CS 317 – Fall 2017 - Lab 1

Drake Song

$T(n)_{avg}$ value, over 5 runs, for each algorithm, over each of several values n:

n	Prime1 (ms)	Prime2 (ms)	Prime3 (ms)	Prime4 (ms)
5	0.6092071545	0.0000000000	0.0000000000	0.0000000000
50	0.9934902190	0.5983829498	0.2001285553	0.0000000000
100	1.5182495117	0.4978458769	0.6981849671	0.0000000000
500	9.4008922577	3.6235332487	2.4020195007	0.5004405980
1000	35.9405994414	7.5044631958	4.0063857172	1.6018390656
5000	828.4489154816	33.7968826294	22.7406978607	5.9134483337
10000	3301.1036918849	76.3344287872	44.5463180537	12.8210067749
25000	20584.5925807953	268.3068275452	121.4374542236	40.2475833893
50000	82046.2965488434	652.2414684296	250.0383377075	88.9007085522
100000	327611.0027313230	1614.6483898163	529.9304960767	208.8375568390

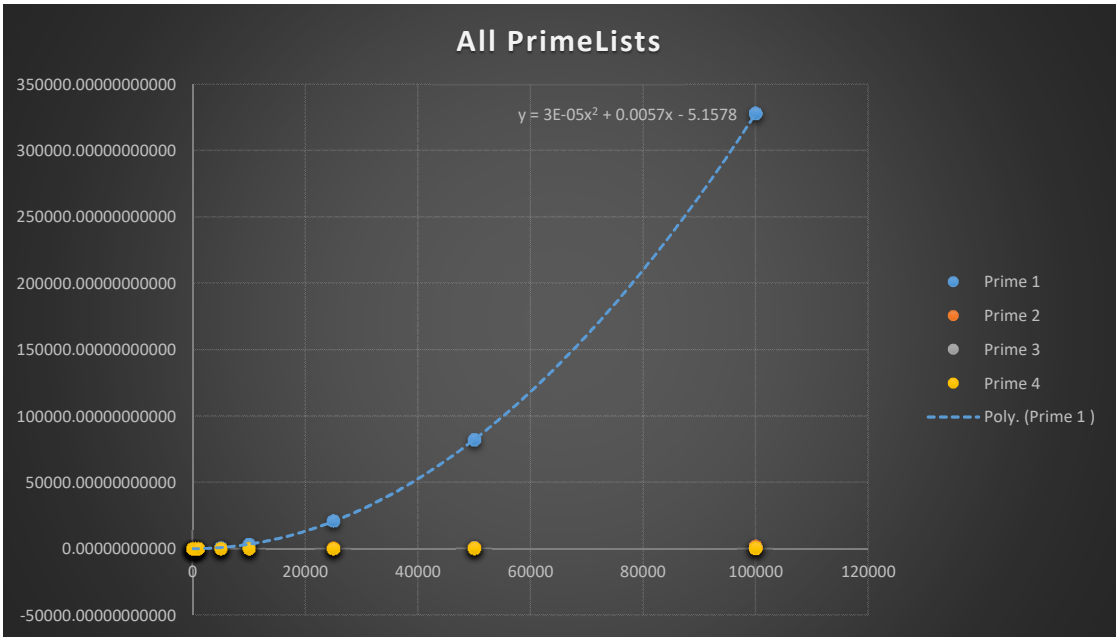
Summarize your results in the following below…

