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

* Coding Phase: Pseudo Code / Flow Chart / Algorithm

ALGORITHM:

* Softwares used

- **Remix IDE** – For writing, compiling, and deploying Solidity smart contracts.
- **Solidity Compiler (solc)** – To compile and optimize smart contract code.
- **MetaMask** – For connecting to the Ethereum test network and executing transactions.
- **Ganache / Sepolia Testnet** – To simulate blockchain environment and measure gas usage.
- **Web3.js / Ethers.js** – For interacting with the contract and analyzing gas costs programmatically.

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

* Implementation Phase: Final Output (no error)

Gas optimization in Solidity enhances the cost-effectiveness and scalability of blockchain applications by minimizing computation, reducing redundant operations, and improving storage management.

1.Optimize Variable Scope and Storage

Limit the use of storage variables since they are the most expensive operations in Ethereum.

Use memory for temporary data and calldata for external input parameters.

```
// Less efficient
function updateData(uint[] memory arr) external { data = arr; }

// More efficient
function updateData(uint[] calldata arr) external { data = arr; }
```

2.Combine Multiple Updates in a Single Operation

Perform all arithmetic or logical computations in memory first before writing to blockchain storage.

```
// Inefficient
balance[msg.sender] += amount;
totalSupply += amount;

// Efficient
uint newBal = balance[msg.sender] + amount;
balance[msg.sender] = newBal;
totalSupply = totalSupply + amount;
```

3.Use Events Instead of Storage for Logging

Use emit events to record information rather than storing every detail on-chain.

```
event Transaction(address indexed user, uint value);
emit Transaction(msg.sender, amount);
```

4.Optimize Loops and Array Operations

Keep loops bounded or use mapping structures instead of large arrays to prevent high gas usage.

```
// Inefficient
for (uint i = 0; i < users.length; i++) {
    users[i].reward += 5;
}

// Better
mapping(address => uint) rewards;
rewards[msg.sender] += 5;
```

5. Use Constants, Immutables, and Struct Packing

Define unchanging variables as constant or immutable to save storage reads.

Pack smaller data types (like bool, uint8) in the same slot to reduce storage consumption.

```
uint8 public constant BONUS = 5;
address public immutable owner;
```

* Implementation Phase: Final Output (no error)

Tip	Benefit
Use `calldata` over `memory`	Lower call data cost
Pack variables	Reduce storage slots
Use `constant` and `immutable`	Save storage reads
Avoid unbounded loops	Prevent OOG errors
Cache external calls	Reduce redundancy
Batch operations	Fewer transactions

* Observations

1.Code optimization using proper data types and memory variables significantly reduces gas usage.
2.Efficient smart contracts lead to faster execution and lower deployment costs on the Ethereum network.

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 :

Regn. No. :

Signature of the Faculty:

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