National University of Computer and Emerging Sciences

Blockchain and Cryptocurrency (CS4049)

Date: April 4th, 2024

Course Instructor(s)

Syeda Tayyaba Bukhari

Sessional-II Exam

Total Time: 1 Hours

Total Marks: 35

Total Questions: 3

Semester: Spring 2024

Campus: Lahore

Dept: Computer Science

Student Name	Roll No	Section	Student Signature
_____	_____	_____	_____

Instructions:

1. Make sure there are total 8 pages including title page.
2. All questions are to be attempted on this paper. **No extra Sheets are allowed. Use last Page for rough work.**
3. Understanding of question is the part of exam.
4. If there is any ambiguity in the paper, benefit will be given to students.

Question No.	1	2	3	Total
Total Marks	4	20	11	35
Obtained Marks				

DO NOT OPEN UNTIL YOU ARE TOLD TO DO SO…..GOOD LUCK 😊

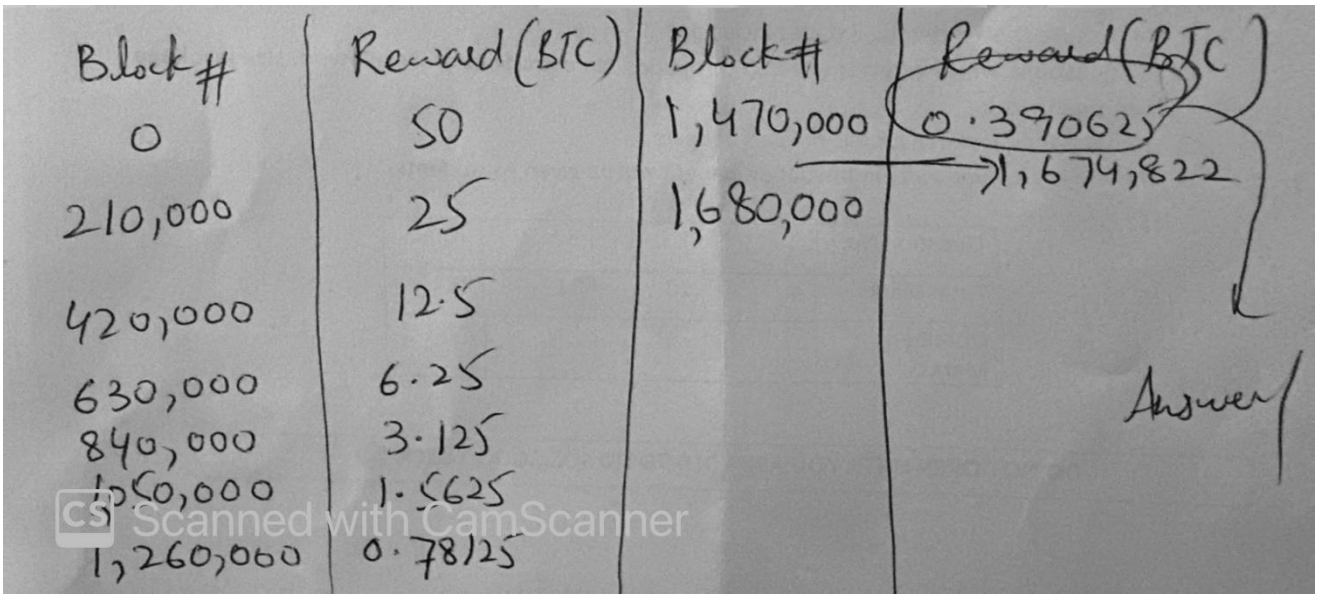
Question 1: Choose the Best Answer. Write your choice in above table either A, B, C or D
Answer Section for Q1 (Any type of overwriting is not allowed): **[4 marks]**

