Theoretical Complexity Calculations

Select-kth-Algorithm1:

T(n) = k + 1 + $\sum_{i=1}^{k} n - i + 1 + 24\sum_{i=1}^{k} n - i + 4$
T(n) = -1/2k(k-2n-1) – 12k(k-2n+1) + k + 1 + 4
T(n) = -1/2k^2 + nk + 1/2k – 12k^2 + 24nk – 12k + k + 1 + 4
T(n) = -12.5k^2 + 25nk – 10.5k + 5
T(n) = -312.5 + 125n – 52.5 + 5
T(n) = 125n – 360

O(n) – linear time complexity

Select-kth-Algorithm2:

T(n) = k + 1 + 21k + $\sum_{i=0}^{k-2} n - i + 9\sum_{i=0}^{k-2} n - i - 1 + 4$
T(n) = 22k + 5 + -1/2(k-1)(k-2n-2) + 9(-1/2(k-1)(k-2n))
T(n) = 22k + 5 + -1/2(k^2 – 2kn-2k-k+2n+2) + 9(-1/2k^2 + kn + 1/2k – n)
T(n) = 22k + 5 – 1/2k^2 + kn + 1.5k – n – 1 – 4.5k^2 + 9kn + 4.5k – 9n
T(n) = 28k + 4 – 5k^2 + 10kn – 10n
T(n) = 40n + 19

O(n) – linear time complexity

Select-kth-Algorithm3:

T(n) = k + 1 + 6k + 1 + k + 9k + n – k + 1 + 1 + 14(n-k+1) + k + 9(k-1) + 4
T(n) = 16k + 4 + n + 14n – 14k + 14 + k + 9k – 9 + 4
T(n) = 12k + 13 + 15n
T(n) = 15n + 73

O(n) – linear time complexity

Select-kth-Algorithm4:

Partition T(n) = n + 1 + 24n + 40
Partition T(n) = 25n + 41

T(base case) = 25n + 58
T(n) = T(n/2) + 25n + 54

Level	Level#	Executions	Input Size	Work	Tot. Work
0	0	1	n	25n+54	25n+54
1	1	1	n/2	25 (n/2)+54	25 (n/2)+54
2	2	1	n/4	25 (n/4)+54	25 (n/4)+54
Above Base	$\log_2$ n-1	$1^{\log_2 n-1}$	$n/2^{\log_2 n-1}$	25 ($n/2^{\log_2 n-1}$)+54	25 ($n/2^{\log_2 n-1}$)+54
Base	$\log_2$ n	$1^{\log_2 n}$	$n/2^{\log_2 n}$	25 ($n/2^{\log_2 n}$)+54	25 ($n/2^{\log_2 n}$)+54

T(n)= 25$\sum_{i=0}^{\log n-1} n/2^i$ + 54
T(n) = 25(2-2 $^{\log_2 n-1}$) + 54
T(n)= 50n – 29

O(n) – linear time complexity

Questions

Does the empirical complexity graph for each algorithm match its theoretical complexity graph in its general shape? Why or why not?

> When using the small input sizes (n=10 to n=1000) the curves for the four algorithms were all over the place and no conclusions could be made. When I made the input sizes bigger (n=10000 to n=190000) the curves became much more clear, they were all linear. This does match the theoretical complexity that I calculated, as expected.

How are the 4 algorithms ordered from least to most efficient in terms of measured time? Is the order the same for small input sizes (n around 10) versus large input sizes (n close to 1000)? Why or why not?

> By looking at Graph 5.1 we can see that for very small input sizes (n=10 to n=1000) Algorithm 4 is by far the most efficient while Algorithm 1, 2, and 3 are all pretty close to the same. When we look at Graph 5.2 though we can see that at some point Algorithm 4 crosses over Algorithm 3 and they all become a little closer. So for lager inputs (n=10000 to n=190000) Algorithm 1 is the worst, followed by Algorithm 2, then Algorithm 4, and Algorithm 3 is the most efficient.

