DESIGN AND ANALYSIS OF ALGORITHMS LAB

NAME: Mohammed Saad Belgi

UID: 2021700005

BATCH: A

BRANCH: CSE DS

EXPT. NO.: 1A

AIM: To implement the various functions e.g., linear, non-linear, quadratic, exponential etc.

CODE:

```c
#include <stdio.h>
#include <math.h>

static double log_2;

// (3/2)^n
double f1(int n)
{
    return pow(1.5, n);
}

// n^3
int f2(int n)
{
    return pow(n, 3);
}

// 2^n
unsigned long long f3(int n)
{
    return pow(2, n);
}

// (log2 n)
double f4(double n)
{
    return (log(n) / log_2);
}

// (log2 n)^2
double f5(int n)
{
    return pow(f4(n), 2);
}
```

```cpp
// log2 (n!)
double f6(int n)
{
    if (n == 0)
        return 0;
    double ans = 0;
    for (int i = 1; i <= n; i++)
    {
        ans += log(i);
    }
    ans /= log_2;
    return ans;
}

// ln(ln(n))
double f7(int n)
{
    return log(log(n));
}

// nlog(n)
double f8(int n)
{
    return n * f4(n);
}

// (log2 (n))^(log2 (n))
double f9(int n)
{
    double a = f4(n);
    return pow(a, a);
}

// n^(log2(log2(n)))
double f10(int n)
{
    return pow(2.0, f4(f4(n)));
}

// n!
unsigned long long f11(int n)
{
    if (n == 0 || n == 1)
        return 1;
    return n * f11(n - 1);
}
```

```c
int main()
{
    log_2 = log(2);
    printf("n  |(3/2)^n                    |n^3     |2^n
     |log2(n)   |(log2(n))^2|(log2(n!))  |ln(ln(n))  |nlog2(n)
|(log2(n))^(log2(n))|n^(log2(log2(n)))|n!                    |\n");
    for (int i = 0; i < 181; i++)
    {
        printf("-");
    }
    printf("\n");
    for (int i = 0; i <= 100; i++)
    {
        printf("%3d", i);
        printf(" | %23.3f", f1(i));
        printf(" | %7d", f2(i));
        printf(" | %19llu", f3(i));
        printf(" | %8.3f", f4(i));
        printf(" | %9.3f", f5(i));
        printf(" | %10.3f", f6(i));
        printf(" | %9.3f", f7(i));
        printf(" | %7.3f", f8(i));
        printf(" | %17.3f", f9(i));
        printf(" | %15.3f", f10(i));
        printf(" | ");
        if (i <= 20)
            printf("%19llu |", f11(i));
        printf("\n");
        for (int i = 0; i < 181; i++)
        {
            printf("-");
        }
        printf("\n");
    }
}
```

EXECUTION AND OUTPUT:

As the output was too large to display with correct formatting on terminal, it was redirected to a text file.

```
PS C:\Users\arifa\Desktop\sem4 work\daa lab\exp0> gcc prog.c -o prog
PS C:\Users\arifa\Desktop\sem4 work\daa lab\exp0> .\prog > output.txt
```

output.txt file:

n	(3/2)^n	n^3	2^n	log2(n)	(log2(n))^2	(log2(n!))	ln(ln(n))	nlog2(n)	(log2(n))^(log2(n))	n^(log2(log2(n)))	n!
0	1.000	0	1	-1.#IO	1.#IO	0.000	-1.#IO	-1.#IO	0.000	-1.#IO	1
1	1.500	1	2	0.000	0.000	0.000	-1.#IO	0.000	1.000	-1.#IO	1
2	2.250	8	4	1.000	1.000	1.000	-0.367	2.000	1.000	1.000	2
3	3.375	27	8	1.585	2.512	2.585	0.094	4.755	2.075	2.075	6
4	5.063	64	16	2.000	4.000	4.585	0.327	8.000	4.000	4.000	24
5	7.594	124	32	2.322	5.391	6.907	0.476	11.610	7.071	7.071	120
6	11.391	216	64	2.585	6.682	9.492	0.583	15.510	11.646	11.646	720
7	17.086	343	128	2.807	7.881	12.299	0.666	19.651	18.136	18.136	5040
8	25.629	512	256	3.000	9.000	15.299	0.732	24.000	27.000	27.000	40320
9	38.443	729	512	3.170	10.048	18.469	0.787	28.529	38.751	38.751	362880
10	57.665	1000	1024	3.322	11.035	21.791	0.834	33.219	53.954	53.954	3628800
11	86.498	1331	2048	3.459	11.968	25.250	0.875	38.054	73.223	73.223	39916800
12	129.746	1728	4096	3.585	12.852	28.835	0.910	43.020	97.231	97.231	479001600
13	194.620	2197	8192	3.700	13.693	32.536	0.942	48.106	126.703	126.703	6227020800
14	291.929	2744	16384	3.807	14.496	36.343	0.970	53.303	162.420	162.420	87178291200
15	437.894	3375	32768	3.907	15.264	40.250	0.996	58.603	205.220	205.220	1307674368000
16	656.841	4096	65536	4.000	16.000	44.250	1.020	64.000	256.000	256.000	20922789888000
17	985.261	4912	131072	4.087	16.707	48.338	1.041	69.487	315.715	315.715	355687428096000
18	1477.892	5832	262144	4.170	17.388	52.508	1.061	75.059	385.379	385.379	6402373705728000
19	2216.838	6858	524288	4.248	18.045	56.755	1.080	80.711	466.070	466.070	121645100408832000
20	3325.257	8000	1048576	4.322	18.679	61.077	1.097	86.439	558.924	558.924	2432902008176640000
21	4987.885	9260	2097152	4.392	19.292	65.470	1.113	92.239	665.143	665.143	
22	7481.828	10647	4194304	4.459	19.887	69.929	1.129	98.107	785.991	785.991	
23	11222.741	12167	8388608	4.524	20.463	74.453	1.143	104.042	922.798	922.798	
24	16834.112	13824	16777216	4.585	21.022	79.038	1.156	110.039	1076.961	1076.961	

25	25251.168	15624	33554432	4.644	21.565	83.682	1.169	116.096	1249.940	1249.940	
26	37876.752	17576	67108864	4.700	22.094	88.382	1.181	122.211	1443.266	1443.266	
27	56815.129	19683	134217728	4.755	22.609	93.137	1.193	128.382	1658.539	1658.539	
28	85222.693	21952	268435456	4.807	23.111	97.944	1.204	134.606	1897.427	1897.427	
29	127834.039	24389	536870912	4.858	23.600	102.802	1.214	140.881	2161.669	2161.669	
30	191751.059	26999	1073741824	4.907	24.078	107.709	1.224	147.207	2453.078	2453.078	
31	287626.589	29790	2147483648	4.954	24.544	112.663	1.234	153.580	2773.535	2773.535	
32	431439.883	32768	4294967296	5.000	25.000	117.663	1.243	160.000	3125.000	3125.000	
33	647159.825	35936	8589934592	5.044	25.446	122.708	1.252	166.465	3509.503	3509.503	
34	970739.737	39303	17179869184	5.087	25.882	127.795	1.260	172.974	3929.152	3929.152	
35	1456109.606	42874	34359738368	5.129	26.310	132.924	1.268	179.525	4386.131	4386.131	
36	2184164.409	46656	68719476736	5.170	26.728	138.094	1.276	186.117	4882.700	4882.700	
37	3276246.614	50653	137438953472	5.209	27.138	143.304	1.284	192.750	5421.198	5421.198	
38	4914369.920	54871	274877906944	5.248	27.541	148.552	1.291	199.421	6004.045	6004.045	
39	7371554.881	59319	549755813888	5.285	27.935	153.837	1.298	206.131	6633.738	6633.738	
40	11057332.321	64000	1099511627776	5.322	28.323	159.159	1.305	212.877	7312.856	7312.856	
41	16585998.481	68921	2199023255552	5.358	28.703	164.517	1.312	219.660	8044.061	8044.061	
42	24878997.722	74088	4398046511104	5.392	29.077	169.909	1.318	226.477	8830.098	8830.098	
43	37318496.583	79507	8796093022208	5.426	29.444	175.335	1.325	233.329	9673.793	9673.793	
44	55977744.875	85183	17592186044416	5.459	29.805	180.795	1.331	240.215	10578.059	10578.059	
45	83966617.312	91124	35184372088832	5.492	30.160	186.286	1.337	247.133	11545.895	11545.895	
46	125949925.968	97335	70368744177664	5.524	30.510	191.810	1.343	254.084	12580.384	12580.384	
47	188924888.952	103823	140737488355328	5.555	30.853	197.365	1.348	261.066	13684.699	13684.699	
48	283387333.428	110592	281474976710656	5.585	31.192	202.950	1.354	268.078	14862.099	14862.099	
49	425081000.143	117649	562949953421312	5.615	31.525	208.564	1.359	275.121	16115.934	16115.934	
50	637621500.214	125000	1125899906842624	5.644	31.853	214.208	1.364	282.193	17449.642	17449.642	

