Group 29 -
Benjamin Tate
Eli Goodwin
Shae Judge
CS 325 -- Section 400
Project 1 Report

Theoretical Run-time Analysis:

<u>Algorithm 1: Enumeration</u>

Theoretical runtime for algo1: $O(n^3)$

```
Function algo1(A[0...n-1]):
    Initialize maxSum = 0
    If n = 1:
        maxSum = A[0]
        Return maxSum
    For i = 0 … (n-1):
        For j = i … (n-1):
            Initialize tempSum = 0
            For k = i … j:
                tempSum += A[k]
                If tempSum > maxSum:
                    maxSum = tempSum
    Return maxSum
```

<u>Algorithm 2: Better Enumeration</u>

Theoretical runtime for algo2: $O(n^2)$

```
Function algo2(A[0...n-1]):
    Initialize maxSum = 0
    If n = 1:
        maxSum = A[0]
        Return maxSum
    For i = 0 … (n-1):
        Initialize tempSum = A[i]
        For j = (i+1) … (n-1):
            tempSum += A[j]
            If tempSum > maxSum:
                maxSum = tempSum
    Return maxSum
```

<u>Algorithm 3: Divide and Conquer</u>

Theoretical runtime for algo3: O(nlgn)

```
Function algo3(A[0...n-1]):
    Initialize maxSum = minimum int
```

```
If n = 1:
        maxSum = A[0]
        Return maxSum
Initialize mid = n / 2
Left[] = A[0...mid-1]
Right[] = A[mid...n-1]

//Recursive calls for cases 1 and 2
left_maxSum = algo3(left)
right_maxSum = algo3(right)
Lr_maxSum = max(left_maxSum, right_maxSum)

Initialize tempSum = 0
Initialize l_sum = r_sum = minimum int

//For case 3:
For i = (mid-1) … 0: //Loop backwards through left half of A
        tempSum += A[i]
        If tempSum > l_sum:
                L_sum = tempSum
For i = mid … (n-1):
        tempSum += A[i]
        If tempSum > r_sum:
                R_sum = tempSum

//Determining which case to use
if lr_maxSum >= l_sum + r_sum:
        //Case 1 or 2
        maxSum = lr_maxSum
Else:
        //Case 3
        maxSum = l_sum + r_sum

Return maxSum
```

<u>Algorithm 4 Pseudocode:</u>

Theoretical Run-Time Analysis for Algorithm 4: O(n)

```
For i=0 to size-1
        (from the last iteration) Make subarray arr1 or max sum that ends with index i-1

        Find subarray arr2 or max sum that ends with index i. This will be arr1, or the
        subarray of just i.
```

(from the last iteration) Make subarray arr3 of max sum be between indexes 0 and i-1.

Find subarray of maxsum between indexes 0 and i. This will be arr2 or arr3.

Testing:
Testing the validity of the algorithms was done using the provided test set contained in MSS_TestProblems.txt. Arrays from the test set were fed to the algorithms and their output was compared with the known results contained in MSS_TestResults.txt. For each algorithm, their output matched perfectly with already known results demonstrating the correctness of the algorithms.

Experimental Analysis:

1. Average running times:

n	Algo1 Average Runtime (microseconds)
100	2698
200	16888
300	35834
400	71381
500	138665
600	238797
700	376276
800	550983
900	795115
1000	1086242

