Theory: (1)

I. Exponential Function using Iterative (Naive) Approach

The iterative (naive) approach involves calculating each term of the series one by one and summing them up. This method is called naive because it directly follows the definition of the Taylor series without any optimizations.

The power function xyx^yxy computes the value of a base xxx raised to an exponent yyy. This can be done using various methods, one of which is the iterative (naive) approach. Here is the theory behind this approach:

Iterative (Naive) Approach

The iterative (naive) approach involves multiplying the base xxx by itself yyy times. This method follows a straightforward loop-based technique to achieve the result.

Steps

1. **Initialization**:
   - Start with a variable mul initialized to 1. This will hold the result of the multiplication.
2. **Iteration**:
   - Use a loop that runs yyy times.
   - In each iteration, multiply mul by xxx.
3. **Output**:
   - After the loop completes, mul will contain the value of xyx^yxy.

Algorithm:

Marwadi University	Marwadi University Faculty of Technology Department of Information and Communication Technology
Subject: DAA (01CT0512)	**AIM: Implementing the Exponential (Power) function to calculate x^n using Divide and Conquer Approach**
Experiment No: 3	**Date:** / **Enrolment No: 92200133023**

Programming Language:

Code:

```
def power(x,y):
   mul=1
   for i in range(y):
      mul=mul*x
   print(mul)

x=2
y=6
power(x,y)
```

Output:

```
64
```

Space complexity: __________

Justification:____
____

Time complexity:

Best case time complexity: __________

Justification:____
____

Worst case time complexity: __________

Justification:____
____

Theory: (2)

Exponential Function with O(N) using Divide and Conquer Approach

The exponential function x^n can be computed using a divide and conquer approach, achieving a time complexity of O(log N). This method recursively breaks down the problem into smaller sub-problems of size x^(n/2) until reaching the base case of x^1.

Key Steps:

Base Case: When n is even, compute x^(n/2) recursively and multiply the results.

Recursive Case: Divide n into two halves, n1 and n2, and compute x^(n1) and x^(n2) recursively.

Combine Results: Multiply the results of the two recursive calls to obtain x^n.

Example: Computing x^13 using divide and conquer:

n is odd, so we use the recursive case.

Divide 13 into n1 = 6 and n2 = 7.

Compute x^6 and x^7 recursively.

Combine results: x^13 = (x^6)^2 * x

Repeat the process for x^6 until reaching the base case of x^1.

Algorithm:

Programming Language:

Code:

```
def power(x,y):
   if(y==0):
      return 1
   elif(y%2==0):
      return (power(x,int(y/2))*power(x,int(y/2)))
   else:
      return (x*power(x,int(y/2))*power(x,int(y/2)))

x=2
y=7
print(power(x,y))
```

Output:

Space complexity: ___________

Justification:____

____

Time complexity:

Best case time complexity: ___________

Justification:____
____

Worst case time complexity: __________

Justification:____
____

Theory: (3)

Exponential Function with O(logN) using Divide and Conquer Approach

Divide and Conquer Concept

Divide and Conquer is a strategy that breaks down a problem into smaller subproblems, solves each subproblem recursively, and then combines the solutions to solve the original problem. For computing xyx^yxy, we can use this strategy as follows:

1. **Base Case**:
   - x0=1x^0 = 1x0=1
   - x1=xx^1 = xx1=x
2. **Divide**:
   - If yyy is even, xy=(xy/2)×(xy/2)x^y = (x^{y/2}) \times (x^{y/2})xy=(xy/2)×(xy/2).
   - If yyy is odd, xy=x×(x(y−1)/2)×(x(y−1)/2)x^y = x \times (x^{(y-1)/2}) \times (x^{(y-1)/2})xy=x×(x(y−1)/2)×(x(y−1)/2).

This approach works because we effectively halve the exponent in each step, thus reducing the number of multiplications significantly.

Algorithm Steps

1. **Check Base Case**:
    - If y=0y = 0y=0, return 1.
    - If y=1y = 1y=1, return xxx.
2. **Recursive Case**:
    - If yyy is even:
        - Compute half_power=power(x,y/2)\text{half_power} = \text{power}(x, y/2)half_power=power(x,y/2).
        - Return half_power×half_power\text{half_power} \times \text{half_power}half_power×half_power.
    - If yyy is odd:
        - Compute half_power=power(x,(y−1)/2)\text{half_power} = \text{power}(x, (y-1)/2)half_power=power(x,(y−1)/2).
        - Return x×half_power×half_powerx \times \text{half_power} \times \text{half_power}x×half_power×half_power.

Algorithm:

Programming Language:

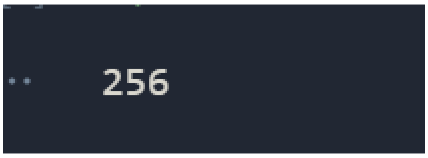
Marwadi University	Marwadi University Faculty of Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	AIM: Implementing the Exponential (Power) function to calculate x^n using Divide and Conquer Approach	
Experiment No: 3	Date:	Enrolment No: 92200133023

Code:

```
def power(x,y):
   if(y==0):
      return 1
   temp = power(x,int(y/2))
   if(y%2==0):
      return (temp*temp)
   else:
      return (x*temp*temp)

x=2
y=8
print(power(x,y))
```

Output:



Space complexity: ___________

Justification:____

____

Time complexity:

Best case time complexity: ___________

Justification:____

____

Worst case time complexity: __________

Justification:____
____