Experiment 1a

NAME : ROHIT SHEKHAR BAVISKAR

UID : 2021700004

BATCH : CSE DS D1

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

PROGRAM :

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

void fun1(int n)
{ // n cube
   int res = n * n * n;
   printf("%d\t", res);
}

void fun2(int n)
{ // log n
   double res = log(n);
   printf("%.3f\t", res);
}

void fun3(int n)
{ // nlogn
   double res = n * log(n);
   printf("%.3f\t", res);
}

void fun4(int n)
{ // 2^logn

   double res = pow(2, log(n));
   printf("%.3f\t", res);
}

void fun5(int n)
{
   double res = pow(log2(n), 0.5); // sqrt(log2n)
   printf("%.3f\t", res);
}

void fun6(int n)
{ // log2 n
   double res = log2(n);
   printf("%.3f\t", res);
}

void fun7(int n)
{ // ln ln n
   double res = log(log(n));
   printf("%.3f\t", res);
}
```

```
void fun8(int n)
{ // (root 2)^ log2n
   double res = pow(1.4142, log2(n));
   printf("%.3f\t", res);
}

void fun9(int n)
{ // n^ (lglgn)
   double res = pow(2, 2 * log2(n));
   printf("%.2f\t", res);
}

void fun10(int n)
{ // log2 n
   double res = pow(log2(n), 2);
   printf("%.3f\t", res);
}
int main()
{
   printf("n\t");
   printf("fun1\t");
   printf("fun2\t");
   printf("fun3\t");
   printf("fun4\t");
   printf("fun5\t");
   printf("fun6\t");
   printf("fun7\t");
   printf("fun8\t");
   printf("fun9\t");
   printf("fun10\t");
   printf("\n");
   for (int i = 0; i < 100; i++)
   {
      printf("%d\t", i);
      fun1(i);
      fun2(i);
      fun3(i);
      fun4(i);
      fun5(i);
      fun6(i);
      fun7(i);
      fun8(i);
      fun9(i);
      fun10(i);
      printf("\n");
   }

