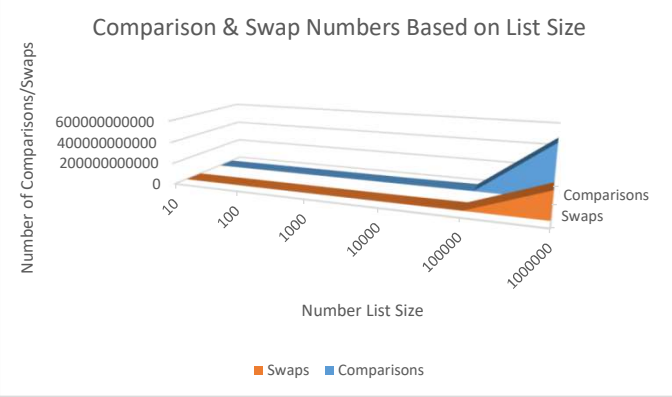
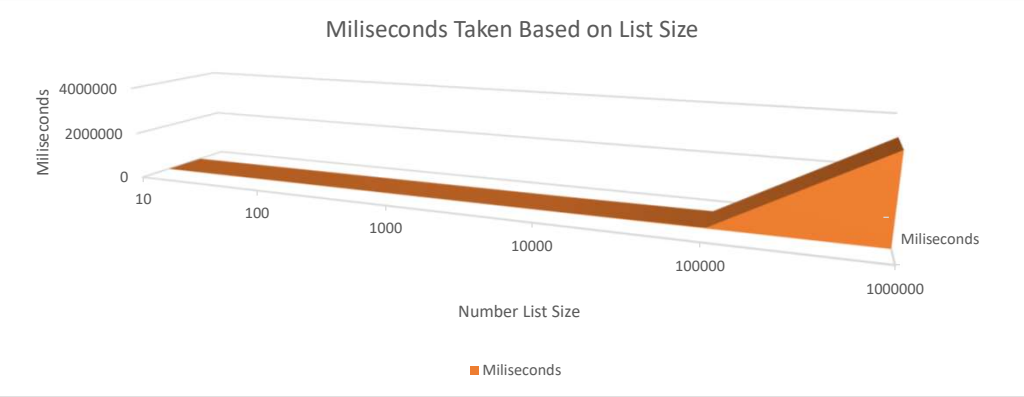
BubbleSort Analysis Sheet

Number List Size	10	100	1000	10000	100000	1000000
Miliseconds	0	0	27	312	30720	3430563
Swaps	24	2765	246734	25026299	2481035035	247375710056
Comparisons	45	4950	499500	49995000	4999950000	499999500000

Average Time (Miliseconds): 576937 (9.61561667 Minutes)





Output:

```
Sorting first 10 elements
Comparisons: 45
Swaps: 24
FINAL LIST: 0 Milliseconds

Sorting first 100 elements
Comparisons: 4950
Swaps: 2765
FINAL LIST: 0 Milliseconds

Sorting first 1000 elements
Comparisons: 499500
Swaps: 246734
FINAL LIST: 27 Milliseconds

Sorting first 10000 elements
Comparisons: 49995000
Swaps: 25026299
FINAL LIST: 312 Milliseconds

Sorting first 100000 elements
Comparisons: 4999950000
Swaps: 2481035035
FINAL LIST: 30720 Milliseconds

Sorting first 1000000 elements
Comparisons: 499999500000
Swaps: 247375710056
FINAL LIST: 3430563 Milliseconds
```

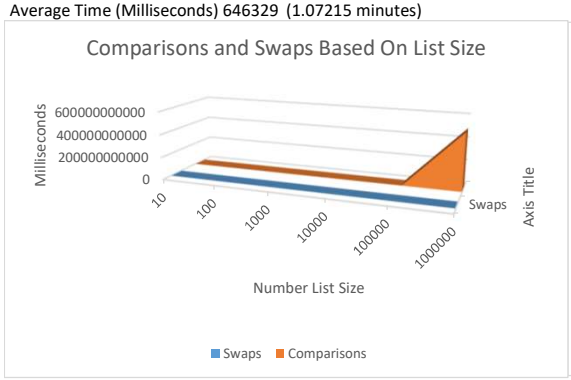
Analysis Notes:

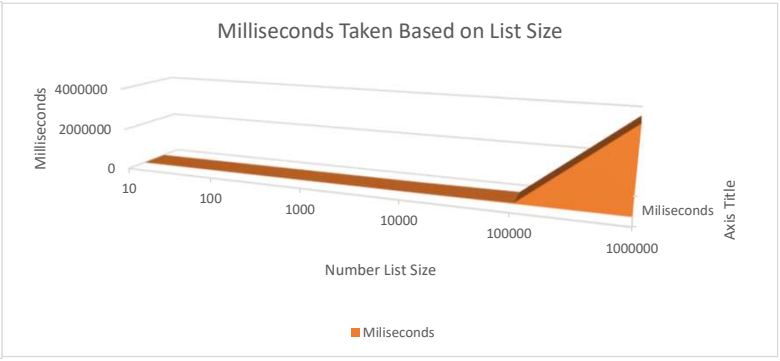
This method isn't exactly the most efficient when it comes to larger quantities of data. It succeeds in both time and comparison/swap efficiency until the number size gets to be about 100000. In the smaller number lists, the time is under 1 second, the swap and comparison numbers do increase but not enough to degrade the time. After 100000, the times are 30.72 seconds (100000) and 57.17605 minutes (1000000). In addition to this, the swap and comparison numbers get so large you have to change the data types from integers to longs. Collecting the data wasn't much of a challenge for me, it took a while for the 1000000 number list (see 57.17605 time), but this could be an issue if this was a program a company wanted to use to sort data. Also, if the computer that's being used to sort the data doesn't have a powerful enough processor, the program can crash, resulting in potential data loss. For better efficiency, I suggest using a method that doesn't need to compare the same item multiple times and can instead sort the number into its place (for example: Selection Sort or Quick Sort).

SelectionSort Analysis Sheet

Number List Size	10	100	1000	10000	100000	1000000
Miliseconds	0	0	20	225	37335	3840391
Swaps	12	339	5672	76458	1003967	12292032
Comparisons	45	4950	499500	49995000	499995000	499999500000

Average Time (Milliseconds) 646329 (1.07215 minutes)





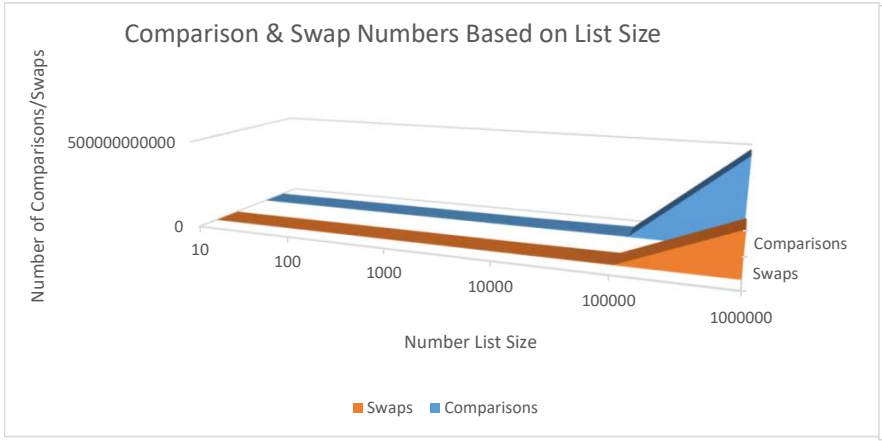
Output:

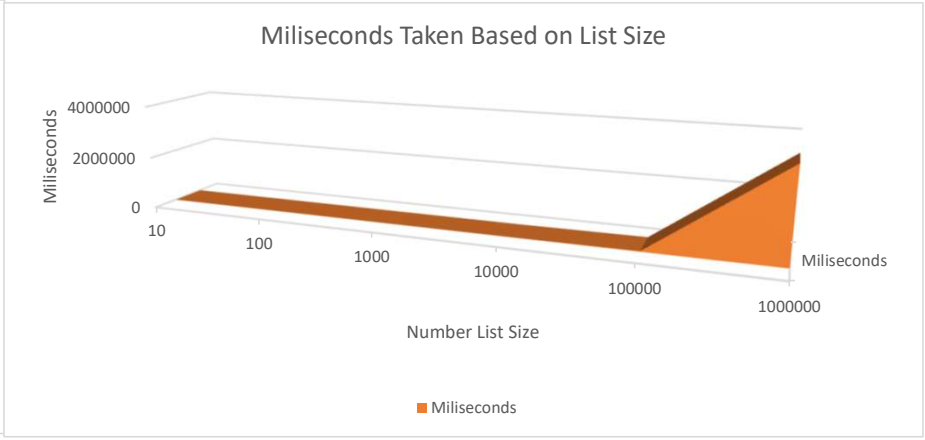
```
Sorting first 10 elements
Comparisons: 45
Swaps: 12
FINAL LIST: 0 Milliseconds

Sorting first 100 elements
Comparisons: 4950
Swaps: 339
FINAL LIST: 0 Milliseconds

Sorting first 1000 elements
Comparisons: 499500
Swaps: 5672
FINAL LIST: 20 Milliseconds

Sorting first 10000 elements
Comparisons: 49995000
Swaps: 76458
FINAL LIST: 225 Milliseconds

Sorting first 100000 elements
Comparisons: 4999950000
Swaps: 1003967
FINAL LIST: 37335 Milliseconds

Sorting first 1000000 elements
Comparisons: 499999500000
Swaps: 12292032
FINAL LIST: 3840391 Milliseconds
```

