| (3) Suppose we added the following bytecode instructions to our language:  SEND\_CHANNEL:  Pops the channel and a value from the stack and send the  value on the channel using a blocking send  RECV\_CHANNEL:  Pops the channel from the stack, receives a value from the channel  (this may block), and push the resulting value back onto the stack  SPAWN:  Pop two functions from the stack and spawn them as concurrent tasks  Describe in a few sentences how each bytecode instruction could be interpreted,  and how your interpreter or language runtime could deal with the blocking nature  of the send and the receive instructions. |
| --- |
| I would expect SPAWN to always occur first in the order of execution, it would be last in the stack. There would be a channel for receiving and a channel for sending, both would be popped off the stack when SPAWN appears in the bytecode (SEND\_CHANNEL and RECV\_CHANNEL would be further down the stack). I would expect two threads to be created and each would handle send and receive scenarios. With the send channel placed in another thread it would not be blocking the main thread, and would reside in its own thread space and only block in that space. Therefore each thread would own its own messaging channel. |

| (4)  Write a function that given a directory, recursively finds all files with a given file  extension in that directory and all sub-directories, and counts the number of lines  in the file and prints it to stdout. |
| --- |
| See submitted code |

| (5) explain some of the ways hashing functions enable blockchain technology |
| --- |
| Key to blockchain distributed ledger technology is ensuring the integrity of the ledger, hash functions make it possible to protect the integrity and immutability of data stored on a distributed ledger. Hashing is applied on the transactions stored in each block, however there are clever techniques used to ensure that hashing isn't done in an inefficient way. **Merkle tree/hash root** is used to hash all the transactions on the block in an efficient way. |

| (6) briefly explain Bitcoin's UTXO model of transaction validation (separate from POW) |
| --- |
| Bitcoin nodes maintain a set of all active UTXOs databases which stands for Unspent Transaction Output.  This active record which have not been spent, is used to validate any transaction that comes in, all the nodes also update the UTXO after every transaction. |

| (7) what is the structure of a Block in bitcoin and how does it relate to the 'blockchain' (merkle tree vs merkle list of merkle trees) |
| --- |
| You have to think of a Block as a data structure in computer science terms, this structure has layers that can be firstly broken up as header and block content. Similar to a linked list when these sets of blocks are linked together through references to the previous block header hash, it ends up forming a chain, hence a blockchain. To ensure integrity hashing is performed on transactions, using the merkle tree approach, this then produces the merkle root. |

| (8) What problem/s are POW/POS trying to solve? discuss/compare (byzantine fault tolerance, reaching a single consensus on a p2p network) |
| --- |
| Proof of work is a form of proof in which one node proves to others that a certain amount of computational effort has been spent and trial and error to find the nonce for a block. Only blocks with a valid nonce can be added to the chain.  Proof of stake was conceived following concerns with proof of work and its energy usage and computational requirements. Proof of stake is consensus mechanisms for blockchains that work by selecting validators in proportion to their stake/investment in the network.  Byzantine fault tolerance is a property of systems that is able to continue operating even when some nodes fail to communicate or act maliciously. |