CS 325 Project 1

Group Members
Stephen McGuckin
Ryan Peterson
Brandon Swanson

Theoretical Runtime Analysis

<u>Algorithm 1</u>

Pseudo code:
```
maxsum = -INFINITY
for i from 0 to end of array index
        for J = i to the end of array index
                currentsum = 0
                for k = i while k <= J
                        current sum = currentsum + array[k]
                if currentsum > maxsum then maxsum = currentsum
```

The statement in the outermost loop is computed exactly N times (N = size of the array). The middle loop is also executed at most N times in each iteration of the outer loop. The inner loop is never executed more than n times. Multiplying these 3 together we get that this algorithm's cost is Θ(n^3).

<u>Algorithm 2</u>

Pseudo code:
```
maxsum = -INFINITY
for  i from 0 to end of array
        currentsum = 0
        for J = i till the end of array
                currentsum = currentsum + array[ J ]
        if currentsum > maxsum then maxsum = currentsum
```

The statement in the outer loop is executed exactly n times. And the inner loop is executed at most n times therefore this algorithm is Θ(n^2).

The inner loop will run n times on its first iteration then n-1, n-2, until it runs 1 time. the sum of these iterations is the sum of numbers 1 through n sequence which is equal to n(n+1)/2 = (n^2 + n) /2 which asymptotically is Θ(n^2)

Algorithm 3

Pseudo code:

```
maxmiddle(ARRAY[],low, mid, end) {

    leftsum = -INFINITY
    For i = mid downto low.
        sum=sum+ARRAY[i]
        if sum>leftsum
            lefsum = sum
            max-left = i

    rightsum = -INFINITY
    For i = mid + 1 upto high.
        sum=sum+ARRAY[i]
        if sum>rightsum
            rightsum = sum
            max-right = i

    return maxl-left,max-right, leftsum+rightsum

 }

maxsubarray3(ARRAY[],low,high) {
    if low = high we have one element return indexes low  mid high

    else
        mid = middle of array
        return the biggest value of the 3(
            1. max_middle(array[], low,mid,high) //max of anything in the middle
            2. maxsubarray3(array[],low,mid) // max from the beginning  to middle
            3. maxsubarray3(array[],mid+1,high) // max from the beginning +1 to end
        )
}
```

by examining the recursion tree we can evaluate that this algorithm does Θ((n) for each level of recursion. there are lg n levels of recursion. Therefore we get Θ(n lg n). We can further prove this by examining the recurrence and applying the master theorem.
Because each call to the algorithm makes a recursive calls on one half of the subarray each, and calls a function that sums all the values of the subarray received by the algorithm, The recurrence can be stated thusly:

$T(n) = 1 \quad for\ n = 1$
$T(n) = 2T(n/2) + n$

This fits the masters theorem format and
$f(n) = \Theta(n^{log_2 2} * log^0 n) = \Theta(n)$ therefore T(n) = Θ(nlgn)

<u>Algorithm 4</u>

Pseudo code:

for i from 0 to size of array

max = max + the value of element in array at index i

if current max <= 0

current max = 0

keep track of greatest max seen so far

This loop only goes through array exactly n times. This is a linear time algorithm and cost $\Theta(n)$.

Proof of Correctness

Problem: Find the subarray of array a that has the greatest sum

Input: a, a start and end point

Output: the start point, end point and sum of the max subarray

Proving max

<u>Precondition:</u> a and b have valid max values

<u>Postcondition:</u> return value is whichever of a and b has the highest max value

<u>Proof:</u>

1. a.max > b.max is true if a has a greater max value than b
2. if (a.max > b.max) return a returns a if a has a greater max value than b
3. else return b returns be if b has a greater max value than a

Proving threemax

<u>Precondition:</u> a, b and c all have valid max values

<u>Postcondition:</u> return value is whichever maxij has the greatest max value

<u>Proof:</u>

1. max(a,b) returns a or b with greatest max
2. max(max(a,b),c) returns greatest between c and max(a,b);
3. return max(max(a,b),c) returns greatest between a,b and c

Proving max_middle

<u>Precondition:</u> a is a valid array, start = x, mid = y, end = z

<u>Postcondition:</u> result represents the subarray with the largest sum that crosses the midpoint

<u>Proof:</u>

Proving for (i = mid-1; i >= start; i--)

<u>Input:</u> an array a, int sum, int left_sum, int start, maxij result

<u>Output:</u> result.i = low bound for the subarray with the greatest sum with mid as it's upper
bound and left_sum = sum of subarray bound by result.i and mid

Loop Invariant: At step k at the start of each loop sum equals the sum of the values of a from
mid to mid-k, left_sum equals the greatest sum found between i=mid and i=mid-k and result.i = the i corresponding to left_sum
Initialization:
k = 0, sum = a[mid], left_sum = a[mid], result.i = 0. Everything is still where it was at initialization
Maintenance:
i = mid-(k+1), sum = sum(from k) + a[i], left_sum = left_sum(from k) unless sum is greater
then left_sum =sum, the same goes for result.i - if sum is less than left_sum(from k) then
result.i = result.i(from k) otherwise result.i = i
Termination:
The loop terminates when mid-k <= start