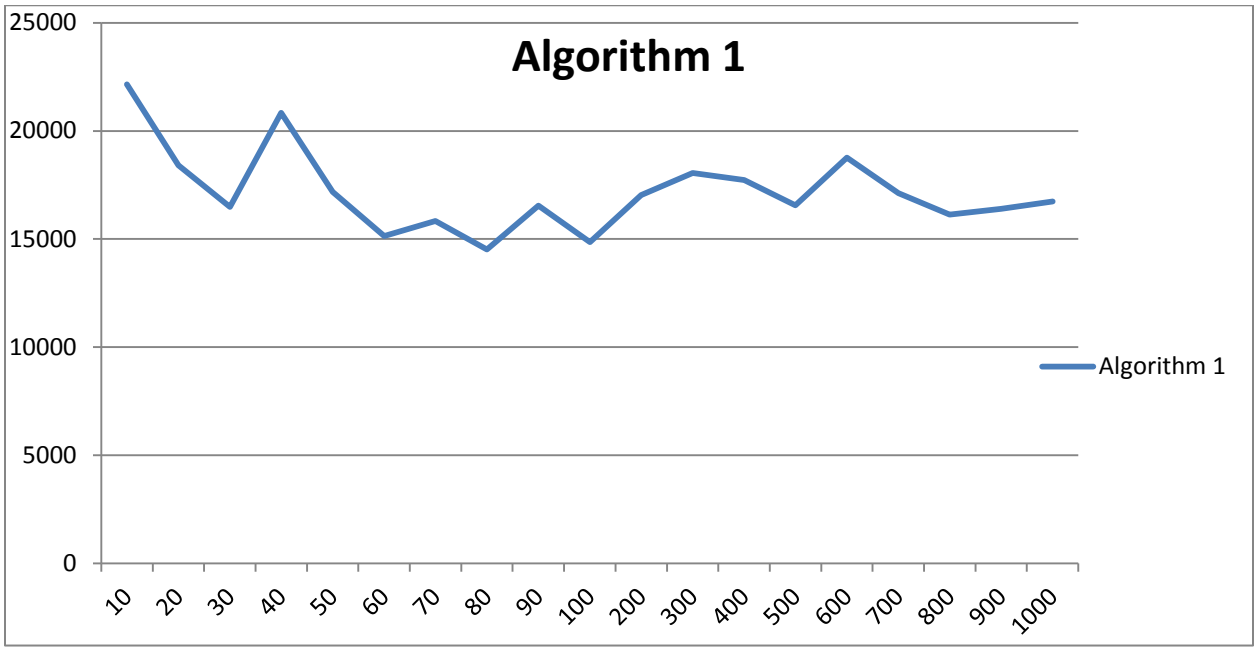
How are the 4 algorithms ordered from least to most efficient in terms of $T_i(n)$? (6) Is the order the same for small input sizes (n around 10) versus large input sizes (n close to 1000)?

> The ordering from least to most efficient in terms of T(n) is the same as it is for the measured time.

