CS3BC Coursework

Module Code: CS3BC
Assignment report Title: Blockchain Coursework Assignment
Student Number: 30002734
Date (when the work was completed):
Actual hrs spent for the assignment:

Part 1

TODO

Part 2

Next, I created two classes: **Block** and **Blockchain**. In **Block.cs**, I declared all of the properties that any block in our chain needs: a timestamp, an index, a reference to the previous block's hash, its own hash, and a list of transactions (which starts out empty in the genesis case). I provided two constructors. The first, parameterless constructor sets the timestamp to `DateTime.Now`, assigns `index = 0`, leaves `prevHash` blank and instantiates an empty transaction list before immediately calling my hashing routine to compute the genesis block's `hash` . The second constructor takes the last block, a list of pending transactions and the miner's address; it sets `index = lastBlock.index + 1`, copies `prevHash = lastBlock.hash`, builds the transaction list (including a reward transaction) and computes the Merkle root before calling `Mine()` to obtain the new block's proof-of-work hash .

The hashing method itself lives in `CreateHash(long nonce)`. It concatenates the block's timestamp, index, previous hash, candidate nonce and, once we add transactions, the Merkle root into a single string. That string is then run through SHA-256, and the resulting byte array is formatted as a hexadecimal string and stored in the block's `hash` field . Because the genesis constructor doesn't perform proof-of-work, it simply uses `CreateHash(0)` once to seal the first block.

In **Blockchain.cs**, I declared a public `List<Block> blocks` and a `transactionPool` to hold pending transactions. In the default constructor I initialize `blocks = new List<Block>() { new Block() };` so that the very first action of the application is to create and append the genesis block to our chain . From there, every time the user clicks "Mine Block", we grab up to five transactions from the pool, pass them and the miner's address into the `new Block(...)` constructor, and append the result to `blocks`.

Finally, to verify everything worked, I added a `GetBlockAsString(int index)` method that returns the chosen block's `ToString()` output. In **Program.cs** (and the UI code behind `BlockchainApp.cs`), I instantiate a single `new Blockchain()` on startup and call `GetBlockAsString(0)` to print the genesis block's details—index, timestamp and its computed SHA-256 hash—into the main RichTextBox.