[CS 297] Special Topics - Blockchain Home Work No. 1

Members:

2012-97976 Eduardo Valdez
2009-79860 Lersan Del Mundo
2016-90653 Karen Michelle Alarcon

System Requirements and/or Dependencies:

- Operating System: MacOS High Sierra
- Applications:
  - Python 3.7
  - Postman

INTRODUCTION

This project is about creating a blockchain from the tutorial found in [1]. We are tasked to modify the code provided in the tutorial to create our own blockchain. In this project, the group had made some changes to the code to mimic the bitcoin application. Specifically, we have performed the following modifications:

a. Create a limit on the number of transactions per block to:
   i. enable democracy in the system and allow the participants esp. those who have limited resources to download a copy of the entire blockchain in their system; and
   ii. avoid Denial of Service attack which can be achieved by creating a large number of blocks constituting fillers such as dust transactions as discussed in [2].
b. Insert a Pseudo-Random Number Generator to add difficulty on the puzzle being solved in the Proof-of-Work algorithm.
c. Use a different hashing algorithm higher than SHA 256 to observe the effect of hashing using higher bits.

CODE IMPLEMENTATION

1. Full Code

The code below demonstrates our ideas on how we improve on simple blockchain:

```
#CS 297 Blockchain HW #1
# 2012-97976  Eduardo Valdez
# 2009-79860  Lersan Del Mundo
# 2016-90653  Karen Michelle Alarcon

import hashlib
import json
from urllib.parse import urlparse
from uuid import uuid4
from time import time
import requests
import random
import math
import requests
from flask import Flask, jsonify, request


#AverageTransactionsPerBlock = #2020
AverageTransactionsPerBlock = 5 #for convenience of testing
'''Maximum Block Size / Average Transaction Size = Average Transactions Per
Block
           1000000 Bytes / 495 Bytes = 2020 Transactions per block'''
class Blockchain:
    def __init__(self):

        self.current_transactions=[]
        self.chain = []
        self.nodes = set()

        # Create the genesis block
        self.new_block(previous_hash='1', proof=100)

    def register_node(self, address):
        """
        Add a new node to the list of nodes

        :param address: Address of node. Eg. 'http://192.168.0.5:5000'
        """

        parsed_url = urlparse(address)
        if parsed_url.netloc:
            self.nodes.add(parsed_url.netloc)
        elif parsed_url.path:
            # Accepts an URL without scheme like '192.168.0.5:5000'.
            self.nodes.add(parsed_url.path)
        else:
            raise ValueError('Invalid URL')


    def valid_chain(self, chain):
        """
        Determine if a given blockchain is valid
```

```
        :param chain: A blockchain
        :return: True if valid, False if not
        """
        last_block = chain[0]
        current_index = 1
        while current_index < len(chain):
            block = chain[current_index]
            print(f'{last_block}')
            print(f'{block}')
            print("\n-----------\n")
            # Check that the hash of the block is correct
            last_block_hash = self.hash(last_block)
            if block['previous_hash'] != last_block_hash:
                return False

            # Check that the Proof of Work is correct
            if not self.valid_proof(last_block['proof'], block['proof'],
last_block_hash):
                return False

            last_block = block
            current_index += 1

        return True
    def remove_none(self,transactions):
        temp = []
        for item in transactions:
            if item!=None:
                temp.append(item)
        return temp
    def resolve_conflicts(self):
        """
        This is our consensus algorithm, it resolves conflicts
        by replacing our chain with the longest one in the network.

        :return: True if our chain was replaced, False if not
        """

        neighbours = self.nodes
        new_chain = None

        # We're only looking for chains longer than ours
        max_length = len(self.chain)

        # Grab and verify the chains from all the nodes in our network
        for node in neighbours:
            response = requests.get(f'http://{node}/chain')

            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                # Check if the length is longer and the chain is valid
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # Replace our chain if we discovered a new, valid chain longer than
ours
```

```
        if new_chain:
            self.chain = new_chain
            return True

        return False

    def new_block(self, proof, previous_hash):
        """
        Create a new Block in the Blockchain

        :param proof: The proof given by the Proof of Work algorithm
        :param previous_hash: Hash of previous Block
        :return: New Block
        """
        print("new block")
        print(self.current_transactions)
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }
        # Reset the current list of transactions
        self.current_transactions = []
        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):
        """
        Creates a new transaction to go into the next mined Block
        :param sender: Address of the Sender
        :param recipient: Address of the Recipient
        :param amount: Amount
        :return: The index of the Block that will hold this transaction
        """
        self.current_transactions.append({
                    'sender': sender,
                    'recipient': recipient,
                    'amount': amount,
        })

        return self.last_block['index'] + 1

    @property
    def last_block(self):
        return self.chain[-1]

    @staticmethod
    def hash(block):
        """
        Creates a SHA-256 hash of a Block

        :param block: Block
        """

        # We must make sure that the Dictionary is Ordered, or we'll have
inconsistent hashes
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

