Task list for addressing issues from the v1.1 Audit Report

Below is the list of tasks to fix the issues identified in the **v1.1** audit report. Each task corresponds to an issue in the report and includes the issue identifier.

IONBridgeRouter Contract

Medium Severity

- [ ] **M01:** Add reentrancy guards to `burn()` and `voteForMinting()` functions in `IONBridgeRouter.sol` by:
  - Inheriting from OpenZeppelin's `ReentrancyGuard` contract.
  - Applying the `nonReentrant` modifier to the `burn()` and `voteForMinting()` functions.

Low Severity

- [ ] **L01:** Add input validation in `IONBridgeRouter.sol` constructor to ensure external addresses are not zero:
  - Add `require` statements to check that `_iceV1`, `_iceV2`, `_bridge`, and `_ionSwap` are not zero addresses.
  - Provide descriptive error messages or custom errors for each check.
- [ ] **L02:** Implement events for critical state-changing functions in `IONBridgeRouter.sol`:
  - Define events `TokensBurned` and `TokensMinted`.
  - Emit `TokensBurned` in `burn()` function upon successful execution.
  - Emit `TokensMinted` in `voteForMinting()` function upon successful execution.

Informational

- [ ] **I01:** Update `_swapIceV1ToV2()` and `_swapIceV2ToV1()` in `IONBridgeRouter.sol` to use safe approval patterns:
  - Use `safeIncreaseAllowance`, `safeDecreaseAllowance`, or set allowance to zero before setting a new value.
  - Replace direct calls to `approve` with safe functions from OpenZeppelin's `SafeERC20` library.
- [ ] **I02:** Use custom errors instead of generic `require` statements in `IONBridgeRouter.sol`:
  - Define custom errors for checks in `burn()` and `voteForMinting()`, such as `error InvalidAmount()` and `error UnauthorizedReceiver()`.
  - Replace `require` statements with `if` conditions that `revert` with the custom errors.
- [ ] **I03:** Change visibility of internal functions in `IONBridgeRouter.sol` to optimize gas usage:
  - Change the visibility of `_swapIceV1ToV2()` and `_swapIceV2ToV1()` from `internal` to `private` if they are not intended for use in derived contracts.

IONSwap Contract

Low Severity

- [-] **L01:** Address potential precision loss in token exchange calculations in `IONSwap.sol`:
  - Review and adjust calculations in `getPooledAmountOut()` and `getOtherAmountOut()` to handle decimal precision more accurately.
  - Consider using a library or implementing fixed-point arithmetic with rounding options.
    Mitigation: Will not fix. The precision loss could happen only for extremely minor amounts, which is not relevant for the tokenomics.

Informational

- [x] **I01:** Optimize function visibility in `IONSwap.sol` to save gas:
  - Change the visibility of `getPooledAmountOut()` and `getOtherAmountOut()` from `public` to `external` since they are not called internally.
- [x] **I02:** Add zero address check for `_receiver` parameter in `withdrawLiquidity()` function in `IONSwap.sol`:
  - Include a check to ensure `_receiver` is not the zero address.
  - Define and use a custom error, e.g., `error InvalidReceiverAddress();`.
- [x] **I03:** Use descriptive custom errors in `IONSwap.sol` constructor validations:
  - Ensure all custom errors are properly defined and provide clear messages.
  - Replace `revert` statements with custom errors for input validations in the constructor.
