Pseudocode:

Validity check for cryptographic values:

Verifying signatures:

```
Procedure sign_message(message_string, signing_key):
    message_string = message_string.encode('ascii')
  return signing_key.sign(message_string)

Procedure send_client_req_msg(client_message, message_type):
  client_signed_messge = sign_message(client_message, client_signing_key)
  client_message = ClientMessage(client_id, message_type, client_signed_message)
  send((message_type, client_message), to = replicas)

Procedure verify_and_get_signed_message(signed_message, verify_key):
  verify_key.verify(signed_message)
  return signed_message.message.decode('ascii')

Procedure receive(message_type, message_object), from_ = sender):
  if message_type == CLIENT_MESSAGE:
      client_id = message_object.id
      client_verify_key = verify_key_list_client[client_id]
      client_msg = verify_and_get_signed_message(message_object.signed_message, cl
ient_verify_key)
```

Computing Hashes:

```
Procedure hashIt(str):
    return hashlib.sha256(str.encode('ascii')).hexdigest()

Procedure process_vote(voteMessage):
    vote_idx = hashIt(  str(voteMessage.ledger_commit_info.vote_info_hash) + str(v
oteMessage.ledger_commit_info.commit_state_id))

Class LedgerNode:
    id = hashIt(str(parent_id) + str(txns))

Procedure macke_vote(b, last_tc):
    vote_info_hash = hashIt( str(b.id) + str(b.round) + str(b.qc.vote_info.id) + s
tr(qc_round) + str(ledger.pending_state(b.id)) )
```

Client requests: De-duplication:

```
initialTransactions = orderedDict() #Add all client incoming client requests to the
dictionary
pendingTransactions = orderedDict() #Currently proposed transactions that are not c
ommited yet
committedTransactions = dict() # Stores the committed txns

Procedure addTransactions(txnId, clientMessage):
    if key not in intialTransactions  and key not in pendingTransactions and key no
t in committedTransactions:
        initialTransactions[txnId] = clientMessage
        return true
    return false
```

```
    # Get the first available txn and message
    Procedure getTransactions():
        if initialTransactions != {}
            {txnId, clientMessage} = initialTransactions[firstKey]
        Return None

    # Remove the txn from dictionaries if committed
    Procedure commitTransactions(txnId):
        If txnId in initialTrasactions:
            Delete initialTransactions[txnID]
        If txnId in pendingTransactions:
            Delete pendingTransactions[txnId]


    # On Receiving proposal message, move from initial dictionary to pending dictionary
    Procedure processTransaction(txnID):
        If txnId in initialTransactions:
            pendingTransactions[txnID] = initialTransactions[txnID]
            delete initialTransactions[txnId]

    # On getting TC, remove  from pending txns dictionary
    Procedure removePendingTransactions(txnId):
        if txnId in pendingTransactions:
            delete pendingTransactions[txnId]
```

Verify that submittedcommand was committed to ledger

In [ ]:

```
#Add acknowledgedtransactions
Mempool.py
def __init__(self):
  self.acknowledged_transactions = {}

#Pass client id dictionary to the replicas
driver.da
def main():
    client_id_to_object = defaultdict(Client)
      for i, client in enumerate(clients):
          client_id_to_object[i] = client
    for i, replica in enumerate(replicas):
        setup(replica, (i, client_id_to_object, replica_id_to_object, verify_key_list_re
plica[i], signing_key_list_replica[i],
                        verify_key_list_replica, verify_key_list_client))

    for i, replica in enumerate(faulty_replicas):
        setup(replica, (i + nreplicas, client_id_to_object, replica_id_to_object, verify
_key_list_replica[i+nreplicas],
                        signing_key_list_replica[i+nreplicas], verify_key_list_replica,
verify_key_list_client))
```

In [ ]:

```
# Send acknowledgement to client
replica.da
class Replica(process):
  def run():
    while not self.run_done or self.mem_pool.exists():
            committed_transactions = self.mem_pool.committed_transactions.keys()
            acknowledged_transactions = self.mem_pool.acknowledged_transactions.keys()
            txn_to_ack = list(committed_transactions - acknowledged_transactions)
            if len(txn_to_ack) > 0:
                if self.replica_id == self.leader_election.get_leader(self.block_tree.cu
rrent_round):
```

```
                    txn_to_ack = txn_to_ack[0]
                    self.mem_pool.acknowledged_transactions[txn_to_ack] = self.mem_pool.
committed_transactions[txn_to_ack]
                    send_ack(str(txn_to_ack))

  def send_ack(txn_to_commit):
      clientId =  int(txn_to_commit.split(',')[0])
      txnID = str(txn_to_commit.split(',')[1])
      send((txnID), to = self.clients[clientId])

# Receive the acknowledgement from replica and delete the corresponding client transactio
n id
client.da
class Client(process):
  def send_client_req_msg(msg : str, txn_id : str, type : str):
        clientDict[txn_id] = client_message
   def receive(msg=(txnId), from_ = replica):
        if txnId in self.clientDict.keys():
            del self.clientDict[txnId]
```

README

In [ ]:

```
DistAlgo:
OS: Ubuntu 20.04, Windows 10, Mac OS Big Sur
Python version: 3.7
DistAlgo version: 1.0.9

Workload generation:
The client workload is read from the initial config file and suitable configuration setup
is made. The corresponding files are config.da and client.da

Timeouts: For timeouts, we have used the formula 4 * delta

Bugs and Limitations:
1. Syncing of replicas that are lagging is missing
2. Message loss and retransmitting of message is not handled

Code size:

Lines of Code metrics were obtained by running the cloc command

Algo: 450 (DistAlgo) + 377(Python) = 827
Other: 80 (DistAlgo)
Total: 377 (Python) + 530 (DistAlgo) = 907


Main files:
src/driver.da
src/replica.da
src/client.da
src/Mempool.py
src/Ledger.py
src/BlockTree.py
src/LeaderElection.py

Contributions:
1. Venkatesh Venugopal
Failure testing, Block Tree, Ledger, Replica implementation, Client implmentation, Safet
y, Leader Election, Mempool implemenatation, Message processing
2. Venkatesan Ravi
Safety, Process Communication, Ledger, Block Tree, Leader Election, Mempool, Message Pro
cessing, Signing and Verfication, Replica and Client implementation
3. Amogh Joshi
Block Tree, Signing and Verification, Message Processing, Replica and Client implementat
ion, Mempool implementation, Leader Election, Ledger implementation
```