- 2 Removals + 2 Additions

```solidity
// SPDX-License-Identifier: AGPL-3.0-or-later

pragma solidity 0.7.5;

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    function add32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    function sub32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }

```

```solidity
// SPDX-License-Identifier: AGPL-3.0-or-later

pragma solidity 0.7.5;

library LowGasSafeMath {
    /// @notice Returns x + y, reverts if sum overflows uint256
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x);
    }

    function add32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x + y) >= x);
    }

    /// @notice Returns x - y, reverts if underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x);
    }

    function sub32(uint32 x, uint32 y) internal pure returns (uint32 z) {
        require((z = x - y) <= x);
    }

    /// @notice Returns x * y, reverts if overflows
    /// @param x The multiplicand
    /// @param y The multiplier
    /// @return z The product of x and y
    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(x == 0 || (z = x * y) / x == y);
    }

    /// @notice Returns x + y, reverts if overflows or underflows
    /// @param x The augend
    /// @param y The addend
    /// @return z The sum of x and y
    function add(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x + y) >= x == (y >= 0));
    }

    /// @notice Returns x - y, reverts if overflows or underflows
    /// @param x The minuend
    /// @param y The subtrahend
    /// @return z The difference of x and y
    function sub(int256 x, int256 y) internal pure returns (int256 z) {
        require((z = x - y) <= x == (y >= 0));
    }

    function div(uint256 x, uint256 y) internal pure returns(uint256 z){
        require(y > 0);
        z=x/y;
    }
}

interface IERC20 {

    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns (uint256);

    function transfer(address recipient, uint256 amount) external returns (bool);

    function allowance(address owner, address spender) external view returns (uint256);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library Address {

    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    function functionCall(
```

```
    function div(uint256 x, uint256 y) internal pure returns(uint256 z){
        require(y > 0);
        z=x/y;
    }
}

interface IERC20 {

    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns (uint256);

    function transfer(address recipient, uint256 amount) external returns (bool);

    function allowance(address owner, address spender) external view returns (uint256);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library Address {

    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    function functionCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _functionCallWithValue(
        address target,
        bytes memory data,
        uint256 weiValue,
        string memory errorMessage
    ) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
```

```
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _functionCallWithValue(
        address target,
        bytes memory data,
        uint256 weiValue,
        string memory errorMessage
    ) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            if (returndata.length > 0) {
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

    function addressToString(address _address) internal pure returns(string memory) {
        bytes32 _bytes = bytes32(uint256(_address));
```

```
    }

    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    function _verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            if (returndata.length > 0) {
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }

    function addressToString(address _address) internal pure returns(string memory) {
        bytes32 _bytes = bytes32(uint256(_address));
        bytes memory HEX = "0123456789abcdef";
        bytes memory _addr = new bytes(42);

        _addr[0] = '0';
        _addr[1] = 'x';

        for(uint256 i = 0; i < 20; i++) {
            _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
            _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
        }

        return string(_addr);

    }
}

contract OwnableData {
    address public owner;
    address public pendingOwner;
}

contract Ownable is OwnableData {
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /// @notice `owner` defaults to msg.sender on construction.
    constructor() {
        owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /// @notice Transfers ownership to `newOwner`. Either directly or claimable by the new pending owner.
    /// Can only be invoked by the current `owner`.
    /// @param newOwner Address of the new owner.
    /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
    /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
    function transferOwnership(
        address newOwner,
        bool direct,
        bool renounce
    ) public onlyOwner {
        if (direct) {
            // Checks
            require(newOwner != address(0) || renounce, "Ownable: zero address");

            // Effects
            emit OwnershipTransferred(owner, newOwner);
            owner = newOwner;
            pendingOwner = address(0);
        } else {
            // Effects
            pendingOwner = newOwner;
        }
    }

    /// @notice Needs to be called by `pendingOwner` to claim ownership.
    function claimOwnership() public {
```

```
        bytes memory HEX = "0123456789abcdef";
        bytes memory _addr = new bytes(42);

        _addr[0] = '0';
        _addr[1] = 'x';

        for(uint256 i = 0; i < 20; i++) {
            _addr[2+i*2] = HEX[uint8(_bytes[i + 12] >> 4)];
            _addr[3+i*2] = HEX[uint8(_bytes[i + 12] & 0x0f)];
        }

        return string(_addr);

    }
}

contract OwnableData {
    address public owner;
    address public pendingOwner;
}

contract Ownable is OwnableData {
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /// @notice `owner` defaults to msg.sender on construction.
    constructor() {
        owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /// @notice Transfers ownership to `newOwner`. Either directly or claimable by the new pending owner.
