CSE/IT 122: Homework 3

Make sure all code follows Doxygen style comments and Linux Kernel coding style. All files mentioned are in the tarball on Canvas.

Problems

Running Time

For the following three problems, create and fill in a table that has a header consisting of the line of code, the cost, and the number of times the line executes. **Be exact in your counts**. Find the running time and simplify (i.e. group like terms) the expression for $T(n)$. Type your answers in Latex or Word, etc. and convert the file to a pdf file named `cse122_firstname_lastname_hw3.pdf`. For problems 1, 2, and 3 just show the table and the simplified answer. For problem 4, show your work of how you analyzed the problem.

1. Find the running time $T(n)$ of:

```
for i = 3 to n - 5
    x = 5 * i
```

2. Find the running time $T(n)$ of:

```
for i = 2 to n - 1
    i = i * i
    break
```

3. Find the running time $T(n)$ of:

```
for k = 1 to n - 1
    max = a[k]
    for j = k + 1 to n
        a[j] = a[j] * max
    a[k] = a[j]
```

4. Make sure to show your work for this problem on how you came up with the count for each line. Find the running time $T(n)$ of:

```
for i = 1 to n
    for j = 1 to i
        for k = 1 to j
            x = i * j * k
```

5. Define $f(n) = o(g(n))$ to mean that $\lim_{n\to+\infty} \frac{f(n)}{g(n)} = 0$

   Show that $log\, n = o(n^{\epsilon})$ for any $\epsilon > 1$. Hint: use l'Hospital's Rule. Show your work and type your answer.

Permutations

6. Suppose you need to generate a random permutation of the first n integers. For example, 5, 3, 2, 1, 4 is a permutation of the first 5 integers. Here are three algorithms to generate a permutation:

   (a) Fill the array `A` from `A[0]` to `A[n - 1]` as follows: To fill `A[i]`, generate random numbers until you get one that is not already in `A[0], A[1], ... , A[i - 1]`

   (b) Same as (a) except use an additional array that keeps track if a number has already been used or not. When a random number, `rand`, is placed into array `A`, you also set `used[rand] = 1`. This means before you fill `A[i]` with a random number, you can test whether the random number has already been used in constant time, rather than in i possible steps of (a).

   (c) Initialize the array so that `A[i] = i + 1`, then use the loop:

   ```
   for (i = 1; i < n; i++)
           swap(&a[i], &a[nrandint(n)]);
   ```

   where `nrandint(n)` generates a random integer between $[1, n]$. For the swap function, write a bit exclusive or swap.

   For each algorithm, you are going to time the results as in Homework 0. Run program (a) for $n = 250, 500, 1000, 2000$; program (b) for $n = 2500, 5000, 10000, 20000, 40000, 80000$; and program (c) for $n = 10000, 20000, 40000, 80000, 160000, 320000, 640000$. For each n, run the program 10 times, so you get a good average. Be patient. Algorithm (a) is really inefficient. Create a table of the results for each algorithm. Your table should capture the run, n, and the time. After each 10 runs, print out the average. For example, the first knuth timing data looks like this:

```
begin knuth runs

run     n       time
1       10000   0.0000
2       10000   0.0000
3       10000   0.0000
4       10000   0.0000
5       10000   0.0000
6       10000   0.0000
7       10000   0.0000
8       10000   0.0000
9       10000   0.0000
10      10000   0.0000

avg for n = 10000 for knuth is 0.0000 seconds

run     n       time
1       20000   0.0000
...
```

Report timing data to 4 decimal places only. The values in the table are tab separated.

On Canvas, you are given code to generate random integers between $[1, n]$ and a test program (`randint.c`, `randint.h`, and `randdemo.c`). Place all three algorithms in a single source code file called `permutation.c`. Name algorithm's (a) function `lucky`, algorithm's (b) function `used`, and algorithm's (c) function `knuth` as it comes from Donald Knuth of Stanford. Create a Makefile to compile `permutation.c`. Your `main()` should just setup the test runs and print the table and averages for you. Add a section name Permutations to the pdf file and copy the table from `stdin` into the pdf file. You should develop and test the algorithms for small cases of n before you run the tests for the larger n.

Make sure your program uses dyanmic memory allocation for all arrays, frees all memory, and your code does not leak memory. Test with valgrind.

Russian Peasant Algorithm

7. Write a program that implements the Russian Peasant Algorithm. For the Russian Peasant Algorithm see
   `http://mathforum.org/dr.math/faq/faq.peasant.html`
   or
   `http://en.wikipedia.org/wiki/Ancient_Egyptian_multiplication`

   Use redirection (see below) for input and to capture the output. Name your source code `russian.c`, the input file `russian.in` , and the output file `russian.out`

   In the input, the first value is m and the second value is n and you calculate $m \cdot n$. m and n are positive integers.

Sample Input

```
2 44
13 7
24 42
```

Sample Output

```
2 * 44 = 88
13 * 7 = 91
24 * 42 = 1008
```

Make sure you follow the formatting of the sample input and output exactly for this problem. Your problem is tested with `diff`. You will lose points if you do not follow the format exactly.

Redirection

In a *nix terminal, redirection lets you change the location of `stdin` and `stdout` to be a file rather than the terminal. This allows you to quickly process input files and write output files without any file I/O operations in your C programs.

To redirect `stdin` use the < operator.

```
$ ./russian < russian.in
```

To redirect `stdout` use the > operator.

```
$ ./russian > russian.out
```

You can combine both operations in one line, which is what you want to do if you need to redirect both `stdin` and `stdout`.

```
$ ./russian < russian.in > russian.out
```

In C, there is nothing special to do for redirection. For example, the following code is a simple program that copies a file (a simple `cp` program). Compiled and run with redirection the program will take the redirected input and write it to the redirected output. As long as the lines of the input file are less than 4096 characters, this will produce an exact copy of the input file.

```
/* simple copy of a file */
/* usage: copy < in > out */
#include <stdio.h>
#define MAX 4096
```

```
int main(void)
{
        char s[MAX];

        while(fgets(s, MAX, stdin)) {
                /* code to process string */
                printf("%s", s);
        }

        return 0;
}
```

Running the above program (named copy) like so:

```
$ ./copy < in > out
```

will copy the contents of `in` into a file named `out`.

For more information on redirection see Chapter 6 of the The Linux Command Line by W. Shotts. The book is available online at `http://linuxcommand.org/tlcl.php`

Submission

Tar your source code files, input files, output files, and your pdf file into a tarball named

`cse122_firstname_lastname_hw3.tar.gz`.

Upload the tarball to Canvas.