Quantstamp

Monday Trade - Spot

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	DEX
Timeline	2025-09-01 through 2025-09-25
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Monday Trade Whitepaper
Diff/Fork information	Forked from: Uniswap V3.
Source Code	• SynFutures/MondayTrade-Spot #e474fa4
Auditors	• Hytham Farah Auditing Engineer • Hamed Mohammadi Auditing Engineer • Tim Sigl Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	13 Fixed: 8 Acknowledged: 4 Mitigated: 1
High severity findings	0
Medium severity findings	2 Fixed: 1 Acknowledged: 1
Low severity findings	6 Fixed: 3 Acknowledged: 2 Mitigated: 1
Undetermined severity findings	0
Informational findings	5 Fixed: 4 Acknowledged: 1

Summary of Findings

Monday Trade - Spot is a Uniswap V3–based protocol that enhances concentrated-liquidity AMMs by introducing on-chain limit order functionality, forming a hybrid orderbook–AMM system. It separates liquidity into two categories: traditional AMM liquidity (identical to Uniswap V3) and order liquidity (limit orders placed at specific price ticks).

Key audit considerations include:

1. A dual-liquidity architecture managed through delegated handler contracts — `LiquidityHandler`, `OrderHandler`, and `SwapHandler` — all orchestrated by the main `MondayPool` contract.
2. The Lazy Crossing mechanism - an order execution mechanism that leverages `orderTickSpacing` for gas efficiency, enabling delayed order settlements as ticks are crossed.
3. Complex state management around partially filled orders, including deferred settlement to minimize gas consumption.
4. Distinct fee structures applied separately to AMM and order liquidity.

Together, these elements create a sophisticated system balancing AMM liquidity depth with orderbook-style precision.

The most significant issue we found is related to logic errors in the limit order system, which may result in traders overpaying (OYSTR-1), as well as MEV opportunities that come at the expense of users who have placed orders earlier (OYSTR-2). No high-severity issues were uncovered during this audit.

Overall, the codebase is well-written but extremely complex. Additional documentation, especially more detailed NatSpec and in-line code comments (OYSTR-8), would greatly aid in understanding of the codebase and is highly recommended.

FIX-REVIEW UPDATE: Most issues have been properly addressed, with only minor adjustments remaining. Test-related concerns have been fully resolved, and both comprehensive test suites now pass successfully. Further improvements to documentation—particularly through expanded NatSpec comments—remain recommended.

ID	DESCRIPTION	SEVERITY	STATUS
OYSTR-1	Orders Are Filled Beyond the Taker's Price Limit Causing Unintended Overpayment	Medium	Fixed
OYSTR-2	MEV from Partially Filled Orders	Medium	Acknowledged
OYSTR-3	Unrestricted Withdrawal Function Allows Arbitrary Caller	Low	Acknowledged
OYSTR-4	Possibility of Arithmetic Overflow Leading to Dos	Low	Fixed
OYSTR-5	Inconsistent Nonce Increment when Canceling Order	Low	Fixed
OYSTR-6	Order Cancellation Griefing	Low	Acknowledged
OYSTR-7	Missing divisibility check between `tickSpacing` and `orderTickSpacing`	Low	Fixed
OYSTR-8	Insufficient In-Code and Technical Documentation	Low	Mitigated
OYSTR-9	`_maxNonceInRange()` scans every tick instead of stepping by `orderTickSpacing`	Informational	Fixed
OYSTR-10	No Upper Limit for Fee Values	Informational	Fixed
OYSTR-11	Minimum Order Amount May Be Set for Uninitialized Tokens	Informational	Acknowledged
OYSTR-12	Tick Spacing Exceeding 256 Ticks Creates Gaps in Order Placement	Informational	Fixed
OYSTR-13	Misleading Event Emission	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following

1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.

2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The scope focused on all the files that are unique to the Monday Trade protocol and not the contracts that are largely untouched from their Uniswap V3 fork (forked at commit `d8b1c635c275d2a9450bd6a78f3fa2484fef73eb`).

Files Included

```
contracts
├── BUProxy.sol
├── Config.sol
├── LiquidityHandler.sol
├── MondayFactory.sol
├── MondayPool.sol
├── MondayPoolDeployer.sol
├── OrderHandler.sol
├── SwapHandler.sol
├── libraries
│   ├── Tick.sol
│   ├── TickBitmap.sol
├── pearl
│   ├── Broker.sol
│   ├── LibMath.sol
│   ├── LibOrder.sol
├── periphery
    └── SwapRouter.sol
```

Files Excluded

```
contracts
├── MondayPoolStorage.sol
├── NoDelegateCall.sol
├── Ownable2Step.sol
├── interfaces
│   ├── callback
│   │   └── *
│   ├── pool
│   │   └── *
│   └── *
├── libraries
│   ├── BitMath.sol
│   ├── FixedPoint128.sol
│   ├── FixedPoint96.sol
│   ├── FullMath.sol
│   ├── LICENSE
│   ├── LICENSE_MIT
│   ├── LiquidityMath.sol
│   ├── LowGasSafeMath.sol
│   ├── Oracle.sol
│   ├── Position.sol
```

```
│   ├── SafeCast.sol
│   ├── SqrtPriceMath.sol
│   ├── SwapMath.sol
│   ├── TickMath.sol
│   ├── TransferHelper.sol
│   └── UnsafeMath.sol
├── periphery
│   ├── NonfungiblePositionManager.sol
│   ├── NonfungibleTokenPositionDescriptor.sol
│   ├── base
│   │   └── *
│   ├── examples
│   │   └── *
│   ├── interfaces
│   │   ├── external
│   │   │   └── *
│   │   └── *
│   ├── lens
│   │   └── *
│   ├── libraries
│   │   └── *
│   └── test
│       └── *
└── test
    └── *
```

Operational Considerations

- If the `makerFee` is a negative value, the protocol earns the leftover fee when the maker's share is deducted from the taker's fees. If the `-makerFee` is exactly equal to `takerFee`, for example `-(-0.3) = 0.3` in `OrderHandler._updateProtocolFees()`, then there is nothing left of the taker's fee because everything goes to the maker, and the protocol does not earn any fees.
- The `OrderHandler.place()` function only allows placing orders at ticks between -500,000 and 500,000, while Uniswap liquidity can be placed between tick -887,272 and tick 887,272. It's unlikely that the price for tokens would move outside of the 500,000 tick range, but if it does, orders could no longer be placed.
- Swap execution may be subject to more variable slippage than traditional AMMs. Specifically, if large limit orders at a tick are canceled before a user's transaction is processed, the user may experience a sudden shift from effectively zero slippage (with sufficient resting liquidity) to standard AMM slippage. This behavior is expected given the hybrid orderbook–AMM design and should be factored into user and integrator assumptions around execution guarantees.

Key Actors And Their Capabilities

Factory Owner

Responsibilities

The Factory Owner governs protocol-level settings for the Monday Trade - Spot Order Book through the `IUniswapV3Factory`. This role manages protocol fees, pool creation, and configuration.

Trust Assumptions

- Will not maliciously adjust fee parameters to extract excessive value from users.
- Will maintain pool configurations in a way that preserves order validation and fair execution.
- Will not use protocol fee authority to destabilize the market.

Exclusive Functions

1. **IUniswapV3Factory (referenced via `factory`):**
    1. Control pool creation and configuration.
    2. Set and update protocol fee parameters (`_updateProtocolFees()`).
    3. Modify fee collection mechanisms impacting protocol revenue.

Config Admin

Responsibilities

The Config Admin manages order-level parameters through the `Config` contract (accessed from `IUniswapV3Factory.factory`). This includes setting fee structures and trade requirements, directly affecting order placement and execution.

Trust Assumptions

- Will not set extreme fee parameters that drain liquidity or harm traders.
- Will set `minOrderAmount` values that prevent dust orders without enabling denial-of-service.
- Will not manipulate order maker fees or taker fees to unreasonably favor specific actors.

Exclusive Functions

1. **IConfig (via `factory.config()`):**
    1. `minOrderAmount()` : Configure minimum order amounts per token.
    2. `getOrderMakerFee()` : Set fees or rebates for order makers.
    3. `getOrderTakerFee()` : Set fees for order takers.
    4. Update additional order-specific parameters as needed.

Liquidity Provider

Responsibilities

The liquidity provider is a role inherited from the UniswapV3 protocol. These are the users who deposit and lock up their funds to be used as liquidity traded in accordance with the AMM price curve.

Order Maker

Responsibilities

The Order Maker places limit orders on specific ticks within the Monday Trade Spot order book. They provide liquidity by placing orders that wait to be filled by counterparties, earning potential maker fees or rebates based on the configured fee structure.

Order Taker

Responsibilities

The Order Taker executes against existing limit orders through the SwapHandler contract, consuming available order liquidity at specific ticks. They pay taker fees for immediate execution against maker orders and drive price discovery through their trading activity. Note that order takers cannot choose which limit order they want to take; rather they must always trade through the oyster swap algorithm, which either provides AMM or orderbook liquidity depending on what is available at the current tick.

Findings

OYSTR-1

Orders Are Filled Beyond the Taker's Price Limit Causing Unintended Overpayment

Medium | Fixed

> **Update**
>
> Marked as "Fixed" by the client.
> Addressed in: `228f5cab805840af30de94f5e7d0cd210b86a80e` .
> The client provided the following explanation:
>
> `add X96Limit check after _stepToNextTick in small loop`

File(s) affected: `SwapHandler.sol`

Description: In the inner loop of `_swapCrossRange()` (L382-492), orders at future order ticks can be filled even when `swap()` stops at `sqrtLimitX96` before the AMM price reaches those ticks. The current guard (`amountRemaining == 0` L410) only covers fully completed swaps where the AMM filled the whole swap amount, not limit-stopped swaps. This allows order to be filled beyond the actual reached price boundary, leading takers to pay more than their limit.

Exploit Scenario:
Prerequisite: AMM liquidity exists across the current swap interval, and orders are placed at order ticks further inside that interval.

