LABORATORY 4 : Running time of an algorithm

OBJECTIVES

- to understand effect of loops
- to estimate running time of an algorithm
- to measure actual running time of an algorithm

BACKGROUND

1. Running time
   Set aside machine speed, running time of a program depends on number of steps the program executes and input size. The more number of execution steps, the longer it takes to finish. The larger the input, the more time it takes to process.

2. Running time analysis
   The goal of running time analysis is to obtain a "pen and paper" estimate of how efficient an algorithm or a program or a data structure is.
   Running time can be simply estimated by counting number of operations the program performs. Most of the time, one line of code is equivalent to one operation.

3. Loops
   Loops hide operations. Counting number of lines is not appropriate for running time estimate when loop is present.
   Loops can be nested. Dependency between loops effects number of times loops repeat.

4. Asymptotic bounds
   Big-O (O) describes what is called an asymptotic upper bound. Rigorously, we can say that $f(x)$ is an asymptotic upper bound of $g(x)$ if and only if there exist two constants, c and n_0, such that $c*f(x) \geq g(x) \geq 0$ for all values of x greater than n_0. If $f(x)$ is an asymptotic upper bound of $g(x)$ this can be denoted by writing $g(x) = O(f(x))$.
   Basically, an asymptotic upper bound of a function is any function which eventually becomes greater than or equal to that function and stays greater than it forever, if it is multiplied by some arbitrary constant.
   It is important to note that even if $f(x)$ is an upper bound of $g(x)$, $f(x) < g(x)$ can hold for all x. Because big-O denotes an upper bound, it should be used to describe the worst-case running of an algorithm.

   Common classes of functions that are encountered when analyzing the running time of an algorithm include O(1), O(n), O(nlogn), $O(n^2)$, $O(n^3)$ and O(n!).

   Other asymptotic bounds are big-Omega (Ω), big-Theta (Θ), little-Omega (ω) and little-O (o).

LABORATORY 4: Pre-lab

1. Read Chapter 2 Analysis (Miller's textbook : Problem Solving with Algorithms and Data Structures) or Chapter 3 Algorithm Analysis (Goodrich's textbook : Data Structures and Algorithms in Python).

2. Search the www on how to record execution time of a Python program / function. There are many ways. You should understand at least one way.
3. Study `matplotlib.pyplot` from https://matplotlib.org/users/pyplot_tutorial.html enough for you to plot an x-y graph, with many lines (from different datasets) in one graph. Use color / symbol to differentiate each of the lines.

LABORATORY 4: In-lab, Post-lab

1. Loops

   1.1. With algorithms `m01 – m12` provided in pseudocode format, analyze (on paper) growth rate of running time of each algorithm finding Big-O.

   1.2. Implement algorithms `m01 – m12` in Python.

   1.3. Write a main function to invoke `m01 – m12` for different values of *n*.

   1.4. For each function (`m01 – m12`), record the execution time, plot the running time (using `pyplot`) and see if it confirms the analysis you did in Pre-lab #2.

   You may repeat each function for a number of times. Then, find the mean execution time.

   Something to think about …
   - what should be values of *n* ? should these values be equal in `m01–m12`
   - how many times you should repeat each function ?
   - should you consider finding trimmed mean instead of arithmetic mean ?

2. Closest points in 2D plane

   Given *n* points in a 2-D x/y plane,

   2.1. write your <u>most efficient</u> `findClosest` algorithm to find the pair of points that are closest to each other

   2.2. analyze the runtime (Big-O, tightest upper bound) of your algorithm

   2.3. use the `Point` class and write a Python program to find the closest pair

   2.4. use random number generator to randomly assign value to each point

   2.5. try at total of 100, 500, 1000, 5000, 10000, 50000 and 100000 points

   2.6. time your `findClosest` method, record the elapsed times and plot them against number of points

   2.7. compare your analysis (2.2) with the actual running time plots. Explain what you found

Submission:

Due dates:

as said in Canvas

Submit your work in lab periods. You are to demonstrate your test plan and program. Prepare to answer questions individually.

```
ALGORITHM m01 is
    Operation : simple loop
    Input  : an integer, n
    Output :

    SET rounds to 0
    SET sum to 0
    WHILE rounds is less than n
        increment sum by 1
        increment rounds by 1
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m02 is
    Operation : simple loop increment by 2
    Input  : an integer, n
    Output :

    SET rounds to 0
    SET sum to 0
    WHILE rounds is less than n
        increment sum by 1
        increment rounds by 2
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m03 is
    Operation : independent nested loops
    Input  : an integer, n
    Output :

    SET round1s to 0
    SET sum to 0
    WHILE round1s is less than n
        SET round2s to 0
        WHILE round2s is less than n
            increment sum by 1
            increment round2s by 1
        ENDWHILE
        increment round1s by 1
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m04 is
    Operation : independent nested loops
    Input  : an integer, n
    Output :

    SET round1s to 0
    SET sum to 0
    WHILE round1s is less than n
        SET round2s to 0
        WHILE round2s is less than n
            increment sum by 1
            increment round2s by 10
        ENDWHILE
        increment round1s by 20
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m05 is
    Operation : two simple loops
    Input  : an integer, n
    Output :

    SET round1s to 0
    SET sum to 0
    WHILE round1s is less than n
        increment sum by 1
        increment round1s by 1
    ENDWHILE
    SET round2s to 0
    WHILE round2s is less than n
        increment sum by 1
        increment round2s by 1
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m06 is
    Operation : nested independent loops
    Input  : an integer, n
    Output :

    SET round1s to 0
    SET sum to 0
    WHILE round1s is less than n
        SET round2s to 0
        WHILE round2s is less than n*n
            increment sum by 1
            increment round2s by 1
        ENDWHILE
        increment round1s by 1
    ENDWHILE
    RETURN sum
```

```
ALGORITHM m07 is
     Operation : nested dependent loops
     Input  : an integer, n
     Output :

     SET round1s to 0
     SET sum to 0
     WHILE round1s is less than n
         SET round2s to 0
         WHILE round2s is less than round1s
             increment sum by 1
             increment round2s by 1
         ENDWHILE
         increment round1s by 1
     ENDWHILE
     RETURN sum
```

```
  ALGORITHM m08 is
     Operation : nested dependent loops
     Input  : an integer, n
     Output :

     SET round1s to 0
     SET sum to 0
     WHILE round1s is less than n
         SET round2s to 0
         WHILE round2s is less than 100*round1s
             increment sum by 1
             increment round2s by 1
         ENDWHILE
         increment round1s by 1
     ENDWHILE
     RETURN sum
```

```
  ALGORITHM m09 is
     Operation : nested dependent and independent loops
     Input  : an integer, n
     Output :

     SET round1s to 0
     SET sum to 0
     WHILE round1s is less than n
         SET round2s to 0
         WHILE round2s is less than n*n
             SET round3s to 0
             WHILE round3s is less than round2s
                 increment sum by 1
                 increment round3s by 1
             ENDWHILE
             increment round2s by 1
         ENDWHILE
         increment round1s by 1
     ENDWHILE
     RETURN sum
```

```
ALGORITHM m10 is
     Operation : simple quadratic loop
     Input  : an integer, n
     Output :

     SET rounds to 1
     SET sum to 0
     WHILE round1s is less than n
         increment sum by 1
         double the value of rounds
     ENDWHILE
     RETURN sum
```

```
  ALGORITHM m11 is
     Operation : simple quadratic loop
     Input  : an integer, n
     Output :

     SET i to n
     SET sum to 0
     WHILE i is greater than 0
         increment sum by 1
         half the value of i
     ENDWHILE
     RETURN sum
```

```
  ALGORITHM m12 is
     Operation : simple loop
     Input  : an integer, n
     Output :

     SET rounds to 1
     SET sum to 0
     WHILE rounds is less than n
         increment sum by 1
         increase value of rounds 10 times
     ENDWHILE
     RETURN sum
```

```
class Point:
    def __init__(self, x_init, y_init):
        self.x = x_init
        self.y = y_init

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    def __repr__(self):
        return "".join(["(", str(self.x), ",", str(self.y), ")"])

    def __str__(self):
        return "(%s,%s)" % (self.x, self.y)

    def distance(self, other):
        # your code goes here
```