```python
    def proof_of_work(self, last_block):
        """
        Simple Proof of Work Algorithm:

         - Find a number p' such that hash(pp') contains leading 4 zeroes
         - Where p is the previous proof, and p' is the new proof

        :param last_block: <dict> last Block
        :return: <int>
        """

        last_proof = last_block['proof']
        last_hash = self.hash(last_block)

        proof = 0
        while self.valid_proof(last_proof, proof, last_hash) is False:
            proof += 1

        return proof

    @staticmethod
    def valid_proof(last_proof, proof, last_hash):
        """
        Validates the Proof

        :param last_proof: <int> Previous Proof
        :param proof: <int> Current Proof
        :param last_hash: <str> The hash of the Previous Block
        :return: <bool> True if correct, False if not.

        """

        guess = f'{last_proof}{proof}{last_hash}'.encode()
        ''' SHA256 Vs. SHA384 Vs SHA512'''
        #guess_hash = hashlib.sha512(guess).hexdigest()
        #guess_hash = hashlib.sha384(guess).hexdigest()
        guess_hash = hashlib.sha256(guess).hexdigest()

        x=random.SystemRandom(time)
        y=math.ceil((x.random()*10))
        while y<4:
            y=math.ceil((x.random()*10))
        z="0"*y
        return guess_hash[:y] == z

# Instantiate the Node
app = Flask(__name__)
# Generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', '')

# Instantiate the Blockchain
blockchain = Blockchain()

@app.route('/mine', methods=['GET'])
def mine():
    # We run the proof of work algorithm to get the next proof...
    last_block = blockchain.last_block
    proof = blockchain.proof_of_work(last_block)
```

```
    # We must receive a reward for finding the proof.
    # The sender is "0" to signify that this node has mined a new coin.
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )

    # Forge the new Block by adding it to the chain
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200



@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

    # Check that the required fields are in the POST'ed data
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Create a new Transaction
    #print(len(blockchain.last_block['transactions']))
    #print(blockchain.last_block['transactions'])
    index = blockchain.new_transaction(values['sender'], values['recipient'],
values['amount'])
    if len(blockchain.current_transactions)<AverageTransactionsPerBlock:
        #index = blockchain.new_transaction(values['sender'],
values['recipient'], values['amount'])
        response = {'message': f'Transaction will be added to Block {index}'}
        return jsonify(response), 201
    else:
        response = {'message': f'Block is full. Transactions will be not added
to Block {index}. Your new transactions will be added to Block{index+1} after
mining.'}
        return jsonify(response), 201

        #response = {'message': f'Block is full. Transactions will be not
added to Block {index}. Mine first then add transactions to Block{index+1}'}

