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

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

Algorithm

1.Start the Solidity smart contract in Remix IDE.
2.Write the initial version of the contract with basic logic (e.g., storing and updating data).
3.Compile and deploy the contract to observe the initial gas consumption.
4.Analyze gas usage using Remix's "Gas Analysis" tool after each function execution.
5.Apply optimization techniques, such as: ○ Using memory instead of storage when possible. ○ Declaring variables with the smallest suitable data type. ○ Combining operations and reducing function calls.
6.Recompile and redeploy the optimized contract.
7.Compare gas usage before and after optimization.
8.Stop after verifying reduced gas consumption and correct functionality.

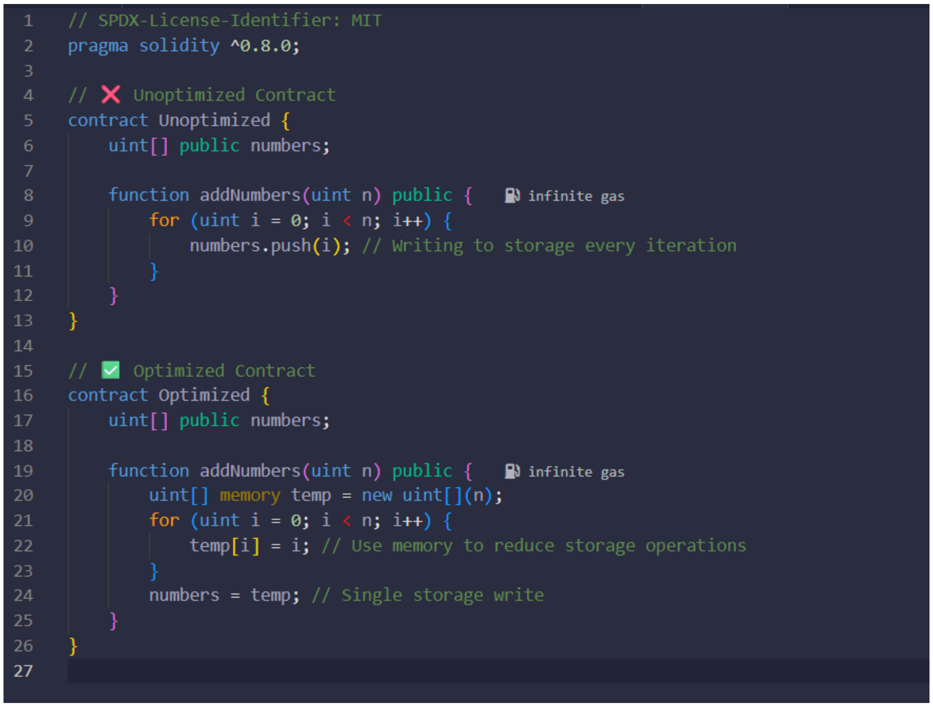
* Softwares used

1.Remix Ide
2.MetaMask
3.Web3.js/Ether.js
4.Sepolia testnet

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

* Implementation Phase: Final Output (no error)

- Open Remix IDE and create a new Solidity file named GasOptimization.sol.
- Select the Solidity compiler version 0.8.0 or above.
- Write a normal contract (unoptimized) and an optimized version.
- Compile both contracts and check gas cost under the “Deploy & Run Transactions” tab.
- Observe and compare gas usage between the two contracts.



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

// ❌ Unoptimized Contract
contract Unoptimized {
    uint[] public numbers;

    function addNumbers(uint n) public {
        for (uint i = 0; i < n; i++) {
            numbers.push(i); // Writing to storage every iteration
        }
    }
}

// ✅ Optimized Contract
contract Optimized {
    uint[] public numbers;

    function addNumbers(uint n) public {
        uint[] memory temp = new uint[](n);
        for (uint i = 0; i < n; i++) {
            temp[i] = i; // Use memory to reduce storage operations
        }
        numbers = temp; // Single storage write
    }
}
```

Compilation of Code (Error Detection):

- Open Remix IDE → Paste both contracts.
- Compile with Solidity 0.8.x.
- No syntax or semantic errors should appear.
- If type mismatch errors appear, verify variable declarations and storage types.
-

* Implementation Phase: Final Output (no error)

- Final Output (No Error):
- Deploy Unoptimized and Optimized contracts.
- Call addNumbers(10) in both contracts.
- In Remix console → Check Gas Used under transaction details:
- Example:
- Unoptimized: ~90,000 gas
- Optimized: ~45,000 gas
- Gas consumption is nearly 50% lower in the optimized version.

* Observations

1.Code structure optimization leads to reduced gas costs
2.Better storage usage and loop optimization improves transaction performance
3.Benchmarking tools are effective for identifying expensive operations and measuring improvements

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 Student:

Name :

Signature of the Faculty: ***Regn. No. :***

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