Assignment 1

***The detailed code can be found in the appendix at the end of the document, or by visiting the GitHub repository: https://github.com/SamW-butW/SC6121-Tokenomics-Assignment1.git**

1. Question 1

1.1 Describe what happens in the transaction to the Uniswap v2: ETH-USDC pool:

Taken out of the pool: 1024.229 USDC

Sent into the pool: 0.590938840873296854 WETH

1.2 What is the position of this transaction within that block ? How does transaction gets their order no within the block? why does it matter ?

① **Position in Block: 27.**

Which means this transaction is the 28th transaction in block 17462685.

② **How Order Is Determined:** Miners have the right to freely decide the order of transactions in this block (only multiple transactions with the same address need to be arranged in ascending order by nonce). Usually miners will prioritize transactions with higher gas prices to maximize revenue; transactions with the same address are sorted by nonce to ensure logical consistency.

③ **Why It Matters:** The execution order of transactions will affect the state on the chain (such as price, balance), and is also the root cause of MEV (miner extractable value), which is related to strategies such as front-end attacks and mezzanine transactions.

1.3 Write a python code to simulate this transaction.

```python
def get_amount_out(amount_in: int, reserve_in: int, reserve_out: int, fee: float) -> int:
    fee_numerator = int((1 - fee) * 1000)
    fee_denominator = 1000
    amount_in_with_fee = amount_in * fee_numerator
    numerator = amount_in_with_fee * reserve_out
    denominator = reserve_in * fee_denominator + amount_in_with_fee
    return numerator // denominator


def main():
    print("=== Uniswap V2 Swap Simulation ===\n")
    amount_in = int(input("1) Enter amount_in (raw units): "))
    token_in  = input("2) Enter token_in symbol: ")
    token0    = input("3) Enter symbol of token0: ")
    token1    = input("4) Enter symbol of token1: ")
    reserve0  = int(input(f"5) Enter reserve0 for {token0}: "))
    reserve1  = int(input(f"6) Enter reserve1 for {token1}: "))
    fee       = float(input("7) Enter fee rate (e.g. 0.003): "))
    if token_in == token0:
        reserve_in, reserve_out = reserve0, reserve1
        token_out = token1
    elif token_in == token1:
        reserve_in, reserve_out = reserve1, reserve0
        token_out = token0
    else:
        print(f"Error: token_in ({token_in}) must be {token0} or {token1}.")
        return
    amount_out = get_amount_out(amount_in, reserve_in, reserve_out, fee)
    print(f"\nSimulation result: Output {token_out} = {amount_out}")
if __name__ == "__main__":
    main()
```

1.4 Test the code

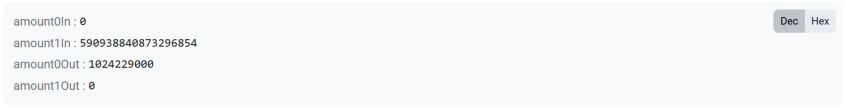
The results obtained are exactly the same as the data in etherscan：1024229000

Note: before the swap transaction, reserve0=26853692230452; reserve1=15446428142167535900067

```
PS E:\maplestory> python homework1.py
=== Uniswap V2 Swap Simulation (Exact On-Chain) ===

1) Enter amount_in (raw units): 590938840873296854
2) Enter token_in symbol (e.g. USDC or WETH): WETH
3) Enter symbol of token0 in the pool: USDC
4) Enter symbol of token1 in the pool: WETH
5) Enter reserve0 for USDC (raw units): 26853692230452
6) Enter reserve1 for WETH (raw units): 15446428142167535900067
7) Enter fee rate (e.g. 0.003 for 0.3%): 0.003

Simulation result:
  Output USDC = 1024229000 (raw units)
```



2. Question 2

The relevant data is obtained through alchemy's Ethereum RPC URL

```
RPC_URL = "https://eth-mainnet.g.alchemy.com/v2/R8R1RiTNWjf95C2-ZnRsYyoP11ysBLwa"
w3 = Web3(Web3.HTTPProvider(RPC_URL))
POOL_ADDR = "0x88e6A0c2dDD26FEEb64F039a2c41296FcB3f5640"
```

2.1 Task1：

① Balance inside the position at block 17618742：

- 84847.420955 USDC； 6.374503 WETH

② Impermanent Loss vs. HODL: 2733.74 USDC

Task1 analysis：

① Step1：Fetching On-Chain Price

```
def get_sqrtPriceX96(block: int) -> int:
    sqX96, *_ = pool_slot0.functions.slot0().call(block_identifier=block)
    return sqX96
def get_price_usdc_per_weth(block: int) -> float:
    sqX96 = get_sqrtPriceX96(block)
    raw = sqX96 / 2**96
    price_raw = raw * raw
    return (1 / price_raw) * 1e12
```

slot0.sqrtPriceX96 stores √P in Q96 fixed-point.

Convert to float: √P = sqrtPriceX96 / 2^96; then P = (√P)^2.

Invert and scale by 1e12 to get USDC per WETH (USDC has 6 decimals, WETH 18).

P0 ≈ 1952.753762 USDC/WETH; P1 ≈ 1954.261780 USDC/WETH.

