Complexity Analysis

7. What is the time complexity of the following function?

```
int question_7(int n) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        for (int j = i; j > 0; j--) {
            count += 1;
        }
    }
    return count;
}
```

A) $\Theta(\log n)$
B) $\Theta(n)$
C) $\Theta(n \log n)$
D) $\Theta(n^2)$
E) none of the above

8. What is the time complexity of the following function?

```
int question_8(int n) {
   int r = 0;
   while (n > 1) {
       n /= 2;
       ++r;
   }
   return r;
}
```

A) $\Theta(\log n)$
B) $\Theta(n)$
C) $\Theta(n \log n)$
D) $\Theta(n^2)$
E) none of the above

9. What is the time complexity of the following function?

```
int question_9(int n) {
   int count = 0;
   int m = static_cast<int>(floor(sqrt(n)));
   for (int i = n/2; i < n; i++) {
       for (int j = 1; j < n; j = 2 * j) {
           for (int k = 0; k < n; k += m) {
                ++count;
                std::cout << "hello world" << std::endl;
           }
       }
   }
   return count;
}
```

A) $\Theta(n^{1/2} \log n)$
B) $\Theta(n \log n)$
C) $\Theta(n^{3/2} \log n)$
D) $\Theta(n^2 \log n)$
E) $\Theta(n^{5/2} \log n)$

10. Refer to the code in question 9. Let *a* be the number of times `hello world` is printed when *n* has a value of 16, and let *b* be the number of times `hello world` is printed when *n* has a value of 6. What is the value of $a - b$?

A) 37
B) 46
C) 69
D) 101
E) 116

Recurrence Relations

24. What is the complexity of the following recurrence relation?

$$T(n) = \begin{cases} c_0, & n = 1 \\ 3T(n-1) + c, & n > 1 \end{cases}$$

A) $\Theta(n)$
B) $\Theta(n^2)$
C) $\Theta(n^3)$
D) $\Theta(2^n)$
E) $\Theta(3^n)$

25. What is the complexity of the following recurrence relation?

$$T(n) = \begin{cases} c_0, & n = 1 \\ 4T\left(\frac{n}{2}\right) + 16n + n^2 + c, & n > 1 \end{cases}$$

A) $\Theta(n)$
B) $\Theta(n \log n)$
C) $\Theta(n^2)$
D) $\Theta(n^2 \log n)$
E) $\Theta(n^4)$

26. What is the complexity of the following recurrence relation?

$$T(n) = \begin{cases} c_0, & n = 1 \\ 5T\left(\frac{n}{25}\right) + \sqrt{n} + c, & n > 1 \end{cases}$$

A) $\Theta(\sqrt{n})$
B) $\Theta(\sqrt{n} \log n)$
C) $\Theta(n)$
D) $\Theta(n^5 \log n)$
E) $\Theta(n^5)$

27. What is the complexity of the following recurrence relation?

$$T(n) = \begin{cases} c_0, & n = 1 \\ 729T\left(\frac{n}{9}\right) + 3n\sqrt[3]{n} + 81n + c, & n > 1 \end{cases}$$

A) $\Theta(\sqrt[3]{n} \log n)$
B) $\Theta(\sqrt[3]{n})$
C) $\Theta(n)$
D) $\Theta(n\sqrt[3]{n})$
E) $\Theta(n^3)$

Mastering the Master Theorem

28. Which of the following recurrence relations can one solve by applying the Master Theorem?

A) $T(n) = nT\left(\frac{n}{3}\right) + \Theta(n^2)$

B) $T(n) = 24T\left(\frac{n}{6}\right) + 32T\left(\frac{n}{8}\right) + \Theta(n^2)$

C) $T(n) = 11T\left(\frac{n}{13}\right) + 23\left(\frac{n^{log_3 9}}{e^{2\pi}\sqrt{n}}\right) + \Theta(n\sqrt[4]{n})$

D) $T(n) = 2T(n-2) + \Theta(n^{55})$

E) none of the above can be solved using the Master Theorem

Identifying the Recurrence

29. Given the function below, calculate the recurrence relation. Assume that `cake(n)` runs in $\log n$ time.

```
void pie(int n) {
  if (n == 1) {
    return;
  }

  pie(n / 7);

  int cookie = n * n;

  for (int i = 0; i < cookie; ++i) {
    for (int j = 0; j < n; ++j) {
      cake(n);
    }
  }

  for (int k = 0; k < n; ++k) {
    pie(n / 3);
  }

  cake(cookie * cookie);
} // pie()
```

A) $T(n) = T\left(\frac{n}{7}\right) + n^2 \log n + nT\left(\frac{n}{3}\right) + \log n$

