NAME :	SHIVAM KAMBLE
UID :	2022301007
EXPERIMENT NO :	01 A
AIM :	To implement the various functions e.g. linear, non-linear, quadratic, exponential etc.

ALGORITHM : Step 1: Start.

Step 2: Declare the variables which are required to perform operations on the functions.

Step 3: Start the loop which starts from 0 th number to 100 th number.

Step 4: i. perform the operation: 3/2^n using pow() ii. Print the result.

Step 5: i. Perform the operation: n^3 using simple multiplication ii. Print the result.

Step 6: i. Perform the operation: n.lg(n) using in built log function in math.h ii. Print the result.

Step 7: i. Perform the operation: lg(n) using in built log function in math.h ii. Print the result.

Step 8: i. Perform the operation: 2^lg(n) using in built log function in math.h and pow() ii. Print the result.

Step 9: i. Perform the operation lg(lg(n)) using in built log function in math.h ii. Print the result.

Step 10: i. Perform the operation: lg(n)^2 using in built log function in math.h and pow() ii. Print the result.

Step 11: i. Perform the operation: n ii. Print the result.

Step 12: i. Perform the operation: 2^n using pow() ii. Print the result.

Step 13: i. Perform the operation: n.2^n using pow() ii. Print the result.

Step 14: End the loop

Step 15: End.

Algorithm for Factorial of numbers from 1 to 20:

Step 1: Start.

Step 2: Declare the variables n, fact

Step 3: Initialize the values n = 20 and fact = 1.

Step 4: Start the loop from 1 to n

Step 5: calculate, fact = fact *I

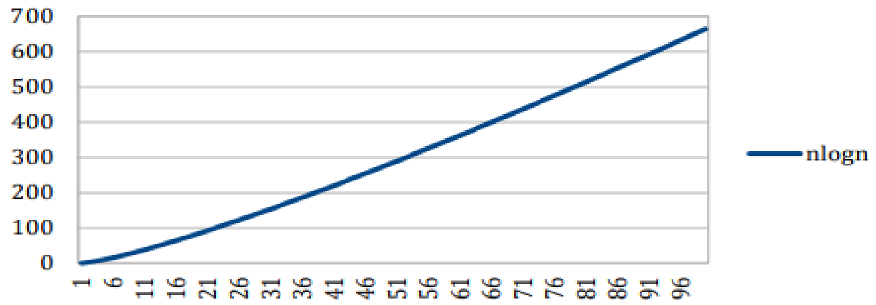
Step 6: print the value of fact

Step 7: End.

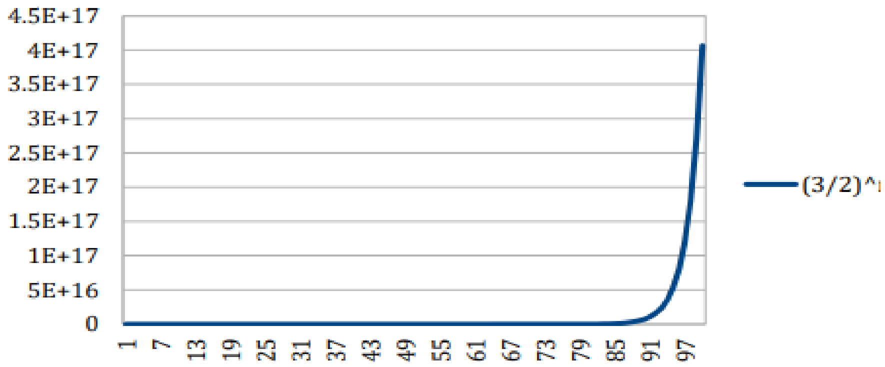
PROGRAM:

```
#include <stdio.h>
#include <math.h>
int main()
{
int i;
long double a,b,c,d,g,k;
float e,f,h;
for(i = 1;i<=100;i++){
a = pow(1.5,i);
printf("%f\n",a);
b = i * i * i;
printf("%f\n",b);
printf("%d\n",i);
c = pow(2,i);
printf("%f\n",c);
d = i*log2(i);
printf("%Lf\n",d);
e = pow(2,log2(i));
printf("%f\n",e);
f = log2(i);
printf("%f\n",f);
g = i*pow(2,i);
printf("%Lf\n",g);
h = log10(log10(i));
printf("%f\n",h);
k = pow(log2(i),2);
printf("%Lf\n",k);
}
return 0;
}
#include <stdio.h>
void fact(int num){
int i;
long f=1;
for(i=1;i<=num;i++) f=f*i;
printf("%d %ld\n",num,f);
}
int main(){ int i;
 for(i = 1; i<=20;i++){ fact(i);
}
return 0;
}
```

