Comparative Analysis of the Functions 2^n, n! and n^n

Ogheneovo, E. E.; Ejiofor, C. and Asagba, P. O.

Department of Computer Science, University of Port Harcourt, Port Harcourt, Nigeria.
edward_ogheneovo@yahoo.com, ejioforifeanyi@yahoo.com, pasagba@yahoo.com

Abstract

In this paper, we have attempted to do comparative analysis of the following functions: 2^n, n! and n^n. We analyzed these functions, discussed how the functions can be computed and also studied how their computational time can be derived. The paper also discussed how to evaluate a given algorithm and determine its time complexity and memory complexity using graphical representation of the various functions, displaying how the function behaves graphically. However, it was noticed that when data are inputted into these functions, they gave cumbersome outputs that make it impossible to determine the execution (computational) time for the functions. We plotted a graph by taking a snapshot of the integer values n = 1 to 10 to compute the functions of 2^n, n!, n^n. From the graph, we noticed that 2^n function had lower growth value; n^n had the largest growth value and n! had slightly greater increase in growth than the 2^n function. From our result, the execution time cannot be computed due to the largeness of the outputs. However, we were able to determine the function with the highest computing time and discovered that the time growth for the functions differs from one to the other.

Keywords Algorithm, Pseudo code, Exponential functions, Recursion, Complexity.

1.0 Introduction

Functions pervade all areas of mathematics and its applications. A function is a relation which associates any given number with another number [5]. Functions can be defined in several ways. We define a function from the set X into the set Y as a set of ordered pairs(x, y) where x is an element of x and y is an element of Y such that for X in x there is only one ordered pair (X, Y) in the function P. the notation used is

$f : X \rightarrow Y$ or $Y = f(x)Y$ or $X \rightarrow f(x)$
or $Y = f(X)$

A function is a mapping or transformation of x into y or f(x). The variable x represents elements of the domain and is called the independent variable. The variable y representing elements of the range and is called the dependent variable (Clarke, 1996). The function y = f(x) is often called single valued function since there is a unique y in the range for each specified x. the converse may not necessarily be true, y = f(x) is the image of x.

Often, a function depends on several independent variables. If there are n independent variables $x_1, x_2, x_3, \ldots, x_n$ and the range is the set of all possible values of corresponding to the domain of $(x_1, x_2, x_3, \ldots, x_n)$. We say that y is a function of x_i's, $y = f(x_1, x_2, x_3, \ldots, x_n)$. Letters other than f may be used to represent a function [3]

2.0 Exponential Functions (2^n And N^n)

Exponential functions are perhaps the most important class of functions in mathematics. We use this type of function to calculate interest on investments, growth and decline rates of populations, forensic investigations as well as in many other applications (Constatinescu, 2004). The application of this function to a value x is written as exp(x). Equivalently, this can be written in the form of e^x, where *e* is a mathematical constant, the base of the natural logarithm, which equals approximately 2.718281828, and is also

known as Euler's number (Schmidt and Makalic, 2009).

As a function of the real variable x, the graph of $y=e^x$ is always positive (above the x axis) and increasing (viewed left-to-right). It never touches the x axis, although it gets arbitrarily close to it (thus, the x axis is a horizontal asymptote to the graph). It's an inverse function [2].

Exponential growth is "bigger" and "faster" than polynomial growth. This means that, no matter what the degree is on a given polynomial, a given exponential function will eventually be bigger than the polynomial. Even though the exponential function may start out really, really small, it will eventually overtake the growth of the polynomial, since it doubles all the time [1]

2.1 Factorial

The number of sequences that can exist with a set of items, derived by multiplying the number of items by the next lowest number until 1 is reached. In Mathematics, product of all whole numbers up to 0 is considered. The special case zero factorial is defined to have value 0! = 1, consistent with the combinatorial interpretation of their being exactly on way to arrange zero objects. The factorial of all non-negative integers less than or equal to n.

n! = n(n-1)(n-2) … 3 x 2 x 1.
where n! represents n factorial
n = number of sets (items)

For instance, the factorial operation is encountered in many different areas of mathematics, notably in combinatory, algebra, and mathematical analysis [13]. Its most basic occurrence is the fact that the definition of the factorial function can also be extended to non-integer arguments, while retrieving its most important properties [4].

