Contents

1 Problem 1 : Introduction

1.1 Description

F5 : ab^x is an exponential function, where a is a constant value, $a \neq 0$ and it also represents starting (initial) value , b is called base and is a positive real number and $b \neq 1$, x is called the exponent (power), it is independent variable . In this function, b is a constant value, whereas x is variable.

1.2 Domain

The domain for exponential function is the set of real numbers.
$x \in R$, $-\infty < x < \infty$, Domain : $\{x \mid x \in R\}$

1.3 Co-Domain

Co-Domain is the set of all possible function output values.
Suppose $y = f(x) = ab^x$, then $-\infty < y < \infty$, so the range will be $[-\infty, \infty]$.

1.4 Characteristics

- In exponential function, if $b > 0$, then it is known as exponential growth function (increasing function). Its graphical representation shown in the left part of the figure 1.
- In exponential function, if $0 < b < 1$, then it is known as exponential decay function (decreasing function). Its graphical representation shown in the right part of the figure 1.
- Exponential function have horizontal asymptote (i.e function approaches to a imaginary horizontal line but never crosses) at $Y = 0$ (i.e X – axis).
- They are continuous function.
- There is no symmetry in exponential function, so they are neither odd nor even function.
- Exponential function is not injective but is surjective.

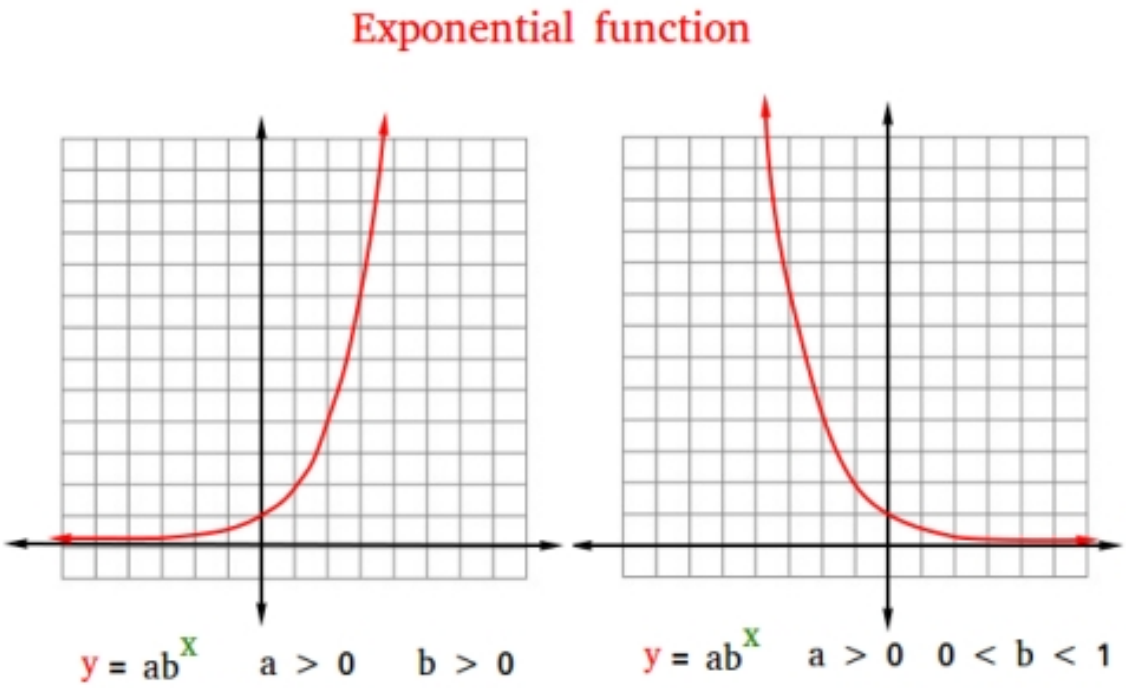

Figure 1: Exponential Function Graph (Growth And Decay).

1.5 Context of Use Model

Figure 2: Exponential Function Graph (Growth And Decay).

2 Problem 2 : Requirements

1. Requirement 1

   - **ID** : REQ1
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : High
   - **Difficulty** : Easy
   - **Description** : In the exponential function ab^x, the input value a should be greater than 0 i.e., $a > 0$ (or else it will result in output of the function to be 0 for every input), also input value of base b must be greater than 1 i.e., $b > 1$ (or else if $b = 1$, it will result in output of the function to be 'a' for every input of x).

2. Requirement 2

   - **ID** : REQ2
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : High
   - **Difficulty** : Easy
   - **Description** : In the exponential function ab^x, the input value of base b must not be negative as it will result will be complex numbers so $b > 0$.

3. Requirement 3

   - **ID** : REQ3
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : High
   - **Difficulty** : Easy
   - **Description** : In the exponential function ab^x, the input value of x must be any real number. i.e $x \in R$.

4. Requirement 4

    - **ID** : REQ4
    - **Version** : 1.0
    - **Type** : Functional Requirement
    - **Priority** : High
    - **Difficulty** : Medium
    - **Description** : The system must take input values of a, b and x from the users and return the output of ab^x function. For example, if $a = 2, b = 3, x = 2$, the output should be 18.

