CS211 - Data Structures and Algorithms

Lab 4

Instructor: Dr. Sharaf Hussain
E-mail: shhussain@uit.edu

Semester: Fall, 2021

1 Objective

The purpose of this lab session is to study algorithms complexity using simulations.

2 Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

3 How to Submit

- Submit lab work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW4.py)

4 Exercises

Task 1: Sample Time Complexity Simulation

The following python code contains the modified versions of the version1() and version2() functions that represent two ways to perform the same task of addition of all the elements of a random matrix. Counter variable is introduced to count the number of statements executed in each function call. Your first task is execute the same task first and check how the increase in the input size increases the number of statements to be executed.

```
from matplotlib import pyplot as plt
import numpy as np
def version1(n):
  totalSum = 0 # Version 1
  matrix= np.random.randint(10, size=(n, n))
  rowSum=[0]*n
  counter=3   #Counts the number of statement excuted, excluding the
counter updates
  for i in range(0,n):
    rowSum[i] = 0
    counter+=1
    for j in range(0, n ) :
      rowSum[i] = rowSum[i] + matrix[i,j]
      totalSum = totalSum + matrix[i,j]
      counter+=2
  return counter
def version2(n):
  totalSum = 0 # Version 1
  matrix= np.random.randint(10, size=(n, n))
  rowSum=[0]*n
```

```
      counter=3    #Counts the number of statement excuted, excluding the
   counter updates
      for i in range(0,n,1):
        rowSum[i] = 0
        counter+=1
        for j in range(0, n ) :
          rowSum[i] = rowSum[i] + matrix[i,j]
          counter+=1
        totalSum = totalSum + rowSum[i]
        counter+=1
      return counter

   def simulation(n):
      steps_version1=[0]*n
      steps_version2 = [0] * n
      for i in range(0,n):
        steps_version1[i]=version1(i)
        steps_version2[i]=version2(i)
      x=list(range(n))
      plt.plot(x,steps_version2)
      plt.plot(x, steps_version1)
      plt.grid(which='both')
      plt.xlabel('Input Size(n)')
      plt.ylabel('Number of Steps')
      plt.legend(['version2','version1'])
      plt.show()

```

The following figure shows the plot for input size in range 0 to n-1

```
simulation(50)
```

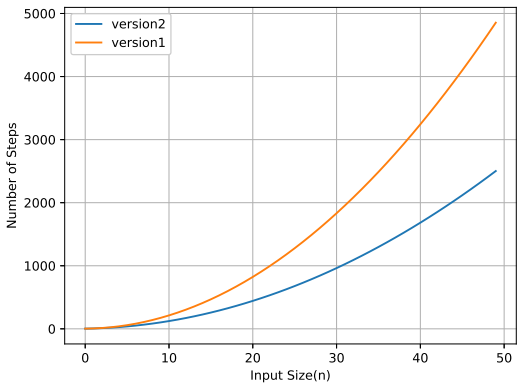

Figure 1

Task 2: List performance

Analyze the list performance of following list functions.

- concatenation
- append
- comprehension
- range function

```
      from timeit import Timer
      import matplotlib.pyplot as plt

      def concatenation():
      l = []
      for i in range(1000):
```

```
        l = l + [i]

        def append():
        l = []
        for i in range(1000):
        l.append(i)

        def comprehension():
        l = [i for i in range(1000)]

        def rangeFunction():
        l = list(range(1000))

        t1 = Timer("concatenation()", "from __main__ import concatenation")
        concatTime = t1.timeit(number=1000)
        print("concatination ", concatTime, "milliseconds")
        t2 = Timer("append()", "from __main__ import append")
        appendTime = t2.timeit(number=1000)
        print("append ", appendTime , "milliseconds")
        t3 = Timer("comprehension()", "from __main__ import comprehension")
        compTime= t3.timeit(number=1000)
        print("comprehension ", compTime, "milliseconds")
        t4 = Timer("rangeFunction()", "from __main__ import rangeFunction")
        rangeTime = t4.timeit(number=1000)
        print("list range ",rangeTime, "milliseconds")


        fig = plt.figure()
        ax = fig.add_axes([0,0,1,1])
        langs = ['concatination', 'append', 'comprehension', 'Range']
        students = [concatTime,appendTime,compTime,rangeTime]
        ax.bar(langs,students)
        plt.show()
```

Task 3: Simulation

Write code to carry out similar simulation for the following functions discussed in last class:

- ex1(n)
- ex2(n)
- ex3(n)
- ex4(n)
- ex5(n)
- ex6(n)
- ex7(n)

Task 4: Best, Worst and Average Case Evaluation

Write a python function that creates a list L of size 1000 where value stored at L[0]=0, L[1]=1 .... and L[999]=999. The elements are then randomly shuffled. Now search for the element 50.
Perform a simulation to find the average, maximum and minimum number of iterations to it which correspond to average, worst and best case scenarios.