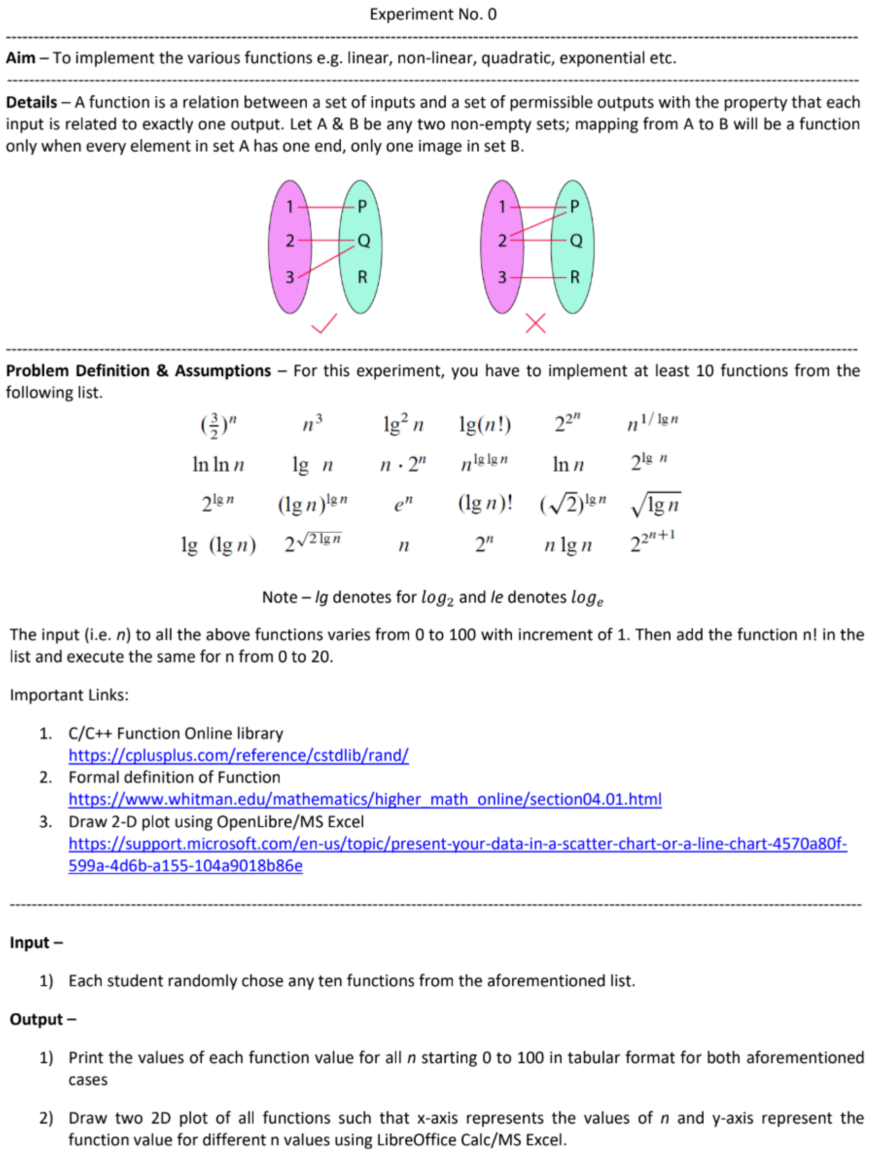
Name	**Advait Ravi Sapkal**
UID No.	**2021700055**
Experiment No.	**1 A**

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

Program 1

PROBLEM STATEMENT :



Experiment No. 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 have to 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.

Important Links:

1. C/C++ Function Online library
   https://cplusplus.com/reference/cstdlib/rand/
2. Formal definition of Function
   https://www.whitman.edu/mathematics/higher_math_online/section04.01.html
3. Draw 2-D plot using OpenLibre/MS Excel
   https://support.microsoft.com/en-us/topic/present-your-data-in-a-scatter-chart-or-a-line-chart-4570a80f-599a-4d6b-a155-104a9018b86e

---

Input –

1) Each student randomly chose any ten functions from the aforementioned list.

Output –

1) Print the values of each function value for all *n* starting 0 to 100 in tabular format for both aforementioned cases

2) Draw two 2D plot of all functions such that x-axis represents the values of *n* and y-axis represent the function value for different n values using LibreOffice Calc/MS Excel.

