Hunter Havard / Victor Ejiasi

CSCI 3302 Pair Programming Assignment 04 (100 Points)

Due: Oct 27, 8:00 AM

GitHub Link: NONE

Objectives:

- Demonstrate understanding of algorithm analysis and Big-O.
- Demonstrate how to compare algorithms experimentally.

Assignment Assistance:

- This homework assignment is due before the date and time specified above.
- You and your partner must work on this TOGETHER.
- This assignment is restricted to you and your partner. You may not receive help from any other person except the instructor or the AARC (help from the AARC must be well documented!).
- Any resource used (other than Dr. Becnel or the course text) must be documented in the code (as comments) detailing the source and describing exactly what was learned and how that information was used. Submissions will be severely penalized if copied in part or in whole from any source.
- If you need help, visit your instructor during his posted office hours. If your schedule cannot accommodate any of these times, then email your instructor to schedule a different time.

Instructions:

Answer the following questions. Show work and include explanations where appropriate. You can handwrite or type your answers. To type equations in MS Word, use Alt + =.

1. List the following from slowest growth rate to highest growth rate.

$$n!, \quad n\log n, \quad n^2, \quad 4^n, \quad n+1000, \quad 2^n+n^3$$

- $n+1000$ - Linear function. It's n with a constant added.
- 4^n - Another linear function, just four times steeper than n.
- $n\log n$ - Log-linear function. Grows faster than any linear function but slower than any polynomial function of n^2or higher.
- n^2- Quadratic function. Takes off faster than $n\log n$ as n grows.
- 2^n+n^3- Cubic term dominates, so this is essentially cubic growth.
- $n!$ - Factorial function. This one grows insanely fast, way beyond any polynomial or exponential function.

2. Using the results from class find the most appropriate O for the following
   a. $3n^2 + 100n + n^4$

$O(n^4)$, the term n^4 is going to be the dominant one as n approaches infinity. All the other terms become negligible in comparison.

   b. $10n + \log(2n + 1) + 1.5^n$

$O(1.5^n)$, the exponential term 1.5^n will outgrow all the other terms as n goes to infinity.

   c. $(2n + 2)(n^2 + 2n + 1)$

$O(n^3)$, since the expanded equation $2n^3 + 4n^2 + 2n + n^2 + 2n + 1$ simplifies to $2n^3 + 5n^2 + 4n +$ 1the dominant term is $2n^3$.

   d. $(2^n + 10n)(3^n + 2^n)$

$O(6^n)$, since the expanded equation $2^n \times 3^n + 2^n \times 2^n + 10n \times 3^n + 10n \times 2^n$ simplifies to $6^n +$ $2^{2n} + 30n \times 3^n + 20n \times 2^n$ the dominant term is 6^n.

3. Exactly how many times will “Hello” print in the following? You can check your answer by running the code block with an added count variable but should make a hypothesis first and have accompanying work showing how you arrived at the hypothesis.

   a.
```
for (int i = 0; i < 100; i++)
    System.out.println(x:"Hello");
```

In this loop, "Hello" will print 100 times. It starts from 0 and goes up to 99.

   b.
```
for (int i = 0; i < 100; i=i+3)
    System.out.println(x:"Hello");
```

"Hello" will print when i is 0, 3, 6, ..., up to 99. That's a total of 34 times (including the start at 0).

   c.
```
for (int i = 0; i < 100; i++)
    for (int j =0; j < 100; j++)
        System.out.println(x:"Hello");
```

In this nested loop, "Hello" will print $100 \times 100 = 10{,}000$ times.

d.
```
for (int i = 0; i < 100; i++)
    for (int j =0; j < i; j++)
        System.out.println(x:"Hello");
```

this loop will print "Hello" for every pair (i, j) where $0 \leq j < i < 100$. This would be the sum of the integers from 0 to 99, which is $299 \times 100 = 4{,}950$.

4. Find the most appropriate O growth rate for the following code blocks. Include accompanying work to justify your solution. Your final answer should be in terms of n, for example, $O(n^3)$:

a.
```
for (int i =0; i < n; i++)
    System.out.println(x:"Hello");
```

This loop iterates n times. So, its time complexity is $O(n)$.

b.
```
 for (int i =0; i < n; i++)
     for (int j =0; j < n; j++)
         System.out.println(x:"Hello");
```

Here, we have a nested loop where both inner and outer loops iterate n times. This gives us $O(n \times n) = O(n^2)$.

c.
```
for (int i = 0; i < n; i++)
    for (int j =0; j < i; j++)
        System.out.println(x:"Hello");
```

For this block, the inner loop iterates i times, where i ranges from 0 to $n - 1$. The total number of iterations would be $0 + 1 + 2 + \cdots + (n - 1) = \frac{n(n-1)}{2}$. So, the time complexity is $O(n^2)$.

d.
```
int i = 1;
while (i < n) {
    System.out.println(x:"Hello");
    i = 2 * i;
}
```

Here, i is doubled in each iteration. So the loop will run $\log_2 n$ times. So, the time complexity is $O(\log n)$.

e.
```
int i = 1;
while (i < n) {
    for (int j = 0; j < n; j++)
        System.out.println(x:"Hello");
    i = 2 * i;
}
```

The inner loop runs n times, while the outer loop runs $\log_2 n$ times. So, the total time complexity is $O(n \log n)$.

5. Implement two versions for an algorithm to find the nth Fibonacci number. One should use recursion as we discussed previously in class. The other should use a loop. The loop version is below. Note: make sure to change the recursive version to use **long** and not **int** data types.

```
private static long fib2(long n) {
    long prev = 1;
    long current = 1;
    for (int i = 2; i <= n; i++) {
        long temp = current;
        current = prev + current;
        prev = temp;
    }
    return current;
}
```

You will use both algorithms to compute the 10^{th}, 20^{th}, 30^{th}, $40^{th,}$ and 50^{th} Fibonacci numbers. Create a table of your results, such as the one below.

N	Time to Compute Recursively	Time to Compute with Loop
10	2823 ns	1079 ns
20	2698 ns	1046 ns
30	2398 ns	1115 ns
40	2433 ns	1326 ns
50	2539 ns	1640 ns

ns = nanoseconds
A nanosecond is 10^9 seconds.
To convert the nanosecond timing to scientific notation, you can divide by 10^9 to convert it to seconds and then use scientific notation for that value.
For example, if the time in nanoseconds is 3000, then in seconds it would be 3×10^{-6}.

Based on the table data, which algorithm is more efficient for computing Fibonacci numbers?

Although both algorithms are efficient and the differences in time are in the nanoseconds, the loop based method is still faster for each value of N we tested. This means the loop based algorithm is more efficient for computing Fibonacci numbers based on the data we collected.

SUBMISSION:

- Only one person needs to turn in/commit the assignment. However, both members of the team are expected to understand the solutions and be able to answer questions about the solution.
- Review the Evaluation below to ensure you have met all the requirements.
- Put your and your partner's names on your answer sheet and upload it to D2L. Also, turn in your Java file from question 5.

EVALUATION

a) Problem 1: 10 points; partial credit is given.
b) Problem 2: 20 points (5 points each).
c) Problem 3: 20 points (5 points each).
d) Problem 4: 25 points (5 points each).
e) Problem 5: 15 points (10 points for table; 5 points for correct answer).
f) Miscellaneous: 10 points for neatness and following directions.