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200
```

```python
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201


@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200


if __name__ == '__main__':
    from argparse import ArgumentParser

    parser = ArgumentParser()
    parser.add_argument('-p', '--port', default=5000, type=int, help='port to 
listen on')
    args = parser.parse_args()
    port = args.port

    app.run(host='0.0.0.0', port=port)
```

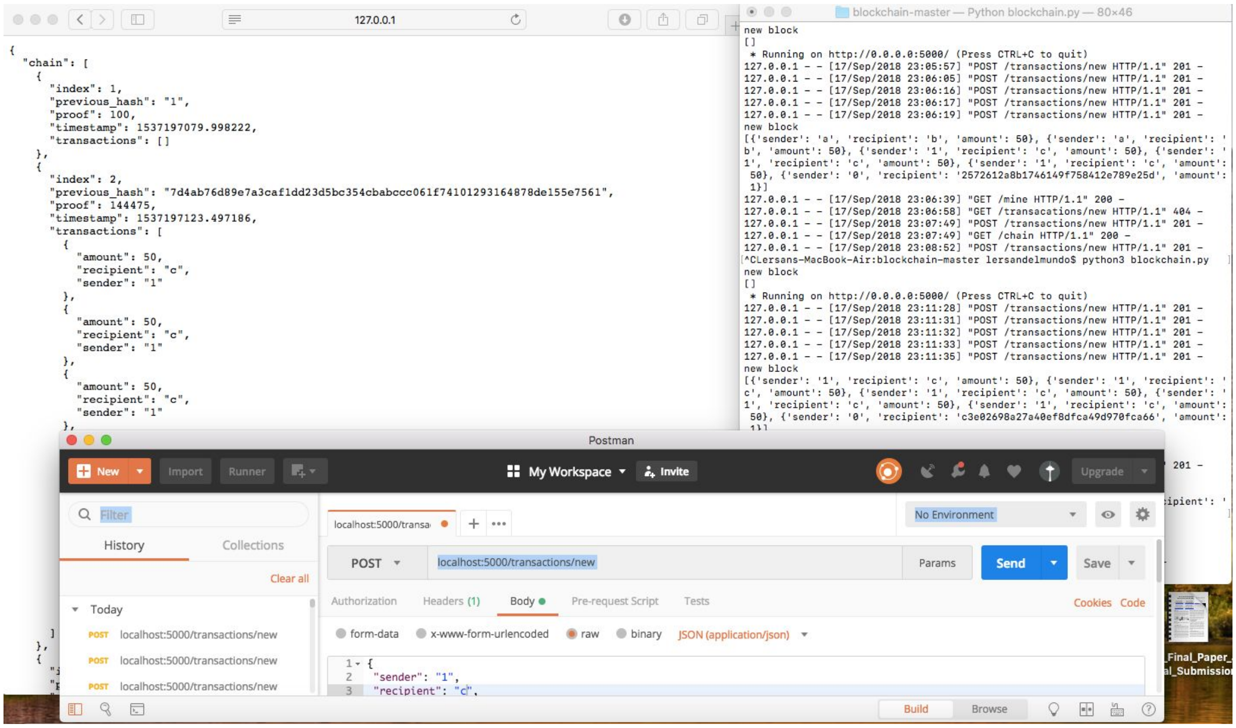


Figure 1. Screenshot of the blockchain working with browser, terminal and postman application.

2. Code Snippets

Table 1. Modified part of the code and its description

Modified Part of the Code	Description

def valid_proof(last_proof, proof,
last_hash):
        """
        Validates the Proof
 	
        :param last_proof: <int> Previous
Proof
        :param proof: <int> Current Proof
        :param last_hash: <str> The hash
of the Previous Block
        :return: <bool> True if correct,
False if not.
 	
        """
 	
        guess =
f'{last_proof}{proof}{last_hash}'.encode(
)
        ''' SHA256 Vs. SHA384 Vs
SHA512'''
	A pseudo-random number which includes a timestamp to ensure exact timeliness is mapped to a nonce instead of static ‘0000’ value.
	
	Another feature of `random()` function is typically used to generate unique values. However, for large data an initialization value using `seed()` would be a better technique for values to behave as expected. The seed value is dependent on the system time for initialization.
	
	Lastly, as a way of ensuring that

<pre>   #guess_hash =
hashlib.sha512(guess).hexdigest()
        #guess_hash =
hashlib.sha384(guess).hexdigest()
        guess_hash =
hashlib.sha256(guess).hexdigest()
 	
        x=random.SystemRandom(time)
        y=math.ceil((x.random()*10))
        while y<4:
            y=math.ceil((x.random()*10))
        z="0"*y
        return guess_hash[:y] == z</pre>
	generated values are never repeated, `SystemRandom()` is used to produce randomness by the system rather than by the application.
	