3.0 Computing Times Of Some Growing Functions.

The time for different functions differs from one to the other. Some functions have a greater time growth than others. For example, we consider the figures 6 and 7 (the graphs) below; it shows how the computing times for 6 of the typical functions on the table grow with a constant equal to 1. You will notice how the times 0(n) and 0(nlogn) grow much more slowly than the others[9]. For large data set, algorithms with a complexity greater than 0(nlogn) are often impractical [14], [8].

An algorithm which is exponential will only be practical for very small values of n and even if we decrease the leading constant, say by a factor of 2 or 3, we will not improve the amount of data we can handle significantly [7].

To see more precisely why a change in the constant, rather than to the order of an algorithm produces very little improvement in running time, we will consider the figure below:

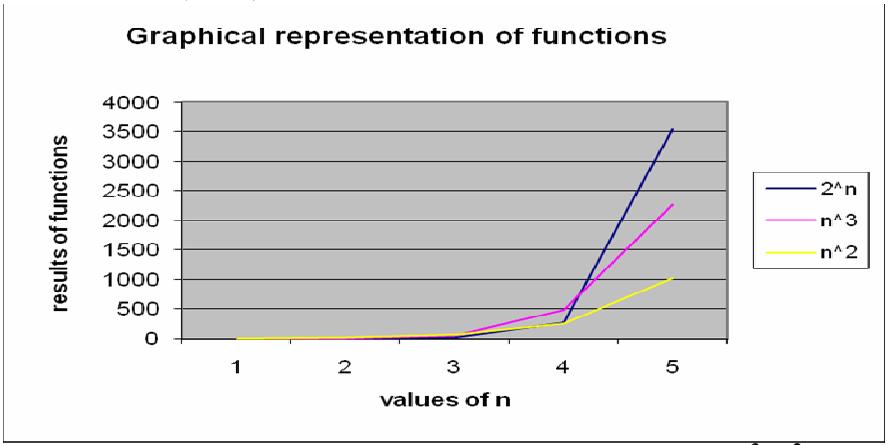

Fig 6: Graphical representation of the functions 2^n,n^3, n^2.

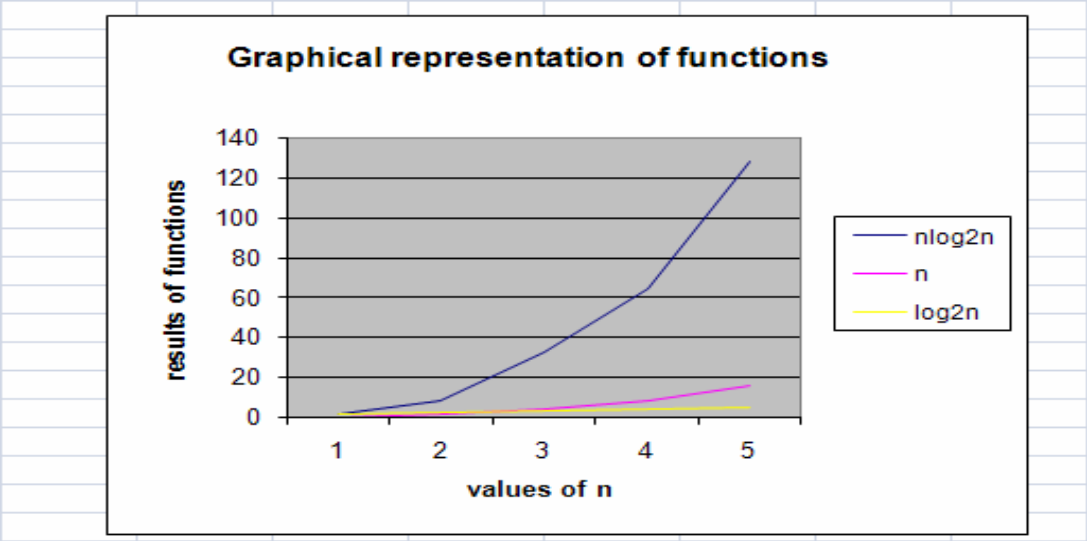

Fig **7: Graphical Representation of the functions $nlog_2n$, n, log_2n**.

