1.
```
def min_length_after_removals(nums):
    left = 0
    right = len(nums) - 1
    pairs = 0
    while left < right:
        if nums[left] < nums[right]:
            pairs += 1
            left += 1
            right -= 1
        else:
            right -= 1
    return len(nums) - 2 * pairs
nums = [1, 2, 3, 4]
print(min_length_after_removals(nums))
```

2.
```
from typing import List, Optional
class TreeNode:
    def _init_(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def sortedArrayToBST(nums: List[int]) -> Optional[TreeNode]:
    if not nums:
        return None
    mid = len(nums) // 2
    root = TreeNode(nums[mid])
    root.left = sortedArrayToBST(nums[:mid])
    root.right = sortedArrayToBST(nums[mid + 1:])
    return root
nums = [-10, -3, 0, 5, 9]
root = sortedArrayToBST(nums)
```

```
from collections import deque
def print_tree(root: Optional[TreeNode]):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    while result and result[-1] is None:
        result.pop()
    return result
print(print_tree(root))
```

3.
```
def find_substrings(words):
    result = []
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break

    return result
words = ["mass", "as", "hero", "superhero"]
print(find_substrings(words))
```

4.
```
def wiggleSort(nums):
```

```
    nums.sort()
    half = len(nums[::2])
    nums[::2], nums[1::2] = nums[:half][::-1], nums[half:][::-1]
nums1 = [1, 5, 1, 1, 6, 4]
wiggleSort(nums1)
print(nums1)

nums2 = [1, 3, 2, 2, 3, 1]
wiggleSort(nums2)
print(nums2)
def isWiggleSorted(nums):
    for i in range(len(nums) - 1):
        if i % 2 == 0 and nums[i] >= nums[i + 1]:
            return False
        if i % 2 == 1 and nums[i] <= nums[i + 1]:
            return False
    return True
print(isWiggleSorted(nums1))
print(isWiggleSorted(nums2))

5. import heapq
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
    def __lt__(self, other):
        return self.val < other.val
def mergeKLists(lists):
    min_heap = []

    for l in lists:
        if l:
```

```
            heapq.heappush(min_heap, l)

    dummy = ListNode()
    current = dummy

    while min_heap:

        smallest_node = heapq.heappop(min_heap)
        current.next = smallest_node
        current = current.next
        if smallest_node.next:
            heapq.heappush(min_heap, smallest_node.next)

    return dummy.next
def to_linked_list(values):
    dummy = ListNode()
    current = dummy
    for value in values:
        current.next = ListNode(value)
        current = current.next
    return dummy.next
def to_list(node):
    result = []
    while node:
        result.append(node.val)
        node = node.next
    return result

lists = [
    to_linked_list([1, 4, 5]),
    to_linked_list([1, 3, 4]),
    to_linked_list([2, 6])
```

```
]

merged_list = mergeKLists(lists)
print(to_list(merged_list))  # Output: [1, 1, 2, 3, 4, 4, 5, 6]
```