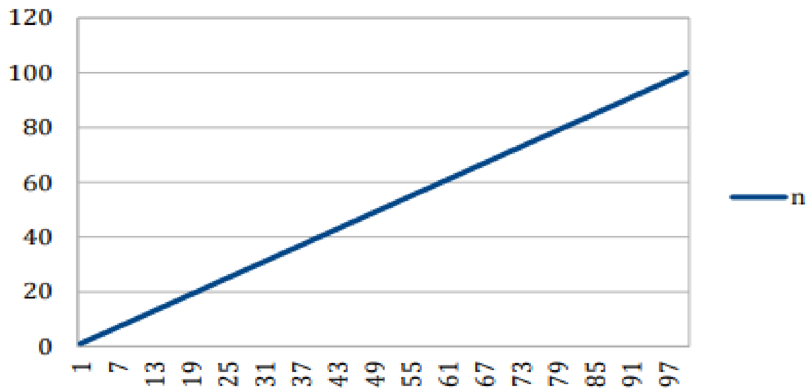
Graph:



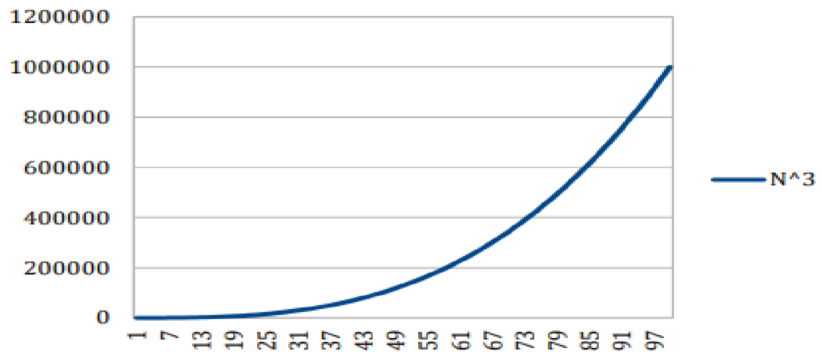
OBSERVATION : It is observed that there is a slow increase between 0 and 1 and after that there is a uniform increase till the 100th value.



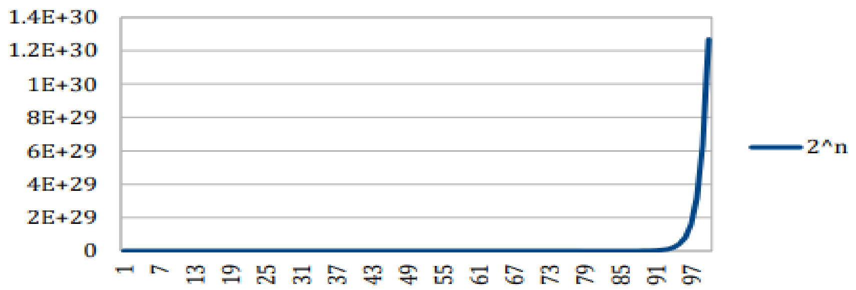
OBSERVATION : The graph doesn't increase for a few iterations and then suddenly rises to millions .



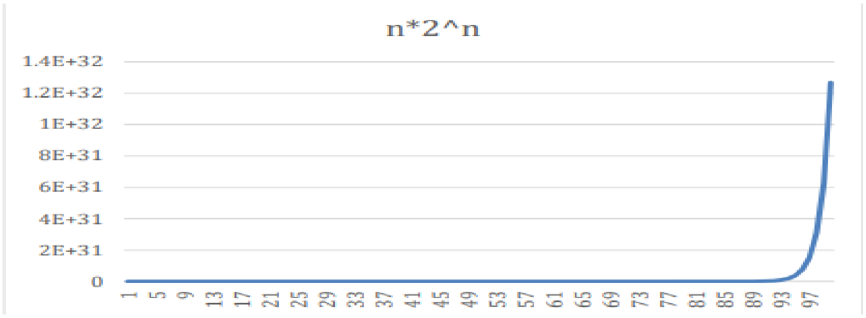
OBSERVATION : The graph increases linearly from 1 to 100 .



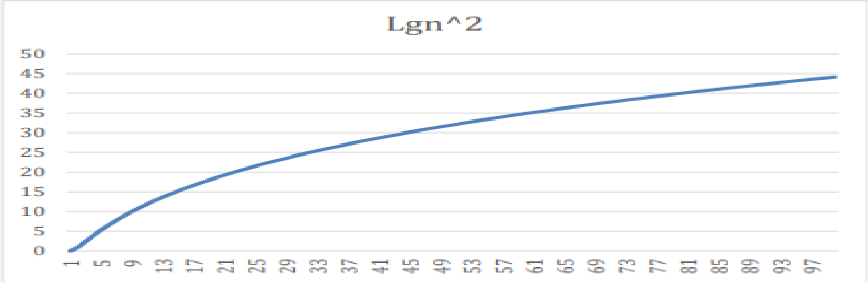
OBSERVATION : For n^3 there is gradual slope and afterwards it increases rapidly .