	Apparently, we modify the code for generating floating points which maps to the number of leading zeros. The minimum number of zeros is set to four '0000' up to ten '0000000000' as its maximum. Theoretically, this can take can exponentially take time to guess the hash of the previous proof and current proof containing x-leading zeros.
	
	Another modification is increasing the number of SHA bits from 256, 384 to 512. Aside from the time that it takes for these increase of bits, there was no much difference to as generating hash values and improving security features. These findings are supported in [6] since all of these are still SHA2 with no encryption capabilities.

<pre>AverageTransactionsPerBlock = 2020 #2020
average number of transactions per block
in the Bitcoin network.
#AverageTransactionsPerBlock = 5 for
testing
'''Maximum Block Size / Average
Transaction Size = Average Transactions
Per Block
           1000000 Bytes / 495 Bytes =
2020 Transactions per block'''
 	
app.route('/transactions/new',
methods=['POST'])
def new_transaction():
    values = request.get_json()
 	
    # Check that the required fields are
in the POST'ed data
    required = ['sender', 'recipient',
'amount']
    if not all(k in values for k in
required):
        return 'Missing values', 400
 	
    # Create a new Transaction</pre>
	Currently, the Bitcoin network has 1MB or 1000000 Bytes. To compute the average this is simply -
	
	Maximum Block Size / Average Transaction Size = Average Transactions Per Block 1000000 Bytes / 495 Bytes = 2020 Transactions.
	
	A new Bitcoin Block should be found by miners every 10 minutes. There are 600 seconds in 10 minutes.
	
	This was updated of limiting the block size to 1MB is to prevent DOS (Denial Of Service) attacks that could be achieved by creating a large number of massive blocks constituting filler (e.g. dust transactions) [2]. This would enable democracy in the system, an example would be Bitcoin Core which

<table>
<tr>
<td>

