EECS 281 Lab 2

Due Friday 9/28 at 11:59pm

There is no pre-lab with Lab 2, but you should still read the lab and be prepared when you arrive. The Computing CARES group is doing a survey whose completion will be graded as part of this lab. The due date is also 9/28.

12 points for Canvas Quiz, 3 points for survey, 5 points for writing code in lab.

Problem 0: Survey (3 points)

As part of the Computing CARES initiative, we want everyone to complete their survey, located here. NONE of the EECS 281 staff will be looking at your responses! The only thing that we will receive from the CARES group is a list of uniqnames of students who have taken the survey. The very first page explains the survey and how your data will be used. On page 2, you can consent to allowing your answers to be used for research, or state that your answers cannot be used. No matter which you select, you will still be given credit for filling out the questions that follow.

Problem 1: Profiling (2 points)

Using `perf`

- Download `lab2.tar.gz` from Canvas, untar it with the command

  ```
  tar -xzvf lab2.tar.gz
  ```

- It will create a folder named `lab2`, with subfolders named `problem1` and `problem2`. Find the file `lab2/problem1/problem1.cpp`. Compile the code on CAEN (with `make`, the `Makefile` is already supplied in that folder ready to go) and then use the `perf` tool (on CAEN) to profile the program. Note that running `problem1` (including the `perf` command) will probably take around 10 seconds, so don't panic if it seems like it's not doing anything. If it finishes after less than 5 seconds, you issued the wrong command(s).
- Type the following commands to build the program, time it, and display the results (be sure that you use `make debug` and `./problem1_debug`, otherwise you will get incorrect results):

  ```
  make debug
  perf record -F 1000 --call-graph dwarf -e cycles:u ./problem1_debug <
  test-1.txt
  perf report
  ```

- When you're done looking at the report, type `q` to quit.

Canvas Questions

(2 questions, 1 point each) Choose the best answer for both questions. You may have to make your terminal window wider to see the full function name(s).

1.1 Approximately what percent of time is spent in the `insertionSort()` function? Make sure that you look at the "self" column of the output.

A. 10%
B. 20%
C. 40%
D. 60%
E. 80%

1.2 Approximately what percent of time is spent running the `vector` subscript operator (it will appear as `std::vector<...>::operator[]`)? Again, use "self" data.

A. 10%
B. 20%
C. 40%
D. 60%
E. 80%

When you're all done, be sure to remove the `perf.data` file: it can be almost 100MB in size! The `make clean` command included in the lab files will remove the executable, object files, and the `perf.data` file.

Problem 2: Time to Sort (5 points)

Below is an algorithm to sort a sequence of `N` integers, `N > 1`. The pseudocode uses 1-based indexing.

```
Algorithm sort0(a[], N):
    for i = 2 to N
        j = i
        while (j > 1) and (a[j - 1] > a[j])
            swap a[j] and a[j - 1]
            --j
```

Notice how each element `i` moves down in the correct position in the sequence. We can expand the `swap` operation in-place to get a new algorithm, `sort1`, that also uses 1-based indexing.

```
Algorithm sort1(a[], N):
      for i = 2 to N
            k = a[i]
            j = i - 1
```

```
while (j > 0) and (a[j] > k)
       a[j + 1] = a[j]
       --j
a[j + 1] = k
```

For full credit on this problem,…

- In `lab2.zip`, find `problem2/sorts.cpp` . In `sorts.cpp`, implement the `sort1` algorithm in the `sort1` function by converting pseudocode to C++.
- Compile `sorts.cpp` in CAEN environment **using `make sorts`** (the `Makefile` is provided and ready for use) **instead of `make`.**
- Run it to measure and record the execution time. Use the `time` utility that is built into the Linux command shell to time your program.
  To use the `time` utility in the command shell, type `time` before the execution of your program:

  ```
  time ./sorts
  ```

  or

  ```
  /usr/bin/time ./sorts
  ```

  For the standard bash shell, the output should look something like this:

  ```
  1.234user 0.001system 0:12.35elapsed 99.9%CPU
  ```

- Run it at least three times and calculate the average `user` time.
- Now, comment out the line that calls `sort1`, and uncomment out the next line that calls `sort2`. You are not required to understand how `sort2` was implemented.
- Re-compile and then re-run the program at least three times with `sort2` instead of `sort1`. Measure and record the average `user` time.
- Answer these questions on Canvas:
  (5 questions, 1 point each)

  Note: For questions 2.1 and 2.2, the value you enter must be within a fairly generous range of numerical values. As long as you're somewhat close you will get credit for your answer.

  2.1 How long (in seconds) did `sort1` take to run for `N = 40000`? _____

  2.2 How long (in seconds) did `sort2` take to run for `N = 40000`? _____

  2.3 The complexity of `sort1` is…
  A. $\Theta(n \log(n))$
  B. $\Theta(n^2)$
  C. $\Theta(n^3)$

2.4 The average-case complexity of `sort2` is…

Hint: you do not have to know the implementation details to answer this question. Check on cppreference.com or cplusplus.com and look up std::sort.

A. $\Theta(n \log(n))$
B. $\Theta(n^2)$
C. $\Theta(n^3)$

2.5 True or False - a $\Theta(n \log(n))$ sort always runs faster than a $\Theta(n^2)$ sort.

A. True
B. False

Problem 3: String versus C-string (2 points)

(2 questions, 1 point each)

3.1 Given the following piece of code, what is the output?

```
char *s = "hello";
char ss[20];
int length = strlen(s);
for (int i = 0; i < length; i++)
    ss[i] = s[length - i];
printf("%s", ss);
```

A. No output is printed
B. olleh
C. The code is buggy with out-of-bound access
D. hello

3.2 Given the following code, and `n` is `strlen(s)`, what is the time complexity of the code? Be careful that the code below is **not** identical to that in the previous problem.

```
char *s = "hello";
char ss[20];
for (int i = 0; i < strlen(s); i++)
    ss[i] = s[length - i];
printf("%s", ss);
```

A. $\Theta(n^2)$
B. $\Theta(n)$

Problem 4: *O*(MG) (3 points)

(3 questions, 1 point each)

Function Complexity

4.1 What is the complexity of this function?

```
void foo(vector<int> &v) {
    int n = static_cast<int>(v.size());
    int rt = static_cast<int>(floor(sqrt(n)));

    for (int i = 0; i < rt; ++i) {
        for (int j = 0; j < rt * rt; j += rt) {
            cout << v[j + i] << " ";
        }
        cout << endl;
    }
}
```

A. $\Theta(\sqrt{n})$
B. $\Theta(n)$
C. $\Theta(n\ \sqrt{n}\ )$
D. $\Theta(n \log_2 n)$
E. $\Theta(n^2)$

The Big Three of Complexity

4.2 Suppose you have some functions, *f*, *g*, and *h*. Which of the following statements is **false**?

A. If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$.
B. If $f = O(g)$ and $g = O(f)$, then $f = \Theta(g)$.
C. It is possible that f is $\Omega(f * g)$, where * is multiplication.
D. Either $f = O(g)$, $f = \Theta(g)$, or $f = \Omega(g)$.
E. If $f + g$ is $O(h)$, then it is possible that f is not $O(h)$.

Recurrence Complexity

4.3 Consider the following recurrence relation:

$$T(n) = \begin{cases} c_0, & n = 1 \\ \sqrt{2}T\left(\frac{n}{2}\right) + c_1, & n > 1. \end{cases}$$

What is the tightest complexity class that you can attribute to this recurrence relation?

A. $T(n) = \Theta(\sqrt{n})$
B. $T(n) = O(\sqrt{n})$
C. $T(n) = \Omega(\log_2 n)$
D. $T(n) = \Theta(n \log_2 n)$
E. $T(n) = \Theta(n^2)$