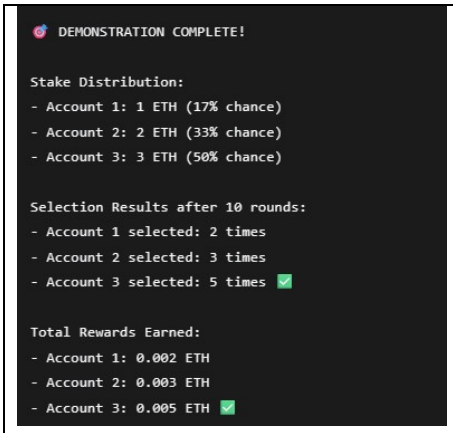
* Implementation Phase: Final Output (no error)



- In PoS, your influence and earnings are proportional to your stake in the network.

* Observations

Statistical Pattern (After 10 selections):

- Account with 3 ETH → Selected ~4-5 times (≈50% probability)
- Account with 2 ETH → Selected ~3 times (≈33% probability)
- Account with 1 ETH → Selected ~2 times (≈17% probability)

Key Proof of Stake Demonstration:

- Higher Stake = Higher Validation Frequency = Higher Rewards

ASSESMENT

Rubrics	Full Mark	Marks Obtained	Remarks
Concept	10		
Planning and Execution/ Practical Simulation/ Programming	10		
Result and Interpretation	10		
Record of Applied and Action Learning	10		
Viva	10		
Total	**50**		

Signature of the Student: PN Archana

Name :

Regn. No. :

Signature of the Faculty:

**As applicable according to the experiment.*
Two sheets per experiment (10-20) to be used.

Implementation Phase: Final Output (no error)

After staking :

```
✅ Transaction successful (for each staker)
- Account 1: Staked 1 ETH
- Account 2: Staked 2 ETH
- Account 3: Staked 3 ETH

totalStaked() → 600000000000000000 (6 ETH in wei)
getStakerCount() → 3
```

After Calling selectValidator() Multiple Times:

Run -1

```
✅ Transaction successful
currentValidator() → 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 (Account with 3 ETH)
rewards(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) → 1000000000000000 (0.001 ETH reward)
```

Run -2

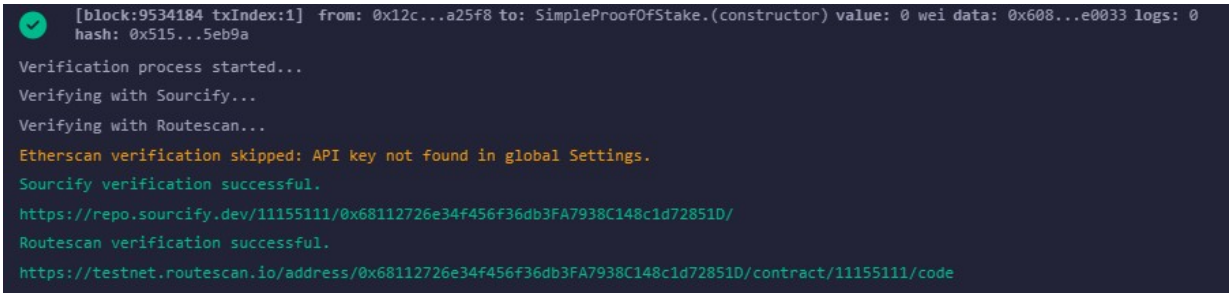
```
✅ Transaction successful
currentValidator() → 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 (Account with 1 ETH)
rewards(0x5B38Da6a701c568545dCfcB03FcB875f56beddC4) → 1000000000000000 (0.001 ETH reward)
```

Run-3

```
✅ Transaction successful
currentValidator() → 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2 (Account with 3 ETH again!)
rewards(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2) → 2000000000000000 (0.002 ETH total)
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

* Implementation Phase: Final Output (no error)




```
Account 1:
- Function: stake()
- Value: 1 ether
- Input: [empty]
- Click Transact

Account 2:
- Switch account
- Function: stake()
- Value: 2 ether
- Input: [empty]
- Click Transact