1. A taker starts a swap with a price limit (`sqrtLimitX96`), moving price within the current swap interval.
2. The AMM advances toward the next order tick, but the swap hits the limit inside the interval (amount remaining > 0; price does not reach that order tick).
3. The swap function enters the inner matching loop and scans for the next order tick in the swap direction.
4. Because the price check is missing, it selects that future order tick even though the price of the order tick was not reached (`state.sqrtPriceX96 != TickMath.getSqrtRatioAtTick(orderTick)`).
5. It proceeds to fill orders at the unreached tick, increasing `partialFilledSize` and charging the taker.
6. **Outcome:** part of the swap executes at a worse price than the taker's limit (orders filled beyond the actually reached boundary).

Recommendation: Only fill orders at a tick when the AMM has actually reached it. Two possible ways to enforce that are the following:

1. Cap the inner-loop “finalTick” by the tick at the `sqrtLimitX96` (direction-aware), so you only consider order ticks up to the limit and not beyond the next swap boundary.
2. After stepping toward an order tick, only fill if the AMM actually reached that tick (`state.sqrtPriceX96 == sqrtRatioAtTick`)

OYSTR-2 MEV from Partially Filled Orders

Medium Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Since our partialFill and withdraw mechanisms need to preserve the current situation, the code will not change. Document has updated in this public link: https://ribbon-roast-c38.notion.site/Integration-Instruction-en-1f5c0a1d6a2e8046a263df92fdd49255
```

File(s) affected: `SwapHandler.sol` , `OrderHandler.sol`

Description: When an order at a tick is partially filled, the `orderNonce` does not increment. Any subsequent orders placed at that tick will therefore inherit the same **nonce**. This creates two classes of exploit scenarios:

1. Race to Withdraw Profits:

Once an order is partially filled, the filled amount can only be withdrawn once. Whoever calls `withdraw()` first will receive the tokens. This means that if the original order owner is not actively monitoring, another party can place a new order at that tick and immediately withdraw the already-filled portion. The original order owner may “never” be able to claim their rightful tokens.

2. Front-Running on Partial Fill:

A trader can observe that an order at a tick has been partially filled (via mempool monitoring), quickly place their own order at the same tick, and immediately call `withdraw()` . This allows them to claim the partially filled amount risk-free, and then proceed to trade at the updated market price.

This behavior creates a risk-free profit opportunity (MEV) for active traders at the expense of passive order owners. Original users are forced to wait until the price once again returns to their tick before they can potentially recoup value.

This attack vector is feasible in both private and public mempools. In public mempools, it is significantly worse, since attackers can easily attempt to position their withdrawal transactions directly after a swap that caused a partial fill.

Exploit Scenario:

- **Setup:** ETH/USDC pool, price = 3000. Alice places 1 ETH sell order at tick 75100.
- **Partial Fill:** A buyer takes 0.4 ETH. Alice earns 1200 USDC, but `orderNonce` (5) is unchanged. `partialFilledSize = 0.4` .
- **Exploit:**
  1. Bob sees the fill.
  2. Places 0.4 ETH order at same tick → inherits nonce 5.
  3. Calls `withdraw()` , passes checks, and withdraws 1200 USDC from Alice’s partial fill.
  4. `partialFilledSize` reduced to 0.
- **Impact:** Alice’s rightful 1200 USDC is already claimed. She must wait for price to return or cancel her order at a loss.

Recommendation: Either document the risks clearly to the user or ensure that anyone who places an order after a trade partially fills that tick will not be able to withdraw profits from that order specifically.

OYSTR-3
Unrestricted Withdrawal Function Allows Arbitrary Caller

Low Acknowledged

Update

Marked as acknowledged since the removal of the function in the router does not restrict the caller of the withdrawal function in any way.

Update

Marked as "Mitigated" by the client.
Addressed in: `a65f29807945c17bf82ce79bdb9a646377fa2524` .
The client provided the following explanation:

```
In order to allow for flexible withdrawal, the withdraw() function keeps not imposing any permission restrictions. The withdraw function in swapRouter() is actually useless and we have deleted it.
```

File(s) affected: `SwapRouter.sol`, `MondayPool.sol`

Description: The `withdraw()` function does not enforce that the caller is the `user`, allowing anyone to initiate withdrawals on behalf of another user. While this design may be intentional to support the `SwapRouter.withdraw()` flow, it represents a deviation from standard access control practices. The severity is limited since funds always reach their rightful owner and no further state changes are possible once an order becomes withdrawable.

However, the caller's ability to control the `needNativeToken` parameter is problematic, as it determines whether tokens are withdrawn as **ETH** or **WETH**. In scenarios where the receiver can only handle one token type, selecting the incorrect format would cause the withdrawn tokens to become stuck at the receiver address.

Recommendation: Add validation to ensure `msg.sender` is either the `SwapRouter` contract or matches the `user` parameter in `withdraw()`. Apply the same validation inside `SwapRouter.withdraw()`.

OYSTR-4 Possibility of Arithmetic Overflow Leading to Dos

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `08ea2ffc5ff8f662280e7824dd1f04918cf7e1f7`.

File(s) affected: `LibOrder.sol`

Description: The `_calAmountAtCurrentTick()` function, when calculating the amount of `token1` during a `zeroForOne` trade, computes `uint256(sqrtPriceX96) * uint256(size)` as input for the `mulDiv()` function. This calculation can overflow and cause DoS when placing or canceling orders with huge size at high ticks.

Exploit Scenario:
Below is a table outlining when an overflow would occur when the max tick is used:

```
| Tick    | sqrtPriceX96 | Min Overflow Amount | Ratio to uint128.max |
|---------|--------------|---------------------|----------------------|
| 500,000 | 5.70 × 10^36 | 20.32 × 10^36       | 5.97%                |
```

Recommendation: Use the same approach as in the other case within this function by first calculating half of the result, storing it in a temporary variable, and then finalizing the computation.

OYSTR-5 Inconsistent Nonce Increment when Canceling Order

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `3200b451d28c3f5f8289d807228bb623e3b42174`.

File(s) affected: `Broker.sol`

Description: `Broker._placeSingleTickOrder()` is used for both placing and canceling orders. During order cancelation, two paths exist:

- Normal cancel (`innerSize ≥ amount`): reduce `innerSize` by amount; nonce is not incremented; `partialFilledSize` is unchanged. If this takes `innerSize` to 0, the order tick nonce stays unchanged.
- Partial-filled cancelation (`innerSize < amount`): set amount to `innerSize` ; set `innerSize` to 0; increment the order-tick nonce (nonce0/nonce1); clear `partialFilledSize` .

The inconsistency is: Both paths can end with “no unfilled liquidity” (`innerSize = 0`), but:

- Normal cancelation keeps the same nonce and leaves `partialFilledSize`
- Partial-filled cancel bumps the nonce and clears `partialFilledSize`

This changes flow for users:

- After normal cancel-to-zero: users can still pass the `tickNonce <= userNonce` check and later use `cancel()` or `withdraw()`
- After partial-filled cancelation: stale `userNonce` fails the check (“Order filled”); users must use `withdraw()`

No critical problem arises from this inconsistency but it may impact the user experience.

Recommendation: When a normal cancel reduces `innerSize` to 0, also increment the corresponding `nonce0/nonce1` and set `partialFilledSize = 0` so both cancel paths close the generation consistently. This makes behavior uniform and keeps cancel/withdraw gating the same across cases.

OYSTR-6 Order Cancellation Griefing

Low | Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
We believe that adding a time lock cannot completely prevent this problem. In addition, just-in-time attacks are also common in AMM, so we decided that this situation does not require further restrictions.
```

File(s) affected: `OrderHandler.sol`, `SwapHandler.sol`

Description: Order cancellation is permissionless for the order owner. This allows the owner to cancel just before execution, forcing swappers to use AMM liquidity at less favorable prices.

Note this attack is only possible if the attacker can listen in on transactions before they land on chain.

Recommendation: Consider adding cancellation restrictions such as time locks, or document the risk clearly to users.

OYSTR-7
Missing divisibility check between `tickSpacing` and `orderTickSpacing`

Low | Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `ff5cfe416c01b25e8dd893c14a9d289ccde6c3d8`.

File(s) affected: `MondayFactory.sol`, `MondayPool.sol`

Description: The contracts validate that `tickSpacing >= orderTickSpacing` but do not enforce that `tickSpacing` is an exact multiple of `orderTickSpacing`. Without this divisibility check, configurations can be created where the two spacings are misaligned, leaving calculations that rely on predictable tick intervals inconsistent.

Recommendation: Add a check at pool creation to ensure divisibility:

```
require(_tickSpacing % orderTickSpacing == 0, "InvalidTickSpacing");
```

This guarantees that calculations based on tick alignment remain consistent.

OYSTR-8 Insufficient In-Code and Technical Documentation

Low | Mitigated

Update

Substantial whitepaper improvements (279 insertions, 194 deletions) but minimal NatSpec additions to contracts; code documentation remains sparse. The issue is marked "mitigated" because improvements to documentation are not substantial enough for a full fix, though we recognize many improvements were made in this regard.

Update

Marked as "Fixed" by the client.
Addressed in: `d53cd5b30116a33ef696fb53797d68e562e0bbaa`.

File(s) affected: `All Contracts`

Description: The protocol includes a well-written and detailed whitepaper, but there is a gap between the theoretical design and the implementation. Current documentation does not adequately:

- Use NatSpec and inline code comments to explain contract behavior.
- Map variables and functions in the codebase to their counterparts in the whitepaper.

- Provide developer-focused technical documentation on how the whitepaper design is realized in code. This makes it harder for auditors and developers to verify correctness or follow the implementation logic directly from the whitepaper.

Recommendation:

- Increase NatSpec coverage for all public and critical functions.
- Add inline code documentation clarifying variable purposes and design rationale.
- Provide a technical guide that explicitly maps whitepaper concepts to code (e.g., which variables and functions implement specific equations or mechanisms).
- Ensure consistent terminology between the whitepaper, code, and documentation.