1	B
2	B

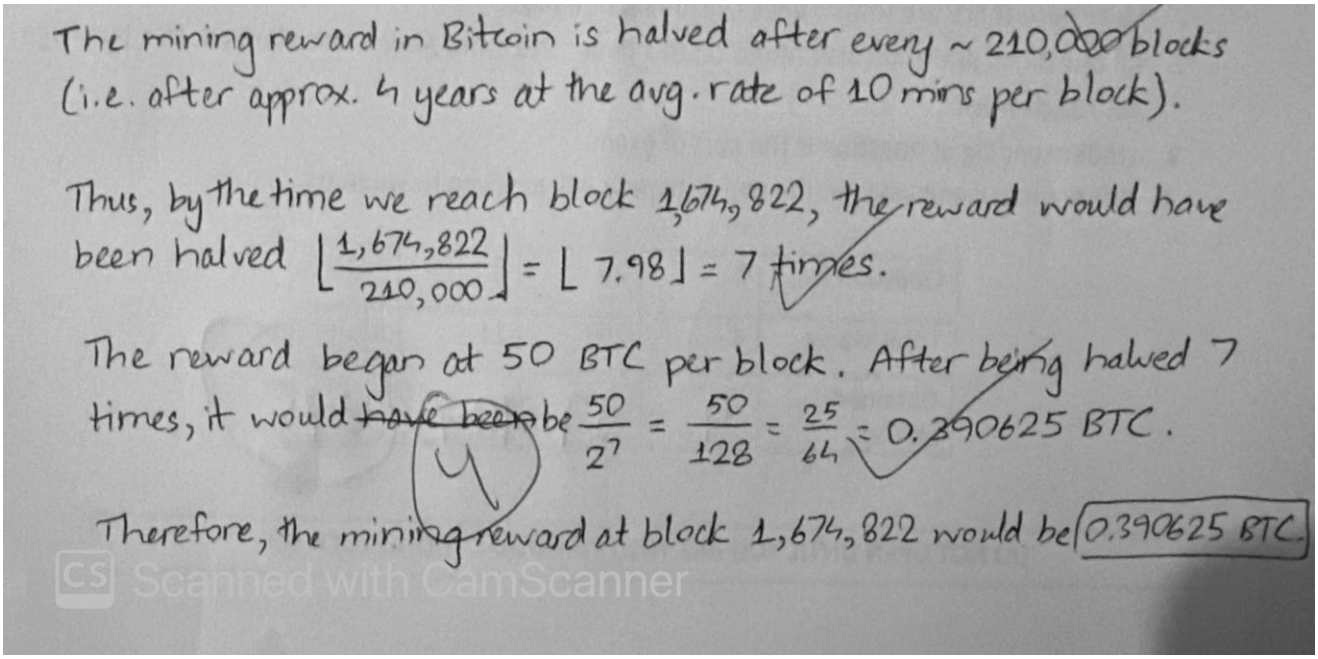
1. What is the purpose of the migrations directory in a Truffle project?
   A. It contains Solidity source files for smart contracts.
   B. It handles the deployment of smart contracts.
   C. It stores JavaScript and Solidity tests.
   D. It contains configuration files for Truffle.
2. What lessons were learned from the DAO attack?
   a) The importance of centralized control in blockchain systems
   b) The need for better governance and code auditing in DAOs
   c) The irreversibility of blockchain transactions
   d) The limitations of smart contracts in decentralized systems

Q2: Answer following questions:

a. In Bitcoin, what will be the mining reward at block number 1,674,822. Show complete working. **[4 marks]**



OR



The mining reward in Bitcoin is halved after every ~ 210,000 blocks (i.e. after approx. 4 years at the avg. rate of 10 mins per block).

Thus, by the time we reach block 1,674,822, the reward would have been halved $\lfloor \frac{1,674,822}{210,000} \rfloor = \lfloor 7.98 \rfloor = 7$ times.

The reward began at 50 BTC per block. After being halved 7 times, it would ~~have been~~ be $\frac{50}{2^7} = \frac{50}{128} = \frac{25}{64} = 0.390625$ BTC.