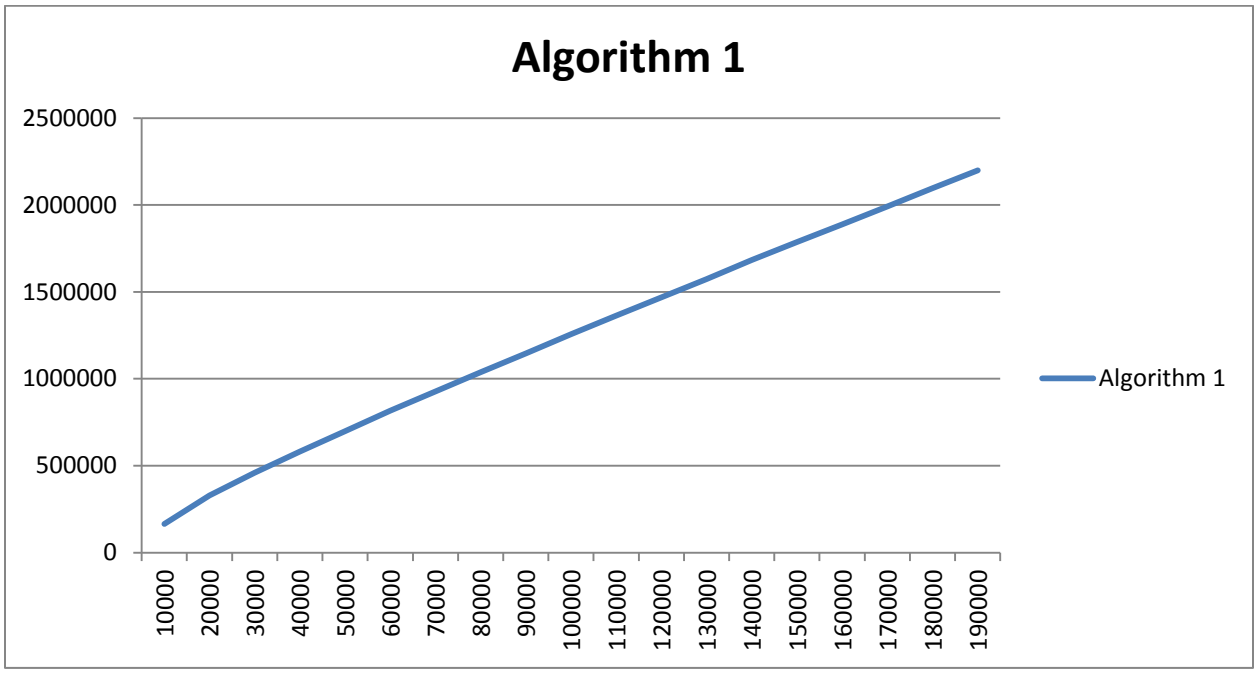
Further Explanation

When I began running the algorithms on different inputs with n=10 to n=1000 I saw very mixed results. The data and graphs were not consistent even when my "m" value was massive. I believe that this is because even an array of size 1000 was not pushing the limits of any of the algorithms and I was looking at too small of a data set. Within the very small scope of 10 to 1000 it was difficult to really see what the differences in the algorithms were. In that scope the times were going up and down due to Windows not being a real time operating system but in the larger range it was easy to see how over a big range the algorithms were linear. I needed to see how the algorithms ran in a much larger range so I made n=10000 and incremented it to n=190000 and once I did this the algorithms really started to show their strengths and weaknesses and the lines started separating. I began to see curves that made sense and showed that they were linear. I made comments in my code to allow the user to run the smaller range (n=10 to n=1000) or the larger range (n=10000 to 190000). Lines 48 to 61 in Programming_Assignment.java is where this is done. I included the graphs for both ranges.

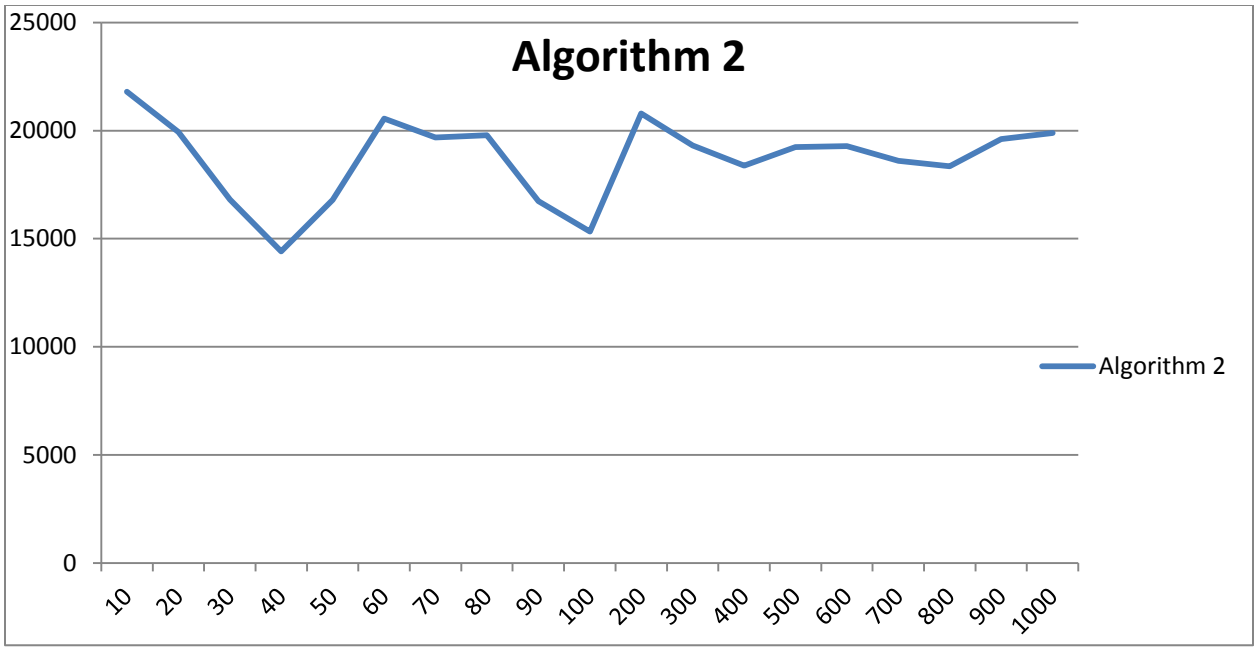
Graphs



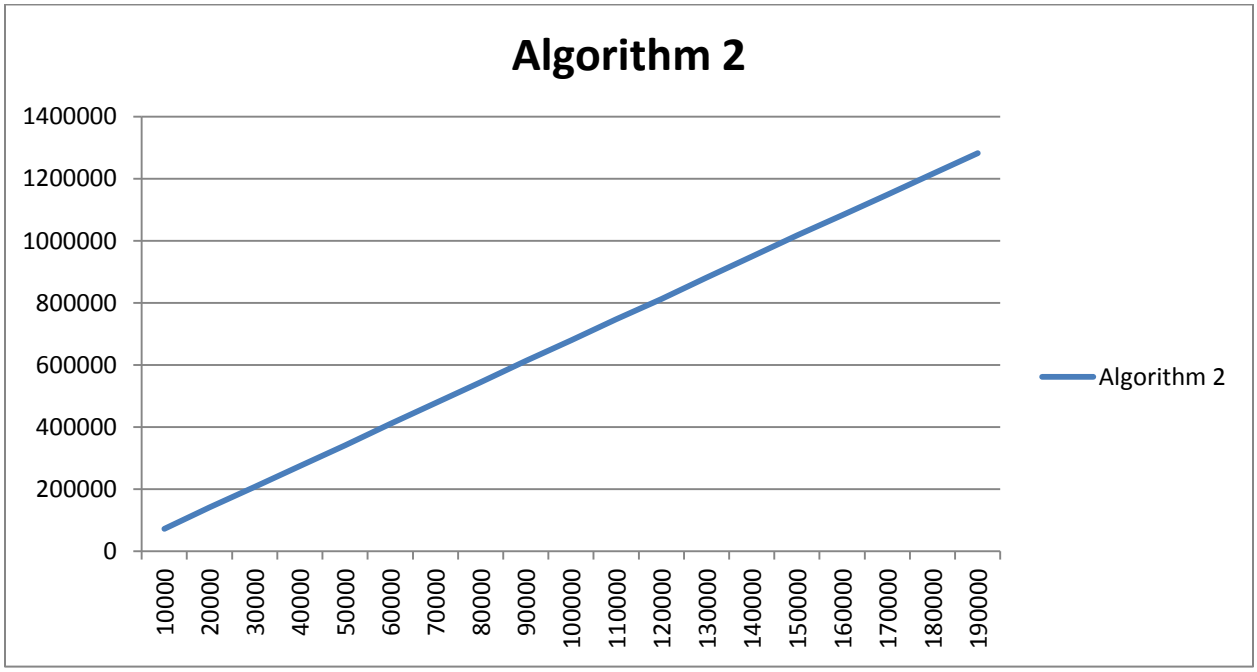
Graph 1.1: Small Data Sets (n=10 to n=1000)



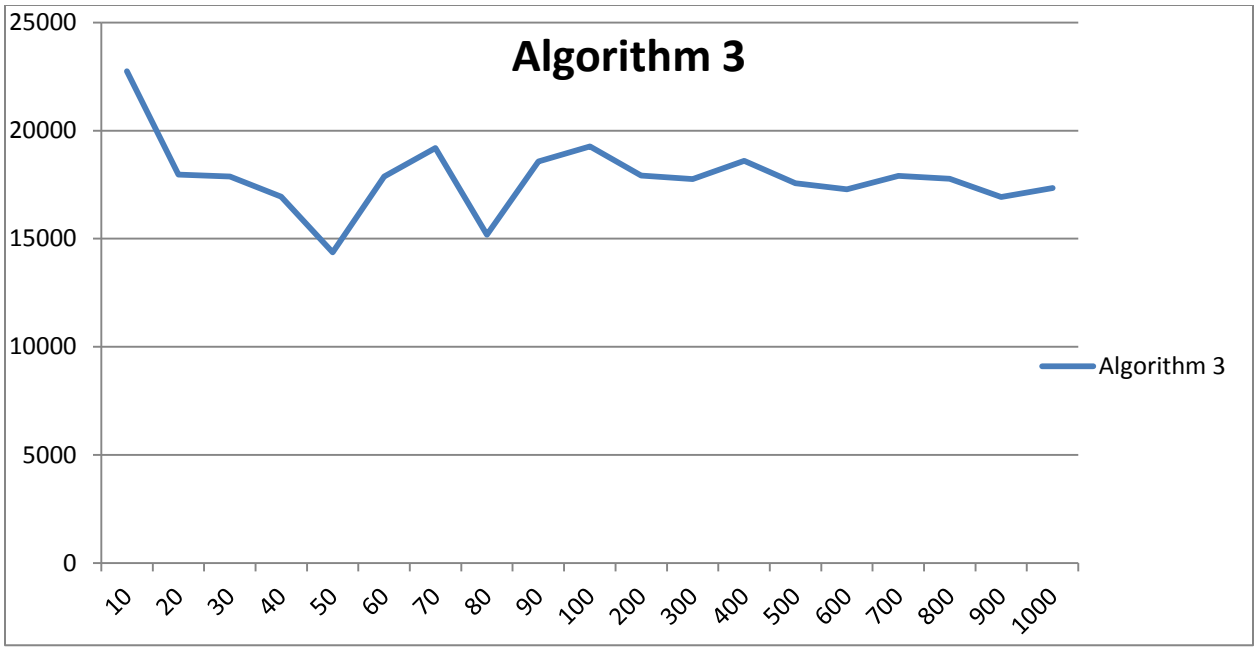
Graph 1.2: Large Data Set (n=10000 to n=190000)



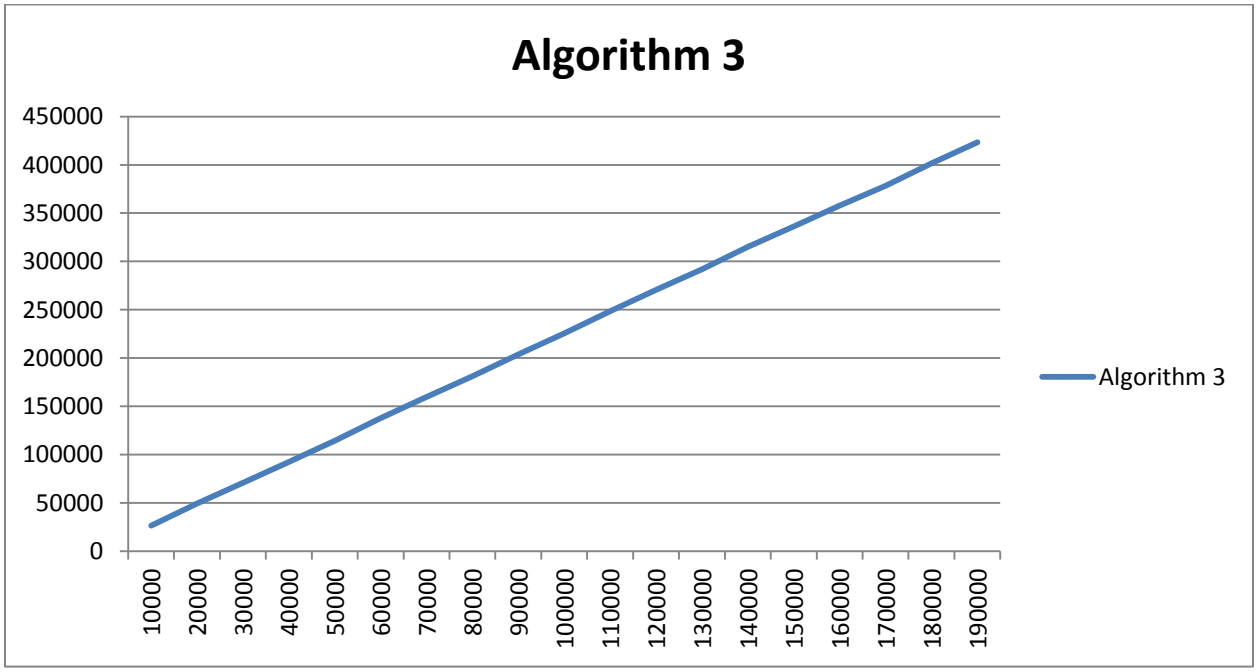
Graph 2.1: Small Data Sets (n=10 to n=1000)



