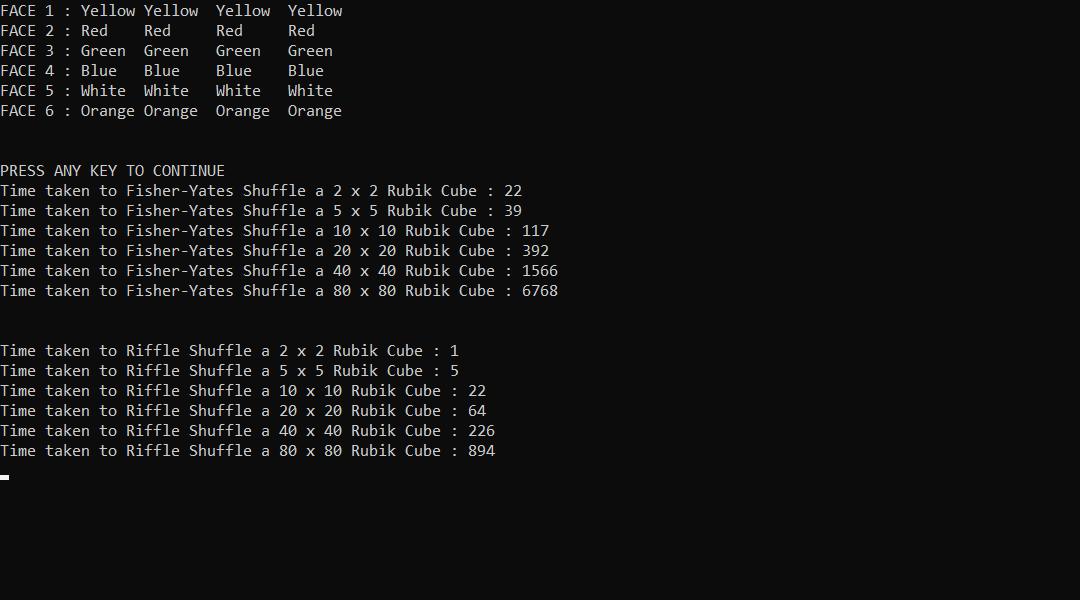
The modern version of the Fisher-Yates shuffle has a time complexity of O(n) as the best case, typical case of O(n), and worst case of O(n). Another common algorithm which is very similar to Fisher-yates is the Sattolo’s Algorithm. The difference between them I very little. One distinctive difference is that Fisher-yates has a loop that repeats itself from 0 to n and Sattolo’s has a loop that repeats itself from 0 to n-1.

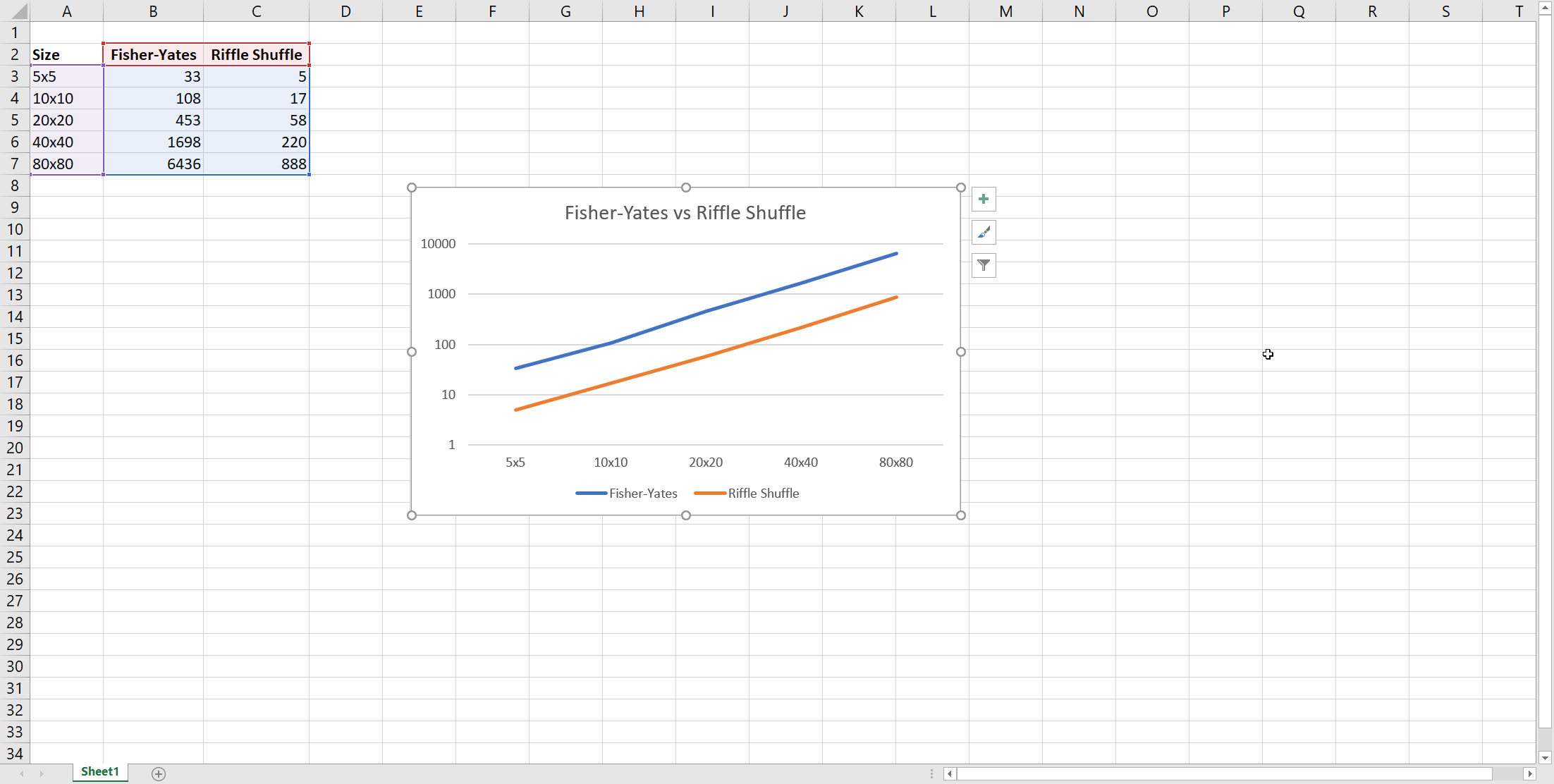
The Riffle Shuffle algorithm is one of the most common ways to shuffle cards. Given a deck of *n* cards, which is then cut into two packets and then the two packets are interleaved from the left and right hands (an ``in-shuffle'') or from the right and left hands (an ``out-shuffle''). In general, card *k* moves to the position originally occupied by the *2kth* card (mod *2n+1*). Therefore, in-shuffling *2n* cards *2n* times (where *2n+1* is [Prime](https://archive.lib.msu.edu/crcmath/math/math/p/p592.htm)) results in the original card order. Similarly, out-shuffling *2n* cards *2n-2* times (where *2n-1* is [Prime](https://archive.lib.msu.edu/crcmath/math/math/p/p592.htm)) results in the original order. This mean that an ordinary deck of 52 cars is retuned to its original order after 7 shuffles. Ramnath and Scully give an algorithm for the shortest sequence of in-and-out shuffles to move a card from arbitrary position *i* to position *j*. This algorithm works for any deck with an even number of cards and has a big-O notation of O(log2n).