```
#print(len(blockchain.last_block['transac
tions']))

#print(blockchain.last_block['transaction
s'])
    index =
blockchain.new_transaction(values['sender
'], values['recipient'],
values['amount'])
    if
len(blockchain.current_transactions)<Aver
ageTransactionsPerBlock:
        #index =
blockchain.new_transaction(values['sender
'], values['recipient'],
values['amount'])
        response = {'message':
f'Transaction will be added to Block
{index}'}
        return jsonify(response), 201
    else:
        response = {'message': f'Block is
full. Transactions will be not added to
Block {index} anymore. Mine first then
add transactions to Block{index+1}'}
        return jsonify(response), 201

        #response = {'message': f'Block
is full. Transactions will be not added
to Block {index}. Mine first then add
transactions to Block{index+1}'}
```

</td>
<td>was the only bitcoin wallet back then, it would require users to download the entire blockchain, if blocks were this would mean those with limited resources or slow computer would never catch up, so some people would be not given a chance to spend their bitcoin[8].</td>
</tr>
</table>

FEEDBACKS/QUESTIONS FOR THE TUTORIAL

1. Why is there no blocksize? Would this mean there is no upper bound or lower bound for the number of transactions per block?
2. Why is it the number of leading 0's in a hash fixed to 4? In bitcoin's white paper, is it not.
3. Why is the block reward fixed to 1? What this means is that the amount of Bitcoin coming into circulation? Isn't it that the Bitcoin block mining reward halves every 210,000 blocks. Presently, there is 12.5 BTC reward per block and the coin reward will decrease from 12.5 to 6.25 coins around 2 years from now?
4. What will happen if the nonce is not a fixed set of zeros? Will it affect the security of the block or the difficulty of solving the puzzle?
5. Is it efficient if a new block would be mined without the prior block being full?
6. What would be the effects of having a fixed block interval? Say 10 minutes?