OYSTR-9

`_maxNonceInRange()` scans every tick instead of stepping by `orderTickSpacing`

Informational | Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `dce0ffba19234ef51e26befc63f7e5fae2736d60` .

File(s) affected: `LibOrder.sol`

Description: `_maxNonceInRange()` linearly scans every tick between range bounds, even though only order ticks (separated by a constant `orderTickSpacing`) can hold meaningful order state. This produces unnecessary `SLOAD`s and makes gas cost scale with the interval size rather than the count of actual order ticks.

Recommendation: Iterate only over order ticks. Align the start to the first multiple in `[lowerTick, upperTick)` and step by `orderTickSpacing`:

Alternatively, use the existing order-bitmaps to jump to initialized order ticks within the range, or maintain per-range aggregated max nonces to avoid iteration entirely.

OYSTR-10 No Upper Limit for Fee Values

Informational | Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `ecbad8bb01600b9ba4d14e375ba5847d2c9b8aa4` and `5b8164abdf10aaf746741facc07a43113b973c46` .

File(s) affected: `config.sol`

Description: The `Config.resetOrderFee()` function allows setting maker and taker fees for the entire protocol, but does not enforce any upper limit on the values. It even allows to set fee values over 100% which may cause calculations to produce unexpected results.

Recommendation: Implement an upper limit on fee values. Since `newMakerFee` may be negative, the limit should be applied to its absolute value.

OYSTR-11
Minimum Order Amount May Be Set for Uninitialized Tokens

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
The setMinOrderAmount() function in config.sol is designed to be able to change the minOrderAmount of a token that has already been set.
```

File(s) affected: `config.sol`

Description: The `Config.setMinOrderAmount()` does not check if the provided `token` parameter is already initialized or not, and allows setting min amount for any token address.

Recommendation: Consider ensuring the token is already initialized by checking if the existing value of `minOrderAmount[token]` is not 0.

OYSTR-12
Tick Spacing Exceeding 256 Ticks Creates Gaps in Order Placement

Informational | Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `275d06cc6ac3f3295ae6062f92c435bef8b3f4d2` .

File(s) affected: `MondayFactory.sol` , `TickBitmap.sol`

Description: The inner bitmap implementation which tracks order ticks restricts tick spacing to a maximum of 256. The `TickBitmap.flipTickWithBitmap()` function calculates bit positions as the absolute difference between order ticks and swap ticks, with bounds validation that reverts when this difference reaches 256 or greater. Tick spacings exceeding 256 create gaps in order placement coverage in which placing orders becomes impossible.

Recommendation: Document the 256 tick spacing limitation and enforce it in the `MondayFactory.enableFeeAmount()` function by adding validation to reject tick spacings greater than 256.

OYSTR-13 Misleading Event Emission

Informational | Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `2aecbf7332ac47944fcd8e10ccd8962b0150af54` and `024d4087c18cb908ac9cb4ca1cc078735db02f9c` .

File(s) affected: `config.sol`

Description: The `Config.addTokens()` function emits the `TokensAdded()` event at the end. However, the value provided to the event is the `tokens` array, which contains all existing tokens in the system, rather than the newly added tokens in the `_tokens` array.

Additionally, the logic of `addTokens()` allows this function to be called only for updating the `minOrderAmount` for tokens, rather than adding new tokens, in which case emitting `TokensAdded()` is unnecessary. Ideally, this function would only be used for adding tokens, as the `setMinOrderAmount()` function already exists for adding existing tokens.

Similarly, the `Config.removeTokens()` function suffers from the same issue when emitting the `TokensRemoved()` event.

Recommendation: Ensure that the correct set of tokens (i.e., the `_tokens` array) is emitted in these events.

Auditor Suggestions

S1 'Dead' Code

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `b96e0f9f843b9e4f36aa9423379adb4564c4e418` .

File(s) affected: `TickBitmap.sol` , `LibMath.sol`

Related Issue(s): SWC-131, SWC-135

Description: "Dead" code refers to code which is never executed and hence makes no impact on the final result of running a program. Dead code raises a concern, since either the code is unnecessary or the necessary code's results were ignored.

- `TickBitmap.getSwapTick()` is not used anywhere and also seems wrong since it does not handle the edge cases outlined in `flipTickWithBitMap()` .
- `TickBitmap#L103-106` remove commented out code.
- `LibMath.sol` from the whole file only `LibMath.maxUint128()` is used.
- `LibMath.maxUint128()` is only used twice in the `Broker` contract and thus should be moved there.

Recommendation: We recommend to remove any unused code.

S2 Clarify Terminology

Fixed

Update

Please note that several instances of swap tick still exists in tests and latex format white paper.

Update

Marked as "Fixed" by the client.
Addressed in: `c12c07af7b2cdda15efe4308993f9d7a6a804a23`.

File(s) affected: `Broker.sol`

Description: The words swap tick and range tick in functions and variables are used as synonyms.

Recommendation: For clarity, choose one of the two terms and stick to it.

S3 Missing Input Validation

Fixed

Update

All fixed, except for 2, which validate the config but not the wETH.

Update

Marked as "Fixed" by the client.
Addressed in: `99244c67864f124bf7ea9071db6453932244a08f`.

File(s) affected: `MondayFactory.sol`, `MondayPool.sol`, `OrderHandler.sol`

Description: There are several key functions within the codebase that lack proper input parameter validations. A non-exhaustive list of such function parameters is provided below:

1. The constructor of the `MondayPool` contract does not validate any of its parameters. Since these are assigned to immutable values, it is especially important to validate these addresses against `address(0)`.
2. The constructor of the `MondayFactory` contract does not validate the `_config` and `_weth` parameters against `address(0)`.
3. The `MondayFactory.setOwner()` function does not validate the `_owner` address against `address(0)`.
4. The `MondayFactory.setNewOrderTickSpacing()` function does not validate the `newOrderTickSpacing` parameter against illegal values such as `0`.
5. Add `require(targetTick % int128(orderTickSpacing) == 0, 'Illegal Order Tick');` in the `OrderHandler._clearUserWithdraw()` function to ensure the call does not fail silently, and the function remains consistent with other functions in the contract.

Recommendation: Consider adding extra input validations throughout the code. Even for administrative functions, it is still recommended to include input validations to ensure proper security. The above list highlights the most important functions that lack such validations.

S4 General Code Improvements

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `109f4a6d3d46a5911163f19d6adea88beffbf465`.
The client provided the following explanation:

```
Expect 2. The config contract is upgradeable, and the pool address may be used in the future.
```

File(s) affected: `config.sol`, `MondayPool.sol`, `OrderHandler.sol`, `TickBitmap.sol`

Description:

1. `Fixed` — In the `Config.addTokens()` function, add a validation to ensure that the sizes of the two input arrays `_tokens` and `_minOrderAmounts` are equal.
2. `N/A` — Remove the `pool` address parameter from the two functions `getOrderTakerFee()` and `getOrderMakerFee()` in the `Config` contract.
3. `Fixed` — Consider adding `_disableInitializers()` in the constructor of the `MondayPool` contract.
4. `Fixed` — In the `OrderHandler.place()` function, the `isToken0` variable is introduced via `bool isToken0 = size > 0 ? true : false;`. However, the same logic (`size > 0 ? true : false`) is later repeated in the code instead of using the local variable `isToken0`.
5. `Fixed` — In `OrderHandler.place():L77~88`, the sequence of `if` and `else` branches is based on the `size` parameter. The last `else` branch reverts if `size` is `0`. To improve code readability and gas consumption, you can add an input validation to ensure that

`size` is not `0` at the beginning of the function instead.

6. `Fixed` — In `TickBitmap.flipTickWithBitmap()`, the `diff` local variable is calculated via the `abs()` function, therefore it cannot be negative, and the validation `if (diff < 0 || diff >= 256)` on line 54 is unnecessary. The same issue also exists in the `nextOrderTickWithinOneWord()` function of the same library.

Recommendation: Consider applying the above suggestions to improve the general state of the code.

S5 Failing and Incomplete Test Suite

Fixed

Update

As of most recent commit `a90133ce52ced24190d1a478819842f4de1bbd5b` the test suite results:

```
959 passing (28s)
20 pending
```

Update

Marked as "Fixed" by the client.
Addressed in: `ea27000a0beb4d459a81b2ab4e3145a68891ae30`.
The client provided the following explanation:

```
In fact, our main added process should be in forge test. However, due to the limitation of solc
compilation version, we encountered an unresolvable stack too deep error when running forge coverage.
All but one of the hardhat tests passed. I fixed this test in the commit. The snapshot test error will
be resolved after deleting the original snapshot file.
```

File(s) affected: `tests`

Description: The test suite spans all major files and contracts but is unreliable in its current state:

- **132 failing tests** out of 1219 total (~10.83% failure rate).
- **20 pending tests** not yet implemented.
- **No coverage report** generated, preventing visibility into untested areas.
- Multiple naming/parameter mismatches in error handling suggest the suite is not consistently run. This reduces confidence in the correctness of the implementation and increases the risk of undetected bugs.

Recommendation: - Resolve failing and pending tests.

- Generate and maintain coverage reports.
- Ensure consistent execution of the suite as part of the development process.
- Align error strings and parameter names between implementation and tests

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Test Suite Results

Test suite is comprehensive and spans all major files and contracts. There are two majour areas of concern:

1. There are a total of 132 failing tests (1067 passing, and 20 pending). This means approximately 10.83% of the test suite is failing. This should be immediately and urgently addressed by the team.

Note, though coverage data is lacking this reflects a limitation of the solc compiler at the moment rather than the issues with the test suite.

FIX-REVIEW UPDATE: New test results:

```
  959 passing (28s)
  20 pending
```

All tests are successfully passing, a substantial improvement. Furthermore, we have been made aware of an additional foundry-based test suite, whose output is now appended immediately below the hardhat tests.

```
  Creating Typechain artifacts in directory typechain for target ethers-v5
Successfully generated Typechain artifacts!


  BitMath
    #mostSignificantBit
      ✔ 0
      ✔ 1
      ✔ 2
      ✔ all powers of 2 (123ms)
      ✔ uint256(-1)
      ✔ gas cost of smaller number
      ✔ gas cost of max uint128
      ✔ gas cost of max uint256
    #leastSignificantBit
      ✔ 0
      ✔ 1
      ✔ 2
      ✔ all powers of 2 (94ms)
      ✔ uint256(-1)
      ✔ gas cost of smaller number
      ✔ gas cost of max uint128
      ✔ gas cost of max uint256

  FullMath
    - check a bunch of random inputs against JS implementation
    #mulDiv
      ✔ reverts if denominator is 0
      ✔ reverts if denominator is 0 and numerator overflows
      ✔ reverts if output overflows uint256
      ✔ reverts if output overflows uint256
      ✔ reverts on overflow with all max inputs
      ✔ all max inputs
      ✔ accurate without phantom overflow
      ✔ accurate with phantom overflow
      ✔ accurate with phantom overflow and repeating decimal
    #mulDivRoundingUp
      ✔ reverts if denominator is 0
      ✔ reverts if denominator is 0 and numerator overflows
      ✔ reverts if output overflows uint256
      ✔ reverts on overflow with all max inputs
      ✔ reverts if mulDiv overflows 256 bits after rounding up
      ✔ reverts if mulDiv overflows 256 bits after rounding up case 2
      ✔ all max inputs
      ✔ accurate without phantom overflow
      ✔ accurate with phantom overflow
      ✔ accurate with phantom overflow and repeating decimal

  LiquidityMath
```

```
  #addDelta
    ✔ 1 + 0
    ✔ 1 + -1
    ✔ 1 + 1
    ✔ 2**128-15 + 15 overflows
    ✔ 0 + -1 underflows
    ✔ 3 + -4 underflows
    ✔ gas add
    ✔ gas sub

NoDelegateCall
  ✔ runtime overhead
  ✔ proxy can call the method without the modifier
  ✔ proxy cannot call the method with the modifier
  ✔ can call the method that calls into a private method with the modifier
  ✔ proxy cannot call the method that calls a private method with the modifier

Oracle
  #initialize
    ✔ index is 0
    ✔ cardinality is 1
    ✔ cardinality next is 1
    ✔ sets first slot timestamp only
    ✔ gas
  #grow
    ✔ increases the cardinality next for the first call
    ✔ does not touch the first slot
    ✔ is no op if oracle is already gte that size
    ✔ adds data to all the slots
    ✔ grow after wrap
    ✔ gas for growing by 1 slot when index == cardinality - 1
    ✔ gas for growing by 10 slots when index == cardinality - 1
    ✔ gas for growing by 1 slot when index != cardinality - 1
    ✔ gas for growing by 10 slots when index != cardinality - 1
  #write
    ✔ single element array gets overwritten
    ✔ does nothing if time has not changed
    ✔ writes an index if time has changed
    ✔ grows cardinality when writing past
    ✔ wraps around
    ✔ accumulates liquidity
  #observe
    before initialization
      ✔ fails before initialize
      ✔ fails if an older observation does not exist
      ✔ does not fail across overflow boundary
      ✔ interpolates correctly at max liquidity
      ✔ interpolates correctly at min liquidity
      ✔ interpolates the same as 0 liquidity for 1 liquidity
      ✔ interpolates correctly across uint32 seconds boundaries
      ✔ single observation at current time
      ✔ single observation in past but not earlier than secondsAgo
      ✔ single observation in past at exactly seconds ago
      ✔ single observation in past counterfactual in past
      ✔ single observation in past counterfactual now
      ✔ two observations in chronological order 0 seconds ago exact
      ✔ two observations in chronological order 0 seconds ago counterfactual
      ✔ two observations in chronological order seconds ago is exactly on first observation
      ✔ two observations in chronological order seconds ago is between first and second
      ✔ two observations in reverse order 0 seconds ago exact
      ✔ two observations in reverse order 0 seconds ago counterfactual
      ✔ two observations in reverse order seconds ago is exactly on first observation
      ✔ two observations in reverse order seconds ago is between first and second
```

```
    ✔ can fetch multiple observations
    ✔ gas for observe since most recent
    ✔ gas for single observation at current time
    ✔ gas for single observation at current time counterfactually computed
  initialized with 5 observations with starting time of 5
    ✔ index, cardinality, cardinality next
    ✔ latest observation same time as latest
    ✔ latest observation 5 seconds after latest
    ✔ current observation 5 seconds after latest
    ✔ between latest observation and just before latest observation at same time as latest
    ✔ between latest observation and just before latest observation after the latest observation
    ✔ older than oldest reverts
    ✔ oldest observation
    ✔ oldest observation after some time
    ✔ fetch many values
    ✔ gas all of last 20 seconds
    ✔ gas latest equal
    ✔ gas latest transform
    ✔ gas oldest
    ✔ gas between oldest and oldest + 1
    ✔ gas middle
  initialized with 5 observations with starting time of 4294967291
    ✔ index, cardinality, cardinality next
    ✔ latest observation same time as latest
    ✔ latest observation 5 seconds after latest
    ✔ current observation 5 seconds after latest
    ✔ between latest observation and just before latest observation at same time as latest
    ✔ between latest observation and just before latest observation after the latest observation
    ✔ older than oldest reverts
    ✔ oldest observation
    ✔ oldest observation after some time
    ✔ fetch many values
    ✔ gas all of last 20 seconds
    ✔ gas latest equal
    ✔ gas latest transform
    ✔ gas oldest
    ✔ gas between oldest and oldest + 1
    ✔ gas middle
full oracle
  - has max cardinality next
  - has max cardinality
  - index wrapped around
    - can observe into the ordered portion with exact seconds ago
    - can observe into the ordered portion with unexact seconds ago
    - can observe at exactly the latest observation
    - can observe at exactly the latest observation after some time passes
    - can observe after the latest observation counterfactual
    - can observe into the unordered portion of array at exact seconds ago of observation
    - can observe into the unordered portion of array at seconds ago between observations
    - can observe the oldest observation 13*65534 seconds ago
    - can observe the oldest observation 13*65534 + 5 seconds ago if time has elapsed
    - gas cost of observe(0)
    - gas cost of observe(200 * 13)
    - gas cost of observe(200 * 13 + 5)
    - gas cost of observe(0) after 5 seconds
    - gas cost of observe(5) after 5 seconds
    - gas cost of observe(oldest)
    - gas cost of observe(oldest) after 5 seconds

SqrtPriceMath
  #getNextSqrtPriceFromInput
    ✔ fails if price is zero
    ✔ fails if liquidity is zero
```

```
    ✔ fails if input amount overflows the price
    ✔ any input amount cannot underflow the price
    ✔ returns input price if amount in is zero and zeroForOne = true
    ✔ returns input price if amount in is zero and zeroForOne = false
    ✔ returns the minimum price for max inputs
    ✔ input amount of 0.1 token1
    ✔ input amount of 0.1 token0
    ✔ amountIn > type(uint96).max and zeroForOne = true
    ✔ can return 1 with enough amountIn and zeroForOne = true
    ✔ zeroForOne = true gas
    ✔ zeroForOne = false gas
  #getNextSqrtPriceFromOutput
    ✔ fails if price is zero
    ✔ fails if liquidity is zero
    ✔ fails if output amount is exactly the virtual reserves of token0
    ✔ fails if output amount is greater than virtual reserves of token0
    ✔ fails if output amount is greater than virtual reserves of token1
    ✔ fails if output amount is exactly the virtual reserves of token1
    ✔ succeeds if output amount is just less than the virtual reserves of token1
    ✔ puzzling echidna test
    ✔ returns input price if amount in is zero and zeroForOne = true
    ✔ returns input price if amount in is zero and zeroForOne = false
    ✔ output amount of 0.1 token1
    ✔ output amount of 0.1 token1
    ✔ reverts if amountOut is impossible in zero for one direction
    ✔ reverts if amountOut is impossible in one for zero direction
    ✔ zeroForOne = true gas
    ✔ zeroForOne = false gas
  #getAmount0Delta
    ✔ returns 0 if liquidity is 0
    ✔ returns 0 if prices are equal
    ✔ returns 0.1 amount1 for price of 1 to 1.21
    ✔ works for prices that overflow
    ✔ gas cost for amount0 where roundUp = true
    ✔ gas cost for amount0 where roundUp = true
  #getAmount1Delta
    ✔ returns 0 if liquidity is 0
    ✔ returns 0 if prices are equal
    ✔ returns 0.1 amount1 for price of 1 to 1.21
    ✔ gas cost for amount0 where roundUp = true
    ✔ gas cost for amount0 where roundUp = false
  swap computation
    ✔ sqrtP * sqrtQ overflows

SwapMath
  #computeSwapStep
    ✔ exact amount in that gets capped at price target in one for zero
    ✔ exact amount out that gets capped at price target in one for zero
    ✔ exact amount in that is fully spent in one for zero
    ✔ exact amount out that is fully received in one for zero
    ✔ amount out is capped at the desired amount out
    ✔ target price of 1 uses partial input amount
    ✔ entire input amount taken as fee
    ✔ handles intermediate insufficient liquidity in zero for one exact output case
    ✔ handles intermediate insufficient liquidity in one for zero exact output case
    gas
      ✔ swap one for zero exact in capped
      ✔ swap zero for one exact in capped
      ✔ swap one for zero exact out capped
      ✔ swap zero for one exact out capped
      ✔ swap one for zero exact in partial
      ✔ swap zero for one exact in partial
      ✔ swap one for zero exact out partial
```