51	956432250.321	132651	2251799813685248	5.672	32.176	219.881	1.369	289.294	18866.754	18866.754
52	1434648375.482	140608	4503599627370496	5.700	32.495	225.581	1.374	296.423	20370.891	20370.891
53	2151972563.222	148876	9007199254740992	5.728	32.809	231.309	1.379	303.580	21965.768	21965.768
54	3227958844.834	157464	18014398509481984	5.755	33.119	237.064	1.384	310.764	23655.193	23655.193
55	4841938267.250	166375	36028797018963968	5.781	33.424	242.845	1.388	317.975	25443.069	25443.069
56	7262907400.876	175616	72057594037927936	5.807	33.725	248.653	1.393	325.212	27333.393	27333.393
57	10894361101.313	185193	144115188075855872	5.833	34.023	254.485	1.397	332.475	29330.259	29330.259
58	16341541651.970	195111	288230376151711744	5.858	34.316	260.343	1.401	339.763	31437.858	31437.858
59	24512312477.955	205379	576460752303423488	5.883	34.605	266.226	1.405	347.076	33660.480	33660.480
60	36768468716.933	215999	1152921504606846976	5.907	34.891	272.133	1.410	354.413	36002.511	36002.511
61	55152703075.400	226981	2305843009213693952	5.931	35.174	278.064	1.414	361.775	38468.440	38468.440
62	82729054613.099	238328	4611686018427387904	5.954	35.452	284.018	1.418	369.160	41062.855	41062.855
63	124093581919.649	250047	9223372036854775808	5.977	35.728	289.995	1.421	376.569	43790.444	43790.444
64	186140372879.473	262144	0	6.000	36.000	295.995	1.425	384.000	46656.000	46656.000
65	279210559319.210	274624	0	6.022	36.269	302.018	1.429	391.454	49664.418	49664.418
66	418815838978.815	287495	0	6.044	36.535	308.062	1.433	398.930	52820.695	52820.695
67	628223758468.223	300763	0	6.066	36.797	314.128	1.436	406.428	56129.937	56129.937
68	942335637702.334	314432	0	6.087	37.057	320.215	1.440	413.947	59597.353	59597.353
69	1413503456553.501	328508	0	6.109	37.314	326.324	1.443	421.488	63228.258	63228.258
70	2120255184830.252	342999	0	6.129	37.568	332.453	1.447	429.050	67028.075	67028.075
71	3180382777245.378	357911	0	6.150	37.819	338.603	1.450	436.632	71002.338	71002.338
72	4770574165868.066	373248	0	6.170	38.068	344.773	1.453	444.235	75156.686	75156.686
73	7155861248802.101	389017	0	6.190	38.314	350.963	1.456	451.857	79496.870	79496.870
74	10733791873203.150	405224	0	6.209	38.557	357.172	1.460	459.500	84028.753	84028.753
75	16100687809804.727	421875	0	6.229	38.798	363.401	1.463	467.161	88758.306	88758.306

76	24151031714707.090	438976	0	6.248	39.037	369.649	1.466	474.842	93691.618	93691.618
77	36226547572060.633	456532	0	6.267	39.273	375.916	1.469	482.543	98834.886	98834.886
78	54339821358090.953	474552	0	6.285	39.506	382.201	1.472	490.261	104194.424	104194.424
79	81509732037136.422	493039	0	6.304	39.738	388.505	1.475	497.999	109776.661	109776.661
80	122264598055704.640	511999	0	6.322	39.967	394.827	1.478	505.754	115588.142	115588.142
81	183396897083556.970	531441	0	6.340	40.194	401.167	1.480	513.528	121635.527	121635.527
82	275095345625335.440	551368	0	6.358	40.418	407.524	1.483	521.319	127925.596	127925.596
83	412643018438003.120	571787	0	6.375	40.641	413.899	1.486	529.128	134465.246	134465.246
84	618964527657004.750	592704	0	6.392	40.862	420.292	1.489	536.955	141261.494	141261.494
85	928446791485507.120	614124	0	6.409	41.080	426.701	1.491	544.798	148321.477	148321.477
86	1392670187228260.500	636056	0	6.426	41.297	433.127	1.494	552.659	155652.451	155652.451
87	2089005280842391.000	658502	0	6.443	41.512	439.570	1.496	560.536	163261.798	163261.798
88	3133507921263586.500	681471	0	6.459	41.724	446.030	1.499	568.430	171157.018	171157.018
89	4700261881895380.000	704969	0	6.476	41.935	452.505	1.502	576.340	179345.738	179345.738
90	7050392822843069.000	728999	0	6.492	42.144	458.997	1.504	584.267	187835.707	187835.707
91	10575589234264604.000	753571	0	6.508	42.351	465.505	1.506	592.209	196634.801	196634.801
92	15863383851396906.000	778687	0	6.524	42.557	472.029	1.509	600.168	205751.021	205751.021
93	23795075777095360.000	804356	0	6.539	42.761	478.568	1.511	608.142	215192.494	215192.494
94	35692613665643040.000	830584	0	6.555	42.963	485.122	1.514	616.131	224967.476	224967.476
95	53538920498464560.000	857374	0	6.570	43.163	491.692	1.516	624.136	235084.351	235084.351
96	80308380747696832.000	884736	0	6.585	43.362	498.277	1.518	632.156	245551.632	245551.632
97	120462571121545250.000	912672	0	6.600	43.559	504.877	1.521	640.192	256377.963	256377.963
98	180693856682317890.000	941192	0	6.615	43.754	511.492	1.523	648.242	267572.118	267572.118
99	271040785023476830.000	970298	0	6.629	43.948	518.121	1.525	656.306	279143.002	279143.002
100	406561177535215230.000	1000000	0	6.644	44.141	524.765	1.527	664.386	291099.655	291099.655

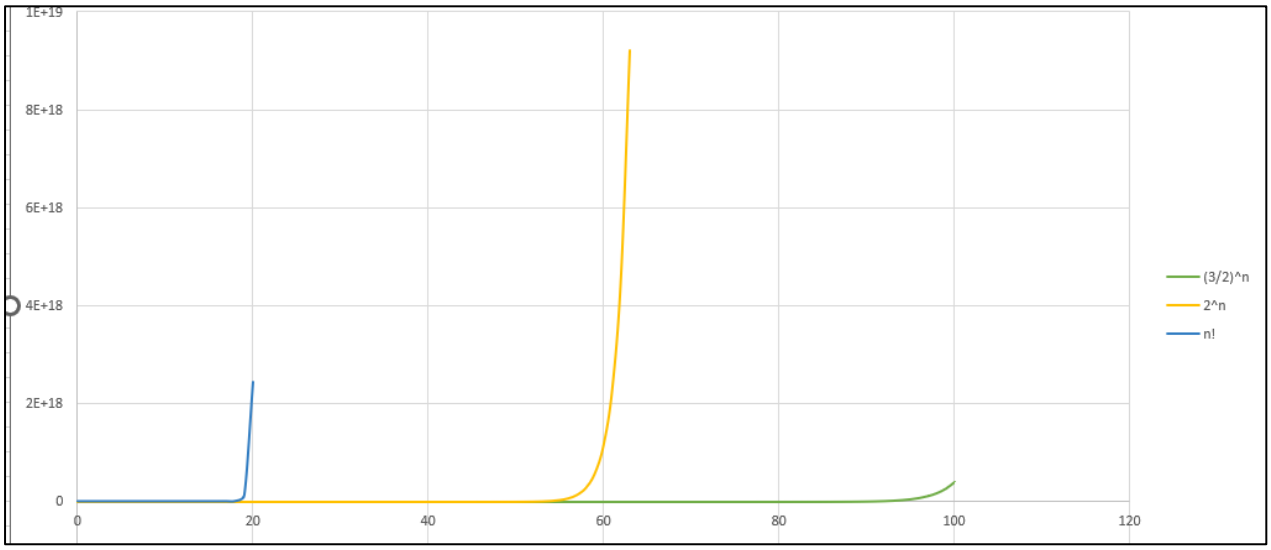
For n>=64, 2^n cannot be calculated on a 64-bit machine, as the largest unsigned integer that can be represent on a 64-bit architecture is $2^{64} - 1$ (18446744073709551615). Hence the output of 2^n for n>=64 was 0.

PLOTTING AND ANALYSING GRAPHS:

In the following discussion, n is a non-negative integer.

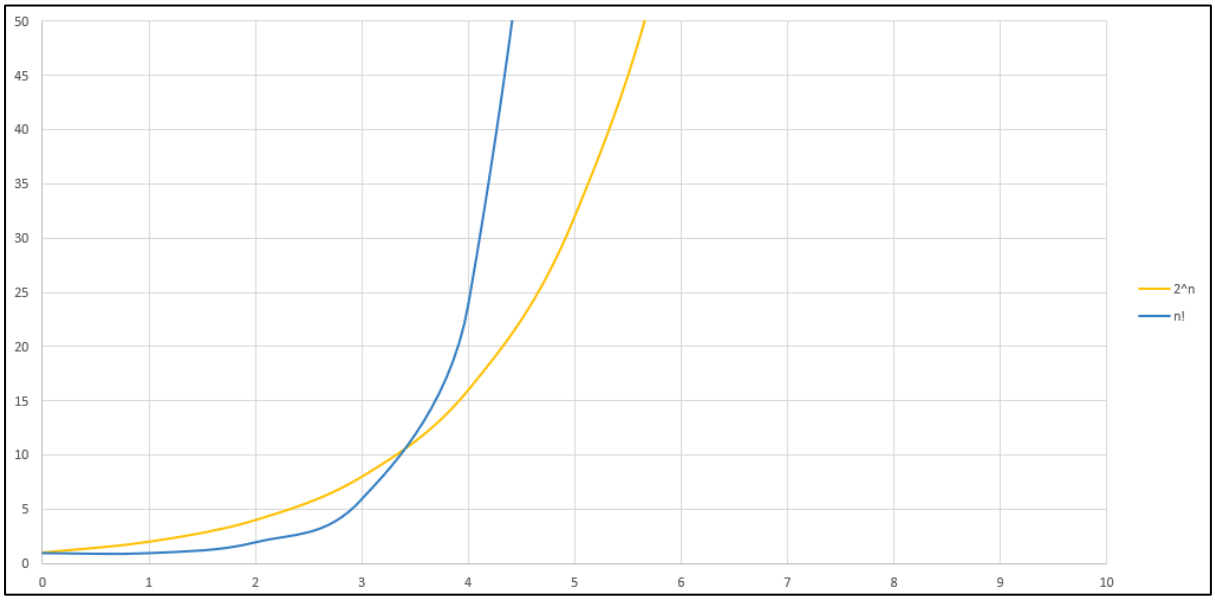
Comparing factorial and exponential functions:

Fastest growing functions were found to be n!, 2^n, 1.5^n respectively.



Largest unsigned integer that can be represented on a 64-bit machine is 2^{63}-1 (9.223372e+18). Hence value of n! can only be calculated for n<=20, as 21! (5.1090942e+19) exceeds this upper bound. Similarly, value of 2^n can only be calculated for n<=63 on a 64-bit machine.

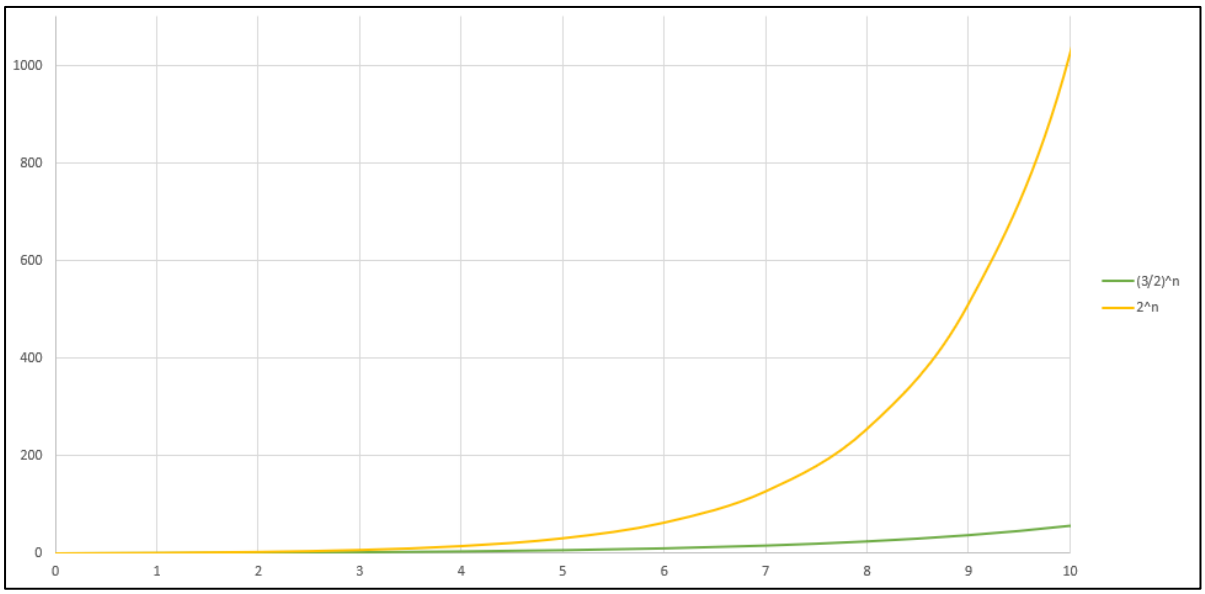
However, n! reaches surpasses this upper bound (at n=21) much faster than 2^n (at n=64).



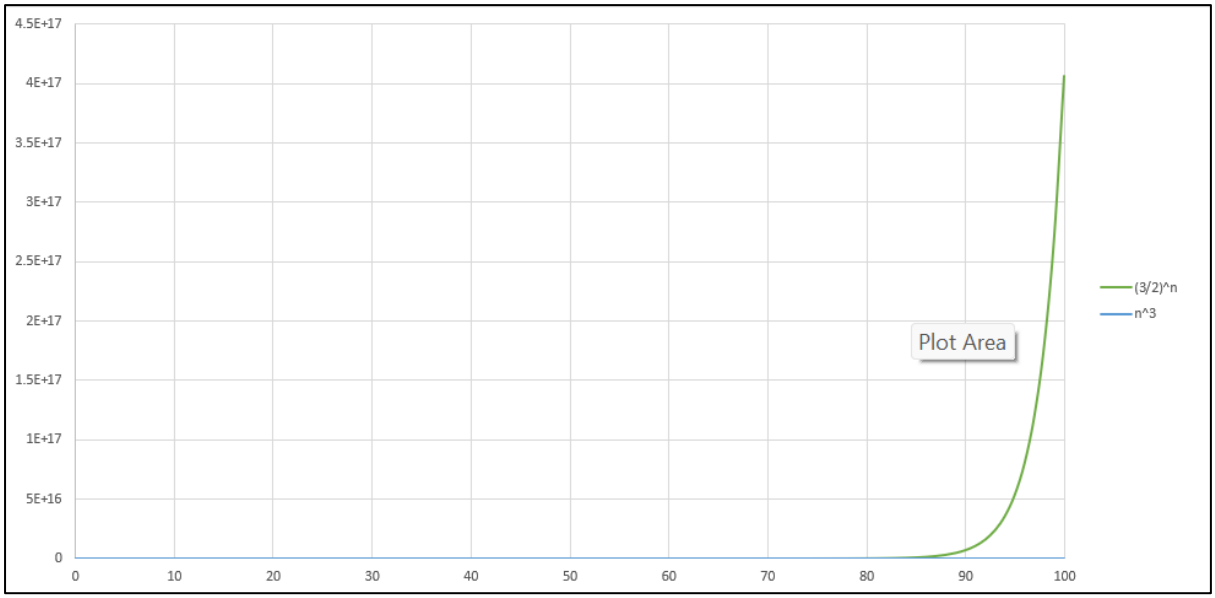
Value of n! is greater than that of 2^n for all n>=4.

In general, factorial function grows much faster than exponential functions with a constant base.

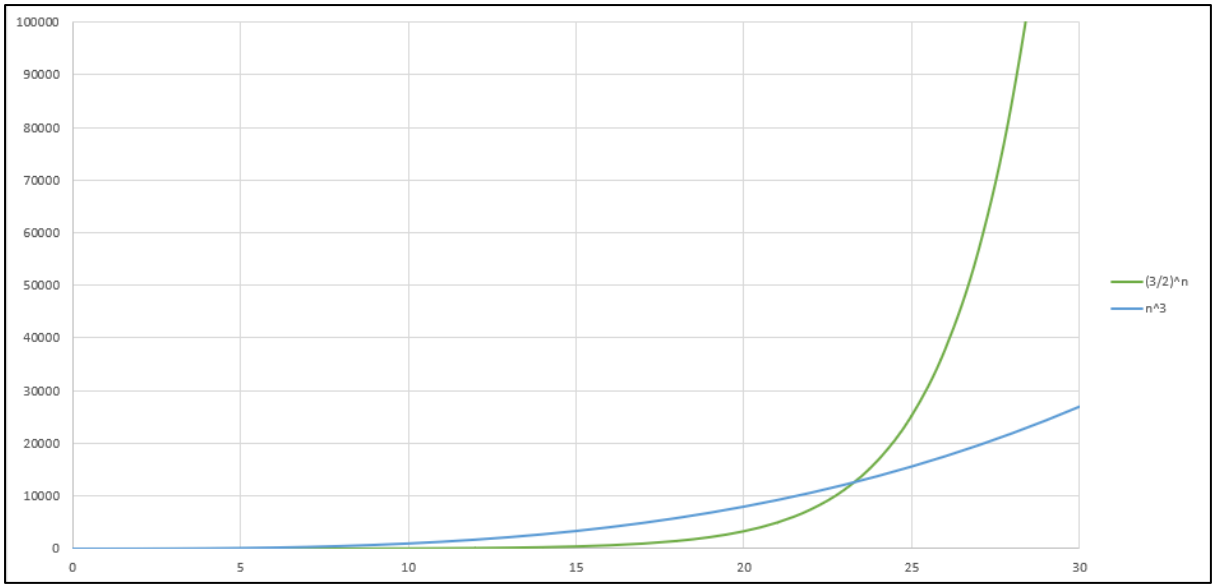
Even though both 1.5^n and 2^n are exponential functions with constant bases with a difference of only 0.5, their growth rates are vastly different. Even for n=10, 2^n value has a greater order of magnitude than 1.5^n.



Comparing exponential and polynomial functions:



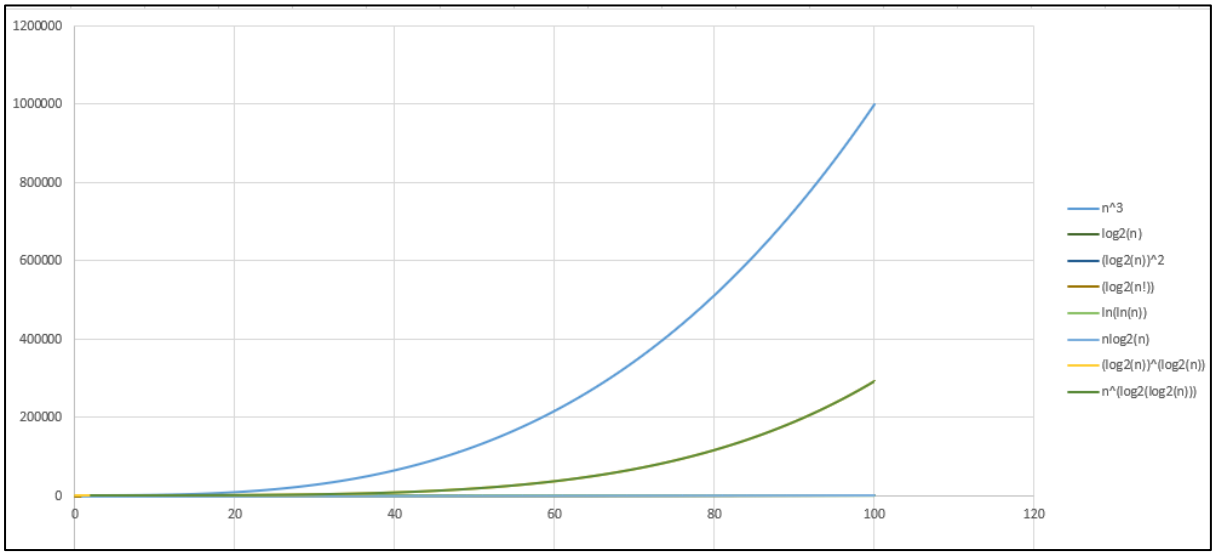
For large values of n, 1.5^n grows much faster than n^3.



However, for n<=23, n^3 is larger than and grows faster than 1.5^n.

In general, exponential functions grow faster than polynomial functions for large values of n.

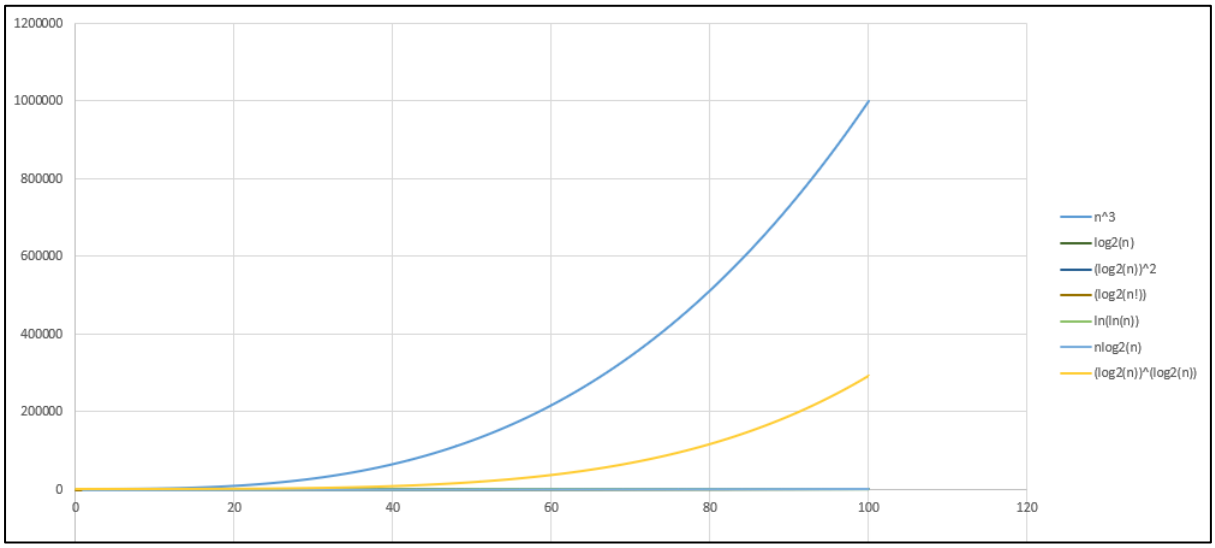
Comparing polynomial and various logarithmic functions:



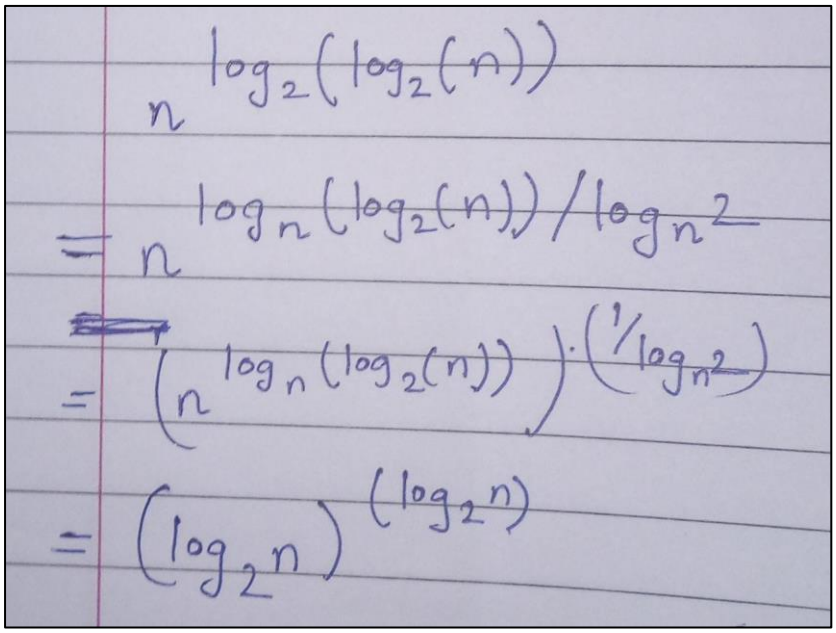
n^3 grows faster than all logarithmic functions on this list.

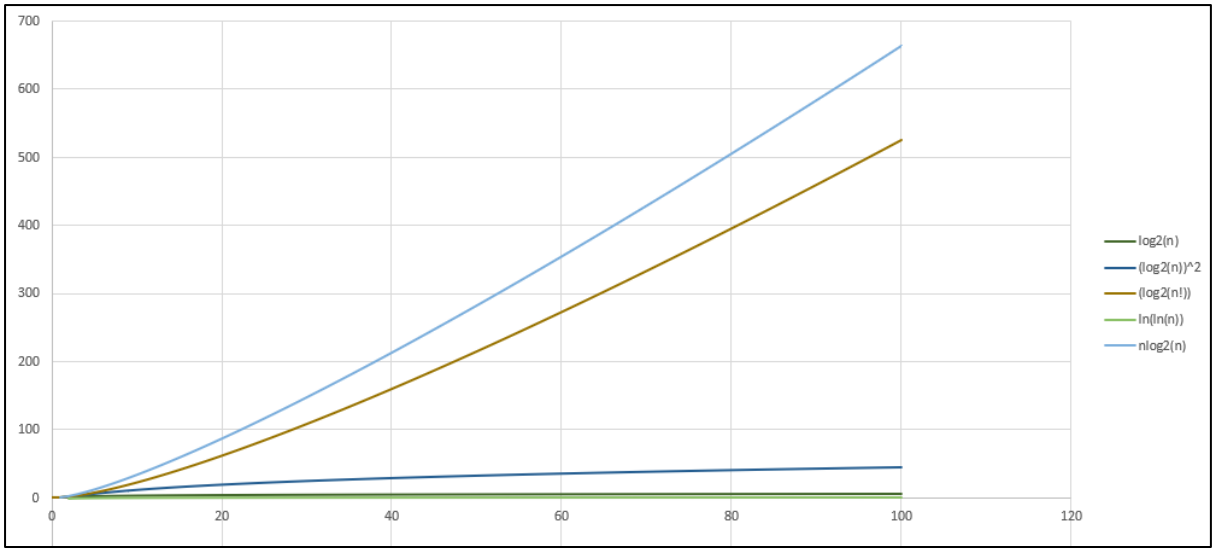
In general, polynomial functions grow faster than logarithmic functions for large values of n.

Among all logarithmic functions, $\log(n)^{(\log(n))}$ and $n^{\log(\log(n))}$ have the highest rate of growth. These two functions are in fact equivalent. Graph of $\log(n)^{(\log(n))}$ is hidden behind $n^{\log(\log(n))}$.

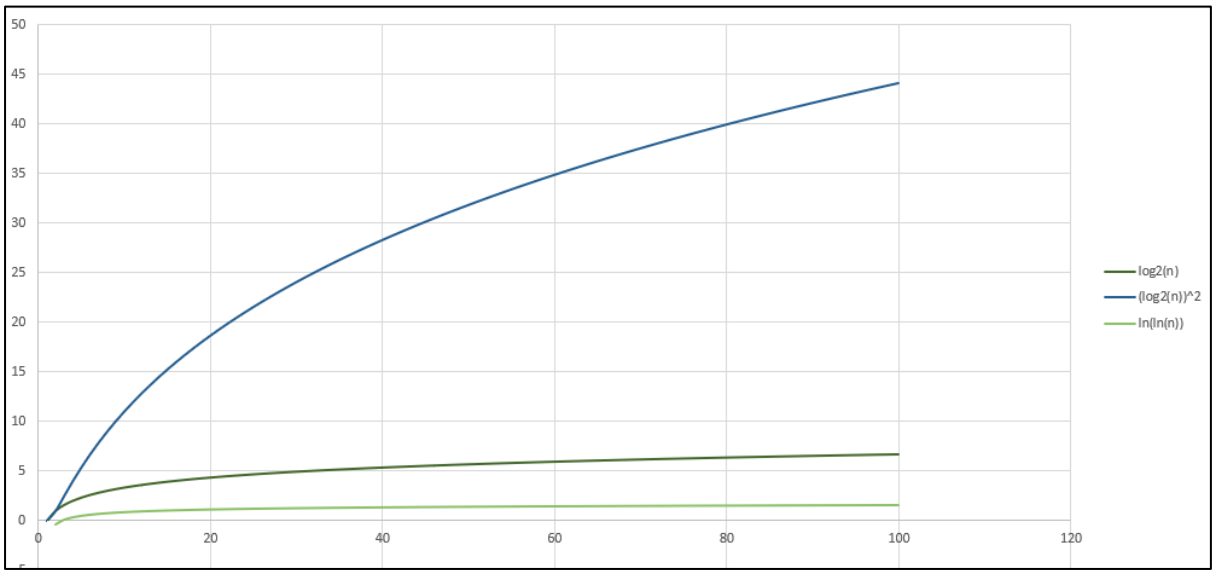


A small mathematical proof of these two functions being equivalent:





n*log(n), having a linear as well as a logarithmic part, grows faster than pure logarithmic functions. Among all pure logarithmic functions, log(n!) grows the fastest as n! has an extremely high growth rate.



$(\log(n))^2$ being the square of log(n), grows faster than log(n). ln(ln(n)) is a double logarithmic function, and hence is the slowest growing function on this list.

CONCLUSION:

Final list of all functions in increasing order of their growth rates:

1. $\ln(\ln(n))$
2. $\log_2(n)$
3. $(\log_2(n))^2$
4. $\log(n!)$
5. $n*\log_2(n)$
6. $n^{\log_2(\log_2(n))}$ OR $\log_2(n)^{\log_2(n)}$
7. n^3
8. $(3/2)^n$
9. 2^n
10. $n!$