② Step2：Calculate Initial WETH Deposit & Liquidity L

```
W0 = 50_000.0 / P0
amount1_wei = int(W0 * 1e18)
sqrtP0    = get_sqrtPriceX96(17618642)
sqrtLower = int((1.0001**(200540/2)) * 2**96)
L = amount1_wei * 2**96 // (sqrtP0 - sqrtLower)
```

Split 100 000 USDC into 50000 USDC + 25.604867 WETH by value.
Compute Q96-encoded ticks for lower boundary.
Uniswap V3 liquidity formula: L = Δy·2^96 / (√P0 – √Plower)
where Δy is WETH deposit in wei. Result: L ≈ 2.202082 × 10^18.

③ Step3：Extracting Position Balances at Exit

```
def get_position_amounts(L: int, block:int, lower:int, upper:int):
    sqrtP = get_sqrtPriceX96(block)
    sqrtL = int((1.0001**(lower/2)) * 2**96)
    sqrtU = int((1.0001**(upper/2)) * 2**96)
    amount1 = L * (sqrtP - sqrtL) // 2**96
    num   = L * (sqrtU - sqrtP) * 2**96
    denom = sqrtP * sqrtU
    amount0 = num // denom
    return amount0, amount1
a0, a1 = get_position_amounts(L, 17618742, 200540, 200560)
amount0_usdc = a0 / 1e6
amount1_weth = a1 / 1e18
print(f"Exit balances: {amount0_usdc:.6f} USDC, {amount1_weth:.6f} WETH")
```

Uniswap V3 continuously rebalances: as price moves up, it swaps WETH → USDC inside your tick range.Final tokens at block 17618742: USDC: 84847.420955、WETH: 6.374503

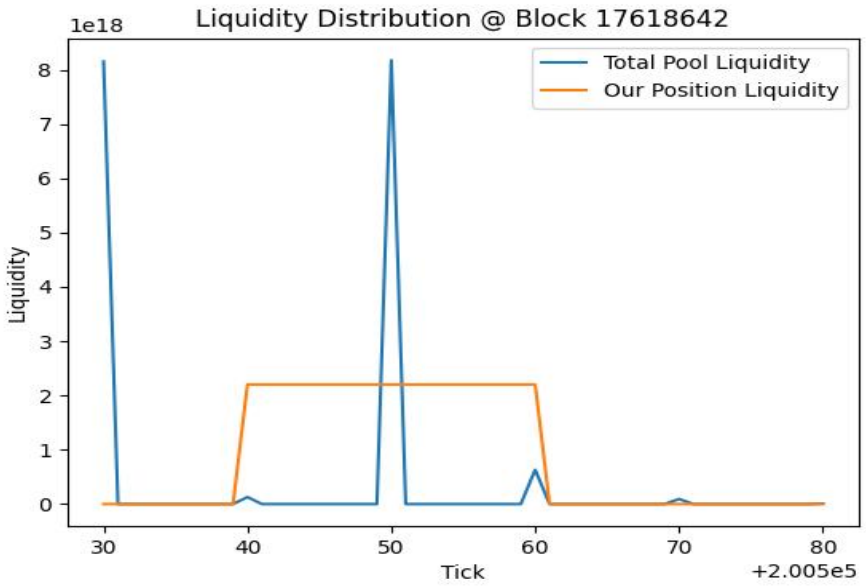
④ Impermanent Loss vs. HODL

```
hodl_value = 50_000.0 + W0 * P1
lp_value   = amount0_usdc + amount1_weth * P1
impermanent_loss = hodl_value - lp_value
print(f"HODL ≈ {hodl_value:.2f} USDC, LP ≈ {lp_value:.2f} USDC, IL ≈ {impermanent_loss:.2f} USDC")
```

HODL: hold 50000 USDC + 25.604867 WETH → value ≈ 100038.61 USDC at P1.
LP: value of exit balances ≈ 97 304.87 USDC.
Impermanent loss = 100038.61 – 97304.87 = 2733.74 USDC.

2.2 Task2

Answer：



Task2 analysis

```
import pandas as pd
import matplotlib.pyplot as plt
ticks = list(range(200530, 200581))
global_liq = [
    8151361812842047647, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    129145973139416183, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    8174698218376832778, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    631520865104068728, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    92961962089749322, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    8840534690793920
]
L_our = 2202082411454851840
our_liq = [L_our if 200540 <= t <= 200560 else 0 for t in ticks]
df = pd.DataFrame({
    'tick': ticks,
    'global_liquidity': global_liq,
    'our_liquidity': our_liq
})
plt.figure()
plt.plot(df['tick'], df['global_liquidity'], label='Total Pool Liquidity')
plt.plot(df['tick'], df['our_liquidity'], label='Our Position Liquidity')
plt.xlabel('Tick')
plt.ylabel('Liquidity')
plt.title('Liquidity Distribution @ Block 17618642')
plt.legend()
plt.show()
```

Explanation：

① Total Pool Liquidity

Tick 200530: 8151361812842047647; Tick 200540: 129145973139416183;

Tick 200550: 817469821837683277 8; Tick 200560: 631520865104068728;

Tick 200570: 92961962089749322; Tick 200580: 8840534690793920;

② Our Position Liquidity

We provided a uniform liquidity of 2.202082×10^{18} across every tick from 200540 to 200560, and zero elsewhere.Visually, this appears as a flat “block” between ticks 200540 and 200560.

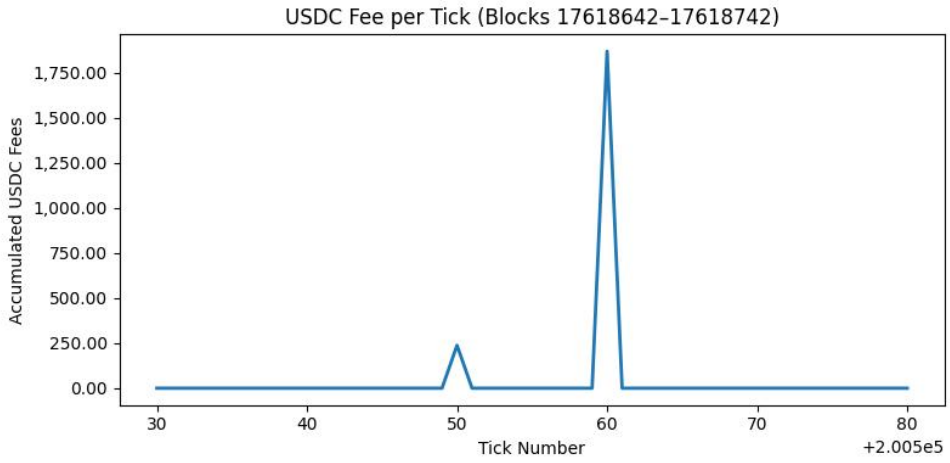
③ Relative Share

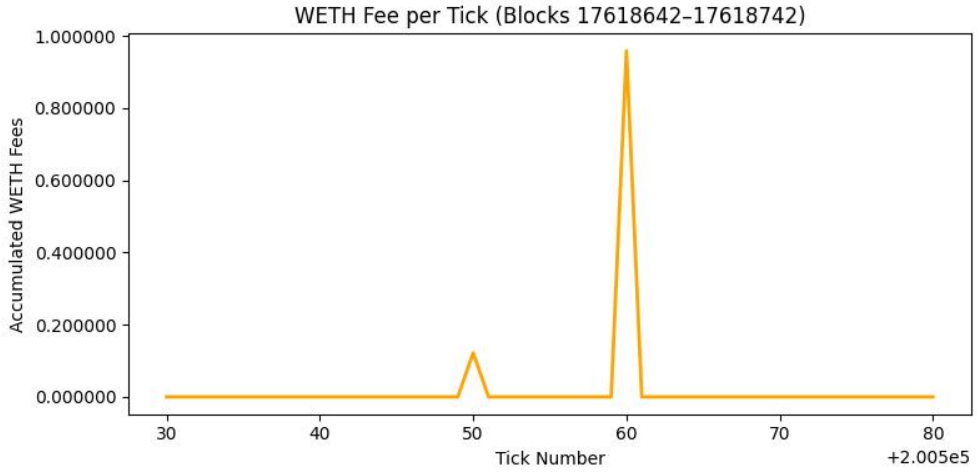
At ticks where both pool and our liquidity exist (200540, 200550, 200560), the share is:

$$\frac{L_{\text{our}}}{L_{\text{pool}}} \approx \begin{cases} 2.20e18/1.29e17 \approx 17.1\times & (\text{tick } 200540) \\ 2.20e18/8.17e18 \approx 27.0\% & (\text{tick } 200550) \\ 2.20e18/6.32e17 \approx 3.5\times & (\text{tick } 200560) \end{cases}$$

2.3 Task3

Answer：





Task3 analysis

① step1 Compute Per-Tick Weight of Our Position

```
# Fetch the pool's liquidity per tick at START_BLOCK
global_liq = fetch_tick_liquidity(START_BLOCK)
# Compute our position's liquidity L at START_BLOCK
L_our   = compute_L_our(START_BLOCK)
# Map each tick to our liquidity (nonzero only between TICK_LOWER and TICK_UPPER)
our_liq = {
    t: (L_our if TICK_LOWER <= t <= TICK_UPPER else 0)
    for t in global_liq
}
# Weight = (our_liq / global_liq) whenever pool has liquidity
weight = {
    t: (our_liq[t] / global_liq[t]) if global_liq[t] > 0 else 0
    for t in global_liq
}
```

read the pool's on-chain liquidityGross at each tick in [200530–200580].

compute own position's liquidity L_our at the same block

The weight at each tick is our share of total liquidity, used to pro-rate fee accrual.

② Fetch and Decode Swap Events

```
# Build the Swap event topic
swap_sig    = "Swap(address,address,int256,int256,uint160,uint128,int24)"
event_topic = w3.keccak(text=swap_sig).hex()
# Retrieve logs between START_BLOCK and END_BLOCK
logs = w3.eth.get_logs({
    "fromBlock": START_BLOCK,
    "toBlock":   END_BLOCK,
    "address":   POOL_ADDR,
    "topics":    [event_topic]
})
print("  raw logs count:", len(logs))
# Decode each log into a Swap event
events = [pool_swap.events.Swap.process_log(lg) for lg in logs]
print("✅ decoded swap events count:", len(events))
```

pull all Swap events for the pool in the 100-block window and decode them to access: amount0, amount1,sqrtPriceX96

③ Allocate Fees Across Crossed Ticks

```
fee0 = {t: 0 for t in global_liq}  # USDC fees
fee1 = {t: 0 for t in global_liq}  # WETH fees
for ev in events:
    a0, a1 = abs(ev["args"]["amount0"]), abs(ev["args"]["amount1"])
    pre    = get_sqrtPriceX96(ev["blockNumber"] - 1) / 2**96
    post   = ev["args"]["sqrtPriceX96"] / 2**96
```

```
# Determine which tick range the price crossed
    low, high = sorted((raw_sqrt_to_tick(pre), raw_sqrt_to_tick(post)))
    crossed   = [t for t in range(low, high + 1)
                 if START_TICK <= t <= END_TICK]
    if not crossed:
        continue
    # Total fee on this swap
    f0 = a0 * FEE_RATE
    f1 = a1 * FEE_RATE
    # Distribute equally across crossed ticks, weighted by our share
    per0 = f0 / len(crossed)
    per1 = f1 / len(crossed)
    for t in crossed:
        fee0[t] += per0 * weight[t]
        fee1[t] += per1 * weight[t]
```

Compute absolute USDC/WETH moved, apply 0.3% fee rate;
Identify ticks crossed by price movement; Split each swap's fees evenly across those ticks;
Multiply by our tick weight to estimate fees earned by our position.

④ Plotting Accumulated Fees

2.4 Task4

① Estimated swap fees earned

- In USDC: 2 107.5336 USDC

- In WETH: 1.080420 WETH

② Portfolio PnL (in USDC): +1 523.83 USDC

Task4 analysis

① Calculating Fee Income

```
# Calculate the fees we earned
total_fee_usdc = sum(fee0.values()) / 1e6
total_fee_weth = sum(fee1.values()) / 1e18
print(f"Estimated fees earned: {total_fee_usdc:.4f} USDC, {total_fee_weth:.6f} WETH")
```

USDC fees: 2 107.5336 USDC, WETH fees: 1.080420 WETH.
Converting WETH fees at the exit price P1≈1954.261780 USDC/WETH:
1.080420×1954.261780≈2111.43USDC
Total fee income ≈ 2107.53 + 2111.43 = 4218.96 USDC

② Position Value at Exit (Excluding Fees)

```
# Compute the end-of-period token balances
a0_raw, a1_raw = get_position_amounts(L_our, END_BLOCK, TICK_LOWER, TICK_UPPER)
amount0_usdc = a0_raw / 1e6
amount1_weth = a1_raw / 1e18
print(f"Position at block {END_BLOCK}: {amount0_usdc:.6f} USDC, {amount1_weth:.6f} WETH")
# Compute LP value and convert fee value to USDC
P1         = get_price_usdc_per_weth(END_BLOCK)
lp_value   = amount0_usdc + amount1_weth * P1
fees_value = total_fee_usdc + total_fee_weth * P1
print(f"LP value (ex-cl fees): {lp_value:.2f} USDC")
print(f"Fees value:            {fees_value:.2f} USDC")
```

USDC balance: 84847.420955、WETH balance: 6.374503

LP position value: 84847.42+6.374503×1954.26178≈97304.87USDC

Fees converted to USDC: 2107.53+(1.080420×1954.26178)≈4218.96USDC

③ Portfolio PnL Calculation

```
initial = 100000.0
pnl     = lp_value + fees_value - initial
print(f"→ Portfolio PnL:      {pnl:.2f} USDC")
```

Starting capital: 100000 USDC

Ending value: 97304.87 + 4218.96 = 101523.83 USDC

Net profit: 101523.83 − 100000 = +1523.83 USD

Appendix (Code)

1. Question1 Code

```
def get_amount_out(amount_in: int, reserve_in: int, reserve_out: int, fee: float) -> int:
    fee_numerator = int((1 - fee) * 1000)
    fee_denominator = 1000
    amount_in_with_fee = amount_in * fee_numerator
    numerator = amount_in_with_fee * reserve_out
    denominator = reserve_in * fee_denominator + amount_in_with_fee
    return numerator // denominator
def main():
    print("=== Uniswap V2 Swap Simulation ===\n")
    amount_in = int(input("1) Enter amount_in (raw units): "))
    token_in  = input("2) Enter token_in symbol: ")
    token0    = input("3) Enter symbol of token0: ")
    token1    = input("4) Enter symbol of token1: ")
    reserve0  = int(input(f"5) Enter reserve0 for {token0}: "))
    reserve1  = int(input(f"6) Enter reserve1 for {token1}: "))
    fee       = float(input("7) Enter fee rate (e.g. 0.003): "))
    if token_in == token0:
        reserve_in, reserve_out = reserve0, reserve1
        token_out = token1
    elif token_in == token1:
        reserve_in, reserve_out = reserve1, reserve0
        token_out = token0
    else:
        print(f"Error: token_in ({token_in}) must be {token0} or {token1}.")
        return
    amount_out = get_amount_out(amount_in, reserve_in, reserve_out, fee)
    print(f"\nSimulation result: Output {token_out} = {amount_out}")
if __name__ == "__main__":
    main()
```

2. Question2 Code

Task1:

```
from web3 import Web3
import math


RPC_URL = "https://eth-mainnet.g.alchemy.com/v2/R8R1RiTNWjf95C2-ZnRsYyoP11ysBLwa"
w3 = Web3(Web3.HTTPProvider(RPC_URL))
POOL_ADDR = "0x88e6A0c2dDD26FEEb64F039a2c41296FcB3f5640"
```

```
TICKS_ABI = [{
    "inputs":[{"internalType":"int24","name":"tick","type":"int24"}],
    "name":"ticks",
    "outputs":[
        {"internalType":"uint128","name":"liquidityGross","type":"uint128"},
        {"internalType":"int128","name":"liquidityNet","type":"int128"}
    ],
    "stateMutability":"view","type":"function"
}]
SLOT0_ABI = [{
    "inputs": [], "name": "slot0",
    "outputs": [
        {"internalType":"uint160","name":"sqrtPriceX96","type":"uint160"},
        {"internalType":"int24","name":"tick","type":"int24"},
        {"internalType":"uint16","name":"observationIndex","type":"uint16"},
        {"internalType":"uint16","name":"observationCardinality","type":"uint16"},
        {"internalType":"uint16","name":"observationCardinalityNext","type":"uint16"},
        {"internalType":"uint8","name":"feeProtocol","type":"uint8"},
        {"internalType":"bool","name":"unlocked","type":"bool"}
    ],
    "stateMutability":"view","type":"function"
}]
pool_slot0 = w3.eth.contract(address=POOL_ADDR, abi=SLOT0_ABI)
def get_sqrtPriceX96(block: int) -> int:
    sqX96, *_ = pool_slot0.functions.slot0().call(block_identifier=block)
    return sqX96
def get_price_usdc_per_weth(block: int) -> float:
    sqX96 = get_sqrtPriceX96(block)
    raw = sqX96 / 2**96
    price_raw = raw * raw
    return (1 / price_raw) * 1e12
def get_L_our(block:int) -> int:
    P0 = get_price_usdc_per_weth(block)
    print(f"P0 (USDC per WETH) at block {block}: {P0:.6f}")
    W0 = 50_000.0 / P0
    amount1 = int(W0 * 1e18)
    print(f"Initial WETH to deposit: {W0:.6f} WETH ({amount1} wei)")
    sqrtP0 = get_sqrtPriceX96(block)
    print(f"sqrtPriceX96 at block {block}: {sqrtP0}")
    raw_sqrtP0 = sqrtP0 / 2**96
    print(f"raw_sqrtP0 (√(res1/res0)) : {raw_sqrtP0:.6e}")
    sqrtL = int((1.0001**(200540/2)) * 2**96)
    raw_sqrtL = sqrtL / 2**96
```

```python
    print(f"sqrtLowerX96 (tick 200540): {sqrtL}, raw_sqrtLower: {raw_sqrtL:.6e}")
    L = amount1 * 2**96 // (sqrtP0 - sqrtL)
    print(f"Calculated L_our (int): {L}")
    print(f"Calculated L_our (scientific): {L:.6e}")
    return L

def get_position_amounts(L: int, block:int, lower:int, upper:int):
    sqrtP = get_sqrtPriceX96(block)
    raw_sqrtP = sqrtP / 2**96
    sqrtL  = int((1.0001**(lower/2)) * 2**96)
    sqrtU  = int((1.0001**(upper/2)) * 2**96)
    raw_sqrtL = sqrtL / 2**96
    raw_sqrtU = sqrtU / 2**96
    print(f"\nsqrtPriceX96 at block {block}: {sqrtP}, raw_sqrtP: {raw_sqrtP:.6e}")
    print(f"sqrtLowerX96: {sqrtL}, raw_sqrtLower: {raw_sqrtL:.6e}")
    print(f"sqrtUpperX96: {sqrtU}, raw_sqrtUpper: {raw_sqrtU:.6e}")
    amount1 = L * (sqrtP - sqrtL) // (2**96)
    num     = L * (sqrtU - sqrtP) * (2**96)
    denom   = sqrtP * sqrtU
    amount0 = num // denom
    print(f"Raw amount1 (WETH wei): {amount1}")
    print(f"Raw amount0 (USDC units): {amount0}")
    return amount0, amount1

if __name__ == "__main__":
    BLOCK0 = 17618642
    BLOCK1 = 17618742
    LOWER, UPPER = 200540, 200560
    L_our = get_L_our(BLOCK0)
    # 取出头寸并打印
    a0, a1 = get_position_amounts(L_our, BLOCK1, LOWER, UPPER)
    amount0_usdc = a0 / 1e6
    amount1_weth = a1 / 1e18
    print(f"\nPosition at block {BLOCK1}: {amount0_usdc:.6f} USDC, {amount1_weth:.6f} WETH")
    P1 = get_price_usdc_per_weth(BLOCK1)
    print(f"P1 (USDC per WETH) at block {BLOCK1}: {P1:.6f}")
    W0 = 50_000.0 / get_price_usdc_per_weth(BLOCK0)
    hodl_val = 50_000.0 + W0 * P1
    lp_val   = amount0_usdc + amount1_weth * P1
    print(f"HODL value: {hodl_val:.2f} USDC")
    print(f"LP   value: {lp_val:.2f} USDC")
    print(f"Impermanent Loss: {hodl_val - lp_val:.2f} USDC")
```

Task 2:

```python
from web3 import Web3
RPC_URL = "https://eth-mainnet.g.alchemy.com/v2/R8R1RiTNWjf95C2-ZnRsYyoP11ysBLwa"
w3 = Web3(Web3.HTTPProvider(RPC_URL))
POOL_ADDR = "0x88e6A0c2dDD26FEEb64F039a2c41296FcB3f5640"
POOL_ABI = [
    {
        "inputs":[{"internalType":"int24","name":"tick","type":"int24"}],
        "name":"ticks",
        "outputs":[
            {"internalType":"uint128","name":"liquidityGross","type":"uint128"},
            {"internalType":"int128","name":"liquidityNet","type":"int128"}
        ],
        "stateMutability":"view",
        "type":"function"
    }
]
pool = w3.eth.contract(address=POOL_ADDR, abi=POOL_ABI)
def fetch_tick_liquidity(start_tick: int, end_tick: int, block: int):
    result = {}
    for tick in range(start_tick, end_tick + 1):
        info = pool.functions.ticks(tick).call(block_identifier=block)
        result[tick] = info[0]
    return result
if __name__ == "__main__":
    START, END = 200530, 200580
    BLOCK = 17618642
    data = fetch_tick_liquidity(START, END, BLOCK)
    for t, liq in data.items():
        print(f"Tick {t}: {liq}")
```

```python
import pandas as pd
import matplotlib.pyplot as plt

# 从之前脚本输出中抄入数据
ticks = list(range(200530, 200581))
global_liq = [
    8151361812842047647, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    129145973139416183, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    8174698218376832778, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    631520865104068728, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    92961962089749322, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    8840534690793920
```

```
]
L_our = 2202082411454851840
our_liq = [L_our if 200540 <= t <= 200560 else 0 for t in ticks]


df = pd.DataFrame({
    'tick': ticks,
    'global_liquidity': global_liq,
    'our_liquidity': our_liq
})


plt.figure()
plt.plot(df['tick'], df['global_liquidity'], label='Total Pool Liquidity')
plt.plot(df['tick'], df['our_liquidity'], label='Our Position Liquidity')
plt.xlabel('Tick')
plt.ylabel('Liquidity')
plt.title('Liquidity Distribution @ Block 17618642')
plt.legend()
plt.show()
```

Task3:

```
from web3 import Web3
import math
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


# ── 配置 ──────────────────────────────────────────────────────
RPC_URL      = "https://eth-mainnet.g.alchemy.com/v2/R8R1RiTNWjf95C2-ZnRsYyoP11ysBLwa"
POOL_ADDR    = "0x88e6A0c2dDD26FEEb64F039a2c41296FcB3f5640"
START_BLOCK  = 17618642
END_BLOCK    = 17618742
START_TICK   = 200530
END_TICK     = 200580
TICK_LOWER   = 200540
TICK_UPPER   = 200560
FEE_RATE     = 0.003  # 0.3%
w3 = Web3(Web3.HTTPProvider(RPC_URL))
# ── ABI ──────────────────────────────────────────────────────
TICKS_ABI = [{
    "inputs":[{"internalType":"int24","name":"tick","type":"int24"}],
    "name":"ticks","outputs":[
      {"internalType":"uint128","name":"liquidityGross","type":"uint128"},
```

```
      {"internalType":"int128","name":"liquidityNet","type":"int128"}],
    "stateMutability":"view","type":"function"}]
SLOT0_ABI =[{
    "inputs": [], "name": "slot0", "outputs":[
      {"internalType":"uint160","name":"sqrtPriceX96","type":"uint160"},
      {"internalType":"int24","name":"tick","type":"int24"},
      {"internalType":"uint16","name":"observationIndex","type":"uint16"},
      {"internalType":"uint16","name":"observationCardinality","type":"uint16"},
      {"internalType":"uint16","name":"observationCardinalityNext","type":"uint16"},
      {"internalType":"uint8","name":"feeProtocol","type":"uint8"},
      {"internalType":"bool","name":"unlocked","type":"bool"}],
    "stateMutability":"view","type":"function"}]
SWAP_ABI = [{
    "anonymous": False,
    "inputs": [
      {"indexed":True,  "internalType":"address","name":"sender","type":"address"},
      {"indexed":True,  "internalType":"address","name":"recipient","type":"address"},
      {"indexed":False, "internalType":"int256",  "name":"amount0","type":"int256"},