ALGORITHM:

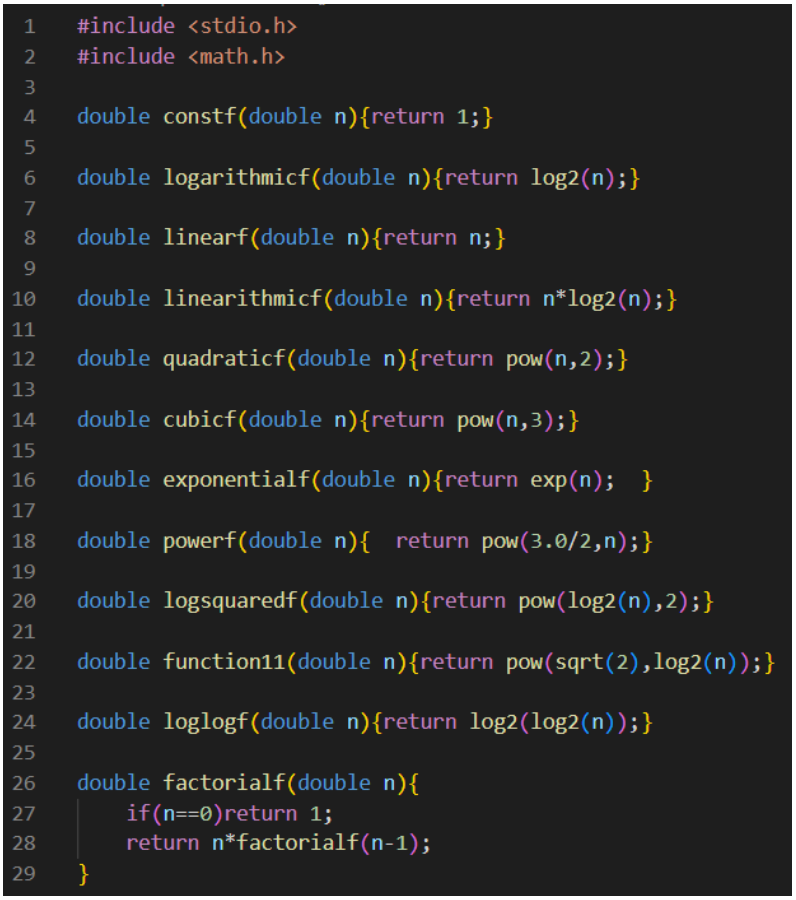
The implemented functions were:

1.	constant :	1
2.	logarithmic:	$lg(n)$
3.	linear:	n
4.	linearithmic:	$nlg(n)$
5.	quadratic:	n^2
6.	cubic:	n^3
7.	factorial:	$n!$
8.	exponential:	e^n
9.	power:	$(3/2)^n - 5000$
10.	squared log:	$lg(n)^2$
11.	function11	$sqrt(2)^{lg(n)}$
12.	log of log:	$lg(lg(n))$