Then:
- Function: selectValidator()
- Value: 0
- Input: [empty]
- Click Transact

Check result:
- Function: currentValidator()
- Value: 0
- Input: [empty]
- Click Call
```

Coding Phase: Pseudo Code / Flow Chart / Algorithm

* Softwares used

- Remix IDE
- Solidity Language
- Sepolia Testnet on Metamask Wallet

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

How to Deploy & Test the Fixed Contract:

1. Deploy the New Contract

- Copy the updated code above
- Deploy it to Sepolia testnet (you'll need to verify again)

2. Correct Way to Stake (3 Methods):

Method 1: Using the Value Field (Recommended)

1. Select the stake() function
2. In the "Value" field, enter the amount in ETH (e.g., 1 ETH)
3. Click "stake" - NO need to send ETH separately

Method 2: Direct ETH Transfer

- The receive() function will automatically call stake()

Method 3: Using Remix Value Field

- Make sure "Value" is set before calling any payable function

3. Step-by-Step Demo:

Account 1:

- Select stake() function
- Set "Value" to **1 ETH**
- Click "Transact"

Account 2:

- Switch accounts
- Select stake() function
- Set "Value" to **2 ETH**
- Click "Transact"

Account 3:

- Switch accounts
- Select stake() function
- Set "Value" to **3 ETH**
- Click "Transact"

4. Test the PoS Selection:

- Call selectValidator() multiple times
- Check currentValidator() after each call
- Call getMyRewards() to see validator earnings

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

```
function getContractBalance() external view returns (uint256) {
        return address(this).balance;
    }

    function getStakerCount() external view returns (uint256) {
        return stakers.length;
    }

    function getMyStake() external view returns (uint256) {
        return stakes[msg.sender];
    }

    function getMyRewards() external view returns (uint256) {
        return rewards[msg.sender];
    }
}
```


*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

```
cumulativeStake += stakes[stakers[i]];
            if (randomNumber < cumulativeStake && stakes[stakers[i]] > 0) {
                currentValidator = stakers[i];
                rewards[currentValidator] += 0.001 ether;
                break;
            }
        }
    }
function amIValidator() external view returns (bool) {
        return msg.sender == currentValidator;
    }

    function withdrawRewards() external {
        uint256 amount = rewards[msg.sender];
        require(amount > 0, "No rewards to withdraw");
        require(address(this).balance >= amount, "Contract has insufficient funds");

        rewards[msg.sender] = 0;
        payable(msg.sender).transfer(amount);
    }

    function unstake() external {
        uint256 amount = stakes[msg.sender];
        require(amount > 0, "No tokens staked");

        stakes[msg.sender] = 0;
        totalStaked -= amount;

        // Remove from stakers array
        for (uint256 i = 0; i < stakers.length; i++) {
            if (stakers[i] == msg.sender) {
                stakers[i] = stakers[stakers.length - 1];
                stakers.pop();
                break;
            }
        }

        payable(msg.sender).transfer(amount);
    }
```

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SimpleProofOfStake {
    address[] public stakers;
    mapping(address => uint256) public stakes;
    mapping(address => uint256) public rewards;

    uint256 public totalStaked;
    address public currentValidator;

    // Add this function to receive ETH
    receive() external payable {
        stake();
    }

    function stake() public payable {
        require(msg.value > 0, "Must send ETH to stake");
        if (stakes[msg.sender] == 0) {
            stakers.push(msg.sender);
        }
        stakes[msg.sender] += msg.value;
        totalStaked += msg.value;
    }

    function selectValidator() public {
        require(totalStaked > 0, "No stakers available");
        require(stakers.length > 0, "No validators");

        // Simple pseudo-random selection
        uint256 randomNumber = uint256(
            keccak256(abi.encodePacked(block.timestamp, block.prevrandao, stakers.length))
        ) % totalStaked;

        uint256 cumulativeStake = 0;

        for (uint256 i = 0; i < stakers.length; i++) {
```

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

Step 4: Check Individual Stakes

For each account, call:

- stakes(ACCOUNT_ADDRESS) - to see how much each staked
- amIValidator() - to check if current validator