3.1 Comparing the Growth of Functions 2^n, N!, N^n

Due to the fact that the execution time of function 2^n, n!, n^n is unreliable, and even though we had to give extra computing load to these functions, we still could not have a visible execution time. However, we decided to compare the growth of the functions in terms of the magnitude of the values they compute.

We implemented these algorithms by using a program in the form of Turbo C^{++} program.

When we entered the consecutive values for n from 1 – 150, the program generated growing output values for the various functions. We noticed that the program could not generate an output for the function when n is greater than 150. We decided to change the type of the value returned by the type of the value assigned to the local variable temp. The program was rerun and we noticed that although it generated values for n > 200, there were some errors (problems) with the results of some of the functions. We noticed that the result generated by n! and n^n started to generate negative integer values from n >= 20. In addition, we also discovered that after some time, n! started generating 0 as output. In other words, it stopped generating results as we continued increasing the integer values for n.

4.0 Discussion Of Results

In this section, we are going to make a certain assertion about the behaviours for the growing functions of 2^n, n!, n^n and we also use a graph plotted of the functions against the values of n to discuss our findings.

- With the graph of the growing function of 2^n, n!, n^n depicted in figure 8

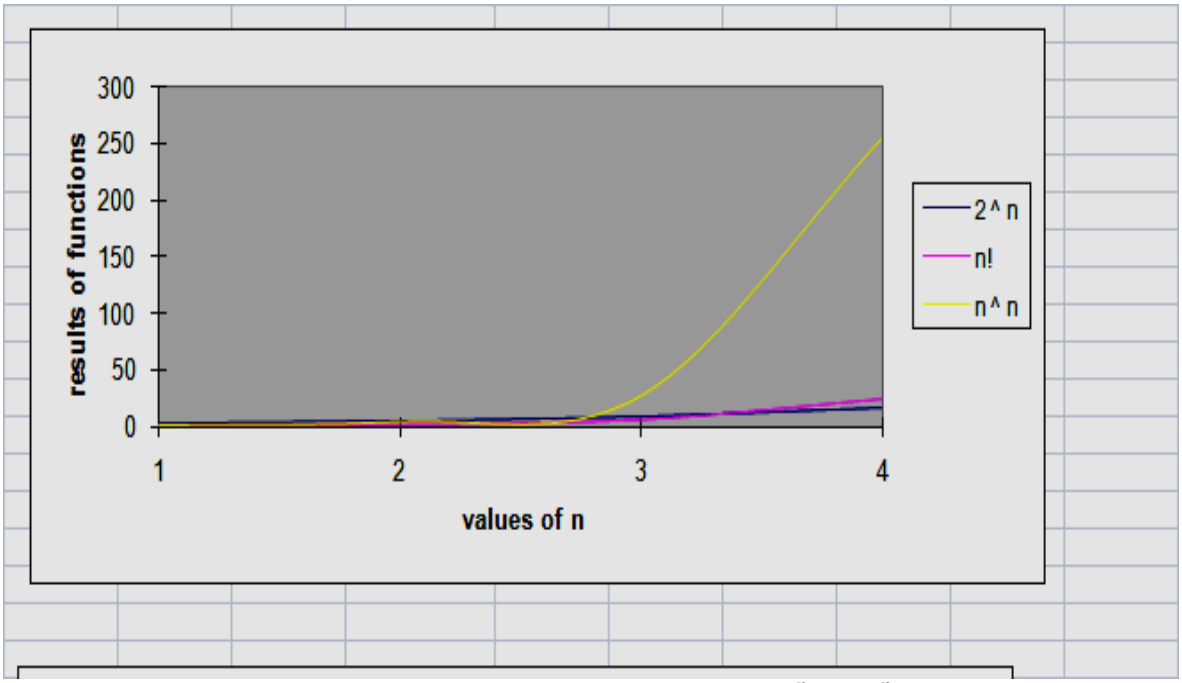

Figure 8: Graphical representation of 2^n, n!, n^n

We plotted a graph by taking a snapshot of the integer values n= 1 to 10 to compute the growing functions of 2^n, n!, n^n. In this graph, we discovered that the 2^n function had a lower growth of value than the n! and n^n functions. We also noticed that the n^n had the largest growth of values than the functions 2^n and n! We observed also that n! had a slightly greater increase in growth than the 2^n function.

