Problem 3: Pseudo-code and Algorithms

Shagun Shagun, 40138455

1 Algorithm and Pseudocode

Following is algorithm and the pseudo-code for ab^x

Algorithm 1 Recursive Approach - ab^x

```
procedure Function5(a, b, x)
    in:  String a, b, x
    out:  double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            result = a * exponential(x)
        else
            result = a * f5Power(b,x)
        return result;

procedure exponential(n)
    in:  int n
    out:  double result
    sum = 1
    m = 1
    for i <= n
    sum = 1 + m * sum/i
    return sum

procedure f5Power(x, n)
    in:  double x, int n
    out:  double result
    if ((n < 0) then
        return 1.0/powerHandler(x, n)
    else
        return powerHandler(x, n)

procedure powerHandler(x, n)
    in:  double x, double n
    out:  double result
    if (n == 0) then return 1
    if (n == 1) then return x
    if (nmod2 == 0) then
        return powerHandler(x * x, n/2)
    else
        return x * powerHandler(x * x, n/2)
```

Algorithm 2 Iterative Approach - ab^x

```
procedure Function5(a, b, x)
    in: String a, b, x
    out: double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            sum = 1
            m = 1 and i = 1
            for i <= x
            sum = 1 + m * sum/i
            end
            return a*sum
        else
            if (x == 0) then return 1
            if (x == 1) then return a*x
            else
                i = 1andpow = 1
                for i <= n
                pow = pow * x
                end
                return a*sum
```

Algorithm 3 Divide and Conquer Approach - ab^x

```
procedure Function5(a, b, x)
    in: String a, b, x
    out: double result
    result = 0
    if ((a || b) == "0") then return result
    else
        if (b == "e") then
            sum = 1
            m = 1 and i = 1
            for i <= x
            sum = 1 + m * sum/i
            end
            return a*sum
        else
            if (x == 0) then return 1
            if (x == 1) then return a*x
            else
                return a*calculatorPower(b,x)

procedure calculatorPower(b, x)
    in: double b, int x
    out: double result
    if (x == 1) then
        return 1
    else if (x mod 2 == 0) then
        return calculatorPower(b, x/2) * calculatorPower(b, x/2)
    else
        return b * calculatorPower(b, x/2) * calculatorPower(b, x/2)
```

2 Algorithm Description

2.1 Algorithm 1

The details of algorithm 1 is given following:
Complexity = $O(logn), \Omega(n)$
Space Complexity = $O(1)$
Approach = Recursion
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(logn)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. $O(logn)$ is the time complexity of the recursive algorithm. This recursive technique has tail recursive to avoid stack overflow issues and handle large inputs more efficiently.
2. Algorithm 1 has more readability and maintainable code.

2.2 Algorithm 2

The details of algorithm 2 is given following:
Complexity = $O(n), \Omega(n)$
Space Complexity = $O(1)$
Approach = Iterative
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(n)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. Because all operations are performed on the heap, iterative algorithms do not incur from stack overflow.
2. Space complexity is O(1).

Disadvantages

1. The complexity is $O(n)$.
2. The algorithm is not efficient.
3. The algorithm has poor readability and maintainability.

2.3 Algorithm 3

The details of algorithm 1 is given following:
Complexity = $O(n), \Omega(n)$
Space Complexity = $O(1)$
Approach = Divide and Conquer
Rationale = The a and b are the constants, depending on the value of b the complexities is being calculated such that if b = any integer then f5Power() will be called. So, the time complexity will be $O(n)$ and space complexity to 1. If b = e then taylor series will execute calling exponential() resulting $\Omega(n)$ time complexity and 1 space complexity.
Advantages

1. The problem has been solved in divide and conquer approach may result in robustness.
2. Space complexity is O(1).

Disadvantages

1. The complexity is $O(n)$.
2. The constant allocation of memory space, which leads to a stack overflow, has a substantial impact on efficiency.

3 Conclusion

By computing power(b, x/2) only once and storing it, the algorithm 2 and algorithm 3 can be optimized to $O(logn)$. The approach used in algorithm stores power only at once. Algorithm Implementation = Algorithm 1

References

[1] Wikiedia: Exponent Function,
`https://en.wikipedia.org/wiki/Exponential_function`

[2] Geeksforgeeks: Power Function,
`https://www.geeksforgeeks.org/write-a-c-program-to-calculate-powxn/`