Book Question: (30 points)

Optimizing code can be (is) a “creative” process, but at times it can be accomplished using established “algorithmic” techniques. For this question, provide code (pseudocode is fine) that is an optimized version of the below three snippets of code, and explain Why your optimized version is “optimized” versus the original one. Your optimized version need not be suitable for running concurrently. If the code cannot be optimized, explain why. Your explanation of why the optimized version is better than the original, does not need to be lengthy. A few sentences should suffice.

**Code Snippet 1:** The code prints to the screen all positive prime integers greater than 2 but smaller than 100

```cpp
int i = 0;
while( i != 100) {
    for(int j=2; j<i; j++) {
        if(i%j==0) break;
        if(j==i-1) cout << i << endl;
    }
    i++;
}
```

**Code Snippet 2:** This code/pseudocode contains an embedded for-loop

```cpp
// already declared 2D arrays x and y
// already declared 1D array z
int i, j;
for (i=0; i<50; i++){
    for (j=0; j<50; j++){
        x[i][j] = y[j][i] + z[i];
    }
}
```

**Code Snippet 3:** A non-trivial for-loop.

```cpp
int n=78643, i, m, p;
int d[n], b[n], c[n], a[n];
for (i=0; i<n; i++){
    a[i] = Math.sqrt(c[i]);
    d[i] = d[i] + c[i-1];
    b[i] = 1 / d[i];
    c[i] = a[i-1] * m + n;
}
```
Coding Task : (70 points)

In lecture and in previous labs and homework assignments we’ve discussed threads, as well as finding the maximum of an array. For this homework assignment you’ll implement a variant of that problem -- finding the maximum entry of a large 2-D array. Note that this task is different than what you did previously, because the threads MUST communicate with each other (return their “largest” found entry to the main method that invoked the threads, which then should compare each thread’s maximum entry found to determine the overall largest entry).

Threading

As you’ve now experienced, just because you create a thread, does not mean that the scheduler will run that thread as soon as that portion of your code where the thread is declared is executed. A scheduler on an n processor computer might indeed even decide to run in series n threads on a single of the n processors if the load on all available processors is high. But there’s some hope. Use what you've learned to force the scheduler to run a thread as soon as it is created.

Coding Details -- Non-Threaded version

• Retrieve from the course website the maxNonThreaded.cpp file. That file contains a complete non-threaded program in which a 2D array of a large dimension is created and populated with random doubles. The main routine then inspects each entry of the 2D array in search for the largest entry.

• Compile maxNonThreaded.cpp.

• Run the program. It should take a non-trivial amount of time (tens of seconds). Modify the code to include a wall time, and calculate the wall time for finding the largest entry.

• Run the program for 20,000 x 20,000, 15,000 x 15,000, 10,000 x 10,000, and 5,000 x 5,000 matrix sizes. Run it twice for each size of matrix, and record the times. Consider modifying the program to accept command-line arguments for easy invocations.

Due to the inherent difficulty of generating a truly random number (using rand() draws from a small range), a very large 2D array (in your case 20,000 x 20,000) will have many duplicates, so the largest entry in one run of the program is often very similar to another run. Therefore, to confirm that the program correctly identifies the largest entry, consider (not required) outputting the indices i and j of the largest entry as a simple check that the program works as intended. On subsequent runs, i and j should be different.
Coding Details -- Threaded version

- Create a new version of the program, `maxThreaded.cpp`, that employs threading. Be sure to launch each thread right away. How you break up the 2D array, and into how many threads, is up to you.

- Your goal is to utilize threading so that the run time of finding the largest entry is significantly reduced when compared to the non-threaded version of the program.

- As you did for the non-threaded program, run the threaded program twice for 20,000 x 20,000, 15,000 x 15,000, 10,000 x 10,000, and 5,000 x 5,000 matrix sizes. Record the times.

Analyze the run-times

- Tally the run-times of your programs, and present your results in tabular form. For example, the following table has the run-time values for the threaded and non-threaded versions (entries are for demonstration purposes; your tallies might differ significantly than those shown here):

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Threaded Run 1</th>
<th>Threaded Run 2</th>
<th>Non-Threaded Run 1</th>
<th>Non-Threaded Run 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000x5000</td>
<td>1.2</td>
<td>1.3</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>10000x10000</td>
<td>2.4</td>
<td>2.2</td>
<td>3.0</td>
<td>4.6</td>
</tr>
<tr>
<td>15000x15000</td>
<td>4.9</td>
<td>5.4</td>
<td>15.4</td>
<td>16.8</td>
</tr>
<tr>
<td>20000x20000</td>
<td>12.7</td>
<td>16.7</td>
<td>48.8</td>
<td>36.6</td>
</tr>
</tbody>
</table>

- Generate a plot of your run-times to clearly demonstrate the utility of multithreading. Be sure to label your axes, and give your plot a title.

Submission. Submit via canvas:

- A document (.doc, .docx, .pdf) with your answers to code optimization questions.
- A document (.doc, .docx, .pdf) with your tabular run-times and plot(s). If you encountered unexpected run times, discuss possible reasons for what you observed.
- Source code for your program `maxNonThreaded.cpp`.
- Source code for your program `maxThreaded.cpp`.

Rubric

<table>
<thead>
<tr>
<th>Question/Component</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Optimization Questions</td>
<td>30 points</td>
</tr>
<tr>
<td><code>maxNonThreaded.cpp</code>, with wall clock code inserted, compiles and runs</td>
<td>10 points</td>
</tr>
<tr>
<td><code>maxThreaded.cpp</code> compiles and runs</td>
<td>20 points</td>
</tr>
<tr>
<td>Threaded version of the code is faster than non-threaded version</td>
<td>20 points</td>
</tr>
<tr>
<td>Tabular run-times</td>
<td>10 points</td>
</tr>
<tr>
<td>Plot of tabular run-times; title and axes labeled</td>
<td>10 points</td>
</tr>
<tr>
<td>Total</td>
<td>100 points</td>
</tr>
</tbody>
</table>