n	Algo2 Average Runtime (microseconds)
100	46
200	183
500	1133

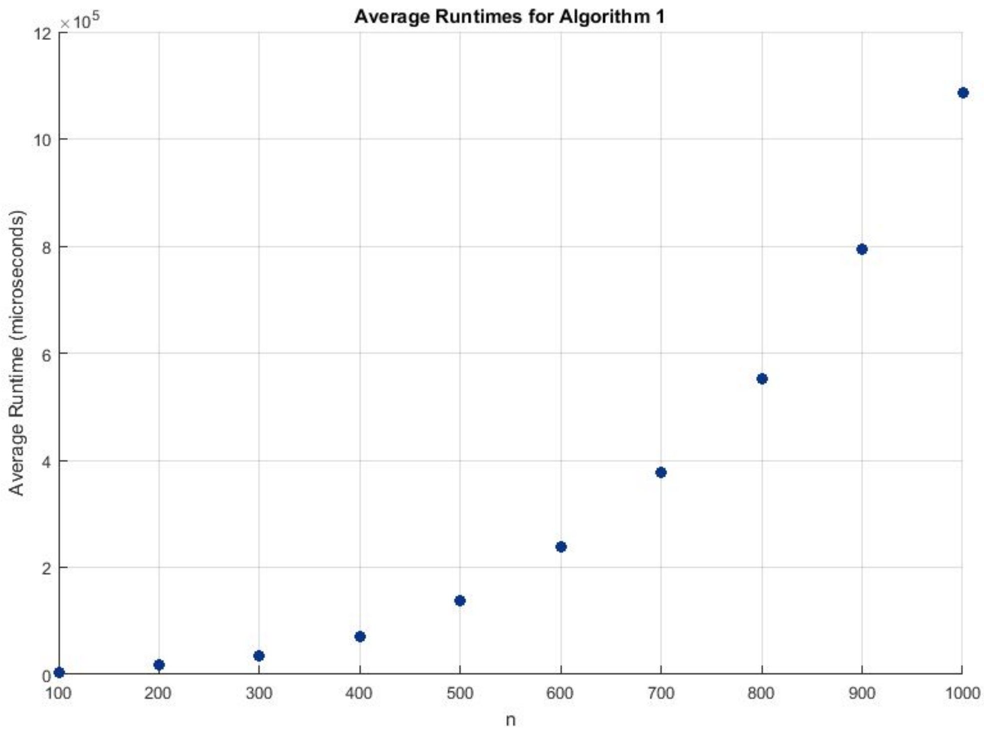
1000	4530
2000	25319
5000	117379
10000	454838
20000	1864108
50000	11750929
100000	46634559

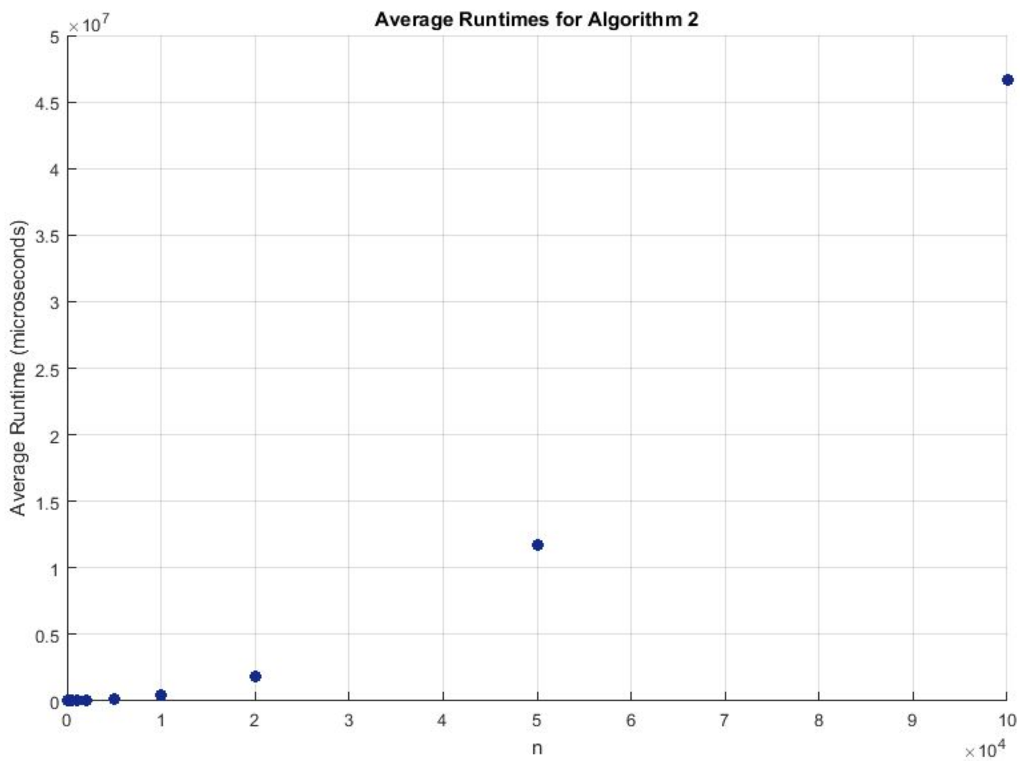
n	Algo3 Average Runtime (microseconds)
20000	22574
50000	35207
100000	72930
200000	123246
500000	312900
1000000	631114
2000000	1289305
5000000	3215504
10000000	6522413
20000000	13179061