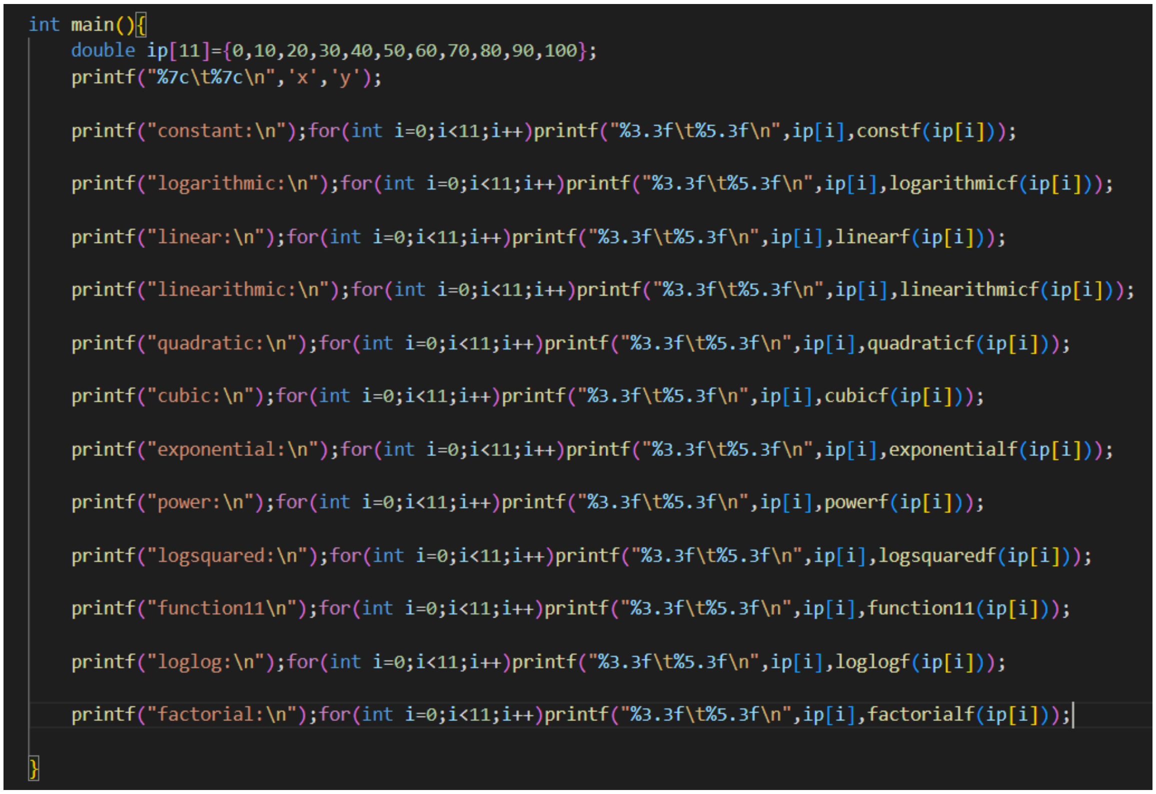
log is taken as lg everywhere

cntd…

PROGRAM:



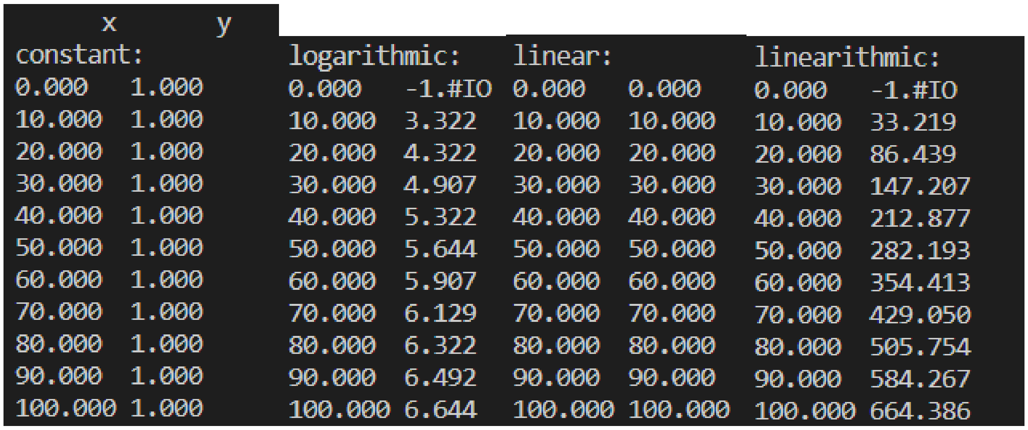
```c
#include <stdio.h>
#include <math.h>

double constf(double n){return 1;}

double logarithmicf(double n){return log2(n);}

double linearf(double n){return n;}

double linearithmicf(double n){return n*log2(n);}

double quadraticf(double n){return pow(n,2);}

double cubicf(double n){return pow(n,3);}

double exponentialf(double n){return exp(n);   }

double powerf(double n){  return pow(3.0/2,n);}

double logsquaredf(double n){return pow(log2(n),2);}

double function11(double n){return pow(sqrt(2),log2(n));}

double loglogf(double n){return log2(log2(n));}

double factorialf(double n){
    if(n==0)return 1;
    return n*factorialf(n-1);
}
```



