Danielle Moore

# Introduction:

The two sorting algorithms I chose were Pancake sorting and selection sorting methods.

Pancake Sorting: In pancake sorting, the object is to limit the amount of reversals required, the sorting algorithm runs through each element of the array and finds the largest. The algorithm then flips the whole array from the start to the largest element found, like a pancake stack. Then restarts the algorithm.

Selection Sorting: In selection sorting, the algorithm runs through the array and finds the largest of the elements and swaps it with the element next to it and continues to swap until it takes the largest element to the end of the array. At that point it restarts from the beginning and repeats.

## Pseudocode for Pancake Sorting:

Flip(array[],i)

Temp,start

while(start<i)

Temp = array[start];

array[start]=array[i];

Array[i] = temp;

Start++;

I- - ;

findMax(array[], n)

Mi, i;

For (mi=0;i=0;i<n;++i)

if(array[i] > array[mi])

mi=i

pancakeSort(array[], n)

for(curr-size = n; curr-size>1; –curr-size)

Int mi = findMax(arr, curr-size)

if(mi!=curr-size-1)

flip(array,mi);

flip(arr, curr-size-1);

## Pseudocode for selection sort

sort(int arr[])

Int n = arr.length;

for(i =0; i< n-1; i++)

min-index=i

for(j=i+1; j<n; j++)

If arr[j] < arr[min-index]

min-index=j

Temp = arr[min-index]

Arr[min-index] = arr[i]

Arr[i] = temp

## Big O Analysis of Pancake Sorting

For each round of sorting, the most flips that would occur would be two. The time complexity of pancake sorting is O(n).

## Big O Analysis of Selection Sorting

There are two loops in the algorithm for selection sorting. The first loop is used to choose the element one by one which would have a time complexity of O(n). The second loop is used to compare the element to the other elements in the array, with a time complexity of O(n). These two loops combined create a time complexity of O(n) \* O(n) which equals O(n^2).

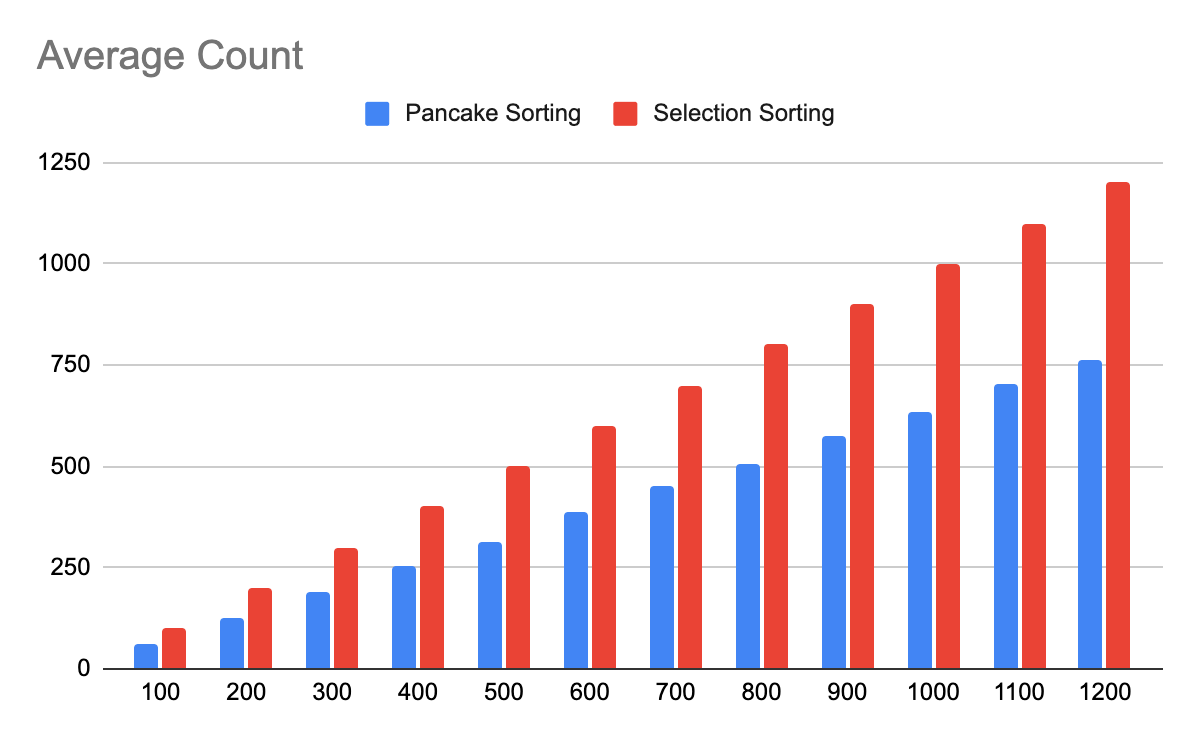
## Problems associated with JVM warmup

Honestly, I did not even think of this as a problem. Although, I’m sure it’s important I still am currently unsure if I did anything to prevent this or not.

## Critical Operations chosen

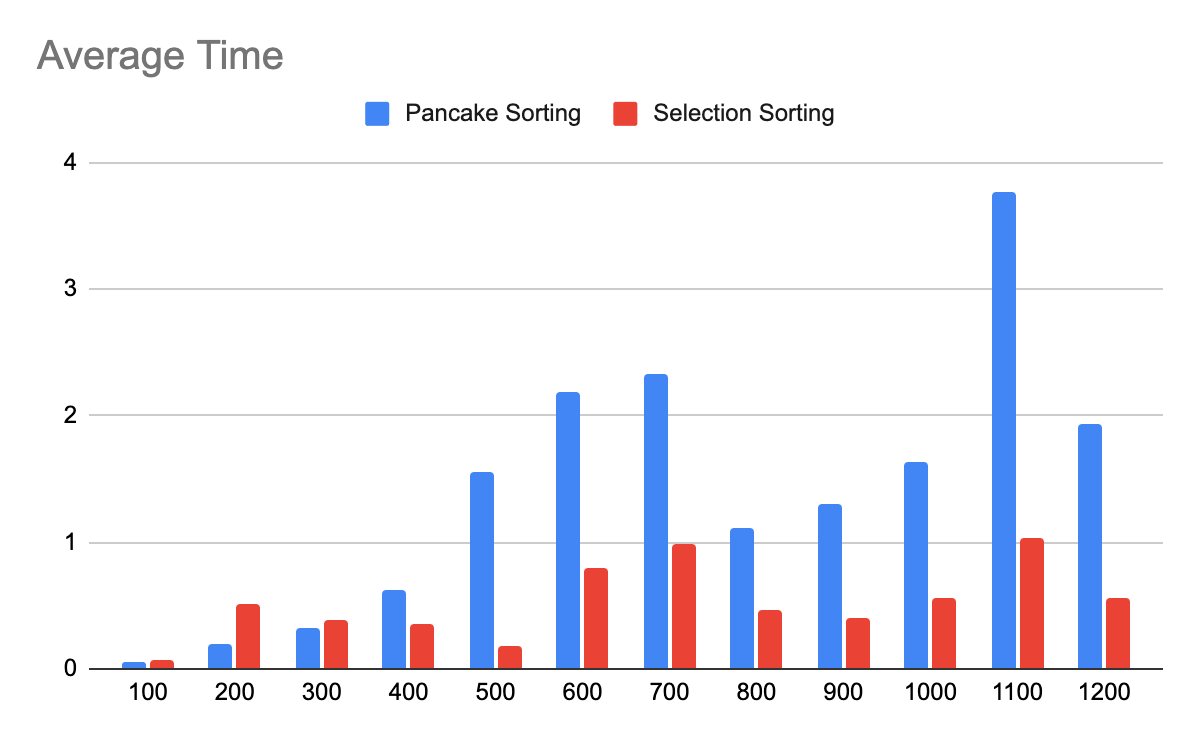
For the pancake sorting I chose to count the first flip as my critical operation, this was because it seemed like one of the most important. If the first flip does not occur, the array should already be sorted. Therefore, I thought that this must be important. For selection sorting, I chose to count when the chosen element got swapped with the first element. I thought this was important because if the elements don’t get swapped it would have already been sorted.

