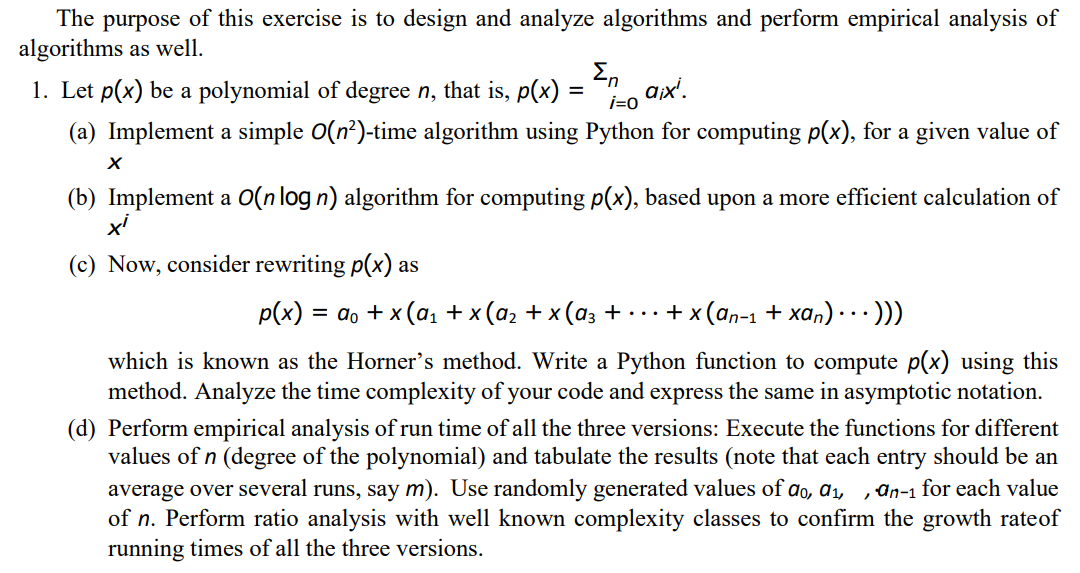
|  |  |
| --- | --- |
| Ex. No: 4 | UIT2201 — Programming and Data Structures |
| 29x-04-2023 |

**Aim:**

To execute the following programs and note the output.

**PART – A**



**Code:**

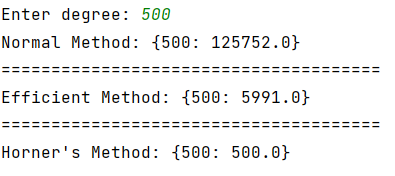
**implementation.py**

count = 0  
def solve\_quadratic(coeffs, x):  
 count = 0  
 coefficients = coeffs  
 count += 1  
 value = 0  
 count += 1  
 for idx in range(0, len(coefficients)):  
 temp = coefficients[idx]  
 count += 1  
 for i in range(len(coefficients) - (idx + 1)):  
 temp \*= x  
 count += 1  
 value += temp  
 count += 1  
 return value, count  
  
def pow(x,y):  
 global count  
 count += 1  
 if y == 0:  
 return 1  
 else:  
 if y % 2 == 0:  
 temp = pow(x, y//2)  
 return temp \* temp  
 else:  
 if y > 0:  
 temp = pow(x, y//2)  
 return x \* temp \* temp  
 else:  
 temp = pow(x, y//2)  
 return (temp \* temp)/x  
  
def efficient\_solve\_quadratic(coeffs, x):  
 global count  
 count = 0  
 coefficients = coeffs  
 count += 1  
 value = 0.0  
 count += 1  
 for idx in range(0, len(coefficients)):  
 count += 1  
 temp = coefficients[idx] \* pow(x, len(coefficients) - (idx + 1))  
 count += 1  
 value += temp  
 count += 1  
  
 return value, count  
  
def horners\_method(coeffs, x):  
 global count   
 count = 0  
 sum = coeffs[0]  
 count += 1  
 for idx in range(1, len(coeffs)):  
 sum = sum \* x + coeffs[idx]  
 count += 1  
 return sum ,count

**analysis.py**

from implementation import solve\_quadratic, efficient\_solve\_quadratic, horners\_method  
import random  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 normalmethod = {}  
 efficientmethod = {}  
 hornersmethod = {}  
 n = int(input("Enter degree: "))  
 if n == 0:  
 print("Degree cannot be zero!")  
 exit()  
 for degree in range(n, n+1):  
 coeffs = []  
 normal = []  
 efficient = []  
 horners = []  
 for \_ in range(1):  
 coeffs = [random.uniform(-100.0, 100.0) for \_ in range(degree)]  
 xvalue = random.uniform(-100.0, 100.0)  
 normal.append(solve\_quadratic(coeffs=coeffs, x=xvalue)[1])  
 efficient.append(efficient\_solve\_quadratic(coeffs=coeffs, x=xvalue)[1])  
 horners.append(horners\_method(coeffs=coeffs, x=xvalue)[1])  
 normalmethod[degree] = sum(normal)/len(normal)  
 efficientmethod[degree] = sum(efficient)/len(efficient)  
 hornersmethod[degree] = sum(horners)/len(horners)  
  
 print("Normal Method:", normalmethod)  
  
 print("======================================")  
   
 print("Efficient Method:", efficientmethod)  
  
 print("======================================")  
  
 print("Horner's Method:", hornersmethod)

**Sample Output:**

****

**Tabulation:**

**A picture containing text, wall

Description automatically generated**

**Table

Description automatically generated**

**Table

Description automatically generated**

**Inference:**

From the above tabulation, we can infer that,

* If f(n)/O(n) diverges to some random value, then our complexity is under-estimated.
* If f(n)/O(n) converges to zero, then our complexity is over-estimated.
* If the value tends to a particular constant value, then our complexity is the best estimate.

In normal method, the time complexity is O(n2).

In recursive method, the time complexity is O(nlogn)

In Horner’s method, the time complexity is O(n).

Lower the time complexity, better and efficient is the algorithm.

Here Horner’s method is the best and efficient algorithm as it has a time complexity of O(n).

**========================**