Question #5 (Very Hard)

Description:

Implement a class LRUCache that represents a Least Recently Used (LRU) cache with a maximum capacity. The cache should support two operations:

1. get(key): Gets the value of the key if the key exists in the cache, otherwise returns -1.
2. set(key, value): Sets or inserts the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting the new item.

The cache must operate in O(1) time complexity for both set and get operations.

Constraints:

- The cache should have a fixed size set at initialization.
- All keys and values are integers.

Use object oriented programming to solve this question.

Starting Code

```
class LRUCache {
    constructor(capacity) {
    }

    get(key) {
        // TODO: Implement the method to retrieve a value from the
cache.
    }

    set(key, value) {
        // TODO: Implement the method to add or update a value in the
cache.
    }
}
```

Sample Usage

```
// Usage Example
const cache = new LRUCache(2);
cache.set(1, 1); // Cache is {1=1}
cache.set(2, 2); // Cache is {1=1, 2=2}
console.log(cache.get(1)); // returns 1
cache.set(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
console.log(cache.get(2)); // returns -1 (not found)
cache.set(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
console.log(cache.get(1)); // returns -1 (not found)
console.log(cache.get(3)); // returns 3
console.log(cache.get(4)); // returns 4
```