OTHER THINGS WE WANT TO TRY

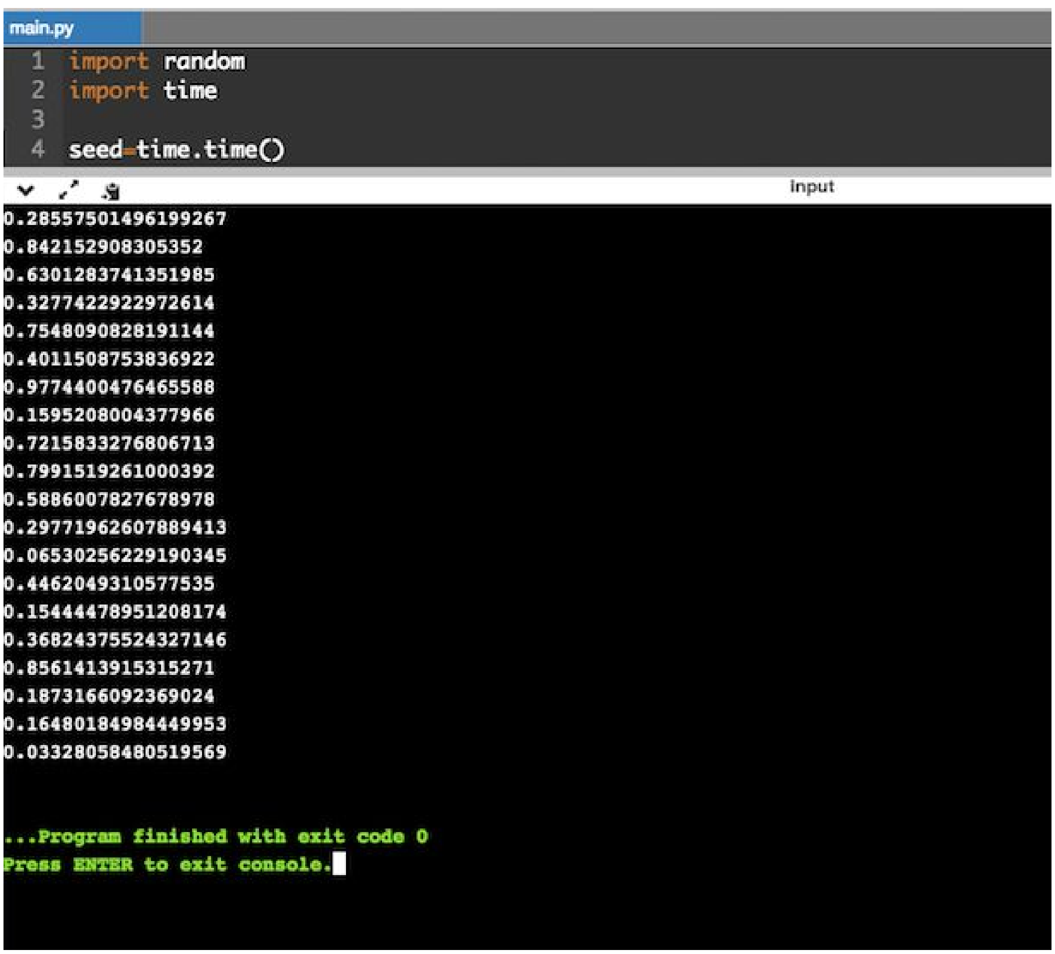
- Improve proof of work to be more useful. Instead of hashing to match certain number of zeros, we can use the computing power to generate Prime Numbers such as PrimeCoin which uses prime chains known as Cunningham chain. Searching for large Prime Numbers is useful for RSA and many more. The Electronic Frontier Foundation (EFF) gives a reward of $250,000 to the first individual or group who discovers a prime number with at least 1,000,000,000 decimal digits.
- Implement other types of consensus algorithm and compare results. For instance Proof of Exercise (PoX) addresses the computational issue on existing Proof of Work (PoW). The idea is to reduce the problem using matrix multiplication, determinants, eigenvectors, and the like.
- Create a use case for the generated numbers after mining since these numbers are considered nothing or of no use. This idea was inspired by the article in
- Increase the blocksize and improve other factors that would be significant to the bandwidth. If bitcoin or this use-case of blockchain is to replace money. It must be up to par with Visa or Mastercard network that could handle around 1700 transactions per second. Currently, Bitcoin can process 3-4 transactions per second [9]

PSEUDO-RANDOM NUMBER GENERATOR

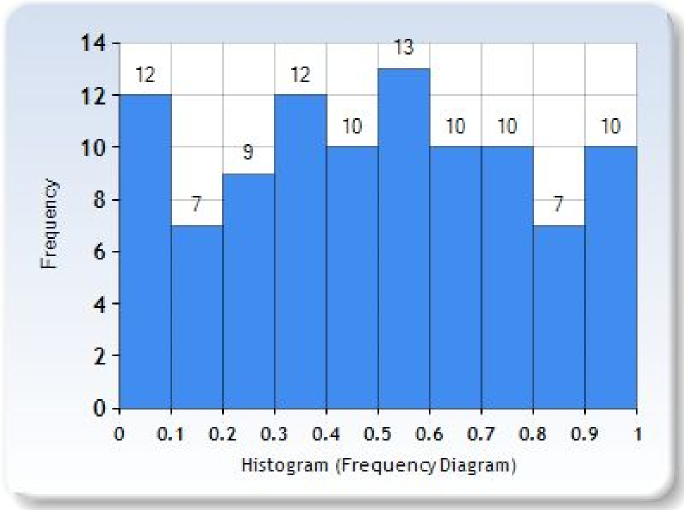
We constructed a simple pseudo-random generator which takes into consideration the system time as an initial seed value. Given a value of n, we simulated the random function by taking SystemRandom() function from the operating system which is completely independent of the application, i.e., Python 3.

We have run the simulator by increasing the number of n by 100. Each n is distributed in 10 equal bins. Then, we plotted the results using the histogram. The figures below show that it nearly aimed a uniform distribution. Apparently, there was no pattern of the discrepancy in counts per class interval. The reason for this is supported by a discussion in [7] that “no pseudo-random number generator perfectly simulates genuine randomness, so there is the possibility that any given application will resonate with some flaw in the generator.”

