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 : Gas Race – Optimizing Smart Contract Efficiency

Objective/Aim:

To analyze gas consumption of different Solidity code patterns, deploy optimized vs. non-optimized smart contracts on a testnet, and observe how programming choices affect gas usage.

Apparatus/Software Used

MetaMask Wallet

1. Brave / Chrome Web Browser
2. Remix IDE – https://remix.ethereum.org
3. Ethereum Sepolia Testnet
4. Testnet ETH from a faucet

Theory/Concept:

Smart contract gas usage depends on:

- Storage operations (SSTORE / SLOAD) – Expensive
- Loops & repeated operations
- Data types (e.g., uint256 vs uint8—same cost on EVM)
- Memory vs Storage
- Function visibility
- Use of calldata vs memory

Gas optimization improves:

- Transaction cost
- Contract efficiency
- User affordability

Procedure:

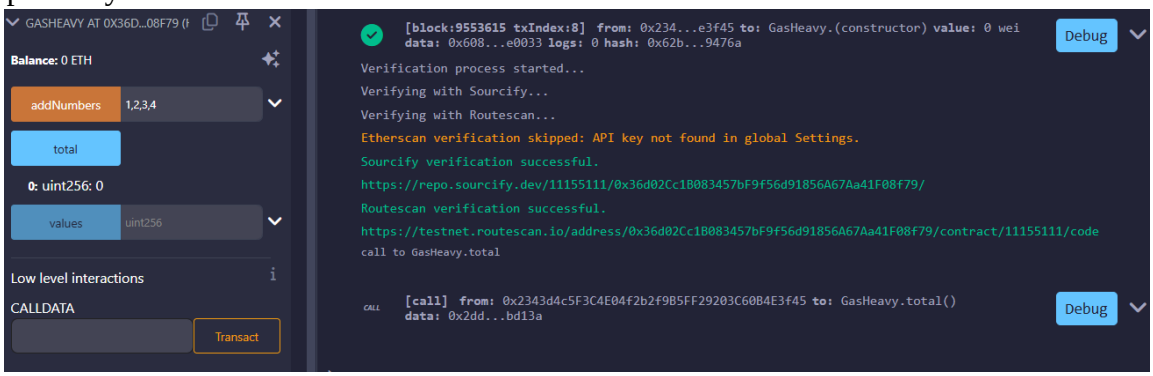
1. Open MetaMask and switch to Sepolia testnet
(Ensure you have a small amount of test ETH.)
2. Open Remix in your browser
Visit https://remix.ethereum.org
3. Create Contract 1 – Non-optimized Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract GasHeavy {
    uint256 public total;
    uint256[] public values;

    function addNumbers(uint256[] memory nums) public {    infinite gas
        for (uint256 i = 0; i < nums.length; i++) {
            values.push(nums[i]);        // storage write (expensive)
            total += nums[i];            // repeated SLOAD + SSTORE
        }
    }
}
```

4. Compile the contract
Use Solidity compiler 0.8.x.
5. Deploy it on Sepolia
Select Injected Provider – MetaMask → Deploy → Confirm transaction.
6. Call addNumbers()
Input a sample array such as:



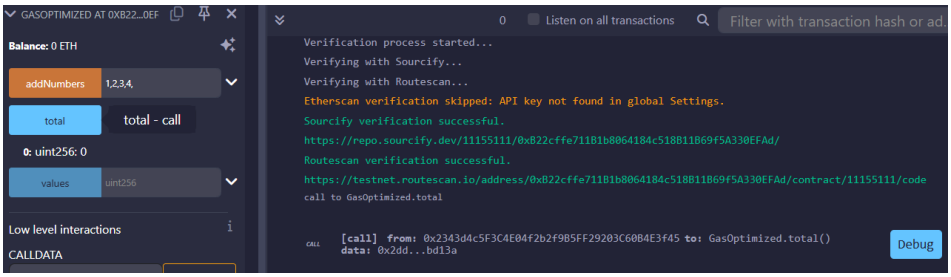
7. Create Contract 2 – Optimized Version
Create another file in Remix and paste:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract GasOptimized {
    uint256 public total;
    uint256[] public values;

    function addNumbers(uint256[] calldata nums) external {    inf
        uint256 temp = total;        // load once
        for (uint256 i = 0; i < nums.length; i++) {
            values.push(nums[i]);    // unavoidable storage write
```

8. Compile the optimized contract
9. Deploy it on Sepolia
Again, choose Injected Provider – MetaMask.
10. Run the same input
Call:



Observation

1. Optimized contract consumed significantly less gas for the same inputs.
2. Using calldata, temporary variables, and reduced SSTORE operations lowered gas usage.
3. Storage writes (values.push) remain expensive but unavoidable.
4. MetaMask clearly displayed the difference in estimated gas during transaction confirmation.
5. Efficient smart-contract design leads to cost savings and better performance on testnets and mainnet.

ASSESSMENT

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 Faculty:

Signature of the Student:

Name :