Stack and Queue

1.最小堆疊 (Min Stack)

```
class MinStack:

    def __init__(self):


    def push(self, val: int) -> None:


    def pop(self) -> None:


    def top(self) -> int:


    def getMin(self) -> int:



# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(val)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

設計一個支援以下操作的堆疊（stack）類別，並且每個操作都能在 **O(1)** 時間內完成：

- `MinStack()`：初始化堆疊物件。
- `push(int val)`：將元素 `val` 推入堆疊頂端。
- `pop()`：移除堆疊頂端的元素。

- `top()`：取得堆疊頂端的元素。
- `getMin()`：取得堆疊中目前的最小元素。

你必須確保上述所有函式的時間複雜度都為 **O(1)**。

輸入：

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
```

輸出：

```
[null,null,null,null,-3,null,0,-2]
```

說明：

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); # 回傳 -3
minStack.pop();
minStack.top();    # 回傳 0
minStack.getMin(); # 回傳 -2
```

2 字典序最小的字串

```
class Solution:
    def orderlyQueue(self, s: str, k: int) → str:
```

給定一個字串 `s` 和一個整數 `k`。

你可以重複執行以下操作任意次數：

> 從字串 s 的前 k 個字母中選擇一個字母，並將它移到字串的最後。

請回傳經過若干次操作後，能夠得到的 **字典序（lexicographically）最小** 的字串。

範例 1：

輸入：

`s = "cba", k = 1`

輸出：

`"acb"`

說明：

- 第一次操作：把第 1 個字元 `'c'` 移到最後 → `"bac"`
- 第二次操作：再把第 1 個字元 `'b'` 移到最後 → `"acb"`

範例 2：

輸入：

`s = "baaca", k = 3`

輸出：

`"aaabc"`

說明：

- 第一次操作：把第 1 個字元 `'b'` 移到最後 → `"aacab"`
- 第二次操作：把第 3 個字元 `'c'` 移到最後 → `"aaabc"`

3. Design Circular Queue

```
class MyCircularQueue:

    def __init__(self, k: int):
```

```
    def enQueue(self, value: int) -> bool:


    def deQueue(self) -> bool:


    def Front(self) -> int:


    def Rear(self) -> int:

# Your MyCircularQueue object will be instantiated and called as such:
# obj = MyCircularQueue(k)
# param_1 = obj.enQueue(value)
# param_2 = obj.deQueue()
# param_3 = obj.Front()
# param_4 = obj.Rear()
# param_5 = obj.isEmpty()
# param_6 = obj.isFull()
```

環狀佇列是一種遵循 **FIFO（先進先出）** 原則的線性資料結構，不同之處在於**最後一個位置會連回第一個位置**，形成一個「環」（又稱 **Ring Buffer**）。

環狀佇列的優點是可以重複利用前方空間。

在一般佇列中，一旦尾端滿了，就不能再加入新元素，即使前方有空位。

而使用環狀佇列，我們可以利用這些空位來儲存新的資料。

請實作以下類別：`MyCircularQueue`

- `MyCircularQueue(k)`：初始化一個大小為 `k` 的環狀佇列。
- `int Front()`：取得佇列的第一個元素；若佇列為空，回傳 `-1`。
- `int Rear()`：取得佇列的最後一個元素；若佇列為空，回傳 `-1`。
- `boolean enQueue(int value)`：將元素插入佇列尾端。若成功，回傳 `True`。
- `boolean deQueue()`：刪除佇列前端元素。若成功，回傳 `True`。

- `boolean isEmpty()`：檢查佇列是否為空。
- `boolean isFull()`：檢查佇列是否已滿。