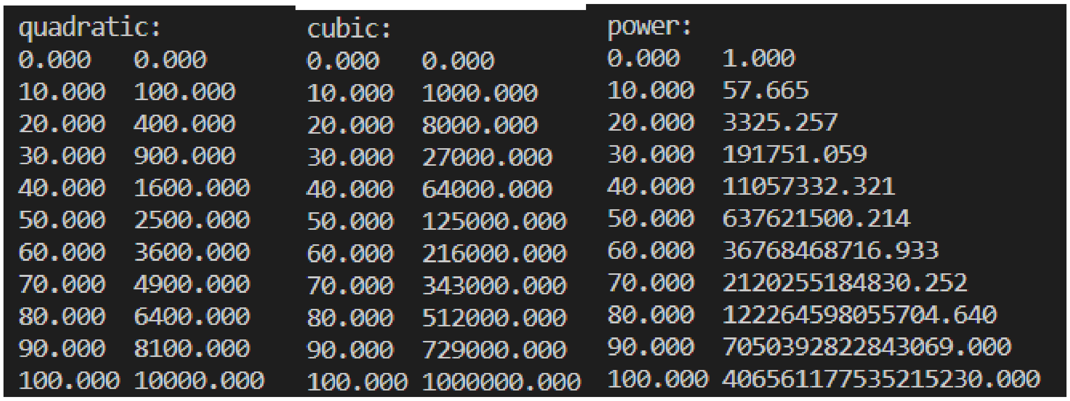
```c
int main(){
    double ip[11]={0,10,20,30,40,50,60,70,80,90,100};
    printf("%7c\t%7c\n",'x','y');

    printf("constant:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],constf(ip[i]));

    printf("logarithmic:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],logarithmicf(ip[i]));

    printf("linear:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],linearf(ip[i]));

    printf("linearithmic:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],linearithmicf(ip[i]));

    printf("quadratic:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],quadraticf(ip[i]));

    printf("cubic:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],cubicf(ip[i]));

    printf("exponential:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],exponentialf(ip[i]));

    printf("power:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],powerf(ip[i]));

    printf("logsquared:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],logsquaredf(ip[i]));

    printf("function11\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],function11(ip[i]));

    printf("loglog:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],loglogf(ip[i]));

    printf("factorial:\n");for(int i=0;i<11;i++)printf("%3.3f\t%5.3f\n",ip[i],factorialf(ip[i]));

}
```

RESULT:

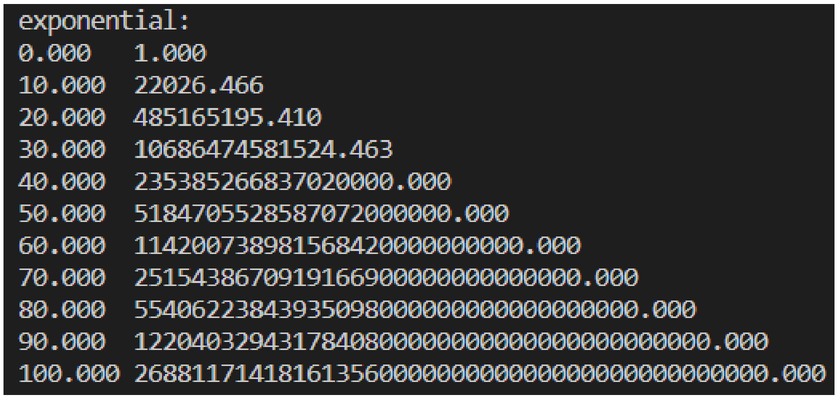
Outputs:



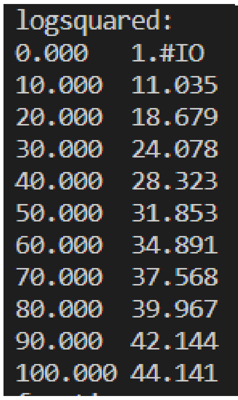
```
       x       y
constant:            logarithmic:      linear:           linearithmic:
0.000    1.000       0.000    -1.#IO   0.000    0.000    0.000    -1.#IO
10.000   1.000       10.000   3.322    10.000   10.000   10.000   33.219
20.000   1.000       20.000   4.322    20.000   20.000   20.000   86.439
30.000   1.000       30.000   4.907    30.000   30.000   30.000   147.207
40.000   1.000       40.000   5.322    40.000   40.000   40.000   212.877
50.000   1.000       50.000   5.644    50.000   50.000   50.000   282.193
60.000   1.000       60.000   5.907    60.000   60.000   60.000   354.413
70.000   1.000       70.000   6.129    70.000   70.000   70.000   429.050
80.000   1.000       80.000   6.322    80.000   80.000   80.000   505.754
90.000   1.000       90.000   6.492    90.000   90.000   90.000   584.267
100.000 1.000        100.000 6.644     100.000 100.000   100.000 664.386
```