SCREENSHOTS



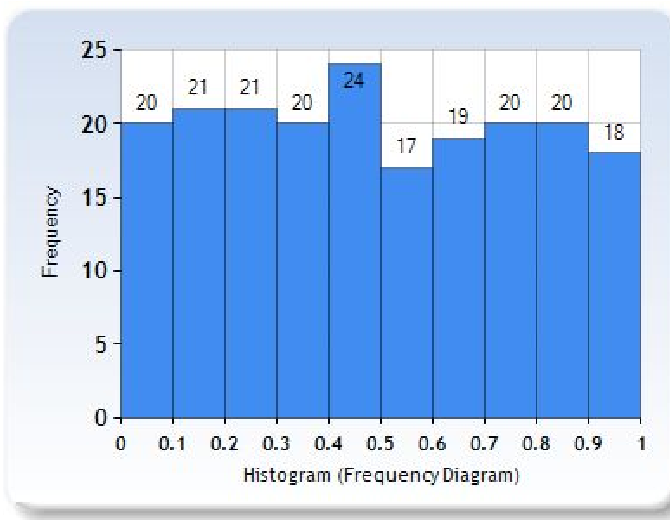
Frequency Table	
Class	Count
0-0.0999	12
0.1-0.1999	7
0.2-0.2999	9
0.3-0.3999	12
0.4-0.4999	10
0.5-0.5999	13
0.6-0.6999	10
0.7-0.7999	10
0.8-0.8999	7
0.9-0.9999	10



Your Histogram	
Mean	0.49402
Standard Deviation (s)	0.28671
Lowest Score	0.001
Highest Score	0.9953
Distribution Range	0.9943
Total Number of Scores	100
Number of Distinct Scores	99
Lowest Class Value	0
Highest Class Value	0.9999
Number of Classes	10
Class Range	0.1

N=100

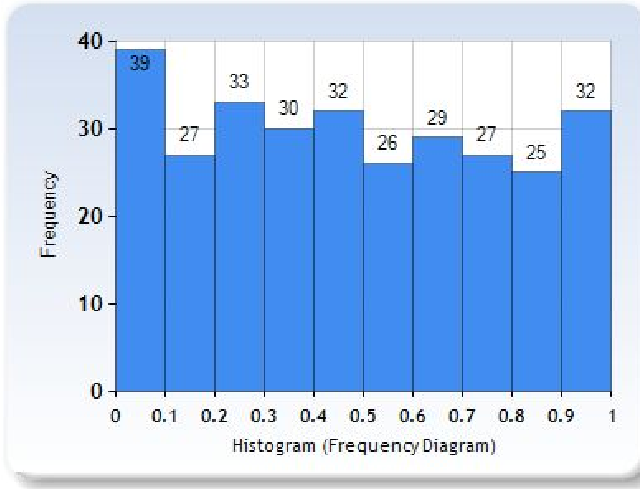
Frequency Table	
Class	Count
0-0.0999	20
0.1-0.1999	21
0.2-0.2999	21
0.3-0.3999	20
0.4-0.4999	24
0.5-0.5999	17
0.6-0.6999	19
0.7-0.7999	20
0.8-0.8999	20
0.9-0.9999	18



Your Histogram	
Mean	0.48912
Standard Deviation (s)	0.28708
Lowest Score	0.0009
Highest Score	0.9934
Distribution Range	0.9925
Total Number of Scores	200
Number of Distinct Scores	195
Lowest Class Value	0
Highest Class Value	0.9999
Number of Classes	10
Class Range	0.1

N=200

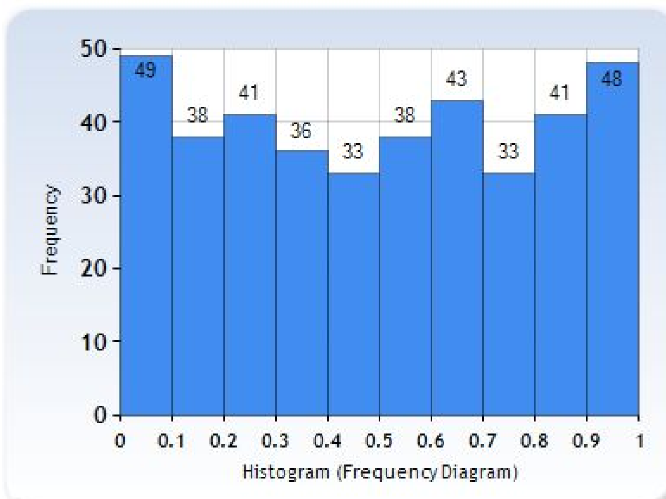
Frequency Table	
Class	Count
0-0.0999	39
0.1-0.1999	27
0.2-0.2999	33
0.3-0.3999	30
0.4-0.4999	32
0.5-0.5999	26
0.6-0.6999	29
0.7-0.7999	27
0.8-0.8999	25
0.9-0.9999	32



Your Histogram	
Mean	0.48098
Standard Deviation (s)	0.29788
Lowest Score	0.0022
Highest Score	0.9985
Distribution Range	0.9963
Total Number of Scores	300
Number of Distinct Scores	291
Lowest Class Value	0
Highest Class Value	0.9999
Number of Classes	10
Class Range	0.1

N=300

Frequency Table	
Class	Count
0-0.0999	49
0.1-0.1999	38
0.2-0.2999	41
0.3-0.3999	36
0.4-0.4999	33
0.5-0.5999	38
0.6-0.6999	43
0.7-0.7999	33
0.8-0.8999	41
0.9-0.9999	48

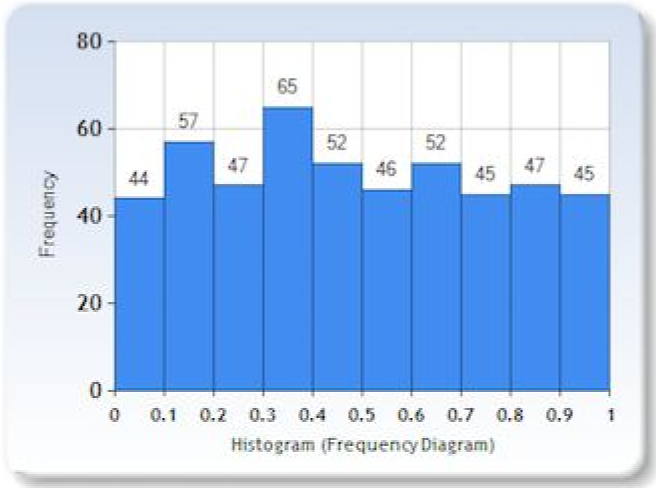


Your Histogram	
Mean	0.50018
Standard Deviation (s)	0.30267
Lowest Score	0.0023
Highest Score	0.9999
Distribution Range	0.9976
Total Number of Scores	400
Number of Distinct Scores	395
Lowest Class Value	0