Overall, the Fisher-Yates algorithm is the most efficient since having the best time complexity from both algorithms.

Task3:

2).





3). Based on my empirical analysis, the Riffle Shuffle algorithm proved to be the fastest. In reality this should not happen as Fisher-Yates’s algorithm is clearly faster than the Riffle Shuffle algorithm.

**Section 2**

Task1:

For the Insertion() method the best case is O(n) whilst the worst and average cases are O(n2). It has a constant time complexity meaning having a time complexity of O(1). For the heapSort() method all the best, worst and typical cases have a O(nlogn) asymptotic speed. It then has a time complexity of O(1).

**Section 3**

Quick Sort has a worst case of O(n2) and an average case of O(n log n). Merge Sort has its best, worst and average case all O(n log n). My prediction is that Quick Sort will be the fastest as it performers better and is more efficient in small arrays such as we have in our code. If the array was larger, then my prediction would be that the Merge sort would be the fastest as its space complexity is O(n) which is far better than the Quick Sort.

**Section 4**

Task 2:

a.1) Pseudo Random Number:

Numbers that appear to be random but in reality, they are not.

2). True Random Number:

Any one number that is drawn from a set of possible values having a uniform distribution.

3) Periodic:

The random number or a sequence of random numbers will eventually repeat themselves.

4) Deterministic:

The random number or sequence of random numbers can be reproduced if the algorithm and the seed are known.

b) Shuffling Strategies produce a sequence of random numbers and shuffle the numbers in the sequence before outputting it. Ideally, to optimize randomness without effecting the performance too much, only one sequence is used and shuffle that sequence. Therefore, generating a sequence of random numbers and re-arrange the order of these numbers will be impossible to tell which number in the sequence was the first number and which random number was used to generate the next one. Discarding Strategies are used to generate more random numbers than we need and discard some of them. This makes the algorithm less deterministic as the sequence of numbers becomes harder to reproduce as the starting point of the algorithm is not known.

c). To improve my Lagged Fibonacci RNG, I will be using the Johan Dhal Shuffling Algorithm. This will help by shuffling parts of a sequence of elements so that the final shuffled sequence contains elements that have been shuffled and are no longer in their original position and elements that have not been shuffled and are still in their original position.

**Appendix**

1). https://en.wikipedia.org/wiki/Fisher–Yates\_shuffle#Sattolo's\_algorithm

2). https://exceptionnotfound.net/understanding-the-fisher-yates-card-shuffling-algorithm/

3). https://exceptionnotfound.net/understanding-the-fisher-yates-card-shuffling-algorithm/

4). https://archive.lib.msu.edu/crcmath/math/math/r/r325.htm

5). https://stackoverflow.com/questions/61101432/a-7-riffle-shuffle-algorithm-in-c-sharp-for-a-rubiks-cube

6). https://www.c-sharpcorner.com/blogs/a-simple-merge-sort-implementation-c-sharp

7). https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-9.php

8). https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/

9). https://www.geeksforgeeks.org/make-mergesort-perform-comparisons-best-case/

10). https://stackoverflow.com/questions/36376888/seeding-multiple-random-number-generators/36376889

11). <https://dev.to/s_awdesh/everyday-im-shuffling-im-a-card--fire-1f8b>

12). <https://www.interviewcake.com/concept/java/heapsort>

13). https://rosettacode.org/wiki/Card\_shuffles#C.23