Shubhan Singh

SE-Comps B/Batch C

2022300118

DAA Experiment 0

Aim – To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.

Details – A function is a relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output. Let A & B be any two non-empty sets. mapping from A to B will be a function only when every element in set A has one end, only one image in set B.

Problem Definition & Assumptions – For this experiment, you must implement at least 10 functions from the following list.

$(\frac{3}{2})^n$	n^3	$\lg^2 n$	$\lg(n!)$	2^{2^n}	$n^{1/\lg n}$
$\ln \ln n$	$\lg\ n$	$n \cdot 2^n$	$n^{\lg \lg n}$	$\ln n$	$2^{\lg\ n}$
$2^{\lg n}$	$(\lg n)^{\lg n}$	e^n	$(\lg n)!$	$(\sqrt{2})^{\lg n}$	$\sqrt{\lg n}$
$\lg\ (\lg n)$	$2^{\sqrt{2 \lg n}}$	n	2^n	$n \lg n$	$2^{2^{n+1}}$

Note – *lg* denotes for log_2 and *le* denotes log_e

The input (i.e. n) to all the above functions varies from 0 to 100 with increment of 1. Then add the function n! in the list and execute the same for n from 0 to 20.

Source code(C language):

- I have printed the output in a csv file so that the data can be easily imported into excel. For this purpose, the values returned by the functions are not returned immediately, and are instead stored in a 2-D array, so that they can be properly formatted.
- Function pointers have been used to reduce the size of code as they allow to traverse an array of functions pointers in a loop, rather than calling each one separately.
- The 10 chosen functions (apart from factorial) are:
  $n, n^3, \log_2 n, \ln n, log_2{}^2 n, \ln \ln n, n \log_2 n, n^{1/\log_2 n}, 2^n, e^n$

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

long double fastpow(long double a, int b){//faster pow function when index is integer
    long double ans=1,power=a;
    while(b>0){
        if(b%2==1){
            ans*=power;
        }
        power*=power;
        b/=2;
    }
    return ans;
```

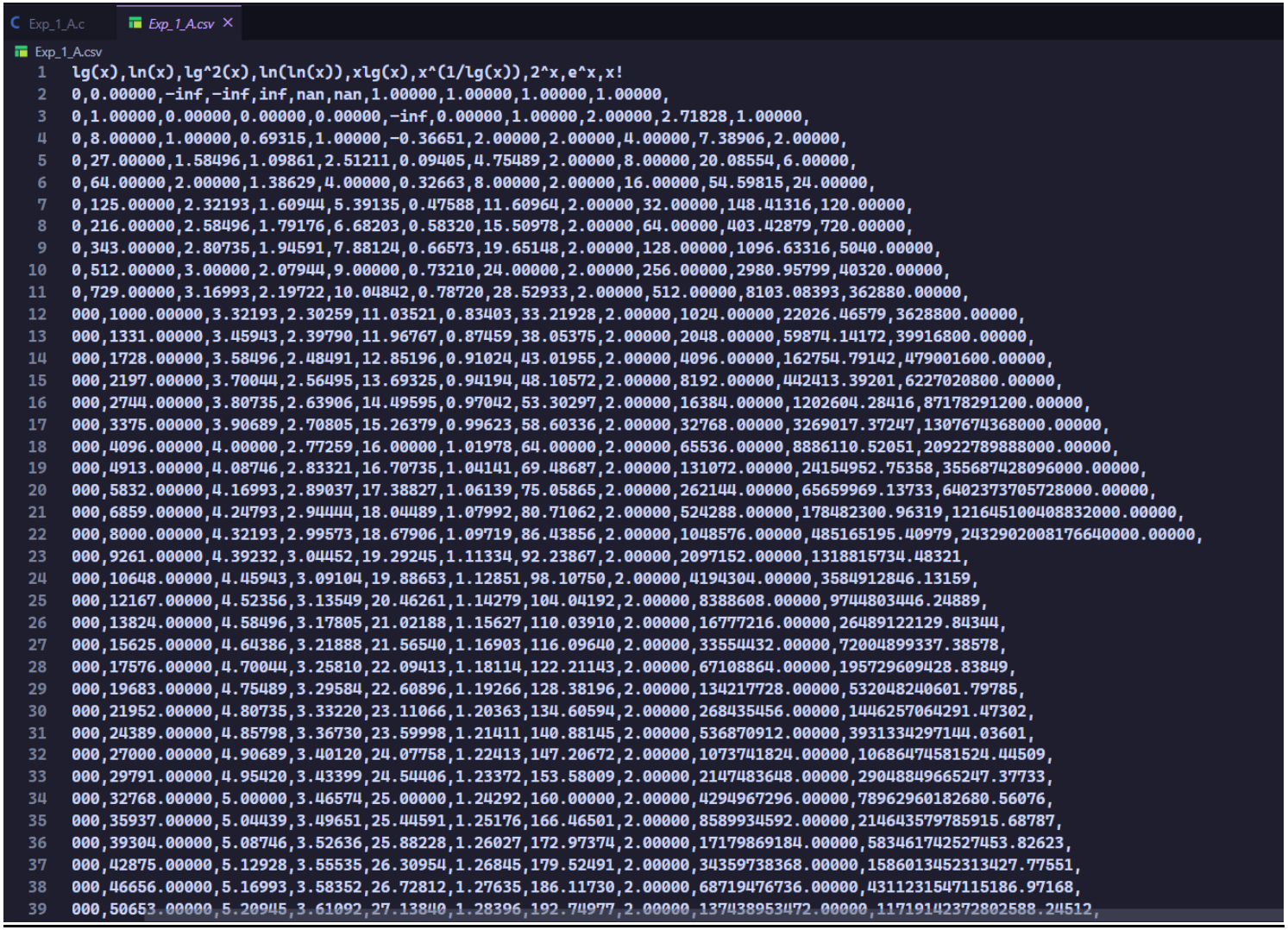
```
}


// all functions have long double return type and all attributes have long double data type
// to allow for use of array of function pointers
long double x(long double x){
    return x;
}

long double x3(long double x){
    return x*x*x;
}

long double lgx(long double x){
    return log2(x);
}

long double lnx(long double x){
    return log(x);
}

long double lg_sq_x(long double x){
    return lgx(x)*lgx(x);
}

long double ln_lnx(long double x){
    return lnx(lnx(x));
```

```
}

long double x_lgx(long double x){
    return x*lgx(x);
}

long double x_pow_1bylgx(long double x){
    return pow(x,pow(lgx(x),-1));
}

long double two_x(long double x){
    return fastpow(2,(int)x);
}

long double e_x(long double x){
    return fastpow(exp(1),(int)x);//using inbuilt function for accurate value of e
}

long double factorial(long double x){
    if(x==0){
        return 1;
    }
    long double ans=1;
    for(int i=x;i>1;i--){
        ans*=i;
    }
    return ans;
```

```c
}

int main(){
    FILE* file=fopen("Exp_1_A.csv","w");//using csv formatting so file is easily opeened in excel
    if(file==NULL){//checking if fopen worked
        printf("Failed to open file\n");
    }

    //i stored all values in an array first so that they can be printed side by side into the file
    long double **arr=malloc(11*sizeof(long double *));//initializing 11 1D arrays to store data
for the 10 functions+factorial
    for(int i=0;i<10;i++){
        arr[i]=malloc(101*sizeof(long double));
    }
    arr[10]=malloc(21*sizeof(long double));//array of size 20 for factorial
    long double (*fnptr_arr[])(long
double)={x,x3,lgx,lnx,lg_sq_x,ln_lnx,x_lgx,x_pow_1bylgx,two_x,e_x,factorial};
    //using an array of function pointers so all functions can be called in a loop
    for(int i=0;i<10;i++){
        for(int n=0;n<=100;n++){
            arr[i][n]=fnptr_arr[i](n);
        }
    }
    for(int n=0;n<=20;n++){
        arr[10][n]=fnptr_arr[10](n);
    }
```

```
    //in the csv file format, contents of each cell are seperated by commas, empty cells are
represented by consecutive
    //commas, and newlines by newline characters
    fprintf(file,"x,x,x^3,lg(x),ln(x),lg^2(x),ln(ln(x)),xlg(x),x^(1/lg(x)),2^x,e^x,x!\n");
    for(int i=0;i<=20;i++){
        fprintf(file,"%d,",i);
        for(int j=0;j<11;j++){
            fprintf(file,"%.5Lf,",arr[j][i]);
        }
        fprintf(file,"\n");
    }
    for(int i=21;i<=100;i++){
        fprintf(file,"%d,",i);
        for(int j=0;j<10;j++){
            fprintf(file,"%.5Lf,",arr[j][i]);
        }
        fprintf(file,"\n");
    }

    fclose(file);
    for(int i=0;i<11;i++){
        free(arr[i]);
    }
    free(arr);
    return 0;
}
```

Output:



File created

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	x	x	x^3	lg(x)	ln(x)	lg^2(x)	ln(ln(x))	xlg(x)	x^(1/lg(x))	2^x	e^x	x!			
2	0	0	0	#NAME?	#NAME?	inf	nan	nan	1	1	1	1			
3	1	1	1	0	0	0	#NAME?	0	1	2	2.71828	1			
4	2	2	8	1	0.69315	1	-0.3665	2	2	4	7.38906	2			
5	3	3	27	1.58496	1.09861	2.51211	0.09405	4.75489	2	8	20.08554	6			
6	4	4	64	2	1.38629	4	0.32663	8	2	16	54.59815	24			
7	5	5	125	2.32193	1.60944	5.39135	0.47588	11.60964	2	32	148.41316	120			
8	6	6	216	2.58496	1.79176	6.68203	0.5832	15.50978	2	64	403.42879	720			
9	7	7	343	2.80735	1.94591	7.88124	0.66573	19.65148	2	128	1096.63316	5040			
10	8	8	512	3	2.07944	9	0.7321	24	2	256	2980.95799	40320			
11	9	9	729	3.16993	2.19722	10.04842	0.7872	28.52933	2	512	8103.08393	362880			
12	10	10	1000	3.32193	2.30259	11.03521	0.83403	33.21928	2	1024	22026.46579	3628800			
13	11	11	1331	3.45943	2.3979	11.96767	0.87459	38.05375	2	2048	59874.14172	39916800			
14	12	12	1728	3.58496	2.48491	12.85196	0.91024	43.01955	2	4096	162754.7914	479001600			
15	13	13	2197	3.70044	2.56495	13.69325	0.94194	48.10572	2	8192	442413.392	6227020800			
16	14	14	2744	3.80735	2.63906	14.49595	0.97042	53.30297	2	16384	1202604.284	87178291200			
17	15	15	3375	3.90689	2.70805	15.26379	0.99623	58.60336	2	32768	3269017.372	1.30767E+12			
18	16	16	4096	4	2.77259	16	1.01978	64	2	65536	8886110.521	2.09228E+13			
19	17	17	4913	4.08746	2.83321	16.70735	1.04141	69.48687	2	131072	24154952.75	3.55687E+14			
20	18	18	5832	4.16993	2.89037	17.38827	1.06139	75.05865	2	262144	65659969.14	6.40237E+15			
21	19	19	6859	4.24793	2.94444	18.04489	1.07992	80.71062	2	524288	178482301	1.21645E+17			
22	20	20	8000	4.32193	2.99573	18.67906	1.09719	86.43856	2	1048576	485165195.4	2.4329E+18			
23	21	21	9261	4.39232	3.04452	19.29245	1.11334	92.23867	2	2097152	1318815734				
24	22	22	10648	4.45943	3.09104	19.88653	1.12851	98.1075	2	4194304	3584912846				
25	23	23	12167	4.52356	3.13549	20.46261	1.14279	104.0419	2	8388608	9744803446				
26	24	24	13824	4.58496	3.17805	21.02188	1.15627	110.0391	2	16777216	26489122130				
27	25	25	15625	4.64386	3.21888	21.5654	1.16903	116.0964	2	33554432	72004899337				
28	26	26	17576	4.70044	3.2581	22.09413	1.18114	122.2114	2	67108864	1.9573E+11				
29	27	27	19683	4.75489	3.29584	22.60896	1.19266	128.382	2	134217728	5.32048E+11				
30	28	28	21952	4.80735	3.3322	23.11066	1.20363	134.6059	2	268435456	1.44626E+12				
31	29	29	24389	4.85798	3.3673	23.59998	1.21411	140.8815	2	536870912	3.93133E+12				
32	30	30	27000	4.90689	3.4012	24.07758	1.22413	147.2067	2	1073741824	1.06865E+13				
33	31	31	29791	4.9542	3.43399	24.54406	1.23372	153.5801	2	2147483648	2.90488E+13				
34	32	32	32768	5	3.46574	25	1.24292	160	2	4294967296	7.8963E+13				
35	33	33	35937	5.04439	3.49651	25.44591	1.25176	166.465	2	8589934592	2.14644E+14				
36	34	34	39304	5.08746	3.52636	25.88228	1.26027	172.9737	2	17179869184	5.83462E+14				

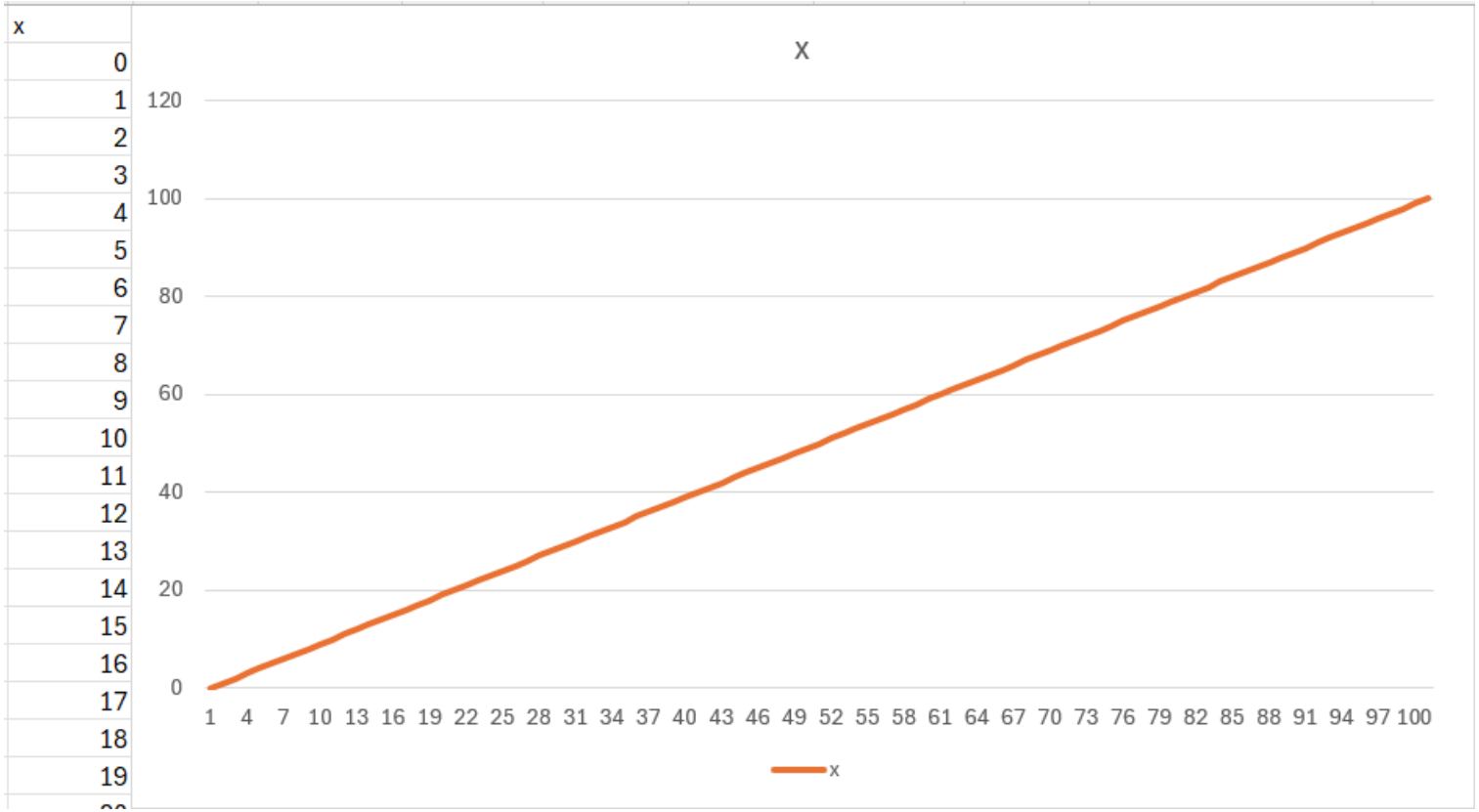
Exp_1_A

(#NAME? refers to -inf)

CSV file

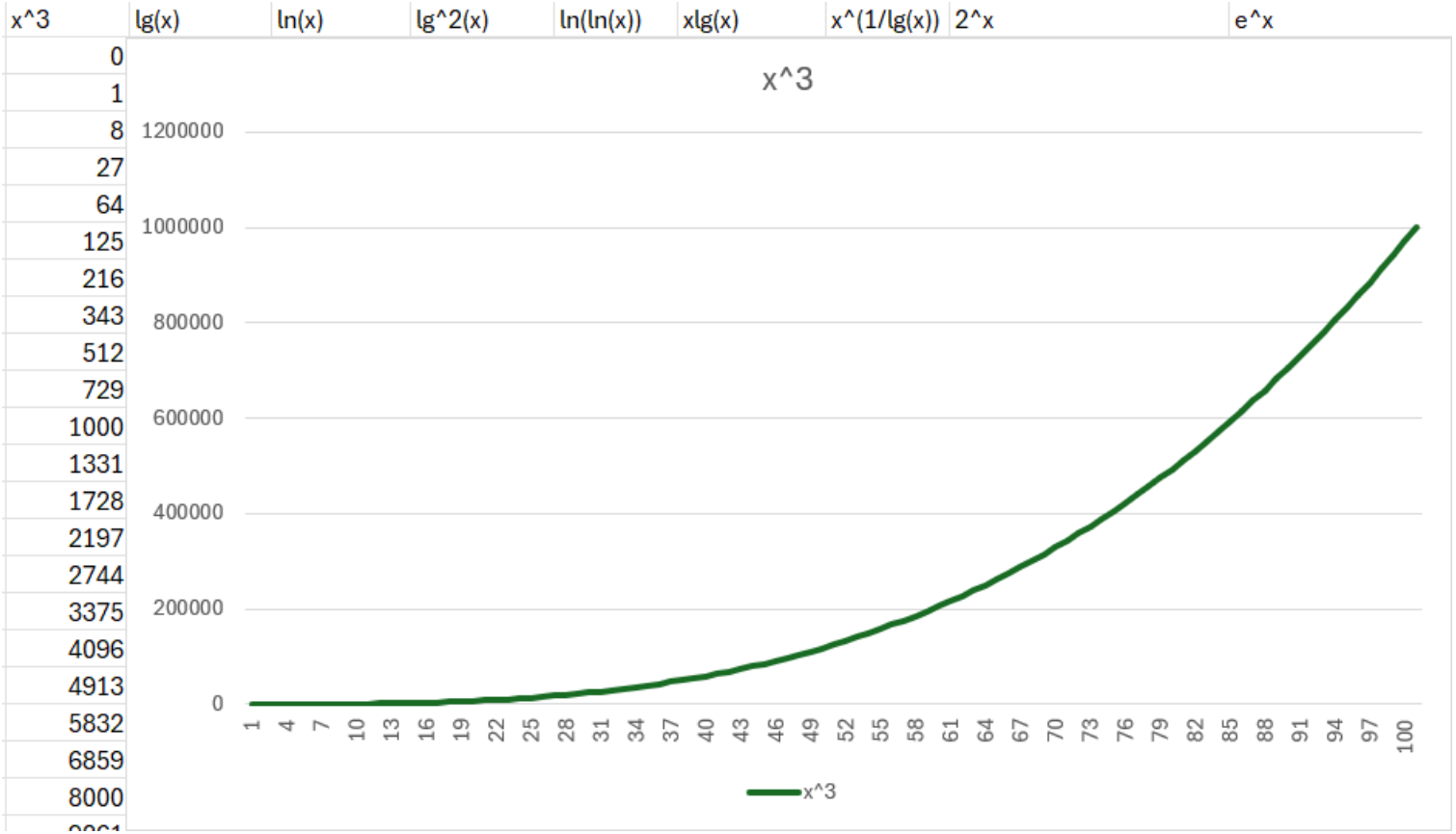
PLOTS:

1. n

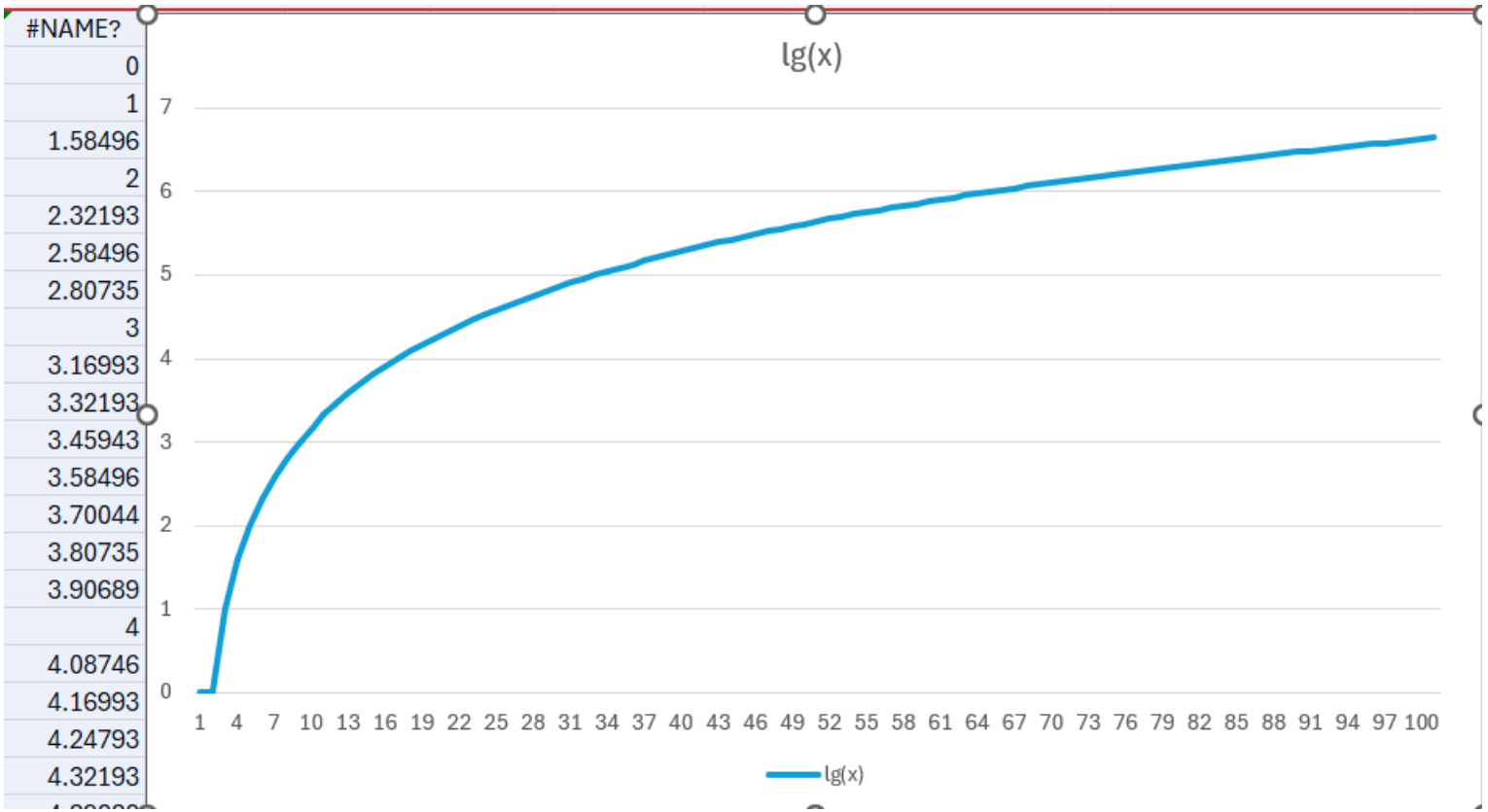

x
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
X
120
100
80
60
40
20
0
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76 79 82 85 88 91 94 97 100
x

2. n^3

x^3	lg(x)	ln(x)	lg^2(x)	ln(ln(x))	xlg(x)	x^(1/lg(x))	2^x	e^x
0								
1								
8								
27								
64								
125								
216								
343								
512								
729								
1000								
1331								
1728								
2197								
2744								
3375								
4096								
4913								
5832								
6859								
8000								



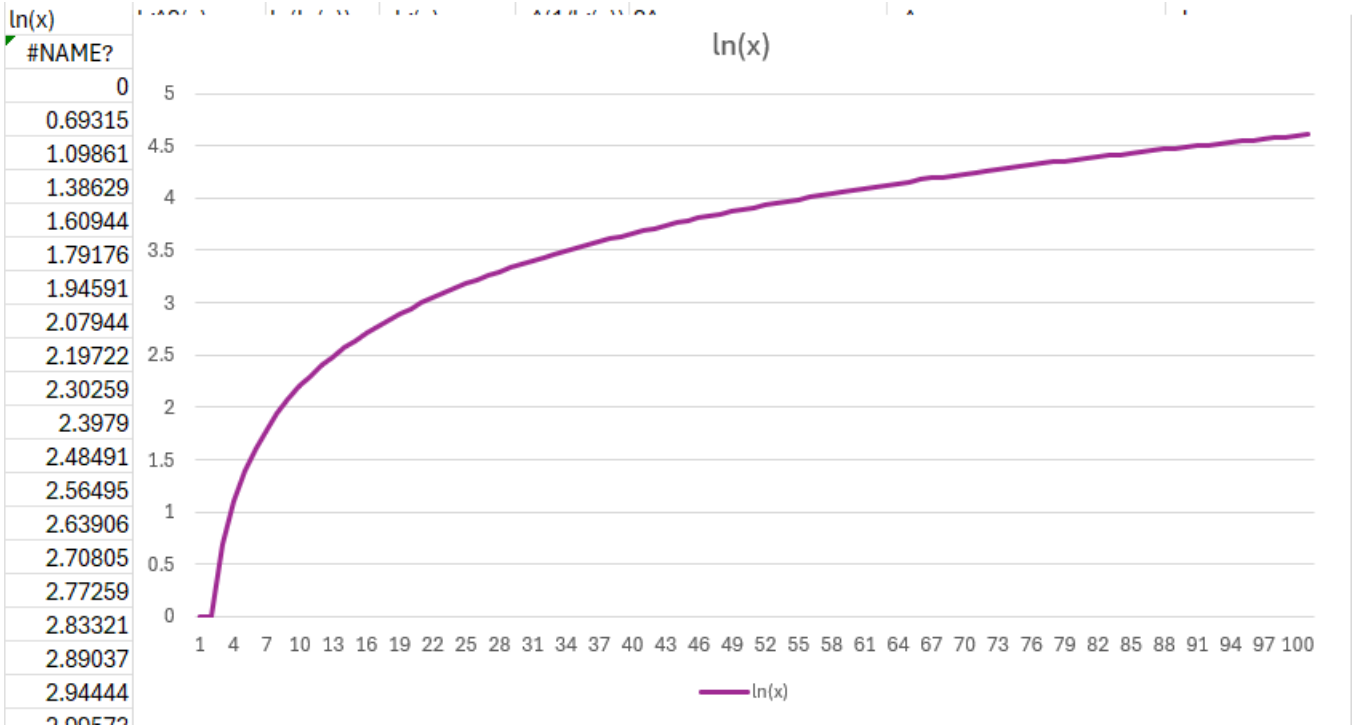
3. $\log_2 n$



(#NAME? refers to -infinity)

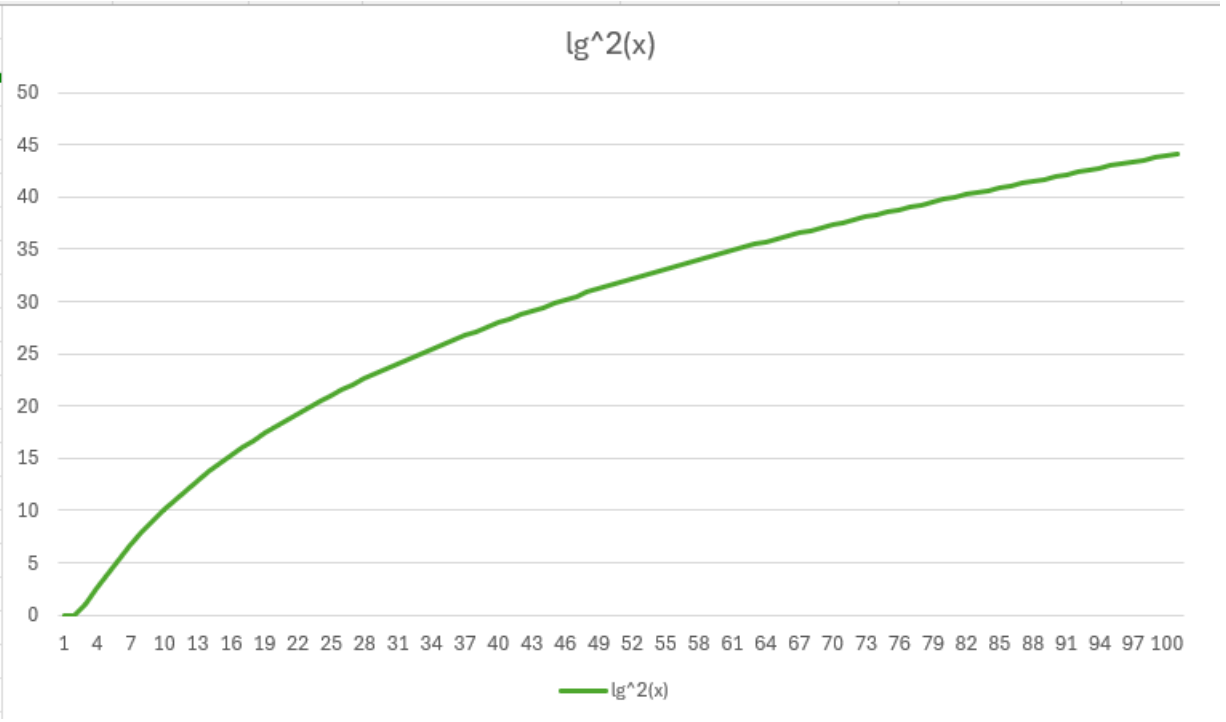
4. $\ln n$

ln(x)
#NAME?
0
0.69315
1.09861
1.38629
1.60944
1.79176
1.94591
2.07944
2.19722
2.30259
2.3979
2.48491
2.56495
2.63906
2.70805
2.77259
2.83321
2.89037
2.94444



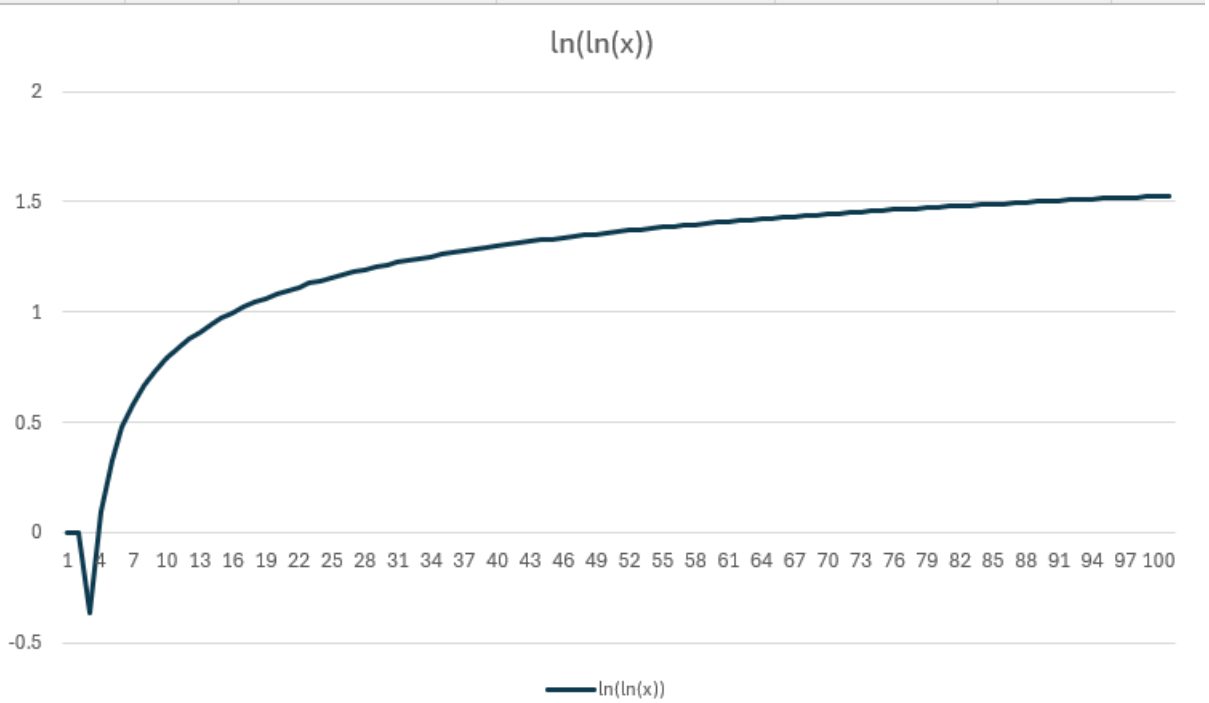
5. $log_2{}^2 n$

lg^2(x)
inf
0
1
2.51211
4
5.39135
6.68203
7.88124
9
10.04842
11.03521
11.96767
12.85196
13.69325
14.49595
15.26379
16
16.70735
17.38827
18.04489



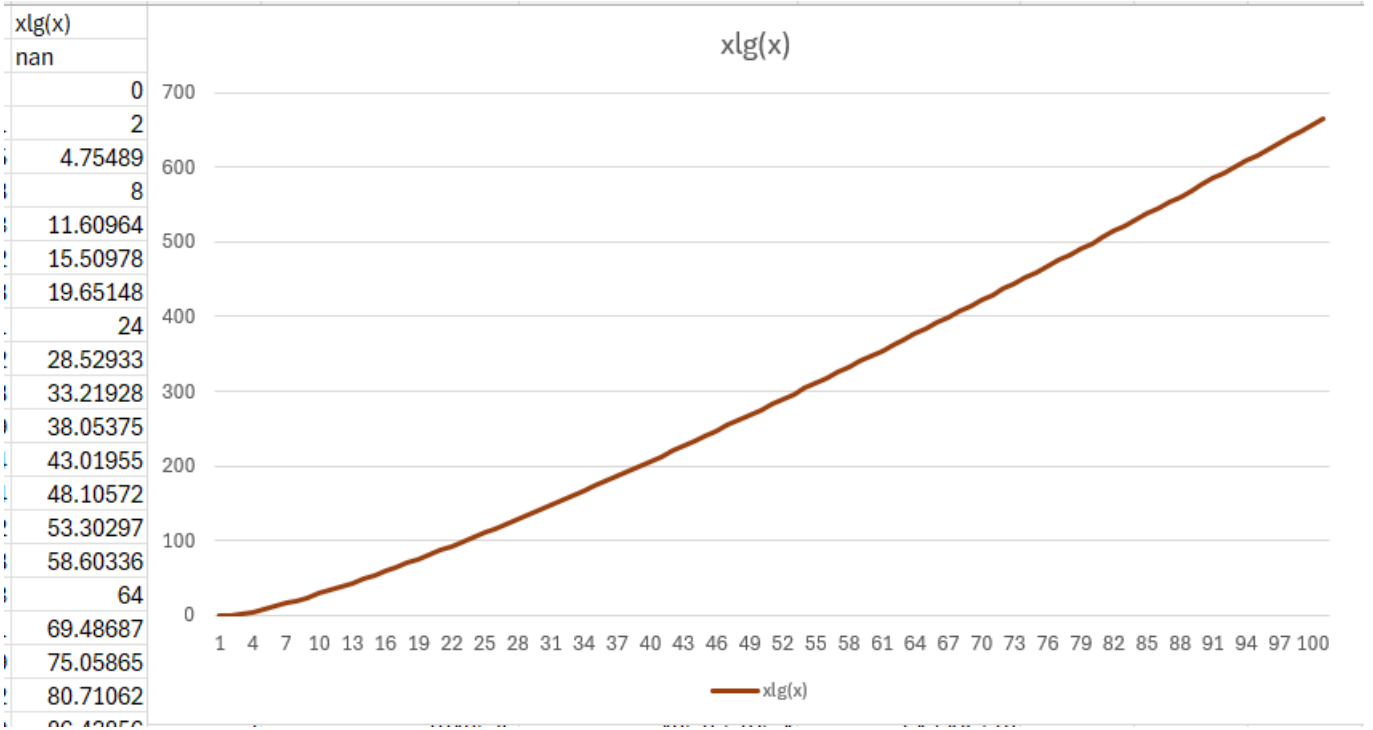
6. $\ln \ln n$

ln(ln(x))
nan
#NAME?
-0.36651
0.09405
0.32663
0.47588
0.5832
0.66573
0.7321
0.7872
0.83403
0.87459
0.91024
0.94194
0.97042
0.99623
1.01978
1.04141
1.06139
1.07992



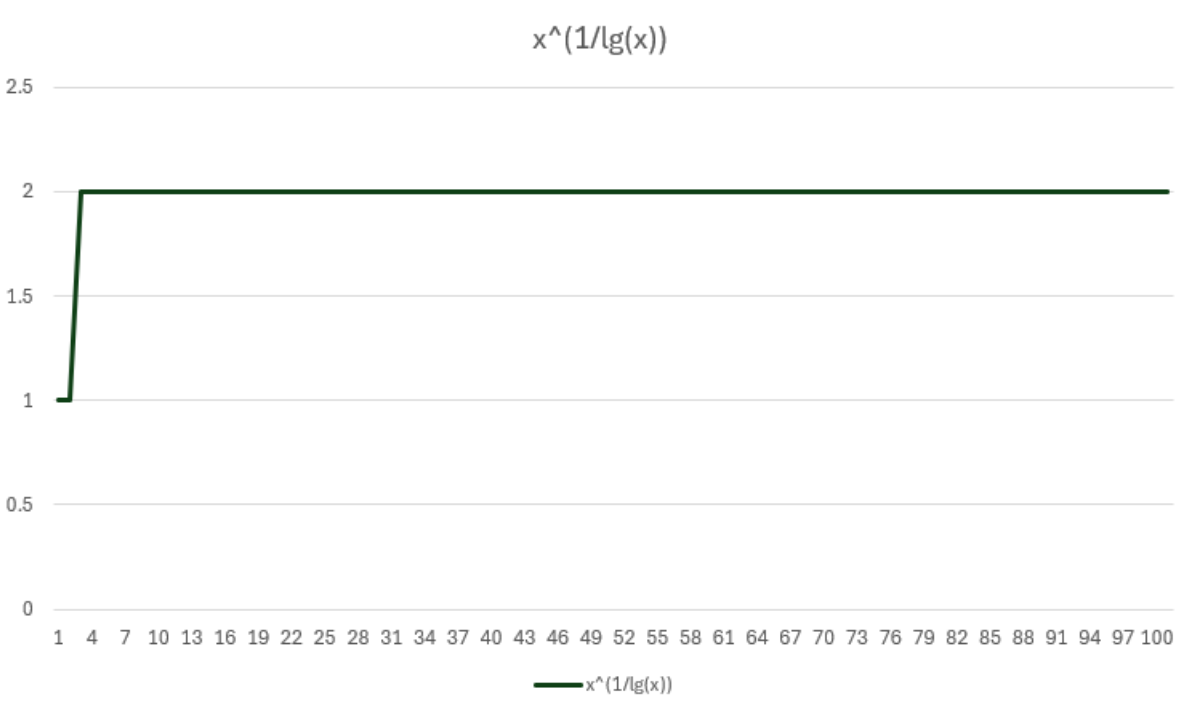
7. $n \log_2 n$

xlg(x)
nan
0
2
4.75489
8
11.60964
15.50978
19.65148
24
28.52933
33.21928
38.05375
43.01955
48.10572
53.30297
58.60336
64
69.48687
75.05865
80.71062



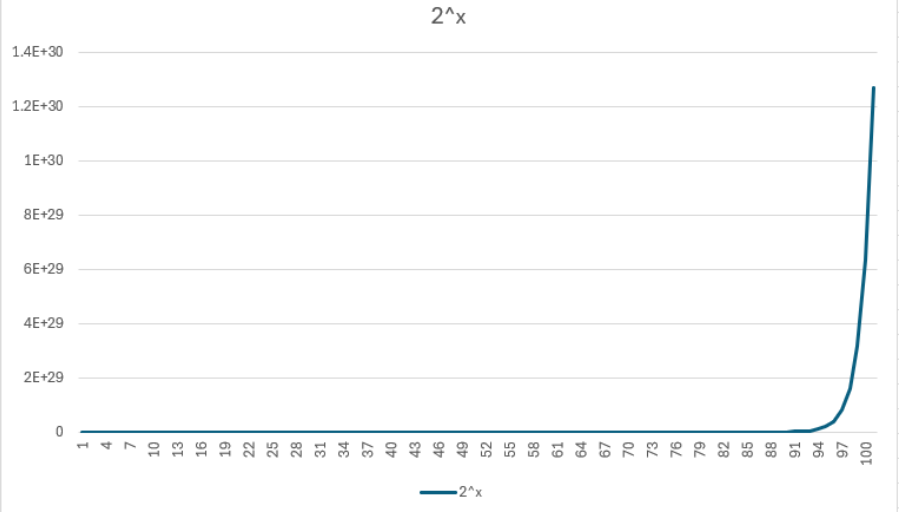
8. $n^{1/\log_2 n}$

x^(1/lg(x))
1
1
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2



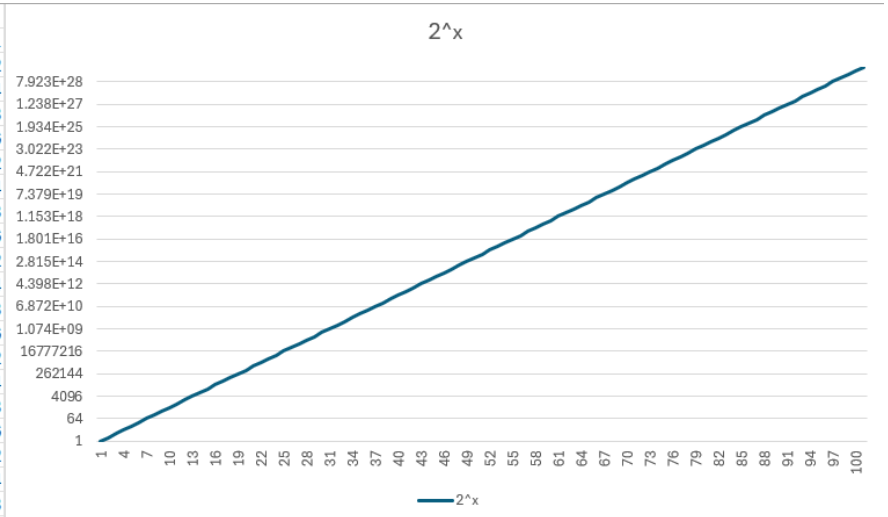
9. 2^n

2^x
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288



(true scale)

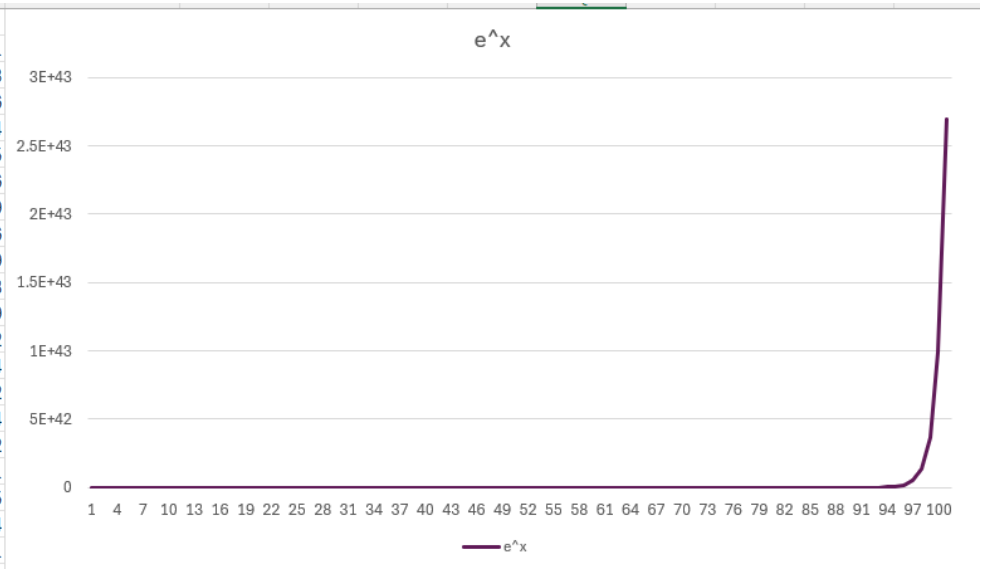
2^x
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288



(Logarithmic scale)

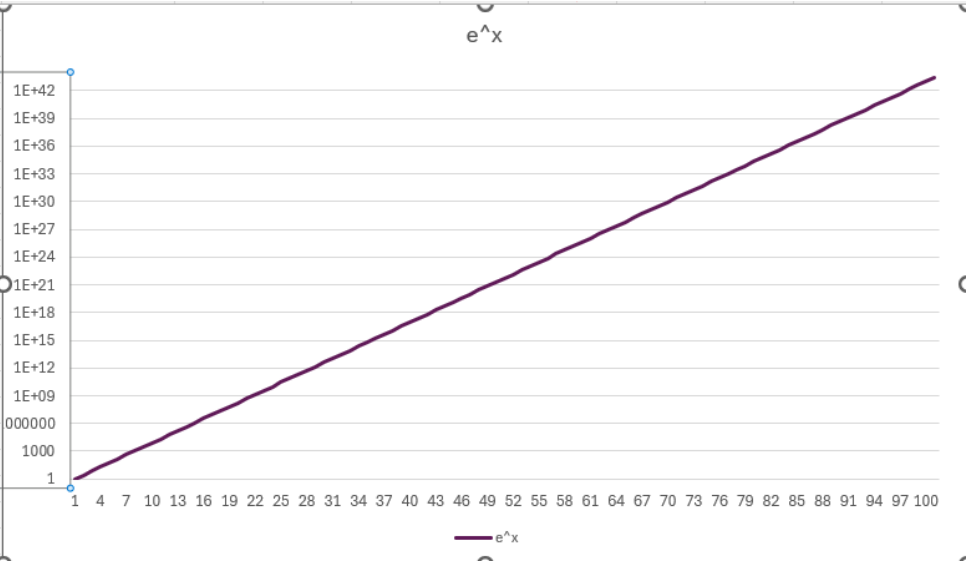
10. e^n

e^x
1
2.71828
7.38906
20.08554
54.59815
148.41316
403.42879
1096.63316
2980.95799
8103.08393
22026.46579
59874.14172
162754.7914
442413.392
1202604.284
3269017.372
8886110.521
24154952.75
65659969.14
178482301



(True scale)

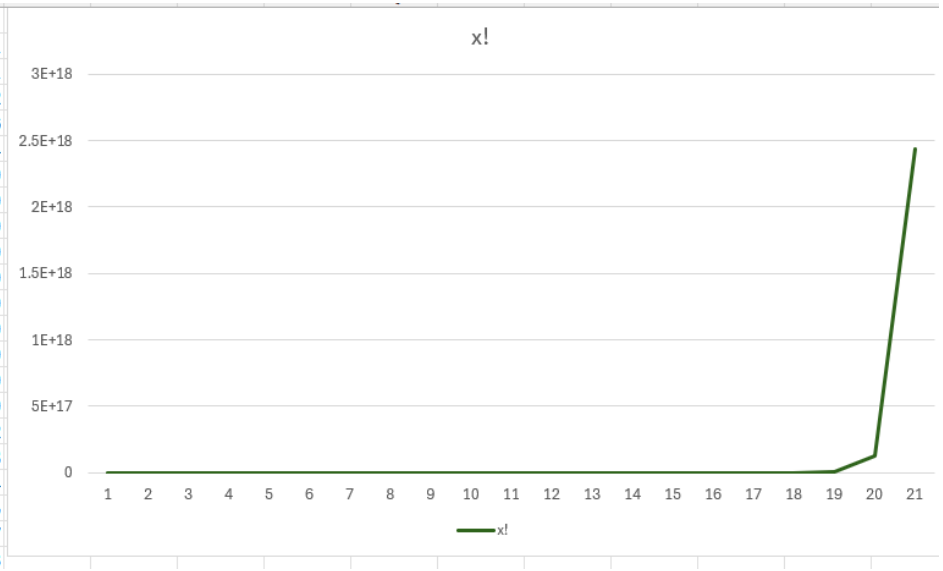
e^x
1
2.71828
7.38906
20.08554
54.59815
148.41316
403.42879
1096.63316
2980.95799
8103.08393
22026.46579
59874.14172
162754.7914
442413.392
1202604.284
3269017.372
8886110.521
24154952.75
65659969.14
178482301



(Logarithmic scale)

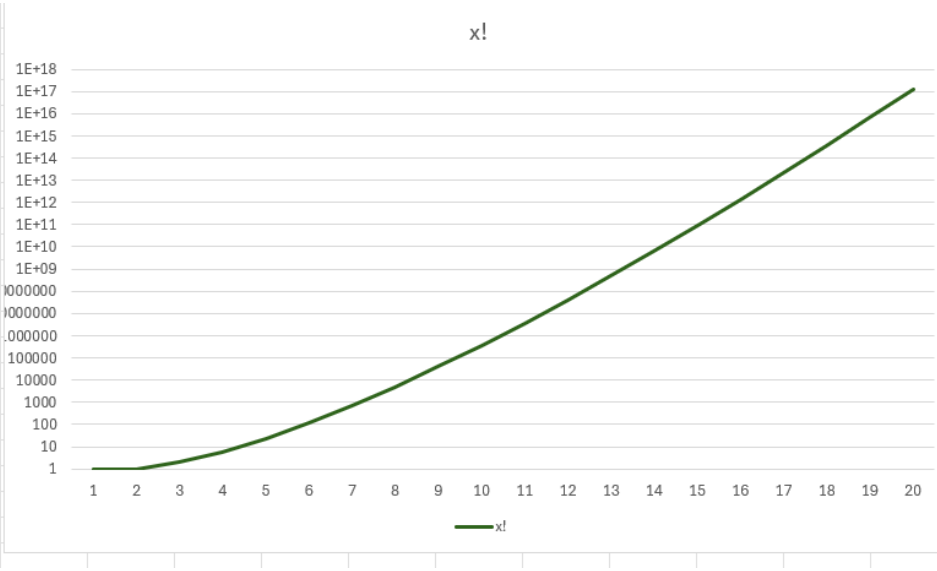
11. n!

x!
1
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1.30767E+12
2.09228E+13
3.55687E+14
6.40237E+15
1.21645E+17
2.4329E+18



(True scale)

x!
1
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1.30767E+12
2.09228E+13
3.55687E+14
6.40237E+15
1.21645E+17
2.4329E+18



(Logarithmic)

Algorithm:

Here's the pseudocode for the functions code:

1. Function fastpow(a, b):
    - Initialize ans to 1 and power to a.
    - While b is greater than 0:
        - If b is odd, multiply ans by power.
        - Square power.
        - Divide b by 2.
    - Return ans.
2. Function x(x):
    - Return x.
3. Function x3(x):
    - Return x cubed.
4. Function lgx(x):
    - Return the base-2 logarithm of x.
5. Function lnx(x):
    - Return the natural logarithm of x.
6. Function lg_sq_x(x):
    - Return the square of the base-2 logarithm of x.
7. Function ln_lnx(x):
    - Return the natural logarithm of the natural logarithm of x.
8. Function x_lgx(x):
    - Return x times the base-2 logarithm of x.
9. Function x_pow_1bylgx(x):
    - Return x raised to the power of the reciprocal of the base-2 logarithm of x.
10. Function two_x(x):
    - Return 2 raised to the power of x.

11. Function e_x(x):
    - Return e (the base of the natural logarithm) raised to the power of x.
12. Function factorial(x):
    - If x is 0, return 1.
    - Initialize ans to 1.
    - For each integer i from x down to 2, multiply ans by i.
    - Return ans.
13. In main():
    - Open a file named "Exp_1_A.csv" for writing.
    - If the file failed to open, print an error message and exit.
    - Allocate memory for an 11-element array of pointers to long double arrays.
    - For each of the first 10 elements of the array, allocate memory for a 101-element long double array.
    - For the 11th element of the array, allocate memory for a 21-element long double array.
    - Initialize an array of function pointers with the functions defined earlier.
    - For each of the first 10 functions and each integer n from 0 to 100, call the function with n as the argument and store the result in the corresponding element of the long double array.
    - For the 11th function and each integer n from 0 to 20, call the function with n as the argument and store the result in the corresponding element of the long double array.
    - Write the headers to the CSV file.
    - For each integer i from 0 to 20, write i and the first 11 elements of the ith row of the long double array to the CSV file.
    - For each integer i from 21 to 100, write i and the first 10 elements of the ith row of the long double array to the CSV file.

- Close the file.
- Free the memory allocated for the long double arrays.
- Free the memory allocated for the array of pointers.

Theory:

1. Function pointers in C:

Function pointers in C are pointers that point to functions, instead of pointing to data like integers, characters, or structures. They can be used to pass functions as parameters to other functions, or to reference a function in a data structure.

2. Long double data type in C:

The long double data type in C is a floating point data type that is more precise than the float and double data types. It is used when higher precision is required.
The size of long double can vary between different platforms and compilers, and varies from 10 to 16 bytes, though it is 10 byles on GNU C compilers. Its type specifier is %Ld.

3. File handling:

- **File Pointer**: In C programming, a file pointer is a special kind of pointer used for interacting with files. It's essentially a pointer to a structure of type FILE that contains information about the file, such as its name, its status (whether it's open or closed), its current position, and more. A file pointer is used as an argument to various file handling functions to perform operations like reading from or writing to a file.

- **fopen**: The fopen function is used to open a file and associate it with a file pointer. It takes two arguments: the name of the file to be opened and the mode in which to open it. The mode can be for reading ("r"), writing ("w"), appending ("a"), and others. If the file is successfully opened, fopen returns a pointer to the FILE structure associated with the file. If the file cannot be opened (for example, if the file does not exist and the mode is "r"), fopen returns NULL.
- **fprintf**: The fprintf function is used to write formatted output to a file. It works similarly to the printf function, but instead of writing to the console, it writes to the file associated with the provided file pointer. The first argument to fprintf is the file pointer, followed by a format string (which can include format specifiers like %d, %s, etc.), and then the values to be formatted and written. The function returns the number of characters written, or a negative value if an error occurs.