Analysis:

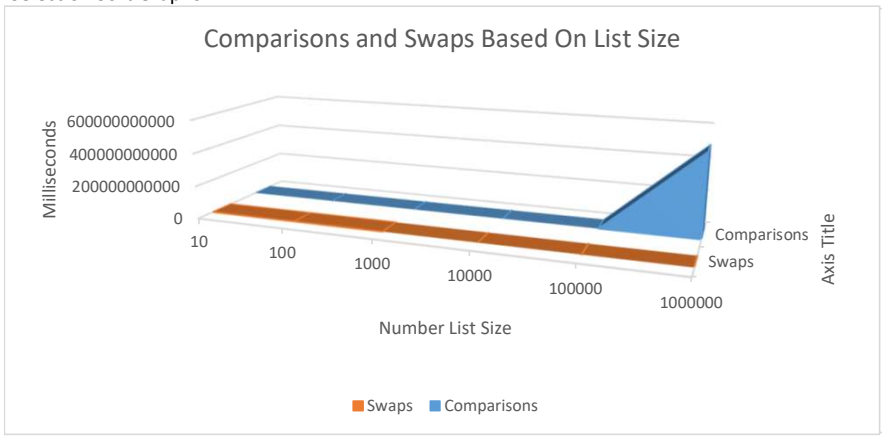
This method isn't still the most efficient when it comes to larger quantities of data. It succeeds in both time and comparison/swap efficiency until the number size gets to be about 100000. In the smaller number lists, the time is under 1 second, the swap and comparison numbers do increase but not enough to degrade the time. After 100000, the times are 37.335 seconds (100000) and 64.00651 minutes (1000000). Collecting the data wasn't much of a challenge for me, it took a while for the 1000000 number list (see 64.00651 time), but this could be an issue if this was a program a company wanted to use to sort data. Also, if the computer that's being used to sort the data doesn't have a powerful enough processor, the program can crash, resulting in potential data loss. For better efficiency, I suggest using a method that doesn't need to compare the same item multiple times and can instead sort the

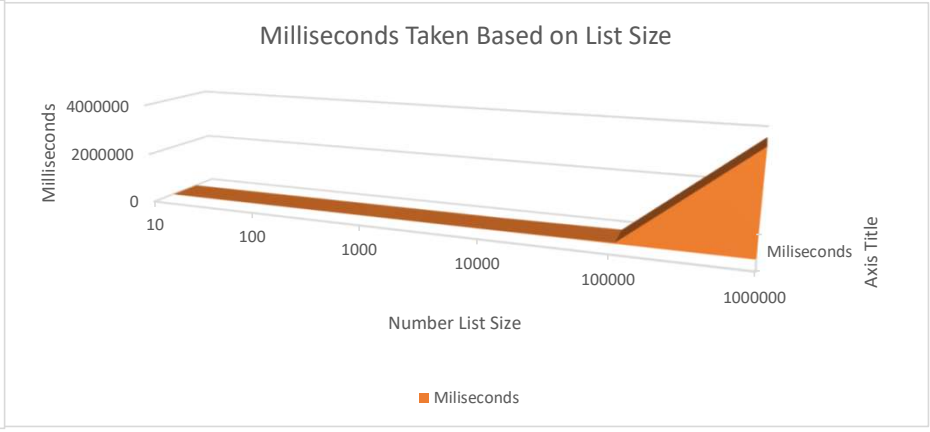
BubbleSort Graphs:





Selection Sort Graphs:





Comparison Analysis:

Both methods are not the most efficient when it comes to larger quantities of data. However, it seems to be that BubbleSort does better with time, having shorter times. SelectionSort succeeds in comparison and swap numbers. They both succeed in time and comparison/swap efficiency until the number size gets to be about 100000. In the smaller number lists, the time is roughly under 1 second, the swap and comparison numbers do increase but not enough to degrade the time. After 100000, the times are much longer though, taking about 30 seconds to an hour. However, at larger numbers, the Selection Sort does significantly better at keeping swap/comparison numbers low.