SSN College of Engineering

Department of Information Technology

UIT2201 — Programming and Data Structures

2022 – 2023

Exercise — 04

U. Pranaav | IT-B | 3122225002093

I. AIM:

The purpose of this exercise is to design and analyze algorithms and perform empirical analysis of algorithms as well.

1. Let p(x) be a polynomial of degree n.

(a) Implement a simple O(n^2))-time algorithm using Python for computing p(x), for a given value of x.

(b) Implement a O(n log n) algorithm for computing p(x), based upon a more efficient calculation of x.

(c) Now, consider rewriting p(x) as

$p(x) = a0 + x (a1 + x (a2 + x (a3 + \cdots + x (an-1 + xan) \cdots)))$

which is known as the Horner's method. Write a Python function to compute p(x) using this method. Analyze the time complexity of your code and express the same in asymptotic notation.

(d) Perform empirical analysis of run time of all the three versions: Execute the functions for different values of n (degree of the polynomial) and tabulate the results (note that each entry should be an average over several runs, say m). Use randomly generated values of a0, a1, , an−1 for each value of n. Perform ratio analysis with well known complexity classes to confirm the growth rate of running times of all the three versions.

II. CODE:

```
# -*- coding: utf-8 -*-


"""

This module provides a series of functions that calculate the
value of a polynomial, given the polynomial coefficient terms
and the value of x in 3 different time complexities, namely
O(n^2), O(nlogn) and O(n). This is a part of the exercises
given under the course UIT2201 (Programming and Data Structures).

In this source code I have executed my own logic. The code
follows good coding practices.
```

```python
Your comments and suggestions are welcome.

Created on Wed Apr 29 2023

Revised on Wed May 01 2023

Original Author: U. Pranaav <pranaav2210205@ssn.edu.in>

"""


def complexity_n2(coefficients,x):

    '''
    The given function takes in the coefficients of a
    polynomial as well as the value of x to calculate
    the value of polynomial at a given x as well as
    number of operations taken.

    The input is not modified in any way and there
    are no side effects.

    args:
        coefficients: the coefficients of the polynomial
        x: the value of x to be substituted

    Returns:
        A tuple of polynomial value and number of
        operations performed.

    '''

    degree = len(coefficients) - 1
    total_sum = 0
    count = 0
    count += 2

    for coeff in coefficients:
        count += 1
        prod = 1
        for i in range(degree):
            count += 1
            prod *= x
        total_sum += prod*coeff
        degree -= 1
        count += 2

    return (total_sum, count)


def power(x, y):

    '''
    The given function calculates the value of x raised
    to the power y in time O(logn).

    The input is not modified in any way and there are no
    side effects.

    args:
        x: the base
        y: the power

    Returns:
        Value of x raised to the power y.
```

```python
    '''

    global ct

    ct += 1

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


def complexity_nlogn(coefficients,x):

    '''
    The given function takes in the coefficients of a
    polynomial as well as the value of x to calculate
    the value of polynomial at a given x as well as
    number of operations taken. This implementation
    calculates value in O(nlogn) time.

    The input is not modified in any way and there
    are no side effects.

    args:
        coefficients: the coefficients of the polynomial
        x: the value of x to be substituted

    Returns:
        The polynomial value at given value x.

    '''

    global ct
    degree = len(coefficients) - 1
    total_sum = 0
    for coeff in coefficients:
        ct += 1
        prod = power(x,degree)
        total_sum += prod*coeff
        degree -= 1

    return total_sum


def horner_method(coefficients,x):

    '''
    The given function takes in the coefficients of a
    polynomial as well as the value of x to calculate
    the value of polynomial at a given x as well as
    number of operations taken. This is the implementation
    of horner method which calculates value in O(n) time.

    The input is not modified in any way and there
    are no side effects.

    args:
        coefficients: the coefficients of the polynomial
        x: the value of x to be substituted

    Returns:
        The polynomial value at given value x.
```

```
    '''

    global homer_ct

    result = coefficients[0]
    homer_ct += 1
    for i in range(1, len(coefficients)):
        homer_ct += 1
        result = result*x + coefficients[i]
    return result


#driver code
if __name__ == '__main__':
    #this part of the code will only be run when the function is called directly
    #it will not be executed when it is imported as a module

    ct = 0
    homer_ct = 0
    coeff = [x for x in range(1,11)]

    print(f"Coefficients of the polynomial being evaluated is: {coeff}")
    print()

    value, count = complexity_n2(coeff,2)
    print("Value of polynomial calculated by algorithm with O(n^2) time complexity: ",value)
    print("Number of comparisons is :",count)
    print()

    print("Value of polynomial calculated by lgorithm with O(nlogn) time complexity: ",complexity_nlogn(coeff,2))
    print("Number of comparisons is :",ct)
    print()

    print("Value of polynomial calculated by horner method: ",horner_method(coeff,2))
    print("Number of comparisons is :",homer_ct)
    print()
```

III. OUTPUT:

Coefficients of the polynomial being evaluated is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Value of polynomial calculated by algorithm with O(n^2) time complexity: 2036
Number of comparisons is : 77

Value of polynomial calculated by lgorithm with O(nlogn) time complexity: 2036
Number of comparisons is : 45

Value of polynomial calculated by horner method: 2036
Number of comparisons is : 10

IV. EMPIRICAL ANALYSIS:

UIT2201 - PROGRAMMING AND DATA STRUCTURES

Ex-04

EMPIRICAL ANALYSIS:

I. Algorithm $O(n^2)$:

n	$f(n)$	$f(n)/n$	$f(n)/n^2$	$f(n)/n \log n$
10	77	7.7	0.77	2.3179
100	5252	52.52	0.5252	2.91
1000	502502	5025.02	0.502502	50.42
10000	50025002	5002.5002	0.5003	376.468
20000	200050002	10002.5	0.50013	700.06

We can see that the closest fit is $f(n)/n^2$

$\therefore$ Time complexity of algorithm is $O(n^2)$.

II. Algorithm $O(n \log n)$:

n	$f(n)$	$f(n)/n$	$f(n)/n^2$	$f(n)/n \log n$
10	45	4.5	0.45	1.3546
100	773	7.73	0.0773	1.1635
1000	10977	10.977	0.01098	1.1014
10000	143617	14.3617	0.001436	1.081
20000	307233	15.36	7.68×10^{-4}	1.075

We can see that the closest fit is $f(n)/n \log n$

$\therefore$ Time complexity of algorithm is $O(n \log n)$.

III. Algorithm Horner's method :

n	f(n)	f(n)/n	f(n)/n²	f(n)/nlogn
10	10	1	0.1	0.3010
100	100	1	0.01	0.1505
1000	1000	1	0.001	0.1003
10000	10000	1	0.0001	0.07525
20000	20000	1	0.00005	0.06999

We can see that the closest fit is f(n)/n.

∴ Time complexity of Horner's method is O(n).