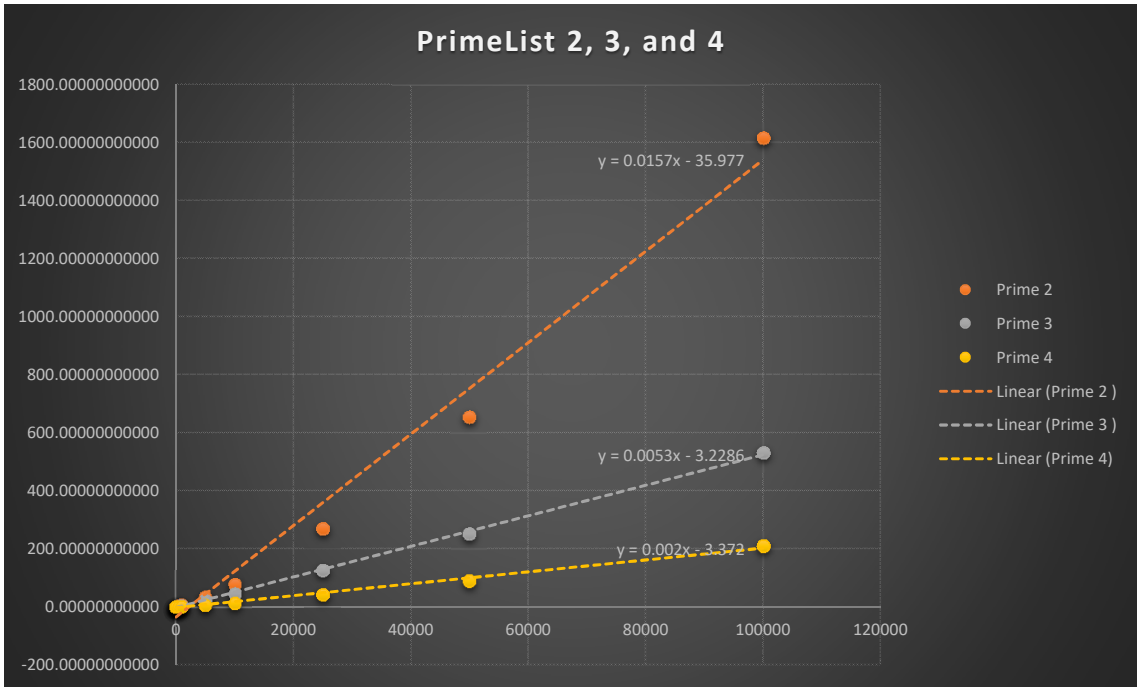
Algorithm	Slope of log-log	Polynomial T(n)
*Prime*1	1.9549 ≈ 2	$0.000032x^2 + 0.0057x - 5.1577$
*Prime*2	1.0734 ≈ 1	0.0157x – 35.9772
*Prime*3	0.9659 ≈ 1	0.0053x – 3.2286
*Prime*4	0.9426 ≈ 1	0.002x – 3.372

Finally, discuss your results in approximately one paragraph:

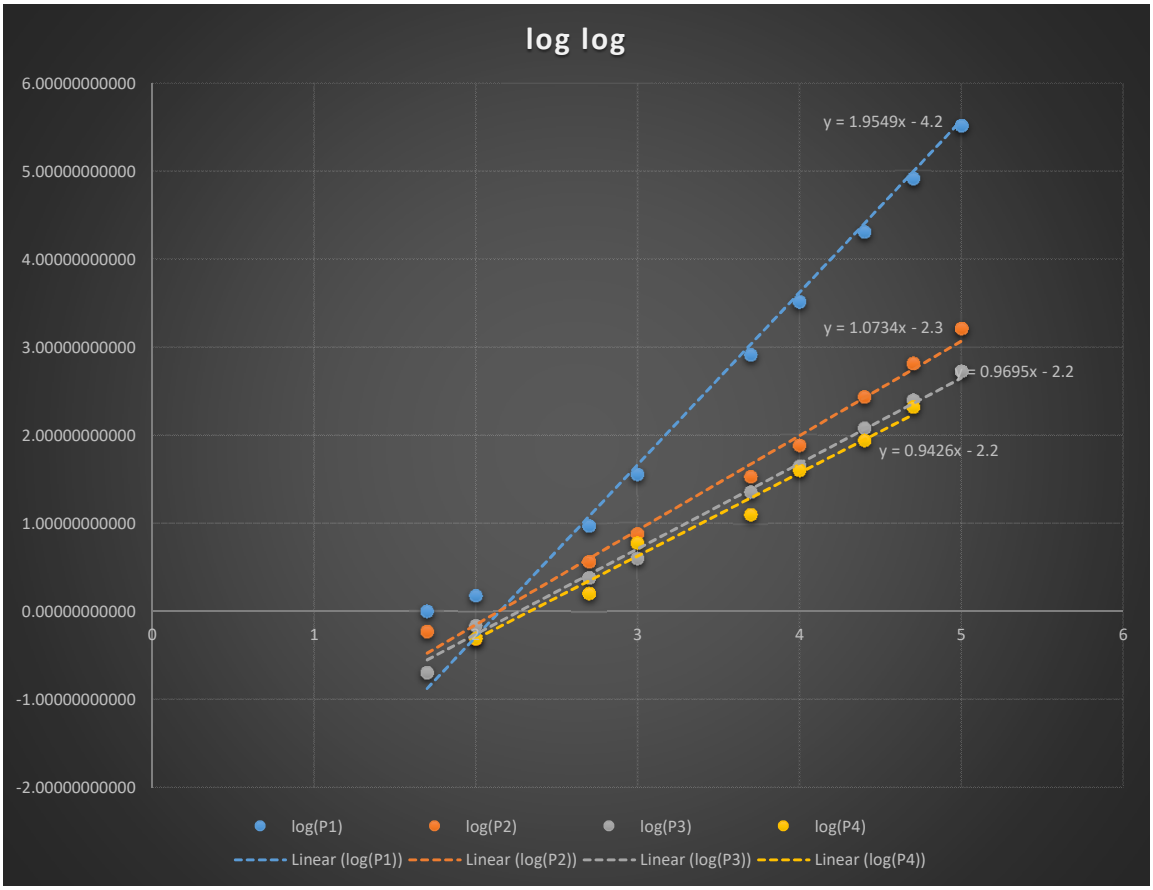
Using Python, I've collected the data. Utilizing Excel, I've graphed the average of the points and figured out the approximate slopes of the log-log functions (had to get rid of some points to make the trend line fit better). Using a Linear System Solver online, I've put into the necessary data and found the approximate Polynomial T(n) for each Primes. Comparing the T(n) with the trend line function on Excel graph, it looks like they're almost identical. Only Prime1 has a quadratic equation while others have linear equations. This is primarily due to the fact that Prime1 has run time of $O(n^2)$ since for every number (say x), it has to go through all the numbers up to x in order to determine if it's prime or not. Others however, follow a linear equation due to shortened run time by sqrt(n) and the use of break. Prim4 is the fastest because it utilizes List (ArrayList in Java).