Proving for (i = mid+1; i <= end; i++)

Input: an array a, int sum, int left_sum, int end, maxij result
Output: result.j = high bound for the subarray with the greatest sum with mid as it's lower
bound and left_sum = sum of subarray bound by result.j and mid
Loop Invariant: At step k at the start of each loop sum equals the sum of the values of a from
mid+1 to mid+k when k > 0, right_sum equals the greatest sum found between i=mid and
i=mid-k and result.j = the i coresponding to right_sum
Initialization:
k = 0, sum = 0, right_sum = 0, result.j = 0.
Maintenance:
i = mid+1+(k+1), sum = sum(from k) + a[i], right_sum = right_sum(from k) unless sum is
greater then right_sum =sum, the same goes for i - if sum is less than right_sum(from k) then
result.j = result.j(from k) otherwise result.j = j
Termination:
The loop terminates when mid+1+k >= end

The two loops (proved above) set result.j and result.i and set left_sum and right_sum
The final step sets result.max to left_sum + right_sum.
left_sum = greatest sum begin to mid with mid as upper bound
right_sum = greatest sum mid to end with mid as lower bound
result.max = greatest sum including mid

Proving r_maxsubarray_3

Input: array a, int low and int high

Output: a maxij representing the subarray that sums to the greatest amount
Base Case: If low = high then the subarray is of size one and a maxij is returned in the form {low,high,a[low]}
Inductive Hypothesis: r_maxsubarray_3 finds the maximum subarray of an array (or subarray) where (high - low) = k
Inductive Step: (high - low) = k + 1
First recursive call takes the array a, int low and the calculated middle (as int high) as input and outputs max subarray of low+1 to mid
Second recursive call takes the array a, the calculated middle plus one (as int low) and int high as input and outputs max subarray of mid+1 to high
max_middle takes array a, int low, the calculated middle as mid and int high as inputs and outputs max subarray including mid
threemax outputs max subarray of the three
Greatest possible subarray is returned
Termination:
Every recursive call halves (high - low) and the recursion terminates at (high - low = 0)

Testing

We tested the algorithms by creating a compiled program that would output the result of each algorithm to a different file each. reading from the MSS_Problems.txt file provided and outputting in the same format as the provided MSS_Results.txt file. Each output could then be automatically compared to the results file using diff. Algorithm 3 sometimes found a different optimal solution but on these small samples could still be verified to be a correct solution. After testing was concluded on these provided tests the algorithms were then tested on larger randomly generated values and then their outputs were compared with each other.

Experimental Analysis

Data for 100 values of n for algorithms 2,3,and 4 and 37 values of n for algorithm 1 are attached as well as plots and fit lines.

The only algorithm with any significant discrepancy from the expected trend line was the 4th algorithm, the linear time algorithm. We believe this was caused by the two conditionals within this algorithm's loop used for tracking the best sum discovered so far and the beginning and end points of that subarray. Each of these conditionals involve variable assignments but are not always triggered on each iteration of the loop. Therefore different arrays with different distributions of positive and negative numbers would cause more variable assignments therefore increasing the time the algorithm takes to finish. This accounts for the distribution of the measured time across both sides of the linear trendline.

Regression model and 10 minute size of N
(functions generated and solved using Matlab and Wolfram Alpha)

Algorithm 1:
Regression Function:

(1351074808815051*n^3)/2417851639229258349412352 - (4764799537088881*n^2)/18889465931478580854784 + (5047297659782551*n)/18446744073709551616 - 1089898065277315/18014398509481984

Largest value of N under 10 minutes:
10377

Algorithm 2:
Regression Function:
(7022048938385443*n^2)/4835703278458516698824704 - (1995149045137059*n)/75557863725914323419136 + 8321871326131415/295147905179352825856

Largest value of N under 10 minutes:
642805

Algorithm 3:
Regression Function:
(1355390088139563*n^2)/618970019642690137449562112 + (1535378422870539*n)/18889465931478580854784 + 2841859988135817/73786976294838206464

