Homework 3: Total 60 points.

Deadline: Thursday, October 15th, 11:59pm.

In this assignment you will practice with Big-O (and Big-Omega and Big-Theta), analyze the running time of code and algorithms, create and then run empirical tests on a simple methods.

Part 1: Analyzing Running Time (20 points)

True/False. In a section labeled Part 1A in your HW3.pdf file, state whether each of the following equalities are true or false. You should use the ***True*** (not abused) meaning of Big-O (and Big-Omega), not just the tightest bound. You should put the number of the question followed by either the word True or False. One answer/line. eg.

1. True
2. True

a) True/False

1. $7n^5 + 4 = O(n^4)$
2. $8n + \log(n) = O(n^2)$
3. $n^n + 2^n = O(3^n)$
4. $n^3 + n^2 + n + 1 = O(n)$
5. $\frac{n^2}{2} + n + 2 = O(n^2)$
6. $n^2 * n^2 + n^2 = \Omega(n^2)$
7. $n^2 + n = \Omega(n^3)$
8. $\frac{n^5}{n^2} + n^2 = \Omega(n)$
9. $\log(\log n) = \Omega(\log n)$
10. $n + 1 = \Omega(1)$
11. $n^2 - 100n - 1000 = \Theta(n)$
12. $n^2 + 4n + 4 = \Theta(n^3)$
13. $100n + \sqrt{n} + 4 = \Theta(n)$
14. $2n * n + 10n + 100 = \Theta(n^2)$
15. $n^2 + log(n) * n^2 = \Theta(n^2)$

b) Prove your answer for 11-15, using the definition of Big-Theta.
Clearly state what c and n_0 you choose for True answer or explain why such constants can't be chosen.

Part 2: Analyzing Running time for simple programs. (20 points)

In this part, you will practice your skills of estimating running time of code snippets. For each piece of code below, state the running time of the snippet in terms of the loop limit variable, *n*. You should assume that all variables are already declared and have a value. You should express your answer using Big-O or Big-Θ (Theta) notation, though your Big-O bound will only receive full credit if it is a **tight** bound. We allow you to use Big-O because it is often the convention to express only upper bounds on algorithms or code, but these upper bounds are generally understood to be as tight as possible. In all cases, your expressed bounds should be simplified as much as possible, with no extra constant factors or additional terms (e.g. O(n) instead of O(2n+5))

For each piece of code, state the running time and then give a short explanation of why that running time is correct. Your explanation should include an (approximate but reasonable) equation for how many instructions are executed, and then a relationship between your equation and your stated bound. Place your answers in your **HW3.pdf** file in a section labeled Part 2.
Here is an example:

Example.
```
for (i = 5; i < n; i++ )
 sum++;
```

Most precise answer:
Running time O(n)
Explanation: There is a single loop that runs n-5 times. Each time the loop runs it executes 1 instruction in the loop header and 1 instruction in the body of the loop. The total number of instructions is 2*(n-5) + 1 (for the last loop check) = 2n-9 = O(n). (also OK: Θ(n)).

A slightly less precise but still acceptable for full credit answer:
Running time: O(n).
Explanation: There is a single loop that runs n-5 times. Each time the loop runs it executes 1 instruction, so the total number of instructions executed is 1* (n-5) = O(n) (also OK: Θ(n)).

```
1) num=0;
   for (i = 1; i<=2*n; i++)
      num++;

2) num=0;
   for (i = 1; i<=n*n; i++)
      num++;


3) num=0;
   for (i = 1; i<=n; i++)
      num = num + n;
```

4) num=0;
```
num=0;
for (i = 1; i<=n; i++)
  for (j = 1; j<=i; j++)
   num = num + i;
```

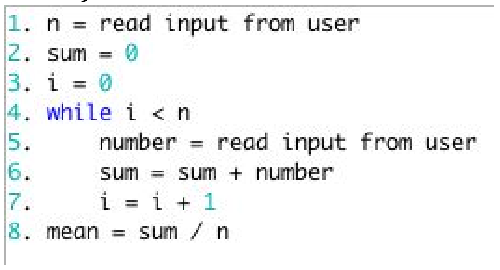
5)
```
num=0;
for (i = 1; i<=100; i++)
  for (j = 1; j<=n; j++)
    num = num + i;
```

6)
```
num=0;
for (i = 1; i<=n; i++)
  for (j = 1; j<=n; j=j*2)
    num = num + i;
```

7)
```
//not nested
for (i = 1; i<=n; i++)
     num++;
for (j = 0; j<=2*n; j++)
     num++;
```

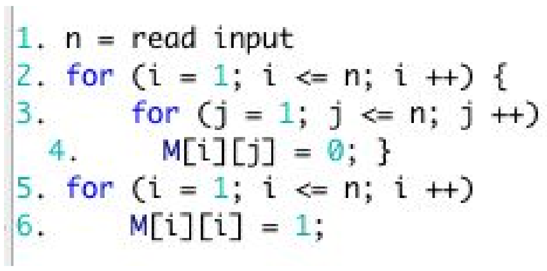
8)
```
for  (i=1; i<500; i++)
 num++;
```

9)



```
n = read input from user
sum = 0
i = 0
while i < n
    number = read input from user
    sum = sum + number
    i = i + 1
mean = sum / n
```

10)



```
n = read input
for (i = 1; i <= n; i ++) {
     for (j = 1; j <= n; j ++)
       M[i][j] = 0; }
for (i = 1; i <= n; i ++)
     M[i][i] = 1;
```

Part 3: In this part you will perform a Runtime Exploration (20)

This problem asks you write two methods and then run a few experiments to measure and compare their running time.

Directions:

1) Create two methods (if you need more parameters, you can add them):
    a. addFrontArray(int [] arr)
    b. addFrontList (LinkedList list)

both of the methods add random integers to the FRONT of either array or linked list. You can use Java's collection LinkedList.

How to measure running time:

*/** Returns the current time in milliseconds. */*

static long System.currentTimeMillis()

/** Returns the current value of the most precise available system timer, in nanoseconds */

static long System.nanoTime()

- If the algorithm can take less than a millisecond use System.nanoTime() !

2) To perform the experiments, you should generate the following tests:

    a. To compute the running time of an addFrontArray method generate 10 arrays of size *n*, compute the running time on each array, and take the average of the 10 running times. You do this to get more accurate running times.

    You need to choose *n* 10 times. It is up to you what *n* to choose but the results should meaningful.

    For example:
    Pick n=10, measure the running time (repeat 10 times to get an average)
    Pick n=20, measure the running time (repeat 10 times to get an average)
    ….
    Pick n=100, measure the running time (repeat 10 times to get an average)

    You probably want to increase n, to see meaningful times.

    b. To compute the running time of an addFrontList() repeat the same steps as above and FOR THE SAME VALUES of n.

You will therefore get 10 running times from each experiment. Plot these running times using your favorite plotting software (Excel, Mathematica, etc.) and connect these points with a smooth curve. You should have 2 plots. Take a screenshot of these plots and add it to your HW3.pdf in a section labeled Part 3.

Write 3-4 sentences explaining what you see based on each experiment and also your answers in your **HW3.pdf** file in a section labeled Part 3.

Turning in your assignment

For this assignment you will only turn in your **HW3.pdf** file using vocareum.
Make sure you hit submit button to submit your code.