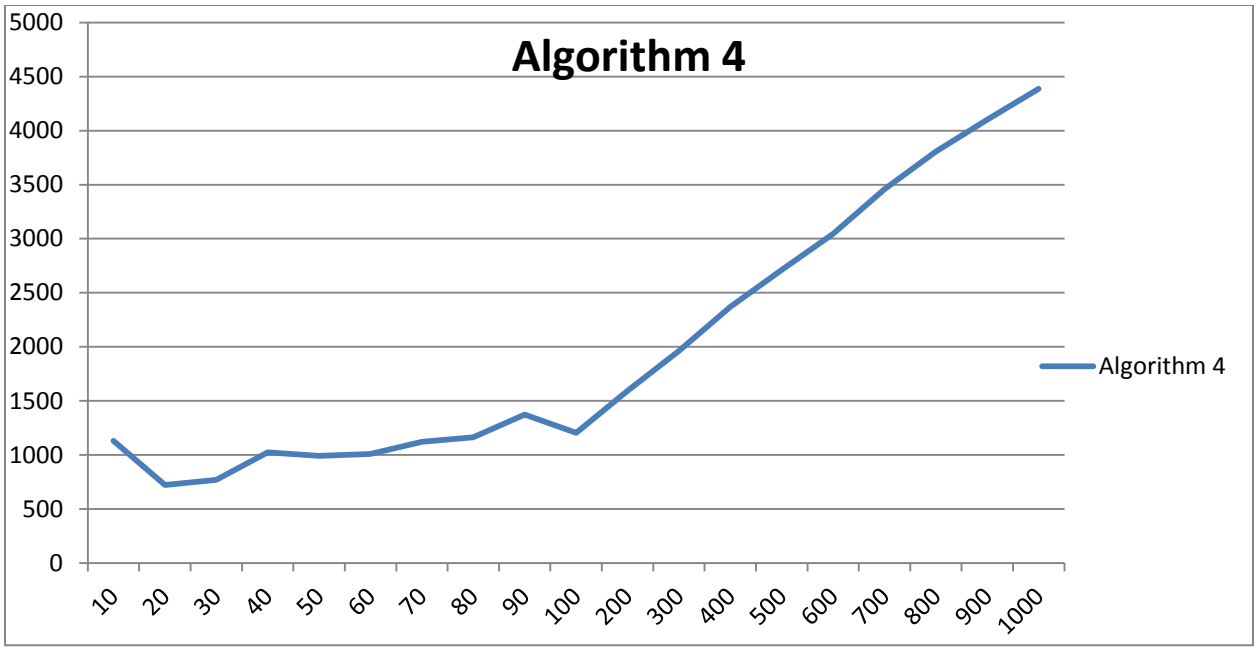
Graph 2.2: Large Data Sets (n=10000 to n=190000)