```
      ✔ swap zero for one exact out partial

Tick
  #tickSpacingToMaxLiquidityPerTick
    ✔ returns the correct value for low fee
    ✔ returns the correct value for medium fee
    ✔ returns the correct value for high fee
    ✔ returns the correct value for entire range
    ✔ returns the correct value for 2302
  #getFeeGrowthInside
    ✔ returns all for two uninitialized ticks if tick is inside
    ✔ returns 0 for two uninitialized ticks if tick is above
    ✔ returns 0 for two uninitialized ticks if tick is below
    ✔ subtracts upper tick if below
    ✔ subtracts lower tick if above
    ✔ subtracts upper and lower tick if inside
    ✔ works correctly with overflow on inside tick
  #update
    ✔ flips from zero to nonzero
    ✔ does not flip from nonzero to greater nonzero
    ✔ flips from nonzero to zero
    ✔ does not flip from nonzero to lesser nonzero
    ✔ does not flip from nonzero to lesser nonzero
    ✔ reverts if total liquidity gross is greater than max
    ✔ nets the liquidity based on upper flag
    ✔ reverts on overflow liquidity gross
    ✔ assumes all growth happens below ticks lte current tick
    ✔ does not set any growth fields if tick is already initialized
    ✔ does not set any growth fields for ticks gt current tick
  #clear
    ✔ deletes all the data in the tick
  #cross
    ✔ flips the growth variables
    ✔ two flips are no op

TickBitmap
  #isInitialized
    ✔ is false at first
    ✔ is flipped by #flipTick
    ✔ is flipped back by #flipTick
    ✔ is not changed by another flip to a different tick
    ✔ is not changed by another flip to a different tick on another word
  #flipTick
    ✔ flips only the specified tick
    ✔ reverts only itself
    ✔ gas cost of flipping first tick in word to initialized
    ✔ gas cost of flipping second tick in word to initialized
    ✔ gas cost of flipping a tick that results in deleting a word
  #nextInitializedTickWithinOneWord
    lte = false
      ✔ returns tick to right if at initialized tick
      ✔ returns tick to right if at initialized tick
      ✔ returns the tick directly to the right
      ✔ returns the tick directly to the right
      ✔ returns the next words initialized tick if on the right boundary
      ✔ returns the next words initialized tick if on the right boundary
      ✔ returns the next initialized tick from the next word
      ✔ does not exceed boundary
      ✔ skips entire word
      ✔ skips half word
      ✔ gas cost on boundary
      ✔ gas cost just below boundary
      ✔ gas cost for entire word
```

```
    lte = true
      ✔ returns same tick if initialized
      ✔ returns tick directly to the left of input tick if not initialized
      ✔ will not exceed the word boundary
      ✔ at the word boundary
      ✔ word boundary less 1 (next initialized tick in next word)
      ✔ word boundary
      ✔ entire empty word
      ✔ halfway through empty word
      ✔ boundary is initialized
      ✔ gas cost on boundary
      ✔ gas cost just below boundary
      ✔ gas cost for entire word

TickMath
  #getSqrtRatioAtTick
    ✔ throws for too low
    ✔ throws for too low
    ✔ min tick
    ✔ min tick +1
    ✔ max tick - 1
    ✔ min tick ratio is less than js implementation
    ✔ max tick ratio is greater than js implementation
    ✔ max tick
    tick -50
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick 50
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick -100
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick 100
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick -250
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick 250
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick -500
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick 500
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick -1000
      ✔ is at most off by 1/100th of a bips
      ✔ result
      ✔ gas
    tick 1000
      ✔ is at most off by 1/100th of a bips
      ✔ result
```

```
    ✔ gas
  tick -2500
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 2500
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -3000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 3000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -4000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 4000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -5000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 5000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -50000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 50000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -150000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 150000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -250000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 250000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick -500000
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 500000
    ✔ is at most off by 1/100th of a bips
```

```
    ✔ result
    ✔ gas
  tick -738203
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
  tick 738203
    ✔ is at most off by 1/100th of a bips
    ✔ result
    ✔ gas
#MIN_SQRT_RATIO
  ✔ equals #getSqrtRatioAtTick(MIN_TICK)
#MAX_SQRT_RATIO
  ✔ equals #getSqrtRatioAtTick(MAX_TICK)
#getTickAtSqrtRatio
  ✔ throws for too low
  ✔ throws for too high
  ✔ ratio of min tick
  ✔ ratio of min tick + 1
  ✔ ratio of max tick - 1
  ✔ ratio closest to max tick
  ratio 4295128739
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 79228162514264337593543950336000000
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 7922816251426433759354395033600000
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 7922816251426433759354395033600
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 990352031428304219919299379 2
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 280113854873930699593659691 13
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 560227709747861399187319382 27
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 792281625142643375935439503 36
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 112045541949572279837463876 454
    ✔ is at most off by 1
    ✔ ratio is between the tick and tick+1
    ✔ result
    ✔ gas
  ratio 224091083899144559674927752 909
    ✔ is at most off by 1
```

```
      ✔ ratio is between the tick and tick+1
      ✔ result
      ✔ gas
    ratio 6338253001141147007483516026 88
      ✔ is at most off by 1
      ✔ ratio is between the tick and tick+1
      ✔ result
      ✔ gas
    ratio 7922816251426433759354395 0
      ✔ is at most off by 1
      ✔ ratio is between the tick and tick+1
      ✔ result
      ✔ gas
    ratio 7922816251426433759354 3
      ✔ is at most off by 1
      ✔ ratio is between the tick and tick+1
      ✔ result
      ✔ gas
    ratio 1461446703485210103287273052203988822378723970341
      ✔ is at most off by 1
      ✔ ratio is between the tick and tick+1
      ✔ result
      ✔ gas

  UniswapV3Factory
    ✔ owner is deployer
    ✔ factory bytecode size
    ✔ pool bytecode size
    ✔ initial enabled fee amounts
    #createPool
      ✔ succeeds for low fee pool
      ✔ succeeds for medium fee pool
      ✔ succeeds for high fee pool
      ✔ succeeds if tokens are passed in reverse
      ✔ fails if token a == token b
      ✔ fails if token a is 0 or token b is 0
      ✔ fails if fee amount is not enabled
      ✔ gas
    #setOwner
      ✔ fails if caller is not owner
      ✔ updates owner
      ✔ emits event
      ✔ cannot be called by original owner
    #enableFeeAmount
      ✔ fails if caller is not owner
      ✔ fails if fee is too great
      ✔ fails if tick spacing is too small
      ✔ fails if tick spacing is too large
      ✔ fails if already initialized
      ✔ sets the fee amount in the mapping
      ✔ emits an event
      ✔ enables pool creation

    UniswapV3Pool arbitrage tests
      protocol fee = 0;
        passive liquidity of 0.010000
          exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
            ✔ not sandwiched
            ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(60ms)
            ✔ backrun to true price after swap only
          exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
            ✔ not sandwiched
```

```
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(65ms)
          ✔ backrun to true price after swap only
      passive liquidity of 1.0000
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(59ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(62ms)
          ✔ backrun to true price after swap only
      passive liquidity of 10.000
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(57ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(63ms)
          ✔ backrun to true price after swap only
      passive liquidity of 100.00
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(56ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(71ms)
          ✔ backrun to true price after swap only (48ms)
    protocol fee = 6;
      passive liquidity of 0.010000
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(60ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(115ms)
          ✔ backrun to true price after swap only
      passive liquidity of 1.0000
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(62ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 and true price of 1.01
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
(64ms)
          ✔ backrun to true price after swap only
      passive liquidity of 10.000
        exact input of 10e18 token0 with starting price of 1.0 and true price of 0.98
          ✔ not sandwiched
          ✔ sandwiched with swap to execution price then mint max liquidity/target/burn max liquidity
```

(204ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 **and** true price of 1.01
          ✔ **not** sandwiched
          ✔ sandwiched with swap to execution price **then** mint max liquidity/target/burn max liquidity
(85ms)
          ✔ backrun to true price after swap only
      passive liquidity of 100.00
        exact input of 10e18 token0 with starting price of 1.0 **and** true price of 0.98
          ✔ **not** sandwiched
          ✔ sandwiched with swap to execution price **then** mint max liquidity/target/burn max liquidity
(60ms)
          ✔ backrun to true price after swap only
        exact input of 10e18 token0 with starting price of 1.0 **and** true price of 1.01
          ✔ **not** sandwiched
          ✔ sandwiched with swap to execution price **then** mint max liquidity/target/burn max liquidity
(65ms)
          ✔ backrun to true price after swap only

  UniswapV3Pool gas tests
    fee is off
    #swapExact0For1
      ✔ first swap in block with **no** tick movement
      ✔ first swap in block moves tick, **no** initialized crossings
      ✔ second swap in block with **no** tick movement
      ✔ second swap in block moves tick, **no** initialized crossings
      ✔ first swap in block, large swap, **no** initialized crossings
      ✔ first swap in block, large swap crossing several initialized ticks
      ✔ first swap in block, large swap crossing a single initialized tick
      ✔ second swap in block, large swap crossing several initialized ticks (38ms)
      ✔ second swap in block, large swap crossing a single initialized tick
      ✔ large swap crossing several initialized ticks after some time passes
      ✔ large swap crossing several initialized ticks second time after some time passes (45ms)
    #mint
      around current price
        ✔ new position mint first in range
        ✔ add to position existing
        ✔ second position in same range
        ✔ add to position after some time passes
      below current price
        ✔ new position mint first in range
        ✔ add to position existing
        ✔ second position in same range
        ✔ add to position after some time passes
      above current price
        ✔ new position mint first in range
        ✔ add to position existing
        ✔ second position in same range
        ✔ add to position after some time passes
    #burn
      around current price
        ✔ burn when only position **using** ticks
        ✔ partial position burn
        ✔ entire position burn but other positions are **using** the ticks
        ✔ burn entire position after some time passes
      below current price
        ✔ burn when only position **using** ticks
        ✔ partial position burn
        ✔ entire position burn but other positions are **using** the ticks
        ✔ burn entire position after some time passes
      above current price
        ✔ burn when only position **using** ticks
        ✔ partial position burn

