John Brinkman
CSE433
Lab 2
9/19/19

Q1: 0000000000001c0533ea776756cb6fdedbd952d3ab8bc71de3cd3f8a44cbaf85
Q2: 00000000000044b9f200000000000000000000000000000000000000000000000
Q3: 64627363
Q4: 00000000000042391c3620056af66ca9ad7cb962424a9b34611915cebb9e1a2a
Q5: For 3 runs the code ran in 13.2 mins, 12.7 mins, and 14.5 mins, so the average time was 13.5 mins. Since the program stopped at nonce 64627363, it went through 73560931 different nonces. So, $\frac{73560931\ nonces}{(13.5mins * 60\ seconds)} = 90{,}816\frac{nonces}{second}$ were tested.
Q6: The full range of nonce values are 0x00000000 to 0xFFFFFFFF. 0xFFFFFFFF is equal to 4294967295. So, $\frac{1\ second}{90{,}816\ nonces} * 4294967295\ nonces = 47{,}293\ seconds = 788\ mins = 13\ hours$

Code:

```
 def hashBlock(version, prevHash, merkleRoot, ts, bits, nonce):
    '''
    Computes the hash value for a Bitcoin block with the given parameters.
    See https://en.bitcoin.it/wiki/Block_hashing_algorithm for details
      on the Bitcoin hash algorithm.
    Note that parameters must be in Little Endian format.

    @param version Bitcoin version
    @param prevHash Hash of previous block
    @param merkleRoot Root of Merkle tree for the block
    @param ts Timestamp
    @param bits 'bits' field of block; encodes threshold below which hash
value must be
    @param nonce 4-byte value that causes hash to be below threshold
extracted from bits
    @return String representing Big Endian hex encoding of hash value
    '''
    # Hint: Don't forget the return value.
    # YOUR CODE HERE
    header_hex = version+prevHash+merkleRoot+ts+bits+nonce

    header_bin = header_hex.decode('hex')
    hash = hashlib.sha256(hashlib.sha256(header_bin).digest()).digest()
    hash.encode('hex_codec')
    #print(hash[::-1].encode('hex_codec'))
    return hash[::-1].encode('hex_codec')

def mineBlock(version, prevHash, merkleRoot, ts, bits):
    '''
    Computes a 4-byte nonce value that will yield a valid Bitcoin block given
      the other header values passed as parameters.
    See https://en.bitcoin.it/wiki/Block_hashing_algorithm for details
      on the Bitcoin hash algorithm.
    Note that parameters must be in Little Endian format.

    @param version: Bitcoin version
```

```
    @param prevHash: Hash of previous block
    @param merkleRoot: root of Merkle tree for the block
    @param ts: timestamp
    @param bits: Bits field of block; encodes threshold below which hash
value must be

    @return nonce value
    '''
    # extract threshold from bits
    # YOUR CODE HERE
    threshold = extractThreshold(bits)
    # mining loop
    # Hints to make problem tractable:
    #   1. The solution nonce is between 0x60000000 and 0x70000000.
    #   2. To iterate through a range of indices, use the xrange()
    #      function rather than the range() function to avoid running
    #      out of memory.
    # YOUR CODE HERE
    # for ...
    for i in xrange(0x60000000,0x70000000):
        # convert nonce to Little Endian hex-encoded string
        # YOUR CODE HERE
        littleEndianNonce = int2LittleEndian(i)
        # get hash value of block using this nonce
        # YOUR CODE HERE
        hashVal =
hashBlock(version,prevHash,merkleRoot,ts,bits,littleEndianNonce)
        # convert hash value to numeric val
        # YOUR CODE HERE (use following template, plug in correct variable
name--
        #  hashStr is hash value from previous step)
        # hashNum = int(hashStr, 16)
        hashNum = int(hashVal,16)
        # test for success; print or return nonce, hash, and threshold upon
success
        # NB: You can print the threshold with statement:
        #   'print qq2hexStr(threshold)'
        # Hint: don't forget the return value.
        # YOUR CODE HERE
        if hashNum < threshold:
            print("Int nonce: " + str(i))
            print("Little Endian nonce: " + str(int2LittleEndian(i)))
            print("Hash value: " + str(hashVal))
            print("Hash Number: " + str(hashNum))
            print qq2hexStr(threshold)
            return

if __name__ == "__main__":
    # main function: set up block parameters and call hashBlock or mineBlock
    # YOUR CODE HERE
    start = time.time()
    version = '01000000'
    # previousHash =
'1dbd981fe6985776b644b173a4d0385ddc1aa2a829688d1e0000000000000000'
    # merkleHashRoot =
'b371c14921b20c2895ed76545c116e0ad70167c5c4952ca201f5d544a26efb53'
```

```
    # timeStamp = 'b4f6d74d'
    # bits = 'f2b9441a'
    # nonce = '071a0c81'

#print(hashBlock(version,previousHash,merkleHashRoot,timeStamp,bits,nonce))

    previousHash =
'd44b8a28a4bc90c5e94acb3ffff8f710d42d085d39c9f349a42c000000000000'
    merkleHashRoot =
'1673404d0ff0a7a605811d5b84e7fb63b84ce0f0f28b91bf8d94304ec1f0f518'
    timeStamp = '264bd44d'
    bits = 'f2b9441a'
    mineBlock(version,previousHash,merkleHashRoot,timeStamp,bits)
    end = time.time()
    print(end-start)
```