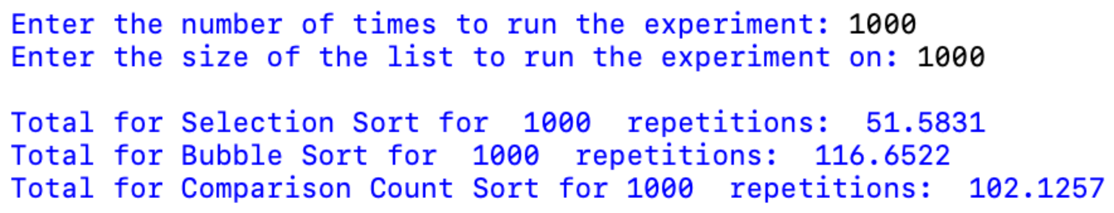
By: Tonia Alderbashi

Optional Project: Sort Wars Report

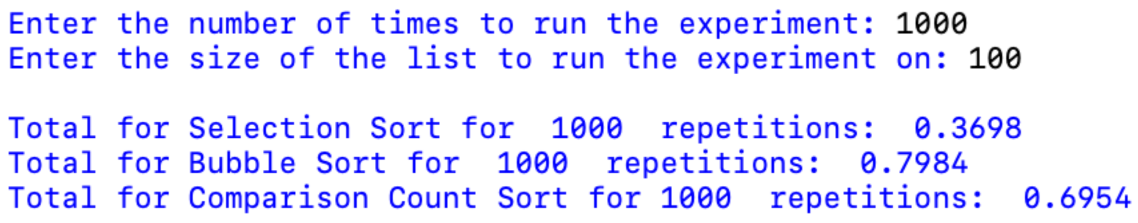
NOTE Where it says, "a list" (whether sort, unsorted or any type of list), it means 1000 runs of sorting the list. Since calculating for one time gave small results that were hard to measure and compare, I ran the algorithm 1000 times on the same list (if it is sorted in any direction), and 1000 times on 1000 different lists for randomly generated (unsorted) lists.*

The best sorting algorithm seems to selection sort. I tested the algorithms on a randomly generated list of multiple different sizes and calculated the amount of time it took each algorithm to sort the lists. Each experiment is performed a certain number of times (that the user specifies) and the total time is returned.

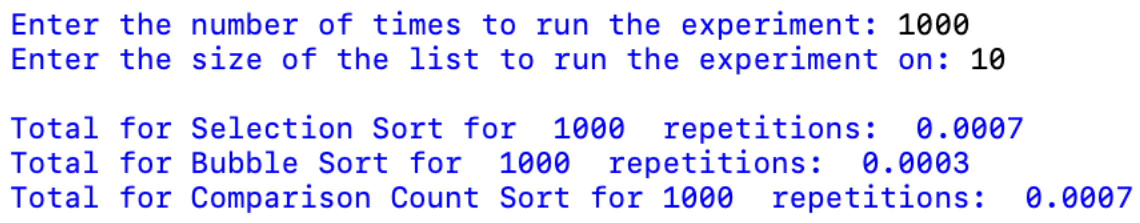
My conclusions are based on the following results:



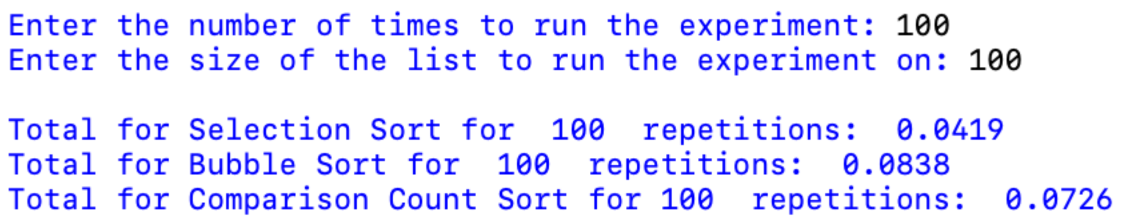
```
Enter the number of times to run the experiment: 1000
Enter the size of the list to run the experiment on: 1000

Total for Selection Sort for  1000  repetitions:  51.5831
Total for Bubble Sort for  1000  repetitions:  116.6522
Total for Comparison Count Sort for 1000  repetitions:  102.1257
```



```
Enter the number of times to run the experiment: 1000
Enter the size of the list to run the experiment on: 100

Total for Selection Sort for  1000  repetitions:  0.3698
Total for Bubble Sort for  1000  repetitions:  0.7984
Total for Comparison Count Sort for 1000  repetitions:  0.6954
```

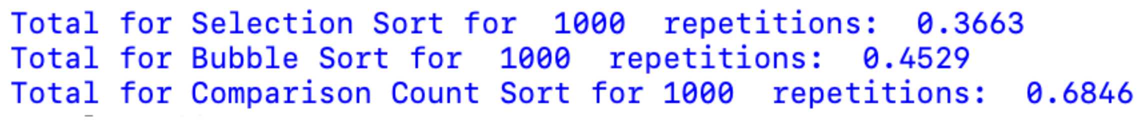


```
Enter the number of times to run the experiment: 1000
Enter the size of the list to run the experiment on: 10

Total for Selection Sort for  1000  repetitions:  0.0007
Total for Bubble Sort for  1000  repetitions:  0.0003
Total for Comparison Count Sort for 1000  repetitions:  0.0007
```



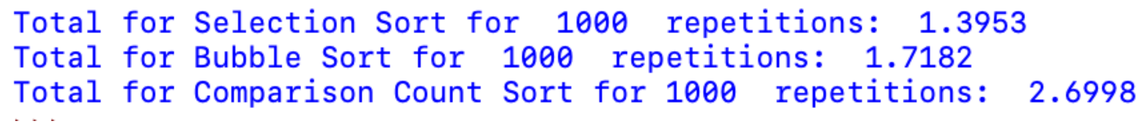
```
Enter the number of times to run the experiment: 100
Enter the size of the list to run the experiment on: 100

Total for Selection Sort for  100  repetitions:  0.0419
Total for Bubble Sort for  100  repetitions:  0.0838
Total for Comparison Count Sort for 100  repetitions:  0.0726
```

The previous experiments were performed on randomly generated lists, the following experiment was performed on a sorted list from 1-100 included. These were the results:



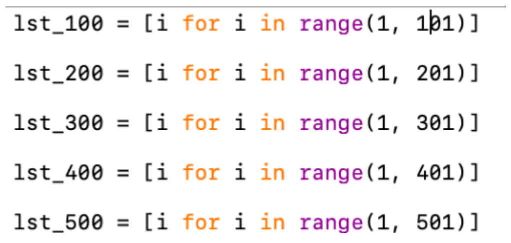
```
Total for Selection Sort for  1000  repetitions:  0.3663
Total for Bubble Sort for  1000  repetitions:  0.4529
Total for Comparison Count Sort for 1000  repetitions:  0.6846
```

Selection and count comparison sort took about the same time to sort a sorted list as they took on an unsorted list, however bubble sort took 43% less time.
I repeated the same experiment (already sorted list) on a list of the numbers 1-200 included. These were the results:



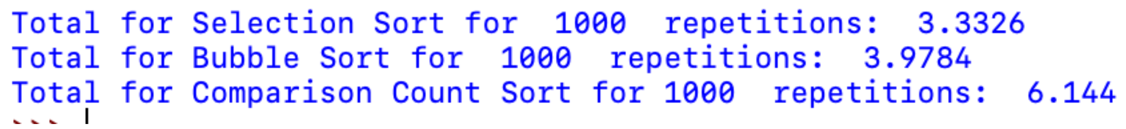
```
Total for Selection Sort for  1000  repetitions:  1.3953
Total for Bubble Sort for  1000  repetitions:  1.7182
Total for Comparison Count Sort for 1000  repetitions:  2.6998
```

I made a new file with sorted lists of various sizes and imported the lists to my program to make easier to work with.