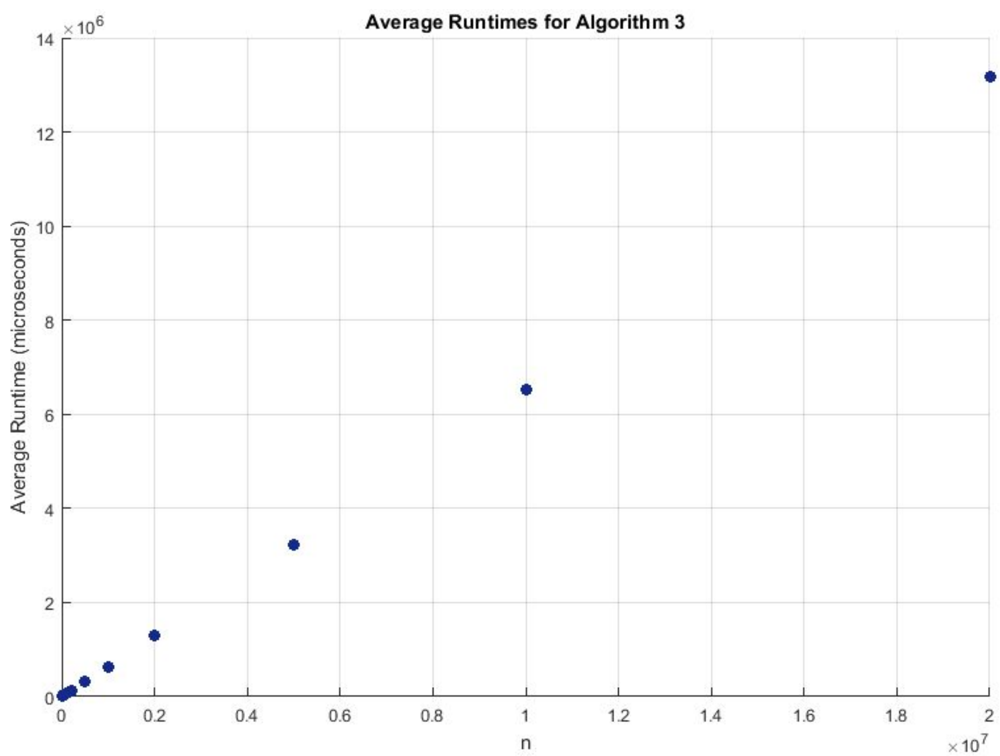
n	Algo4 Average Runtime (microseconds)
100	1
200	2
500	5
1000	10

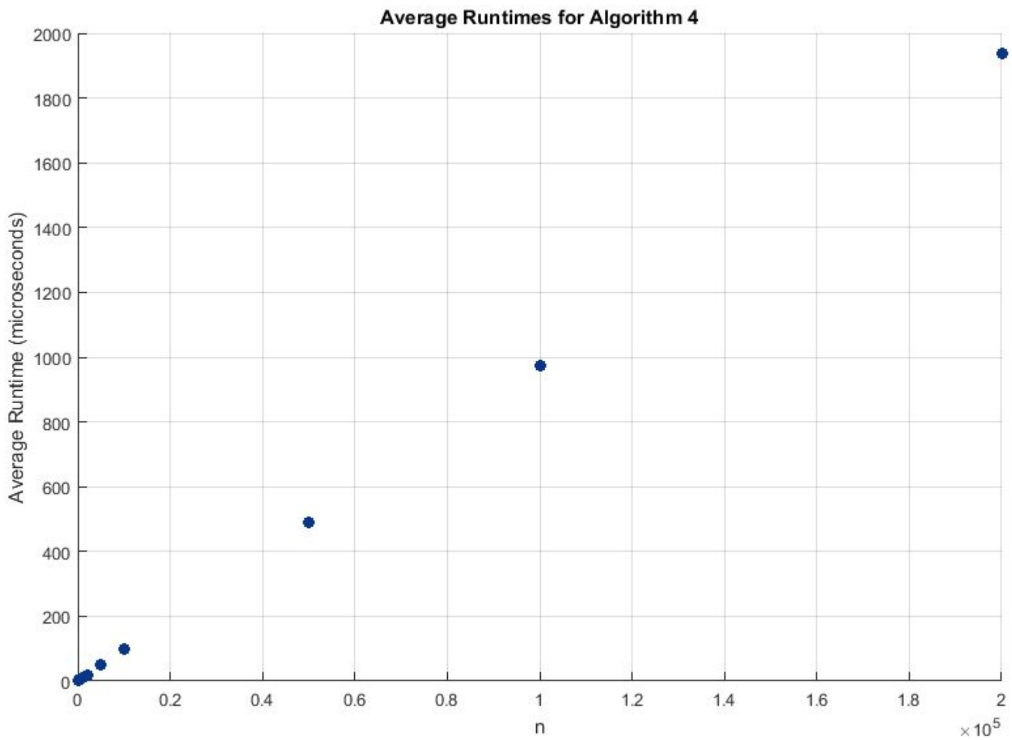
2000	20
5000	50
10000	98
50000	488
100000	973
200000	1936

2. Running time plots:




Average Runtimes for Algorithm 2
Average Runtime (microseconds)
n
×10^7
×10^4
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
1
2
3
4
5
6
7
8
9
10


Average Runtimes for Algorithm 3
Average Runtime (microseconds)
n
×10^6
×10^7
0
2
4
6
8
10
12
14
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2



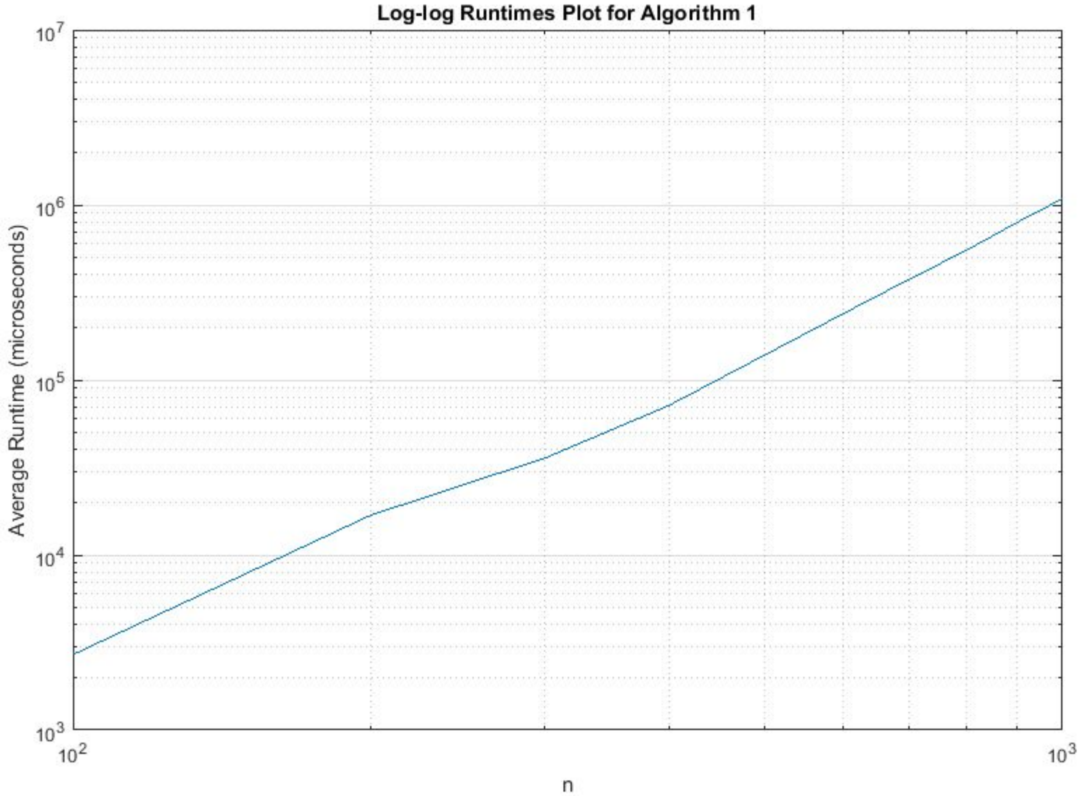
3. Regression models:
    - Algorithm 1:
        - $T_1(n) = 0.001152n^3 - 0.1136n^2 + 49.57n - 714.9$
        - $R^2 = 0.9999$
    - Algorithm 2:
        - $T_2(n) = 0.004639n^2 + 2.628n - 8624$
        - $R^2 = 1.0000$
    - Algorithm 3:
        - $T_3(n) = 0.02718n*lg(n) + 81910$
        - $R^2 = 0.9997$
    - Algorithm 4:
        - $T_4(n) = 0.009686n + 1.073$
        - $R^2 = 1.0000$
4. Discrepancies:
    - **Algorithm 1:** For algorithm 1 there were not any apparent discrepancies between expected runtime and our results. We knew that it would unable to efficiently run values of n larger than 10000 given its cubic runtime complexity. Our runtime test program would hang for several hours with values greater than 20000 making it impractical to test beyond n values of 1000. But for the values of n we did run the regression had a R^2 of .9999 indicating a good fit.
    - **Algorithm 2:** Algorithm 2 did not have any discrepancies between expected runtime and our results. We knew it would be roughly $O(n^2)$ for the complexity the results of the regression reflect this as well for R^2 is equal to 1.00.

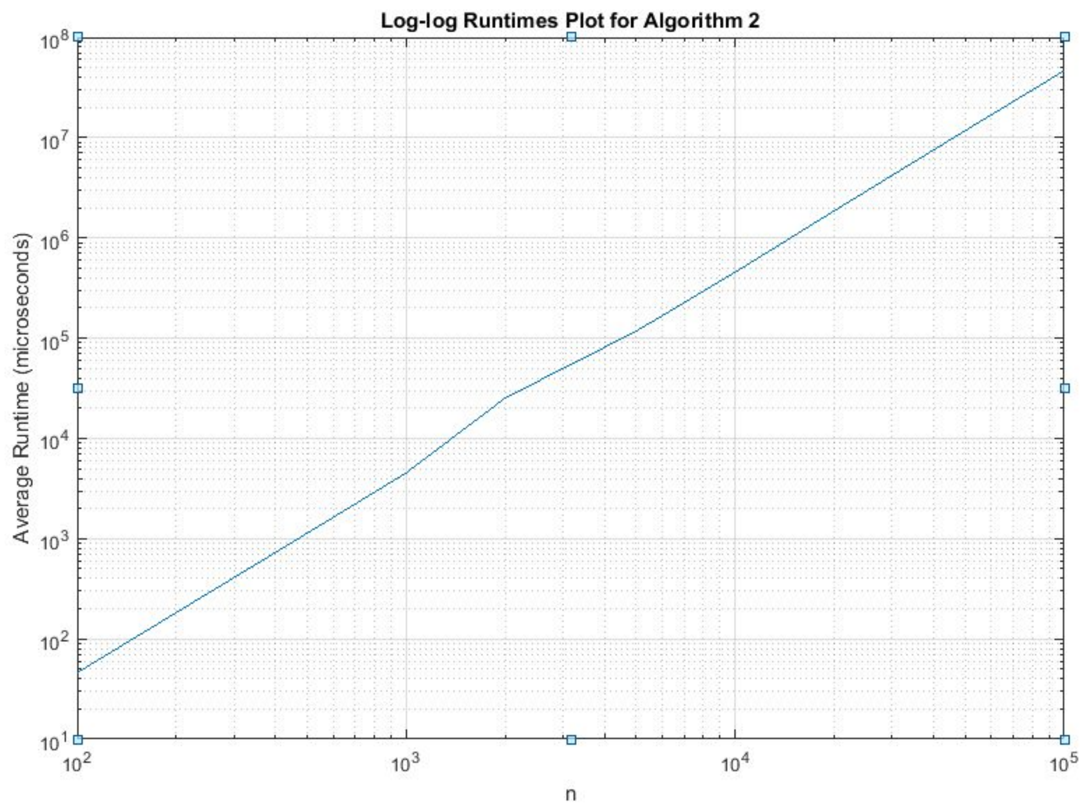
- **Algorithm 3:** Algorithm 3 did appear to have some mild discrepancies relative to the expected runtime, specifically it was expected to have a O(n lgn) runtime due to its divide and conquer approach for finding the maximum subarray, but had a runtime more closely resembling O(n) when inspecting the plot. The regression for Algorithm 3 using f(n) = a*n lgn + c as a model had an R^2 value of .9997 indicating that its runtime was in fact f(n) = O(n lgn), despite appearing as a straight line. This is okay because, Algorithm 3 is still bound by $O(n^2)$ from above and Ω(n) from below. The runtimes would likely begin to fit an n lgn curve even better at higher n, but it was not feasible to test them with any higher values of n than we already did.
- **Algorithm 4:** Algorithm 4 was expected to have a linear runtime and the regression analysis demonstrates that it does indeed have a linear time with the regression having an R^2 of 1.00.