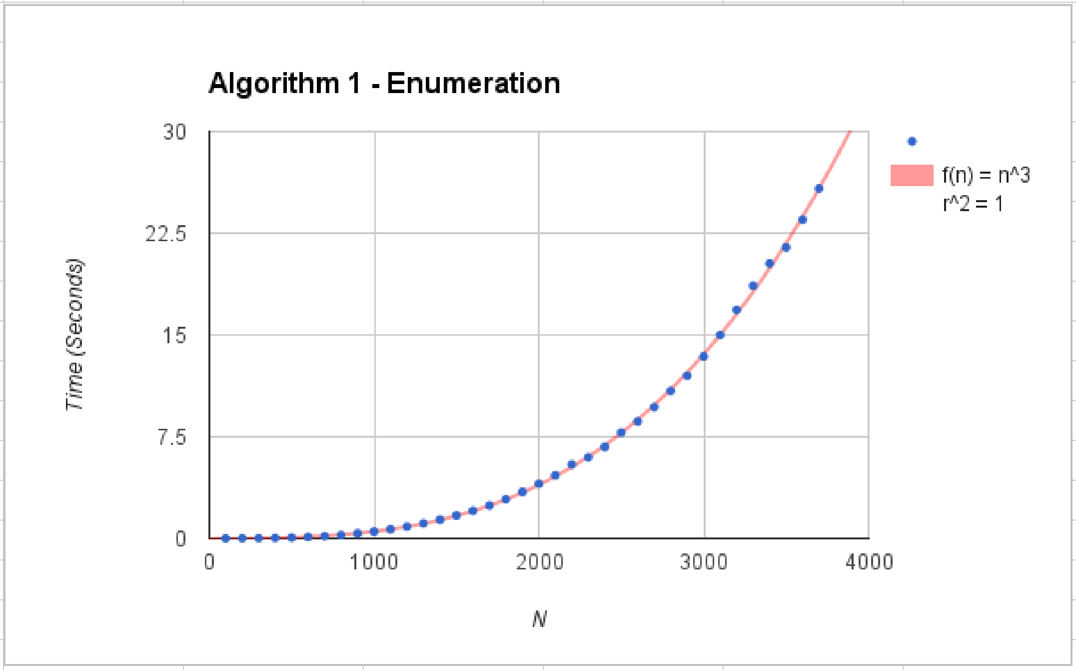
Largest value of N under 10 minutes:
16534510

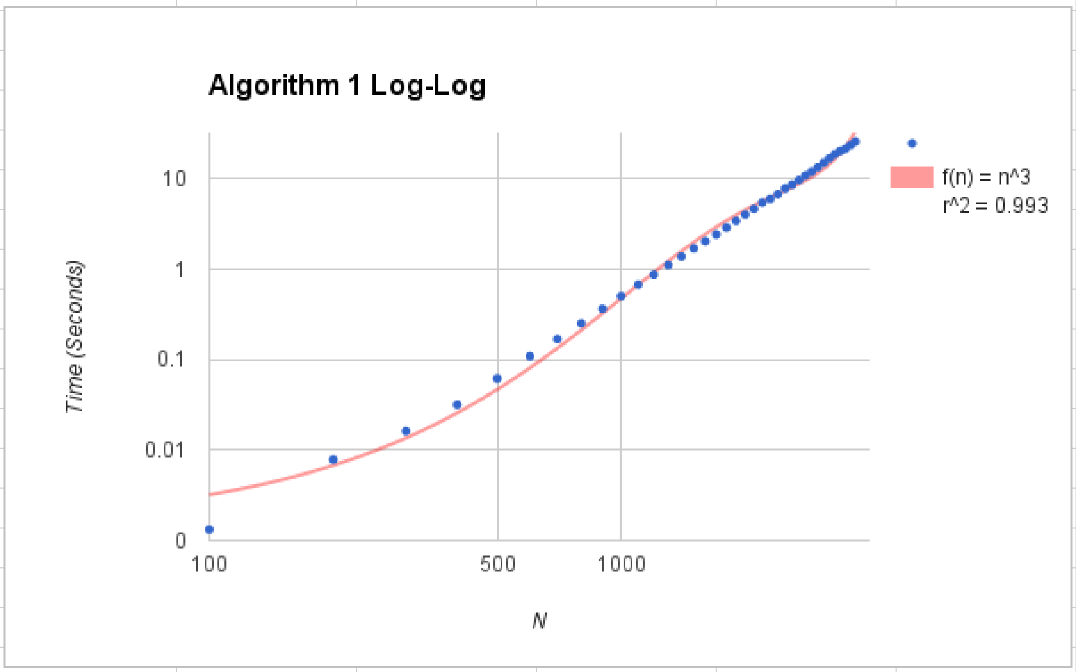
Algorithm 4:
Regression Function:
(2154593805482993*n)/604462909807314587353088 + 4781038308868933/590295810358705651712

Largest value of N under 10 minutes:
168327663463

Algorithm 1

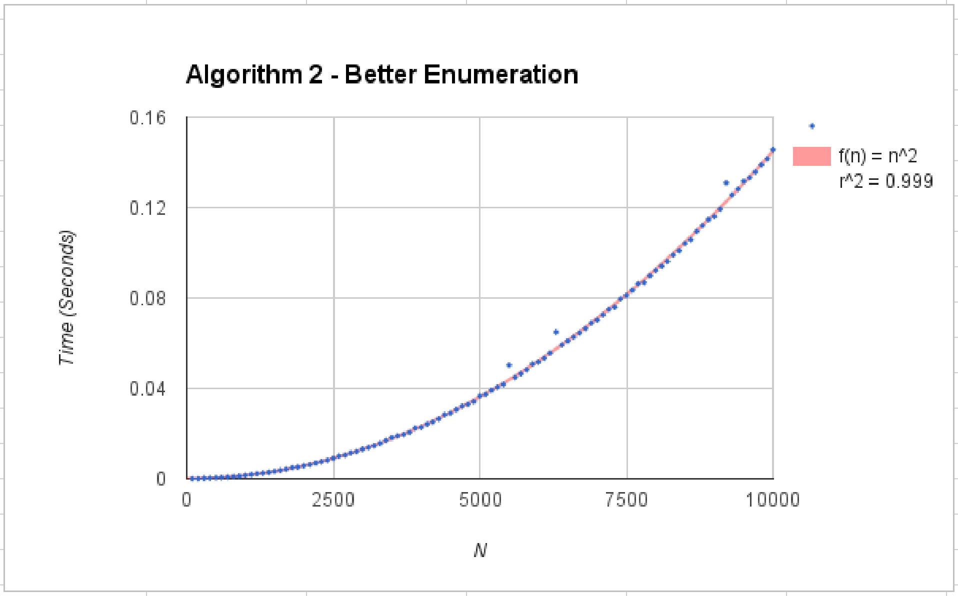
n	Time
100	0.001322
200	0.00784
300	0.016186
400	0.031726
500	0.061655
600	0.109024
700	0.168709
800	0.251702
900	0.364345
1000	0.503787
1100	0.673981
1200	0.870073
1300	1.11374
1400	1.37986
1500	1.698796
1600	2.037167
1700	2.42587
1800	2.888099
1900	3.426554
2000	4.026957
2100	4.653129
2200	5.455892
2300	5.984718
2400	6.748713
2500	7.814335
2600	8.62929
2700	9.678947
2800	10.8769
2900	11.995591
3000	13.414695
3100	15.001214
3200	16.848036
3300	18.631853
3400	20.277542
3500	21.459803
3600	23.500699
3700	25.795256

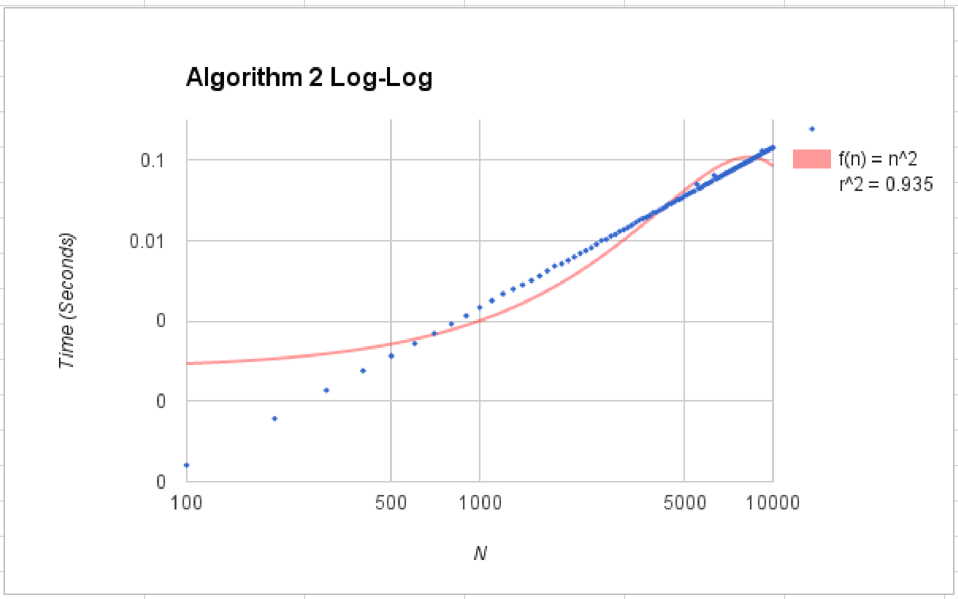




Algorithm 2

n	Time
100	0.000016
200	0.000061
300	0.000137
400	0.000241
500	0.000369
600	0.000525
700	0.000699
800	0.000918
900	0.001162
1000	0.001467
1100	0.001792
1200	0.002175
1300	0.002496
1400	0.002815
1500	0.003201
1600	0.003653
1700	0.004206
1800	0.004867
1900	0.00518
2000	0.005707
2100	0.006279
2200	0.006923
2300	0.007517
2400	0.008176
2500	0.008987
2600	0.010029
2700	0.010423
2800	0.011406
2900	0.011975
3000	0.013043
3100	0.013722
3200	0.014571
3300	0.015549
3400	0.01691
3500	0.018118
3600	0.018939
3700	0.019574
3800	0.020545
3900	0.022368
4000	0.022736
4100	0.024103
4200	0.025111
4300	0.026493
4400	0.028351
4500	0.029021
4600	0.03058
4700	0.032169
4800	0.032888
4900	0.034131
5000	0.036507



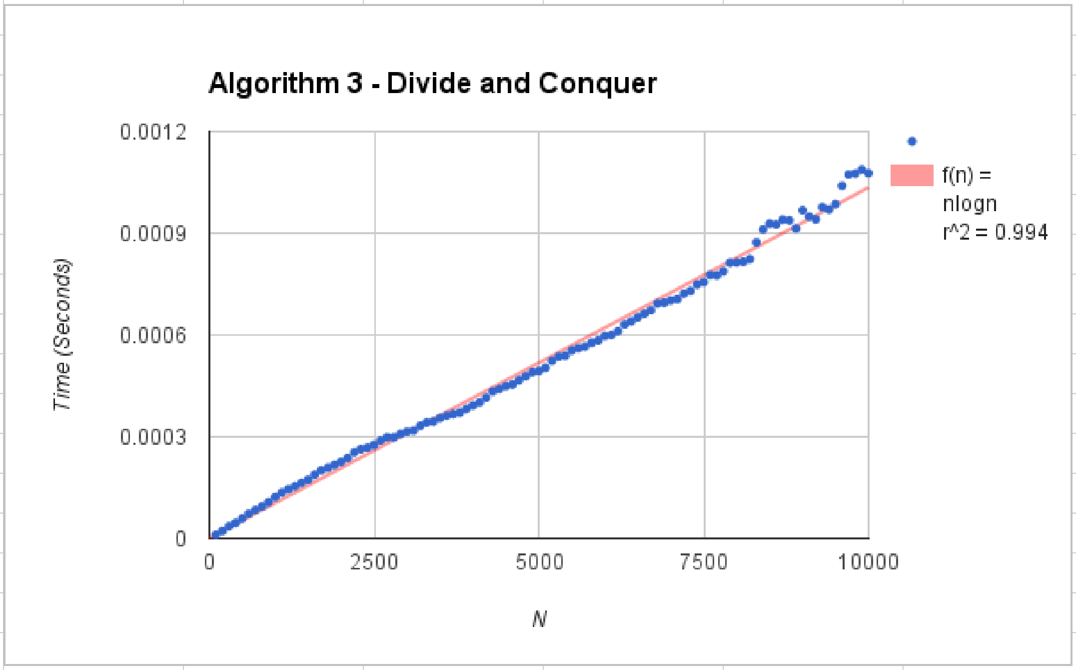


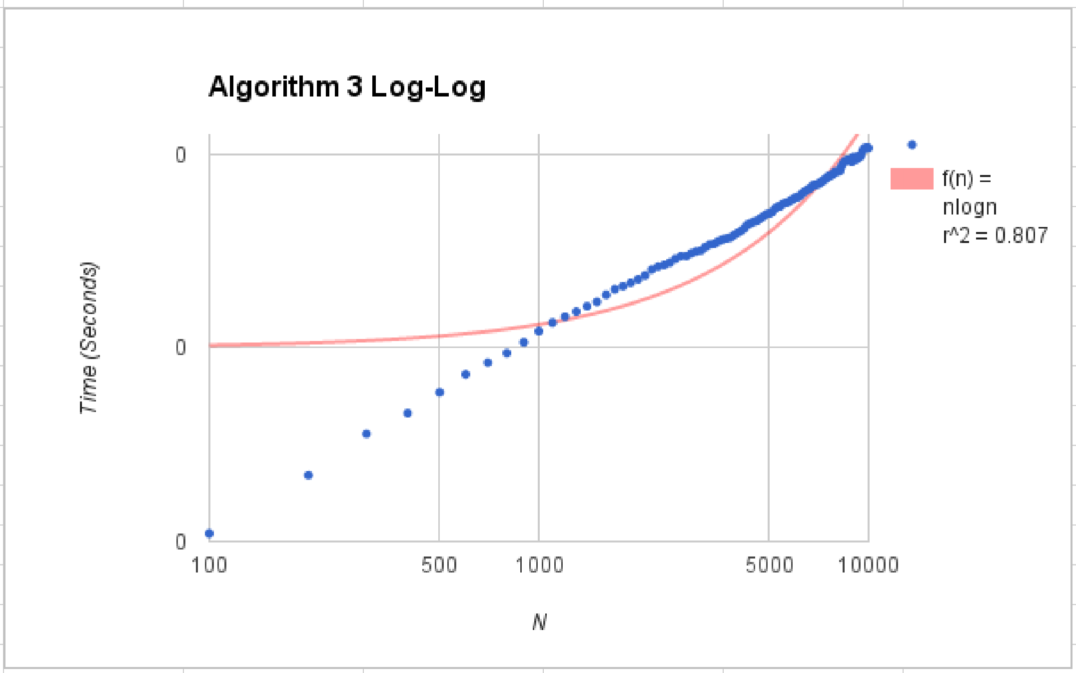
Algorithm 2

n	Time
5100	0.037388
5200	0.039064
5300	0.040574
5400	0.04172
5500	0.050303
5600	0.044862
5700	0.046475
5800	0.048253
5900	0.05079
6000	0.05158
6100	0.053306
6200	0.055605
6300	0.06487
6400	0.059217
6500	0.06094
6600	0.062716
6700	0.06451
6800	0.066451
6900	0.068915
7000	0.070261
7100	0.072556
7200	0.074973
7300	0.075976
7400	0.079525
7500	0.081063
7600	0.083387
7700	0.086383
7800	0.086857
7900	0.089833
8000	0.092167
8100	0.094074
8200	0.096172
8300	0.099018
8400	0.100985
8500	0.104037
8600	0.105814
8700	0.109433
8800	0.112031
8900	0.114746
9000	0.116114
9100	0.119281
9200	0.13096
9300	0.125549
9400	0.128259
9500	0.131652
9600	0.133291
9700	0.13573
9800	0.13899
9900	0.141595
10000	0.14567

Algorithm 3

n	Time
100	0.000011
200	0.000022
300	0.000036
400	0.000046
500	0.000059
600	0.000073
700	0.000084
800	0.000094
900	0.000107
1000	0.000122
1100	0.000135
1200	0.000145
1300	0.000154
1400	0.000164
1500	0.000173
1600	0.000188
1700	0.000201
1800	0.000208
1900	0.000217
2000	0.000226
2100	0.000237
2200	0.000254
2300	0.000263
2400	0.000268
2500	0.000276
2600	0.000289
2700	0.000298
2800	0.000298
2900	0.000308
3000	0.000315
3100	0.000318
3200	0.000332
3300	0.000342
3400	0.000345
3500	0.000355
3600	0.000362
3700	0.000367
3800	0.000371
3900	0.000382
4000	0.000392
4100	0.000401
4200	0.000415
4300	0.000434
4400	0.000441





Algorithm 3

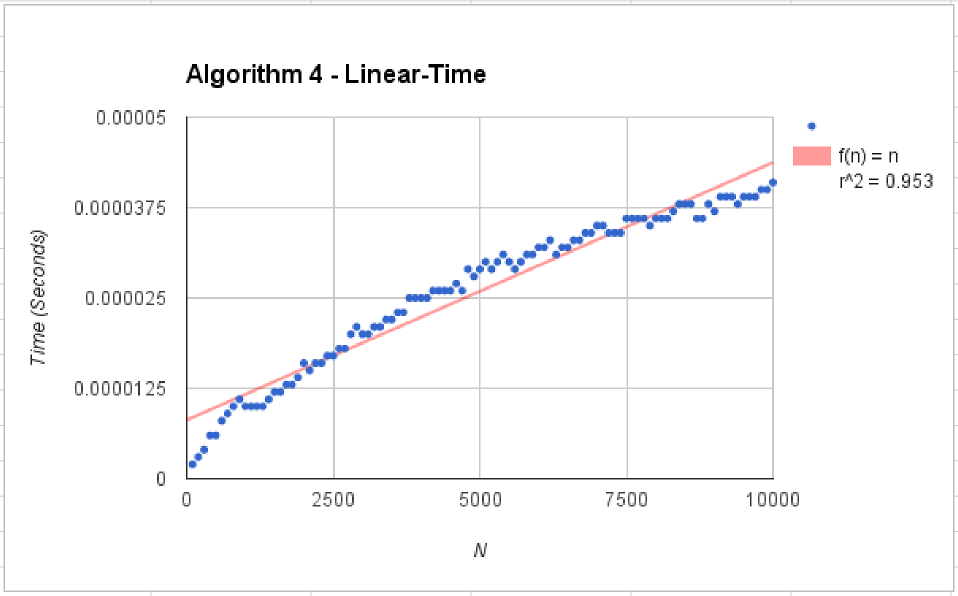
n	Time						
4500	0.000449						
4600	0.000454						
4700	0.000467						
4800	0.000478						
4900	0.00049						
5000	0.000494						
5100	0.000503						
5200	0.000524						
5300	0.000536						
5400	0.000539						
5500	0.000555						
5600	0.000561						
5700	0.000566						
5800	0.000577						
5900	0.000584						
6000	0.000597						
6100	0.0006						
6200	0.000611						
6300	0.000631						
6400	0.00064						
6500	0.000652						
6600	0.000663						
6700	0.000673						
6800	0.000693						
6900	0.000695						
7000	0.000702						
7100	0.000706						
7200	0.000722						
7300	0.00073						
7400	0.000749						
7500	0.000756						
7600	0.000777						
7700	0.000776						
7800	0.000788						
7900	0.000813						
8000	0.000814						
8100	0.000816						
8200	0.000824						
8300	0.000873						
8400	0.000911						
8500	0.000928						
8600	0.000926						
8700	0.00094						
8800	0.000938						

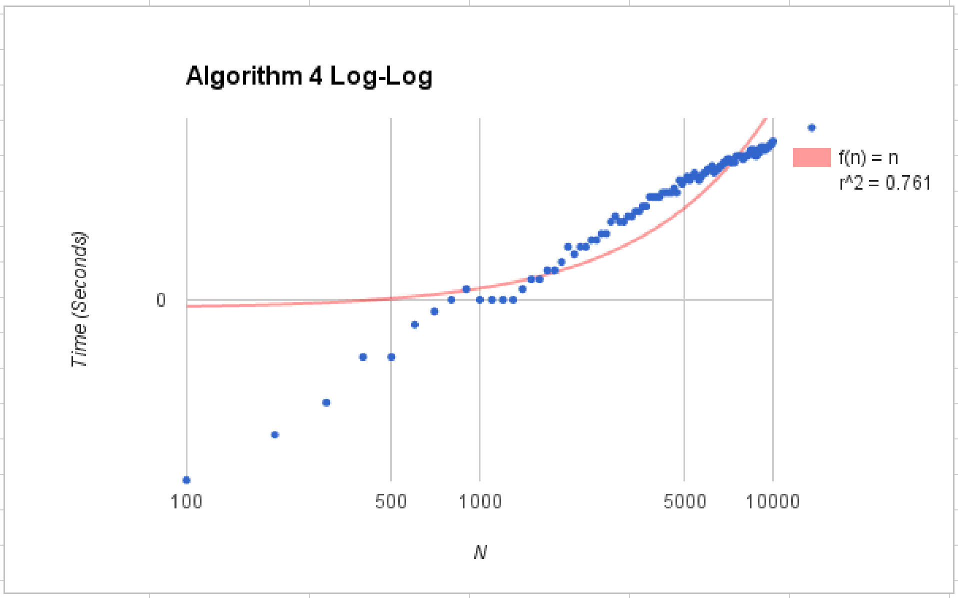
Algorithm 3

n	Time
8900	0.000914
9000	0.000968
9100	0.000949
9200	0.000941
9300	0.000977
9400	0.00097
9500	0.000986
9600	0.00104
9700	0.001073
9800	0.001076
9900	0.001088
10000	0.001077

Algorithm 4

n	Time
100	0.000002
200	0.000003
300	0.000004
400	0.000006
500	0.000006
600	0.000008
700	0.000009
800	0.00001
900	0.000011
1000	0.00001
1100	0.00001
1200	0.00001
1300	0.00001
1400	0.000011
1500	0.000012
1600	0.000012
1700	0.000013
1800	0.000013
1900	0.000014
2000	0.000016
2100	0.000015
2200	0.000016
2300	0.000016
2400	0.000017
2500	0.000017
2600	0.000018
2700	0.000018
2800	0.00002
2900	0.000021
3000	0.00002
3100	0.00002
3200	0.000021
3300	0.000021
3400	0.000022
3500	0.000022
3600	0.000023
3700	0.000023
3800	0.000025
3900	0.000025
4000	0.000025
4100	0.000025
4200	0.000026
4300	0.000026
4400	0.000026
4500	0.000026
4600	0.000027
4700	0.000026
4800	0.000029
4900	0.000028
5000	0.000029



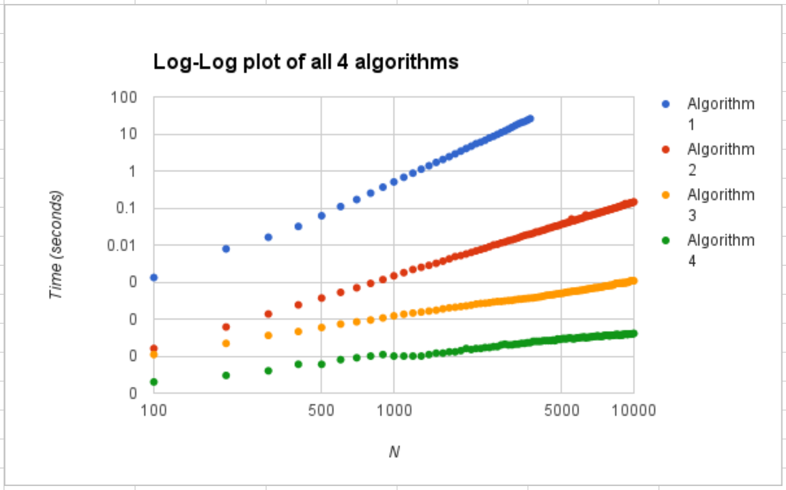


Algorithm 4

n	Time
5100	0.00003
5200	0.000029
5300	0.00003
5400	0.000031
5500	0.00003
5600	0.000029
5700	0.00003
5800	0.000031
5900	0.000031
6000	0.000032
6100	0.000032
6200	0.000033
6300	0.000031
6400	0.000032
6500	0.000032
6600	0.000033
6700	0.000033
6800	0.000034
6900	0.000034
7000	0.000035
7100	0.000035
7200	0.000034
7300	0.000034
7400	0.000034
7500	0.000036
7600	0.000036
7700	0.000036
7800	0.000036
7900	0.000035
8000	0.000036
8100	0.000036
8200	0.000036
8300	0.000037
8400	0.000038
8500	0.000038
8600	0.000038
8700	0.000036
8800	0.000036
8900	0.000038
9000	0.000037
9100	0.000039
9200	0.000039
9300	0.000039
9400	0.000038
9500	0.000039
9600	0.000039
9700	0.000039
9800	0.00004
9900	0.00004
10000	0.000041

All Algorithms

N	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
100	0.001322	0.000016	0.000011	0.000002
200	0.00784	0.000061	0.000022	0.000003
300	0.016186	0.000137	0.000036	0.000004
400	0.031726	0.000241	0.000046	0.000006
500	0.061655	0.000369	0.000059	0.000006
600	0.109024	0.000525	0.000073	0.000008
700	0.168709	0.000699	0.000084	0.000009
800	0.251702	0.000918	0.000094	0.00001
900	0.364345	0.001162	0.000107	0.000011
1000	0.503787	0.001467	0.000122	0.00001
1100	0.673981	0.001792	0.000135	0.00001
1200	0.870073	0.002175	0.000145	0.00001
1300	1.11374	0.002496	0.000154	0.00001
1400	1.37986	0.002815	0.000164	0.000011
1500	1.698796	0.003201	0.000173	0.000012
1600	2.037167	0.003653	0.000188	0.000012
1700	2.42587	0.004206	0.000201	0.000013
1800	2.888099	0.004867	0.000208	0.000013
1900	3.426554	0.00518	0.000217	0.000014
2000	4.026957	0.005707	0.000226	0.000016
2100	4.653129	0.006279	0.000237	0.000015
2200	5.455892	0.006923	0.000254	0.000016
2300	5.984718	0.007517	0.000263	0.000016
2400	6.748713	0.008176	0.000268	0.000017
2500	7.814335	0.008987	0.000276	0.000017
2600	8.62929	0.010029	0.000289	0.000018
2700	9.678947	0.010423	0.000298	0.000018
2800	10.8769	0.011406	0.000298	0.00002
2900	11.995591	0.011975	0.000308	0.000021
3000	13.414695	0.013043	0.000315	0.00002
3100	15.001214	0.013722	0.000318	0.00002
3200	16.848036	0.014571	0.000332	0.000021
3300	18.631853	0.015549	0.000342	0.000021
3400	20.277542	0.01691	0.000345	0.000022
3500	21.459803	0.018118	0.000355	0.000022
3600	23.500699	0.018939	0.000362	0.000023
3700	25.795256	0.019574	0.000367	0.000023
3800		0.020545	0.000371	0.000025
3900		0.022368	0.000382	0.000025
4000		0.022736	0.000392	0.000025
4100		0.024103	0.000401	0.000025
4200		0.025111	0.000415	0.000026
4300		0.026493	0.000434	0.000026
4400		0.028351	0.000441	0.000026
4500		0.029021	0.000449	0.000026
4600		0.03058	0.000454	0.000027
4700		0.032169	0.000467	0.000026
4800		0.032888	0.000478	0.000029
4900		0.034131	0.00049	0.000028
5000		0.036507	0.000494	0.000029
5100		0.037388	0.000503	0.00003
5200		0.039064	0.000524	0.000029
5300		0.040574	0.000536	0.00003
5400		0.04172	0.000539	0.000031
5500		0.050303	0.000555	0.00003
5600		0.044862	0.000561	0.000029
5700		0.046475	0.000566	0.00003
5800		0.048253	0.000577	0.000031
5900		0.05079	0.000584	0.000031
6000		0.05158	0.000597	0.000032
6100		0.053306	0.0006	0.000032
6200		0.055605	0.000611	0.000033



All Algorithms

N	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
6300		0.06487	0.000631	0.000031
6400		0.059217	0.00064	0.000032
6500		0.06094	0.000652	0.000032
6600		0.062716	0.000663	0.000033
6700		0.06451	0.000673	0.000033
6800		0.066451	0.000693	0.000034
6900		0.068915	0.000695	0.000034
7000		0.070261	0.000702	0.000035
7100		0.072556	0.000706	0.000035
7200		0.074973	0.000722	0.000034
7300		0.075976	0.00073	0.000034
7400		0.079525	0.000749	0.000034
7500		0.081063	0.000756	0.000036
7600		0.083387	0.000777	0.000036
7700		0.086383	0.000776	0.000036
7800		0.086857	0.000788	0.000036
7900		0.089833	0.000813	0.000035
8000		0.092167	0.000814	0.000036
8100		0.094074	0.000816	0.000036
8200		0.096172	0.000824	0.000036
8300		0.099018	0.000873	0.000037
8400		0.100985	0.000911	0.000038
8500		0.104037	0.000928	0.000038
8600		0.105814	0.000926	0.000038
8700		0.109433	0.00094	0.000036
8800		0.112031	0.000938	0.000036
8900		0.114746	0.000914	0.000038
9000		0.116114	0.000968	0.000037
9100		0.119281	0.000949	0.000039
9200		0.13096	0.000941	0.000039
9300		0.125549	0.000977	0.000039
9400		0.128259	0.00097	0.000038
9500		0.131652	0.000986	0.000039
9600		0.133291	0.00104	0.000039
9700		0.13573	0.001073	0.000039
9800		0.13899	0.001076	0.00004
9900		0.141595	0.001088	0.00004
10000		0.14567	0.001077	0.000041