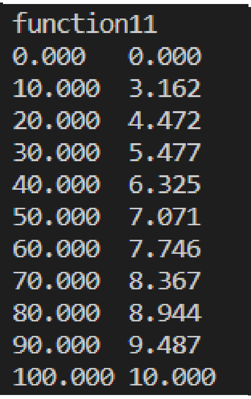
```
quadratic:           cubic:               power:
0.000    0.000       0.000    0.000       0.000    1.000
10.000   100.000     10.000   1000.000    10.000   57.665
20.000   400.000     20.000   8000.000    20.000   3325.257
30.000   900.000     30.000   27000.000   30.000   191751.059
40.000   1600.000    40.000   64000.000   40.000   11057332.321
50.000   2500.000    50.000   125000.000  50.000   637621500.214
60.000   3600.000    60.000   216000.000  60.000   36768468716.933
70.000   4900.000    70.000   343000.000  70.000   2120255184830.252
80.000   6400.000    80.000   512000.000  80.000   122264598055704.640
90.000   8100.000    90.000   729000.000  90.000   7050392822843069.000
100.000 10000.000    100.000 1000000.000  100.000 406561177535215230.000
```



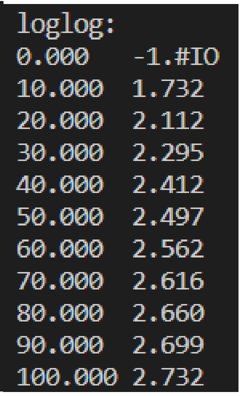
```
exponential:
0.000    1.000
10.000   22026.466
20.000   485165195.410
30.000   10686474581524.463
40.000   235385266837020000.000
50.000   5184705528587072000000.000
60.000   114200738981568420000000000.000
70.000   2515438670919166900000000000000.000
80.000   55406223843935098000000000000000000.000
90.000   1220403294317840800000000000000000000000.000
100.000 26881171418161356000000000000000000000000000.000
```



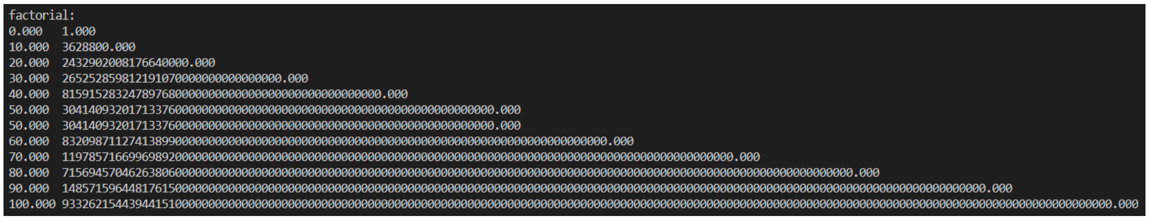
```
logsquared:
0.000    1.#IO
10.000   11.035
20.000   18.679
30.000   24.078
40.000   28.323
50.000   31.853
60.000   34.891
70.000   37.568
80.000   39.967
90.000   42.144
100.000 44.141
```



```
function11
0.000    0.000
10.000   3.162
20.000   4.472
30.000   5.477
40.000   6.325
50.000   7.071
60.000   7.746
70.000   8.367
80.000   8.944
90.000   9.487
100.000 10.000
```



