CSE221_LabAssignment01_Summer2022

Submission Guidelines:

1. You should implement the given codes in python for this first assignment. Other codes can be done in any language.
2. For **problem 1 and 4**, write and download separate python files named **problem1.py** and **problem4.py** respectively.
3. For **problem 2 and 5**, write down the explanations in your copy, create a pdf file with the images of your hand written answers and name it **explanations.pdf**.
4. For **problem 3**, you need to download the image(.png) of the graph with a value of n good enough to differentiate between the two implementations. Name it **problem3.png**.
5. Finally zip all the files and rename this zip file as per this format : **LabSectionNo_ID_CSE221LabAssignmentNo_Summer2022.zip**. [Example : **LabSection01_21101XXX_CSE221LabAssignment01_Summer2022.zip**]
6. You **MUST** follow all the guidelines, naming/file/zipping convention stated above. **Failure to do so will result in straight 50% mark deduction.**

[1] File I/O: [10 marks]

Parity: A number has even parity if it's an even number, and odd parity if it's an odd number.
Palindrome: A palindrome is a sequence of characters which reads the same backward as forward, such as "madam", "racecar" or "bob".

Given pairs of a number and a string, check the parity of the number and whether the string is a palindrome or not. In case of float/ decimal, indicate that it cannot have parity. In a text file, some pairs will be given in separate lines. Read the words from a text (input.txt) file, do the above mentioned operations, and save the outputs in another text (output.txt) file using File I/O operations. Finally, in a text file named "**records.txt**", write the percentage of odd, even and no parity, and percentage of palindromes and non-palindromes. Ideally you should store the inputs from the text file into a data structure (e.g. array, list etc.). You can either:

- pass the array as an argument to the **isPalindrome** function and return the output array, OR,
- you can check the words one by one using a loop and return true/false

Sample input (inside **input.txt** file): [Download input text file from here]

1 madam
2 apple
3.6 racecar
89 parrot
45.2 discord

Sample output (inside output.txt file):

1 has odd parity and madam is a palindrome
2 has even parity and apple is not a palindrome
3.6 cannot have parity and racecar is a palindrome
89 has odd parity and parrot is not a palindrome
45.2 cannot have parity and discord is not a palindrome

Sample output (inside **record.txt** file):

Percentage of odd parity: 40%
Percentage of even parity: 20%
Percentage of no parity: 40%
Percentage of palindrome: 40%
Percentage of non-palindrome: 60%

Pseudocode for isPalindrome function:

```
Word <- input
IF word=null/empty THEN
        Return not palindrome
N<- length of word
For i<N/2
        If word[i] != word[N-1-i]
                Return not palindrome
        i++
Return palindrome
```

[2] N-th Fibonacci Number: [3 marks]

You are given two different codes for finding the n-th fibonacci number. Find the time complexity of both the implementations and compare the two.

Implementation - 1

```
def fibonacci_1(n):
    if n <= 0:
        print("Invalid input!")
    elif n <= 2:
        return n-1
    else:
        return fibonacci_1(n-1)+fibonacci_1(n-2)

n = int(input("Enter a number: "))
nth_fib = fibonacci_1(n)
print("The %d-th fibonacci number is %d" % (n, nth_fib))
```

Implementation - 2

```
def fibonacci_2(n):
    fibonacci_array = [0,1]
    if n < 0:
        print("Invalid input!")
    elif n <= 2:
        return fibonacci_array[n-1]
    else:
        for i in range(2,n):
            fibonacci_array.append(fibonacci_array[i-1] + fibonacci_array[i-2])
        return fibonacci_array[-1]

n = int(input("Enter a number: "))
nth_fib = fibonacci_2(n)
print("The %d-th fibonacci number is %d" % (n, nth_fib))
```

[3] Graph Plot: [2 marks]

Append the following code segment after the implementations given in the previous problem. [Yes, The code is given. Just Copy-Paste it]. This will generate a graph with the value of **n** along the x-axis and **time required** along the y-axis. You can see both the curves in the same graph for better comparison. Generate graphs for different values of n and see how the performances change drastically for larger values of n.

Code for plotting graph:

```
import time
import math
import matplotlib.pyplot as plt
import numpy as np

#change the value of n for your own experimentation
n = 30

x = [i for i in range(n)]
y = [0 for i in range(n)]
z = [0 for i in range(n)]

for i in range(n-1):
    start = time.time()
    fibonacci_1(x[i+1])
    y[i+1]= time.time()-start
    start = time.time()
    fibonacci_2(x[i+1])
    z[i+1]= time.time()-start

x_interval = math.ceil(n/10)
plt.plot(x, y, 'r')
plt.plot(x, z, 'b')
plt.xticks(np.arange(min(x), max(x)+1, x_interval))
plt.xlabel('n-th position')
plt.ylabel('time')

plt.title('Comparing Time Complexity!')
plt.show()
```

[4] Matrix Multiplication: [5 marks]

Write a program that will take two **n x n** matrices as input and give their product as output. Follow the **following pseudocode** to implement the program. **Analyse the time complexity** of the program after implementation.

Pseudocode:

```
Procedure Multiply_matrix(A,B)
        Input A,B nxn matrix
        Output C nxn matrix

begin
        Initialize C as a nxn zero matrix

        for i = 0 to n-1
            for j = 0 to n-1
                for k = 0 to n-1
                    C[i,j] += A[i,k]*B[k,j]
                end for
            end for
        end for
end Multply_matrix
```

[5] : Recursion Tree Time Complexity (5 marks)

Find the Worst case time Complexity of the following recursive functions

1. T(n) = T(n/2)+n-1, T(1) = 0

2. T(n) = T(n-1)+n -1, T(1) = 0

3. T(n)=T(n/3)+2T(n/3)+n

4. Proof that for T(n)=2T(n/2) + n^2, the worst case complexity will be n^2.