Deliverable 1 **Rahul Sharma**
SOEN 6011 **40020034**
Function 6 : $f(x) = ab^x$ Date: 18/07/2019

Problem 1

1 Introduction

F6: $f(x) = ab^x$ is a exponential function where a is a constant, b is a base and x is a exponent or power. Here, x is a real variable. This function is one of the most commonly used function in mathematics.

2 Domain & Co-Domain

2.1 Domain

- it includes all the real numbers.
  To be specific, For $b > 0$, $x \in R$

2.2 Co-Domain

- $For\ b > 0$, range is $[0, \infty)$ where $b \in R$ and $x \in R$.
- $For\ b = 0$ and $y = 0$, range is 1 and $For\ b = 0$ and $x > 0$, range is 0.
- $For\ b < 0$, range is $(-\infty, \infty)$ where $b \in R$ and $x \in Z$ [1].

2.3 Restrictions

- a cannot be zero

3 Characteristics

- ***Growth*** : When $b > 0$, the function is called an exponential growth function. It can be seen on the left side of figure 1[2].
- ***Decay*** : When $b < 0$, the function is called an exponential decay function. It is shown on the right hand side of the graph.
- **Injectivity & Surjectivity** : This function is not injective which means it is not one-to-one function but it is surjective which means it is onto function.
- ***Commutativity***: This function is not commutative which means $x^y \neq y^x$ for $x \neq y$.

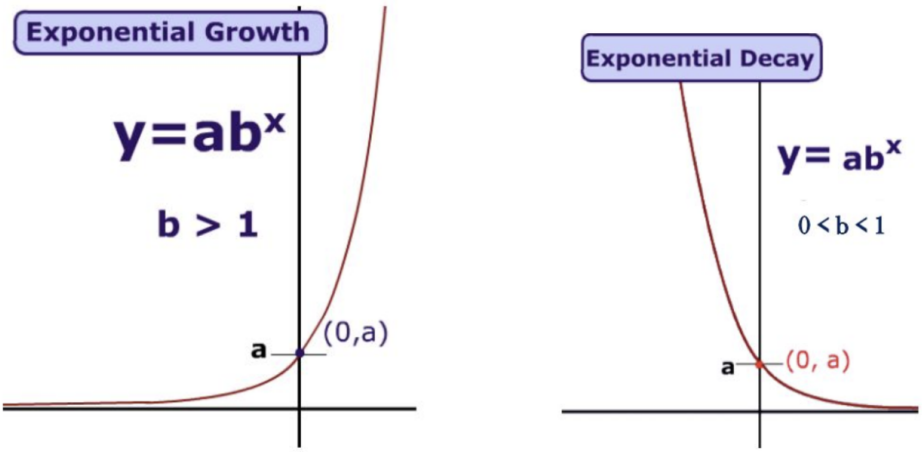

Figure 1: Exponential growth and Exponential Decay graph

Problem 2

Requirements

First Requirement

- **ID =** FR1
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** System shall take an input x, b and a to give an output of ab^x function. Example :for $x = 2, b = 1, a = 1$, $ab^x = 1$

Second Requirement

- **ID =** FR2
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** ab^x function does not depend on any other function.

Third Requirement

- **ID =** FR3
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** User shall give an input from all real numbers for x.

Fourth Requirement

- **ID =** FR4
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** when user gives other input then integer that is string then system shall not accept and should show input not properly defined.

Fifth Requirement

- **ID =** FR5
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** when user gives doesn't specify all three inputs a, b and x, the system should not accept and should throw an error.

Problem 3

Pseudocode and Algorithm

Calculate: $y = ab^x$

ALGORITHM 1: Iterative algorithm to calculate ab^x

1. function **exponent_iterative(a,b,x)**
in: double number x, a, b
out: double number sum
2. $sum \leftarrow 1$
3. $temp \leftarrow 1$
4. for $temp \leq x$ do
5. $\quad sum \leftarrow sum * b$
6. $\quad temp \leftarrow temp + 1$
7. end for
8. $sum \leftarrow sum * a$
9. return sum

We keep track of our result in a variable called sum, which is initially equal to 1. Then we loop from 1 to x number of times, incremented by one on each iteration and on each iteration we multiply sum by b. At the end of the loop value of sum is equal to b^x. At the last step we return our desired result of $a * sum$.

ALGORITHM 2: Recursive algorithm to calculate ab^x

1. function **exponent_recursive(a,b,x)**
in: double number x, a, b
out: double number result
3. $power \leftarrow exponent\_helper(b, x)$
4. $result = power * a$
5. return result

1. function **exponent_helper(b,x)**
in: double number x, b
out: double number sum
2. if $x < 0$ then
3. $b \leftarrow 1.0/b$
4. $x \leftarrow -x$
5. return $exponent\_helper(b, x)$
6. else if $x = 0$ then
7. return 1.0
8. else if $x = 1$ then
9. return b
10. else if $x \mod 2 = 0$ then
11. $b \leftarrow b * b$
12. $x \leftarrow x/2$
13. return $exponent\_helper(b, x)$
14. else
15. $b \leftarrow b * b$
16. $x \leftarrow x - 1$
17. $x \leftarrow x/2$
18. return $exponent\_helper(b, x)$
19. end if

We define a helper function called exponent_helper which calculates b^x. In the base case when $x = 0$, we return 1, otherwise when $x = 1$ we return b. When x is even we recurse on $b = b * b$ and $x = x/2$. In case when x is odd we recurse on $b = b * b$ and $x = (x - 1)/2$. In the end, in our main function exponenet_recursive we multiply the result of exponent_helper to the value of a and return our result.

Advantages and Disadvantages

Algorithm 1:

Advantages:
1. In terms of space complexity, iterative algorithms don't suffer from stack overflow because all operations are done on the heap.
2. They are fairly intuitive to read and follow for human beings.
Disadvantages:
1. The time complexity of the iterative algorithm is $O(n)$, hence it is not very efficient for larger inputs in terms of time.
2. Proper terminating condition for loop is needed or else we might get stuck in infinite loop.

Algorithm 2:

Advantages:
1. The time complexity of our recursive algorithm is $O(logn)$. Our version of recursive algorithm is optimized and tail recursive so that we don't get stack overflow error. So even for large inputs, it handles it very well.
2. Recursion has higher maintainability than loop. There's little to no modification needed if we handle base case correctly
Disadvantages:
1. When using recursion, it needs system continuously allocates memory space, thus it has a bad effect on efficiency leading to stack overflow.
2. The algorithm is difficult to understand, especially for edge cases and it is very easy to make mistakes in handling base case. 3. It is not really very readable.

Conclusion

My decision is to go with the recursive algorithm mostly because of the time complexity of the algorithm. The calculator should give result to the user in fairly less time and given the option between iterative and recursive solution, recursive would definitely perform well.

References

[1] MathBitsNotebook, `https://mathbitsnotebook.com/Algebra1/FunctionGraphs/FNGTypeExponential.html`

[2] TutorialsPoint, `https://www.tutorialspoint.com/java/lang/math_pow.htm`

[3] R. T. Barker and D. W. Biers, "Software usability testing: Do user self-consciousness and the laboratory environment make any difference?" in Part 2 (of 2), October 24, 1994 - October 28, 1994, .

[4] S. L. Sparagen and A. Riback, "Flexible software interface design," Ergonomics Des., vol. 7, (4), pp. 4-8, 1999.

[5] M. Hon, G. Russell and M. Welch, "Open source software considerations for law enforcement," IT Professional, vol. 12, (6), pp. 18-23, 2010. Available: http://dx.doi.org/10.1109/MITP.2010.121. DOI: 10.1109/MITP.2010.121.

[6] J. Ragot, D. Maquin and A. Kondo, "Control design for loop transfer recovery," in Part 1 (of 3), October 2, 1994 - October 5, 1994, .

[7] B. Haberman and H. Averbuch, "The case of base cases: Why are they so difficult to recognize? student difficulties with recursion," in ITiCSE 2002 - Proceedings of the 7th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, June 24, 2002 - June 28, 2002, Available: http://dx.doi.org/10.1145/544414.544441. DOI: 10.1145/544414.544441.