981. Time Based Key-Value Store

Create a timebased key-value store class `TimeMap`, that supports two operations.

1. `set(string key, string value, int timestamp)`

- Stores the `key` and `value`, along with the given `timestamp`.

2. `get(string key, int timestamp)`

- Returns a value such that `set(key, value, timestamp_prev)` was called previously, with `timestamp_prev <= timestamp`.
- If there are multiple such values, it returns the one with the largest `timestamp_prev`.
- If there are no values, it returns the empty string (`""`).

Example 1:

Input: inputs = ["TimeMap","set","get","get","set","get","get"],
inputs = [[],["foo","bar",1],["foo",1],["foo",3],["foo","bar2",4],["foo",4],["foo",5]]
Output: [null,null,"bar","bar",null,"bar2","bar2"]
Explanation: TimeMap kv;
kv.set("foo", "bar", 1); // store the key "foo" and value "bar" along with timestamp = 1
kv.get("foo", 1); // output "bar"
kv.get("foo", 3); // output "bar" since there is no value corresponding to foo at
timestamp 3 and timestamp 2, then the only value is at timestamp 1 ie "bar"
kv.set("foo", "bar2", 4);
kv.get("foo", 4); // output "bar2"
kv.get("foo", 5); //output "bar2"

Example 2:

Input: inputs = ["TimeMap","set","set","get","get","get","get","get"],
inputs = [[],["love","high",10],["love","low",20],["love",5],["love",10],["love",15],["love",20],["love",25]]
Output: [null,null,null,"","high","high","low","low"]

Note:

1. All key/value strings are lowercase.
2. All key/value strings have length in the range `[1, 100]`
3. The `timestamps` for all `TimeMap.set` operations are strictly increasing.
4. `1 <= timestamp <= 10^7`

5. `TimeMap.set` and `TimeMap.get` functions will be called a total of `120000` times (combined) per test case.

```python
def __init__(self):
        """
        Initialize your data structure here.
        """
        self.hash = {}
        self.stamp = []
        self.hash2 = {}


    def set(self, key: str, value: str, timestamp: int) -> None:
        if key in self.hash:
            self.hash[key].append([value,timestamp])
        else:
            self.hash[key]=[[value,timestamp]]

    def get(self, key: str, timestamp: int) -> str:
        temp = self.hash[key]
        if temp[0][1]>timestamp:
            return ""
        if temp[-1][1]<timestamp:
            return temp[-1][0]
        # print(temp)
        stamp = self.binarySearch(temp,timestamp)
        if stamp:
            return stamp
        else:
            return ""
        # print(stamp)

    def binarySearch(self,arr,target):
        lo = 0
        hi = len(arr)-1
        while lo<=hi:
            mid = lo+(hi-lo)//2
            if arr[mid][1]==target:
                return arr[mid][0]
            elif arr[mid][1]>target:
                hi = mid-1
            else:
                lo = mid+1
        return arr[hi][0]
```

The binary search over target gives us either the target or the greatest element less than the target