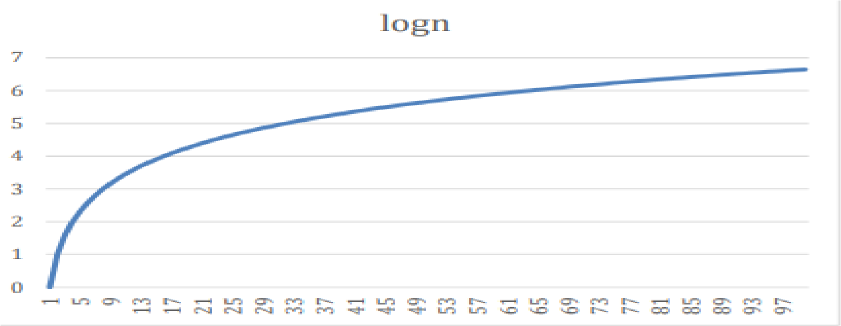
OBSERVATION : For 2^n the graph doesn't have a rise for few iterations and then it suddenly rises to millions .



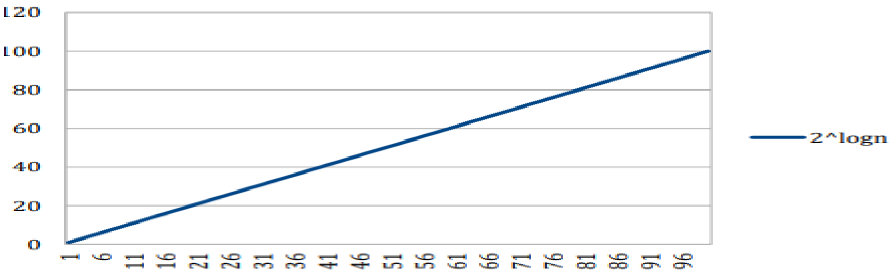
OBSERVATION : For n*2^n the graph doesn't have a rise for few iterations and then it suddenly rises to millions .



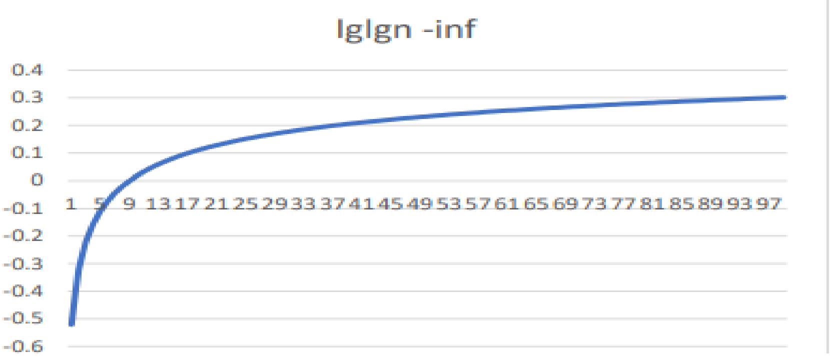
<u>OBSERVATION :</u> We can see a gradual slope according to the graph .



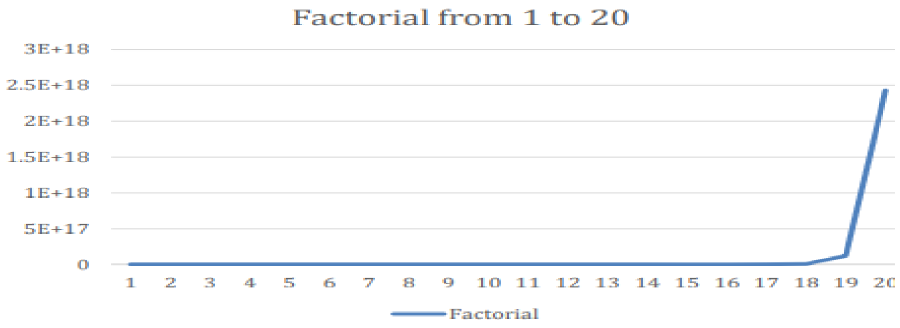
<u>OBSERVATION :</u> We can see a slope where the angle of slop is gradually decreasing lesser than lgn^2 .



<u>OBSERVATION :</u> The graph is linear showing linear increase in the values .



OBSERVATION : Since log(log(1)) is -infinity the graph starts from the negative quadrant and goes up after which we observe normal logarithmic graph behaviour which is linear increase over a few values followed by slow increase.



OBSERVATION : The Graph doesn’t increase at the start but increases towards the end rapidly .

Conclusion: Thus, after running 10 functions on numbers from 1 to 100 We conclude that functions with power increase rapidly after some slow growth and functions with log have linear increase for some values after which we observe slow growth.