Homework 22 MEV

1. Listen to the mempool using ether.js (or similar web3.py etc), you can use the code from the lesson.

```
// Use "ethers": "5.4.0"
const ethers = require('ethers');

// Connect to the mainnet
let provider = new
ethers.providers.JsonRpcProvider('https://mainnet.infura.io/v3/YOUR_API_KEY_
HERE');

provider.on('pending', (tx) => {
    provider.getTransaction(tx).then((transaction) => {
        console.log(transaction);
    })
});
```

2. Can you find a way to filter your mempool listener and get only pancake swap transactions?

```
// Use "ethers": "5.4.0"
const ethers = require('ethers');

// Connect to the mainnet
let provider = new
ethers.providers.JsonRpcProvider('https://mainnet.infura.io/v3/YOUR_API_KEY_
HERE');

// Define the contract address and abi for Pool USDC/USDT
let contractAddress = "0x04c8577958CcC170EB3d2CCa76F9d51bc6E42D8f";

let abi = [
    "event Swap(address indexed sender, address indexed recipient, int256
amount0, int256 amount1, uint160 sqrtPriceX96, uint128 liquidity, int24
tick, uint128 protocolFeesToken0, uint128 protocolFeesToken1)"
];

// Create a contract instance
let contract = new ethers.Contract(contractAddress, abi, provider);
```

```
// Listen for Swap events
contract.on("Swap", (sender, recipient, amount0, amount1, sqrtPriceX96,
liquidity, tick, protocolFeesToken0, protocolFeesToken1, event) => {
    console.log({
        sender: sender,
        recipient: recipient,
        amount0: amount0,
        amount1: amount1,
        sqrtPriceX96: sqrtPriceX96,
        liquidity: liquidity,
        tick: tick,
        protocolFeesToken0: protocolFeesToken0,
        protocolFeesToken1: protocolFeesToken1
    });
});
```

3. How might you mitigate MEV & front running if you were building your own Dapp?

- Use custom RPC endpoints that block MEV such as: https://mevblocker.io/
- Use flashbots to send transactions directly to miners
- Use MEV resistant Rollups
- Deploy to a layer 2 whether there is no mempool.
- Use chainlink Fair Sequencing Service (FSS) to get a fair ordering of transactions.

4. Have a look at the example sandwich bot and see make sure you understand how it works Repo.

Simple Check to make sure the caller is equal (eq()) to the memUser.

```
if iszero(eq(caller(), memUser)) {
                // Ohm (3, 3) makes your code more efficient
                // WGMI
                revert(3, 3)
            }
```

Here we are extracting vars from calldata. `CalldataLoad` 32bytes (a slot) from the specified start position. `shr` shifts the bits right to pad the value with zeros depending on the size of the var. For example:

`let token := shr(96, calldataload(0x00))` is loading the first parameter from calldata, which is an address (20 bytes). Addresses are 20 bytes, but EVM words are 32 bytes. So, it's necessary to shift right (shr) by 96 bits to remove the leading zeroes. The calldataload function loads 32 bytes from calldata from the specified start position, in this case 0x00.

```
    // bytes20
     let token := shr(96, calldataload(0x00))
     // bytes20
     let pair := shr(96, calldataload(0x14))
     // uint128
     let amountIn := shr(128, calldataload(0x28))
     // uint128
     let amountOut := shr(128, calldataload(0x38))
     // uint8
     let tokenOutNo := shr(248, calldataload(0x48))
```

Here we store the function signature for the transfer function in memory. Then we store the pair swap address and the amount to transfer.

Finally, we call the token contract setting making sure have at least 5000 gas left to execute the rest of the contract. If the call fails, we revert.

```
    // transfer function signature
         mstore(0x7c, ERC20_TRANSFER_ID)
         // destination
         mstore(0x80, pair)
         // amount
         mstore(0xa0, amountIn)

         let s1 := call(sub(gas(), 5000), token, 0, 0x7c, 0x44, 0, 0)
         if iszero(s1) {
             // WGMI
             revert(3, 3)
         }
```

In the next part we save the pair_swap_id to memory. Then we check if the tokenOutNo is 0 or 1. If it's 0, we save the amountOut to memory at 0x80 and 0 to 0xa0. If it's 1, we save 0 to 0x80 and amountOut to 0xa0. Basically setting the trade direction.

Then we save the address of the contract to memory at 0xc0 and 0x80 bytes of empty bytes to 0xe0.

Finally, we call the pair contract with the memory location of the pair_swap_id, the amountOut and the address of the contract. If the call fails, we revert.

```
mstore(0x7c, PAIR_SWAP_ID)
        // tokenOutNo == 0 ? ....
switch tokenOutNo
case 0 {
    mstore(0x80, amountOut)
    mstore(0xa0, 0)
}
case 1 {
    mstore(0x80, 0)
    mstore(0xa0, amountOut)
}
// address(this)
mstore(0xc0, address())
// empty bytes
mstore(0xe0, 0x80)

let s2 := call(sub(gas(), 5000), pair, 0, 0x7c, 0xa4, 0, 0)
if iszero(s2) {
    revert(3, 3)
}
```