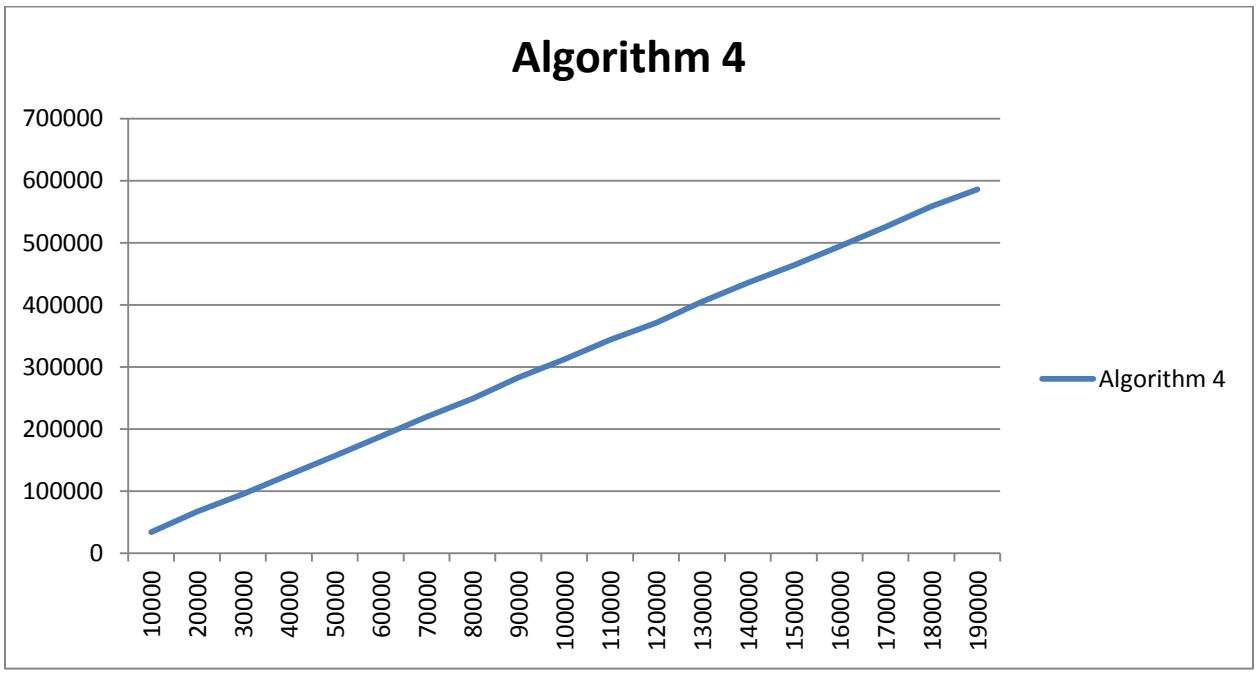
Graph 3.1: Small Data Sets (n=10 to n=1000)



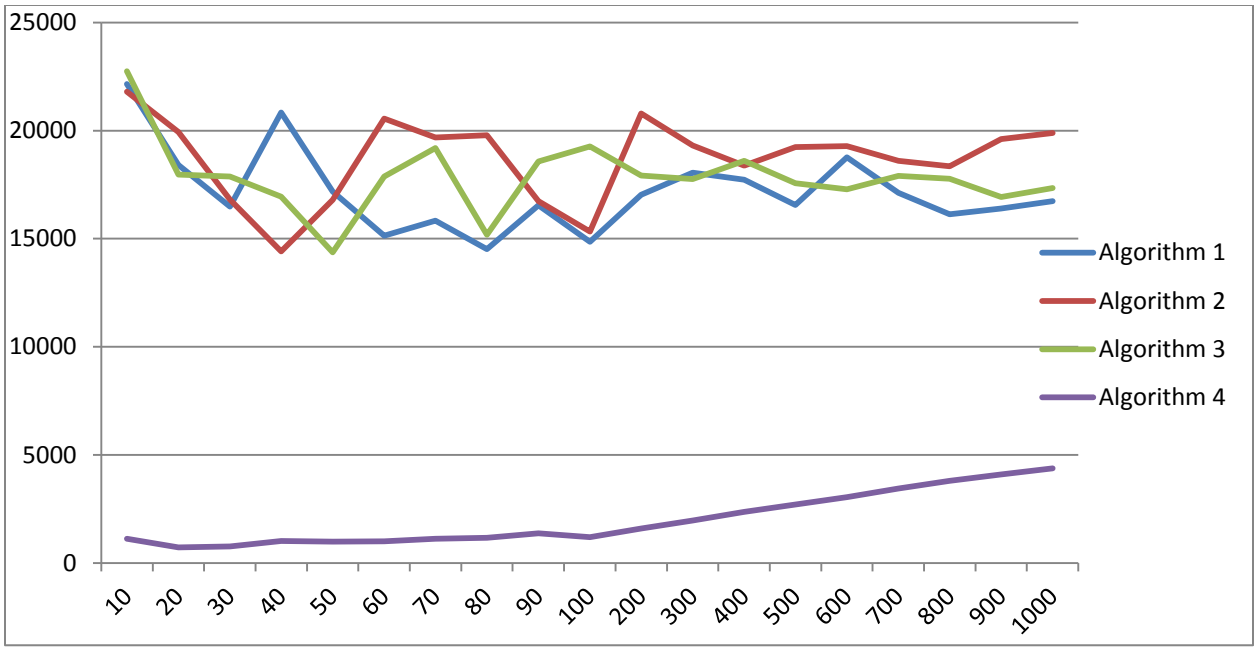
Graph 3.2: Large Data Sets (n=10000 to n=190000)