B) $T(n) = T\left(\frac{n}{7}\right) + n^2 \log n + nT\left(\frac{n}{3}\right) + 2\log n$

C) $T(n) = T\left(\frac{n}{7}\right) + n^3 \log n + nT\left(\frac{n}{3}\right) + \log n$

D) $T(n) = T\left(\frac{n}{7}\right) + n^3 \log n + nT\left(\frac{n}{3}\right) + 2\log n$

E) $T(n) = T\left(\frac{n}{7}\right) + n^3 \log n + nT\left(\frac{n}{3}\right) + 4\log n$

When Push Comes to Shove

32. Suppose you are implementing a vector with an underlying dynamic array, but with a special pushing method known as `shove_back()`. This method works normally when the array is not full (mimicking `push_back()`). When the array fills up, however, the `shove_back()` method creates a new dynamic array that is double the size of the old array, copies the elements from the old array to the new array, sorts the new array, and then deletes the old array. What is the amortized time complexity of the `shove_back()` operation?

A) $\Theta(1)$

B) $\Theta(\log n)$

C) $\Theta(n)$

D) $\Theta(n \log n)$

E) $\Theta(n^2)$

Hint: you may assume that sorting is a O(n log n) process

Autograder Blues

143. Your friend is currently over time on Project 1. Which of the following is most likely not a reason why this may be occurring?

- **A)** your friend is not using the STL
- **B)** your friend's incorrect error checking is causing their program to terminate early
- **C)** your friend is not passing large objects by reference
- **D)** your friend is storing unnecessary information
- **E)** none of the above

Real-Time Measurements

144. After timing your project implementation, you realized that your system time was considerably larger than your user time. Which of the following could be a reason why this may be happening?

- **A)** your program leaked memory
- **B)** your program was not allocated enough CPU
- **C)** your program spent a lot of time doing complex mathematical calculations
- **D)** your program spent a lot of time iterating through large containers
- **E)** your program spent a lot of time doing file and stream I/O

Running Out of Time

145. Suppose you timed a program and obtained the following data:

```
user time: 9.032s
system time: 7.974s
elapsed time: 20.019s
```

The percentage of CPU that this program received is closest to

- **A)** 40%
- **B)** 45%
- **C)** 80%
- **D)** 85%
- **E)** 99%

Perf Perfection

Your friend is currently implementing Project 3 from the Winter 2018 semester, which involves building a database system similar to SQL. However, their implementation is consistently over time! To fix this issue, your friend decides to run `perf` on their program. However, they aren't too familiar with how `perf` works, so they send the output to you. The outcome of this `perf` report is shown below:

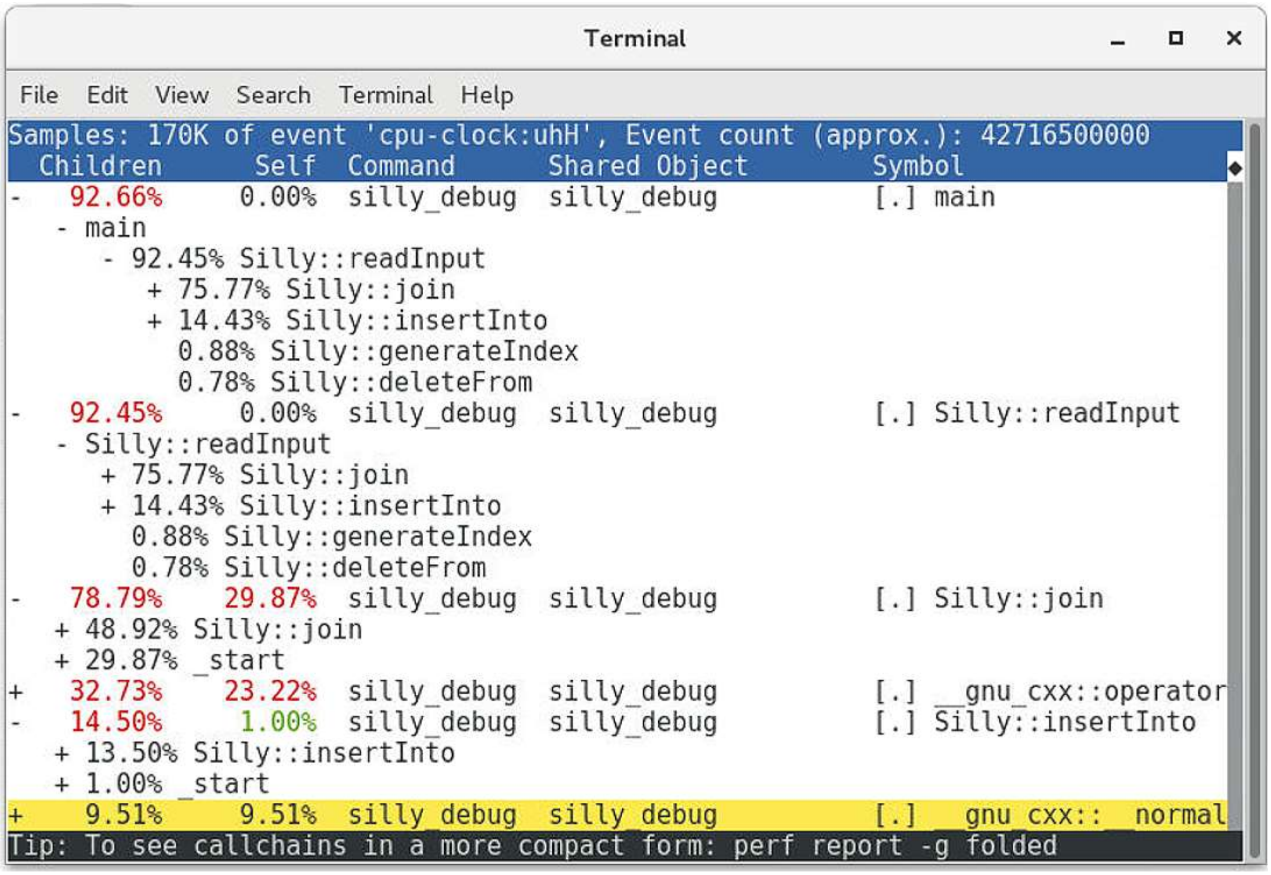
EECS 281 Winter 2018
Project 3: SillyQL

Due Tuesday, March 27th 2018 at 11:59 pm
Checkpoint due: Friday, March 16th 2018 at 11:59 pm

Overview

A relational database is the most common means of data storage and retrieval in the modern age. A relational database model stores data into one or more tables, where each table has columns and rows. The columns represent a value that can be attributed to an entry (such as "color", "name", or "ID") and the rows represent individual entries or records (such as "Paoletti", "my cat", or "BBB 1695"). You may find it helpful to think about rows as

Cars marketplace			
vendor	Model	Price	Mileage
Chevrolet	Corvette	17226	25965.0
Chevrolet	Corvette	34229	46429.0
Chevrolet	Corvette	27982	50209.0
Chevrolet	Corvette	51825	72998.0
Chevrolet	Corvette	52845	34364.0
Chevrolet	Malibu	37874	37273.0
Chevrolet	Malibu	35600	71441.0
Chevrolet	Malibu	52447	46700.0
Chevrolet	Malibu	27129	36254.0



```
Samples: 170K of event 'cpu-clock:uhH', Event count (approx.): 42716500000
  Children      Self  Command      Shared Object         Symbol
-   92.66%     0.00%  silly_debug  silly_debug           [.] main
   - main
      - 92.45% Silly::readInput
         + 75.77% Silly::join
         + 14.43% Silly::insertInto
           0.88% Silly::generateIndex
           0.78% Silly::deleteFrom
-   92.45%     0.00%  silly_debug  silly_debug           [.] Silly::readInput
   - Silly::readInput
      + 75.77% Silly::join
      + 14.43% Silly::insertInto
        0.88% Silly::generateIndex
        0.78% Silly::deleteFrom
-   78.79%    29.87%  silly_debug  silly_debug           [.] Silly::join
   + 48.92% Silly::join
   + 29.87% _start
+   32.73%    23.22%  silly_debug  silly_debug           [.] __gnu_cxx::operator
-   14.50%     1.00%  silly_debug  silly_debug           [.] Silly::insertInto
   + 13.50% Silly::insertInto
   + 1.00% _start
+    9.51%     9.51%  silly_debug  silly_debug           [.] __gnu_cxx::__normal
Tip: To see callchains in a more compact form: perf report -g folded
```

146. Approximately what percentage of total time is spent in the `Silly::join()` function? For this calculation, also include functions that are called within `Silly::join()`.
    - **A)** 10%
    - **B)** 30%
    - **C)** 50%
    - **D)** 80%
    - **E)** 90%

147. Given this `perf` report, what should your friend do first?
    - **A)** find ways to optimize the `Silly::join()` function
    - **B)** find ways to optimize the `Silly::insertInto()` function
    - **C)** find ways to optimize the `Silly::generateIndex()` function
    - **D)** find ways to optimize the `Silly::deleteFrom()` function
    - **E)** all of the above options are equally ideal