Task C (7 Marks)

Now, create a function `move_kitty` that takes in a move and house object, and returns a new house object with a kitty moved (if the move is valid). If the move is not valid, the game should return the house object unchanged.

```
function move_kitty (move, house){
var source_room = list_ref(house, get_source(move) - 1);
var destination_room = list_ref(house, get_destination(move) - 1);
var kitty = head(source_room);
if(is_empty_list(destination_room) || kitty <
head(destination_room)){
        return build_list(3, function(index){
        if(index === (get_source(move) - 1)){
            return tail(list_ref(house, index));
        } else if(index === (get_destination(move) - 1)){
            return pair(kitty, list_ref(house, index));
        } else{
            return list_ref(house, index);
        }
    });
} else{
    return house;
}
}
```

Task D (5 Marks)

Now, create a function `apply_moves` that takes in a list of moves and house object, and returns a new house object with each move (starting from the last move until the first) applied to the input house

```
function apply_moves (moves, house){
    return accumulate(move_kitty, house, moves);
}
```

Task E (10 Marks)

It's time to figure out how to move the kitties! Create a function `find_moves` that takes in a house object, an integer `num`, and two room indices (1 and 2), that returns a list of moves such that applying the list of moves (from the last move in the list to the first) yields a house with the *num* kitties in room1 moved to room2.

```
function find_moves(game, num, room1, room2){
if(num === 1){
    return list(make_move(room1, room2));
} else{
    var other_room = 6 - (room1 + room2);
    var first_list = find_moves(game, num - 1, room1, other_room);
    var big_move = make_move(room1, room2);
    var second_list =find_moves(game, num - 1, other_room, room2);
    return append(second_list, pair(big_move, first_list));
}
}
```

Task H (2 Marks)

What is the runtime complexity of the function you defined in E? Give your answer in Big-O notation, to the tightest possible bound.

Time Complexity: $O(2^n)$