911. Online Election
We will use two arrays to store time and winner at that time in order. Then when we query the winner in some time, we will use binary search to get index from time array, and get winner from winner array. TC is O(logn)

```
from bisect import *
class TopVotedCandidate:

    def __init__(self, persons: List[int], times: List[int]):
        self.votes_times = []
        self.votes_number = []
        max_index = None
        votes_count = [0] * (max(persons) + 1)
        for i, t in zip(persons, times):
          votes_count[i] += 1
          if max_index == None or votes_count[i] >= votes_count[max_index]:
            max_index = i
            self.votes_times.append(t)
            self.votes_number.append(i)

    def q(self, t: int) -> int:
        idx = bisect(self.votes_times, t)
        return self.votes_number[idx - 1]
```

846. Hand of Straights
We will use a hashmap to store our (element, present_time) key pair value. Then we will get an array which stores our sorted keys. We will iterate from left of keys by k elements. Once there is an element missing(its value is 0) or previous one is larger than the latter one we will return False. In the end, if nothing happens, we will return True. TC is O(n)

```
from collections import Counter, deque
class Solution:
    def isNStraightHand(self, hand: List[int], W: int) -> bool:
        if W == 1:
          return True
        if len(hand) % W != 0:
          return False

        counter = Counter(hand)
        counter_keys = deque(sorted(counter.keys()))
        while counter_keys:
          key = counter_keys.popleft()
          count = counter[key]
          key += 1
          for i in range(W - 1):
```

```
            if key not in counter or counter[key] < count or not counter_keys:
                return False
            counter[key] -= count
            if counter[key] == 0:
              if counter_keys[0] != key:
                return False
              counter_keys.popleft()
            key += 1
        return True
```

133. Clone Graph

We will use dfs to clone graph, we will use a dict to record all value's node we create, if there is an existing node, we will return it. Or we will create this new node and traverse its neighbors and append it into our new node's neighbors. TC is O(n)

```
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        graph_map = {}
        def dfs(node):
          if node.val in graph_map:
            return graph_map[node.val]
          new_node = Node(node.val, [])
          graph_map[node.val] = new_node
          for n in node.neighbors:
            new_node.neighbors.append(dfs(n))
          return new_node
        dfs(node)
        return graph_map[node.val]
```

400. Nth Digit

We will iterate from 1 to 10 to check whether n is in 1-9, 10-99, 100-999… Then we will find that number and that digit. TC is O(1)

```
class Solution:
    def findNthDigit(self, n: int) -> int:
        n -= 1
        for digits in range(1, 11):
          first = 10 ** (digits - 1)
          if n < 9 * first * digits:
            return int(str(first + n // digits)[n % digits])
          n -= 9 * first * digits
```

722. Remove Comments

We will use regex to replace those comments with '', we will join source using '\n' and then split it by '\n', in the end, we could get string's array without comments. TC is O(n)

```
import re
class Solution:
```

```
def removeComments(self, source: List[str]) -> List[str]:
    return filter(None, re.sub('//.*|/\*(.|\n)*?\*/', '', '\n'.join(source)).split('\n'))
```