n	Prime 1	Prime 2	Prime 3	Prime 4
5	0.60920715447	0.00000000000	0.00000000000	0.00000000000
50	0.99349021902	0.59838294984	0.20012855530	0.00000000000
100	1.51824951172	0.49784587688	0.69818496706	0.00000000000
500	9.40089225768	3.62353324866	2.40201950074	0.50044059796
1000	35.94059944138	7.50446319584	4.00638571725	1.60183906556
5000	828.44891548158	33.79688262938	22.74069786068	5.91344833374
10000	3301.10369188490	76.33442878724	44.54631805366	12.82100677492
25000	20584.59258079530	268.30682754516	121.43745422362	40.24758338928
50000	82046.29654884340	652.24146842956	250.03833770752	88.90070855220
100000	327611.00273132300	1614.64838981630	529.93049607668	208.83755683900





log(n)	log(P1)	log(P2)	log(P3)	log(P4)
0.69897	-0.21523500499			
1.69897	-0.00283640419	-0.22302078964	-0.69869093973	
2	0.18134315012	-0.30290508531	-0.15602950641	
2.69897	0.97316907532	0.55913225065	0.38057652889	-0.30064746566
3	1.55558531629	0.87531963203	0.60275275894	0.20461888101
3.69897	2.91826573375	1.52887664349	1.35680378807	0.77184080670
4	3.51865916629	1.88272046010	1.64881181353	1.10792212968
4.39794	4.31354227567	2.42863172421	2.08435265423	1.60473980887
4.69897	4.91405898241	2.81440840707	2.39800660299	1.94890522238
5	5.51535847900	3.20807796383	2.72421891271	2.31980860382



n	trial	prime1	prime2	prime3	prime4
5	1	1.0221004486	0.0000000000	0.0000000000	0.0000000000
	2	1.0142326355	0.0000000000	0.0000000000	0.0000000000
	3	1.0097026883	0.0000000000	0.0000000000	0.0000000000
	4	0.0000000000	0.0000000000	0.0000000000	0.0000000000
	5	0.0000000000	0.0000000000	0.0000000000	0.0000000000
	avg	0.6092071545	0.0000000000	0.0000000000	0.0000000000
50	1	0.9841918940	0.0000000000	0.0000000000	0.0000000000
	2	1.0001659393	1.0006427765	0.0000000000	0.0000000000
	3	0.9903907776	0.9913444519	0.0000000000	0.0000000000
	4	0.9906291962	0.0000000000	1.0006427765	0.0000000000
	5	1.0020732880	0.9999275208	0.0000000000	0.0000000000
	avg	0.9934902190	0.5983829498	0.2001285553	0.0000000000
100	1	1.4865398407	0.5004405975	0.5002021790	0.0000000000
	2	2.0005702972	0.0000000000	1.0008811951	0.0000000000
	3	2.0754337311	0.0000000000	0.9896755219	0.0000000000
	4	1.0204315186	1.0027885437	1.0001659393	0.0000000000
	5	1.0082721710	0.9860002432	0.0000000000	0.0000000000
	avg	1.5182495117	0.4978458769	0.6981849671	0.0000000000
500	1	8.9898109436	3.0014514920	2.0015239716	1.0008811951
	2	10.9980106354	3.0021667480	3.0014514923	0.0000000000
	3	8.0049037933	5.1097869873	2.0029544830	0.5006790161
	4	8.0037117004	5.0034523010	2.0015239716	1.0006427786
	5	11.0080242157	2.0008087150	3.0026435852	0.0000000000
	avg	9.4008922577	3.6235332487	2.4020195007	0.5004405980
1000	1	36.5421772003	7.5049400330	4.0209293365	2.0012855530
	2	36.0369682312	7.0052146912	4.0028095245	1.0006427765
	3	36.0405445099	8.0010890961	4.0028095240	3.0057430267
	4	35.0244045250	7.0052146912	4.0025711060	1.0006427765
	5	36.0589027405	8.0058574677	4.0028090952	1.0008811951
	avg	35.9405994414	7.5044631958	4.0063857172	1.6018390656
5000	1	826.8566131592	32.5639247894	22.0155715942	6.0136318207
	2	829.6489715576	34.0144634247	22.0160484313	6.5073966980
	3	832.0326805115	35.0093841553	22.0668315887	5.0129890442
	4	825.3982067108	33.8699817657	23.0169296265	6.0141086578
	5	828.3081054688	33.5266590118	24.5881080627	6.0191154480
	avg	828.4489154816	33.7968826294	22.7406978607	5.9134483337
10000	1	3287.5754833221	73.5733509064	46.0357666016	13.0321979523
	2	3297.1332073212	74.6176242828	43.0970191956	12.0179653168
	3	3300.1337281318	75.0448703766	43.0314540863	13.0083560944
	4	3287.8327369690	75.7150650024	42.0305728912	13.0269527435
	5	3332.8433036804	82.7212333680	48.5367774936	13.0195617676
	avg	3301.1036918849	76.3344287872	44.5463180537	12.8210067749
25000	1	20545.0928211212	272.4277973175	126.8544197083	41.1808490753
	2	20641.5278911591	266.6149139404	119.1573143005	40.0283336639
	3	20633.7335109711	262.3808383942	117.1705722809	39.9568080902
	4	20602.2589206696	272.9809284210	126.9195079803	40.0280952454
	5	20500.3497600555	267.1296596527	117.0854568481	40.0438308716
	avg	20584.5925807953	268.3068275452	121.4374542236	40.2475833893
50000	1	82484.3301773071	650.6209373474	240.4413223267	89.6654129028
	2	82061.7308616638	642.5268650055	245.2104091644	88.5748833220
	3	81933.3755970001	643.3169841766	242.7785396576	88.0651473999
	4	81806.5919876099	665.9986972809	258.4340572357	89.6036624908
	5	81945.4541206360	658.7438583374	263.3273601532	88.5944366455
	avg	82046.2965488434	652.2414684296	250.0383377075	88.9007085522
100000	1	327447.6284980770	1614.8319244385	547.5475781160	208.8568210602
	2	326668.4720516200	1617.1636581421	537.8677845001	210.1857662201
	3	327583.3201408380	1637.0341777802	541.3825511932	206.6252231598
	4	327469.5429801940	1590.0194644928	512.9907131195	209.8319530487
	5	328886.0499858850	1614.1927242279	509.8638534546	208.6880207062
	avg	327611.0027313230	1614.6483898163	529.9304960767	208.8375568390

n	n2	n3	n4
5	25	125	625
50	2500	125000	6250000
100	10000	1000000	100000000
500	250000	125000000	62500000000
1000	1000000	1000000000	1000000000000
5000	25000000	125000000000	625000000000000
10000	100000000	1000000000000	10000000000000000
25000	625000000	15625000000000	390625000000000000
50000	2500000000	125000000000000	6250000000000000000
100000	10000000000	1000000000000000	100000000000000000000
191655	13251262525	1141751126125120	10665125106260600000

