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 and optimize the gas consumption of smart contracts deployed on the Ethereum blockchain by implementing efficient coding practices and comparing different contract versions.

Apparatus/Software Used:

- Laptop/PC
- PowerPoint/Word for documentation
- Internet for research

Theory/Concept:

a. Gas in Blockchain:

In Ethereum, **gas** is the computational cost required to execute operations in a smart contract. Every function call, loop, or storage write consumes a certain amount of gas.

The **gas fee** is calculated as:

Gas Fee=Gas Used×Gas Price\text{Gas Fee} = \text{Gas Used} \times \text{Gas Price}Gas Fee=Gas Used×Gas Price

b. Importance of Gas Optimization:

Optimizing gas usage:

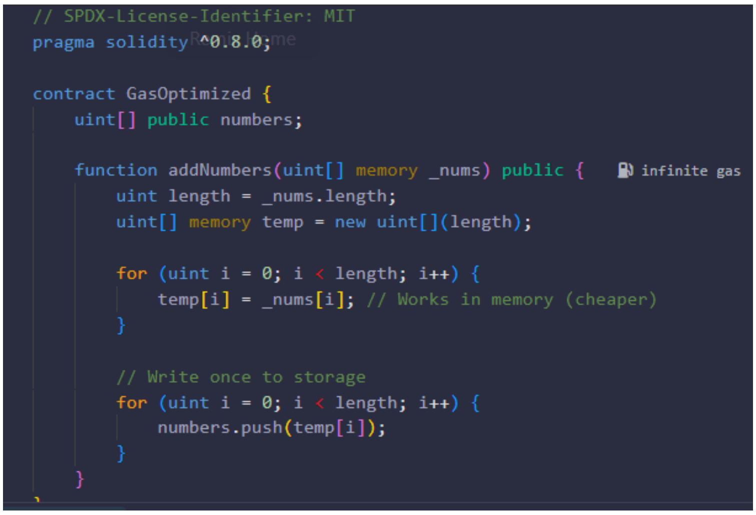
- Reduces transaction costs for users.
- Improves efficiency of smart contracts.
- Prevents unnecessary blockchain bloat.

c. Common Optimization Techniques:

1. Use `memory` instead of `storage` for temporary variables.
2. Avoid unnecessary loops or redundant calculations.

Procedure:

- Define two versions of a smart contract – unoptimized and optimized.
- Each contract will perform a similar operation (e.g., summing numbers or storing data).
- Deploy both contracts on Remix IDE.
- Use Remix Gas Reporter to record gas used for each function call.
- Compare results and identify which version is more efficient.
- Document and analyze the optimization impact.



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

contract GasOptimized {
    uint[] public numbers;

    function addNumbers(uint[] memory _nums) public {
        uint length = _nums.length;
        uint[] memory temp = new uint[](length);

        for (uint i = 0; i < length; i++) {
            temp[i] = _nums[i]; // Works in memory (cheaper)
        }

        // Write once to storage
        for (uint i = 0; i < length; i++) {
            numbers.push(temp[i]);
        }
    }
```

Observation:

- Writing directly to **storage** inside loops significantly increases gas cost.
- Using **memory** variables and minimizing state changes reduces gas usage.
- Each `SSTORE` (storage write) operation is expensive, costing ~20,000 gas.
- Optimized contracts perform the same logic with lower gas consumption and faster execution.

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**		