Eli Sobylak
HW 2

20-oct-15

Ch 2

2.2 $T_1(N) = O(f(N))$
$T_2(N) = O(f(N))$

a. $T_1(N) + T_2(N) = O(f(N))$

$T_1(N) + T_2(N) = O(f(N) + f(N))$

false

b. $T_1(N) - T_2(N) = O(f(N))$

true

c. $\frac{T_1(N)}{T_2(N)} = O1$

true

d. $T_1(N) = O(T_2(N))$

true

2.3 which grows faster?

$N \log N$ $\qquad$ $N^{1+\epsilon/\sqrt{\log N}}$, $\epsilon > 0$

$N^{1+\epsilon/\sqrt{\log N}}$ grows way faster since it is exponential and $N \log N$ is actually closer to linear.

2.7

a.

(1)
```
Sum = 0;
for(i=0; i<N; i++)
    ++Sum
```
(N+1) (N)

points: 1, 2N+1, N

total: 3N+3

$\Theta(N)$

(2)
```
Sum = 0
for(i=0; i<N; i++)
  for(j=0; j<N; j++)
      ++Sum
```
points: 1, 2N+1 > 2N²+1, 2N+1, N

total: 2N²+N+1

$\Theta(N^2)$

(3)
```
sum=0
for(i=0; i<N; i++)
  for(j=0; j<N*N; j++)
      ++Sum
```
(N+2) (N)

points: 1, 2N+1 > 4N²+8N+5, 2N+3, N

total: 4N²+7N+5

$\Theta(N^2)$

(4)
```
sum=0
for(i=0; i<N; i++)
  for(j=0; j<i*i; j++)
    for(k=0; k<j; k++)
        ++Sum
```
points: 1, 2N+1, 2N+3, 2N+1, N

total: N³

$\Theta(N^3)$

(5)
```
Sum=0
for(i=0; i<n; i++)
  for(j=0; j<i*i; j++)
    for(k=0; k<j; k++)
        ++Sum
```
points: 1, 2N+1, 2N+3, 2N+1, N

total: N³

$\Theta(N^3)$

(6)
```
Sum = 0                                          | 1
for(i=0; i<N; i++)                               | 2N+1
    for(j=1; j<i*i; j++)                         | 3N+1
        for(k=0; k<j; k++)                       | 2N+1
            if(j % i == 0)                       | 2N+1
                Sum++                            | N
                                          total  | N^4
```

$O(N^4)$

b.

run	N = 10	1000	2000	4000
(1)	0.1s	0.1s	0.1s	0.1s
(2)	0.1s	0.1s	0.4s	1.5s
(3)	0.1s	N=200 0.7s	N=300 2.4s	N=400 5.5s
(4)	0.1s	0.1s	0.2s	0.7s
(5)	0.3s	N=80 1.7s	N=55 4.1s	N=60 6.4s
(6)	0.1s	N=100 1.1s	N=150 5.6s	N=200 19.4s

time

c. The fondonss above do match the run-times I calculated by hand in part a.

2.14

$$f(x) = 4x^4 + 8x^3 + x + 2 \quad , \quad x = 3$$

```
acc = 0
for c in (coeff)
    acc = acc * x + c
return acc
```

a)

1: acc = 0 * 3 + 4
   acc = 4

2: acc = 4 * 3 + 8
   acc = 20

3: acc = 20 * 3 + 1
   acc = 61

4: acc = 61 * 3 + 2
   acc = 185

b) It works because a function at a specific value will return a constant

c)

```
def horner(coeffs, x):          1
    acc = 0                     1
    for c in reversed(coeffs):  2N+1
        acc = acc * x + c       3
    print acc                   2N+5
```

$O(N)$

2.15

Takes array a of sorted nos and a search element i

```
int low = 0
int high = a.length() - 1

while (low <= high)
    int middle = (low + high) / 2

    if (a[middle] < i)
        low = middle + 1
    else if (a[middle] > i)
        high = middle - 1
    else
        print middle

print "not found"
```

run-time analysis

Counts by line:

1

2

N

N

N

N

1

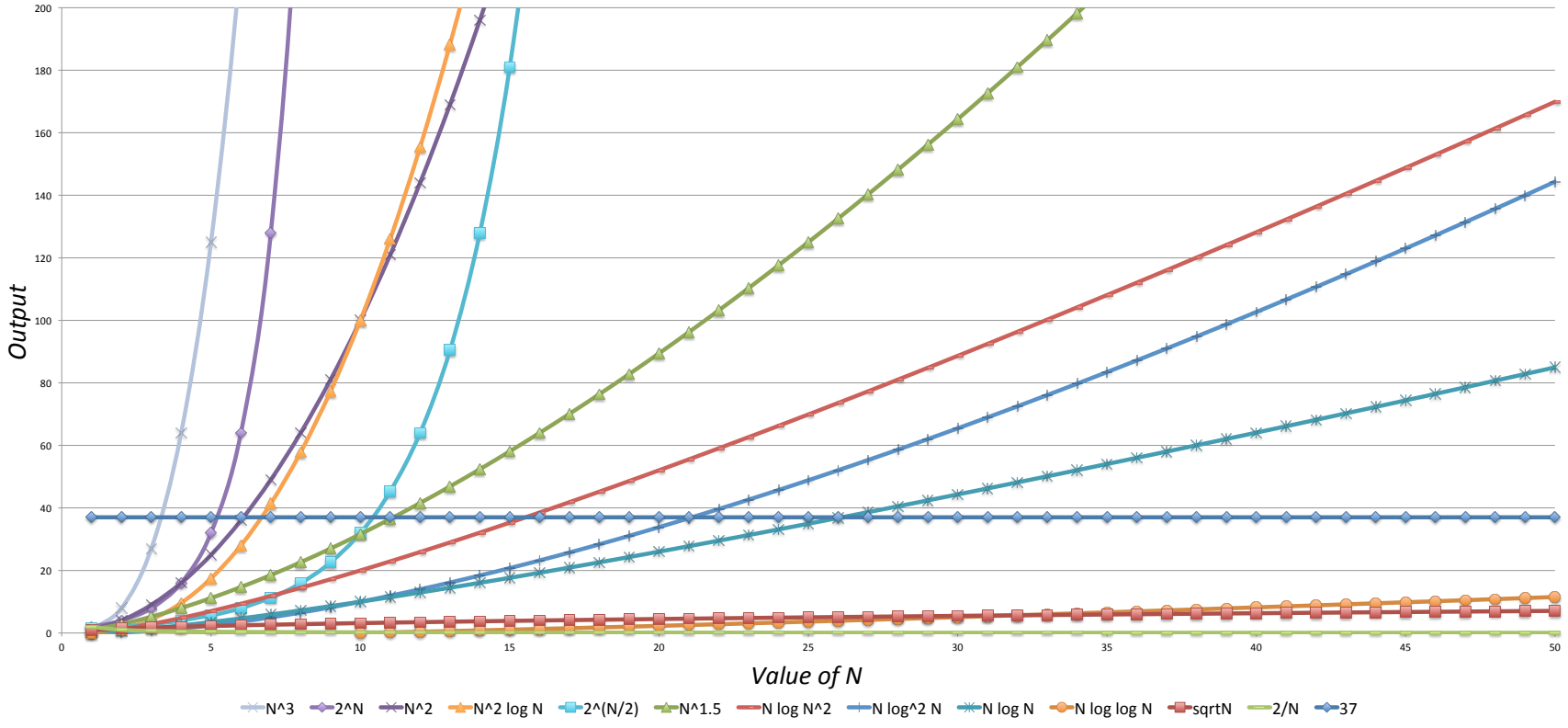
1

1

The run-time analysis would be $\Theta(N^3)$

Graphical Representation of Value of N vs. the Output of a function on value of N

This is a graphical representation of the problem in the textbook. I have done this because this give me more of an understanding on how these functions behave than if I were to see results in numbers in a table (which I have also provided) I have understood how fast these various functions grow by putting them in descending growth rate in the legend. The exception being the constant, 37.

