Adrastia

SMART CONTRACT AUDIT

ZOKYO.

July 12th 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

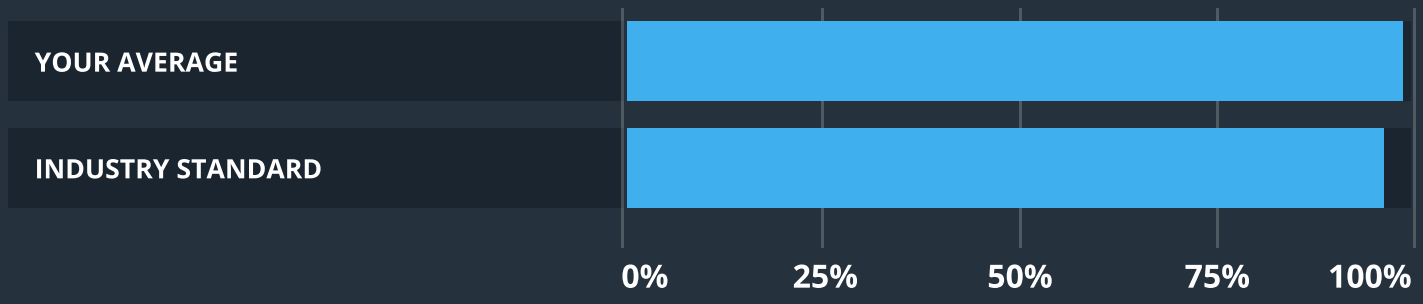
TECHNICAL SUMMARY

This document outlines the overall security of the Adrastia smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Adrastia smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Adrastia team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

Auditing Strategy and Techniques Applied 3
Executive Summary 5
Protocol Overview 6
Structure and Organization of Document 16
Complete Analysis 17
Code Coverage and Test Results for all files (by Zokyo Security) 35
Code Coverage and Test Results for all files (by Adrastia) 37

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Adrastia repository.
https://github.com/adrastia-oracle/adrastia-core
Initial commit: 8de065e7bb373a4925706890466e45bbaf2a4fa6, development branch
Last audited commit: 5317f85a3dc4c3bea8394610f1780ffde0e62be1

https://github.com/adrastia-oracle/adrastia-periphery
Initial commit: 27d6577c21be10b5c702aeb073c58754dac5f1c8, development branch
Last audited commit: b8ae70c88d476e9cb1ca614a07ff2d3add1456cc

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- CurveLiquidityAccumulator.sol
- CurvePriceAccumulator.sol
- UniswapV2LiquidityAccumulator.sol
- UniswapV2PriceAccumulator.sol
- UniswapV3LiquidityAccumulator.sol
- UniswapV3PriceAccumulator.sol
- AbstractAccumulator.sol
- LiquidityAccumulator.sol
- PriceAccumulator.sol
- AccumulationLibrary.sol
- ObservationLibrary.sol
- AbstractOracle.sol
- AggregatedOracle.sol
- PeriodicAccumulationOracle.sol
- PeriodicOracle.sol
- ManagedCurveLiquidityAccumulator.sol
- ManagedCurvePriceAccumulator.sol
- ManagedUniswapV2PriceAccumulator.sol
- ManagedUniswapV3LiquidityAccumulator.sol
- ManagedUniswapV3PriceAccumulator.sol
- ManagedPeriodicAccumulationOracle.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Adrastia smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.
2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.
4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

Adrastia protocol implements set of TWAP oracles for liquidity and price with aggregation of observations from Curve and Uniswap protocols. Aggregation also includes several protection mechanisms together with control of update time to be within the specified period.

Auditors team has carefully checked all the contracts and native tests. The protocol has impressive native unit-tests coverage with coverage of several ranges of token liquidity including several edge cases with extreme values of price. Though, auditors team has provided additional testing of edge-cases together with gas-usage checks .

There were no critical issues found, but auditors team had several questions according to the unclear logic of corrupted oracles handling, according to the observations update and correct validation of observations. Though, after the consultation with Adrastia team, all issues were resolved or verified. All comments from both Zokyo and Adrastia sides are reflected in the report.

The code has high quality but it needs to be mentioned, that the protocol lacks documentation and especially the comprehensive schemes of the business logic. Nevertheless, Zokyo Security team has performed a deep dive into the logic of contracts and attached the resulting schemes to the report.

Although, the oracles system works together with observations from Curve and Uniswap, the correctness of oracles setup is up to the deployer of the system. Adrastia oracles work as the aggregator of oracles, but there is still a certain level of uncertainty in case of other observation sources usage or in case of usage of reasonably large number of sources.

PROTOCOL OVERVIEW

Adrastia - is price and liquidity aggregator. Contracts are separated into accumulators and oracles. Both accumulators and oracles have a quote token. Contracts store prices of other tokens against quote token and liquidities of tokens and quote tokens (In frames on common LP Pair on Uniswap V2 or Pool in Curve/Uniswap V3).

Accumulators are divided into price accumulators and liquidity accumulators.

Price accumulators retrieve and store the price of other tokens against quote tokens from Uniswap V2, Uniswap V3 and Curve. Price accumulator stores latest price of tokens and cumulative price(In order to retrieve the price at specified time period).

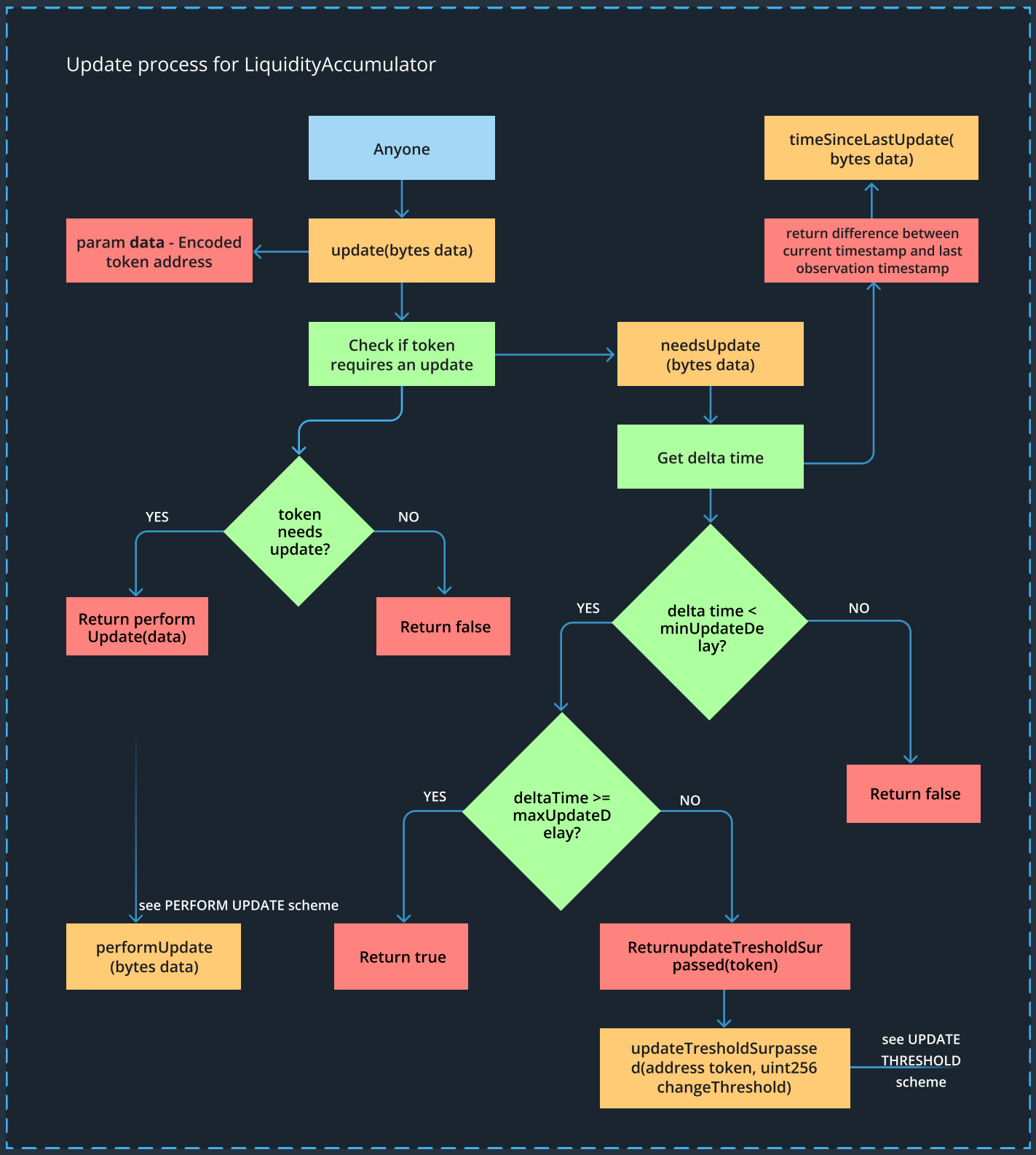
Liquidity accumulators retrieve and store liquidities of tokens and quote tokens from Uniswap V2, Uniswap V3 and Curve. Liquidities are retrieved from Uniswap V2 common LP pairs or Curve/ Uniswap V3 common pools. Liquidity accumulator stores latest liquidities for tokens and quote token and cumulative liquidities(In order to retrieve liquidities at specified time period).

Oracles are divided into PeriodicAccumulationOracle and AggregatedOracle.

PeriodicAccumulation oracles retrieve price and liquidities from accumulators. One oracle has a pair of price/liquidity accumulator (For Uniswap V2/Uniswap V3/ Curve). Oracle stores price/ cumulative price/liquidities for tokens and quote tokens/cumulative liquidities.

AggregatedOracle collects liquidities and prices from PeriodicAccumulation oracles. Such oracle stores average weighte price and common liquidities from all oracles.