   return 0;
}
```

Theory :

Time Complexity: The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.

The valid algorithm takes a finite amount of time for execution. The time required by the algorithm to solve given problem is called ***time complexity*** of the algorithm. Time complexity is very useful measure in algorithm analysis.

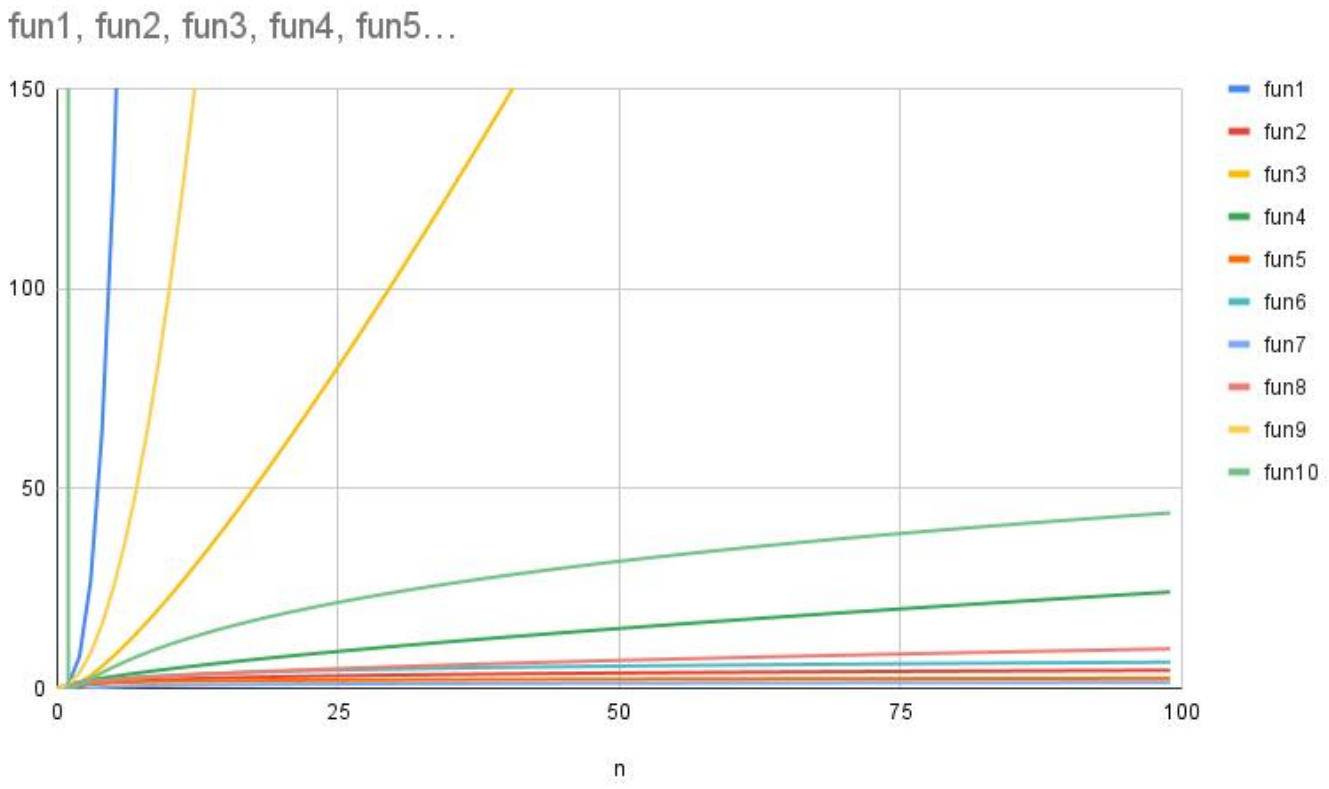
It is the time needed for the completion of an algorithm. To estimate the time complexity, we need to consider the cost of each fundamental instruction and the number of times the instruction is executed

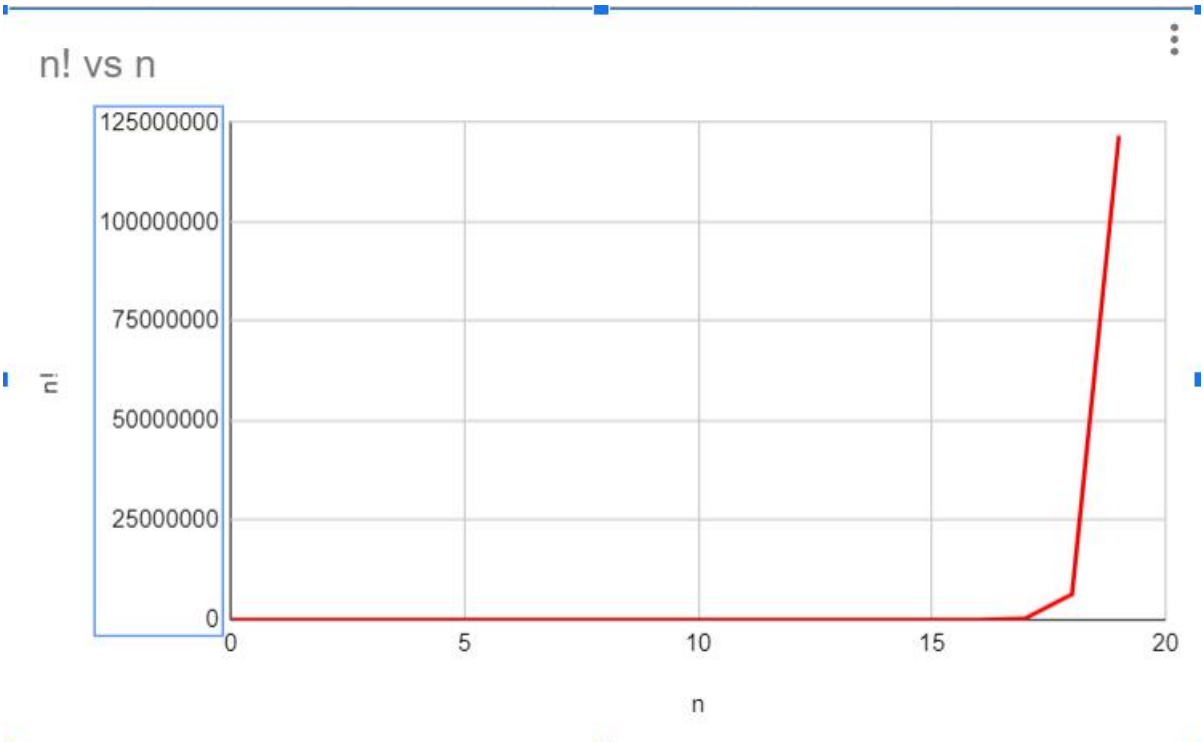
Order of growth is how the time of execution depends on the length of the input. In the above example, it is clearly evident that the time of execution quadratically depends on the length of the array. Order of growth will help to compute the running time with ease

Algorithm :

Here we have made 10 functions as follows :

1) Fun1 is the function that returns cube of a number passed in it.
2) Fun1 is the function that returns the logarithm of a number passed in it.
3) Fun1 is the function that returns the n times logarithm of a number n passed in it.
4) Fun1 is the function that returns the 2 raised to the logarithm of a number passed in it.
5) Fun1 is the function that returns the square root of logarithm of a number passed in it.
6) Fun1 is the function that returns the logarithm of a number passed in it.
7) Fun1 is the function that returns the logarithm of the logarithm of a number passed in it.
8) Fun1 is the function that returns square root of 2 raised to the logarithm of a number passed in it.
9) Fun1 is the function that returns 2 raised to twice the logarithm of a number passed in it.
10) Fun1 is the function that returns the square logarithm of a number passed in it.





CONCLUSION :

In the above experiment I plotted the graph of 10 different functions.
I observed that as the function changes their corresponding value also changes.
I some cases there is a drastic change, whereas sometimes the change is gradual.
Here, fun2, fun5, fun8 are the functions where there is gradual increase in their values.fun3, fun9 are the functions in which the increase in the values is more

whereas fun1 is changing the values very drastically with respect to n. I also plotted the graph for n factorial. N! graph was the one with most drastic change in its values.