Highest Class Value	0.9999
Number of Classes	10
Class Range	0.1

N=400

Frequency Table	
Class	*Count*
0-0.0999	44
0.1-0.1999	57
0.2-0.2999	47
0.3-0.3999	65
0.4-0.4999	52
0.5-0.5999	46
0.6-0.6999	52
0.7-0.7999	45
0.8-0.8999	47
0.9-0.9999	45

Your Histogram	
Mean	0.48788
Standard Deviation (s)	0.28188
Lowest Score	0.0025
Highest Score	0.9998
Distribution Range	0.9973
Total Number of Scores	500
Number of Distinct Scores	485
Lowest Class Value	0
Highest Class Value	0.9999
Number of Classes	10
Class Range	0.1



N=500

REFERENCES

[1] "Learn Blockchains by Building One – Hacker Noon." *Hacker Noon*, Hacker Noon, 24 Sept. 2017, hackernoon.com/learn-blockchains-by-building-one-117428612f46. Accessed 4 Sept. 2018.

[2] Madeira, Antonio. "What Is the Block Size Limit." *CryptoCompare*, 10 Sept. 2018, www.cryptocompare.com/coins/guides/what-is-the-block-size-limit/. Accessed 17 Sept. 2018.

[3] Nakamoto, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System." https://bitcoin.org/bitcoin.pdf

[4] "Block Size And Transactions Per Second." *Block Size And Transactions Per Second* | BitcoinPlus.org, www.bitcoinplus.org/blog/block-size-and-transactions-second.

[5] Jenkinson, Gareth. "Tulips, Bubbles, Obituaries: Peering Through the FUD About Crypto." *Cointelegraph*, Cointelegraph, 17 Sept. 2018, cointelegraph.com/news/tulips-bubbles-obituaries-peering-through-the-fud-about-crypto.

[6] Gilbert, Henri, and Helena Handschuh. "Security Analysis of SHA-256 and Sisters." *Selected Areas in Cryptography Lecture Notes in Computer Science*, 2004, pp. 175–193., doi:10.1007/978-3-540-24654-1_13.

[7] "Uniform Random Numbers." https://statweb.stanford.edu/~owen/mc/Ch-unifrng.pdf

[8] "A Primer on the Bitcoin Block Size Debate." *NewsBTC*, 1 Oct. 2015, www.newsbtc.com/2015/10/01/a-primer-on-the-bitcoin-block-size-debate/. Accessed 10 Sept. 2018.

[9] "Bitcoin and Ethereum vs Visa and PayPal – Transactions per Second." *Altcoin Today*, 22 Apr. 2017, altcointoday.com/bitcoin-ethereum-vs-visa-paypal-transactions-per-second/. Accessed by 17 Sept. 2018.