**đề 12**

Câu 1:Nêu ý tưởng Trộn run và minh họa:3,6,7,9,2,5,8,4,1

Ý tưởng

**Trộn Run** là một thuật toán **sắp xếp ngoài (External Sorting)**, được sử dụng để sắp xếp dữ liệu có kích thước lớn vượt quá khả năng lưu trữ của bộ nhớ chính (RAM). Ý tưởng của thuật toán dựa trên việc chia nhỏ dữ liệu, sắp xếp từng phần, và sau đó trộn chúng lại với nhau theo từng bước để có dãy dữ liệu hoàn chỉnh.

Minh họa:

**Dữ liệu ban đầu (f0):**  
3 6 7 9 2 5 8 4 1

### ****Bước 1: Phân bố m = 1 phần tử từ fo vào f1,f2(1 phần tử mỗi run)****

f1: 3 7 2 8 1

f0:3 6 7 9 2 5 8 4 1

f2: 6 9 5 4

**Trộn f1 và f2 vào f0:**

F0: 3 6 7 9 2 5 4 8 1

### ****Bước 2: Phân bố m = 2\*m = 2 phần tử từ fo vào f1,f2 (2 phần tử mỗi run)****

f1: 3 6, 2 5, 1

f0: 3 6 7 9 2 5 4 8 1

f2: 7 9, 4 8

**Trộn f1 và f2 vào f0:**

f0:3 6 7 9 2 4 5 8 1

### ****Bước 3: Phân bố m = 2\*m = 4 phần tử từ fo vào f1,f2 (4 phần tử mỗi run)****

f1: 3 6 7 9, 1

f0: 3 6 7 9 2 4 5 8 1

f2: 2 4 5 8

**Trộn f1 và f2 vào f0:**  
f0:2 3 4 5 6 7 8 9 1

### ****Bước 4: Phân bố m = 2\*m = 8 phần tử từ fo vào f1,f2 (8 phần tử mỗi run)****

f1: 2 3 4 5 6 7 8 9

f0: 2 3 4 5 6 7 8 9 1

f2: 1

**Trộn f1 và f2 vào f0:**

f0:1 2 3 4 5 6 7 8 9

### ****Bước 5: Phân bổ m=2\*m = 16 phần tử từ fo vào f1,f2 Dừng thuật toán****

Khi chiều dài của run (m) lớn hơn hoặc bằng tổng số phần tử trong dãy ban đầu (n), thuật toán dừng.

### ****Kết quả cuối cùng:****

Dãy số sau khi được sắp xếp:  
1 2 3 4 5 6 7 8 9

Câu 2:Cài đặt cấu trúc hàng đợi ưu tiên sử dụng cấu trúc đống(heap)

a.Hàng DUT bao gồm:thêm push() ,loại phần tử ra pop() và lấy các giá trị ở đầu hàng đợi top()

b.Minh họa bằng hình vẽ chèn 8,3,9,1,7,5,6,4 và rút tất cả ra khỏi hàng đợi

\* Cài đặt cấu trúc hàng đợi ưu tiên sử dụng cấu trúc đống

#include "bits/stdc++.h"

using namespace std;

class PriorityQueue{

vector<int> heap;

void heapifyUp(int index){

if(index == 0) return;

int parent = (index - 1)/2;

if(heap[parent] < heap[index]){

swap(heap[parent], heap[index]);

heapifyUp(parent);

}

}

void heapifyDown(int index) {

int left = 2 \* index + 1;

int right = 2 \* index + 2;

int largest = index;

if (left < heap.size() && heap[left] > heap[largest]) {

largest = left;

}

if (right < heap.size() && heap[right] > heap[largest]) {

largest = right;

}

if (largest != index) {

swap(heap[index], heap[largest]);

heapifyDown(largest);

}

}

public:

void insert(int value) {

heap.push\_back(value);

heapifyUp(heap.size() - 1);

}

int top() {

if (heap.empty()) throw runtime\_error("Hang doi rong!");

return heap[0];

}

void pop() {

if (heap.empty()) throw runtime\_error("Hang doi rong!");

heap[0] = heap.back();

heap.pop\_back();

heapifyDown(0);

}

bool empty() {

return heap.empty();

}

void printHeap() {

for (int val : heap) {

cout << val << " ";

}

cout << endl;

}

};

int main() {

PriorityQueue pq;

vector<int> values = {8, 3, 9, 1, 7, 5, 6, 4};

for (int value : values) {

pq.insert(value);

pq.printHeap();

}

while (!pq.empty()) {

cout << "Phan tu uu tien cao nhien: " << pq.top() << endl;

pq.pop();

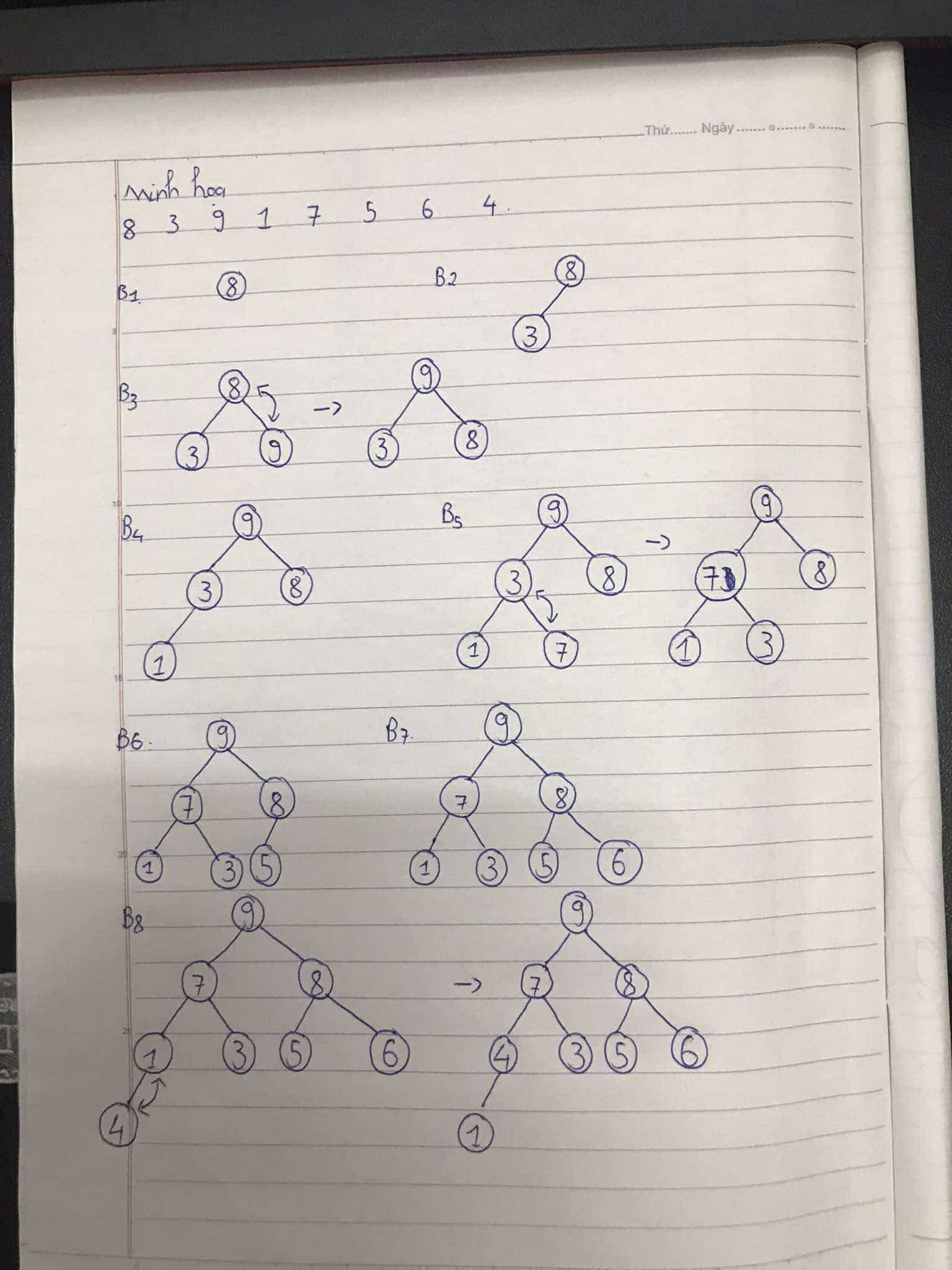
pq.printHeap();

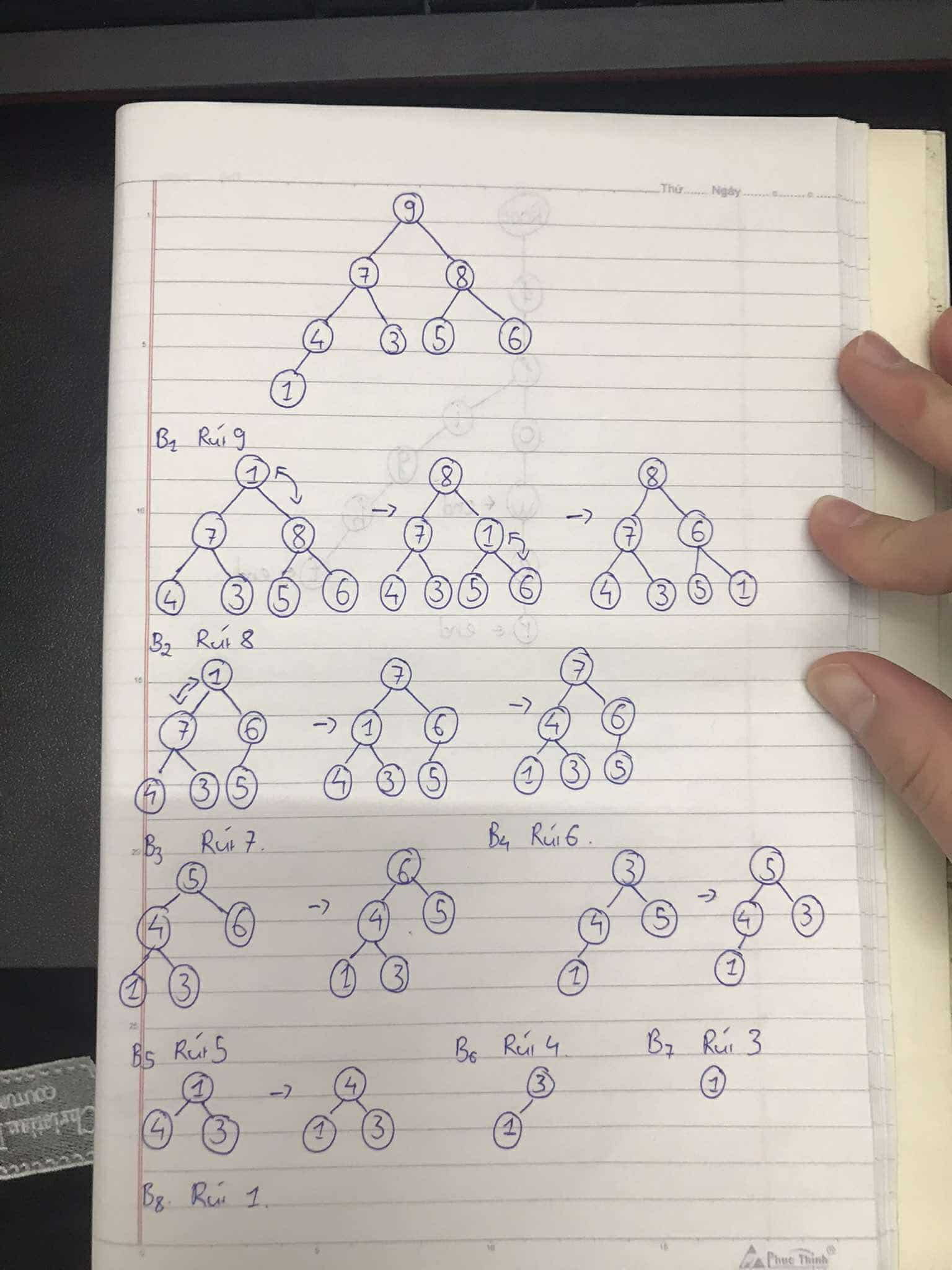
}

return 0;

Minh họa

8 3 9 1 7 5 6 4





Câu 3:Viết chương trình tìm mốt và trung vị = counting sort. Minh họa A={1,4,1,2,7,1,2,5,3,6}

Câu 3

Tìm mốt và trung vị của một mảng số sử dụng phương pháp sắp xếp đếm phân phối(counting sort). Minh họa các bước tìm mốt với mảng số A = {1, 4, 1, 2, 7, 1, 2, 5, 3, 6}.  
Chú ý: mốt là số có tần suất lớn nhất trong mảng. Với một mảng các số được sắp theothứ tự tăng dần, trung vị là số chính giữa mảng nếu mảng có số lẻ phần tử, là trungbình của hai số chính giữa mảng nếu mảng có số chẵn phần tử.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void countingSortAndFindModeMedian(const vector<int>& A) {

// Bước 1: Tìm giá trị lớn nhất trong mảng

int maxVal = \*max\_element(A.begin(), A.end());

// Bước 2: Đếm tần suất của từng giá trị

vector<int> count(maxVal + 1, 0);

for (int num : A) {

count[num]++;

}

// Bước 3: Tìm mode (số có tần suất lớn nhất)

int mode = 0, maxFrequency = 0;

for (int i = 0; i <= maxVal; i++) {

if (count[i] > maxFrequency) {

maxFrequency = count[i];

mode = i;

}

}

// Bước 4: Sắp xếp mảng bằng Counting Sort

vector<int> sortedArray;

for (int i = 0; i <= maxVal; i++) {

for (int j = 0; j < count[i]; j++) {

sortedArray.push\_back(i);

}

}

// Bước 5: Tìm trung vị (median)

int n = sortedArray.size();

double median = 0.0;

if (n % 2 == 1) {

// Nếu số lượng phần tử lẻ

median = sortedArray[n / 2];

} else {

// Nếu số lượng phần tử chẵn

median = (sortedArray[n / 2 - 1] + sortedArray[n / 2]) / 2.0;

}

// Kết quả

cout << "Mảng đã sắp xếp: ";

for (int num : sortedArray) {

cout << num << " ";

}

cout << endl;

cout << "Mode (mốt): " << mode << endl;

cout << "Median (trung vị): " << median << endl;

}

int main() {

vector<int> A = {1, 4, 1, 2, 7, 1, 2, 5, 3, 6};

countingSortAndFindModeMedian(A);

return 0;

}

Minh họa

A = {1, 4, 1, 2, 7, 1, 2, 5, 3, 6}

Bước 1:Đếm số lần xuất hiện của từng phần tử trong mảng cần sắp xếp

Index :0 1 2 3 4 5 6 7 8 9

Count: 0 3 2 1 1 1 1 1 0 0

Bước2: Sửa mảng count sao cho chỉ số của mỗi phần tử lưu tổng số lần đếm trước đó 🡺vị trí của chúng trong mảng đầu ra

Index :0 1 2 3 4 5 6 7 8 9

Count: 0 3 5 6 7 8 9 10 10 10

Bước3: Xuất từng phần tử trong chuỗi đầu vào tại ví trí là tổng số lần đếm trừ đi 1 và trừ số lần đếm đi 1.

Index :0 1 2 3 4 5 6 7 8 9

Count: 1 1 1 2 2 3 4 5 6 7

### ****Bước 4: Tìm Mode (Mốt)****

1. **Mode (Mốt)** là số có tần suất lớn nhất trong mảng count.
2. **Duyệt qua mảng** count **để tìm giá trị lớn nhất:**
   1. count[1] = 3 (lớn nhất).
   2. Mode = **1**.