1- LRU-Cache pro

I used an **OrderedDict** which is a subclass that remembers the order that keys were first inserted.

```
    # initialising capacity

    def __init__(self, capacity: int):

        self.cache = OrderedDict()

        self.capacity = capacity


    # Return the value of the key

    # that is queried in O(1) and return -1 if

    # don't find the key in out cache.

    # And also move the key to the end

    # to show that it was recently used.

    def get(self, key: int):

        if key not in self.cache:

            return -1

        else:

            self.cache.move_to_end(key)

            return self.cache[key]


       # Add key and move the key to the end to show that it was recently
used.

    # Check whether the length of

    # ordered dictionary has exceeded capacity,

    # If so remove the first key (least recently used)

    def put(self, key: int, value: int):

        self.cache[key] = value

        self.cache.move_to_end(key)
        if len(self.cache) > self.capacity:
            self.cache.popitem(last = False)
```

Time Complexity = O(1)

Space Complexity = O(n) n is the number of element we put in the cache.