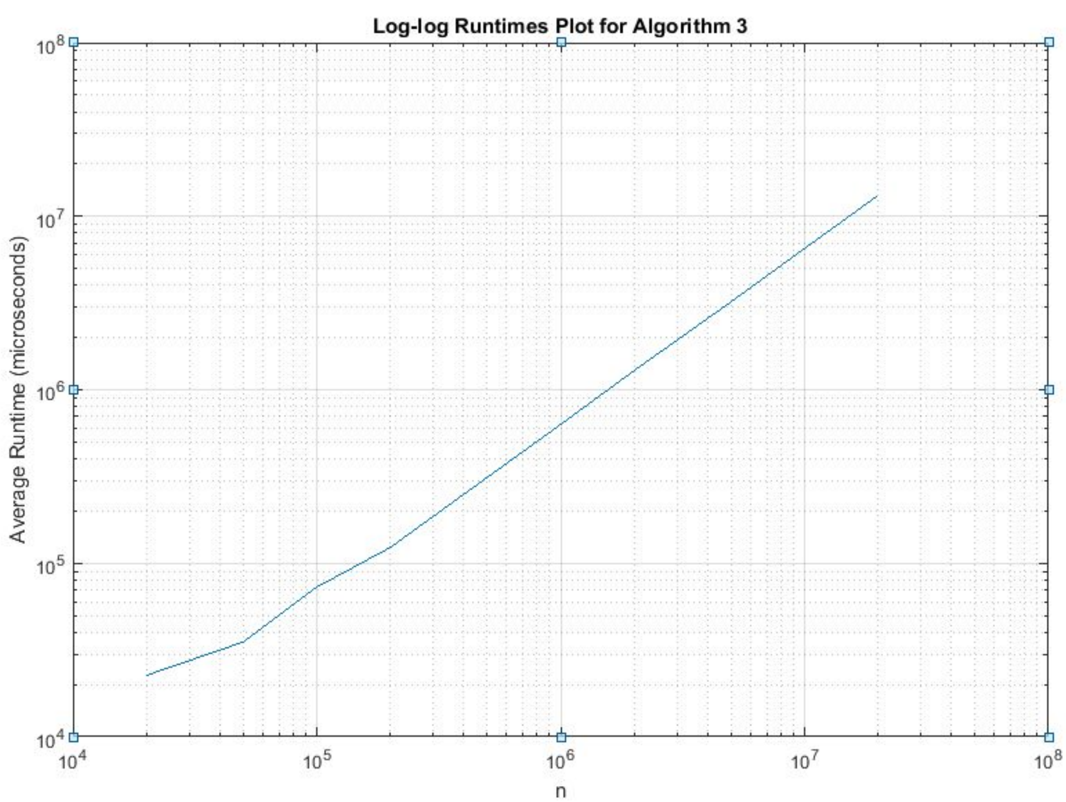
5. Largest input in given time:

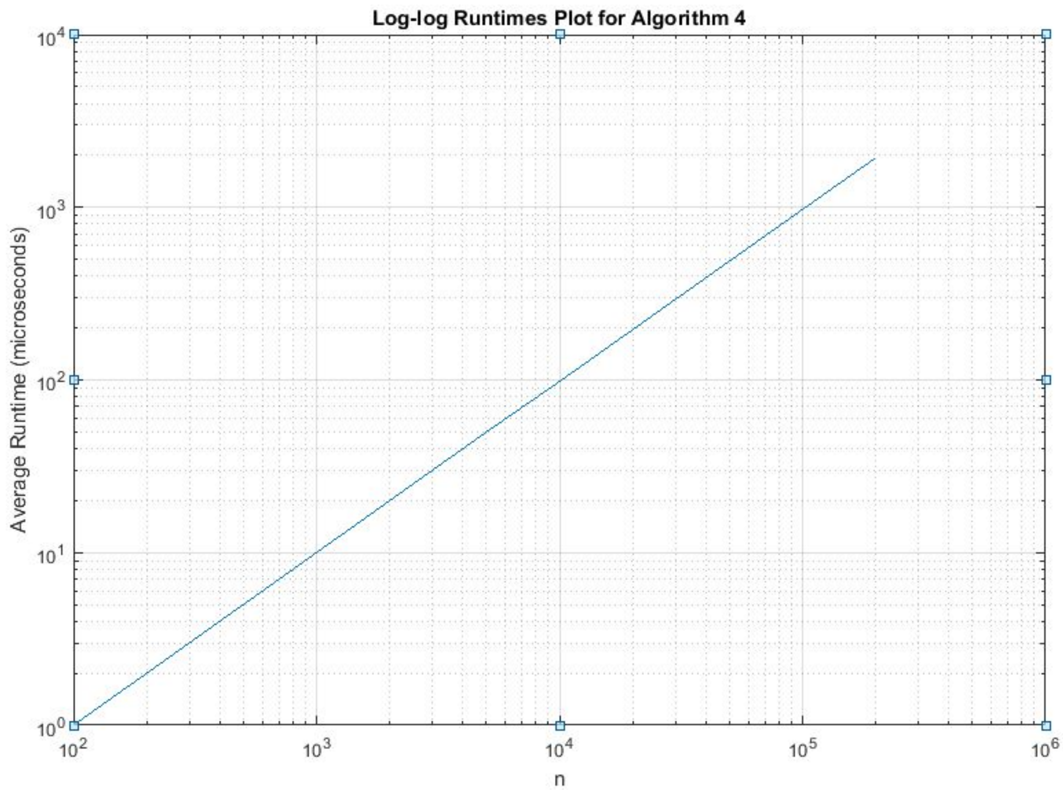
	Largest value of n possible for Algorithm...			
Time (microseconds)	**1**	**2**	**3**	**4**
5000000	1655	32570	7897091	516208850
10000000	2081	46160	15289665	1032417811
60000000	3763	113447	83758161	6194507425

6. Log-log plots:




Log-log Runtimes Plot for Algorithm 2
Average Runtime (microseconds)
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^2
10^3
10^4
10^5
n


Log-log Runtimes Plot for Algorithm 3
Average Runtime (microseconds)
10^4
10^5
10^6
10^7
10^8
10^4
10^5
10^6
10^7
10^8
n



7. Combined plot: