TUẦN 7

Họ & Tên: Phạm Nguyễn Phương Duy
MSSV: 19110290

Sử dụng thuật toán Boyer-Moore Majority Vote

Bài 1

```python
def majority_element(arr):
    n = len(arr)
    count = [0] * 10 # Khởi tạo mảng count với kích thước 10
    element = [None] * 10 # Khởi tạo mảng element với kích thước 10

    for i in range(n):
        j = 0
        while j < 10:
            if element[j] == arr[i]: # Nếu phần tử hiện tại xuất hiện trong mảng element
                count[j] += 1 # Tăng giá trị tương ứng trong mảng count lên 1
                break
            elif count[j] == 0: # Nếu có vị trí nào trong mảng count có giá trị bằng 0
                element[j] = arr[i] # Gán giá trị của phần tử hiện tại vào vị trí tương ứng trong mảng element
                count[j] = 1 # Gán giá trị tương ứng trong mảng count bằng 1
                break
            j += 1
        else:
            for k in range(10):
                count[k] -= 1 # Giảm giá trị của tất cả các phần tử trong mảng count đi 1

    result = []
    for i in range(10):
        actual_count = sum(1 for x in arr if x == element[i]) # Tính số lần xuất hiện thực sự của phần tử trong mảng arr
        if actual_count > n/10: # Nếu số lần xuất hiện này lớn hơn n/10
            result.append(element[i]) # Thêm phần tử đó vào danh sách kết quả

    return result

# Test
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1, 3, 6, 9]
result = majority_element(arr)
if len(result) == 0:
    print("Không có phần tử lấn chiếm cấp độ 10")
else:
    print(f"Phần tử lấn chiếm cấp độ 10: {result}")
```

```
Phần tử lấn chiếm cấp độ 10: [1, 3, 6, 9]
```

2/ - Có thể có thuật toán có độ phức tạp O(n) để tìm ra phần tử “lấn chiếm cấp độ 10” trong một mảng. Một ví dụ là thuật toán Boyer-Moore Majority Vote.
Thuật toán này cho phép tìm ra phần tử xuất hiện nhiều hơn n/2 lần trong mảng với độ phức tạp O(n). Ý tưởng cơ bản của thuật toán này là duyệt qua từng phần tử trong mảng và đếm số lần xuất hiện của một phần tử bằng cách sử dụng một biến đếm. Nếu phần tử tiếp theo trong mảng bằng với phần tử hiện tại, chúng ta tăng biến đếm lên 1. Nếu không, chúng ta giảm biến đếm đi 1. Nếu biến đếm bằng 0, chúng ta cập nhật phần tử hiện tại và gán biến đếm bằng 1.

Bài 2

[27]

```python
import random

def merge_arrays(A, B):
    # Khởi tạo kết quả
    result = []

    # Khởi tạo chỉ số cho hai mảng
    i = j = 0

    # Duyệt qua hai mảng và thêm phần tử nhỏ hơn vào kết quả
    while i < len(A) and j < len(B):
        if A[i] < B[j]:
            result.append(A[i])
            i += 1
        else:
            result.append(B[j])
            j += 1

    # Thêm các phần tử còn lại của mảng A (nếu có)
    while i < len(A):
        result.append(A[i])
        i += 1

    # Thêm các phần tử còn lại của mảng B (nếu có)
    while j < len(B):
        result.append(B[j])
        j += 1

    return result

# Test

N = 10 # Số lượng phần tử trong mỗi mảng
A = sorted([random.randint(1, 100) for _ in range(N)])
B = sorted([random.randint(1, 100) for _ in range(N)])

print(f"Mảng A: {A}")
print(f"Mảng B: {B}")
print(f"Mảng được sắp xếp: {merge_arrays(A, B)}")
```

```
Mảng A: [20, 28, 29, 42, 51, 57, 62, 69, 96, 97]
Mảng B: [2, 8, 13, 22, 30, 37, 51, 75, 78, 83]
Mảng được sắp xếp: [2, 8, 13, 20, 22, 28, 29, 30, 37, 42, 51, 51, 57, 62, 69, 75, 78, 83, 96, 97]
```