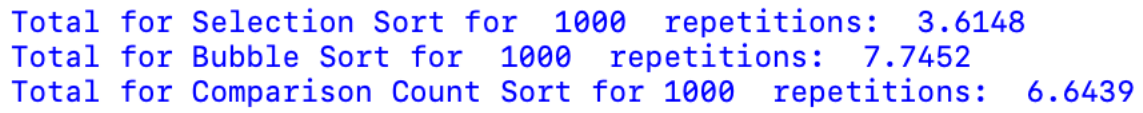
```python
lst_100 = [i for i in range(1, 101)]

lst_200 = [i for i in range(1, 201)]

lst_300 = [i for i in range(1, 301)]

lst_400 = [i for i in range(1, 401)]

lst_500 = [i for i in range(1, 501)]
```

I got the following times for 300 sorted elements:



```
Total for Selection Sort for  1000  repetitions:  3.3326
Total for Bubble Sort for  1000  repetitions:  3.9784
Total for Comparison Count Sort for 1000  repetitions:  6.144
```

And the following for an unsorted list of 300 elements:



```
Total for Selection Sort for  1000  repetitions:  3.6148
Total for Bubble Sort for  1000  repetitions:  7.7452
Total for Comparison Count Sort for 1000  repetitions:  6.6439
```

The biggest difference again is observed with bubble sort.

Then I repeated the experiment with sorted vs. unsorted lists 400 and 500 elements.

These results were obtained from sorting a sorted list of 400 elements:

```
Total for Selection Sort for  1000  repetitions:  6.8755
Total for Bubble Sort for  1000  repetitions:  8.0384
Total for Comparison Count Sort for 1000  repetitions:  12.5581
```

The following results are of sorting an unsorted list of 400 elements:

```
Total for Selection Sort for  1000  repetitions:  6.8369
Total for Bubble Sort for  1000  repetitions:  14.4769
Total for Comparison Count Sort for 1000  repetitions:  12.7717
```

The difference seems to be in bubble sort.

These results were obtained from sorting a sorted list of 500 elements:

```
Total for Selection Sort for  1000  repetitions:  13.1064
Total for Bubble Sort for  1000  repetitions:  15.8405
Total for Comparison Count Sort for 1000  repetitions:  15.1178
```

The following results were obtained by sorting an unsorted list of 500 elements:

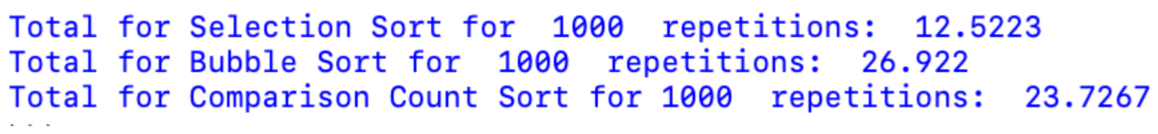
```
Total for Selection Sort for  1000  repetitions:  11.5189
Total for Bubble Sort for  1000  repetitions:  24.6837
Total for Comparison Count Sort for 1000  repetitions:  21.9586
```

The difference is observed in bubble sort and comparison count sort, bubble sort slowed down more. Selection sort took less time and I found that interesting, so I ran the experiment again and got the following results:

```
Total for Selection Sort for  1000  repetitions:  11.7438
Total for Bubble Sort for  1000  repetitions:  25.0906
Total for Comparison Count Sort for 1000  repetitions:  22.2823
```

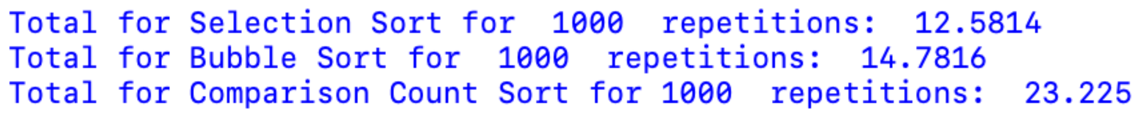
Once again selection sort took less time, so I ran the experiment a few more times. Since the lists are generated randomly and the computer I am running the experiment on might spend more or less time on the same experiment due to other programs running internally, I am considering the possibility of attributing the difference in selection sort on factors other than the algorithm itself.

This is how much it took to sort an unsorted list of 500 elements:



```
Total for Selection Sort for  1000  repetitions:  12.5223
Total for Bubble Sort for  1000  repetitions:  26.922
Total for Comparison Count Sort for 1000  repetitions:  23.7267
```

And this is how much it took to sort an already sorted list:

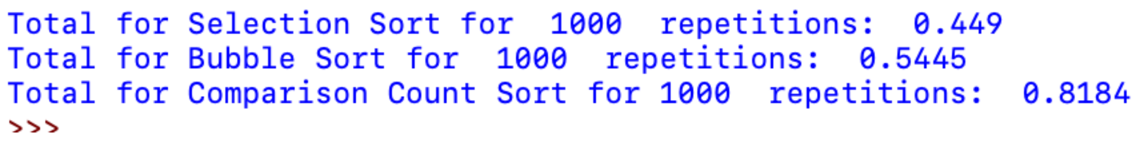


```
Total for Selection Sort for  1000  repetitions:  12.5814
Total for Bubble Sort for  1000  repetitions:  14.7816
Total for Comparison Count Sort for 1000  repetitions:  23.225
```

The time in the last experiment seems about the same.

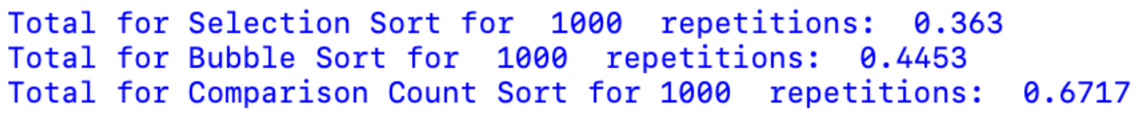
The next experiment is to run the three algorithms on reversed lists of different sizes. (Sorted in non-increasing order).

The results of running the algorithms on an inversely sorted list of 100 elements:



```
Total for Selection Sort for  1000  repetitions:  0.449
Total for Bubble Sort for  1000  repetitions:  0.5445
Total for Comparison Count Sort for 1000  repetitions:  0.8184
>>>
```

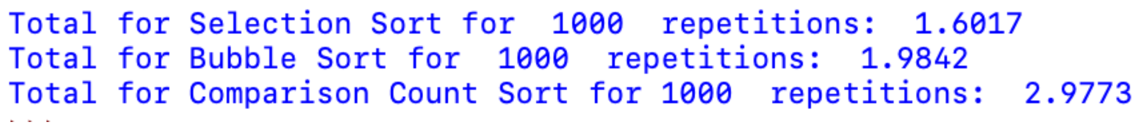
Whereas it took the following amount of time for a sorted list of the same size:



```
Total for Selection Sort for  1000  repetitions:  0.363
Total for Bubble Sort for  1000  repetitions:  0.4453
Total for Comparison Count Sort for 1000  repetitions:  0.6717
```

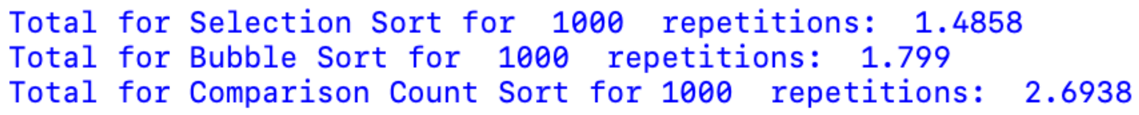
It took a little less time for the sorted list, as expected. I continued the experiment with lists of other sizes:

Reversed list of 200 elements:



```
Total for Selection Sort for  1000  repetitions:  1.6017
Total for Bubble Sort for  1000  repetitions:  1.9842
Total for Comparison Count Sort for 1000  repetitions:  2.9773
```

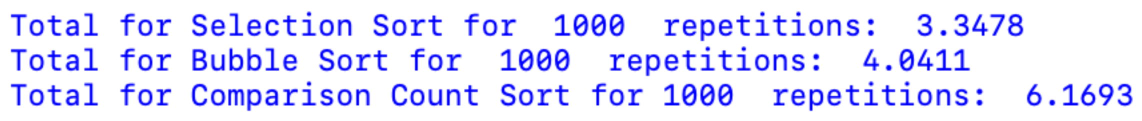
Sorted list of 200 elements:



```
Total for Selection Sort for  1000  repetitions:  1.4858
Total for Bubble Sort for  1000  repetitions:  1.799
Total for Comparison Count Sort for 1000  repetitions:  2.6938
```

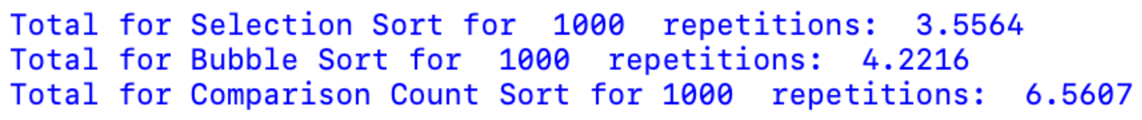
Took less time for the sorted list, as expected.

Reversed list of 300 elements:



```
Total for Selection Sort for  1000  repetitions:  3.3478
Total for Bubble Sort for  1000  repetitions:  4.0411
Total for Comparison Count Sort for 1000  repetitions:  6.1693
```

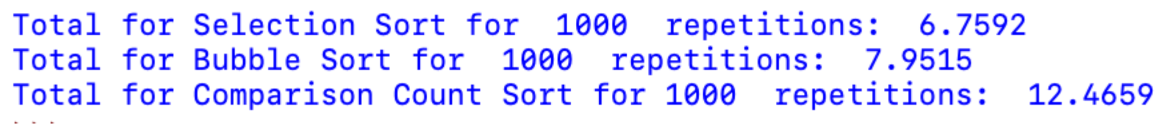
Sorted list of 300 elements:



```
Total for Selection Sort for  1000  repetitions:  3.5564
Total for Bubble Sort for  1000  repetitions:  4.2216
Total for Comparison Count Sort for 1000  repetitions:  6.5607
```

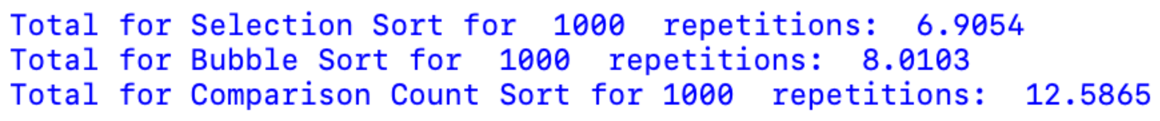
This experiment took more time to sort an already sorted list.

Reversed list of 400 elements:



```
Total for Selection Sort for  1000  repetitions:  6.7592
Total for Bubble Sort for  1000  repetitions:  7.9515
Total for Comparison Count Sort for 1000  repetitions:  12.4659
```

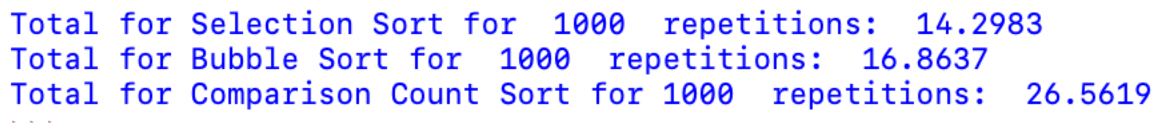
Sorted list of 400 elements:



```
Total for Selection Sort for  1000  repetitions:  6.9054
Total for Bubble Sort for  1000  repetitions:  8.0103
Total for Comparison Count Sort for 1000  repetitions:  12.5865
```

