428. Serialize and Deserialize N-ary Tree
We will use pre-order traverse to serialize the tree. We will use ‘#’ to split different branches. In the end, we would use blank to join our strings. When we want to deserialize, we will first get our elements by split by blank. Then we will create branch trees splitting by ‘#’. And then we could build our nodes recursively. TC is O(n)

```
"""
# Definition for a Node.
class Node(object):
    def __init__(self, val, children):
        self.val = val
        self.children = children
"""
from collections import deque
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: Node
        :rtype: str
        """
        serial = []

        def preOrder(node):
            if not node:
                return

            serial.append(str(node.val))
            for n in node.children:
                preOrder(n)
            serial.append('#')

        preOrder(root)
        return ' '.join(serial)

    def deserialize(self, data):
        """Decodes your encoded data to tree.

        :type data: str
        :rtype: Node
        """
        print(data)
        dq = deque(data.split())
```

```
        if not dq:
            return
        root = Node(int(dq.popleft()), [])

        def helper(node):
            while dq[0] != '#':
                child = Node(int(dq.popleft()), [])
                node.children.append(child)
                helper(child)
            dq.popleft()
        helper(root)
        return root
```

635. Design Log Storage System

We will use an array to store (id, timestamp) pair. Because there are only 300 pairs maximum. So each time when we will compare every pair's timestamp to check whether it's in the bound. TC is O(1), O(n)

```
class LogSystem:

    def __init__(self):
        self.log = []
        self.index = {'Year': 4, 'Month': 7, 'Day': 10, 'Hour': 13, 'Minute': 16, 'Second': 19}

    def put(self, id: int, timestamp: str) -> None:
        self.log.append((id, timestamp))

    def retrieve(self, s: str, e: str, gra: str) -> List[int]:
        s = s[:self.index[gra]]
        e = e[:self.index[gra]]

        return [id for id, t in self.log if s <= t[:self.index[gra]] <= e]
```

48. Rotate Image

We will change four coordinates at the same time. Also, we need to iterate through our coordinates without repeat. TC is O(n)

```
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        """
        Do not return anything, modify matrix in-place instead.
        """
        rows = len(matrix)
        length = rows - 1
```

```
        start = 0
        while length > 0:
            for d_i in range(length):
                cur = start + d_i
                matrix[cur][rows - start - 1], matrix[rows - start - 1][rows - cur - 1], matrix[rows - cur - 1][start], matrix[start][cur] = matrix[start][cur], matrix[cur][rows - start - 1], matrix[rows - start - 1][rows - cur - 1], matrix[rows - cur - 1][start]

            length -= 2
            start += 1
```

535. Encode and Decode TinyURL

We will use two dict to record url2code and code2url two-way key-value pairs. In encode, we will check whether longUrl is in url2code. If it is, we will return code directly. If not, we will create a random one and check whether we have used this code. If not, we will record the associated url and code in two dicts. For decode, we will return code2url(shortUrl) directly. TC O(1), O(1)

```
import random
class Codec:
    def __init__(self):
        self.url2code = {}
        self.code2url = {}
        self.code = 'abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ'

    def encode(self, longUrl):
        """Encodes a URL to a shortened URL.

        :type longUrl: str
        :rtype: str
        """
        if longUrl in self.url2code:
            return self.url2code[longUrl]
        while True:
            newUrl = 'http://tinyurl.com/' + ''.join([random.choice(self.code) for _ in range(6)])
            if newUrl not in self.code2url:
                self.code2url[newUrl] = longUrl
                self.url2code[longUrl] = newUrl
                return self.url2code[longUrl]

    def decode(self, shortUrl):
        """Decodes a shortened URL to its original URL.

        :type shortUrl: str
```

```
        :rtype: str
        """
        return self.code2url[shortUrl]
```

236. Lowest Common Ancestor of a Binary Tree

We will return an array including p.val or q.val each traverse. Then we will check whether left branch and right branch and current node could cover both q and p. If they could, we will return the current node. TC is O(n)

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') ->
'TreeNode':
        visited = {}
        self.node = None
        def traverse(node):
          if not node or self.node:
            return []
          left = traverse(node.left)
          right = traverse(node.right)
          temp = left + right + [node.val]
          if p.val in temp and q.val in temp:
            if not self.node:
              self.node = node
          if node == p or node == q:
            return [node.val] + left + right
          return left + right
        traverse(root)
        return self.node
```