```
      ✔ entire position burn but other positions are using the ticks
      ✔ burn entire position after some time passes
  #poke
    ✔ best case
  #collect
    ✔ close to worst case
  #increaseObservationCardinalityNext
    ✔ grow by 1 slot
    ✔ no op
  #snapshotCumulativesInside
    ✔ tick inside
    ✔ tick above
    ✔ tick below
fee is on
  #swapExact0For1
    ✔ first swap in block with no tick movement
    ✔ first swap in block moves tick, no initialized crossings
    ✔ second swap in block with no tick movement
    ✔ second swap in block moves tick, no initialized crossings
    ✔ first swap in block, large swap, no initialized crossings
    ✔ first swap in block, large swap crossing several initialized ticks
    ✔ first swap in block, large swap crossing a single initialized tick
    ✔ second swap in block, large swap crossing several initialized ticks
    ✔ second swap in block, large swap crossing a single initialized tick
    ✔ large swap crossing several initialized ticks after some time passes (40ms)
    ✔ large swap crossing several initialized ticks second time after some time passes (44ms)
  #mint
    around current price
      ✔ new position mint first in range
      ✔ add to position existing
      ✔ second position in same range
      ✔ add to position after some time passes
    below current price
      ✔ new position mint first in range
      ✔ add to position existing
      ✔ second position in same range
      ✔ add to position after some time passes
    above current price
      ✔ new position mint first in range
      ✔ add to position existing
      ✔ second position in same range
      ✔ add to position after some time passes
  #burn
    around current price
      ✔ burn when only position using ticks
      ✔ partial position burn
      ✔ entire position burn but other positions are using the ticks
      ✔ burn entire position after some time passes
    below current price
      ✔ burn when only position using ticks
      ✔ partial position burn
      ✔ entire position burn but other positions are using the ticks
      ✔ burn entire position after some time passes
    above current price
      ✔ burn when only position using ticks
      ✔ partial position burn
      ✔ entire position burn but other positions are using the ticks
      ✔ burn entire position after some time passes
  #poke
    ✔ best case
  #collect
    ✔ close to worst case
  #increaseObservationCardinalityNext
```

```
        ✔ grow by 1 slot
        ✔ no op
      #snapshotCumulativesInside
        ✔ tick inside
        ✔ tick above
        ✔ tick below

  UniswapV3Pool
    ✔ constructor initializes immutables
    ✔ tick transition cannot run twice if zero for one swap ends at fractional price just below tick
(74ms)
    #initialize
      ✔ fails if already initialized
      ✔ fails if starting price is too low
      ✔ fails if starting price is too high
      ✔ can be initialized at MIN_SQRT_RATIO
      ✔ can be initialized at MAX_SQRT_RATIO - 1
      ✔ sets initial variables
      ✔ initializes the first observations slot
      ✔ emits a Initialized event with the input tick
    #increaseObservationCardinalityNext
      ✔ can only be called after initialize
      ✔ emits an event including both old and new
      ✔ does not emit an event for no op call
      ✔ does not change cardinality next if less than current
      ✔ increases cardinality and cardinality next first time
    #mint
      ✔ fails if not initialized
      after initialization
        ✔ protocol fees accumulate as expected during swap
        ✔ positions are protected before protocol fee is turned on
        ✔ poke is not allowed on uninitialized position (87ms)
        failure cases
          ✔ fails if tickLower greater than tickUpper
          ✔ fails if tickLower less than min tick
          ✔ fails if tickUpper greater than max tick
          ✔ fails if amount exceeds the max
          ✔ fails if total amount at tick exceeds the max (41ms)
          ✔ fails if amount is 0
        success cases
          ✔ initial balances
          ✔ initial tick
        above current price
          ✔ transfers token0 only
          ✔ max tick with max leverage
          ✔ works for max tick
          ✔ removing works
          ✔ adds liquidity to liquidityGross (54ms)
          ✔ removes liquidity from liquidityGross
          ✔ clears tick lower if last position is removed
          ✔ clears tick upper if last position is removed
          ✔ only clears the tick that is not used at all
          ✔ does not write an observation
        including current price
          ✔ price within range: transfers current price of both tokens
          ✔ initializes lower tick
          ✔ initializes upper tick
          ✔ works for min/max tick (186ms)
          ✔ removing works
          ✔ writes an observation
        below current price
          ✔ transfers token1 only
          ✔ min tick with max leverage
```

```
        ✔ works for min tick
        ✔ removing works
        ✔ does not write an observation
  #burn
    ✔ does not clear the position fee growth snapshot if no more liquidity (39ms)
    ✔ clears the tick if its the last position using it
    ✔ clears only the lower tick if upper is still used
    ✔ clears only the upper tick if lower is still used
  #observe
    ✔ current tick accumulator increases by tick over time
    ✔ current tick accumulator after single swap
    ✔ current tick accumulator after two swaps
  miscellaneous mint tests
    ✔ mint to the right of the current price
    ✔ mint to the left of the current price
    ✔ mint within the current price
    ✔ cannot remove more than the entire position
    ✔ collect fees within the current price after swap (43ms)
  post-initialize at medium fee
    k (implicit)
      ✔ returns 0 before initialization
      post initialized
        ✔ returns initial liquidity
        ✔ returns in supply in range
        ✔ excludes supply at tick above current tick
        ✔ excludes supply at tick below current tick
        ✔ updates correctly when exiting range
        ✔ updates correctly when entering range
  limit orders
    ✔ limit selling 0 for 1 at tick 0 thru 1
    ✔ limit selling 1 for 0 at tick 0 thru -1
    fee is on
      ✔ limit selling 0 for 1 at tick 0 thru 1
      ✔ limit selling 1 for 0 at tick 0 thru -1
  #collect
    ✔ works with multiple LPs
    works across large increases
      ✔ works just before the cap binds
      ✔ works just after the cap binds
      ✔ works well after the cap binds
    works across overflow boundaries
      ✔ token0
      ✔ token1
      ✔ token0 and token1
    #feeProtocol
      ✔ is initially set to 0
      ✔ can be changed by the owner
      ✔ cannot be changed out of bounds
      ✔ cannot be changed by addresses that are not owner
      ✔ position owner gets full fees when protocol fee is off
      ✔ swap fees accumulate as expected (0 for 1)
      ✔ swap fees accumulate as expected (1 for 0) (38ms)
      ✔ position owner gets partial fees when protocol fee is on
      ✔ fees collected by lp after two swaps should be double one swap
      ✔ fees collected after two swaps with fee turned on in middle are fees from last swap (not
confiscatory)
      ✔ fees collected by lp after two swaps with intermediate withdrawal
      #collectProtocol
        ✔ returns 0 if no fees
        ✔ can collect fees
        ✔ fees collected can differ between token0 and token1
    #tickSpacing
      tickSpacing = 12
```

```
    post initialize
      ✔ mint can only be called for multiples of 12
      ✔ mint can be called with multiples of 12
      ✔ swapping across gaps works in 1 for 0 direction (38ms)
      ✔ swapping across gaps works in 0 for 1 direction
  #flash
    ✔ fails if not initialized
    ✔ fails if no liquidity
    after liquidity added
      fee off
        ✔ emits an event
        ✔ transfers the amount0 to the recipient
        ✔ transfers the amount1 to the recipient
        ✔ can flash only token0
        ✔ can flash only token1
        ✔ can flash entire token balance
        ✔ no-op if both amounts are 0
        ✔ fails if flash amount is greater than token balance
        ✔ calls the flash callback on the sender with correct fee amounts
        ✔ increases the fee growth by the expected amount
        ✔ fails if original balance not returned in either token
        ✔ fails if underpays either token
        ✔ allows donating token0
        ✔ allows donating token1
        ✔ allows donating token0 and token1 together
      fee on
        ✔ emits an event
        ✔ increases the fee growth by the expected amount
        ✔ allows donating token0
        ✔ allows donating token1
        ✔ allows donating token0 and token1 together
  #increaseObservationCardinalityNext
    ✔ cannot be called before initialization
    after initialization
      ✔ oracle starting state after initialization
      ✔ increases observation cardinality next
      ✔ is no op if target is already exceeded
  #setFeeProtocol
    ✔ can only be called by factory owner
    ✔ fails if fee is lt 4 or gt 10
    ✔ succeeds for fee of 4
    ✔ succeeds for fee of 10
    ✔ sets protocol fee
    ✔ can change protocol fee
    ✔ can turn off protocol fee
    ✔ emits an event when turned on
    ✔ emits an event when turned off
    ✔ emits an event when changed
    ✔ emits an event when unchanged
  #lock
    ✔ cannot reenter from swap callback
  #snapshotCumulativesInside
    ✔ throws if ticks are in reverse order
    ✔ throws if ticks are the same
    ✔ throws if tick lower is too low
    ✔ throws if tick upper is too high
    ✔ throws if tick lower is not initialized
    ✔ throws if tick upper is not initialized
    ✔ is zero immediately after initialize
    ✔ increases by expected amount when time elapses in the range
    ✔ does not account for time increase above range
    ✔ does not account for time increase below range
    ✔ time increase below range is not counted
```

```
    ✔ time increase above range is not counted
    ✔ positions minted after time spent
    ✔ overlapping liquidity is aggregated
    ✔ relative behavior of snapshots
  fees overflow scenarios
    ✔ up to max uint 128
    ✔ overflow max uint 128
    ✔ overflow max uint 128 after poke burns fees owed to 0
    ✔ two positions at the same snapshot (41ms)
    ✔ two positions 1 wei of fees apart overflows exactly once (52ms)
  swap underpayment tests
    ✔ underpay zero for one and exact in
    ✔ pay in the wrong token zero for one and exact in
    ✔ overpay zero for one and exact in
    ✔ underpay zero for one and exact out
    ✔ pay in the wrong token zero for one and exact out
    ✔ overpay zero for one and exact out
    ✔ underpay one for zero and exact in
    ✔ pay in the wrong token one for zero and exact in
    ✔ overpay one for zero and exact in
    ✔ underpay one for zero and exact out
    ✔ pay in the wrong token one for zero and exact out
    ✔ overpay one for zero and exact out