Therefore, the mining reward at block 1,674,822 would be 0.390625 BTC.

b. UTXOs: **[4 marks]**

1.	A	-> Me	1.11BTC
2.	B	-> Me	0.91BTC
3.	C	-> Me	0.31BTC
4.	D	-> Me	0.08BTC
5.	E	-> Me	0.99BTC
6.	F	-> Me	0.16BTC
7.	G	-> Me	0.22BTC

I want to buy a laptop for 1 BTC, a book worth of 0.85 and a pen worth of 0.25 with 0.12 traction fee.

c. Write down the core purpose of bitcoin and Ethereum? **[2 marks]**

d. Write the abbreviations of following: **[4 marks]**

1. **DAOs:**_____
2. **UTXOs:**_____
3. **ECDSA:**_____
4. **BTG:**_____

e. Soft Fork is backward compatible, what does that mean? **[2 marks]**

Chain is compatible with both, old and new rules.

f. Write API for Digital Signatures **[2 marks]**

g. Write Exact Date when Segwit has introduced in Bitcoin Network. [2 marks]

Q3 Complete the require statements in Solidity **[4+7 marks]**

Part a:

Complete the require statements:

```
contract Election {
        struct Candidate {// Model a Candidate
        uint id;
        string name;
        uint voteCount;
    }
    // Store accounts that have voted
    mapping(address => bool) public voters;
    // Read/write candidates
    mapping(uint => Candidate) public candidates;
    // Store Candidates Count
    uint public candidatesCount;
    function Election () public {
        addCandidate("Candidate 1");
        addCandidate("Candidate 2");
    }
    function addCandidate (string _name) private {
        candidatesCount ++;
        candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
    }

    function vote (uint _candidateId) public {
        // require that they haven't voted before

            require(!voters[msg.sender])

        // require a valid candidate

            require(_candidateId >0&&_candidateId <=candidatesCount)

        // record that voter has voted

            voters[msg.sender]=true

        // update candidate vote Count

            candidates[_candidateId].voteCount++;
    }


}
```

Part b: Complete the below code.

```
contract Marketplace {
    struct Product {
        uint256 id;
        string name;
        uint256 price;
        address payable seller;
        bool isAvailable;
    }
    // Mapping to store products by their IDs
    mapping(uint256 => Product) public products;
    // Counter for generating unique product IDs
    uint256 public productCount;
    // Event emitted when a product is added to the marketplace
    event ProductAdded(uint256 id, string name, uint256 price, address seller);
    // Event emitted when a product is purchased from the marketplace
    event ProductPurchased(uint256 id, string name, uint256 price, address buyer);
    constructor() {
        productCount = 0;
    }
    // Function to add a new product to the marketplace
    function addProduct(string memory _name, uint256 _price) public {
        // Require that the price is greater than zero
        require(_price > 0, "Price must be greater than zero.");
        // Increment the product count and assign the new ID
        productCount++;
        // Create a new product and store it in the mapping
        products[productCount] = Product(productCount, _name, _price, msg.sender, true);
        // Emit the ProductAdded event
        ProductAdded(productCount, _name, _price, msg.sender);
    }
```

```
    // Function to purchase a product from the marketplace
    function purchaseProduct(uint256 _id) public payable {
        // Retrieve the product based on the provided ID
        Product storage product = products[_id];
        // Require that the product exists and is available for purchase
            require(product.id != 0, "Product does not exist.");
            require(product.isAvailable, "Product is not available for purchase.");
        // Require that the buyer sends sufficient funds
            require(msg.value >= product.price, "Insufficient funds.");
        // Update the availability status of the product
            product.isAvailable = false;

        // Transfer the payment to the seller
        product.seller.transfer(product.price);
        // Emit the ProductPurchased event
            ProductPurchased(product.id, product.name, product.price, msg.sender);
    }
}
```

Rough Sheet: