Homework 3: Runtime

Deadline: Wednesday, January 27th, 11:59pm

Overview:

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 simple methods.

Part 1: Analyzing Running Time	(20 points)
a. True/False	* (15 points)
b. Short Answer	* (5 points)
Part 2: Analyzing Running Time for Simple Programs	(20 points)
Part 3: Running time of Linked List and Array List functions	(20 points)
Part 4: In this part you will perform a Runtime Exploration	(20 points)
Total:	**(80 points)**

Turning in Your Assignment:

Required files:

HW3_1.txt
HW3_2.pdf
HW3_3.txt
HW3_4.txt
Plots.pdf
Project3.java

Make sure you hit **submit** button to submit your code.

Part 1: Analyzing Running Time

a. True/False

In a section labeled Part 1A in your **HW3_1.txt** file, state whether each of the following equalities are true or false.
You should put the number of the question followed by either the word True or False. One answer/line. eg:

```
1. True
2. True
```

You should use the ***true*** (not abused) meaning of Big-O (and Big-Omega), not just the tightest bound.

Questions:

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. Explanation

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.

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))

Answer format:

Place your answers in your **HW3_2.pdf** file.

For each piece of question, state the running time of the code snippet 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.

Example Question:

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

Example 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: $\Theta(n)$).

Questions:

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;
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: Running time of Linked List and ArrayList functions

In this section, you will analyze minimum time needed to implement several operations on a Linked List and Array List.

In your **HW3_3.txt** file, in a section labeled Part 3, give the minimum required running time to execute each of the following operations in a doubly linked list with head and tail pointers, and Array List *in the worst case*, using Big-Ω notation, assuming n is the number of elements in the list. Following each expression, include a 1-2 sentence argument about why an algorithm to perform the given operation could not run faster than the bound you give. As above, full credit will be given only for tight Big-Ω bounds. That is, it is not sufficient to say that all operations take Ω(1). This is trivially true for any piece of code. (Though in some cases this will be the tightest Big-Ω bound).

Example Question:
Adding a value to the start of the list.
Example Answer:
Running time: Ω(1). You have a pointer to the head node in the list, so adding an element involves creating a new node (which is not dependent on the length of the list), setting the links in the new node, and changing the values of the head references. This takes somewhere around 10 steps to perform, and 10=Ω(1).

Example Question:
Adding a value to the start of the array list.
Example Answer:
Running time: Ω(n). You have to move everything over one cell to create a room for a new element. It will be done using a for loop that iterates through all elements. Since there are n elements in the list, it takes Ω(n) to add a new element to the beginning of the array list.

Questions (refers to both Linked list and Array List:

1. Making a copy of the list.
2. Adding a value to the end of the list.
3. Removing the first value from the list.
4. Removing the last value from the list.
5. Determining whether the list contains some value V.

Part 4: In this part you will perform a Runtime Exploration

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

How to measure running time

Returns current time in milliseconds:

```
static long System.currentTimeMillis()
```

Directions:

1) In a file named **Project3.java**, create two methods:

- addManyFrontArray(int [] arr)
- addManyFrontList (LinkedList list)

These methods will add random integers to the **front** of an array or linked list passed as an argument, respectively. You can use Java's collection Linked List or use your own implementation from homework 2.

Example: if you add number 1, 3, 5, 7 then the resulting list is going to be 7, 5, 3, 1.

2) You will run your methods on various n times and then calculate the average for these runs for more accurate data.

a. Generate 10 arrays of size *1000* using addManyFrontArray, compute the running time on each generated each array, and take the average of the 10 running times. **Record this value.**
b. Repeat on n of the following sizes:
[2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000]
c. Repeat the parts a and b on addManyFrontList.

3) Using the values from the above experiments create two plots. One plot will contain the average runtimes for increasing values of n for addManyFrontArray. The other plot will do the same for addManyFrontList. For each plot, the 10 data points will be connected by a smooth curve. The y-axis will be n and the x-axis will be time. You may use any software you wish, but you will be penalized if your graphs are not clear. Screenshot these graphs and include them in **plots.pdf.**

4) **In HW3_4.txt**, write a 3-4 sentences explaining the results of your experiment. Clearly explain the expected (theoretical) running time and the observed running times.