  UniswapV3Pool swap tests
    low fee, 1:1 price, 2e18 max range liquidity
      ✔ swap exactly 1.0000 token0 for token1
      ✔ swap exactly 1.0000 token1 for token0
      ✔ swap token0 for exactly 1.0000 token1
      ✔ swap token1 for exactly 1.0000 token0
      ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
      ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
      ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
      ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
      ✔ swap exactly 0.000000000000010000 token0 for token1
      ✔ swap exactly 0.000000000000010000 token1 for token0
      ✔ swap token0 for exactly 0.000000000000010000 token1
      ✔ swap token1 for exactly 0.000000000000010000 token0 (38ms)
      ✔ swap token1 for token0 to price 2.5000
      ✔ swap token0 for token1 to price 0.40000
      ✔ swap token0 for token1 to price 2.5000
      ✔ swap token1 for token0 to price 0.40000
    medium fee, 1:1 price, 2e18 max range liquidity
      ✔ swap exactly 1.0000 token0 for token1
      ✔ swap exactly 1.0000 token1 for token0
      ✔ swap token0 for exactly 1.0000 token1
      ✔ swap token1 for exactly 1.0000 token0
      ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
      ✔ swap exactly 1.0000 token1 for token0 to price 2.0000 (72ms)
      ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
      ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
      ✔ swap exactly 0.000000000000010000 token0 for token1 (118ms)
      ✔ swap exactly 0.000000000000010000 token1 for token0
      ✔ swap token0 for exactly 0.000000000000010000 token1
      ✔ swap token1 for exactly 0.000000000000010000 token0
      ✔ swap token1 for token0 to price 2.5000
      ✔ swap token0 for token1 to price 0.40000
      ✔ swap token0 for token1 to price 2.5000
      ✔ swap token1 for token0 to price 0.40000
    high fee, 1:1 price, 2e18 max range liquidity
      ✔ swap exactly 1.0000 token0 for token1
      ✔ swap exactly 1.0000 token1 for token0
      ✔ swap token0 for exactly 1.0000 token1
```

```
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.000000000000010000 token0 for token1
    ✔ swap exactly 0.000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.000000000000010000 token1
    ✔ swap token1 for exactly 0.000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  medium fee, 10:1 price, 2e18 max range liquidity
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0 (60ms)
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.000000000000010000 token0 for token1
    ✔ swap exactly 0.000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.000000000000010000 token1
    ✔ swap token1 for exactly 0.000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  medium fee, 1:10 price, 2e18 max range liquidity
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.000000000000010000 token0 for token1
    ✔ swap exactly 0.000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.000000000000010000 token1
    ✔ swap token1 for exactly 0.000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  medium fee, 1:1 price, 0 liquidity, all liquidity around current price
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.000000000000010000 token0 for token1
    ✔ swap exactly 0.000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.000000000000010000 token1
    ✔ swap token1 for exactly 0.000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
```

```
    ✔ swap token1 for token0 to price 0.40000
  medium fee, 1:1 price, additional liquidity around current price
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  low fee, large liquidity around current price (stable swap)
    ✔ swap exactly 1.0000 token0 for token1 (155ms)
    ✔ swap exactly 1.0000 token1 for token0 (96ms)
    ✔ swap token0 for exactly 1.0000 token1 (73ms)
    ✔ swap token1 for exactly 1.0000 token0 (81ms)
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000 (143ms)
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  medium fee, token0 liquidity only
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1 (44ms)
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  medium fee, token1 liquidity only
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
```

```
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  close to max price
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000 (40ms)
    ✔ swap token0 for token1 to price 2.5000 (39ms)
    ✔ swap token1 for token0 to price 0.40000
  close to min price
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  max full range liquidity at 1:1 price with default fee
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  initialized at the max ratio
    ✔ swap exactly 1.0000 token0 for token1 (52ms)
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000 (40ms)
```

```
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
    ✔ swap exactly 0.0000000000000010000 token1 for token0
    ✔ swap token0 for exactly 0.0000000000000010000 token1
    ✔ swap token1 for exactly 0.0000000000000010000 token0
    ✔ swap token1 for token0 to price 2.5000
    ✔ swap token0 for token1 to price 0.40000
    ✔ swap token0 for token1 to price 2.5000
    ✔ swap token1 for token0 to price 0.40000
  initialized at the min ratio
    ✔ swap exactly 1.0000 token0 for token1
    ✔ swap exactly 1.0000 token1 for token0
    ✔ swap token0 for exactly 1.0000 token1
    ✔ swap token1 for exactly 1.0000 token0
    ✔ swap exactly 1.0000 token0 for token1 to price 0.50000
    ✔ swap exactly 1.0000 token1 for token0 to price 2.0000
    ✔ swap token0 for exactly 1.0000 token1 to price 0.50000
    ✔ swap token1 for exactly 1.0000 token0 to price 2.0000
    ✔ swap exactly 0.0000000000000010000 token0 for token1
      ✔ swap exactly 0.0000000000000010000 token1 for token0
      ✔ swap token0 for exactly 0.0000000000000010000 token1
      ✔ swap token1 for exactly 0.0000000000000010000 token0
      ✔ swap token1 for token0 to price 2.5000
      ✔ swap token0 for token1 to price 0.40000
      ✔ swap token0 for token1 to price 2.5000
      ✔ swap token1 for token0 to price 0.40000

  UniswapV3Pool
    ✔ constructor initializes immutables
    multi-swaps
      ✔ multi-swap


Snapshot Summary
 › 513 snapshots written from 12 test suites.

  959 passing (28s)
  20 pending



Ran 2 tests for forge-test/debug/OriginPool.t.sol:OriginalV3PoolDebugTest
[PASS] testWithoutAmountOut() (gas: 92137)
[PASS] testWithoutAmountOut_test2() (gas: 121581)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 7.35ms (372.17µs CPU time)

Ran 4 tests for forge-test/debug/StepToNextTickDirectTest.t.sol:StepToNextTickDirectTest
[PASS] test_StepToNextTick_EdgeCase_DifferentAmountThresholds() (gas: 166246)
[PASS] test_StepToNextTick_WhenAmountIsSufficient_PriceMovesToNextTick() (gas: 45623)
[PASS] test_StepToNextTick_WhenCurrentTickEqualsNextTick_StateRemainsUnchanged() (gas: 44101)
[PASS] test_StepToNextTick_ZeroForOneTrue_Behavior() (gas: 47634)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 7.76ms (2.07ms CPU time)

Ran 6 tests for forge-test/OrderPoolWithoutOrder.t.sol:OrderPoolSwapWithoutOrdersTest
[PASS] testInitialState() (gas: 27392)
[PASS] testSwapExact0For1_all_orders() (gas: 225938)
[PASS] testSwapExact0For1_all_orders_with_limit() (gas: 185126)
[PASS] testSwapExact0For1_ori_0001() (gas: 181290)
[PASS] testSwapExact0For1_ori_001() (gas: 220354)
```

```
[PASS] testSwapExact1For0_all_orders() (gas: 218431)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 9.24ms (2.96ms CPU time)

Ran 8 tests for forge-test/OriginalV3Pool.t.sol:OriginalV3PoolTest
[PASS] testInitialState() (gas: 18816)
[PASS] testSwapExact0For1_all_orders() (gas: 171596)
[PASS] testSwapExact0For1_all_orders_with_limit() (gas: 135931)
[PASS] testSwapExact0For1_ori_0001() (gas: 136636)
[PASS] testSwapExact0For1_ori_001() (gas: 167010)
[PASS] testSwapExact1For0_all_orders() (gas: 167268)
[PASS] testSwapExact1For0_partial_order() (gas: 113158)
[PASS] testV0CornerCase_TickTransition0_1() (gas: 6112265)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 9.18ms (3.03ms CPU time)

Ran 2 tests for forge-test/UniswapV3Factory.t.sol:MondayFactoryTest
[PASS] testSetNewConfig() (gas: 39997)
[PASS] testSetOrderTickSpacing() (gas: 850606)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 7.65ms (1.35ms CPU time)

Ran 3 tests for forge-test/debug/StepToNextTickTest.t.sol:StepToNextTickTest
[PASS] test_StepToNextTick_EdgeCase_MinimalAmountAtTickBoundary() (gas: 973237)
[PASS] test_StepToNextTick_WhenAmountIsSufficient_PriceMovesToNextTick() (gas: 477493)
[PASS] test_StepToNextTick_WhenCurrentTickEqualsNextTick_StateRemainsUnchanged() (gas: 475505)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 7.23ms (5.85ms CPU time)

Ran 13 tests for forge-test/orderPoolSwap.t.sol:OrderPoolSwapTest
[PASS] testInitialState() (gas: 27465)
[PASS] testSwapExact0For1() (gas: 241770)
[PASS] testSwapExact0For1_all_orders_int() (gas: 449741)
[PASS] testSwapExact0For1_all_orders_two_step_exactly() (gas: 337049)
[PASS] testSwapExact0For1_all_orders_two_step_with_limit() (gas: 494522)
[PASS] testSwapExact0For1_all_orders_with_limit() (gas: 444769)
[PASS] testSwapExact0ForOut1_all_orders() (gas: 447522)
[PASS] testSwapExact0ForOut1_partial_order() (gas: 268319)
[PASS] testSwapExact1For0_all_orders() (gas: 437733)
[PASS] testSwapExact1For0_partial_order() (gas: 256269)
[PASS] testSwapExact1ForOut0_all_orders() (gas: 420598)
[PASS] testSwapExact1ForOut0_partial_order() (gas: 245227)
[PASS] testSwapFeeEvent() (gas: 429278)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 10.44ms (8.06ms CPU time)

Ran 4 tests for forge-test/ExtremeTickSpacing.t.sol:ExtremeTickSpacingTest
[PASS] testCase1_PartialFill1To0() (gas: 2006852)
[PASS] testCase2_PartialFill0To1() (gas: 1004668)
[PASS] testCase3_ExactOut1To0() (gas: 1989043)
[PASS] testCase4_ExactOut0To1() (gas: 996406)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 13.76ms (11.55ms CPU time)