# Average Count



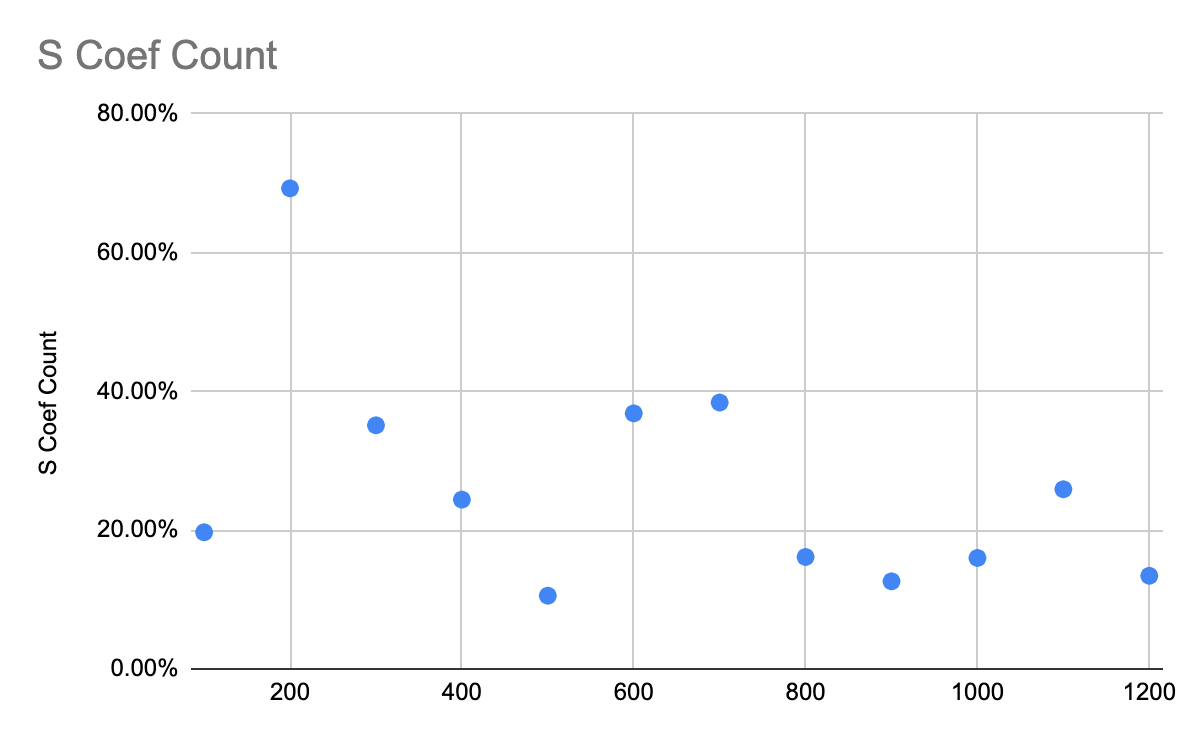
As shown in the graph above, on average selection sort had to swap each element because the average count was the same as the size of the data set given. The pancake sort algorithm did less swaps when tested. This shows the relation between the time complexity, that the pancake sort algorithm took less swaps than the selection sort algorithm.

## Average Time

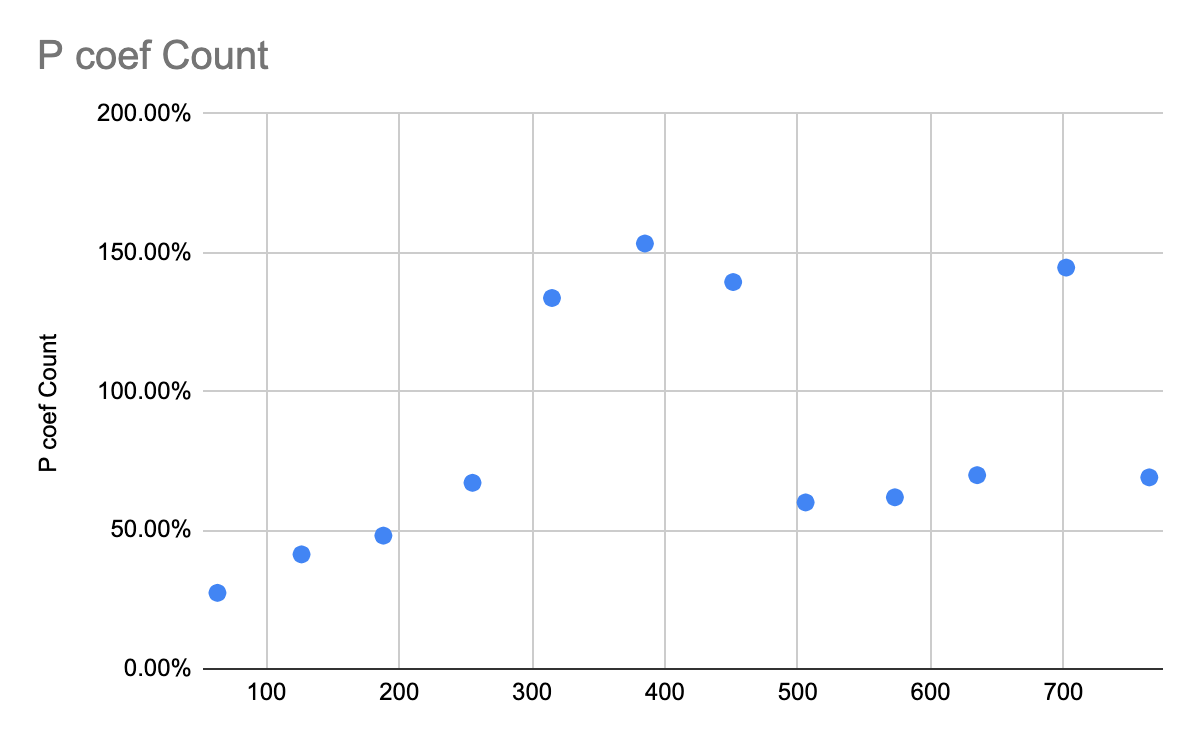


The pancake sorting algorithm took more time than the selection sorting algorithm did in the above chart and the tests I ran. I believe this is because instead of swapping one element for another it has to flip an entire array over. Although there seems to be no immediate pattern of it seeming to take longer for either algorithm to run through the sortings, it does look like at the worst case scenario it would take the longest the bigger the set of elements got.

## Coefficient of Count

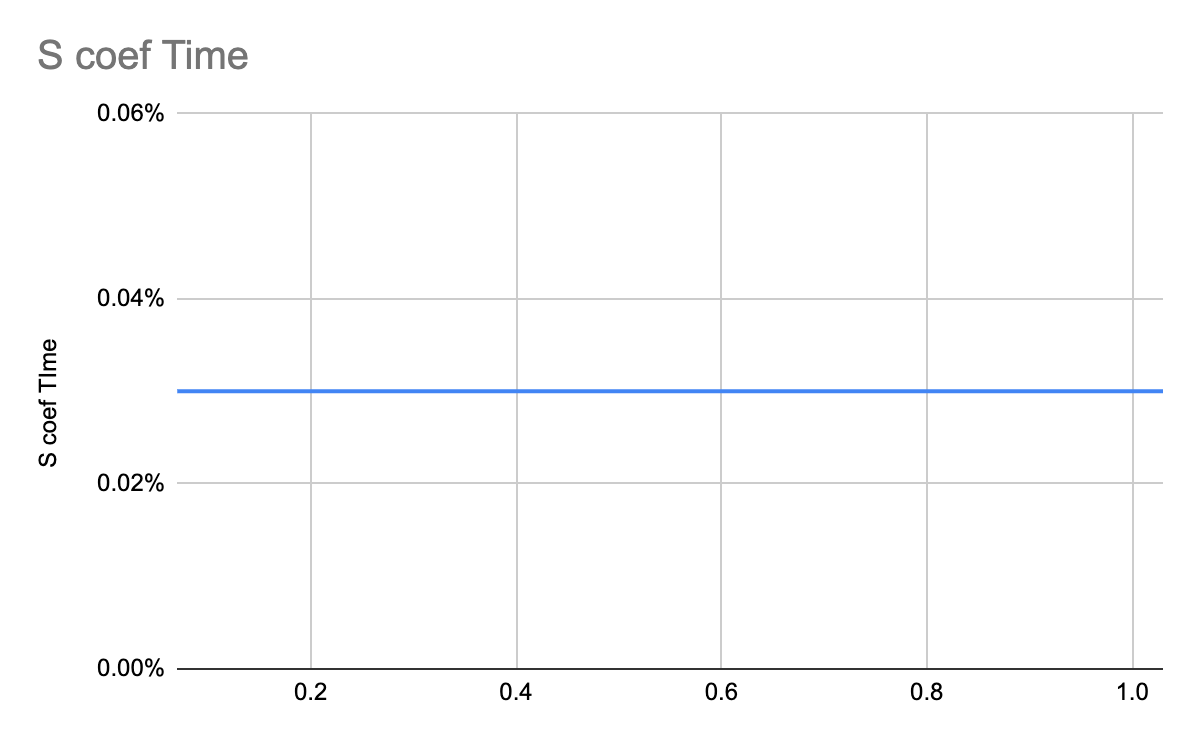


In the selection sorting algorithm, all the coefficients of variance calculations ended up being under 100% which means that the variability in the count number for each of the 40 runs in the 12 different sizes was very small. This makes sense because in selection sorting the algorithm runs through each number so the amount of swaps needed to occur would be relatively close to the same because it doesn’t change the rest of the array if one swap needs to occur.