      {"indexed":False, "internalType":"int256",  "name":"amount1","type":"int256"},
      {"indexed":False, "internalType":"uint160", "name":"sqrtPriceX96","type":"uint160"},
      {"indexed":False, "internalType":"uint128", "name":"liquidity","type":"uint128"},
      {"indexed":False, "internalType":"int24",   "name":"tick","type":"int24"}],
    "name":"Swap","type":"event"}]
pool_ticks = w3.eth.contract(address=POOL_ADDR, abi=TICKS_ABI)
pool_slot0 = w3.eth.contract(address=POOL_ADDR, abi=SLOT0_ABI)
pool_swap  = w3.eth.contract(address=POOL_ADDR, abi=SWAP_ABI)
def fetch_tick_liquidity(block: int):
    data = {}
    for t in range(START_TICK, END_TICK+1):
        lg, _ = pool_ticks.functions.ticks(t).call(block_identifier=block)
        data[t] = lg
    return data
def get_sqrtPriceX96(block: int) -> int:
    return pool_slot0.functions.slot0().call(block_identifier=block)[0]
def get_price_usdc_per_weth(block: int) -> float:
    sqX96 = get_sqrtPriceX96(block)
    raw   = sqX96 / 2**96
    price_raw = raw * raw
    return (1 / price_raw) * 1e12
def compute_L_our(block: int) -> int:
    P0       = get_price_usdc_per_weth(block)
    weth_amt = 50_000.0 / P0
    amount1  = int(weth_amt * 1e18)
    sqrtP0   = get_sqrtPriceX96(block)
```

```
    sqrtL    = int((1.0001**(TICK_LOWER/2)) * 2**96)
    return amount1 * 2**96 // (sqrtP0 - sqrtL)
def raw_sqrt_to_tick(raw_sqrt: float) -> int:
    return int(math.log(raw_sqrt*raw_sqrt) / math.log(1.0001))
# 1) 全局 & 头寸流动性 & 权重
global_liq = fetch_tick_liquidity(START_BLOCK)
L_our      = compute_L_our(START_BLOCK)
our_liq    = {t: (L_our if TICK_LOWER <= t <= TICK_UPPER else 0) for t in global_liq}
weight     = {t: (our_liq[t]/global_liq[t]) if global_liq[t] > 0 else 0 for t in global_liq}
# 2) 拉日志：一定要正确签名
swap_sig    = "Swap(address,address,int256,int256,uint160,uint128,int24)"
event_topic = w3.keccak(text=swap_sig).hex()
logs = w3.eth.get_logs({
    "fromBlock": START_BLOCK,
    "toBlock":   END_BLOCK,
    "address":   POOL_ADDR,
    "topics":    [event_topic]
})
print("  raw logs count:", len(logs))
# 3) 解码：用 process_log
events = [pool_swap.events.Swap.process_log(lg) for lg in logs]
print("✅ decoded swap events count:", len(events))
# 4) 累计平均分配手续费
fee0 = {t: 0 for t in global_liq}  # USDC fees (raw)
fee1 = {t: 0 for t in global_liq}  # WETH fees (raw)
for ev in events:
    args = ev["args"]
    blk  = ev["blockNumber"]
    pre_raw   = get_sqrtPriceX96(blk-1) / 2**96
    post_raw  = args["sqrtPriceX96"] / 2**96
    pre_tick  = raw_sqrt_to_tick(pre_raw)
    post_tick = raw_sqrt_to_tick(post_raw)
    low, high = sorted((pre_tick, post_tick))
    crossed   = [t for t in range(low, high+1) if START_TICK <= t <= END_TICK]
    if not crossed: continue
    f0   = abs(args["amount0"]) * FEE_RATE
    f1   = abs(args["amount1"]) * FEE_RATE
    per0 = f0 / len(crossed)
    per1 = f1 / len(crossed)
    for t in crossed:
        fee0[t] += per0 * weight[t]
        fee1[t] += per1 * weight[t]

ticks = list(global_liq.keys())
```

```
# USDC 图
usdc_vals = [fee0[t] / 1e6 for t in ticks]
plt.figure(figsize=(8,4))
plt.plot(ticks, usdc_vals, linestyle='-', linewidth=2)          # 去掉 marker
plt.xlabel("Tick Number")
plt.ylabel("Accumulated USDC Fees")
plt.title(f"USDC Fee per Tick (Blocks {START_BLOCK}-{END_BLOCK})")
plt.gca().yaxis.set_major_formatter(
    ticker.FuncFormatter(lambda x, _: f"{x:,.2f}")
)
plt.tight_layout()
plt.show()
# WETH 图
weth_vals = [fee1[t] / 1e18 for t in ticks]
plt.figure(figsize=(8,4))
plt.plot(ticks, weth_vals, color='orange', linestyle='-', linewidth=2)  # 同样去掉 marker
plt.xlabel("Tick Number")
plt.ylabel("Accumulated WETH Fees")
plt.title(f"WETH Fee per Tick (Blocks {START_BLOCK}-{END_BLOCK})")
plt.gca().yaxis.set_major_formatter(
    ticker.FuncFormatter(lambda x, _: f"{x:,.6f}")
)
plt.tight_layout()
plt.show()
total_weth_fee = sum(fee1.values()) / 1e18
print(f"Total WETH fee earned: {total_weth_fee:.6f} WETH")
print(f"Max WETH fee on a single tick: {max(fee1.values()) / 1e18:.6f} WETH")
```