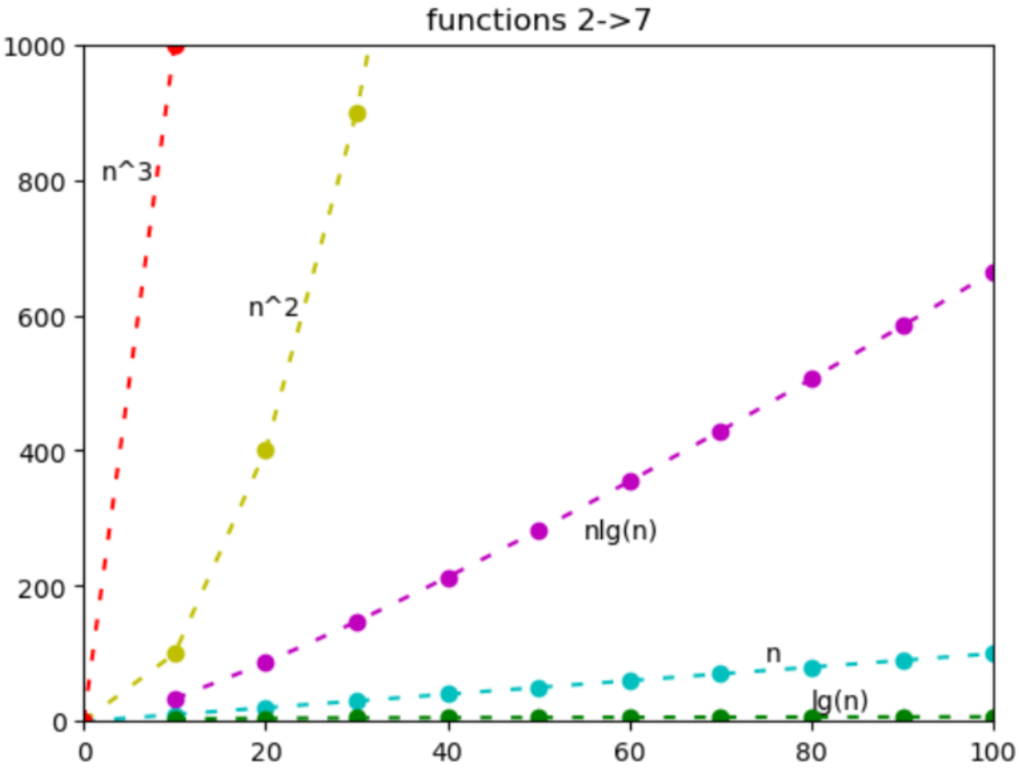
```
loglog:
0.000    -1.#IO
10.000   1.732
20.000   2.112
30.000   2.295
40.000   2.412
50.000   2.497
60.000   2.562
70.000   2.616
80.000   2.660
90.000   2.699
100.000 2.732
```

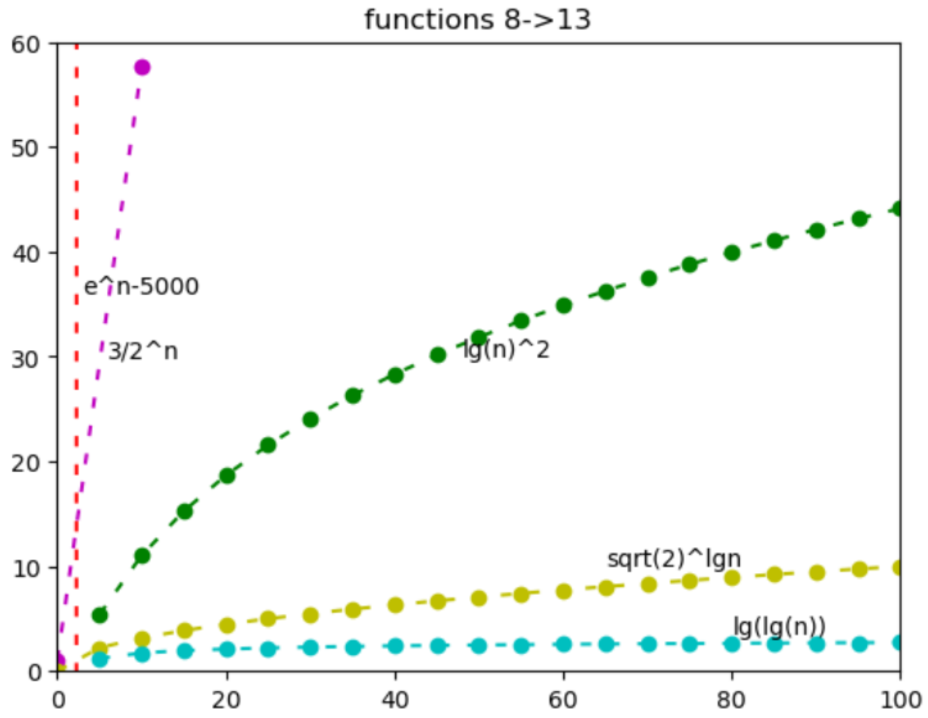


```
factorial:
0.000   1.000
10.000  3628800.000
20.000  2432902008176640000.000
30.000  265252859812191070000000000000000.000
40.000  815915283247897680000000000000000000000000000000.000
50.000  30414093201713376000000000000000000000000000000000000000000000000.000
50.000  30414093201713376000000000000000000000000000000000000000000000000.000
60.000  8320987112741389900000000000000000000000000000000000000000000000000000000000000000.000
70.000  11978571669969892000000000000000000000000000000000000000000000000000000000000000000000000000000000000.000
80.000  71569457046263806000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.000
90.000  1485715964481761500000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.000
100.000 93326215443944151000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.000
```

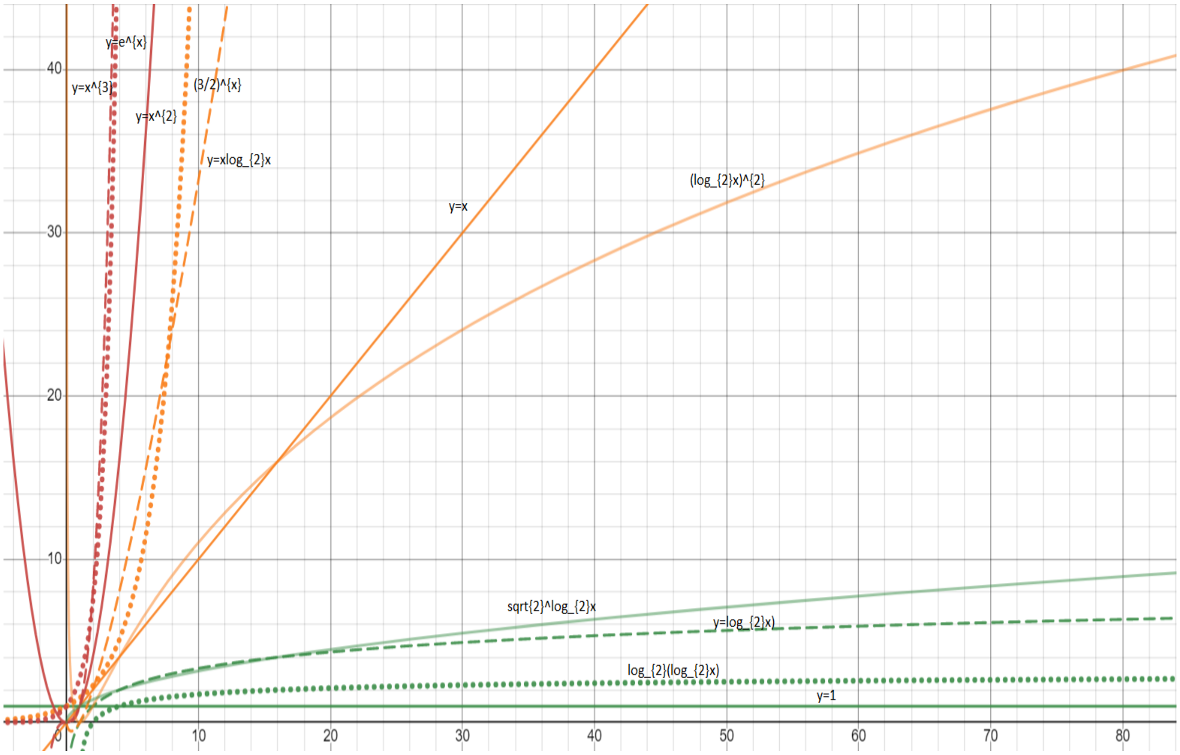
Plots:

Final plots of results: y = f(x)





For all functions: y = f(x)



cntd…

Conclusions:

1. Overall, y = k has the least rate of increase, which is zero.
2. Other functions having an extremely low rate of increase are: lg(lg(n)) , $\text{sqrt}(2)^{\lg(n)}$ and lg(n)
3. y = x has constant slope.
4. Any positive power/exponential function increases steeply with x.
5. e^x has the highest slope, followed by x^3 (considering significant values of x)
6. $y = \lg(x)^2$ has very weird behavior, going to an extremely high slope for x below 1, but declining in slope even more than y = x when x is greater than 1.
7. y = x! of course is the function with the overall highest steepness, with anything greater than 12! being impossible to compute. (For 4-byte int)
8. log functions are not defined for 0.