P1 Sum	P2 Sum	P3 Sum	P4 Sum
434419.9069	2657.552222	976.0000227	358.8225836

P1x Sum	P2x Sum	P3x Sum	P4x Sum
37415224042.18	201726307.73	69095356.74	26494610.07

P1x2 Sum
3494441998948910.00

P1x	P2x	P3x	P4x	P1x2	P2x2	P3x2	P4x2
3.05	0.00	0.00	0.00	15.23	0.00	0.00	0.00
49.67	29.92	10.01	0.00	2483.73	1495.96	500.32	0.00
151.82	49.78	69.82	0.00	15182.50	4978.46	6981.85	0.00
4700.45	1811.77	1201.01	250.22	2350223.06	905883.31	600504.88	125110.15
35940.60	7504.46	4006.39	1601.84	35940599.44	7504463.20	4006385.72	1601839.07
4142244.58	168984.41	113703.49	29567.24	20711222887.04	844922065.73	568517446.52	147836208.34
33011036.92	763344.29	445463.18	128210.07	330110369188.49	7633442878.72	4454631805.37	1282100677.49
514614814.52	6707670.69	3035936.36	1006189.58	12865370362997.10	167691767215.73	75898408889.76	25154739618.30
4102314827.44	32612073.42	12501916.89	4445035.43	205115741372108.00	1630603671073.90	625095844268.80	222251771380.50
32761100273.13	161464838.98	52993049.61	20883755.68	3276110027313230.00	16146483898163.00	5299304960766.80	2088375568390.00

	P1	P2	P3	P4
C	-5.15776545510969			
B	0.00569679139104	-35.97721715121190	-3.22859560116248	-3.37198697270451
A	0.00003270478305	0.01574352035435	0.00526094272892	0.00204817225393

```
# Python 3.6
# CS 317 Algorithm Analysis
# Lab 1
# Drake Song

import time

def primelist1(n):
    print("2 is prime")
    for j in range(3, n+1):
        isPrime = True
        for i in range(2, j):
            if j%i == 0:
                isPrime = False
        if isPrime:
            print("{} is prime".format(j))


def primelist2(n):
    print("2 is prime")
    for j in range(3, n+1):
        isPrime = True
        for i in range(2, int(j**(0.5))+1):
            if j%i == 0:
                isPrime = False
        if isPrime:
            print("{} is prime".format(j))


def primelist3(n):
    print("2 is prime")
    for j in range(3, n+1):
        isPrime = True
        for i in range(2, int(j**(0.5))+1):
            if j%i == 0:
                isPrime = False
                break
        if isPrime:
            print("{} is prime".format(j))


def primelist4(n):
    primeList = [2]
    for j in range(3, n+1):
        isPrime = True
        for i in range(2, int(j**(0.5))+1):
            if j%i == 0:
                isPrime = False
```

```
                break
        if isPrime:
            primeList.append(j)
    print(primeList)


a = int(input("Enter a number: "))
while a < 2:
    a = int(input("Please enter an integer greater than 1: "))

print("\nPrimeList1")
tic1 = time.time()
primelist1(a)
toc1 = time.time()

print("\nPrimeList2")
tic2 = time.time()
primelist2(a)
toc2 = time.time()

print("\nPrimeList3")
tic3 = time.time()
primelist3(a)
toc3 = time.time()

print("\nPrimeList4")
tic4 = time.time()
primelist4(a)
toc4 = time.time()

print("")
print("Time1: {}ms".format(round(1000*(toc1-tic1), 10)))
print("Time2: {}ms".format(round(1000*(toc2-tic2), 10)))
print("Time3: {}ms".format(round(1000*(toc3-tic3), 10)))
print("Time3: {}ms".format(round(1000*(toc4-tic4), 10)))
```