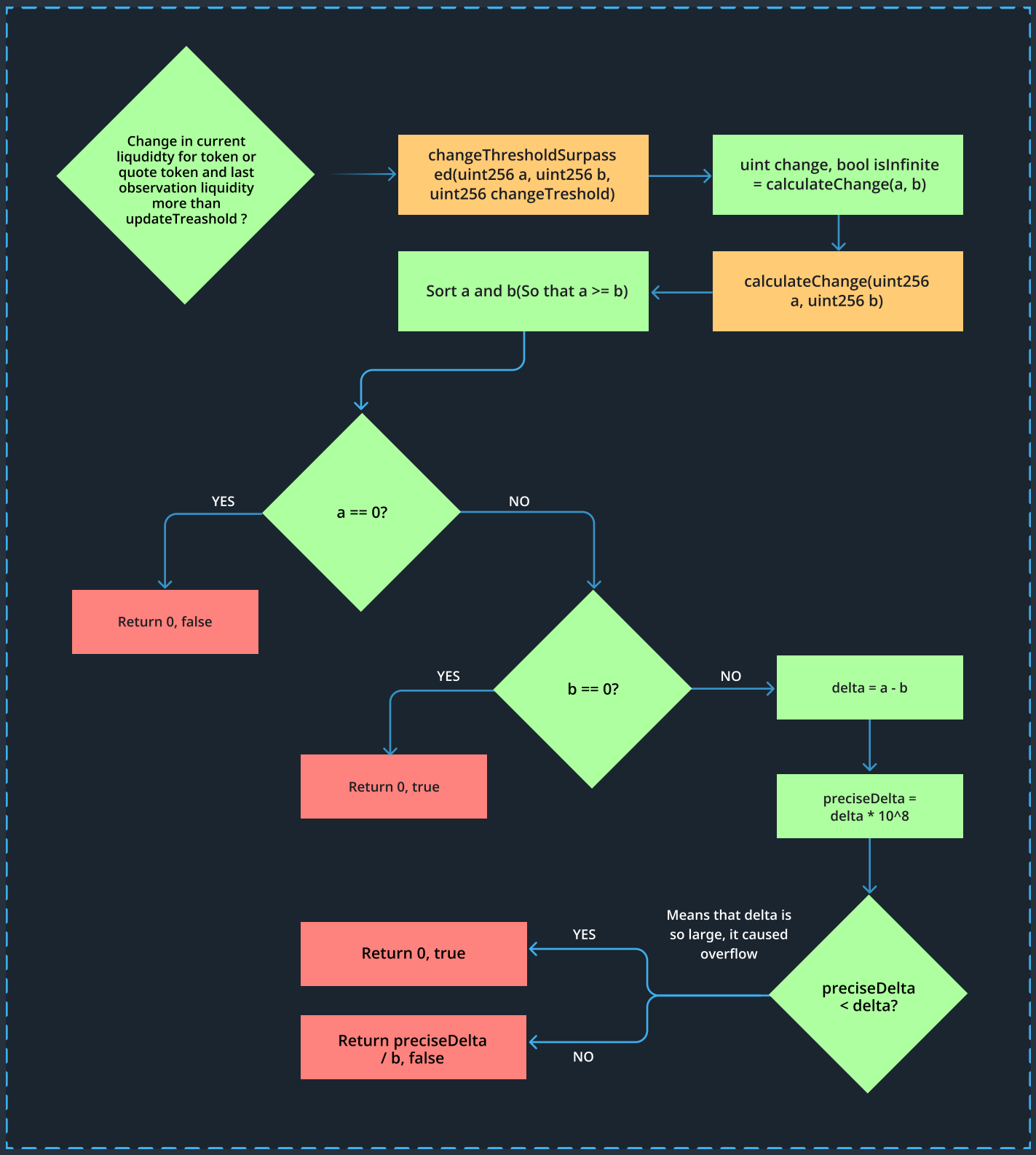
LIQUDITYACCUMULATOR.SOL



LIQUDITYACCUMULATOR.SOL (PERFORM UPDATE)

LIQUDITYACCUMULATOR.SOL (UPDATE THRESHOLD)

ABSTRACTACCUMULATOR.SOL

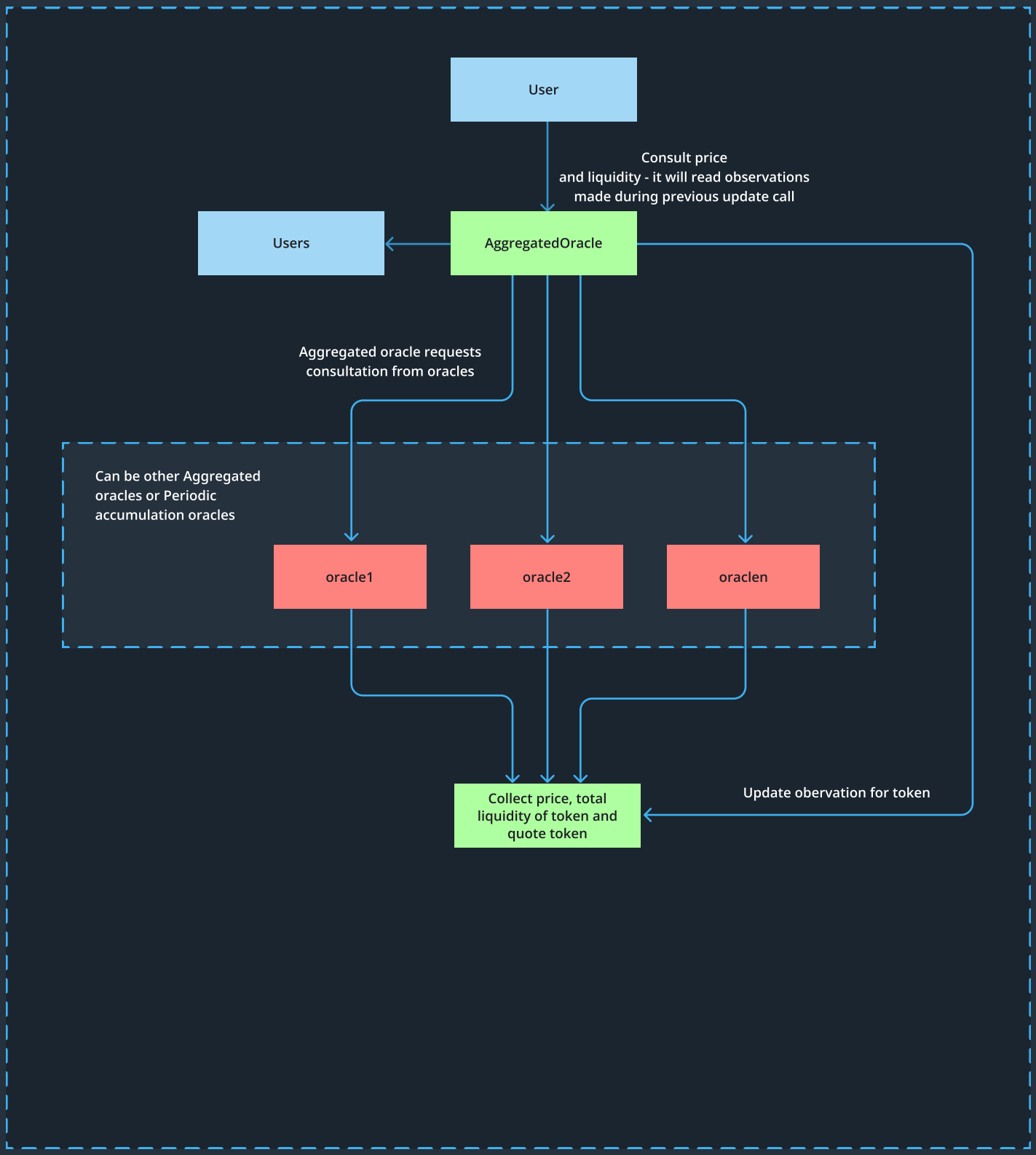


PRICEACCUMULATOR.SOL

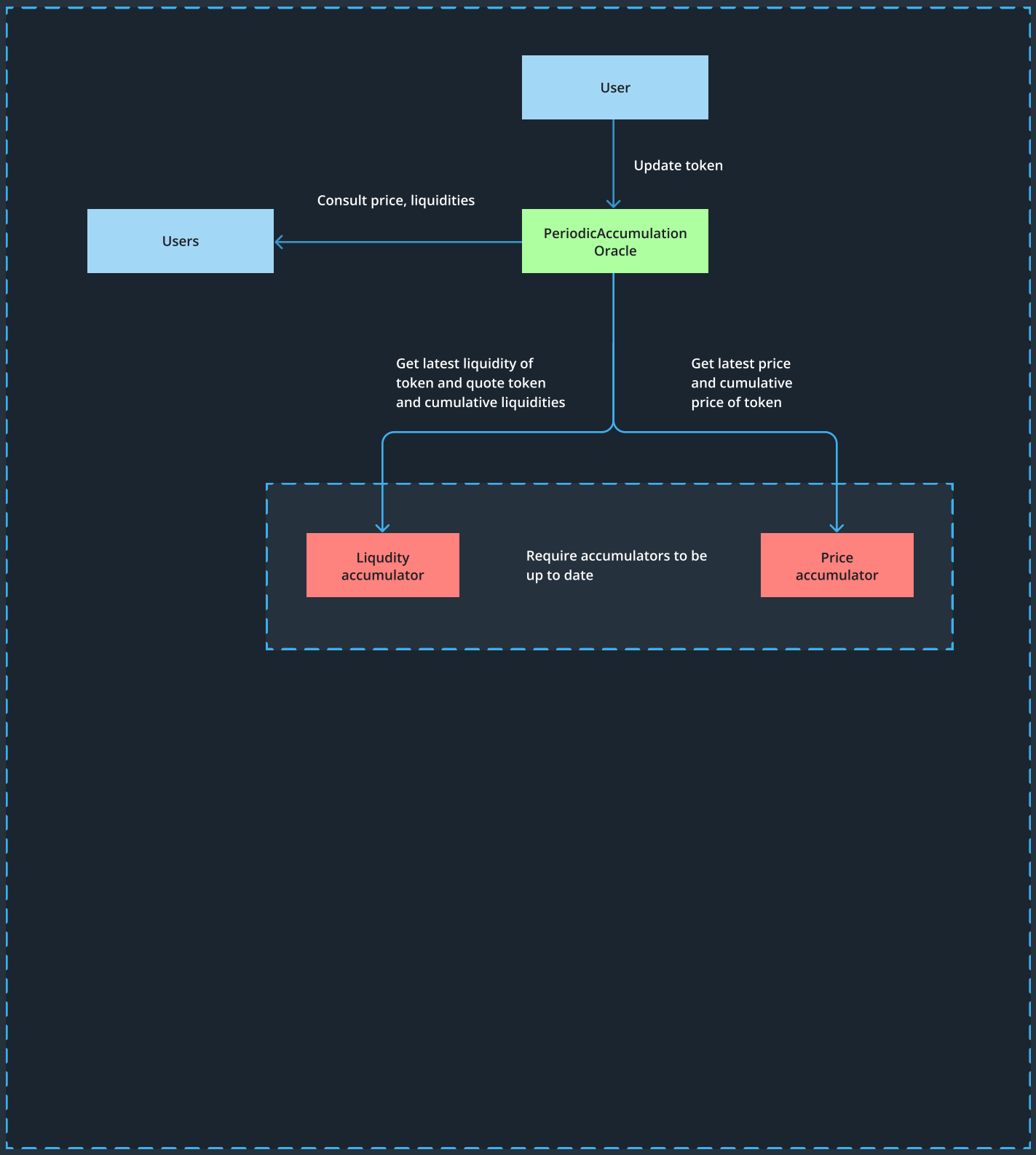
PRICEACCUMULATOR.SOL (FEED UPDATE)

PRICEACCUMULATOR.SOL (FETCH PRICE)

AGGREGATEDORACLE SCHEME



PERIODICACCUMULATIONORACLE SCHEME



STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

HIGH | VERIFIED

Missed validation of potentially corrupted oracle.

AggregatorOracle.sol: aggregateUnderlying(), line 413 (Note comment)
The note says about the issue which may occur in case of insanely high price. In such a case the division result of oQuoteTokenLiquidity / oPrice will equal 0. Thus, the oracle response will be marked as valid, but the resulting price will have lower denominator and will be corrupted. Such an issue can not occur in normal obstacles, but it can occur in case of corrupted oracle which returns harmful data. Since the contract itself has not cross-validation between the oracles, and the only sanity checks performed in validateUnderlyingConsultation() are based on the data from the Oracle itself, such an issue may occur. And in such case data can be manipulated from the outer source (which have no verification procedure) to pass the sanityCheckTvlDistributionRatio() check and still have the mentioned impact.
Issue is marked as high, due the high impact on the logic and actually high risk of happening.

Although the question of cross-validation of oracles is out of the scope of the audit due the nature of oracles, the potential data corruption by some of the oracles is a serious issue since it directly influences the logic of current contracts set.