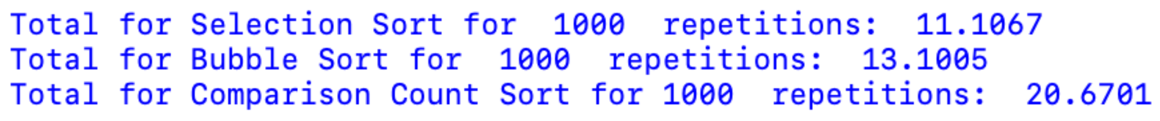
The difference is more noticeable with longer lists. It takes longer to sort an already sorted list for all three algorithms.

Reversed list of 500 elements:



```
Total for Selection Sort for  1000  repetitions:  14.2983
Total for Bubble Sort for  1000  repetitions:  16.8637
Total for Comparison Count Sort for 1000  repetitions:  26.5619
```

Sorted list of 500 elements:

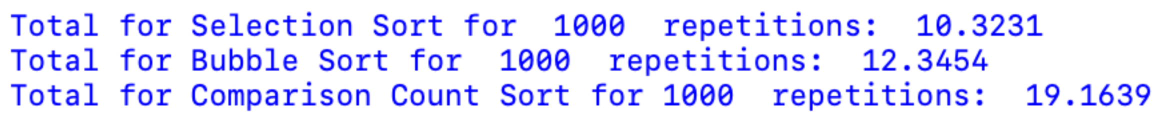


```
Total for Selection Sort for  1000  repetitions:  11.1067
Total for Bubble Sort for  1000  repetitions:  13.1005
Total for Comparison Count Sort for 1000  repetitions:  20.6701
```

In the case of 500 elements the data supports the intuitive assumption that the reversed list would take more time than a sorted list.

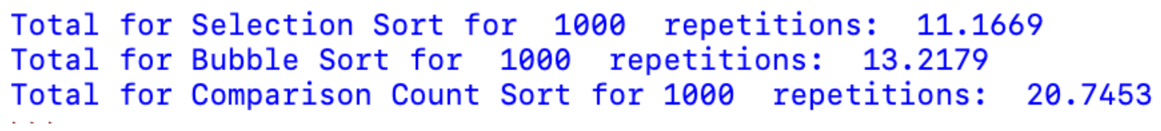
The next experiment is to put a 1 in a random place between 0 and 25 in one list and compare the run time to a list with one in a random place between 475 and 500, while the rest of the elements are zero in both lists.

List of 0s with a one in a random position between 0 and 25:



```
Total for Selection Sort for  1000  repetitions:  10.3231
Total for Bubble Sort for  1000  repetitions:  12.3454
Total for Comparison Count Sort for 1000  repetitions:  19.1639
```

List of 0s with a one in a random position between 475 and 500:



```
Total for Selection Sort for  1000  repetitions:  11.1669
Total for Bubble Sort for  1000  repetitions:  13.2179
Total for Comparison Count Sort for 1000  repetitions:  20.7453
```

It took more time to sort a list with the one at the end, as expected.

These experiments showed the relationship between the three algorithms and how each differs from the others in different scenarios, and how the size or different ordering of the input affected the same algorithm.

In conclusion, selection sort seemed to perform the best in almost all scenarios, except when it comes to short lists, of the size less than 100, where only sometimes it would take longer than bubble sort, but that can be attributed to factors outside of the algorithm time complexity and efficiency. Therefore, selection sort seems to be our best sorting algorithm so far.

Selection sort is the fastest so the “best” does not change, however the second best does.

Bubble sort seemed to generally perform better than comparison count sort, except for unsorted lists of 400 and 500 elements. However, bubble sort performed better when the same lists were already sorted.