1.
```
def find_substrings(words):
    substrings = []
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                substrings.append(words[i])
                break
    return substrings
words = ["mass","as","hero","superhero"]
output = find_substrings(words)
print(output)
```

2.
```
def wiggleSort(nums):
    nums.sort()
    n = len(nums)
    half = (n + 1) // 2
    left = nums[:half][::-1]
    right = nums[half:][::-1]
    nums[::2] = left
    nums[1::2] = right
    return nums
nums = [1, 5, 1, 1, 6, 4]
output = wiggleSort(nums)
print(output)
```

3.
```
from collections import deque

def updateMatrix(mat):
    if not mat or not mat[0]:
        return mat
```

```
    m, n = len(mat), len(mat[0])
    dist = [[float('inf')] * n for _ in range(m)]
    queue = deque()
    for i in range(m):
        for j in range(n):
            if mat[i][j] == 0:
                dist[i][j] = 0
                queue.append((i, j))

    directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]

    while queue:
        x, y = queue.popleft()
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < m and 0 <= new_y < n:
                if dist[new_x][new_y] > dist[x][y] + 1:
                    dist[new_x][new_y] = dist[x][y] + 1
                    queue.append((new_x, new_y))

    return dist

mat1 = [[0,0,0],[0,1,0],[0,0,0]]
output1 = updateMatrix(mat1)

mat2 = [[0,0,0],[0,1,0],[1,1,1]]
output2 = updateMatrix(mat2)
print(output2)
```

4.
```
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 i, lst in enumerate(lists):
        if lst:
            heapq.heappush(min_heap, (lst.val, i, lst))

    dummy = ListNode()
    current = dummy

    while min_heap:
        val, i, node = heapq.heappop(min_heap)

        current.next = ListNode(val)
        current = current.next

        if node.next:
            heapq.heappush(min_heap, (node.next.val, i, node.next))

    return dummy.next

def array_to_linked_list(arr):
    if not arr:
        return None
```

```
    head = ListNode(arr[0])
    current = head
    for value in arr[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

def linked_list_to_array(head):
    arr = []
    current = head
    while current:
        arr.append(current.val)
        current = current.next
    return arr

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

merged_head = mergeKLists(lists)
print(linked_list_to_array(merged_head))
```

5.
```
from bisect import bisect_right
from collections import defaultdict

def makeArrayIncreasing(arr1, arr2):
    arr2 = sorted(set(arr2))
    dp = {-1: 0}
```

```
    for num in arr1:
        new_dp = defaultdict(lambda: float('inf'))
        for key in dp:

            if num > key:
                new_dp[num] = min(new_dp[num], dp[key])

            idx = bisect_right(arr2, key)
            if idx < len(arr2):
                new_dp[arr2[idx]] = min(new_dp[arr2[idx]], dp[key] + 1)

        dp = new_dp

    if dp:
        return min(dp.values())
    else:
        return -1

arr1 = [1, 5, 3, 6, 7]
arr2 = [1, 3, 2, 4]
print(makeArrayIncreasing(arr1, arr2))
```