Step 5: View on Block Explorer

Go to your verified contract on Routescan:
https://testnet.routescan.io/address/0xaA449DdEA83759349Fcbb1De01e11C0Fc8e817B7/contract/11155111/code

Here you can:

- See all transactions
- View contract source code
- Check internal state

Expected Results:

- Account with 3 ETH stake → ~50% chance of selection
- Account with 2 ETH stake → ~33% chance
- Account with 1 ETH stake → ~17% chance

Quick Commands Summary:

1. stake() + send ETH → Become a validator
2. selectValidator() → Randomly pick validator (weighted by stake)
3. currentValidator() → See who was chosen
4. stakes(address) → Check anyone's stake amount
5. totalStaked() → See total network stake

This demonstrates the core PoS principle: "The more you stake, the more likely you are to be chosen as validator!"

Coding Phase: Pseudo Code / Flow Chart / Algorithm:

Expected Output:

when Your contract is successfully verified and deployed. Now let's interact with it to demonstrate Proof of Stake. Here's what to do next:

Step 1: Get Test ETH (Required for Staking)

Get Sepolia ETH from a faucet:

- Go to: https://sepoliafaucet.com/
- Or: https://cloud.google.com/application/web3/faucet/ethereum/sepolia
- Connect your wallet and get test ETH (you'll need this for staking)

Step 2: Interact with Your Deployed Contract

In Remix, go to "Deploy & Run Transactions" tab and:

Demo Scenario: Simulate 3 Validators

Account 1 (Default):

1. Make sure you have some test ETH
2. Call stake() function with 1 ETH in the "Value" field
3. Click "Transact"

Account 2:

1. Switch to Account 2 in the dropdown above deployed contracts
2. Call stake() with 2 ETH in "Value" field
3. Click "Transact"

Account 3:

1. Switch to Account 3
2. Call stake() with 3 ETH in "Value" field
3. Click "Transact"

Step 3: Select Validators & See PoS in Action

Now demonstrate Proof of Stake:

1. Check total staked:
    - Call totalStaked() - should show 6 ETH
2. Select a validator:
    - Call selectValidator()
    - This will randomly pick one based on their stake weight
3. Check who was selected:
    - Call currentValidator() to see the winner
    - The address with higher stake (3 ETH) has higher probability!
4. Repeat the selection:
    - Call selectValidator() 5-10 times
    - Each time, check currentValidator()
    - You'll see that accounts with more stake get chosen more often

Centurion University
Shaping Lives... Empowering Communities...

School: ..... Campus: .....

Academic Year: ..... Subject Name: ..... Subject Code: .....

Semester: ..... Program: ..... Branch: ..... Specialization: .....

Date: .....

Applied and Action Learning

(Learning by Doing and Discovery)

Name of the Experiement :

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Stake Your Claim – Proof of Stake Simulation :

1. Setup
   - Go to remix.ethereum.org
   - Create a new file called SimplePoS.sol
   - Paste the above code
   - Compile with Solidity 0.8.0+
2. Deploy
   - Go to the "Deploy & Run Transactions" tab
   - Select "JavaScript VM" as environment
   - Deploy the contract
3. Interactive Demo Steps:

Step 1 - Multiple Users Stake:
   - Switch between different accounts (Account 1, Account 2, Account 3)
   - For each account, call stake() with different values (e.g., 1 ETH, 2 ETH, 3 ETH)
   - Use the "Value" field in Remix to send ETH with the transaction

Step 2 - Select Validator:
   - Call selectValidator() - this will randomly pick a validator weighted by their stake
   - Check who was selected by calling currentValidator()

Step 3 - See the Effect:
   - The selected validator gets a reward (0.001 ETH)
   - Call rewards(validatorAddress) to see their accumulated rewards
   - Call amIValidator() from different accounts to check

Step 4 - Repeat:
   - Call selectValidator() multiple times
   - Observe that accounts with higher stakes get selected more frequently

Key Concepts Demonstrated:

1. Staking: Users lock ETH to participate
2. Weighted Random Selection: Higher stake = higher chance of being validator
3. Rewards: Selected validators earn rewards
4. Economic Incentives: Stake more to validate more, earn more

*As applicable according to the experiment.
Two sheets per experiment (10-20) to be used.