5. Requirement 5

    - **ID** : REQ5
    - **Version** : 1.0
    - **Type** : Functional Requirement
    - **Priority** : High
    - **Difficulty** : Easy
    - **Description** : If any of the input values a, b or x are not provided by the user, the system should not accept that input and ask user to provide the missing values.

6. Requirement 6

    - **ID** : REQ6
    - **Version** : 1.0
    - **Type** : Functional Requirement
    - **Priority** : High
    - **Difficulty** : Easy
    - **Description** : If the input values are not of integer type, the system must not accept it and handle the error and ask for integer values as input.

7. Requirement 7

   - **ID** : REQ7
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : Medium
   - **Difficulty** : Easy
   - **Description** : If the user enters any large input value which the system cannot handle, it should throw an exception and handle it accordingly.

8. Requirement 8

   - **ID** : REQ8
   - **Version** : 1.0
   - **Type** : Non-Functional Requirement
   - **Priority** : High
   - **Difficulty** : Medium
   - **Description** : The computation of the function must be performed within some desired time to provide better performance.

3 Problem 3 : Algorithms

Two algorithms (Algorithm 1 and Algorithm 2) have been selected for implementing the function $y = ab^x$. The pseudocode for each of the algorithm is provided in this section followed by their detailed description and the advantages as well as the disadvantages of each approach are discussed later.

3.1 Algorithms and Pseudocode

Algorithm 1 Iterative-Exponent(a,b,x)

Input : double number a, b, x
Output : value of the function ab^x represented by double number result

1: $temp \leftarrow 1$
2: **while** $x > 0$ **do**
3: $\quad temp \leftarrow temp \times b$
4: $\quad x \leftarrow x - 1$
5: **end while**
6: $result \leftarrow a \times temp$
7: return $result$

In this algorithm, the iterative approach is used for calculation of the given function, ab^x, Initially a temporary variable called $temp$ is initialized to 1 and to calculate b^x, While loop is used for all the values of x (exponent) and in each iteration, $temp$ is multiplied by b (base) and the value of x is decremented by 1. The loop continue until the exponent value x reaches to 0. At last, a is multiplied with the $temp$ value $(a * b^x)$ and the output is obtained in the result and it is returned.

Algorithm 2 Recursive-Exponent(a,b,x)

Input : double number a, b, x

Output : value of the function ab^x represented by double number result

1: $temp \leftarrow recursive\_power(b, x)$
2: $result \leftarrow a \times temp$
3: return $result$

Algorithm 3 recursive_power(b,x)

Input : double number b, x.

Output : value of b^x will be returned.

1: $holder \leftarrow recursive\_power(b, x/2)$
2: **if** $x = 0$ **then**
3: return 1
4: **else if** $x = 1$ **then**
5: return b
6: **else if** $x \bmod 2 = 0$ **then**
7: return $holder \times holder$
8: **else if** $x > 0$ **then**
9: return $holder \times holder \times b$
10: **else**
11: return $holder \times holder/b$
12: **end if**

In this algorithm, the recursive approach using divide and conquer strategy is used for the calculation of the given function ab^x. $recursive\_power(b, x)$ algorithm is used for calculating the value of b^x, the recursive call to the function with value base b and $x = x/2$ i.e $recursive\_power(b, x/2)$ is stored in holder, so that it needs to be computed just once and can be used later on. In the base case, if the value of x is 0, value 1 is returned, otherwise, if $x = 1$, value b is returned, if the exponent is divisible by 2, the algorithm recurses on $(holder * holder)$ and for odd value of exponent, it recurses on $(holder * holder * b)$, for negative value of exponent, the algorithm recurses on $(holder * holder/b)$. At last, a is multiplied with the returned value from $recursive\_power(b, x)$ and the output is obtained in the $result$, and it is returned.

3.2 Advantages and Disadvantages

3.2.1 Algorithm 1

Iterative Approach

Advantages:

1. Iterative algorithms are simple and easy to develop; they can be easily understood by the reader.

2. In Iterative approach there is no overhead of function calls and they do not use stack memory and hence don't suffer from stack overflow.

Disadvantages:

1. Some iterative algorithms are slower when compared to other approaches, Algorithm 1 which was described previously has time complexity of O(n) , so for larger inputs it is inefficient.

2. If the termination condition of control variable is not defined properly, it may lead to infinite loop.

3.2.2 Algorithm 2

Recursive Approach

Advantages:

1. Recursive Algorithms have less time complexity for certain problems; Algorithm 2 which was described previously has time complexity of $O(logn)$ which is better than that of Algorithm 1.

2. Recursive Algorithms are very useful in situations where particular solution is to be applied repeatedly.

Disadvantages:

1. Recursion is more difficult to understand in some algorithms and tracing them is difficult

2. Recursive algorithms utilize too much memory and when the base case is not defined properly it may lead to infinite loop or crashing of the CPU.

Decision

Algorithm 2 (recursive approach) would be a better option for the implementation of the function ab^x as it has lower time complexity and the computation of the function must be performed quickly, so it is preferable to use the recursive algorithm rather than the iterative algorithm.