42. Trapping Rain Water
We will scan from both sides and always add the water we could trap, then we will move lower layer to center. TC is O(n), SC is O(1)

```
class Solution:
    def trap(self, height: List[int]) -> int:
        left, right = 0, len(height) - 1
        max_water = 0
        cur_height = 0
        while left <= right:
          cur_height = max(cur_height, min(height[left], height[right]))
          if height[left] < height[right]:
            max_water += max(cur_height - height[left], 0)
            left += 1
          else:
            max_water += max(cur_height - height[right], 0)
            right -= 1
        return max_water
```

146. LRU Cache

```
class Node:
  def __init__(self, key, val):
    self.key = key
    self.val = val
    self.next = None
    self.prev = None

class LRUCache:

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.dummy = Node(0, 0)
        self.dummy.next = self.dummy
        self.dummy.prev = self.dummy
        self.map = {}

    def get(self, key: int) -> int:
        if key in self.map:
          node = self.map[key]
          node.prev.next = node.next
          node.next.prev = node.prev
          node.next = self.dummy.next
          self.dummy.next = node
          node.next.prev = node
          node.prev = self.dummy
```

```
            return node.val
        else:
            return -1

    def put(self, key: int, value: int) -> None:
        if key in self.map:
            node = self.map[key]
            node.prev.next = node.next
            node.next.prev = node.prev
            node.val = value
        else:
            if self.capacity > 0:
                self.capacity -= 1
            else:
                del_node = self.dummy.prev
                del_node.prev.next = self.dummy
                self.dummy.prev = del_node.prev
                del self.map[del_node.key]
            node = Node(key, value)
        self.map[key] = node
        node.next = self.dummy.next
        self.dummy.next = node
        node.next.prev = node
        node.prev = self.dummy




# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

151. Reverse Words in a String

```
class Solution:
    def reverseWords(self, s: str) -> str:
        return ' '.join(reversed(s.split()))
```

5. Longest Palindromic Substring

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        max_length = 0
```

```
    max_substring = ''
    for i, _ in enumerate(s):
      sub = self.helper(i, i + 1, s)
      if len(sub) > max_length:
        max_length = len(sub)
        max_substring = sub
      sub = self.helper(i - 1, i + 1, s)
      if len(sub) > max_length:
        max_length = len(sub)
        max_substring = sub
    return max_substring

  def helper(self, l, r, s):
    while l >= 0 and r < len(s):
      if s[l] == s[r]:
        l -= 1
        r += 1
      else:
        break

    return s[l + 1: r]
```

22. Generate Parentheses

```
class Solution:
  def generateParenthesis(self, n: int) -> List[str]:
      result = []
      self.helper('', 0, 0, n, result)
      return result
  def helper(self, cur, l, r, n, result):
    if l == r and l == n:
      result.append(cur)
      return
    elif l > n or r > n:
      return
    if l < n:
      self.helper(cur + '(', l + 1, r, n, result)
    if l > r:
      self.helper(cur + ')', l, r + 1, n, result)
```