5.0 **Conclusion**

The execution time of functions cannot be calculated due to the largeness of the outputs when a value is inputted. However, we were able to determine the function with the highest computing time from the altitude of the curves in the graphs plotted. The time growth for functions differs from one to the other. Some grow much slowly than others while others are immensely fast. However, the execution time could not be computed for the functions 2^n, n!, and n^n.

References

[1] Abramowitz and Stegun, (1972), *Exponential Functions, In Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables,* New York, Dover, pp. 69 – 71.

[2] Ahlfors, L. V. (1953), *Complex Analysis,* McGraw-Hill book Company Inc., U.S.A., pp. 56 – 80.

[3] Anyanwu, S. A. C. (2002), *Elementary Modern Algebra and trigonometry,* Markowitz, Centre for Research and Development, Port Harcourt, pp. 203.

[4] Borwein, P. (1985), *Complexity of Calculating Factorials,* Journal of Algorithm, Vol. 6, pp. 376 – 380.

[5] Clark, G. D. (1996), *Basic Calculus and co-ordinate Geometry for First Year University Students ,* GODSONS Books, Port Harcourt, pp. 1 – 3.

[6] Constantinescu, E. (2004), *Inequalities for Logarithmic and Exponential Functions, General Mathematics*, Vol. 12, No. 2, pp. 47 – 52.

[7] Gerety, C. and Cull, P. (1986), *Time Complexity of the Towers of Hanoi Problem,* ACM SIGACT News, Vol. 18, No. 1, pp. 80 – 87.

[8] Heileman, G. L. (1996), *Data Structures, Algorithms and Object-Oriented Programming,* Mo-Hill Book Co., Singapore, pp. 23 – 46.

[9] Horowitz, E. and Sahni, S. (1978), *Fundamentals of Computer Algorithms,* Library of Congress Cataloguing, pp. 20 – 39.

[10] Kruse, R. C. (1994), *Data Structures and Program Design,* Prentice-Hall, New Jersey, pp. 34 – 56.

[11] Sahni, S. (1998), *Data Structure, Algorithms and Application in C++,* Mc-Hill Book Co., Singapore, pp. 15 – 39.

[12] Schmids, D. F. and Makalic, E. (2009), Universal Models for the Exponential Distribution, IEEE *Transactions on Information Theory*, Vol. 55, No. 7, pp. 3087 – 3090.

[13] Wikipedia, the Free Encyclopaedia, Factorials.

[14] Wirth, N. (1976), *Algorithms and Data structures*, prentice-Hall, New Jersey, pp. 20 – 47.

Appendix A: Program Codes

```
#include <iostream.h>
#include <math.h>
#include <time.h>
#include <stdio.h>
#define size 1000
   double factorial(long);
int main()
{
long number, fact;
double  expon[size], factn[size], npowern[size];
cout<<" \n Enter the value of n: ";
cin>>number;
if(number < 0 )
{
 cout<<" You have entered a wrong input!"<<"\n";
 cout<<"\n Program stops!";
 return 0;
}
for(int i = 1; i <= number; i++)
{
expon[i] = pow(2, i);
factn[i]=factorial(i);
npowern[i]= pow(i, i);

}
cout<<"\tn     2 ^ n      n!       n ^ n \n";
cout<<"\t===   =====      ===      ===== \n";
for(int k = 1; k <= number; k++)
cout<<"\t"<<k<<"     "<<expon[k]<<"          "
<<factn[k]<<"        "<< npowern[k]<<"\n";
getchar();
 return 0;
 }
 double factorial(long n)
 {
  double temp;   if(n == 1)return 1;   if(n > 1)temp=n * factorial(n - 1);   return temp;  }
```