    /// Can only be invoked by the current `owner`.
    /// @param newOwner Address of the new owner.
    /// @param direct True if `newOwner` should be set immediately. False if `newOwner` needs to use `claimOwnership`.
    /// @param renounce Allows the `newOwner` to be `address(0)` if `direct` and `renounce` is True. Has no effect otherwise.
    function transferOwnership(
        address newOwner,
        bool direct,
        bool renounce
    ) public onlyOwner {
        if (direct) {
            // Checks
            require(newOwner != address(0) || renounce, "Ownable: zero address");

            // Effects
            emit OwnershipTransferred(owner, newOwner);
            owner = newOwner;
            pendingOwner = address(0);
        } else {
            // Effects
            pendingOwner = newOwner;
        }
    }

    /// @notice Needs to be called by `pendingOwner` to claim ownership.
    function claimOwnership() public {
        address _pendingOwner = pendingOwner;

        // Checks
        require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");

        // Effects
        emit OwnershipTransferred(owner, _pendingOwner);
        owner = _pendingOwner;
        pendingOwner = address(0);
    }

    /// @notice Only allows the `owner` to execute the function.
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the owner");
        _;
    }
}

interface ITreasury {
    function mintRewards( address _recipient, uint _amount ) external;
}

contract Distributor is Ownable {
    using LowGasSafeMath for uint;
    using LowGasSafeMath for uint32;



    /* ====== VARIABLES ====== */

    IERC20 public immutable TIME;
    ITreasury public immutable treasury;

    uint32 public immutable epochLength;
    uint32 public nextEpochTime;

    mapping( uint => Adjust ) public adjustments;

    event LogDistribute(address indexed recipient, uint amount);
    event LogAdjust(uint initialRate, uint currentRate, uint targetRate);
    event LogAddRecipient(address indexed recipient, uint rate);
    event LogRemoveRecipient(address indexed recipient);

    /* ====== STRUCTS ====== */

    struct Info {
        uint rate; // in ten-thousandths ( 5000 = 0.5% )
        address recipient;
    }
    Info[] public info;

    struct Adjust {
        bool add;
        uint rate;
        uint target;
    }



```

```
        address _pendingOwner = pendingOwner;

        // Checks
        require(msg.sender == _pendingOwner, "Ownable: caller != pending owner");

        // Effects
        emit OwnershipTransferred(owner, _pendingOwner);
        owner = _pendingOwner;
        pendingOwner = address(0);
    }

    /// @notice Only allows the `owner` to execute the function.
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the owner");
        _;
    }
}

interface ITreasury {
    function mintRewards( address _recipient, uint _amount ) external;
}

contract Distributor is Ownable {
    using LowGasSafeMath for uint;
    using LowGasSafeMath for uint32;



    /* ====== VARIABLES ====== */

    IERC20 public immutable TIME;
    ITreasury public immutable treasury;

    uint32 public immutable epochLength;
    uint32 public nextEpochTime;

    mapping( uint => Adjust ) public adjustments;

    event LogDistribute(address indexed recipient, uint amount);
    event LogAdjust(uint initialRate, uint currentRate, uint targetRate);
    event LogAddRecipient(address indexed recipient, uint rate);
    event LogRemoveRecipient(address indexed recipient);

    /* ====== STRUCTS ====== */

    struct Info {
        uint rate; // in ten-thousandths ( 5000 = 0.5% )
        address recipient;
    }
    Info[] public info;

    struct Adjust {
        bool add;
        uint rate;
        uint target;
    }



    /* ====== CONSTRUCTOR ====== */

    constructor( address _treasury, address _time, uint32 _epochLength, uint32 _nextEpochTime ) {
        require( _treasury != address(0) );
        treasury = ITreasury(_treasury);
        require( _time != address(0) );
        TIME = IERC20(_time);
        epochLength = _epochLength;
        nextEpochTime = _nextEpochTime;
    }



    /* ====== PUBLIC FUNCTIONS ====== */

    /**
        @notice send epoch reward to staking contract
     */
    function distribute() external returns ( bool ) {
        if ( nextEpochTime <= uint32(block.timestamp) ) {
            nextEpochTime = nextEpochTime.add32( epochLength ); // set next epoch time

            // distribute rewards to each recipient
            for ( uint i = 0; i < info.length; i++ ) {
                if ( info[ i ].rate > 0 ) {
                    treasury.mintRewards( // mint and send from treasury
                        info[ i ].recipient,
                        nextRewardAt( info[ i ].rate )
                    );
                    adjust( i ); // check for adjustment
                }
                emit LogDistribute(info[ i ].recipient, nextRewardAt( info[ i ].rate ));
            }
            return true;
        } else {
            return false;
        }
    }



    /* ====== INTERNAL FUNCTIONS ====== */

    /**
        @notice increment reward rate for collector
     */
    function adjust( uint _index ) internal {
        Adjust memory adjustment = adjustments[ _index ];
        if ( adjustment.rate != 0 ) {
            uint initial = info[ _index ].rate;
            uint rate = initial;
            if ( adjustment.add ) { // if rate should increase
                rate = rate.add( adjustment.rate ); // raise rate
                if ( rate >= adjustment.target ) { // if target met
```

```
    /* ====== CONSTRUCTOR ====== */

    constructor( address _treasury, address _time, uint32 _epochLength, uint32 _nextEpochTime ) {
        require( _treasury != address(0) );
        treasury = ITreasury(_treasury);
        require( _time != address(0) );
        TIME = IERC20(_time);
        epochLength = _epochLength;
        nextEpochTime = _nextEpochTime;
    }



    /* ====== PUBLIC FUNCTIONS ====== */

    /**
        @notice send epoch reward to staking contract
     */
    function distribute() external returns ( bool ) {
        if ( nextEpochTime <= uint32(block.timestamp) ) {
            nextEpochTime = nextEpochTime.add32( epochLength ); // set next epoch time

            // distribute rewards to each recipient
            for ( uint i = 0; i < info.length; i++ ) {
                if ( info[ i ].rate > 0 ) {
                    treasury.mintRewards( // mint and send from treasury
                        info[ i ].recipient,
                        nextRewardAt( info[ i ].rate )
                    );
                    adjust( i ); // check for adjustment
                }
                emit LogDistribute(info[ i ].recipient, nextRewardAt( info[ i ].rate ));
            }
            return true;
        } else {
            return false;
        }
    }



    /* ====== INTERNAL FUNCTIONS ====== */

    /**
        @notice increment reward rate for collector
     */
    function adjust( uint _index ) internal {
        Adjust memory adjustment = adjustments[ _index ];
        if ( adjustment.rate != 0 ) {
            uint initial = info[ _index ].rate;
            uint rate = initial;
            if ( adjustment.add ) { // if rate should increase
                rate = rate.add( adjustment.rate ); // raise rate
                if ( rate >= adjustment.target ) { // if target met
```

```
                    rate = adjustment.target;
                    delete adjustments[ _index ];
                }
            } else { // if rate should decrease
                rate = rate.sub( adjustment.rate ); // lower rate
                if ( rate <= adjustment.target ) { // if target met
                    rate = adjustment.target;
                    delete adjustments[ _index ];
                }
            }
            info[ _index ].rate = rate;
            emit LogAdjust(initial, rate, adjustment.target);
        }
    }



    /* ====== VIEW FUNCTIONS ====== */

    /**
        @notice view function for next reward at given rate
        @param _rate uint
        @return uint
     */
    function nextRewardAt( uint _rate ) public view returns ( uint ) {
        return TIME.totalSupply().mul( _rate ).div( 1000000 );
    }

    /**
        @notice view function for next reward for specified address
        @param _recipient address
        @return uint
     */
    function nextRewardFor( address _recipient ) external view returns ( uint ) {
        uint reward;
        for ( uint i = 0; i < info.length; i++ ) {
            if ( info[ i ].recipient == _recipient ) {
                reward = nextRewardAt( info[ i ].rate );
            }
        }
        return reward;
    }



    /* ====== POLICY FUNCTIONS ====== */

    /**
        @notice adds recipient for distributions
        @param _recipient address
        @param _rewardRate uint
     */
    function addRecipient( address _recipient, uint _rewardRate ) external onlyOwner {
        require( _recipient != address(0), "IA" );
        require(_rewardRate <= 5000, "Too high reward rate");
```

```
                    rate = adjustment.target;
                    delete adjustments[ _index ];
                }
            } else { // if rate should decrease
                rate = rate.sub( adjustment.rate ); // lower rate
                if ( rate <= adjustment.target ) { // if target met
                    rate = adjustment.target;
                    delete adjustments[ _index ];
                }
            }
            info[ _index ].rate = rate;
            emit LogAdjust(initial, rate, adjustment.target);
        }
    }



    /* ====== VIEW FUNCTIONS ====== */

    /**
        @notice view function for next reward at given rate
        @param _rate uint
        @return uint
     */
    function nextRewardAt( uint _rate ) public view returns ( uint ) {
        return TIME.totalSupply().mul( _rate ).div( 1000000 );
    }

    /**
        @notice view function for next reward for specified address
        @param _recipient address
        @return uint
     */
    function nextRewardFor( address _recipient ) external view returns ( uint ) {
        uint reward;
        for ( uint i = 0; i < info.length; i++ ) {
            if ( info[ i ].recipient == _recipient ) {
                reward = nextRewardAt( info[ i ].rate );
            }
        }
        return reward;
    }



    /* ====== POLICY FUNCTIONS ====== */

    /**
        @notice adds recipient for distributions
        @param _recipient address
        @param _rewardRate uint
     */
    function addRecipient( address _recipient, uint _rewardRate ) external onlyOwner {
        require( _recipient != address(0), "IA" );
        require(_rewardRate <= 50000, "Too high reward rate");
        require(info.length <= 4, "limit recipients max to 5");
        info.push( Info({
            recipient: _recipient,
            rate: _rewardRate
        }));
        emit LogAddRecipient(_recipient, _rewardRate);
    }

    /**
        @notice removes recipient for distributions
        @param _index uint
        @param _recipient address
     */
    function removeRecipient( uint _index, address _recipient ) external onlyOwner {
        require( _recipient == info[ _index ].recipient, "NA" );
        info[_index] = info[info.length-1];
        adjustments[_index] = adjustments[ info.length-1 ];
        info.pop();
        delete adjustments[ info.length-1 ];
        emit LogRemoveRecipient(_recipient);
    }

    /**
        @notice set adjustment info for a collector's reward rate
        @param _index uint
        @param _add bool
        @param _rate uint
        @param _target uint
     */
    function setAdjustment( uint _index, bool _add, uint _rate, uint _target ) external onlyOwner {
        require(_target <= 5000, "Too high reward rate");
        adjustments[ _index ] = Adjust({
            add: _add,
            rate: _rate,
            target: _target
        });
    }
}
```

```
        require(info.length <= 4, "limit recipients max to 5");
        info.push( Info({
            recipient: _recipient,
            rate: _rewardRate
        }));
        emit LogAddRecipient(_recipient, _rewardRate);
    }

    /**
        @notice removes recipient for distributions
        @param _index uint
        @param _recipient address
     */
    function removeRecipient( uint _index, address _recipient ) external onlyOwner {
        require( _recipient == info[ _index ].recipient, "NA" );
        info[_index] = info[info.length-1];
        adjustments[_index] = adjustments[ info.length-1 ];
        info.pop();
        delete adjustments[ info.length-1 ];
        emit LogRemoveRecipient(_recipient);
    }

    /**
        @notice set adjustment info for a collector's reward rate
        @param _index uint
        @param _add bool
        @param _rate uint
        @param _target uint
     */
    function setAdjustment( uint _index, bool _add, uint _rate, uint _target ) external onlyOwner {
        require(_target <= 50000, "Too high reward rate");
        adjustments[ _index ] = Adjust({
            add: _add,
            rate: _rate,
            target: _target
        });
    }
}
```