How to analyze given algorithm/pseudocode

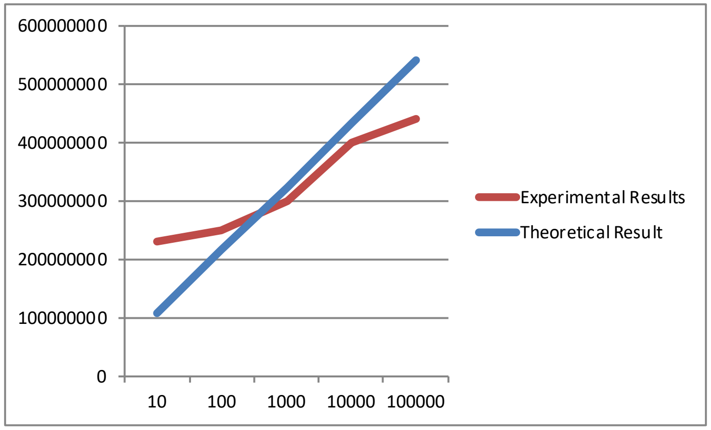
Suppose we are given some pseudocode using for loops etc.

Step 1 We analyze the pseudocode using the techniques discussed in class (nested for loops, etc).
Using this, we are able to derive a hypothesis on what the time complexity is, asymptotically.

Step 2 Now, we enter the numerical/practical portion where we have to validate our hypothesis.
To validate our hypothesis, we implement the pseudocode in any language like Java/C# etc and find the elapsed time.
Then, we compare the theoretically computed value (the hypothesis) with the numerically calculated elapsed time.

Step 2b However, we have one detail here.
The theoretical values do not have any units, since we only say something like O(n^2).
The numerical values have units like millisecond, nanosecond etc.
For this we need to use scaling on one of the values.

n	Experimental Result, in ns	Theoretical Result	Scaling Constant	Adjusted Theoretical Result
10	230878766	3.32192809		108258292
100	250789567	6.64385619		216516584
1000	300457235	9.96578428		324774876
10000	400895231	13.2877124		433033168
100000	440853582	16.6096405		541291460
	324774876.2	9.96578428	32588993.2	



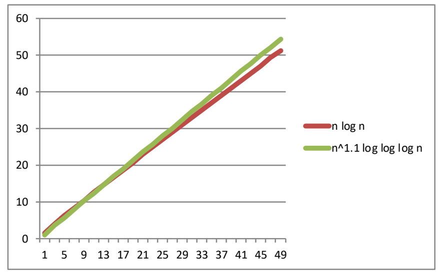
Step 3 Now we simply plot the two series (experimental result vs. adjusted theoretical result)

Step 4 Analyze the plots and reach a conclusion.
If the plots are diverging that is a hint that our analysis (and the subsequent hypothesis) may not be correct.
If the experintal result plot is too jumpy, we can try higher n values (computer too fast for small calculations)

How to compare two Asymptotic Functions Numerically

Suppose we have to compare n log n vs. n^1.1 log log log n

Some n values..	To compare, we simply calculate numericall		*To Plot, we take a log based 10 of all of these*		
n	n logn	n^1.1 log log log n	*log(a,10)*	*log(b,10)*	*log(c,10)*
10	33.21928095	9.976433658	1	1.52139023	0.99897532
1000	9965.784285	3451.54795	3	3.99851148	3.53801391
100000	1660964.047	638567.5564	5	6.22036023	5.80520685
10000000	232534966.6	109383809	7	8.36648827	8.03895304
1000000000	29897352854	18216762685	9	10.4756327	10.2604712
1E+11	3.65412E+12	2.99138E+12	11	12.5627829	12.4758717
1E+13	4.31851E+14	4.87164E+14	13	14.6353336	14.6876755
1E+15	4.98289E+16	7.8912E+16	15	16.6974815	16.8971431
1E+17	5.64728E+18	1.27346E+19	17	18.7518391	19.1049861
1E+19	6.31166E+20	2.04947E+21	19	20.8001438	21.3116421
1E+21	6.97605E+22	3.29153E+23	21	22.8436095	23.5173976
1E+23	7.64043E+24	5.27776E+25	23	24.8831181	25.7224494
1E+25	8.30482E+26	8.45158E+27	25	26.9193302	27.9269378
1E+27	8.96921E+28	1.35197E+30	27	28.952754	30.1309665
1E+29	9.63359E+30	2.16079E+32	29	30.9837882	32.3346134
1E+31	1.0298E+33	3.45095E+34	31	33.0127519	34.5379392
1E+33	1.09624E+35	5.50797E+36	33	35.0399042	36.7409912
1E+35	1.16267E+37	8.78633E+38	35	37.0654583	38.9438077
1E+37	1.22911E+39	1.40094E+41	37	39.089592	41.1464194
1E+39	1.29555E+41	2.23281E+43	39	41.1124548	43.3488518
1E+41	1.36199E+43	3.55734E+45	41	43.1341741	45.5511259
1E+43	1.42843E+45	5.66578E+47	43	45.1548587	47.7532593
1E+45	1.49487E+47	9.02126E+49	45	47.1746027	49.9552672
1E+47	1.56131E+49	1.43603E+52	47	49.1934881	52.1571622
1E+49	1.62774E+51	2.28536E+54	49	51.2115863	54.3589553



What the curve shows is that while the functions are similar, they are still DIVERGING

We observe that n^1.1 log log logn is growing faster (albeit ever so slightly) compared to n log n