Recommendation:
Verify the correct validation of the number of required valid answers from the aggregated oracles.

Post-audit:
after the consultation with the Adrastia team it was verified, that the logic of division is correct, since oQuoteTokenLiquidity is shifted to the left by 120 bits before it's used. Therefore, as long as oQuoteTokenLiquidity is not equal to 0, it will be larger than oPrice since prices are stored in 112-bit numbers. So, it is impossible for the division result of oQuoteTokenLiquidity / oPrice to equal 0.

MEDIUM | RESOLVED

Curve pool interface should be verified.

CurvePriceAccumulator.sol
Currently, the curve price accumulator is using the interface of the curve pool with the function get_dy() with the following parameters (int128 i, int128 j, uint256 dx). However there is another version of this function in the Curve protocol with parameters (uint256 i, uint256j, uint256 dx). For example, such an interface is utilized by TriCryptoPool. Due to interface mismatch, some curve pools can't be supported.

Recommendation:
Add a support of another get_dy() interface or verify the correctness of the implementation.

LOW | RESOLVED

Parameters lack validation.

LiquidityAccumulator.sol, PriceAccumulator.sol: cunstructor.
Parameters minUpdateDelay_ and maxUpdateDelay_ should be validated, so that maxUpdateDelay_ is greater than minUpdateDelay_.

Recommendation:
Validate parameters in constructor.

INFORMATIONAL | UNRESOLVED

Repeatable calculations.

LiquidityAccumulator.sol, PriceAccumulator.sol: update().
Function calls fetchLiqudity() or fetchPrice() two times, increasing gas spendings for function update(). First time it is called during checking that the token needs an update (In function changeThresholdSurpassed(address token, uint256 changeThreshold)). Then it is called in the function performUpdate().

Recommendation:
Consider returning the values of liquidities and price after first calculation and reuse it in the function performUpdate().

INFORMATIONAL | RESOLVED

Unhandled period value.

AggregatorOracle.sol: calculateMaxAge().
Since period for the PeriodicOracle can be set to 0, this function has unhandled case which can influence the contracts correct usage by the dApp.

Recommendation:
Add handling of zero period case.

From client:
The setting of a period of 0 seconds was disallowed, since calculating data over a period of 0 seconds doesn't make sense. Also the issue is fixed where AggregatedOracle#calculateMaxAge returns 0 because as outlined in the interface documentation, a max-age of 0 gets the instant rates as of the latest block, which is not what's expected of the aggregated oracle.

Minimal number of responses.

AggregatorOracle.sol: canUpdate(), performUpdate().
Both functions for the update check and update performance use the same minimal number of valid oracles updates: validResponses >= 1, requesting minimum 1 correct response.
Since the oracle aggregates data from several sources and the aggregation itself is a powerful tool against the incorrect data consuming, it is important to carefully evaluate the results gathered from those sources. Thus, more sources have delivered correct data - more trust the aggregator has. Thus, the validation upon at least 1 source seems not enough for the complex oracles system.
Issue is marked as info, since it requires business logic verification from the team, thus may not be classified as security issues.

Recommendation:
Verify the correct validation of the number of required valid answers from the aggregated oracles.

From client:
Added support for the minimum number of underlying oracle consultations to be changed from the default. By the Customer's experience with using oracles built around Uniswap and Curve, as long as one DEX has enough liquidity to make performing arbitrage profitable (assuming the presence of arbitrageurs), only 1 response is needed. The sanity checks for minimum quote token liquidity and minimum token liquidity value of each underlying oracle can be used to ensure this. With that being said, the more liquidity backing up the price, the more reliable and accurate the price is and the more costly it is to attack. Oracle consumers can use the consult function to retrieve price and liquidities, then they can alternate to a fallback oracle if there's not enough liquidity backing up the price.

INFORMATIONAL | VERIFIED

Potential violated oracle handling.

AggregatorOracle.sol: canUpdate(), performUpdate().
There is pretty high risk of the violation of one or several of aggregated Oracles (due to the hack of that oracle, or mistake in calculations, or pool draining, or corrupted data writing), thus there can be not full confidence in all of aggregated sources. Such oracles nature puts a question of the reliability of the AggregatorOracle in case if one or several of aggregated sources are violated. And for such case the protocol has no procedure of the oracle "de-violation": no pause, no corrupted oracle deletion, no reverts, no zero-data return instead of price (e.g. return of 0s in case of oracles violated). Thus, even if the oracle will be immediately re-deployed, it still leaves the place for corrupted data usage that may lead to exploits of "oracle manipulation" type. So, the protocol needs the procedure of (first of all) violated sources detection, and prevention of their influence on the final price.
Issue is marked as info, since it requires business logic verification from the team, thus may not be classified as security issues. Though in this case the issue may lead to critical results in case of ignoring violated sources and using the average price over corrupted data.

- negative threshold is equal to 100%;
- positive threshold is equal to 10000%.

Recommendation:
Verify the correctness of the sources, or bring the procedure of the violation detection and removal/deactivation OR bring the procedure of the whole oracle deactivation.

From client:
The functionality is confirmed to correspond the design. It is the responsibility of the oracle consumer to validate that the aggregated oracle and all of the underlying oracles are set up correctly and in alignment with their requirements and assumptions. The system has been designed to be immutable so as to guarantee that corrupted or corruptible underlying oracles are not maliciously introduced while the aggregated oracle is in use. Also, it is confirmed by the Adrastia team, that there are plans to establish a DAO later down the road with the ability to swap the underlying oracles and/or pause the oracles, but only when the protocol is sufficiently decentralized.

From client:
Requiring minimum token liquidity value and also minimum quote token liquidity handles cases where the underlying pools are drained. In such cases where this validation fails, the underlying oracle will be ignored.
There are some other attack vectors involving Uniswap V3 where there's one-sided liquidity or where there are large gaps between ticks causing the price to not get updated. If the aforementioned validation doesn't catch this, the TVL distribution ratio validation typically will, but not always. Therefore, extra caution must be taken when including or using the Uniswap V3 oracle.
If all underlying oracles fail such validations, or if the underlying oracles aren't up-to-date, the aggregated oracle will not update. It's recommended for oracle consumers to use consult(token, maxAge) where the maxAge is the required freshness of the data, in seconds. In the case where the oracle hasn't been updated in the last maxAge seconds, the function will revert. Consumers can then catch the exception and fall back on an alternative oracle.

INFORMATIONAL | VERIFIED

Observation update interpretation.

PeriodicAccumulationOracle.sol: performUpdate().
updatedObservation variable can be set to true only with the price updated, but without liquidity updated. The logic allows to have only price updated within the observation (line 101 - setting of the variable after the successful price update) but still emit event about the update (line 155 - the condition accepts set updatedObservation value and default missingPrice setting to false).

Verify, that there is no need for a liquidity update for such logic (since in case if liquidity is not updated from the LiquidityAccumulator - there is no influence on the event emitting).

Issue is marked as info, since it requires business logic verification from the team, thus may not be classified as security issues.

Recommendation:
Verify the correctness of the observation update interpretation.

From client:
It's okay for liquidities to be reported as 0 in the case where the oracle doesn't have enough information to calculate liquidity. Liquidities are used in the aggregated oracle, and in the case that the oracle reports 0 liquidity, it'll be ignored by the aggregator (note: this is confirmed by the auditors team).
The price is the important component, so in the case where the oracle doesn't have enough information to calculate the price, it doesn't update the timestamp. This will cause consultation to revert, with the message 'MISSING_OBSERVATION'. This will also make it so that no 'Updated' events are emitted. All-in-all, the end result in this case will seemingly be that no observation is made.

INFORMATIONAL | RESOLVED

Assumption for updated accumulators.

PeriodicAccumulationOracle.sol: performUpdate(), line 84.
The note states, that for the gas optimizations the logic assumes, that accumulators are up-to-date. Auditors need verification from the Adrastia team according the reasons behind the assumption.
Issue is marked as info, since it requires business logic verification from the team, thus may not be classified as security issues.
Issue will not be displayed in the report, but will be described in the executive summary and the protocol overview sections, since it relates to the core interaction between the contracts.

Post-audit:
Adrastia team has confirmed the issue and added validation to the PeriodicAccumulationOracle with checks for accumulators to be updated.

	CurveLiquidityAccumulator.sol	CurvePriceAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	UniswapV2LiquidityAccumulator.sol	UniswapV2PriceAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	UniswapV3LiquidityAccumulator.sol	UniswapV3PriceAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	AbstractAccumulator.sol	LiquidityAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	PriceAccumulator.sol	AccumulationLibrary.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	ObservationLibrary.sol	AbstractOracle.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	AggregatedOracle.sol	PeriodicAccumulationOracle.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	PeriodicOracle.sol	ManagedCurveLiquidityAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	ManagedCurvePriceAccumulator.sol	ManagedUniswapV2Liquidity Accumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	ManagedUniswapV2PriceAccumulator.sol	ManagedUniswapV3LiquidityAccumulator.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	ManagedUniswapV3Price Accumulator.sol	ManagedPeriodicAccumulation Oracle.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Adrastia in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Adrastia contract requirements for details about issuance amounts and how the system handles these.

```
    Gas test for maximum oracles fit in Aggregated oracles
Around 1470 fit
      ✓ Gas test for how much oracles fits in aggregated oracle
Protocol test
    Uniswap V2 liquidity accumulator
      ✓ Should update liquidity for Uniswap V2 liquidity accumulator
      ✓ Update accumulations for Uniswap V2 liquidity accumulator
      ✓ Should revert if max age reached
    Uniswap V3 liquidity accumulator
      ✓ Should update liquidity for Uniswap V3 liquidity accumulator
      ✓ Should revert if max age reached
    Curve liquidity accumulator
      ✓ Should update liquidity for Curve liquidity accumulator
      ✓ Update accumulations for Curve liquidity accumulator
      ✓ Should revert if max age reached
    Uniswap V2 price accumulator
      ✓ Should update price for Uniswap V2 price accumulator
      ✓ Update price accumulation for Uniswap V2 Price accumulator
      ✓ Should revert if max age reached
    Uniswap V3 price accumulator
      ✓ Should update price for Uniswap V3 price accumulator
      ✓ Should revert if max age reached
    Curve price accumulator
      ✓ Should update price for Curve price accumulator
      ✓ Update price accumulation for Curve Price accumulator
      ✓ Should revert if max age reached
    Periodic accumulation oracle for Uniswap V2
      ✓ Should update liquidity and price
      ✓ Should revert if max age reached
```

```
Periodic accumulation oracle for Uniswap V3
  ✓ Should update liquidity and price
  ✓ Should revert if max age reached
Periodic accumulation oracle for Curve
  ✓ Should update liquidity and price
  ✓ Should revert if max age reached
Aggregated oracle
  ✓ Should update observations
  ✓ Should revert if max age reached
  ✓ Should not update Aggregated oracle if oracles are not up to date
26 passing (11m)
```

Notice! Zokyo Secured team has checked the whole set of contracts to operate correctly, with special attention to the oracles implementation. Additional set of unit-tests were written to verify several edge-cases, verify correctness of observations update and check gas-usage .

As the result - all Adrastia tests are verified, the oracles correct ovservations update was verified as well. Original functionality is well-tested, reviewed by Zokyo Secured team and, as a conclusion, has all necessary security checks.

Zokyo Secured team has carefully checked the whole set of unit tests (in the section below) and checked the original coverage of those tests. During the audit tests were verified for the correctness, wholesomeness, sufficient coverage, meaning of the scenarios, overall structure and the correct implementation (in spite of the covered functionality).

As the result - all tests are verified and the coverage is evaluated as conforming for security requirements. Original functionality has necessary coverage, standard and sub-standard functionality was verified by Zokyo Secured auditors in separate set of exploratory testing scenarios.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Adrastia team

As part of our work assisting Adrastia team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Adrastia team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Adrastia team. All of them were also carefully checked by the auditors' team.

```
AddressLibrary#isContract
    ✓ Should return false for our account address
    ✓ Should return true for a contract address
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
    ✓ everts 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
SafeCastExt#toUint112
    ✓ Should convert 0 to uint112 without error
    ✓ Should convert (2^112)-1 to uint112 without error
```

```
    ✓ Should revert with value 2^112
    ✓ Should revert with value (2^256)-1
CurveLiquidityAccumulator#constructor
    ✓ Should revert when given a [pool] quote token is not in the pool (our quote token is invalid)
    ✓ Should revert when given a [pool] quote token is not in the pool (our quote token is valid)
    ✓ Should set our quote token properly with a different pool quote token
CurveLiquidityAccumulator#canUpdate
    ✓ Should return false when given an invalid token
    ✓ Should return true when given a valid token
CurveLiquidityAccumulator#fetchLiquidity
    ✓ Should revert when given an invalid token
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 100000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 100000}
LiquidityAccumulator#fetchLiquidity
    ✓ tokenLiquidity = 0, quoteTokenLiquidity = 0
    ✓ tokenLiquidity = 1, quoteTokenLiquidity = 1
    ✓ tokenLiquidity = 10, quoteTokenLiquidity = 5
    ✓ tokenLiquidity = 101000000000000000000, quoteTokenLiquidity = 102000000000000000000
LiquidityAccumulator#getCurrentAccumulation
    ✓ Reverts if the last observation is uninitialized
LiquidityAccumulator#needsUpdate
    ✓ Shouldn't need update if delta time is less than the min update delay (update threshold passed)
    ✓ Shouldn't need update if delta time is less than the min update delay (update threshold not passed)
    ✓ Should need update if delta time is within min and max update delay (update threshold passed)
    ✓ Shouldn't need update if delta time is within min and max update delay (update threshold not
      passed)
    ✓ Should need update if delta time is >= max update delay (update threshold passed)
    ✓ Should need update if delta time is >= max update delay (update threshold not passed)
LiquidityAccumulator#canUpdate
  Can't update when it
    ✓ Doesn't need an update
  Can update when it
    ✓ Needs an update
LiquidityAccumulator#changeThresholdSurpassed
    ✓ Should evaluate to false with liquidities {0, 0} and update threshold undefined
    ✓ Should evaluate to true with liquidities {0, 100} and update threshold undefined
    ✓ Should evaluate to true with liquidities {100, 0} and update threshold undefined
    ✓ Should evaluate to false with liquidities {100, 101} and update threshold undefined
    ✓ Should evaluate to true with liquidities {100, 102} and update threshold undefined
    ✓ Should evaluate to true with liquidities {100, 103} and update threshold undefined
    ✓ Should evaluate to true with liquidities {100, 1000000} and update threshold undefined
```

✓ Should evaluate to false with liquidities {101, 100} and update threshold undefined
✓ Should evaluate to true with liquidities {102, 100} and update threshold undefined
✓ Should evaluate to true with liquidities {103, 100} and update threshold undefined
✓ Should evaluate to true with liquidities {1000000, 100} and update threshold undefined
Should evaluate to true with liquidities
✓ {115792089237316195423570985008687907853269984665640564039457584007913129639935, 100} and update threshold undefined
Should evaluate to true with liquidities {100,
✓ 115792089237316195423570985008687907853269984665640564039457584007913129639935} and update threshold undefined
Should evaluate to false with liquidities
{115792089237316195423570985008687907853269984665640564039457584007913129639
✓ 9935,
115792089237316195423570985008687907853269984665640564039457584007913129639
935} and update threshold undefined
✓ Should evaluate to false with liquidities {1000000000000000000, 1000000000000000000} and update threshold undefined
Should evaluate to true with liquidities {1000000000000000000, 1020000000000000000} and
✓ update threshold undefined √ Should evaluate to true with liquidities {1020000000000000000, 1000000000000000000} and update threshold undefined
✓ Should evaluate to true with liquidities {1000000000000000000000000000, 1020000000000000000000000000} and update threshold undefined
✓ Should evaluate to true with liquidities {1020000000000000000000000000, 1000000000000000000000000000} and update threshold undefined

LiquidityAccumulator#calculateLiquidity

Should evaluate to 1,1 using accumulations
✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1}, {"cumulativeTokenLiquidity":1,"cumulativeQuoteTokenLiquidity":1,"timestamp":2}}
Should evaluate to 0,1 using accumulations
✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1}, {"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":1,"timestamp":2}}
Should evaluate to 1,0 using accumulations
✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1}, {"cumulativeTokenLiquidity":1,"cumulativeQuoteTokenLiquidity":0,"timestamp":2}}
Should evaluate to 0,0 using accumulations
✓ {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":1}, {"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":2}}
Should evaluate to 1000000,1000000 using accumulations
✓ {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":1}, {"cumulativeTokenLiquidity":2000000,"cumulativeQuoteTokenLiquidity":2000000,"timestamp":2}}

```
    Should evaluate to 0,1000000 using accumulations
  ✓ {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":1},
    {"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":2000000,"timestamp":2}}
    Should evaluate to 1000000,0 using accumulations
  ✓ {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":1},
    {"cumulativeTokenLiquidity":2000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":2}}
    Should evaluate to 100000,100000 using accumulations
  ✓ {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":1},
    {"cumulativeTokenLiquidity":2000000,"cumulativeQuoteTokenLiquidity":2000000,"timestamp":11}}
    Should evaluate to 10,10 using accumulations
    {{"cumulativeTokenLiquidity":1000000,"cumulativeQuoteTokenLiquidity":1000000,"timestamp":100
  ✓ 000},
    {"cumulativeTokenLiquidity":2000000,"cumulativeQuoteTokenLiquidity":2000000,"timestamp":2000
    00}}
    Should evaluate to 10,10 using accumulations {{"cumulativeTokenLiquidity":
    {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"cumulativeQuoteTokenLiquidity":
  ✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"timestamp":10},
    {"cumulativeTokenLiquidity":9,"cumulativeQuoteTokenLiquidity":9,"timestamp":11}}
    Should revert with LiquidityAccumulator: TIMESTAMP_CANNOT_BE_ZERO using accumulations
  ✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":0},
    {"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":0}}
    Should revert with LiquidityAccumulator: TIMESTAMP_CANNOT_BE_ZERO using accumulations
  ✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":0},
    {"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1}}
    Should revert with LiquidityAccumulator: DELTA_TIME_CANNOT_BE_ZERO using accumulations
  ✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1},
    {"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1}}
    Should revert using accumulations
  ✓ {{"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":1},
    {"cumulativeTokenLiquidity":0,"cumulativeQuoteTokenLiquidity":0,"timestamp":0}}
LiquidityAccumulator#update
  ✓ Should update (initial) using args {"initialLiquidity":false,"overrideNeedsUpdate":
    {"needsUpdate":true}}
  ✓ Shouldn't update (initial) using args {"initialLiquidity":false,"overrideNeedsUpdate":
    {"needsUpdate":false}}
  ✓ Should update (initial) using args {"initialLiquidity":{"token":
    {"type":"BigNumber","hex":"0x00"},"quoteToken":{"type":"BigNumber","hex":"0x00"}}}
    Should update (initial) using args {"initialLiquidity":{"token":
  ✓ {"type":"BigNumber","hex":"0x00"},"quoteToken":
    {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}}
```

```
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"}}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"quoteToken":
  {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"}}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"}},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"quoteToken":
  {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"}},"overrideNeedsUpdate":
  {"needsUpdate":false}}
```

```
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialLiquidity":{"token":
✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"quoteToken":
  {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update using args {"initialLiquidity":{"token":
  {"type":"BigNumber","hex":"0x00"},"quoteToken":
✓ {"type":"BigNumber","hex":"0x00"}},"secondLiquidity":{"token":
  {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update using args {"initialLiquidity":{"token":
  {"type":"BigNumber","hex":"0x00"},"quoteToken":
✓ {"type":"BigNumber","hex":"0x00"}},"secondLiquidity":{"token":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update using args {"initialLiquidity":{"token":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"secondLiquidity":{"token":
  {"type":"BigNumber","hex":"0x00"},"quoteToken":
  {"type":"BigNumber","hex":"0x00"}},"overrideNeedsUpdate":{"needsUpdate":true}}
```

```
    ✓ Should update using args {"initialLiquidity":{"token":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"secondLiquidity":{"token":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialLiquidity":{"token":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"},"quoteToken":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"}},"secondLiquidity":{"token":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialLiquidity":{"token":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"secondLiquidity":{"token":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"},"quoteToken":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"}},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialLiquidity":{"token":
      {"type":"BigNumber","hex":"0x015af1d78b58c40000"},"quoteToken":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"}},"secondLiquidity":{"token":
      {"type":"BigNumber","hex":"0x0410d586a20a4c0000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialLiquidity":{"token":
      {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"quoteToken":
      {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"}},"secondLiquidity":{"token":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"quoteToken":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Shouldn't update when deltaTime = 0
    ✓ Shouldn't update when validateObservation returns false
LiquidityAccumulator#supportsInterface(interfaceId)
    ✓ Should support ILiquidityAccumulator
    ✓ Should support ILiquidityOracle
    ✓ Should support IQuoteToken
    ✓ Should support IUpdateable
    ✓ Should support IAccumulator
LiquidityAccumulator#consultLiquidity(token)
    ✓ Should revert when there's no observation
    ✓ Should return fixed values when token == quoteToken
    ✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
```

✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1000000000000000000)

LiquidityAccumulator#consultLiquidity(token, maxAge = 0)

✓ tokenLiquidity = 0 and quoteTokenLiquidity = 0
✓ tokenLiquidity = 0 and quoteTokenLiquidity = 1
✓ tokenLiquidity = 0 and quoteTokenLiquidity = 1000000000000000000
✓ tokenLiquidity = 0 and quoteTokenLiquidity = 51922968585348276285304963292200 95
✓ tokenLiquidity = 1 and quoteTokenLiquidity = 0
✓ tokenLiquidity = 1 and quoteTokenLiquidity = 1
✓ tokenLiquidity = 1 and quoteTokenLiquidity = 1000000000000000000
✓ tokenLiquidity = 1 and quoteTokenLiquidity = 51922968585348276285304963292200 95
✓ tokenLiquidity = 1000000000000000000 and quoteTokenLiquidity = 0
✓ okenLiquidity = 1000000000000000000 and quoteTokenLiquidity = 1
✓ tokenLiquidity = 1000000000000000000 and quoteTokenLiquidity = 1000000000000000000
✓ tokenLiquidity = 1000000000000000000 and quoteTokenLiquidity = 51922968585348276285304963292200 95
✓ tokenLiquidity = 51922968585348276285304963292200 95 and quoteTokenLiquidity = 0
✓ tokenLiquidity = 51922968585348276285304963292200 95 and quoteTokenLiquidity = 1
✓ tokenLiquidity = 51922968585348276285304963292200 95 and quoteTokenLiquidity = 1000000000000000000
✓ tokenLiquidity = 51922968585348276285304963292200 95 and quoteTokenLiquidity = 51922968585348276285304963292200 95

LiquidityAccumulator#consultLiquidity(token, maxAge = 60)

✓ Should revert when there's no observation
✓ Should revert when the rate is expired (deltaTime > maxAge)
✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
✓ Should return fixed values when token == quoteToken
✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1000000000000000000)

LiquidityAccumulator#validateObservation(token, tokenLiquidity, quoteTokenLiquidity)

✓ Should revert when caller is a smart contract

Caller is not a smart contract

✓ Should return true when provided liquidity levels match the observed levels
✓ Should return false when the observed token liquidity is too different from the provided value
✓ Should return false when the observed quote token liquidity is too different from the provided value

```
    ✓ Should return false when the observed liquidity levels are too different from the provided values
UniswapV2LiquidityAccumulator
  UniswapV2LiquidityAccumulator#canUpdate
  Can't update when
    ✓ token = address(0)
    ✓ token = quoteToken
    ✓ The pool doesn't exist
  Can update when
    ✓ The pool exists
  UniswapV2LiquidityAccumulator#fetchLiquidity
    ✓ Should revert when the pair does not exist
    ✓ Should revert if token == quoteToken
    ✓ Should revert if token == address(0)
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 100000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 100000}
UniswapV3LiquidityAccumulator
  UniswapV3LiquidityAccumulator#canUpdate
  Can't update when
    ✓ token = address(0)
    ✓ token = quoteToken
  Can update when
    ✓ token != quoteToken
  UniswapV3LiquidityAccumulator#fetchLiquidity
    ✓ Shouldn't revert when there are no pools
    ✓ Should revert if token == quoteToken
    ✓ Should revert if token == address(0)
    ✓ Pools with zero in-range liquidity should be ignored
  Providing liquidity for poolFees = [ 500 ]
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 100000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 100000}
  Providing liquidity for poolFees = [ 500, 3000 ]
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 100000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 100000}
  Providing liquidity for poolFees = [ 500, 3000, 10000 ]
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 100000, quoteTokenLiquidity = 10000}
    ✓ Should get liquidities {tokenLiqudity = 10000, quoteTokenLiquidity = 100000}
AggregatedOracle#constructor
    ✓ Should deploy correctly with valid arguments
```

```
    ✓ Should revert if no underlying oracles are provided
    ✓ Should revert if duplicate general oracles are provided
    ✓ Should revert if duplicate token specific oracles are provided
    ✓ Should revert if duplicate general / token specific oracles are provided
    ✓ Should revert if duplicate general / token specific oracles are provided
AggregatedOracle#needsUpdate
    ✓ Should require an update if no observations have been made
    ✓ Should require an update if deltaTime == period
    ✓ Should require an update if deltaTime > period
    ✓ Shouldm't require an update if deltaTime < period
    ✓ Shouldm't require an update if deltaTime == 0
AggregatedOracle#canUpdate
  Can't update when it
    ✓ Doesn't need an update
    ✓ Needs an update but there are no valid underlying oracle responses
    ✓ Needs an update but all of the underlying oracles fail validation
  Can update when it needs an update and when
    ✓ An underlying oracle needs an update
    ✓ An underlying oracle doesn't need an update but it has valid data
AggregatedOracle#consultPrice(token)
    ✓ Should revert when there's no observation
    ✓ Should get the set price (=1)
    ✓ Should get the set price (=1e18)
    ✓ Should return fixed values when token == quoteToken
AggregatedOracle#consultPrice(token, maxAge = 0)
    ✓ Should get the set price (=1)
AggregatedOracle#consultPrice(token, maxAge)
    ✓ Should revert when there's no observation
    ✓ Should revert when the rate is expired (deltaTime > maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
    ✓ Should get the set price (=1)
    ✓ Should get the set price (=1e18)
    ✓ Should return fixed values when token == quoteToken
AggregatedOracle#consultLiquidity(token)
    ✓ Should revert when there's no observation
    ✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
    ✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
    ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
    ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity
      (=1000000000000000000)
AggregatedOracle#consultLiquidity(token, maxAge = 0)
    ✓ Should get the set price (=1)
```

AggregatedOracle#consultLiquidity(token, maxAge)

- ✓ Should revert when there's no observation
- ✓ Should revert when the rate is expired (deltaTime > maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1000000000000000000)

AggregatedOracle#consult(token)

- ✓ Should revert when there's no observation
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)

AggregatedOracle#consult(token, maxAge = 0)

- ✓ Should get the set price (=1)
- ✓ Should revert when there are no valid responses
- ✓ Should revert when the price exceeds uint112.max
- ✓ Should report token liquidity of uint112.max when it exceeds that
- ✓ Should report quote token liquidity of uint112.max when it exceeds that
- ✓ Should report liquidities of uint112.max when they exceeds that

AggregatedOracle#consult(token, maxAge)

- ✓ Should revert when there's no observation
- ✓ Should revert when the rate is expired (deltaTime > maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)

- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)

AggregatedOracle#update w/ 1 underlying oracle

- ✓ Should update successfully
- ✓ Has correct price and quote token liquidity when the oracle has delta +2 quote token decimals
- ✓ Has correct price and quote token liquidity when the oracle has delta -2 quote token decimals
- ✓ Shouldn't use old rates
- ✓ Should catch underlying consult errors and not update
- ✓ Should catch underlying update errors and update
- ✓ Should catch underlying update errors (w/ reason) and update
- ✓ Shouldn't update when there aren't any valid consultations
- ✓ Shouldn't update when the underlying oracle's price is 0
- ✓ Shouldn't update when the underlying oracle's quote token liquidity is 0
- ✓ Shouldn't update when the underlying oracle's price and quote token liquidity is 0

AggregatedOracle#update w/ 2 underlying oracle

- ✓ Should set observation liquitities to (2^112)-1 when total liquitities >= 2^112
- ✓ Should update successfully w/ same prices and liquidities
- ✓ Should update successfully w/ differing prices and liquidities

AggregatedOracle#update w/ 1 general underlying oracle and one token specific oracle

- ✓ Should call update on both the general oracle and the token specific oracle (underlying)
- ✓ Should call update on only the general oracle (underlying)

AggregatedOracle#update w/ 1 underlying oracle and a minimum token liquidity value

- ✓ Shouldn't update when the underlying oracle has less token liquidity value than the minimum
- ✓ Should update successfully when the underlying oracle has the minimum token liquidity value

AggregatedOracle#update w/ 1 underlying oracle and a minimum quote token liquidity

- ✓ Shouldn't update when the underlying oracle has less token quote token liquidity than the minimum
- ✓ Should update successfully when the underlying oracle has the minimum quote token liquidity

AggregatedOracle#update w/ 1 underlying oracle and an allowed TVL distribution range

```
    ✓ Shouldn't update when the underlying oracle has one-sided liquidity (100:1)
    ✓ Should update successfully when the underlying oracle has balanced liquidity (10:1)
    ✓ Should update successfully when the underlying oracle has balanced liquidity (1:1)
AggregatedOracle#update w/ 2 underlying oracles but one failing validation
    ✓ Should update successfully using only the data from the passing oracle
AggregatedOracle#sanityCheckQuoteTokenLiquidity
  Minimum quote token liquidity = 0
    ✓ Should return true when quoteTokenLiquidity = 0
    ✓ Should return true when quoteTokenLiquidity = 1
    ✓ Should return true when quoteTokenLiquidity = 999999999999999999
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000001
    ✓ Should return true when quoteTokenLiquidity = 999999999999999999999999999
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000001
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639934
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639935
  Minimum quote token liquidity = 1000000000000000000
    ✓ Should return false when quoteTokenLiquidity = 0
    ✓ Should return false when quoteTokenLiquidity = 1
    ✓ Should return false when quoteTokenLiquidity = 999999999999999999
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000001
    ✓ Should return true when quoteTokenLiquidity = 999999999999999999999999999
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000001
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639934
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639935
  Minimum quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when quoteTokenLiquidity = 0
    ✓ Should return false when quoteTokenLiquidity = 1
    ✓ Should return false when quoteTokenLiquidity = 999999999999999999
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000000
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000001
    ✓ Should return false when quoteTokenLiquidity = 999999999999999999999999999
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000000000001
```

```
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639934
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639935
  Minimum quote token liquidity =
  115792089237316195423570985008687907853269984665640564039457584007913129639935
    ✓ Should return false when quoteTokenLiquidity = 0
    ✓ Should return false when quoteTokenLiquidity = 1
    ✓ Should return false when quoteTokenLiquidity = 999999999999999999
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000000
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000001
    ✓ Should return false when quoteTokenLiquidity = 999999999999999999999999999
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000000000000000
    ✓ Should return false when quoteTokenLiquidity = 1000000000000000000000000001
    ✓ Should return false when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639934
    ✓ Should return true when quoteTokenLiquidity =
      115792089237316195423570985008687907853269984665640564039457584007913129639935
AggregatedOracle#sanityCheckTokenLiquidityValu
  Minimum token liquidity value = 0
  Token decimals = 0
  Price = 0
  Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 0
  Price = 1
  Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000
  Token liquidity = 1000000000000000000
    ✓ Should return true when quoteTokenLiquidity = 1000000000000000000
  Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
  Price = 1000000
  Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
```

```
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000000
Token decimals = 1
Price = 0
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
```

```
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000000
Token decimals = 6
Price = 0
Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
```

```
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token decimals = 18
Price = 0
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000
Price = 1000000000000000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return true when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Minimum token liquidity value = 1000000000000000000
Token decimals = 0
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000
Price = 1000000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000000000000000000000000000000
Token decimals = 1
Price = 0
```

```
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 100000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000000
Token decimals = 6
Price = 0
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token decimals = 18
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1
```

```
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000
Price = 1000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Minimum token liquidity value = 1000000000000000000000000000
Token decimals = 0
Price = 0
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
```

```
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000000
Token decimals = 1
Price = 0
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000
Token liquidity = 0
  ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
  ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000000000
Token decimals = 6
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
```

```
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token decimals = 18
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
```

```
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return true when tokenLiquidityValue = 1000000000000000000000000000000000
Minimum token liquidity value = 115792089237316195423570985008687907853269984665640564039457
Token decimals = 0
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000000000000(
Token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return false when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000000000
  Token decimals = 1
  Price = 0
  Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
  Price = 1
  Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000
  Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000
  Price = 1000000
  Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000
  Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000
  Price = 1000000000000000000
  Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000
  Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000
  Price = 1000000000000000000000000
  Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
  Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000000000
  Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue =
      1000000000000000000000000000000000000000000000000
  Token decimals = 6
  Price = 0
  Token liquidity = 0
```

```
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Price = 1
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000
Price = 1000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000
Price = 1000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000000000
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue =
    1000000000000000000000000000000000000000000000
Token decimals = 18
Price = 0
Token liquidity = 0
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
Token liquidity = 1000000000000000000000000000
    ✓ Should return false when tokenLiquidityValue = 0
```

```
    Price = 1
    Token liquidity = 0
      ✓ Should return false when tokenLiquidityValue = 0
    Token liquidity = 1000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1
    Token liquidity = 1000000000000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000000
    Price = 1000000
    Token liquidity = 0
      ✓ Should return false when tokenLiquidityValue = 0
    Token liquidity = 1000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000
    Token liquidity = 1000000000000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000000000000
    Price = 1000000000000000000
    Token liquidity = 0
      ✓ Should return false when tokenLiquidityValue = 0
    Token liquidity = 1000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000000000000000
    Token liquidity = 1000000000000000000000000000
      ✓  Should return false when tokenLiquidityValue = 1000000000000000000000000000
    Price = 1000000000000000000000000
    Token liquidity = 0
      ✓ Should return false when tokenLiquidityValue = 0
    Token liquidity = 1000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000
    Token liquidity = 1000000000000000000000000000
      ✓ Should return false when tokenLiquidityValue = 1000000000000000000000000000000000
  AggregatedOracle#sanityCheckTvlDistributionRatio
    Token decimals = 0
    Price = 0
    Token liquidity = 0
    Quote token liquidity = 0
      ✓ Should return false when tvl distribution ratio = 0
    Quote token liquidity = 0
      ✓ Should return false when tvl distribution ratio = 0
    Quote token liquidity = 0
      ✓ Should return false when tvl distribution ratio = 0
    Quote token liquidity = 0
      ✓ Should return false when tvl distribution ratio = 0
    Token liquidity = 1000000000000000000
    Quote token liquidity = 1000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Price = 1
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
```

```
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 90
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 110000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 110
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 100
Price = 1000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
```

```
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 90909090
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 110000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 110000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000
Price = 1000000000000000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 90909090909
Token liquidity = 10000000000000000000000000000
Quote token liquidity = 10000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 10000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000000000000000000000
Quote token liquidity = 10000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000000000000
Quote token liquidity = 10000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 90909090909090909090
Token liquidity = 110000000000000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000000000000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000000000
Price = 100000000000000000000000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 90909090909090909
Token liquidity = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓  Should return false when tvl distribution ratio = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 909090909090909090909090909090
Token liquidity = 110000000000000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 110000000000000000000000000000000000
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000000000000000000000
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000000000000000
Token decimals = 1
Price = 0
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 110000000000000000000000000
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 0
Price = 1
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return true when tvl distribution ratio = 10
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 10000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 10
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 9
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 11000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 11
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 10
Price = 1000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
```

```
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 10000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 10000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 10000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 9090909
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 11000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 11000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 10000000
Price = 1000000000000000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
```

```
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 9090909090
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 9090909090909090909
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 11000000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 11000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Price = 1000000000000000000000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 10000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 9090909090909090
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1000000000000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 9090909090909090909090909
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1100000000000000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1000000000000000000000000000
Token decimals = 6
Price = 0
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Price = 1
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 110000000000000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000000
    ✓ hould return false when tvl distribution ratio = 110000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Price = 1000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 100000000000000000000000000000
Quote token liquidity = 100000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000
Quote token liquidity = 100000000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 100000000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 90
Token liquidity = 110000000000000000000000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 110000000000
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 110
Quote token liquidity = 110000000000000000000000000000
    ✓ Should return true when tvl distribution ratio = 100
Price = 1000000000000000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 90909
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 90909090909090
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 1100000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000000000
```

```
Price = 1000000000000000000000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 90909090909
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 90909090909090909090
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 11000000000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 11000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 10000000000000000000
Token decimals = 18
Price = 0
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
```

```
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Price = 1
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
```

```
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 110000000000000000000000000
Quote token liquidity = 110000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Price = 1000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 110000000000000000000000000
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 110000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Price = 1000000000000000000
Token liquidity = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 100000000000000000000000000
Quote token liquidity = 100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
```

```
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 100
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 90
Token liquidity = 1100000000000000000000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 110000000000
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 110
Quote token liquidity = 1100000000000000000000000000
  ✓ Should return true when tvl distribution ratio = 100
Price = 1000000000000000000000000
Token liquidity = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 0
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Token liquidity = 1000000000000000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 0
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000000000000
Quote token liquidity = 1000000000000000000000000000
  ✓ Should return false when tvl distribution ratio = 100000000
Quote token liquidity = 1000000000000000000000000000
```

```
    ✓ Should return false when tvl distribution ratio = 90909090
  Token liquidity = 1100000000000000000000000000
  Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 0
  Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000000000000
  Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 110000000
  Quote token liquidity = 1100000000000000000000000000
    ✓ Should return false when tvl distribution ratio = 100000000
AggregatedOracle#validateUnderlyingConsultation
  Sanity check tvl distribution ratio = true
  Sanity check quote token liquidity = true
  Sanity check token liquidity value = true
    ✓ Should return true
  Sanity check token liquidity value = false
    ✓ Should return false
  Sanity check quote token liquidity = false
  Sanity check token liquidity value = true
    ✓ Should return false
  Sanity check token liquidity value = false
    ✓ Should return false
  Sanity check tvl distribution ratio = false
  Sanity check quote token liquidity = true
  Sanity check token liquidity value = true
    ✓ Should return false
  Sanity check token liquidity value = false
    ✓ Should return false
  Sanity check quote token liquidity = false
  Sanity check token liquidity value = true
    ✓ Should return false
  Sanity check token liquidity value = false
    ✓ Should return false
AggregatedOracle#supportsInterface(interfaceId)
    ✓ Should support IAggregatedOracle
    ✓ Should support IOracle
    ✓ Should support IPeriodic
    ✓ Should support IPriceOracle
    ✓ Should support ILiquidityOracle
    ✓ Should support IQuoteToken
    ✓ Should support IUpdateable
PeriodicAccumulationOracle#constructor
```

```
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0x0000000000000000000000000000000000000000","priceAccumulator":"0
    x0000000000000000000000000000000000000000","quoteToken":"0x0000000000000000000000000
    0000000000000000","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0x0000000000000000000000000000000000000000","priceAccumulator":"0
    x0000000000000000000000000000000000000000","quoteToken":"0xA0b86991c6218b36c1d19D4a
    2e9Eb0cE3606eB48","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0x0000000000000000000000000000000000000000","priceAccumulator":"0
    xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","quoteToken":"0x0000000000000000000000000
    0000000000000000","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0x0000000000000000000000000000000000000000","priceAccumulator":"0
    xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","quoteToken":"0xA0b86991c6218b36c1d19D4a
    2e9Eb0cE3606eB48","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","priceAccumulator":"0
    x0000000000000000000000000000000000000000","quoteToken":"0x0000000000000000000000000
    0000000000000000","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","priceAccumulator":"0
    x0000000000000000000000000000000000000000","quoteToken":"0xA0b86991c6218b36c1d19D4a
    2e9Eb0cE3606eB48","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","priceAccumulator":"0
    xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","quoteToken":"0x0000000000000000000000000
    0000000000000000","period":{"type":"BigNumber","hex":"0x64"}}
    Should construct the oracle correctly with params
✓   {"liquidityAccumulator":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","priceAccumulator":"0
    xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48","quoteToken":"0xA0b86991c6218b36c1d19D4a
    2e9Eb0cE3606eB48","period":{"type":"BigNumber","hex":"0x64"}}
PeriodicAccumulationOracle#needsUpdate
    ✓ Should require an update if no observations or accumulations have been made
    ✓ Should require an update if deltaTime == period
    ✓ Should require an update if deltaTime > period
    ✓ Shouldn't require an update if deltaTime < period
    ✓ Shouldn't require an update if deltaTime == 0
PeriodicAccumulationOracle#canUpdate
    ✓ Can update when it needs an update and both of the accumulators have been initialized
    ✓ Can't update when it needs an update but both of the accumulators haven't been initialized
    ✓ Can't update when it needs an update but the price accumulator hasn't been initialized
    ✓ Can't update when it needs an update but the liquidity accumulator hasn't been initialized
```

- ✓ Can't update when it doesn't needs an update and both of the accumulators have been initialized
- ✓ Can't update when it doesn't need an update and both of the accumulators haven't been initialized
- ✓ Can't update when it doesn't need an update and the price accumulator hasn't been initialized
- ✓ Can't update when it doesn't need an update and the liquidity accumulator hasn't been initialized

PeriodicAccumulationOracle#consultPrice(token)

- ✓ Should revert when there's no observation
- ✓ Should get the set price (=1)
- ✓ Should get the set price (=1e18)
- ✓ Should return fixed values when token == quoteToken

PeriodicAccumulationOracle#consultPrice(token, maxAge = 0)

- ✓ Should get the set price (=1)

PeriodicAccumulationOracle#consultPrice(token, maxAge)

- ✓ Should revert when there's no observation
- ✓ Should revert when the rate is expired (deltaTime > maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
- ✓ Should get the set price (=1)
- ✓ Should get the set price (=1e18)
- ✓ Should return fixed values when token == quoteToken

PeriodicAccumulationOracle#consultLiquidity(token)

- ✓ Should revert when there's no observation
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1000000000000000000)

PeriodicAccumulationOracle#consultLiquidity(token, maxAge = 0)

- ✓ Should get the set liquidities (2,3)

PeriodicAccumulationOracle#consultLiquidity(token, maxAge)

- ✓ Should revert when there's no observation
- ✓ Should revert when the rate is expired (deltaTime > maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1) and quote token liquidity (=1000000000000000000)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1)
- ✓ Should get the set token liquidity (=1000000000000000000) and quote token liquidity (=1000000000000000000)

PeriodicAccumulationOracle#consult(token)

- ✓ Should revert when there's no observation
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)

PeriodicAccumulationOracle#consult(token, maxAge = 0)

- ✓ Should get the set price (1) and liquidities (2,3)

PeriodicAccumulationOracle#consult(token, maxAge)

- ✓ Should revert when there's no observation
- ✓ Should revert when the rate is expired (deltaTime > maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
- ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
- ✓ Should return fixed values when token == quoteToken
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1), and quote token liquidity (=1000000000000000000)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1)
- ✓ Should get the set price (=1000000000000000000), token liquidity (=1000000000000000000), and quote token liquidity (=1000000000000000000)

PeriodicAccumulationOracle#update

- ✓ Shouldn't update anything if the current accumulations' timestamps equals the last
- ✓ Shouldn't update observation timestamp or emit Updated when there's no price
- ✓ Should revert if token == quoteToken
- ✓ Should revert if token == address(0)
- ✓ Shouldn't update if not needed

Return value differences with different accumulations

- ✓ Should return false when both of the accumulations haven't changed
- ✓ Should return true when the liquidity accumulation has been updated, but the price accumulation has not
- ✓ Should return true when the price accumulation has been updated, but the liquidity accumulation has not
- ✓ Should return true when the both the price and the liquidity accumulations have been updated

token decimals = 18, quoteToken decimals = 18

- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=500000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=500000000000000000000
- ✓ Should update successfully with tokenLiquidity=500000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=500000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=500000000000000000000 and quoteTokenLiquidity=500000000000000000000

token decimals = 6, quoteToken decimals = 18

- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=500000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000
- ✓ Should update successfully with tokenLiquidity=100000000000000000000 and quoteTokenLiquidity=100000000000000000000

```
    ✓ Should update successfully with tokenLiquidity=100000000000000000000000 and
      quoteTokenLiquidity=500000000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000000 and
      quoteTokenLiquidity=100000000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000000 and
      quoteTokenLiquidity=100000000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000000 and
      quoteTokenLiquidity=500000000000000000000000
  token decimals = 18, quoteToken decimals = 6
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=500000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=500000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000 and
      quoteTokenLiquidity=500000000000000000000
  token decimals = 6, quoteToken decimals = 6
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=500000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
    ✓ Should update successfully with tokenLiquidity=100000000000000000000 and
      quoteTokenLiquidity=500000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000 and
      quoteTokenLiquidity=100000000000000000000
```

```
    ✓ Should update successfully with tokenLiquidity=500000000000000000000000 and
      quoteTokenLiquidity=1000000000000000000000
    ✓ Should update successfully with tokenLiquidity=500000000000000000000000 and
      quoteTokenLiquidity=500000000000000000000000
PeriodicAccumulationOracle#supportsInterface(interfaceId)
    ✓ Should support IOracle
    ✓ Should support IPeriodic
    ✓ Should support IPriceOracle
    ✓ Should support ILiquidityOracle
    ✓ Should support IQuoteToken
    ✓ Should support IUpdateable
    ✓ Should support IHasLiquidityAccumulator
    ✓ Should support IHasPriceAccumulator
CurvePriceAccumulator#constructor
    ✓ Should revert when given a [pool] quote token is not in the pool (our quote token is invalid)
    ✓ Should revert when given a [pool] quote token is not in the pool (our quote token is valid)
    ✓ Should set our quote token properly with a different pool quote token
CurvePriceAccumulator#canUpdate
    ✓ Should return false when given an invalid token
    ✓ Should return true when given a valid token
CurvePriceAccumulator#fetchPrice
    ✓ Should revert when given an invalid token
    ✓ price = 0
    ✓ price = 1
    ✓ price = 1000000000000000000
    ✓ price = 1000000000000000000000000
PriceAccumulator#fetchPrice
    ✓ price = 0
    ✓ price = 1
    ✓ price = 10
    ✓ price = 101000000000000000000
PriceAccumulator#getCurrentAccumulation
    ✓ Reverts if the last observation is uninitialized
PriceAccumulator#needsUpdate
    ✓ Shouldn't need update if delta time is less than the min update delay (update threshold passed)
    ✓ Shouldn't need update if delta time is less than the min update delay (update threshold not passed)
    ✓ Should need update if delta time is within min and max update delay (update threshold passed)
    ✓ Shouldn't need update if delta time is within min and max update delay (update threshold not
      passed)
    ✓ Should need update if delta time is >= max update delay (update threshold passed)
    ✓ Should need update if delta time is >= max update delay (update threshold not passed)
PriceAccumulator#canUpdate
  Can't update when it
```

```
    ✓ Doesn't need an update
  Can update when it
    ✓ Needs an update
PriceAccumulator#changeThresholdSurpassed
    ✓ Should evaluate to false with prices {0, 0} and update threshold 2000000
    ✓ Should evaluate to true with prices {0, 100} and update threshold 2000000
    ✓ Should evaluate to true with prices {100, 0} and update threshold 2000000
    ✓ Should evaluate to false with prices {100, 101} and update threshold 2000000
    ✓ Should evaluate to true with prices {100, 102} and update threshold 2000000
    ✓ Should evaluate to true with prices {100, 103} and update threshold 2000000
    ✓ Should evaluate to true with prices {100, 1000000} and update threshold 2000000
    ✓ Should evaluate to false with prices {101, 100} and update threshold 2000000
    ✓ Should evaluate to true with prices {102, 100} and update threshold 2000000
    ✓ Should evaluate to true with prices {103, 100} and update threshold 2000000
    ✓ Should evaluate to true with prices {1000000, 100} and update threshold 2000000
      Should evaluate to true with prices
    ✓ {115792089237316195423570985008687907853269984665640564039457584007913129639935,
      100} and update threshold 2000000
      Should evaluate to true with prices {100,
    ✓ 115792089237316195423570985008687907853269984665640564039457584007913129639935}
      and update threshold 2000000
      Should evaluate to false with prices
    ✓ {115792089237316195423570985008687907853269984665640564039457584007913129639935,
      115792089237316195423570985008687907853269984665640564039457584007913129639935}
      and update threshold 2000000
    ✓ Should evaluate to false with prices {1000000000000000000, 1000000000000000000} and update
      threshold 2000000
    ✓ Should evaluate to true with prices {1000000000000000000, 1020000000000000000} and update
      threshold 2000000
    ✓ Should evaluate to true with prices {1020000000000000000, 1000000000000000000} and update
      threshold 2000000
    ✓ Should evaluate to true with prices {100000000000000000000000000000,
      102000000000000000000000000000} and update threshold 2000000
    ✓ Should evaluate to true with prices {102000000000000000000000000000,
      100000000000000000000000000000} and update threshold 2000000
PriceAccumulator#calculatePrice
    ✓ Should evaluate to 1 using accumulations {{"cumulativePrice":0,"timestamp":1},
      {"cumulativePrice":1,"timestamp":2}}
    ✓ Should evaluate to 0 using accumulations {{"cumulativePrice":0,"timestamp":1},
      {"cumulativePrice":0,"timestamp":2}}
    ✓ Should evaluate to 0 using accumulations {{"cumulativePrice":1000000,"timestamp":1},
      {"cumulativePrice":1000000,"timestamp":2}}
```

```
    ✓ Should evaluate to 1000000 using accumulations {{"cumulativePrice":1000000,"timestamp":1},
      {"cumulativePrice":2000000,"timestamp":2}}
    ✓ Should evaluate to 100000 using accumulations {{"cumulativePrice":1000000,"timestamp":1},
      {"cumulativePrice":2000000,"timestamp":11}}
    ✓ Should evaluate to 10 using accumulations {{"cumulativePrice":1000000,"timestamp":100000},
      {"cumulativePrice":2000000,"timestamp":200000}}
    ✓ Should evaluate to 10 using accumulations {{"cumulativePrice":
      {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"timestamp":10},
      {"cumulativePrice":9,"timestamp":11}}
    ✓ Should revert with PriceAccumulator: TIMESTAMP_CANNOT_BE_ZERO using accumulations
      {{"cumulativePrice":0,"timestamp":0}, {"cumulativePrice":0,"timestamp":0}}
    ✓ Should revert with PriceAccumulator: TIMESTAMP_CANNOT_BE_ZERO using accumulations
      {{"cumulativePrice":0,"timestamp":0}, {"cumulativePrice":0,"timestamp":1}}
    ✓ Should revert with PriceAccumulator: DELTA_TIME_CANNOT_BE_ZERO using accumulations
      {{"cumulativePrice":0,"timestamp":1}, {"cumulativePrice":0,"timestamp":1}}
    ✓ Should revert using accumulations {{"cumulativePrice":0,"timestamp":1},
      {"cumulativePrice":0,"timestamp":0}}
PriceAccumulator#update
    ✓ Should update (initial) using args {"overrideNeedsUpdate":{"needsUpdate":true}}
    ✓ Shouldn't update (initial) using args {"overrideNeedsUpdate":{"needsUpdate":false}}
    ✓ Should update (initial) using args {"initialPrice":{"type":"BigNumber","hex":"0x00"}}
    ✓ Should update (initial) using args {"initialPrice":{"type":"BigNumber","hex":"0x00"}}
    ✓ Should update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}
    ✓ Should update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}
    ✓ Should update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"}}
    ✓ Should update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"}}
    ✓ Should update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"}}
    ✓ Shouldn't update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":false}}
    ✓ Shouldn't update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":false}}
    ✓ Shouldn't update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
      {"needsUpdate":false}}
    ✓ Shouldn't update (initial) using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
      {"needsUpdate":false}}
```

```
  Shouldn't update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Shouldn't update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"overrideNeedsUpdate":
  {"needsUpdate":false}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x0ad78ebc5ac6200000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update (initial) using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update using args {"initialPrice":{"type":"BigNumber","hex":"0x00"},"secondPrice":
✓ {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update using args {"initialPrice":{"type":"BigNumber","hex":"0x00"},"secondPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"secondPrice":
  {"type":"BigNumber","hex":"0x00"},"overrideNeedsUpdate":{"needsUpdate":true}}
  Should update using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"secondPrice":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
  Should update using args {"initialPrice":
✓ {"type":"BigNumber","hex":"0x02b5e3af16b1880000"},"secondPrice":
  {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
  {"needsUpdate":true}}
```

```
    ✓ Should update using args {"initialPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"secondPrice":
      {"type":"BigNumber","hex":"0x02b5e3af16b1880000"},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialPrice":
      {"type":"BigNumber","hex":"0x015af1d78b58c40000"},"secondPrice":
      {"type":"BigNumber","hex":"0x0410d586a20a4c0000"},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Should update using args {"initialPrice":
      {"type":"BigNumber","hex":"0xffffffffffffffffffffffffffffff"},"secondPrice":
      {"type":"BigNumber","hex":"0x056bc75e2d63100000"},"overrideNeedsUpdate":
      {"needsUpdate":true}}
    ✓ Shouldn't update when deltaTime = 0
    ✓ Shouldn't update when validateObservation returns false
PriceAccumulator#supportsInterface(interfaceId)
    ✓ Should support IPriceAccumulator
    ✓ Should support IPriceOracle
    ✓ Should support IQuoteToken
    ✓ Should support IUpdateable
    ✓ Should support IAccumulator
PriceAccumulator#consultPrice(token)
    ✓ Should revert when there's no observation
    ✓ Should get the set price (=1)
    ✓ Should get the set price (=1e18)
    ✓ Should return fixed values when token == quoteToken
PriceAccumulator#consultPrice(token, maxAge = 0)
    ✓ price = 0
    ✓ price = 1
    ✓ price = 1000000000000000000
    ✓ price = 51922968585348276285304963292200 95
PriceAccumulator#consultPrice(token, maxAge = 60)
    ✓ Should revert when there's no observation
    ✓ Should revert when the rate is expired (deltaTime > maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime == maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime < maxAge)
    ✓ Shouldn't revert when the rate is not expired (deltaTime == 0)
    ✓ Should get the set price (=1)
    ✓ Should get the set price (=1e18)
    ✓ Should return fixed values when token == quoteToken
PriceAccumulator#validateObservation(token, price)
    ✓ Should revert when caller is a smart contract
  Caller is not a smart contract
    ✓ Should return true when provided price matches the observed price
```

```
    ✓ Should return false when the observed price is too different from the provided value
UniswapV2PriceAccumulator#computeWholeUnitAmount
    ✓ Should verify that a token with 0 decimals has a whole unit amount of 1
    ✓ Should verify that a token with 1 decimals has a whole unit amount of 10
    ✓ Should verify that a token with 6 decimals has a whole unit amount of 1000000
    ✓ Should verify that a token with 18 decimals has a whole unit amount of 1000000000000000000
UniswapV2PriceAccumulator
  UniswapV2PriceAccumulator#canUpdate
  Can't update when
    ✓ token = address(0)
    ✓ token = quoteToken
    ✓ The pool doesn't exist
    ✓ The pool has no liquidity
    ✓ The pool has no liquidity (tokens transferred but not minted)
  Can update when
    ✓ The pool exists and has liquidity
  UniswapV2PriceAccumulator#fetchPrice
  Should revert when
    ✓ The pool is not found
    ✓ The pool has no liquidity
    ✓ The pool has no liquidity (tokens transferred but not minted)
    ✓ token = address(0)
    ✓ token = quoteToken
  token < quoteToken
  token decimals = 6, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      10000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 10000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000 and quoteTokenAmount =
      5000000000000000000
      fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000 and quoteTokenAmount =
    ✓ 3000000000000000000      √ fetchPrice(token) = 0 with tokenAmount =
      500000000000000000000000000 and quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 10000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 10000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 1000000000000000000
```

```
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000 and
      quoteTokenAmount = 5000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000 and
      quoteTokenAmount = 3000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 6, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount
      = 1000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000 and quoteTokenAmount =
      5000000000000000000
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000 and quoteTokenAmount =
      3000000000000000000      √ fetchPrice(token) = 0 with tokenAmount =
      500000000000000000000000000 and quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 10000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000 and
      quoteTokenAmount = 5000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000 and
      quoteTokenAmount = 3000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token > quoteToken
  token decimals = 6, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      10000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 10000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000 and quoteTokenAmount =
      5000000000000000000
    ✓ etchPrice(token) = 600000 with tokenAmount = 5000000000000000000 and quoteTokenAmount =
      3000000000000000000
```

```
    ✓ fetchPrice(token) = 0 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 10000000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 10000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000 and
      quoteTokenAmount = 5000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000 and
      quoteTokenAmount = 3000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 6, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000 and quoteTokenAmount
      = 10000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 10000000000000000000 and quoteTokenAmount =
      1000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000 and quoteTokenAmount =
      5000000000000000000
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000 and quoteTokenAmount =
      3000000000000000000
    ✓ fetchPrice(token) = 0 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 10000000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 10000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 10000000000000000000 and
      quoteTokenAmount = 1000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000 and
      quoteTokenAmount = 5000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000 and
      quoteTokenAmount = 3000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 500000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
UniswapV3PriceAccumulator#calculatePriceFromSqrtPrice
  token < quoteToken
```

```
        ✓ Calculates correct price with tokenAmount = 340282366920938463463374607431768211456 and
          quoteTokenAmount = 2
        ✓ Calculates correct price with tokenAmount = 2 and quoteTokenAmount =
          340282366920938463463374607431768211456
    token > quoteToken
        ✓ Calculates correct price with tokenAmount = 340282366920938463463374607431768211456 and
          quoteTokenAmount = 2
        ✓ Calculates correct price with tokenAmount = 2 and quoteTokenAmount =
          340282366920938463463374607431768211456
UniswapV3PriceAccumulator#computeWholeUnitAmount
        ✓ Should verify that a token with 0 decimals has a whole unit amount of 1
        ✓ Should verify that a token with 1 decimals has a whole unit amount of 10
        ✓ Should verify that a token with 6 decimals has a whole unit amount of 1000000
        ✓ hould verify that a token with 18 decimals has a whole unit amount of 1000000000000000000
UniswapV3PriceAccumulator#computeAddress
        ✓ Reverts when token0 == token1
        ✓ Reverts when token0 > token1
UniswapV3PriceAccumulator
    UniswapV3PriceAccumulator#canUpdate
    Can't update when
        ✓ token = address(0)
        ✓ token = quoteToken
        ✓ The pool doesn't exist
        ✓ he pool has no liquidity
    Can update when
        ✓ The pool exists and has liquidity
    UniswapV3PriceAccumulator#fetchPrice
    Should revert when
        ✓ No pools exist
        ✓ The pools have no liquidity
        ✓ token = address(0)
        ✓ token = quoteToken
    token < quoteToken
    token decimals = 6, quote token decimals = 18
        ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000000 and quoteTokenAmour
          = 1000000000000000000000
        ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000000 and
          quoteTokenAmount = 1000000000000000000000
        ✓ fetchPrice(token) = 100000 with tokenAmount = 1000000000000000000000 and quoteTokenAmour
          = 1000000000000000000000
        ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000000 and quoteTokenAmour
          = 5000000000000000000000
```

```
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 0 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 6, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ etchPrice(token) = 10000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 0 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
  token decimals = 18, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
```

```
token > quoteToken
token decimals = 6, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 0 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
token decimals = 18, quote token decimals = 18
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 3000000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 5000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 5000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
token decimals = 6, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 10000000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000 with tokenAmount = 1000000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666 with tokenAmount = 3000000000000000000000 and quoteTokenAmour
      = 500000000000000000000
    ✓ fetchPrice(token) = 600000 with tokenAmount = 5000000000000000000000 and quoteTokenAmount
      = 300000000000000000000
    ✓ fetchPrice(token) = 0 with tokenAmount = 5000000000000000000000000000 and quoteTokenAmour
      = 300000000000000000000
token decimals = 18, quote token decimals = 6
    ✓ fetchPrice(token) = 1000000000000000000 with tokenAmount = 1000000000000000000000 and quo
```

```
    ✓ fetchPrice(token) = 10000000000000000000 with tokenAmount = 100000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 100000000000000000 with tokenAmount = 100000000000000000000 and
      quoteTokenAmount = 100000000000000000000
    ✓ fetchPrice(token) = 1666666666666666666 with tokenAmount = 300000000000000000000 and
      quoteTokenAmount = 500000000000000000000
    ✓ fetchPrice(token) = 600000000000000000 with tokenAmount = 500000000000000000000 and
      quoteTokenAmount = 300000000000000000000
    ✓ fetchPrice(token) = 600000000000 with tokenAmount = 500000000000000000000000000000 and
      quoteTokenAmount = 300000000000000000000
SimpleQuotationMetadata
  Using USDC
    ✓ quoteTokenName() = "USD Coin"
    ✓ quoteTokenSymbol() = "USDC"
    ✓ quoteTokenDecimals() = 6
  Using MKR
    ✓ uoteTokenName() = "Maker"
    ✓ quoteTokenSymbol() = "MKR"
    ✓ quoteTokenDecimals() = 18
  Using a random contract address
    ✓ quoteTokenName() = ""
    ✓ quoteTokenSymbol() = ""
    ✓ quoteTokenDecimals() = 18
ManagedCurveLiquidityAccumulator#update
  Only accounts with oracle updater role can update
    ✓ Accounts with oracle updater role can update (336ms)
    ✓ Accounts without oracle updater role cannot update (371ms)
  Smart contracts can't update
    ✓ Can't update in the constructor (486ms)
    ✓ Can't update in a function call (395ms)
  All accounts can update
    ✓ Accounts with oracle updater role can update (211ms)
    ✓ Accounts without oracle updater role can update (289ms)
ManagedCurveLiquidityAccumulator#supportsInterface(interfaceId)
    ✓ Should support IAccessControl
    ✓ Should support IAccessControlEnumerable
ManagedCurvePriceAccumulator#update
  Only accounts with oracle updater role can update
    ✓ Accounts with oracle updater role can update (287ms)
    ✓ Accounts without oracle updater role cannot update (230ms)
  Smart contracts can't update
    ✓ Can't update in the constructor (295ms)
    ✓ Can't update in a function call (348ms)
```

```
    All accounts can update
      ✓ Accounts with oracle updater role can update (164ms)
      ✓ Accounts without oracle updater role can update (278ms)
ManagedCurvePriceAccumulator#supportsInterface(interfaceId)
      ✓ Should support IAccessControl
      ✓ Should support IAccessControlEnumerable
ManagedUniswapV2LiquidityAccumulator#update
    Only accounts with oracle updater role can update
      ✓ Accounts with oracle updater role can update (723ms)
      ✓ Accounts without oracle updater role cannot update (214ms)
    Smart contracts can't update
      ✓ Can't update in the constructor (267ms)
      ✓ Can't update in a function call (391ms)
    All accounts can update
      ✓ Accounts with oracle updater role can update (325ms)
      ✓ Accounts without oracle updater role can update (306ms)
ManagedUniswapV2LiquidityAccumulator#supportsInterface(interfaceId)
      ✓ Should support IAccessControl
      ✓ Should support IAccessControlEnumerable
ManagedUniswapV2PriceAccumulator#update
    Only accounts with oracle updater role can update
      ✓ Accounts with oracle updater role can update (212ms)
      ✓ Accounts without oracle updater role cannot update (138ms)
    Smart contracts can't update
      ✓ Can't update in the constructor (265ms)
      ✓ Can't update in a function call (415ms)
    All accounts can update
      ✓ Accounts with oracle updater role can update (211ms)
      ✓ Accounts without oracle updater role can update (256ms)
ManagedUniswapV2PriceAccumulator#supportsInterface(interfaceId)
      ✓ Should support IAccessControl
      ✓ Should support IAccessControlEnumerable
ManagedUniswapV3LiquidityAccumulator#update
    Only accounts with oracle updater role can update
      ✓ Accounts with oracle updater role can update (1978ms)
      ✓ Accounts without oracle updater role cannot update (219ms)
    Smart contracts can't update
      ✓ Can't update in the constructor (358ms)
      ✓ Can't update in a function call (440ms)
    All accounts can update
      ✓ Accounts with oracle updater role can update (330ms)
      ✓ Accounts without oracle updater role can update (331ms)
```

```
ManagedUniswapV3LiquidityAccumulator#supportsInterface(interfaceId)
    ✓ Should support IAccessControl
    ✓ Should support IAccessControlEnumerable
ManagedUniswapV3PriceAccumulator#update
  Only accounts with oracle updater role can update
    ✓ Accounts with oracle updater role can update (493ms)
    ✓ Accounts without oracle updater role cannot update (148ms)
  Smart contracts can't update
    ✓ Can't update in the constructor (297ms)
    ✓ Can't update in a function call (371ms)
  All accounts can update
    ✓ Accounts with oracle updater role can update (329ms)
    ✓ Accounts without oracle updater role can update (349ms)
ManagedUniswapV3PriceAccumulator#supportsInterface(interfaceId)
    ✓ Should support IAccessControl
    ✓ Should support IAccessControlEnumerable
ManagedPeriodicAccumulationOracle#update
  Only accounts with oracle updater role can update
    ✓ Accounts with oracle updater role can update (168ms)
    ✓ Accounts without oracle updater role cannot update (59ms)
  All accounts can update
    ✓ Accounts with oracle updater role can update (140ms)
    ✓ Accounts without oracle updater role can update (147ms)
ManagedPeriodicAccumulationOracle#supportsInterface(interfaceId)
    ✓ Should support IAccessControl
    ✓ Should support IAccessControlEnumerable
  1328 passing (26m)
```

FILE	% STMTS	% BRANCH	% FUNCS
CurveLiquidityAccumulator.sol	100	100	100
CurvePriceAccumulator.sol	100	100	100
UniswapV2LiquidityAccumulator.sol	100	100	100
UniswapV2PriceAccumulator.sol	100	100	100
UniswapV3LiquidityAccumulator.sol	100	100	100
UniswapV3PriceAccumulator.sol	100	100	100
AbstractAccumulator.sol	100	100	100
LiquidityAccumulator.sol	100	100	100
PriceAccumulator.sol	100	100	100
AccumulationLibrary.sol	100	100	100
ObservationLibrary.sol	100	100	100
AbstractOracle.sol	100	100	100
AggregatedOracle.sol	100	100	100
PeriodicAccumulationOracle.sol	100	100	100
PeriodicOracle.sol	100	100	100
ManagedCurveLiquidity Accumulator.sol	100	100	100
ManagedCurvePrice Accumulator.sol	100	100	100
ManagedUniswapV2Liq uidityAccumulator.sol	100	100	100
ManagedUniswapV2 PriceAccumulator.sol	100	100	100
ManagedUniswapV3Liq uidityAccumulator.sol	100	100	100
ManagedUniswapV3 PriceAccumulator.sol	100	100	100
ManagedPeriodicAccum ulationOracle.sol	100	100	100
All files	**100**	**100**	**100**

We are grateful to have been given the opportunity to work with the Adrastia team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Adrastia team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.