Task4：

```
from web3 import Web3
import math
import pandas as pd


RPC_URL      = "https://eth-mainnet.g.alchemy.com/v2/R8R1RiTNWjf95C2-ZnRsYyoP11ysBLwa"
POOL_ADDR    = "0x88e6A0c2dDD26FEEb64F039a2c41296FcB3f5640"
START_BLOCK  = 17618642
END_BLOCK    = 17618742
START_TICK   = 200530
END_TICK     = 200580
TICK_LOWER   = 200540
TICK_UPPER   = 200560
FEE_RATE     = 0.003  # 0.3%
```

```
w3 = Web3(Web3.HTTPProvider(RPC_URL))
TICKS_ABI = [{
    "inputs":[{"internalType":"int24","name":"tick","type":"int24"}],
    "name":"ticks","outputs":[
      {"internalType":"uint128","name":"liquidityGross","type":"uint128"},
      {"internalType":"int128","name":"liquidityNet","type":"int128"}],
    "stateMutability":"view","type":"function"}]
SLOT0_ABI =[{
    "inputs": [], "name": "slot0", "outputs":[
      {"internalType":"uint160","name":"sqrtPriceX96","type":"uint160"},
      {"internalType":"int24","name":"tick","type":"int24"},
      {"internalType":"uint16","name":"observationIndex","type":"uint16"},
      {"internalType":"uint16","name":"observationCardinality","type":"uint16"},
      {"internalType":"uint16","name":"observationCardinalityNext","type":"uint16"},
      {"internalType":"uint8","name":"feeProtocol","type":"uint8"},
      {"internalType":"bool","name":"unlocked","type":"bool"}],
    "stateMutability":"view","type":"function"}]
SWAP_ABI = [{
    "anonymous": False,
    "inputs": [
      {"indexed":True,  "internalType":"address","name":"sender","type":"address"},
      {"indexed":True,  "internalType":"address","name":"recipient","type":"address"},
      {"indexed":False, "internalType":"int256",  "name":"amount0","type":"int256"},
      {"indexed":False, "internalType":"int256",  "name":"amount1","type":"int256"},
      {"indexed":False, "internalType":"uint160", "name":"sqrtPriceX96","type":"uint160"},
      {"indexed":False, "internalType":"uint128", "name":"liquidity","type":"uint128"},
      {"indexed":False, "internalType":"int24",   "name":"tick","type":"int24"}],
    "name":"Swap","type":"event"}]
pool_ticks = w3.eth.contract(address=POOL_ADDR, abi=TICKS_ABI)
pool_slot0 = w3.eth.contract(address=POOL_ADDR, abi=SLOT0_ABI)
pool_swap  = w3.eth.contract(address=POOL_ADDR, abi=SWAP_ABI)
def fetch_tick_liquidity(block: int):
    d={}
    for t in range(START_TICK, END_TICK+1):
        lg,_ = pool_ticks.functions.ticks(t).call(block_identifier=block)
        d[t]=lg
    return d
def get_sqrtPriceX96(block:int)->int:
    return pool_slot0.functions.slot0().call(block_identifier=block)[0]
def get_price_usdc_per_weth(block:int)->float:
    s = get_sqrtPriceX96(block)/2**96
    return (1/(s*s))*1e12
def compute_L_our(block:int)->int:
    P0 = get_price_usdc_per_weth(block)
```

```
    weth_amt = 50000.0/P0
    amt1 = int(weth_amt*1e18)
    sqrtP0 = get_sqrtPriceX96(block)
    sqrtL  = int((1.0001**(TICK_LOWER/2))*2**96)
    return amt1*2**96//(sqrtP0-sqrtL)
def raw_sqrt_to_tick(r:float)->int:
    return int(math.log(r*r)/math.log(1.0001))
def get_position_amounts(L:int, block:int, lower:int, upper:int):
    sqrtP = get_sqrtPriceX96(block)
    sqrtL = int((1.0001**(lower/2))*2**96)
    sqrtU = int((1.0001**(upper/2))*2**96)
    amt1 = L*(sqrtP-sqrtL)//(2**96)
    num = L*(sqrtU-sqrtP)*(2**96)
    den = sqrtP*sqrtU
    amt0 = num//den
    return amt0, amt1
# 1) 全局 & 权重
global_liq = fetch_tick_liquidity(START_BLOCK)
L_our      = compute_L_our(START_BLOCK)
our_liq    = {t:(L_our if TICK_LOWER<=t<=TICK_UPPER else 0) for t in global_liq}
weight     = {t:our_liq[t]/global_liq[t] if global_liq[t]>0 else 0 for t in global_liq}
# 2) 拉 Swap 日志
topic = w3.keccak(text="Swap(address,address,int256,int256,uint160,uint128,int24)").hex()
logs  = w3.eth.get_logs({
    "fromBlock":START_BLOCK,"toBlock":END_BLOCK,
    "address":POOL_ADDR,"topics":[topic]
})
events = [pool_swap.events.Swap.process_log(lg) for lg in logs]
# 3) 累计手续费分配
fee0={t:0 for t in global_liq}
fee1={t:0 for t in global_liq}
for ev in events:
    a0, a1 = abs(ev["args"]["amount0"]), abs(ev["args"]["amount1"])
    pre = get_sqrtPriceX96(ev["blockNumber"]-1)/2**96
    post= ev["args"]["sqrtPriceX96"]/2**96
    t0, t1 = sorted((raw_sqrt_to_tick(pre), raw_sqrt_to_tick(post)))
    crossed=[t for t in range(t0,t1+1) if START_TICK<=t<=END_TICK]
    if not crossed: continue
    tot0, tot1 = a0*FEE_RATE, a1*FEE_RATE
    per0,per1 = tot0/len(crossed), tot1/len(crossed)
    for t in crossed:
        fee0[t]+= per0*weight[t]
        fee1[t]+= per1*weight[t]
```

```python
# 4) 计算
# 4.1 我们赚到的手续费
total_fee_usdc = sum(fee0.values())/1e6
total_fee_weth = sum(fee1.values())/1e18
# 4.2 头寸末值（Task1 中逻辑）
a0_raw, a1_raw = get_position_amounts(L_our, END_BLOCK, TICK_LOWER, TICK_UPPER)
amount0_usdc = a0_raw/1e6
amount1_weth = a1_raw/1e18
# 4.3 最终价值与 PnL
P1 = get_price_usdc_per_weth(END_BLOCK)
lp_value = amount0_usdc + amount1_weth*P1
fees_value = total_fee_usdc + total_fee_weth*P1
initial   = 100000.0
pnl       = lp_value + fees_value - initial
#
print(f"Estimated fees earned: {total_fee_usdc:.4f} USDC, {total_fee_weth:.6f} WETH")
print(f"Position at block {END_BLOCK}: {amount0_usdc:.6f} USDC, {amount1_weth:.6f} WETH")
print(f"LP value (ex-cl fees): {lp_value:.2f} USDC")
print(f"Fees value:            {fees_value:.2f} USDC")
print(f"-> Portfolio PnL:      {pnl:.2f} USDC")
```