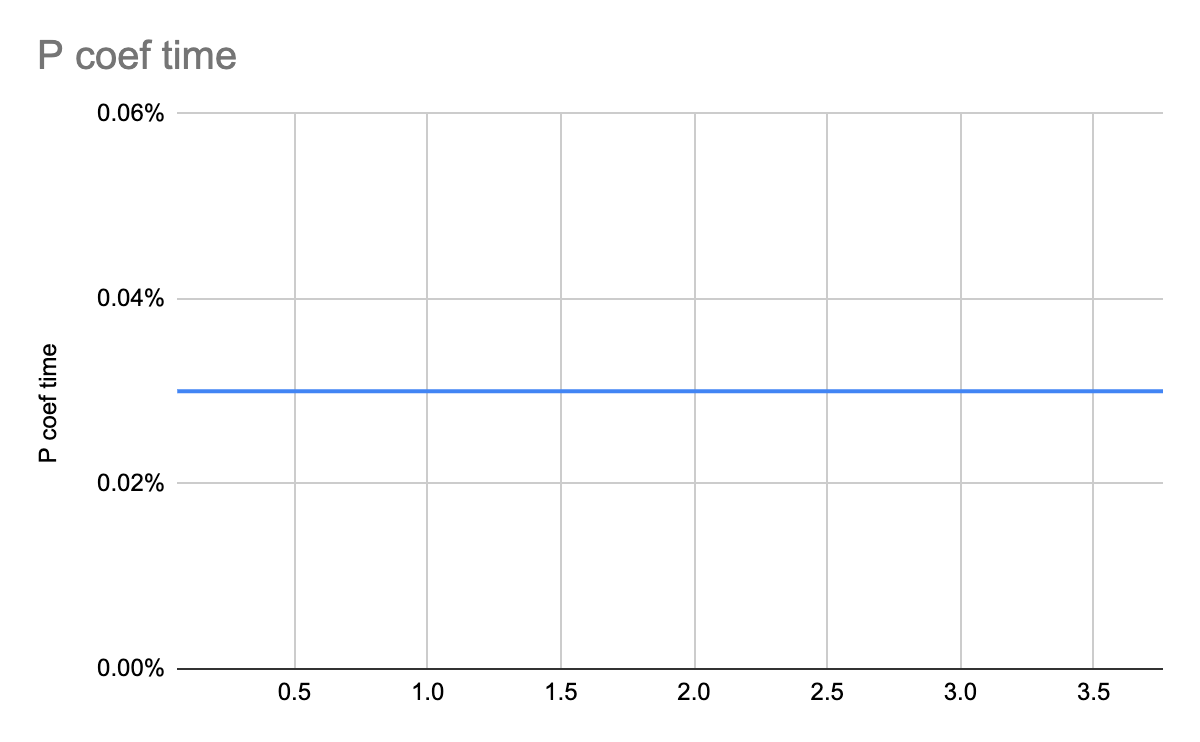


The coefficient of variation for Pancake sorting count was between 27% and 153%. The four data points above show that for some of the tests run there was a high variability between the count numbers. The other runs show that they were under 100% which also means a low variability between the count numbers. Although, this makes sense because the pancake sorting method is not a method that would show low variability depending on the amount of elements present because the flip of the whole stack would change the order in which the numbers got read.

## Coefficient of Variance Time Selection Sort



The coefficient of variance for time in the selection sort algorithm was 0.03% which means that there was very little variability in the time it took to run each sort.



Again the coefficient of variance for time in the pancake sort algorithm was 0.03% which means that the variation in time it took to run each of the sorts for each of the different amounts of elements was very little.

## Conclusion

In conclusion, the two different sorting algorithms I used worked very differently to each other. The pancake sorting algorithm worked well with less swaps but had an increase in the amount of time it took to run each sort on average. The selection sort algorithm did a great job at running quickly but also on average took more swaps.