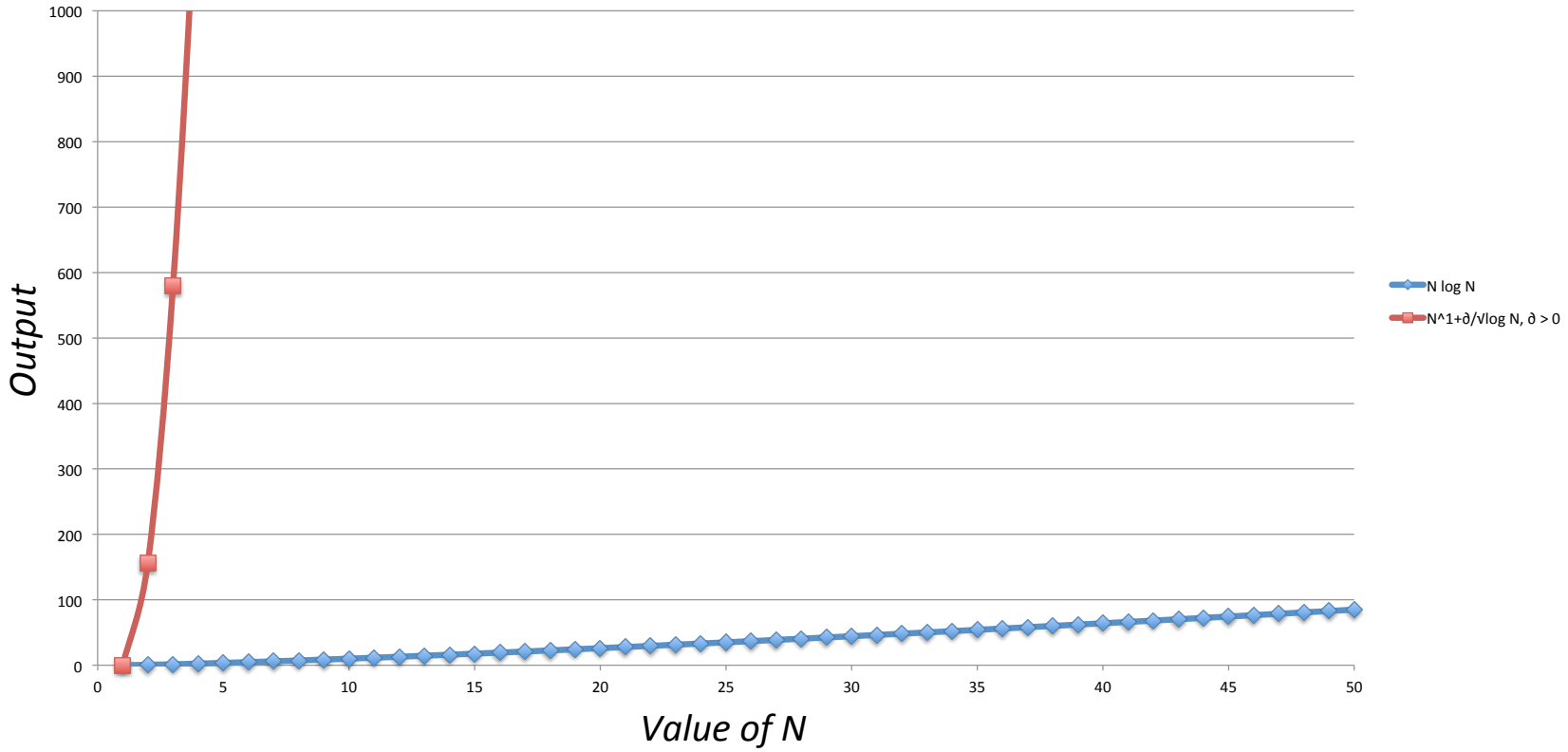


Data for question 2.1

N	37	sqrtN	N^1.5	N^2	N log N	N log log N	N log^2 N	N log N^2	2/N	2^N	2^(N/2)	N^2 log N	N^3
1	37	1	1	1	0	#NUM!	0	0	2	2	1.41421356	0	1
2	37	1.41421356	2.82842712	4	0.60205999	-1.0427805	0.18123812	1.20411998	1	4	2	1.20411998	8
3	37	1.73205081	5.19615242	9	1.43136376	-0.9641137	0.68293408	2.86272753	0.66666667	8	2.82842712	4.29409129	27
4	37	2	8	16	2.40823997	-0.8814409	1.44990493	4.81647993	0.5	16	4	9.63295986	64
5	37	2.23606798	11.1803399	25	3.49485002	-0.7777073	2.44279533	6.98970004	0.4	32	5.65685425	17.4742501	125
6	37	2.44948974	14.6969385	36	4.6689075	-0.6536159	3.63311621	9.337815	0.33333333	64	8	28.013445	216
7	37	2.64575131	18.5202592	49	5.91568628	-0.5116503	4.99933488	11.8313726	0.28571429	128	11.3137085	41.409804	343
8	37	2.82842712	22.627417	64	7.2247199	-0.3541518	6.5245722	14.4494398	0.25	256	16	57.7977592	512
9	37	3	27	81	8.58818258	-0.1830712	8.1952089	17.1763652	0.22222222	512	22.627417	77.2936433	729
10	37	3.16227766	31.6227766	100	10	0	10	20	0.2	1024	32	100	1000
11	37	3.31662479	36.4828727	121	11.4553195	0.19375975	11.929486	22.9106391	0.18181818	2048	45.254834	126.008515	1331
12	37	3.46410162	41.5692194	144	12.950175	0.39713268	13.9755859	25.9003499	0.16666667	4096	64	155.402099	1728
13	37	3.60555128	46.8721666	169	14.4812636	0.60922038	16.1313073	28.9625272	0.15384615	8192	90.509668	188.256427	2197
14	37	3.74165739	52.3832034	196	16.0457925	0.8292639	18.3905326	32.091585	0.14285714	16384	128	224.641095	2744
15	37	3.87298335	58.0947502	225	17.6413689	1.05661533	20.7478597	35.2827378	0.13333333	32768	181.019336	264.620533	3375
16	37	4	64	256	19.2659197	1.29071622	23.1984789	38.5318394	0.125	65536	256	308.254716	4096
17	37	4.12310563	70.0927956	289	20.9176317	1.53108103	25.7380773	41.8352633	0.11764706	131072	362.038672	355.599738	4913
18	37	4.24264069	76.3675324	324	22.5949051	1.77728429	28.3627631	45.1898102	0.11111111	262144	512	406.708292	5832
19	37	4.35889894	82.8190799	361	24.2963184	2.02895052	31.0690047	48.5926368	0.10526316	524288	724.077344	461.63005	6859
20	37	4.47213595	89.4427191	400	26.0205999	2.28574619	33.853581	52.0411998	0.1	1048576	1024	520.411998	8000
21	37	4.58257569	96.2340896	441	27.7666052	2.5473733	36.7135411	55.5332104	0.0952381	2097152	1448.15469	583.098709	9261
22	37	4.69041576	103.189147	484	29.533299	2.81356419	39.6461704	59.066598	0.09090909	4194304	2048	649.732578	10648
23	37	4.79583152	110.304125	529	31.3197402	3.08407725	42.6489621	62.6394805	0.08695652	8388608	2896.30938	720.354025	12167
24	37	4.89897949	117.575508	576	33.1250698	3.35869345	45.7195937	66.2501396	0.08333333	16777216	4096	795.001675	13824
25	37	5	125	625	34.9485002	3.63721336	48.8559067	69.8970004	0.08	33554432	5792.61875	873.712505	15625
26	37	5.09901951	132.574507	676	36.789307	3.91945475	52.055889	73.5786141	0.07692308	67108864	8192	956.521983	17576
27	37	5.19615242	140.296115	729	38.6468216	4.2052505	55.3176601	77.2936433	0.07407407	134217728	11585.2375	1043.46418	19683
28	37	5.29150262	148.162073	784	40.5204249	4.49444686	58.6394583	81.0408498	0.07142857	268435456	16384	1134.5719	21952
29	37	5.38516481	156.169779	841	42.4095419	4.78690193	62.0196292	84.8190839	0.06896552	536870912	23170.475	1229.87672	24389
30	37	5.47722558	164.316767	900	44.3136376	5.08248442	65.456616	88.6272753	0.06666667	1073741824	32768	1329.40913	27000
31	37	5.56776436	172.600695	961	46.2322125	5.3810725	68.9489508	92.464425	0.06451613	2147483648	46340.95	1433.19859	29791
32	37	5.65685425	181.019336	1024	48.1647993	5.68255285	72.4952466	96.3295986	0.0625	4294967296	65536	1541.27358	32768
33	37	5.74456265	189.570567	1089	50.11096	5.98681984	76.0941913	100.22192	0.06060606	8589934592	92681.9	1653.66168	35937
34	37	5.83095189	198.252364	1156	52.0702832	6.29377477	79.7445409	104.140566	0.05882353	17179869184	131072	1770.38963	39304
35	37	5.91607978	207.062792	1225	54.0423816	6.60332523	83.4451144	108.084763	0.05714286	34359738368	185363.8	1891.48335	42875
36	37	6	216	1296	56.02689	6.91538455	87.1947891	112.05378	0.05555556	68719476736	262144	2016.96804	46656
37	37	6.08276253	225.062214	1369	58.0234638	7.2298713	90.992496	116.046928	0.05405405	1.37439E+11	370727.6	2146.86816	50653
38	37	6.164414	234.247732	1444	60.0317767	7.5467088	94.8372161	120.063553	0.05263158	2.74878E+11	524288	2281.20751	54872
39	37	6.244998	243.554922	1521	62.0515197	7.86582479	98.7279768	124.103039	0.05128205	5.49756E+11	741455.2	2420.00927	59319
40	37	6.32455532	252.982213	1600	64.0823997	8.18715099	102.663849	128.164799	0.05	1.09951E+12	1048576	2563.29599	64000
41	37	6.40312424	262.528094	1681	66.1241381	8.51062288	106.643943	132.248276	0.04878049	2.19902E+12	1482910.4	2711.08966	68921
42	37	6.4807407	272.191109	1764	68.1764702	8.83617931	110.667407	136.35294	0.04761905	4.39805E+12	2097152	2863.41175	74088
43	37	6.55743852	281.969857	1849	70.2391436	9.16376234	114.733425	140.478287	0.04651163	8.79609E+12	2965820.8	3020.28317	79507
44	37	6.63324958	291.862982	1936	72.3119178	9.49331693	118.841215	144.623836	0.04545455	1.75922E+13	4194304	3181.72438	85184
45	37	6.70820393	301.869177	2025	74.3945631	9.82479078	122.990023	148.789126	0.04444444	3.51844E+13	5931641.6	3347.75534	91125
46	37	6.78232998	311.987179	2116	76.4868603	10.1581341	127.179126	152.973721	0.04347826	7.03687E+13	8388608	3518.39557	97336
47	37	6.8556546	322.215766	2209	78.5885993	10.4932995	131.407829	157.177199	0.04255319	1.40737E+14	11863283.2	3693.66417	103823
48	37	6.92820323	332.553755	2304	80.6995794	10.8302416	135.675461	161.399159	0.04166667	2.81475E+14	16777216	3873.57981	110592
49	37	7	343	2401	82.8196079	11.1689174	139.981377	165.639216	0.04081633	5.6295E+14	23726566.4	4058.16079	117649
50	37	7.07106781	353.553391	2500	84.9485002	11.5092856	144.324954	169.897	0.04	1.1259E+15	33554432	4247.42501	125000

Graphical Representation of the Value of N vs. the Output either function on N

The same goes for this graph. From doing this I can clearly see that N log N grows slower than the other function. And I would believe that this is due to the fact that the other function has an exponent which is a pretty good guarantee of quick growth.



Data for question 2.3

N	N log N	N^1+∂/√log N, ∂ > 0
1	0	#DIV/0!
2	0.60205999	156.5481788
3	1.43136376	579.3734277
4	2.40823997	1269.699633
5	3.49485002	2208.905005
6	4.6689075	3376.877212
7	5.91568628	4755.197538
8	7.2247199	6327.641634
9	8.58818258	8080.039541
10	10	10000
11	11.4553195	12076.63603
12	12.950175	14300.32628
13	14.4812636	16662.51653
14	16.0457925	19155.55673
15	17.6413689	21772.5678
16	19.2659197	24507.3323
17	20.9176317	27354.2043
18	22.5949051	30308.03457
19	24.2963184	33364.10812
20	26.0205999	36518.09168
21	27.7666052	39765.98927
22	29.533299	43104.10439
23	31.3197402	46529.00759
24	33.1250698	50037.50861
25	34.9485002	53626.63221
26	36.789307	57293.59719
27	38.6468216	61035.79806
28	40.5204249	64850.78886
29	42.4095419	68736.26906
30	44.3136376	72690.07092
31	46.2322125	76710.1484
32	48.1647993	80794.5672
33	50.11096	84941.49589
34	52.0702832	89149.19795
35	54.0423816	93416.02458
36	56.02689	97740.40831
37	58.0234638	102120.8571
38	60.0317767	106555.9491
39	62.0515197	111044.3277
40	64.0823997	115584.6975
41	66.1241381	120175.8199
42	68.1764702	124816.5097
43	70.2391436	129505.6317
44	72.3119178	134242.0976
45	74.3945631	139024.8632
46	76.4868603	143852.9257
47	78.5885993	148725.3216
48	80.6995794	153641.1241
49	82.8196079	158599.4412
50	84.9485002	163599.4139

Eli Sobylak
20-oct-15

2.7: My code for 2.7 implemented in python

Implemented in Python

```
def algoOne(num):
sum = 0
for i in xrange(0,num):
        sum+=1
print sum

algoOne(num)

def algoTwo(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, num):
                sum+=1
print sum

algoTwo(num)

def algoThree(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0,num*num):
                sum+=1
print sum

algoThree(num)

def algoFour(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, i):
                sum+=1
print sum

algoFour(num)
```

```
def algoFive(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, i*i):
                for k in xrange(0, j):
                        sum+=1
print sum

algoFive(num)

def algoSix(num):
sum = 0
for i in xrange(1,num):
        for j in xrange(1, i*i):
                if(j % i == 0):
                        for k in xrange(0, j):
                                sum+=1
print sum

algoSix(num)
```

Eli Sobylak
20-oct-15

2.7: My code for 2.7 implemented in python

Implemented in Python

```
def algoOne(num):
sum = 0
for i in xrange(0,num):
        sum+=1
print sum

algoOne(num)

def algoTwo(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, num):
                sum+=1
print sum

algoTwo(num)

def algoThree(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0,num*num):
                sum+=1
print sum

algoThree(num)

def algoFour(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, i):
                sum+=1
print sum

algoFour(num)
```

```
def algoFive(num):
sum = 0
for i in xrange(0,num):
        for j in xrange(0, i*i):
                for k in xrange(0, j):
                        sum+=1
print sum

algoFive(num)

def algoSix(num):
sum = 0
for i in xrange(1,num):
        for j in xrange(1, i*i):
                if(j % i == 0):
                        for k in xrange(0, j):
                                sum+=1
print sum

algoSix(num)
```