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.

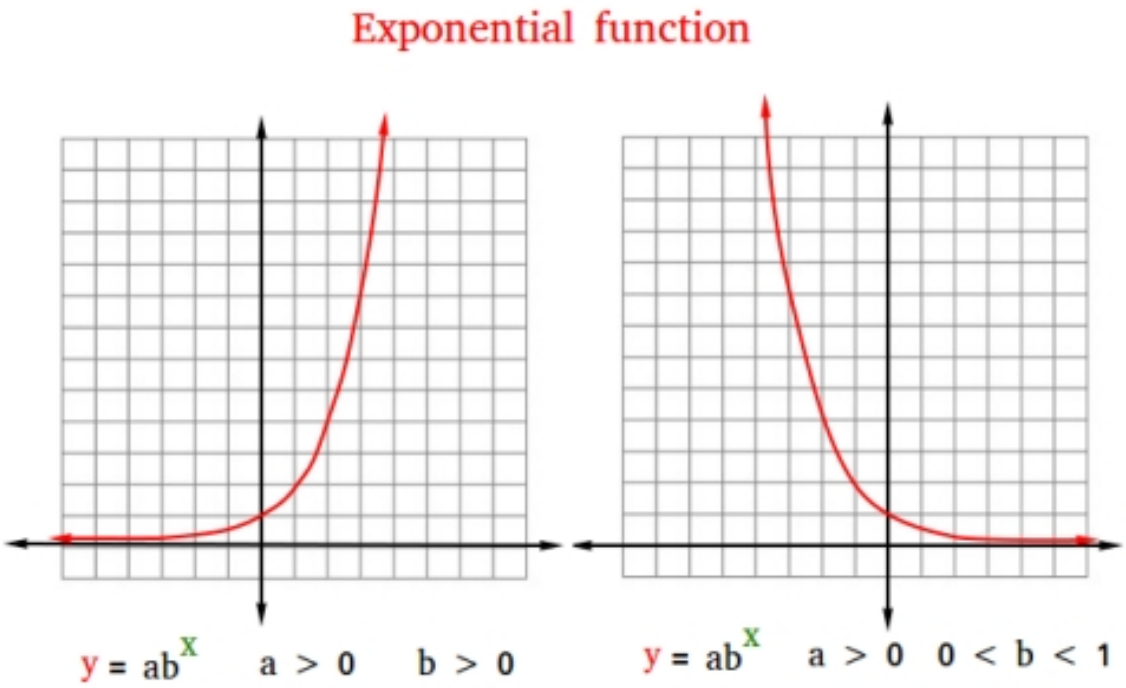

Figure 1: Exponential Function Graph (Growth And Decay).

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: if x ≠ (Integer)x then
2:    return fraction_power(b, x)
3: end if
4: helper ← recursive_power(b, x/2)
5: if x < 0 then
6:    b ← 1.0/b
7:    x ← −x
8:    if x mod 2 = 0 then
9:       return helper × helper
10:   else
11:      x ← x − 1
12:      return helper × helper × b
13:   end if
14: else if x = 0 then
15:   return 1.0
16: else if x = 1 then
17:   return b
18: else if x mod 2 = 0 then
19:   return helper × helper
20: else
21:   x ← x − 1
22:   return helper × helper × b
23: 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 helper, so that it needs to be computed just once and can be used later on. If the value of exponent x is fractional, it is checked and $fraction\_power(b, x)$ will compute all the power values of such exponents. For negative value of exponent, value of base $b$is reciprocated and that of exponent x is negated and on the basis of the parity the algorithm performs the computation. 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 $(helper*helper)$ and for odd value of exponent, the value of exponent is decremented by 1 and then it recurses on $(helper * helper * b)$, 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.

Algorithm 4 fractional_power(b,x)

Input : double number b, x.
Output : value of b^x will be returned.

1: $exponentValue \leftarrow 0$
2: $iterations \leftarrow 150$
3: $logValue \leftarrow logarithm\_calculator(b)$
4: **while** $iterations >= 0$ **do**
5: $powerValue \leftarrow recursive\_power((x * logValue), iterations)$
6: $factorialValue \leftarrow factorial\_calculator(iterations)$
7: $exponentValue \leftarrow exponentValue + (powerValue/factorialValue)$
8: $iterations \leftarrow iterations - 1$
9: **end while**
10: $reciprocalExponentValue \leftarrow 1/exponentValue$
11: **if** $x > 0$ **then**
12: return $exponentValue$
13: **else**
14: return $reciprocalExponentValue$
15: **end if**

Algorithm 5 logarithm_calculator(n)

Input : double number n.
Output : natural logarithmic value of n $ln(n)$ will be returned.

1: $iterations \leftarrow 100$
2: $logValue \leftarrow 0$
3: **if** $n < 0$ **then**
4: $n \leftarrow -n$
5: **end if**
6: $baseValue \leftarrow (n-1)/(n+1)$
7: **while** $iterations > 0$ **do**
8: **if** $iterations \bmod 2 \neq 0$ **then**
9: $logValue \leftarrow logValue + recursive\_power(baseValue, iterations)/iterations$
10: **end if**
11: $iterations \leftarrow iterations - 1$
12: **end while**
13: return $logValue * 2$

Algorithm 6 factorial_calculator(n)

Input : double number n.
Output : factorial value of n will be returned.

```
1: if n = 0 then
2:     return 1
3: end if
4: return (n * factorial_calculator(n − 1))
```

Algorithm $fractional\_power(b, x)$ helps in finding the power of any fractional exponent, the given exponent and base are converted as, $b^x = (e^{ln(b)})^x = e^{xln(b)}$, where the natural logarithm value for b is computed using the $logarithm\_calculator(n)$ algorithmFinally, the computation of $e^{xln(b)}$ is performed by calculating numerator(powerValue) by using $recursive\_power$, passing $x * valueofln(x)$ as base and *iteration number* as exponent and the denominator is the factorial value of the iteration number obtained using the $factorial\_calulcator(n)$ algorithm, numerator and denominator are divided and their result is stored in the $exponentValue$ which is returned after completion of all the given number of iterations.

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.

4 Problem 4

4.1 Debugger

4.1.1 Description

Debugging provides the facility for identifying and eliminating bugs, errors present in the program. This process can help in tracing the reasons behind the improper functional behaviour of any program. For debugging, I have used the IntelliJ debugger tool for java language and depending on the various installed plugins, it provides support for debugging code in different languages as well. IntelliJ debugger allows the facility to run the program step by step and keep a track of the values of the associated variables.

4.1.2 Advantages :

1. Breakpoints are special kind of markers which when encountered the debugger temporarily halts the program execution (suspended program state) and it makes easier to analyse and debug those special lines.

2. It allows to trace the values of the variable throughout the program, such trace would be helpful to identify the points where the incorrect values are assigned to the variables, also it provides information about the status of the threads.

3. It enables execution of any arbitrary piece of code in the middle of the program's execution as well as throw exceptions to test the functional behaviour of the program under different circumstances. Also allows running multiple debugging sessions at the same time.

4. IntelliJ debugger provides a feature called stepping tool which provides control over the step-by-step execution of the program and helps in deducing the source of the error.

5. IntelliJ debugger supports the feature to modify and adjust the parts of code without requiring to termination of the ongoing debugging process, it eliminates the need to modify the values of the variable and run the debugger again from start.

4.1.3 Disadvantages :

1. With the help of debugger, the location of the problem can be identified but the logical error associated with it cannot be known.

2. Debugger does not provide insights on design issues as well as structure of the code and there is a learning curve associated with it.

3. Debugger faces several problems while dealing with multithreading programs.

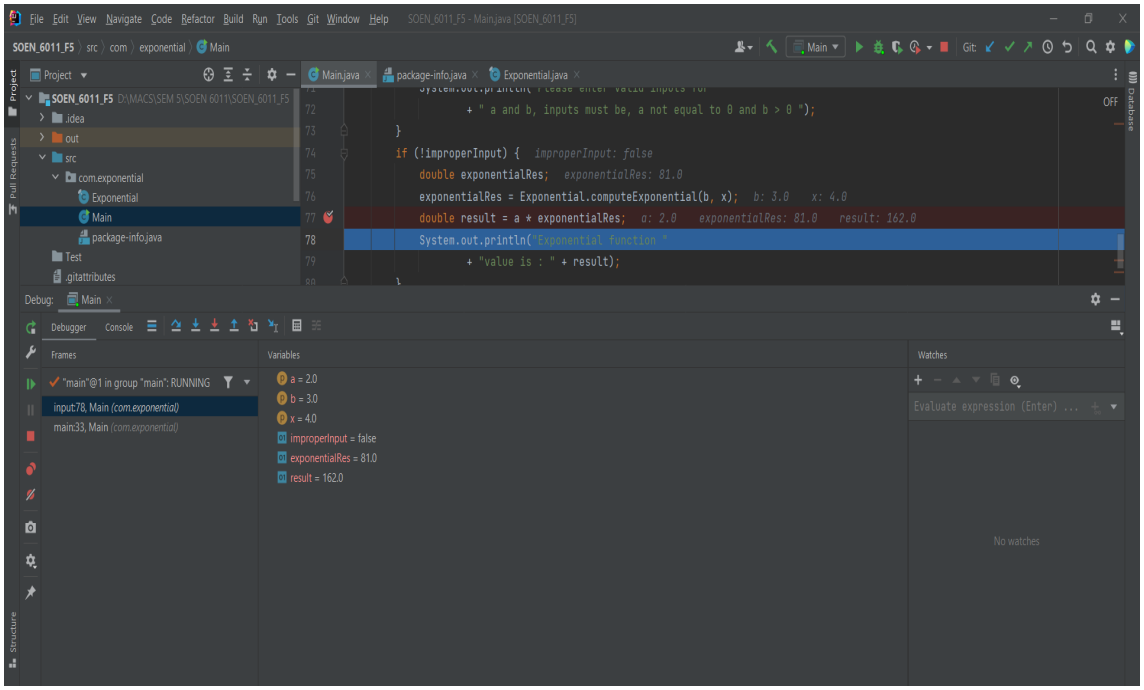

Figure 3: Debugging computation of exponential function

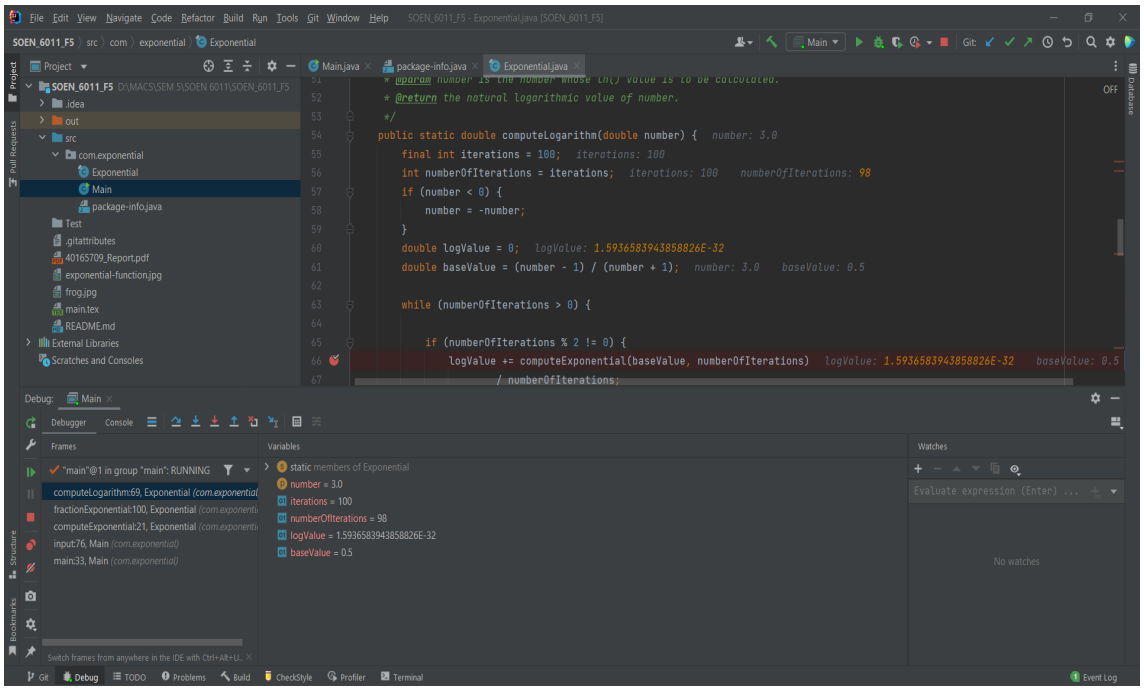

Figure 4: Debugging computation of fractional exponent value

4.2 Quality Attributes

This section highlights and describes several efforts made to achieve the desired quality attributes related to the java program .

1. **Efficient** – The program uses recursive algorithm to compute the exponential value and it provides time complexity of $O(logn)$ and it is the same for space complexity due to recursive stack space. It is more efficient and better choice in comparison to other approaches. As a result, the program runs quickly, and the function value is computed within required time window.

2. **Maintainable** – The program is partitioned into various methods each responsible for certain kind of computation, I have divided the user interface, error handling part ($Main.java$) from the core logic part ($Exponential.java$) related to the calculation of exponential function value. It allows easy modification of features/functionalities as well as adding new ones. It makes the code maintainable.

3. **Correctness** – I have designed several test cases to check the program against all the possible values of the base and exponent including invalid inputs, fractional exponent, negative exponents, every test case successfully runs and hence the developed program provides correct results and agrees to the desired specifications.

4. **Robust** – Input values are validated as they are entered by the user and all the invalid inputs are rejected. On entering such inputs, the program displays corresponding error message and asks for new valid input values from the user. In this way error handling is performed to avoid any kind of failure.

5. **Usable** – The program provides text-based user interface to the user for entering the values and displays the computed result. The error messages are meaningful and can be easily understood. It provides functionality for user to continue the computation for other values or exit the system. In this way, the program is made user-friendly.

6. **Portable** - The program is developed in java language, so it can be executed on other platforms as well as systems without requiring any modifications.