function rent(address[] _nft, uint256[] _tokenId,
uint2[] _id, uint16[] _rentDuration)

returnIt(address[] nft, uint[] tokenId, uint[] id)

claimCollateral (↑)

stopLending (↑)

```
/**
 * @dev determines what nft standrad we are dealing with
 */
function _safeTransfer(
    address _from,
    address _to,
    address _nft,
    uint256 _tokenId
) private {
    bool isERC721 = _isERC721(_nft);
    bool isERC1155 = _isERC1155(_nft);

    if (isERC721) {
        IERC721(_nft).transferFrom(_from, _to, _tokenId);
    } else if (isERC1155) {
        // TODO: change the amount
        IERC1155(_nft).safeTransferFrom(_from, _to, _tokenId, 1, "");
    } else {
        revert("unsupported _from");
    }
}
```

lend (address[] nft, uint[] tokenId, uint16[] maxRentDur,
bytes4[] dailyRentPrice, bytes4[] nftPrice).

Implement a contract that is a map?
Enumerable Map? And then pass meta into function that is sufficient to reconstruct the map.

1155
nftAddress → tokenId #1 → amount
→ tokenId #2 → amount

(nftAddress, -11-, -11-)
implies 1155

id #1, id #2, id #3 ← ascending order

only #1, #2, #3

batchTransfer

! Properties: !
Time complexity O(n)
Space complexity O(1)

two pointer approach
last, curr

Define the set of NFTs as AA, AB, BA, AC, BC, BD, BE
- where the ones that start with "A" are 721s and the ones that start with "B" are 1155s
- we distinguish between 721s and 1155s via the amount array.
If the entry is 0, then it's 721. Else 1155

lastIx = first in nfts
for nft in nfts
currIx = nft
if (curr == last) continue
# curr != last
# batch transfer
# last Ix - start of same address
# curr Ix - end of same address
same logic as before.

Reads the loop until it finds a new value
BA, BB, AA

assign to curr Ix after conditional
evaluated at indices

evaluate with loop in first. This is so except that curr Ix doesn't need a increment

Assumptions:
- all nfts in the array are unique
- amounts value 0 denotes 721, else 1155