CSUC CSCI-311 - Algorithms and Data Structures

Lab Assignment 1

Goal: The goal of this lab is to start thinking in terms of C++ and to start to think about the importance of algorithm choice with respect to run time.

Submission: C++ solutions to these problems should be written in a file called `lab_1.cpp` (a skeleton C++ program file as well as the test cases are available on the corresponding Canvas link) and the file should be submitted on ingenious(https://inginious.csuchico.edu/, and you need to on campus network or vpn to use it. Simply signup on my course on ingenious and submit your code to the corresponding task labelled accordingly.).

Coding Style: Note that your submission should use good C++ coding style and should include appropriate comments (for readability and maintainability). Specifically, your code must follow common C++ coding conventions for Naming, Indentation, and Comments. Points will be deducted if these are not present.

Collaboration: There will be time in lab to discuss these problems in small groups and I highly encourage you to collaborate with one another outside of class. However, you must write up your own solutions **independently** of one another. In addition, do not post solutions in any way. Also, please include a list of the people you work with in a comment section at the top of your submission.

Have fun!

Assignment Date:	**Aug 26, 2024**
Due Date:	**11:59pm on Sep 01, 2024**
Grace Due Date:	**11:59pm on Sep 04, 2024**

Grading Notes: 1) Assignment is due on the Due Date. 2) For each day late after the Due Date, there will be 10% penalty on the assignment's grades. 3) Submission is not accepted after the Grace Date. In other words, you will receive zero pts if your submission is not received by the Grace Date.

Special Nodes: Before getting to Lab1, you must finish learning Lab0 to prepare for consequent lab assignments. Lab0 is available at the corresponding Canvas link in the lab assignment section.

Grading: Coding Style 5 pts, Test Cases 95 pts, Total 100 pts.

NOTE* YOU MUST DO ALL THE TESTINGS ON THE ECC-LINUX SERVER AND ONLY USE INGINIOUS FOR SUBMISSION GRADING! YOU CAN ONLY SUBMIT 3 TIMES ON INGINIOUS.

1. Given any positive integer n, define

$$f(n) := \begin{cases} n/2 & , n \text{ is even} \\ 3n+1 & , n \text{ is odd.} \end{cases}$$

The Collatz conjecture states that the sequence

$$a_i := \begin{cases} n & , i = 0 \\ f(a_{i-1}) & , i > 0 \end{cases}$$

eventually reaches 1. Write a function `int collatzLength(int n)` that determines the first i for which $a_i = 1$ for a given n. For example, when $n = 17$, the sequence a_i begins

17,52,26,13,40,20,10,5,16,8,4,2,1, …

and $a_{12} = 1$. Thus, `collatzLength(17)` should return 12.

2. Write a function `void printStats(const vector<int> &v)` that prints the minimum, mean, and maximum of the values in the given vector separated by spaces. For example,

```
vector<int> v = {1, 2, 3, 4, 5};
printStats(v);
```

should print

```
1 3 5
```

If the given vector is empty, your function should print `Empty vector`.

Note that passing the vector `v` by reference (as opposed to by value) is usually preferred. This saves both time and space. When the vector will not be changed by the function, include the keyword `const` as we have done here.

3. Write a function `int sumMultiples(const vector<int> &nums, int n)` that returns the sum of all multiples of values in the vector `nums` less than `n`. For example,

```
vector<int> nums = {3, 5};
int n = 30;
sumMultiples(nums, n);
```

should return 195. In particular, $195 = 3 + 5 + 6 + 9 + 10 + 12 + 15 + 18 + 20 + 21 + 24 + 25 + 27$. Assume that the elements of `nums` are positive integers.

4. Write a function `void greaterThanK(vector<int> &v, int k)` that removes all values less than or equal to `k` from the given vector. For example,

```
vector<int> v = {4, 9, 2, 3, 7};
int k = 4;
greaterThanK(v, k);
```

should change the vector `v` to `{9, 7}`.

5. Given two vectors `a` and `b`, write a function `bool isSubarray(const vector<string> &a, const vector<string> &b)` that returns true if `a` is a

subarray of b and false otherwise. For example,

```
vector<string> a = {"a", "t", "c"};
vector<string> b = {"a", "a", "t", "a", "t", "c", "g"};
isSubarray(a, b);
```

should return true since the sequence `{"a", "t", "c"}` appears in b starting at index 3. On the other hand,

```
vector<string> a = {"c", "g", "c"};
vector<string> b = {"a", "a", "t", "a", "t", "c", "g"};
isSubarray(a, b);
```

should return false since the sequence `{"c", "g", "c"}` does not appear in b.

6. Prime numbers cannot be written as the product of two integers both greater than 1. Put another way, if p is prime, then $p \bmod i \neq 0$ for all $2 \leq i < p$. It turns out that primes play an integral role in cryptography and in computer security.

   a. Write a function `bool isPrimeA(int n)` that checks the primality of `n` by checking whether or not $n \bmod i = 0$ for any $2 \leq i < n$.

   b. Write a function `int sumPrimesA(int n)` that returns the sum of all prime numbers less than `n`. This function should call `isPrimeA`.

   c. Notice that factors always come in pairs. In particular, if $p \bmod i = 0$, then there must be some integer q such that $q \cdot i = p$. Furthermore, at least one of q and i must be greater than or equal to $\sqrt{p}$. This means that, instead of checking if $n \bmod i = 0$ for any $2 \leq i < n$, we only need to consider $2 \leq i \leq \sqrt{n}$. Write a function `bool isPrimeB(int n)` using this fact. This function should be nearly identical to `isPrimeA`.

   d. Write a function `int sumPrimesB(int n)` that returns the sum of all prime numbers less than `n`. This function should be nearly identical to `sumPrimesA` but should use `isPrimeB` instead of `isPrimeA`.

   e. The sieve of Eratosthenes is another algorithm for finding all prime numbers less than some threshold n. We can implement this algorithm in C++ with the following steps. First, create a vector containing the numbers 2 up to $n - 1$. Next, for i from 2 to $\sqrt{n}$, if i has not been marked, mark all multiples of i ($2i$, $3i$, …) in the vector. One way to "mark" an index in this case is by setting it to 0. Remaining non-zero values in the vector will be prime.

      Write a function `int sieveOfEratosthenes(int n)` using this idea to find and return the sum of primes less than `n`.

   For fun: Once you have completed these functions, run `lab_1.cpp` with input 10. This will show how long each function takes to find the sum of primes below 10, 100, 1,000, 10,000, 100,000, and 500,000 once. Consider how

these approaches compare with respect to time and think about how this might relate to the asymptotic run time of these functions.

The program will also output the sum of primes less than 1 million as determined by your sieve of Eratosthenes implementation. This number might be negative. Does this make sense? What might be causing this? Try changing your function so that it returns a `long` instead of an `int`.