Ran 3 tests for forge-test/debug/SwapBug.t.sol:OrderFilledWhileNotAtPrice
[PASS] test_OrderFillBehavior_AtTick24080_WithSqrtPriceVariations() (gas: 1290187)
[PASS] test_OrderFillBehavior_AtTick24082_WithSqrtPriceVariations() (gas: 1558073)
[PASS] test_UnreachedFutureOrderTick_IsNotFillable_ButCurrentCodeFills() (gas: 621190)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 6.86ms (5.37ms CPU time)

Ran 1 test for forge-test/MonadOnline.t.sol:MonadOnlineTest
[FAIL: vm.envString: environment variable "MONAD_TESTNET_RPC" not found] setUp() (gas: 0)
Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 2.36ms (0.00ns CPU time)

Ran 11 tests for forge-test/periphery/SwapRouterPlace.t.sol:SwapRouterPlaceTest
[PASS] testBatchWithdrawToNativeToken() (gas: 1659291)
[PASS] testCancelPlaceWithNativeToken() (gas: 273446)
[PASS] testCancelWithNativeToken() (gas: 536571)
[PASS] testDefaultRecipient() (gas: 275669)
```

```
[PASS] testPlaceBothDirections() (gas: 496027)
[PASS] testPlaceMultipleOrders() (gas: 610492)
[PASS] testPlaceOrder() (gas: 294099)
[PASS] testPlaceWithNativeToken() (gas: 288334)
[PASS] testRevert_PlaceWithIncorrectNativeAmount() (gas: 37042)
[PASS] testWithdrawToNativeToken() (gas: 1170669)
[PASS] test_revert_invalidMinOrderAmount() (gas: 425337)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 24.00ms (17.17ms CPU time)

Ran 2 tests for forge-test/PartialFilledOrder.t.sol:PartialFilledOrderTest
[PASS] testMultiplePartialFills_CheckToken0() (gas: 1675445)
[PASS] testMultiplePartialFills_CheckToken1() (gas: 1700316)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.97ms (13.51ms CPU time)

Ran 4 tests for forge-test/periphery/QuoterV2.t.sol:QuoterV2Test
[PASS] testQuoteExactInputSingle() (gas: 195492)
[PASS] testQuoteExactOutputSingle() (gas: 382582)
[PASS] testQuoteWithInsufficientLiquidity() (gas: 1190085)
[PASS] testQuoteWithPriceLimit() (gas: 192197)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 9.06ms (7.42ms CPU time)

Ran 24 tests for forge-test/BranchCorner.t.sol:BranchCornerCase
[PASS] test10_stepToNext_tickSpacing1() (gas: 926625)
[PASS] test10_stepToNext_tickSpacing2() (gas: 1010197)
[PASS] test10_stepToNext_tickSpacing60() (gas: 957195)
[PASS] testFeeAmountCorrect_tickSpacing1() (gas: 926602)
[PASS] testFeeAmountCorrect_tickSpacing2() (gas: 794372)
[PASS] testFeeAmountCorrect_tickSpacing60() (gas: 850045)
[PASS] testFinalTickTotalSizeIsZero_tickSpacing1() (gas: 926603)
[PASS] testFinalTickTotalSizeIsZero_tickSpacing2() (gas: 794350)
[PASS] testFinalTickTotalSizeIsZero_tickSpacing60() (gas: 850001)
[PASS] testNextRangeTick_tickSpacing1() (gas: 671593)
[PASS] testNextRangeTick_tickSpacing2() (gas: 982421)
[PASS] testNextRangeTick_tickSpacing60() (gas: 632618)
[PASS] testNoLeftOrder_tickSpacing1() (gas: 926602)
[PASS] testNoLeftOrder_tickSpacing2() (gas: 760208)
[PASS] testNoLeftOrder_tickSpacing60() (gas: 849956)
[PASS] testNotCrossEmptyRangeTick_tickSpacing1() (gas: 869602)
[PASS] testNotCrossEmptyRangeTick_tickSpacing2() (gas: 818289)
[PASS] testNotCrossEmptyRangeTick_tickSpacing60() (gas: 980998)
[PASS] testStepNextEqFinalTick_tickSpacing1() (gas: 871503)
[PASS] testStepNextEqFinalTick_tickSpacing2() (gas: 789544)
[PASS] testStepNextEqFinalTick_tickSpacing60() (gas: 823317)
[PASS] testStepToNext_01_tickSpacing1() (gas: 1013242)
[PASS] testStepToNext_01_tickSpacing2() (gas: 1010933)
[PASS] testStepToNext_01_tickSpacing60() (gas: 825373)
Suite result: ok. 24 passed; 0 failed; 0 skipped; finished in 28.55ms (23.67ms CPU time)

Ran 8 tests for forge-test/CornerCase.t.sol:CornerCaseTest
[PASS] testRedundantAmountIn() (gas: 1484744)
[PASS] testSwapExact0For1_all_orders_two_step_exactly() (gas: 1607319)
[PASS] testSwapExact0For1_all_orders_two_step_exactly_liq2() (gas: 1490806)
[PASS] testSwapExactRangeTickInnerSize() (gas: 1460887)
[PASS] testTickStatus() (gas: 1254348)
[PASS] testTickTransition0_1_with_orders() (gas: 2641158)
[PASS] testTickTransition1_0_with_orders() (gas: 2288374)
[PASS] testTickTransitionCannotRunTwiceIfZeroForOneSwapEndsAtFractionalPriceJustBelowTick() (gas:
1903643)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 29.05ms (18.27ms CPU time)

Ran 17 tests for forge-test/OrderPoolWithdraw.t.sol:OrderPoolWithdrawTest
[PASS] testCancelOrderAfterPartialFill_Token0_Nonce0Increment() (gas: 546771)
[PASS] testCancelOrderAfterPartialFill_Token1_Nonce1Increment() (gas: 539404)
```

```
[PASS] testCancelWithDifferentSizes() (gas: 408446)
[PASS] testOrderTickSpacingValidation() (gas: 741136)
[PASS] testReadOrderStatus_FullyFilled() (gas: 442280)
[PASS] testReadOrderStatus_NoFill() (gas: 256172)
[PASS] testReadOrderStatus_PartiallyFilled() (gas: 687846)
[PASS] testWithdrawAfterSwap0For1_ExactIn_Out1Ether() (gas: 677250)
[PASS] testWithdrawAfterSwap0For1_ExactOut1Ether() (gas: 692807)
[PASS] testWithdrawAfterSwap0For1_ExactOut1_5Ether() (gas: 725092)
[PASS] testWithdrawAfterSwap0For1_ExactOut1_5Ether_withMakerFee() (gas: 722371)
[PASS] testWithdrawAfterSwap0For1_ExactOut_2Ether() (gas: 832304)
[PASS] testWithdrawAfterSwap1For0_ExactIn_Out1Ether() (gas: 616774)
[PASS] testWithdrawAfterSwap1For0_ExactIn_Out1_5Ether() (gas: 669808)
[PASS] testWithdrawAfterSwap1For0_ExactOut1Ether() (gas: 617080)
[PASS] testWithdrawAfterSwap1For0_ExactOut1_5Ether() (gas: 668592)
[PASS] testWithdrawAfterSwap1For0_ExactOut1_5Ether_withMakerFee() (gas: 676136)
Suite result: ok. 17 passed; 0 failed; 0 skipped; finished in 6.78ms (14.85ms CPU time)

Ran 1 test for forge-test/LimitTickSpacing_2.t.sol:LimitTickSpacingTest_2
[PASS] test_2TickSpacing_Case1_PartialFill1To0() (gas: 2204772)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.61ms (4.31ms CPU time)

Ran 9 tests for forge-test/OrderPoolPlace.t.sol:OrderPoolPlaceTest
[PASS] testBatchCancel() (gas: 495330)
[PASS] testBatchPlace() (gas: 564382)
[PASS] testBatchWithdraw() (gas: 966797)
[PASS] testCancelOrderEvent() (gas: 298053)
[PASS] testPartialFillAndCancel() (gas: 641781)
[PASS] testPlaceAndCancelMultipleOrders() (gas: 503937)
[PASS] testPlaceAndCancelOrder() (gas: 281461)
[PASS] testPlaceOrderWithInvalidAmount() (gas: 308226)
[PASS] testPlaceSwapPlace() (gas: 959074)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 9.46ms (5.94ms CPU time)

Ran 6 tests for forge-test/DenseOrderSwap.t.sol:DenseOrderSwapTest
[PASS] testLargeSwap() (gas: 391262)
[PASS] testMixSwap_10order() (gas: 1445186)
[PASS] testMixSwap_5order() (gas: 1409918)
[PASS] testSmallSwap() (gas: 1597617)
[PASS] testSmallSwapNegativeTicks() (gas: 4124084)
[PASS] testSwapThenPlaceThenSwap() (gas: 918895)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 31.50ms (25.09ms CPU time)

Ran 3 tests for forge-test/lib-test/FullMath.t.sol:FullMathTest
[PASS] testFuzzMulDivReversibility(uint256,uint24) (runs: 256, μ: 5243, ~: 5217)
[PASS] testMulDivEdgeCases() (gas: 66663)
[PASS] testMulDivReversibility() (gas: 9135)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 10.27ms (10.35ms CPU time)

Ran 3 tests for forge-test/lib-test/LibOrder.t.sol:LibOrderTest
[PASS] testCalAmountAtCurrentTickEdgeCases() (gas: 700609)
[PASS] testFuzzCalAmountAtCurrentTick(uint128,int24) (runs: 256, μ: 25814, ~: 19828)
[PASS] testOverflowProtectionDifference() (gas: 83240)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 46.99ms (43.36ms CPU time)

Ran 21 test suites in 229.33ms (298.09ms CPU time): 133 tests passed, 1 failed, 0 skipped (134 total
tests)

Failing tests:
Encountered 1 failing test in forge-test/MonadOnline.t.sol:MonadOnlineTest
[FAIL: vm.envString: environment variable "MONAD_TESTNET_RPC" not found] setUp() (gas: 0)

Encountered a total of 1 failing tests, 133 tests succeeded
```

Code Coverage

No coverage data can be generated due to stack too deep errors.

Changelog

- 2025-09-25 - Initial Report
- 2025-10-23 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.