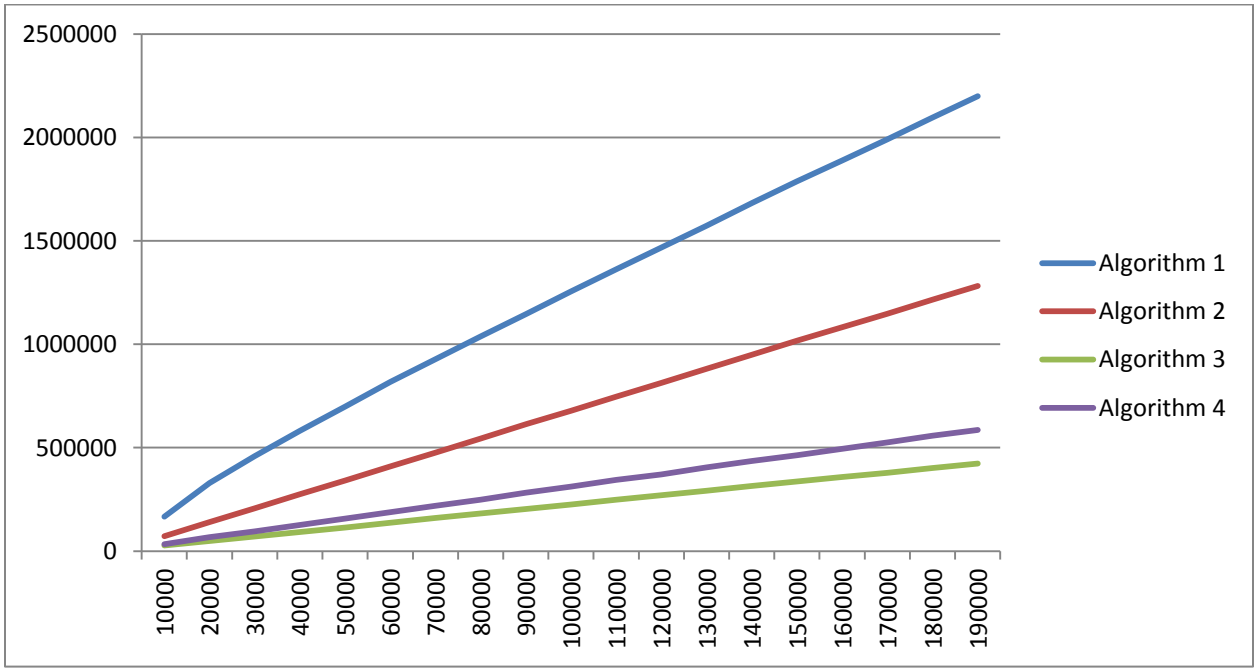
Graph 4.1: Small Data Sets (n=10 to n=1000)



Graph 4.2: Large Data Sets (n=10000 to n=190000)



Graph 5.1: Small Data Sets (n=10 to n=1000)



Graph 5.2: Large Data Sets (n=10000 to n =190000)