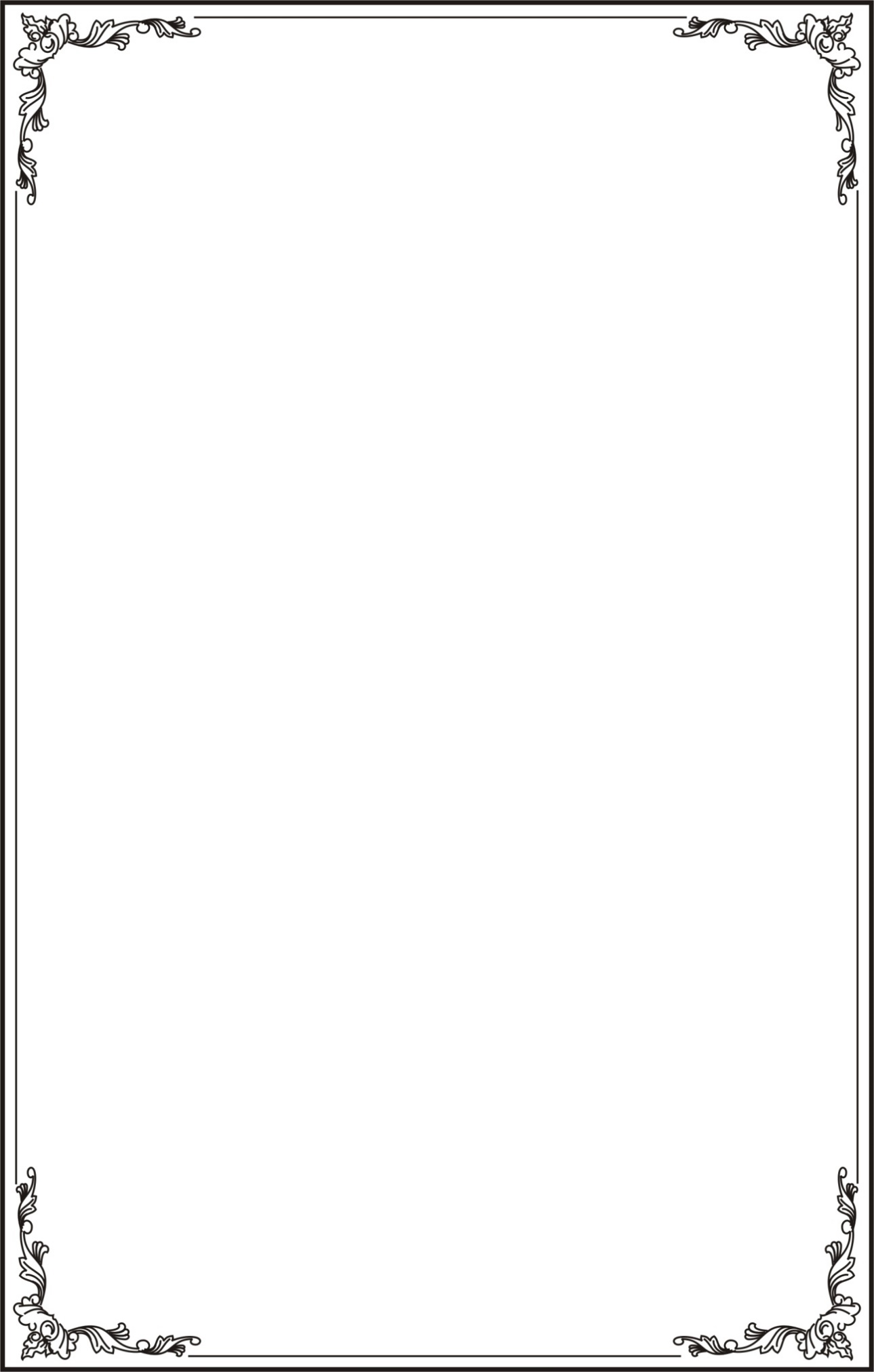
Logo, icon

Description automatically generated

**ĐỀ TÀI:**

**THUẬT TOÁN SẮP XẾP**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT**

**THÀNH PHỐ HỒ CHÍ MINH**

**Mã môn học: [DASA230179\_22\_1\_09](https://utex.hcmute.edu.vn/course/view.php?id=25009" \o "Cau truc du lieu va giai thuat_ Nhom 09)**

**Giảng viên: Huỳnh Xuân Phụng**

**Nhóm 18**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Nguyễn Quỳnh Nhi | 21110582 |
| 2 | Lỷ Toàn Phát | 21110910 |
| 3 | Vũ Thị Bích Ngọc | 21110905 |

**Mục lục**

[I. Comparision Counting: 1](#_Toc21977)

[1. Thuật toán: 1](#_Toc6599)

[2. Code: 1](#_Toc27628)

[3. Chạy tay: 2](#_Toc5406)

[II. Distribution Counting 3](#_Toc3670)

[1. Thuật Toán: 3](#_Toc2360)

[2. Code: 3](#_Toc22050)

[3. Chạy tay: 4](#_Toc8995)

[III. Straight Selection Sort 8](#_Toc7061)

[1. Thuật toán ( Straight Selection Sort ). 8](#_Toc25557)

[2. Code 8](#_Toc18049)

[3. Chạy tay: 8](#_Toc15270)

[IV. Shell sort: 10](#_Toc22125)

[1. Thuật toán: 10](#_Toc25188)

[2. Code: 10](#_Toc2246)

[3. Chạy tay: 11](#_Toc22847)

[V. Quick Sort 14](#_Toc30119)

[1. Thuật toán: 14](#_Toc11489)

[2. Code: 14](#_Toc14)

[3. Chạy tay: 15](#_Toc359)

**I. Comparision Counting:**

1. **Thuật toán:**

Sắp xếp mảng R có n phần tử, với các giá trị là K, bằng cách sử dụng một mảng đếm COUNT cũng có n phần tử. Đếm số lượng K bé hơn một K cho trước. Sau khi thuật toán kết thúc, COUNT[i+1] sẽ chỉ ra vị trí của Rj.

***Bước 1:*** Cài đặt mảng COUNT có tất cả các giá trị bằng 0.

***Bước 2****:* [Vòng lặp i] Thực hiện bước 3, for (i = 0; i < N; i++).

***Bước 3:*** [Vòng lặp j] Thực hiện bước 4, for (i = N - 1; i >= 1; i--).

***Bước 4:*** [So sánh Ki với Kj] Nếu Ki<Kj tăng COUNT[j] thêm 1, ngược lại tăng COUNT[i] thêm 1.

1. **Code:**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

struct Record

{

char data;

int key;

};

void ComparisionCounting(Record M[], int N) {

int\* COUNT = (int\*)malloc(sizeof(int));

int KEY[100];

int i, j;

for (i = 0; i < N; i++) {

COUNT[i] = 0;

}

for (i = N - 1; i >= 1; i--)

{

for (j = i - 1; j >= 0; j--)

{

if (M[i].key <= M[j].key)

COUNT[j] += 1;

else

COUNT[i] += 1;

}

}

for (i = 0; i < N; i++) {

KEY[COUNT[i]] = M[i].key;

}

for (int i = 0; i < N; i++)

{

cout << KEY[i] << " ";

}

}

int main() {

Record MANG[5];

MANG[0].key = 1; MANG[1].key = 5;

MANG[2].key = 12; MANG[3].key = 3;

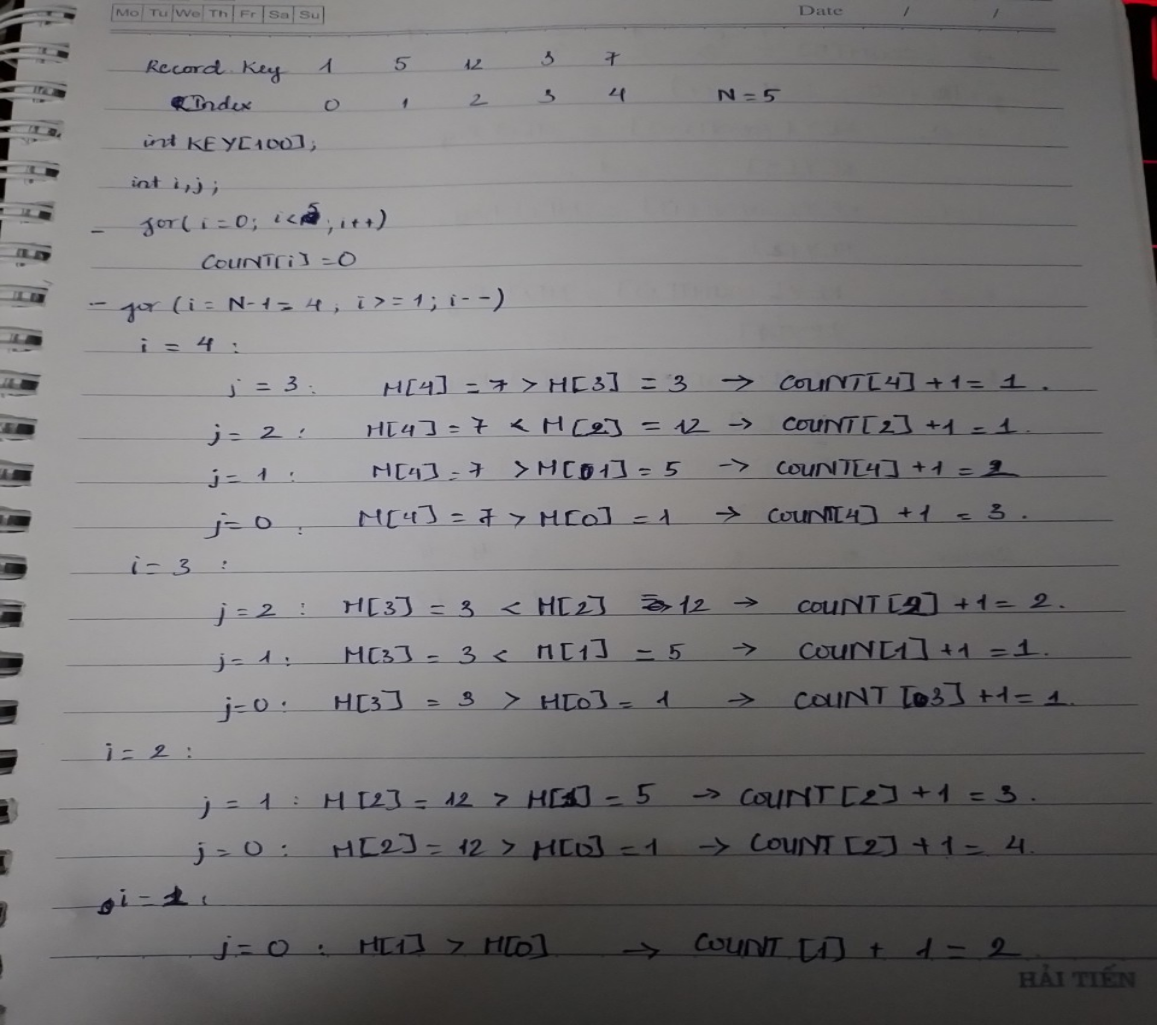
MANG[4].key = 7;

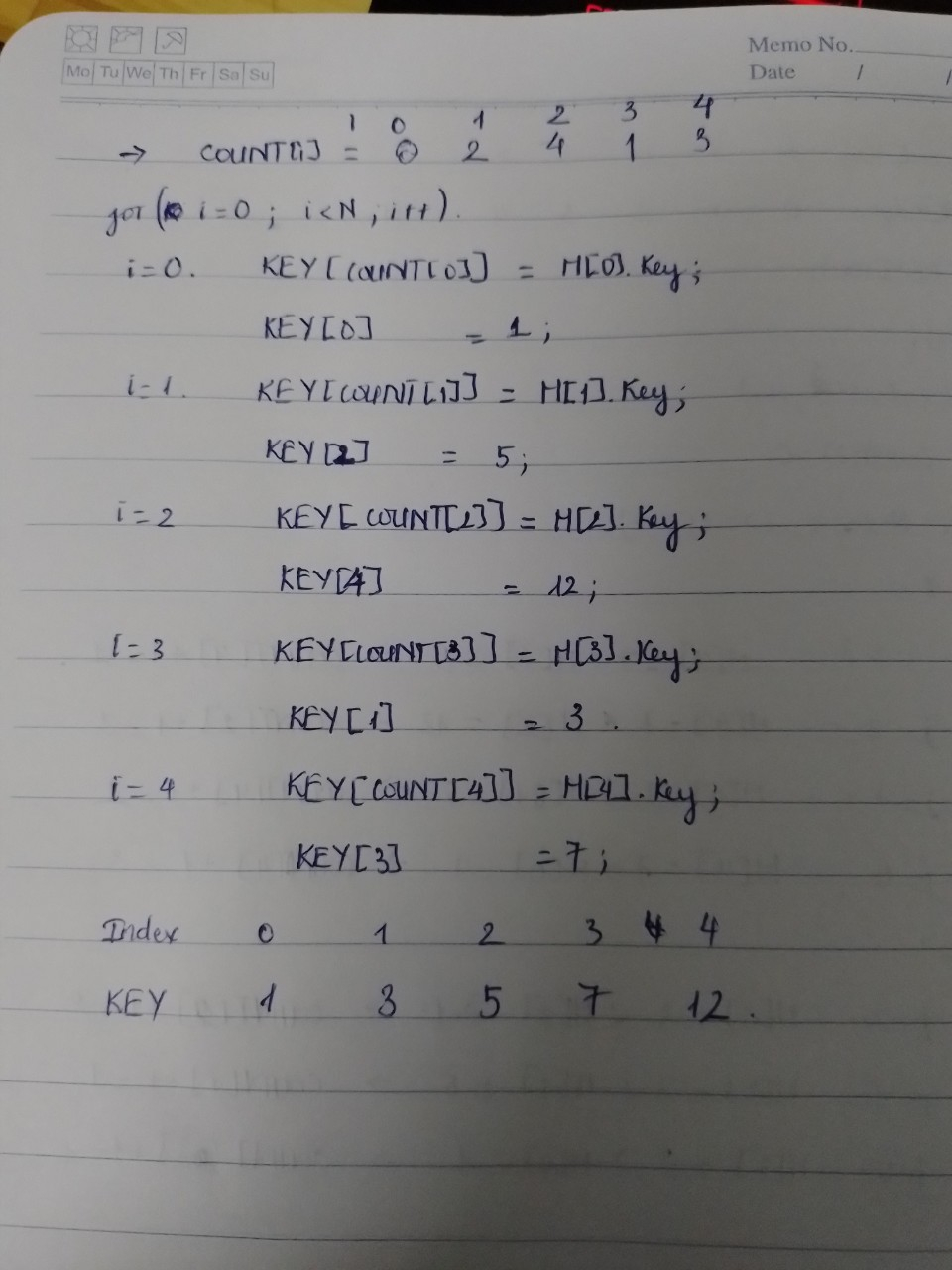
ComparisionCounting(MANG, 5);

return 1;

}

1. **Chạy tay:**

****

****

**II. Distribution Counting**

1. **Thuật Toán:**

Sắp xếp mảng M có n phần tử. Mảng S là mảng M đã được sắp xếp.

**Bước 1:** Tìm u và v tương ứng với các giá trị min và max của mảng.

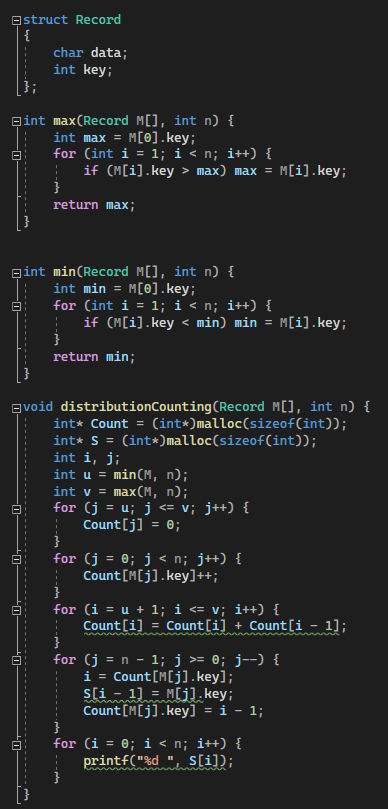
**Bước 2:** Khởi tạo mảng đếm COUNT bằng 0 trong khoảng u đến v.

**Bước 3:** Tăng giá trị của COUNT[M[ j ]] lên 1 đơn vị với 0 ≤ j < n - 1.

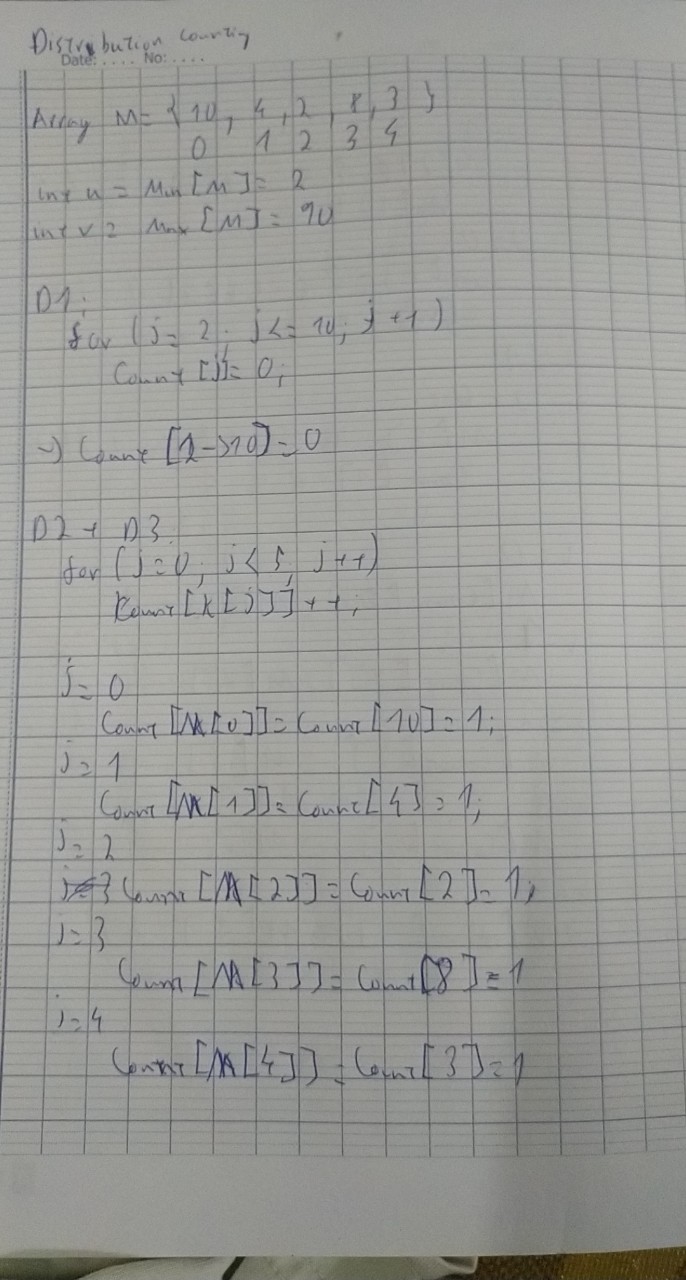
**Bước 4:** Tăng giá trị của COUNT[M[ i ]] lên COUNT[M[i - 1]] đơn vị với u < i ≤ v.

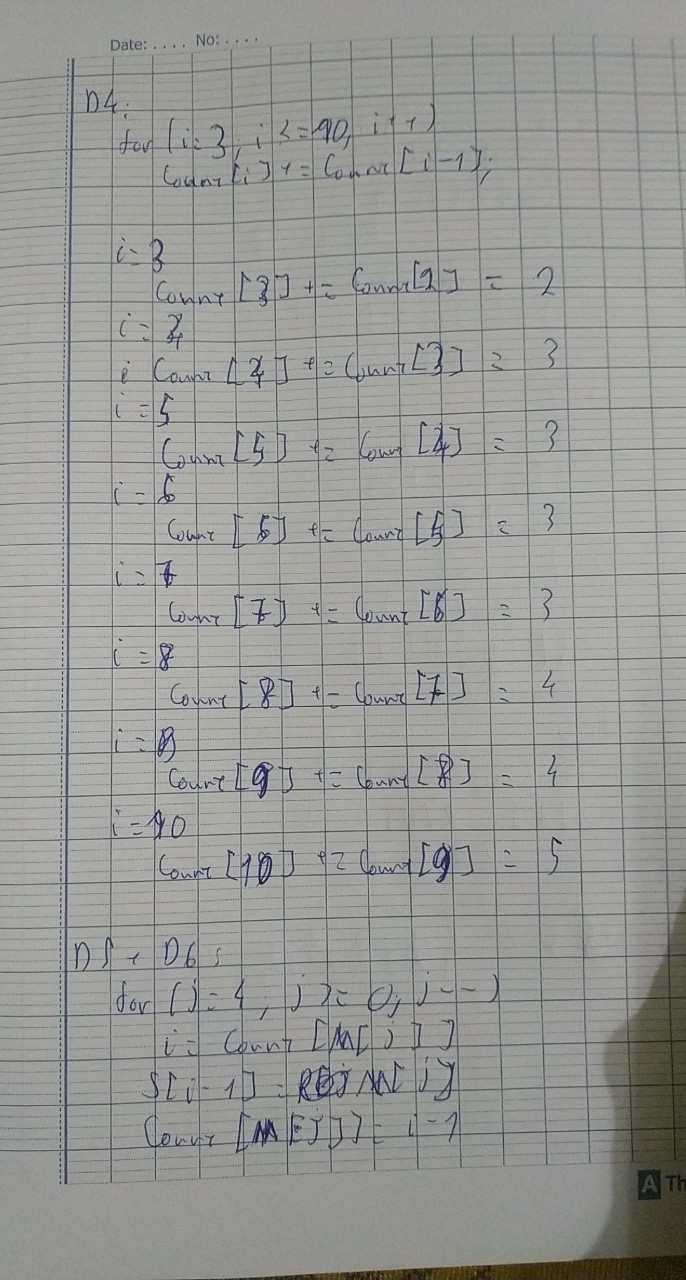
**Bước 5:** Chạy bước này với n > j ≥ 0. Gán i bằng COUNT[M[ j ]], gán S[ i - 1 ] bằng M[ j ], gán COUNT[M[ j ]] = i - 1. Kết thúc thuật toán.

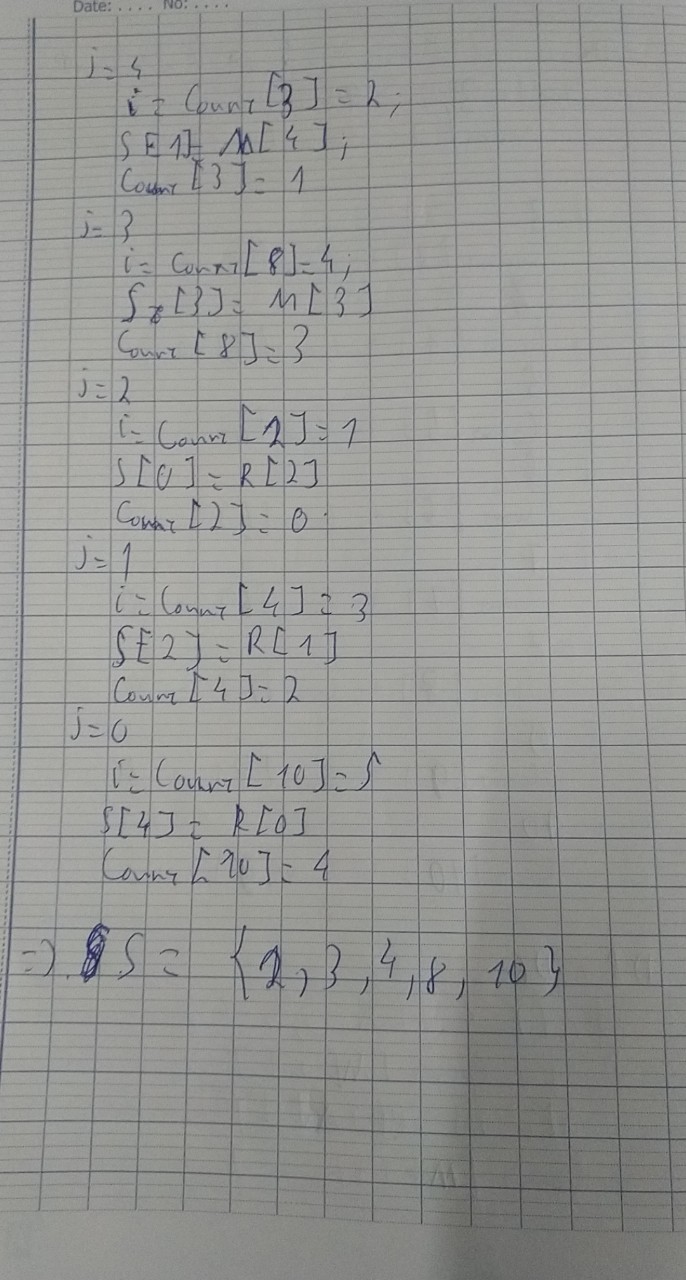
1. **Code:**

****

1. **Chạy tay:**





****

**III. Straight Selection Sort**

1. **Thuật toán ( Straight Selection Sort ).**

Các bản ghi Rn..R1 được sắp xếp lại ngay tại chỗ; sau khi được sắp xếp lại, các khóa của chúng sẽ theo đúng thứ tự là K1 <= … <=Kn. Việc sắp xếp dựa theo phương pháp này, ngoại trừ việc chọn phần tử lớn nhất trước, sau đó sẽ chọn phần tử lớn thứ hai… sẽ trở nên thuận tiện hơn.

**Bước 1:** [ Lặp lại trên j ] Thực hiện các bước 2 và 3 khi cho j = n, n-1…2

**Bước 2:** [ Tìm giá trị lớn nhất ] Thông qua các phím Kj, Kj-1,... K1 để tìm giá trị lớn nhất, đặt nó là Ki, trong đó i càng lớn càng tốt.

**Bước 3:** [Trao đổi với Rj ] Trao đổi hai giá trị R[i] và R[j] ( bây giờ các bản ghi Rj… Rn đã ở vị trí cuối cùng của nó.

1. **Code**

struct Record

{

char data;

int K;

};

void swap(int& a, int& b)

{

int temp = a;

a = b;

b = temp;

}

void SelectionSort(Record R[], int n)

{

for (int j = n; j >= 2; j--)

{

int max = j;

for (int i = j - 1; i >= 1; i--) {

if (R[i].K > R[max].K)

max = i;

}

swap(R[max].K, R[j].K);

}

// PrinRecordKey(R, n);

}

1. **Chạy tay:**

Array M = { 5, 3, 7, 8, 1};

K 1 2 3 4 5

S1. for(int j = 5; j>= 2; j-- )

{

// j = 5

max = 5;

R[4] = 8 > R[5] = 1

-> max = 4;

R[3] = 7 < R[4] = 8

-> max = 4

R[2] = 3 < R[4] = 8

-> max = 4

R[1] = 5 < R[4] = 8

-> max = 4

swap(R[4], R[5])

-> M = { 5, 3, 7, 1, 8 }

K 1 2 3 4 5

// j = 4

max = 4;

R[3] = 7 > R[4] = 1

-> max = 3;

R[2] = 3 < R[3] = 7

-> max = 3;

R[1] = 5 < R[3] = 7

-> max = 3;

swap(R[3], R[4])

-> M = { 5, 3, 1, 7, 8 }

K 1 2 3 4 5

// j = 3

max = 3;

R[2] = 3 > R[3] = 1

-> max = 2;

R[1] = 5 > R[2] = 3

-> max = 3;

swap(R[1], R[3])

-> M = { 1, 3, 5, 7, 8 }

K 1 2 3 4 5

// j = 2

max = 2;

R[1] = 1 < R[2] = 3

-> max = 3;

}

-> S = {1, 3, 5, 7, 8};

**IV. Shell sort:**

1. **Thuật toán:**

Sắp xếp các phần tử cách xa nhau và liên tục giảm khoảng cách giữa các phần tử được sắp xếp. Với trình tự N/2, N/4, …, 1.

***Bước 1:*** Khởi tạo giá trị khoảng cách.

***Bước 2****:* Chia mảng thành các mảng con nhỏ hơn tương ứng với mỗi gap. Sắp xếp lại các phần tử trong khoảng gap.

***Bước 3:***

i = i + 1;

Nếu i < gap : Lặp lại bước 2.

Nếu i >= gap: Dừng.

1. **Code:**

#include<iostream>

using namespace std;

void shellSort(int \*M, int n)

{

for (int gap = n / 2; gap > 0; gap /= 2)

{

for (int i = gap; i < n; i += 1)

{

int temp = M[i];

int j;

for (j = i; j >= gap && M[j - gap] > temp; j -= gap)

{

M[j] = M[j - gap];

}

M[j] = temp;

}

}

for (int i = 0; i < n; i++)

{

cout << M[i] << " ";

}

}

int main()

{

int\* M = (int\*)malloc(sizeof(int));

int n;

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> M[i];

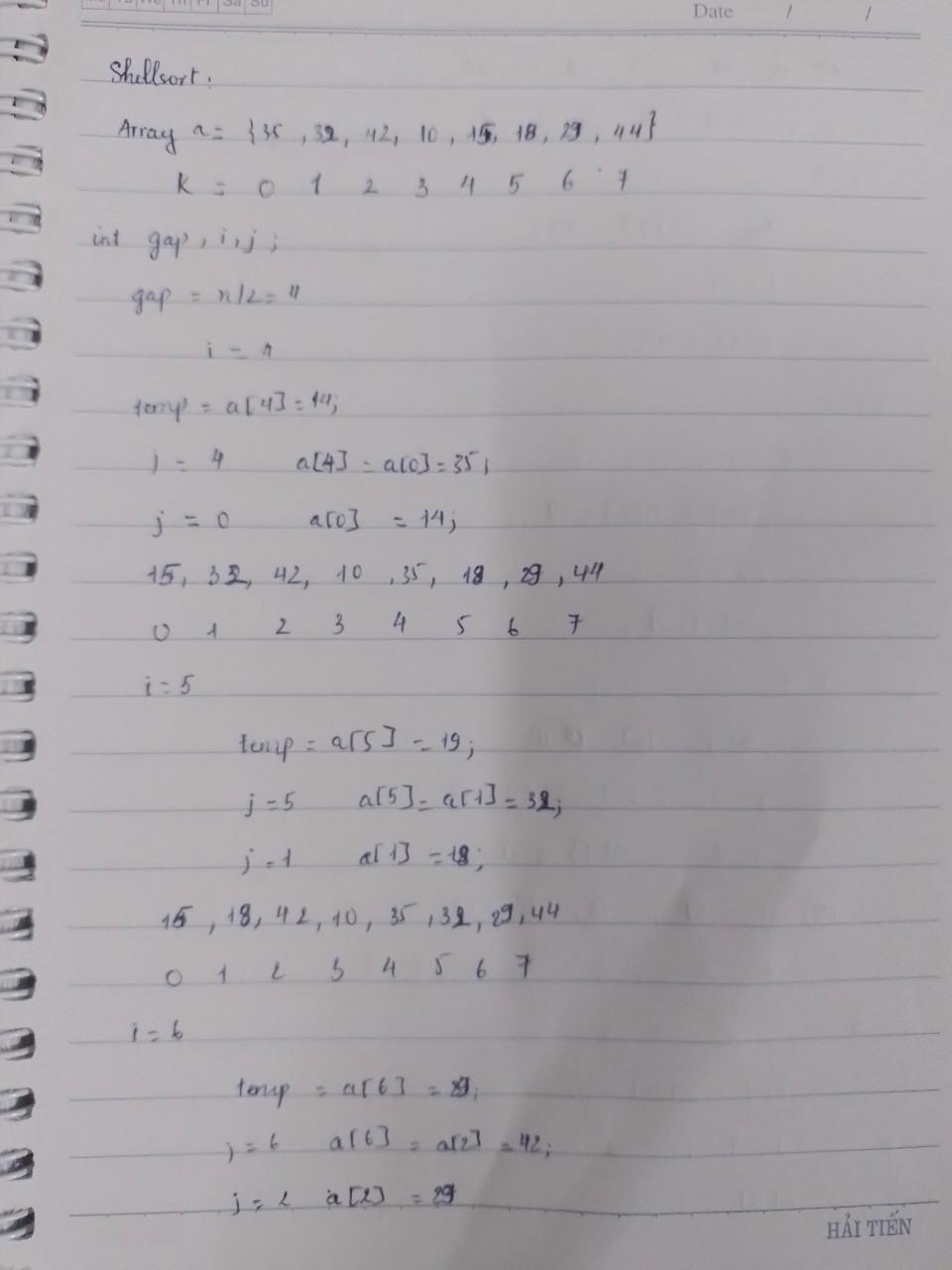
}

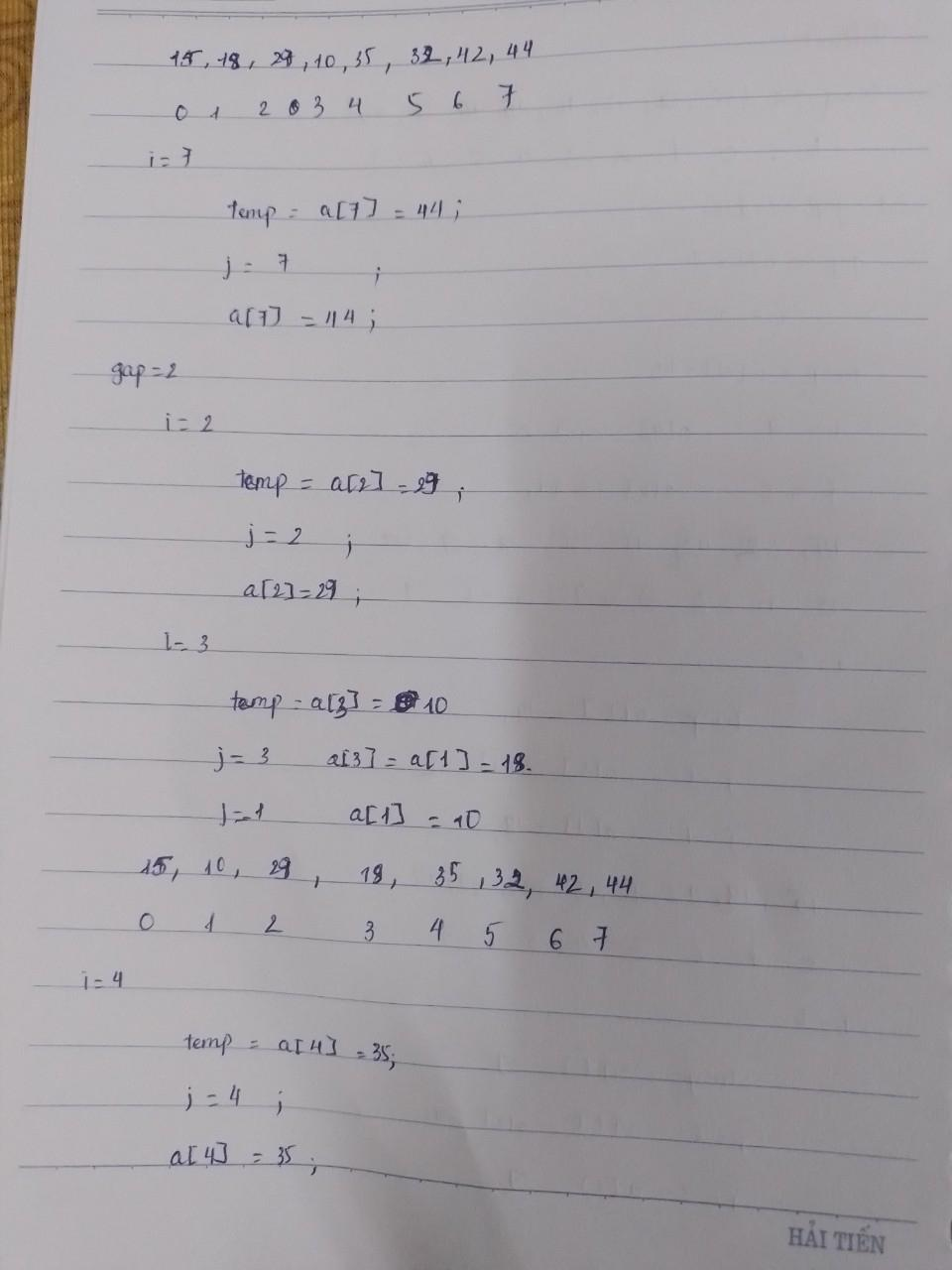
shellSort(M, n);

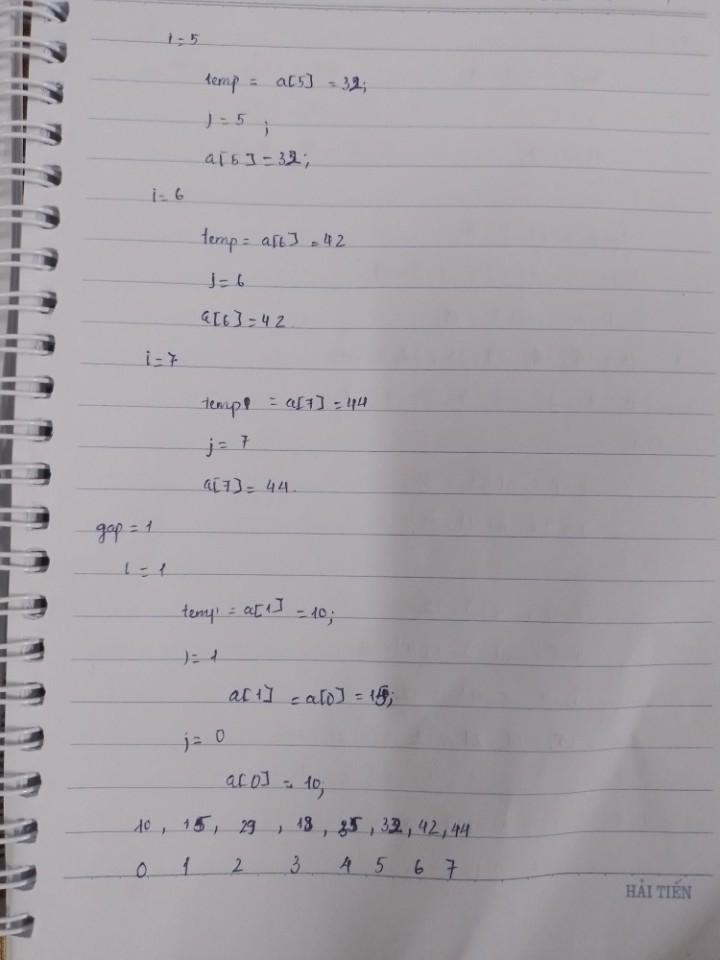
return 0;

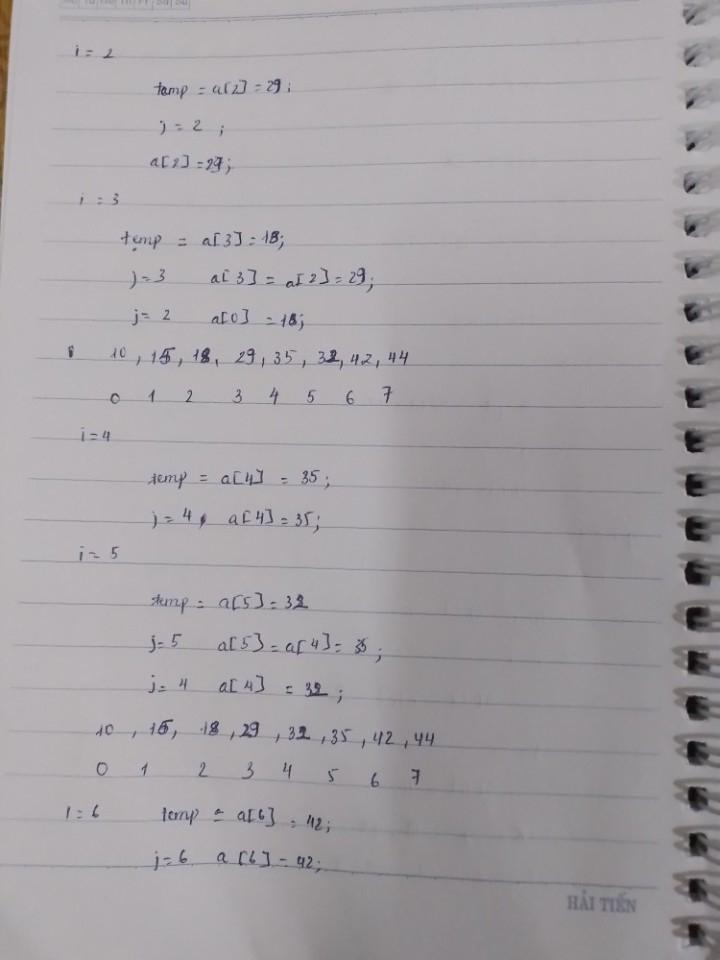
}

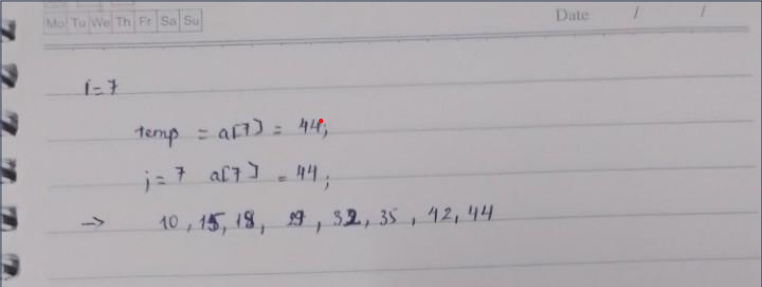
1. **Chạy tay:**

****

****

****

****

****

**V. Quick Sort**

1. **Thuật toán:**

Sắp xếp mảng M có n phần tử. Với ***LOW*** là số thứ tự thấp nhất trong mảng và ***HIGH*** là số thứ tự cao nhất trong mảng

**Bước 1**: Chọn phần tử cuối cùng của mảng làm phần tử gốc để so sánh.

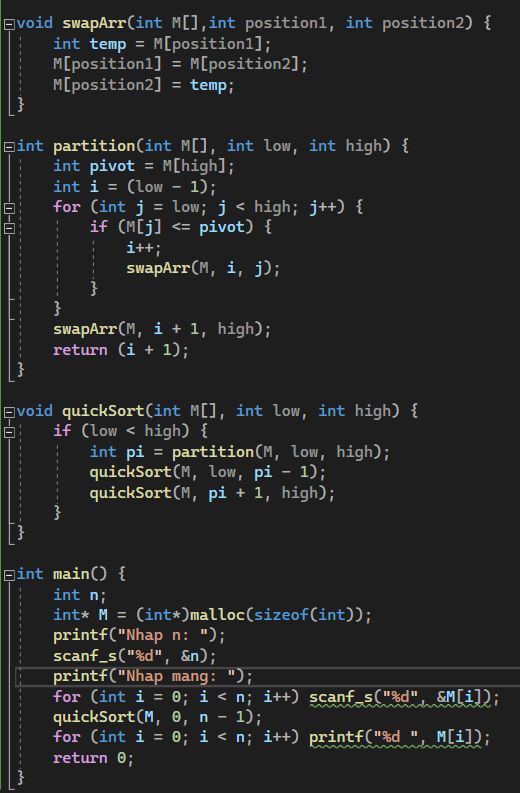
**Bước 2**: Khởi tạo biến i = ***LOW*** 1.

**Bước 3:** Với low ≤ j < ***HIGH***, nếu M[ j ] nhỏ hơn hoặc bằng với phần tử gốc, tăng biến i lên 1 đơn vị và đổi vị trí 2 phần tử M[ i ] và M[ j ].

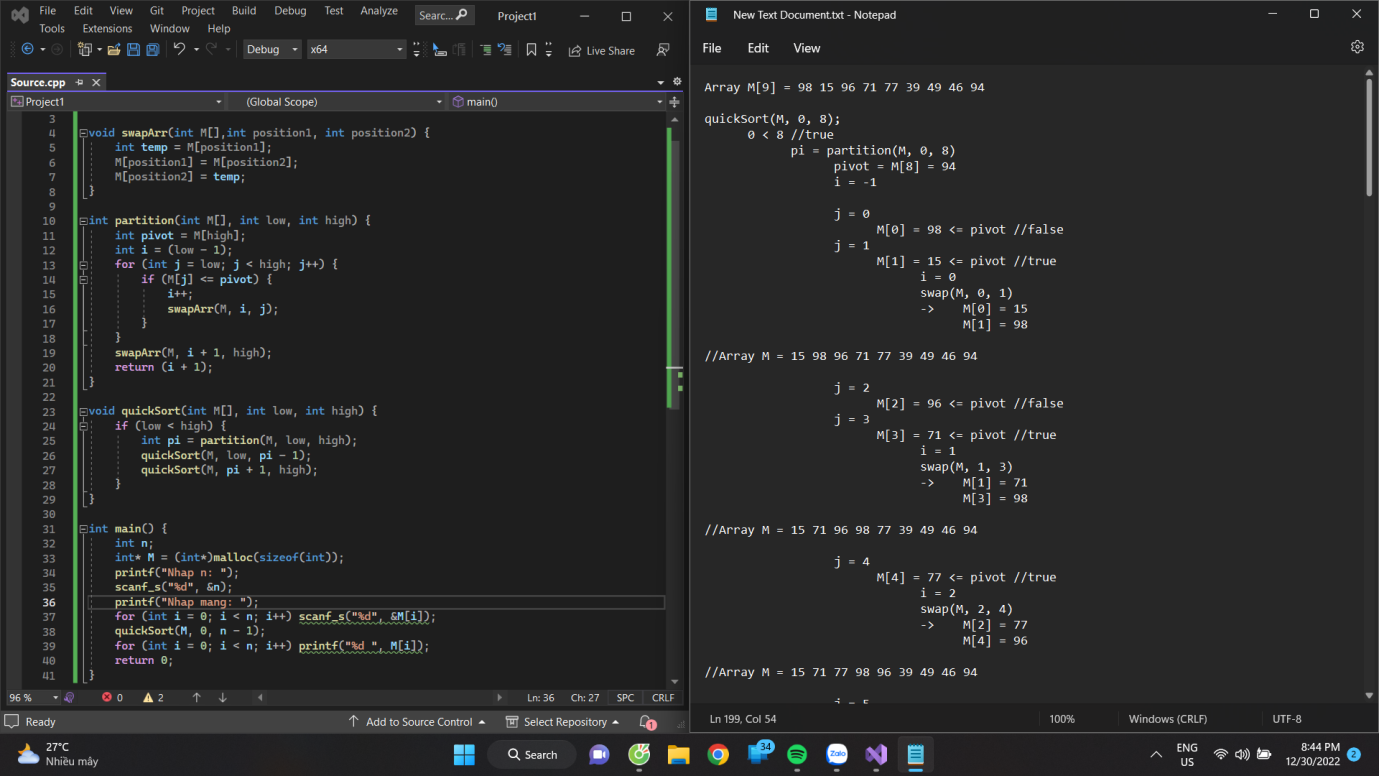
**Bước 4:** Sau khi hoàn thành vòng lặp, đổi vị trí 2 phần tử M[i + 1] và M[***HIGH***].

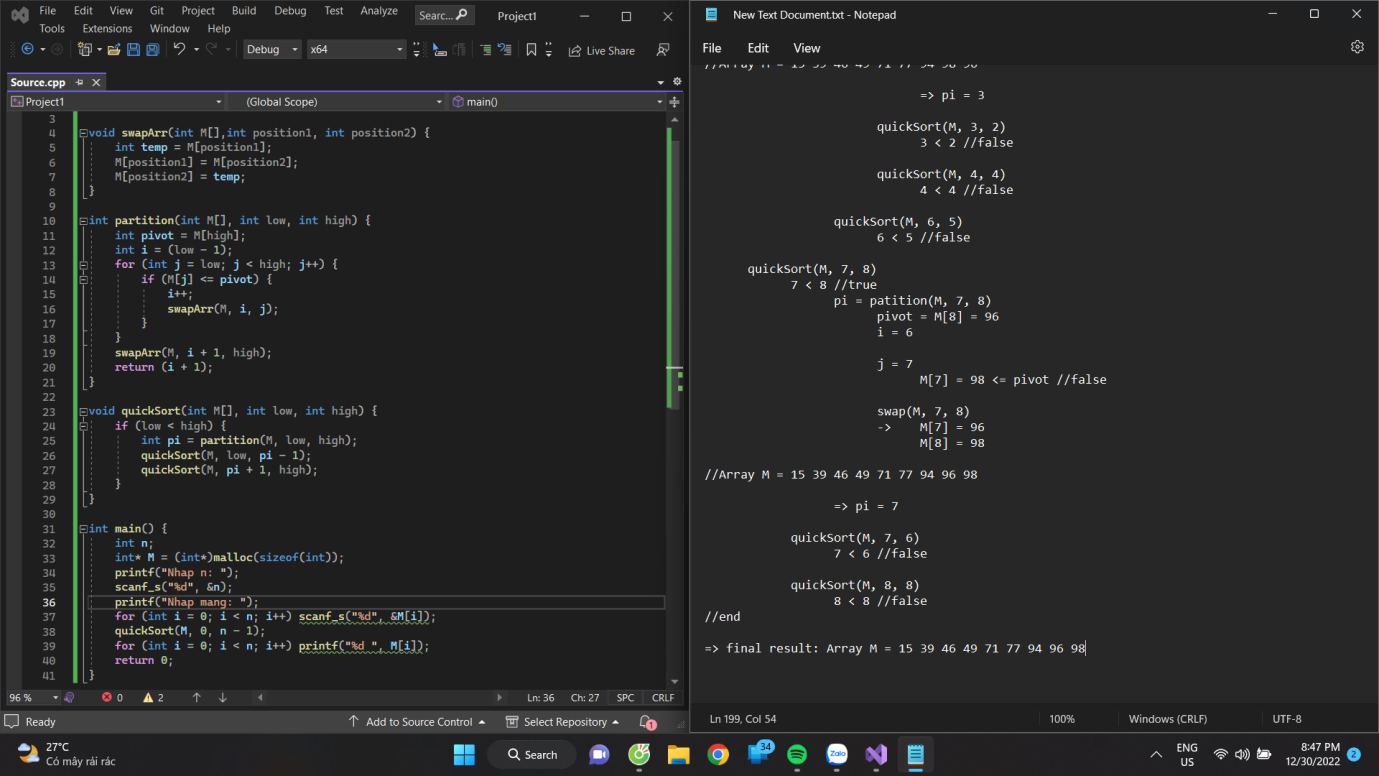
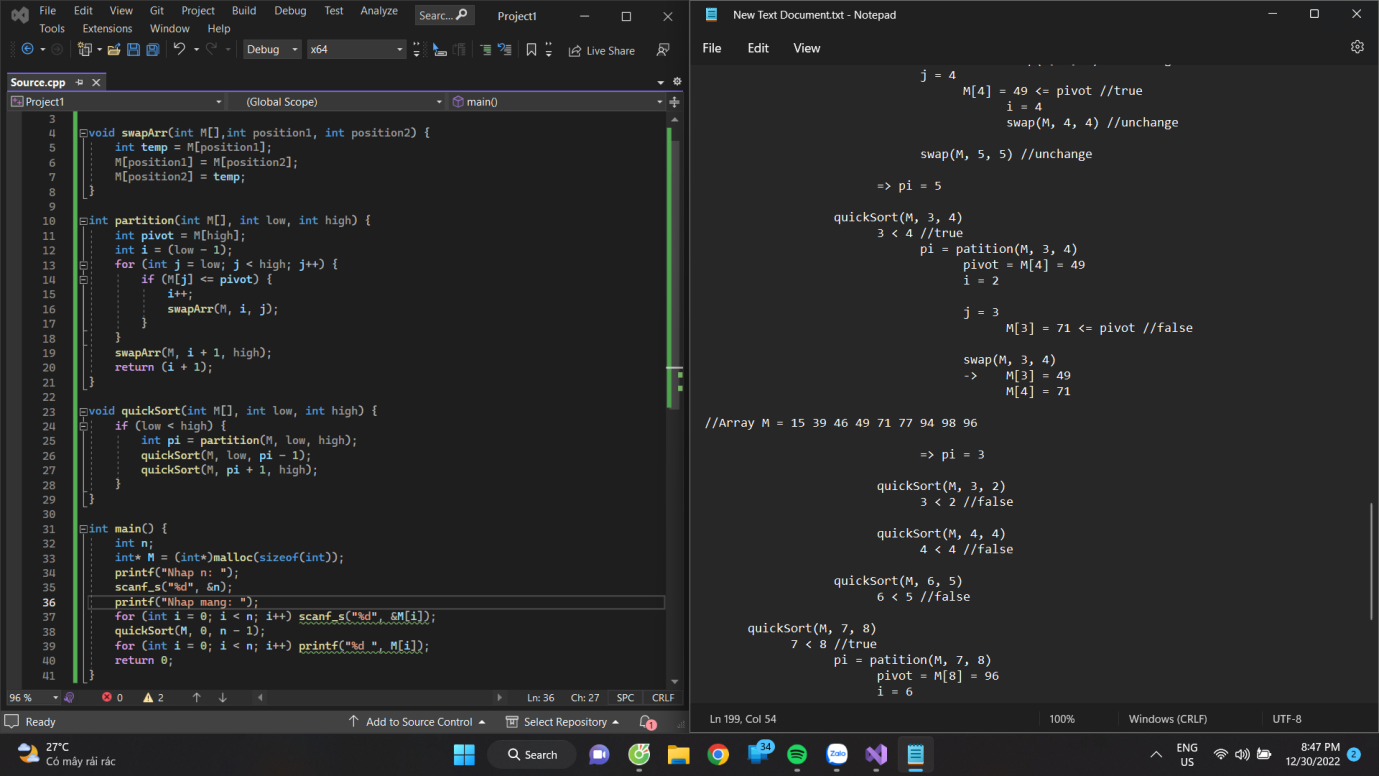
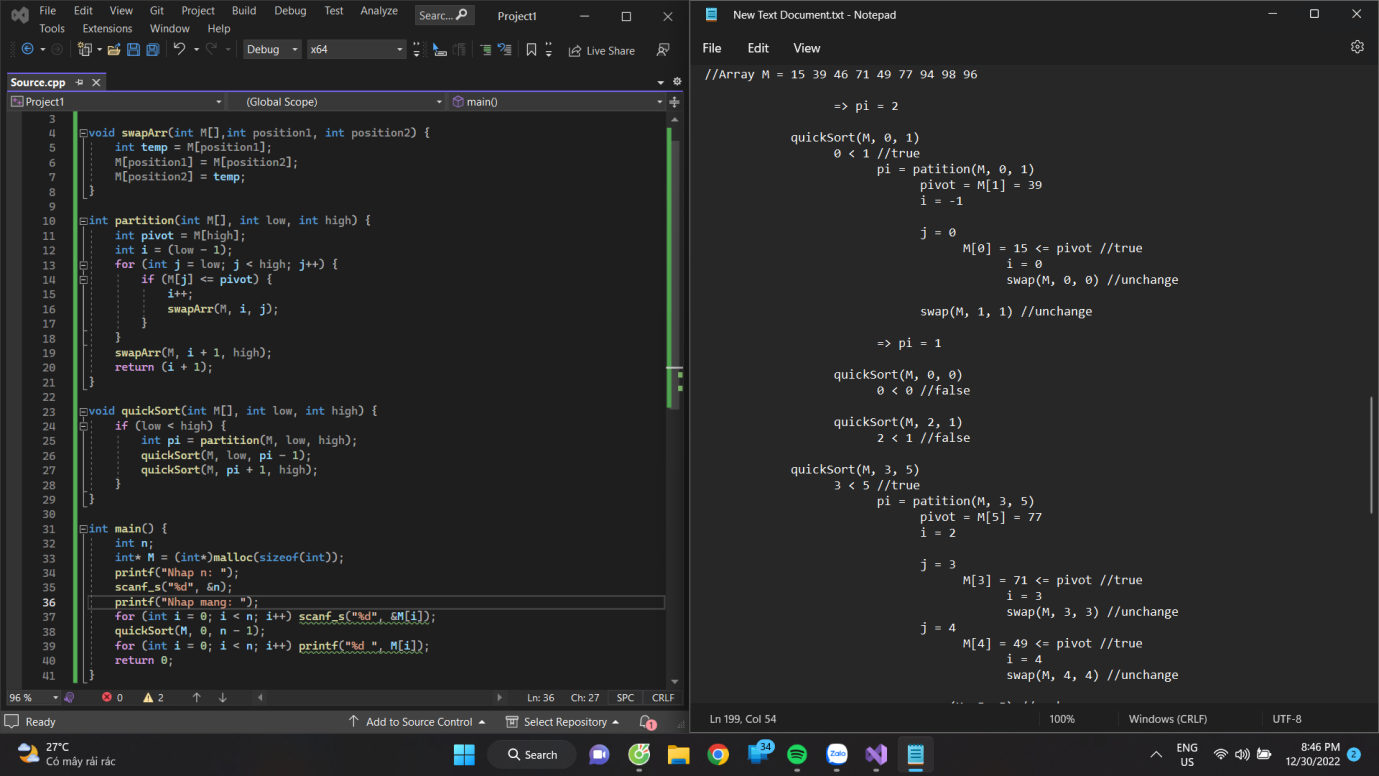
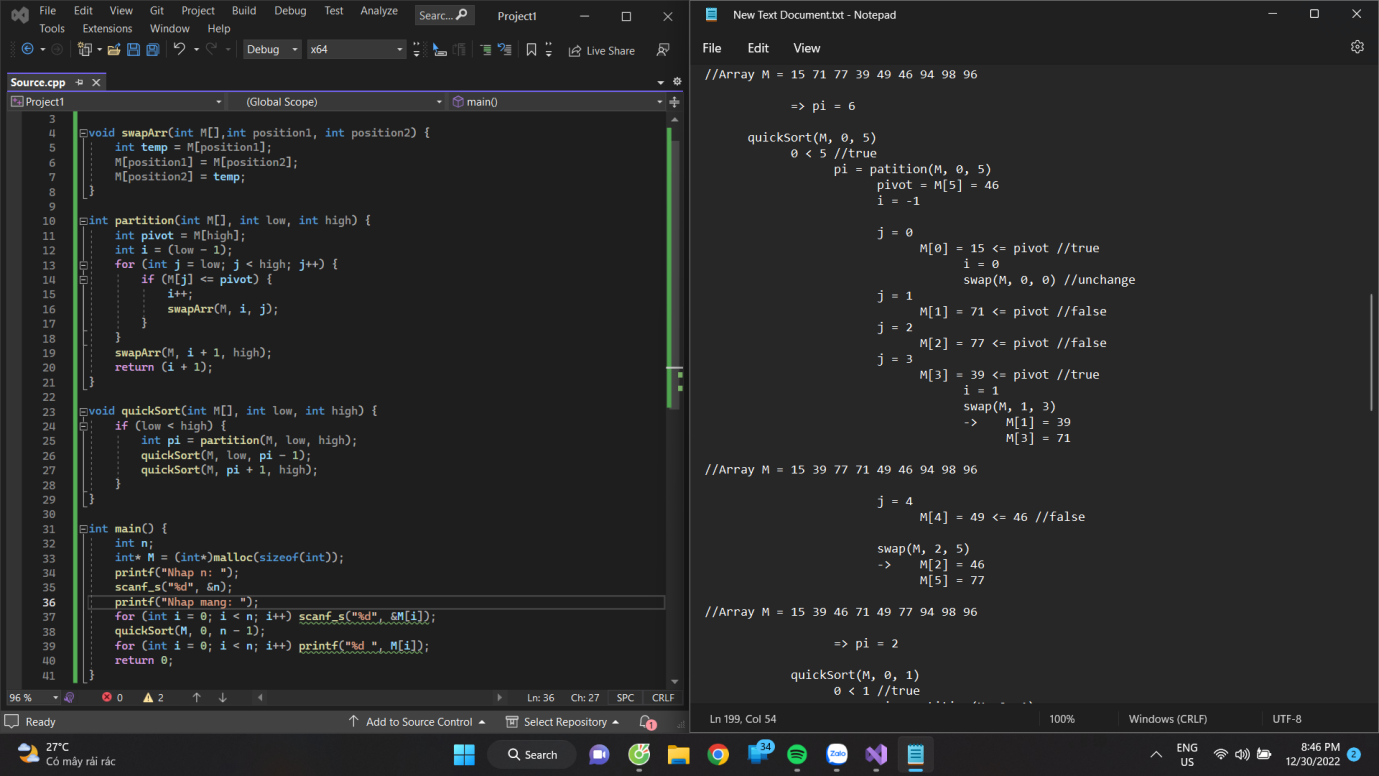
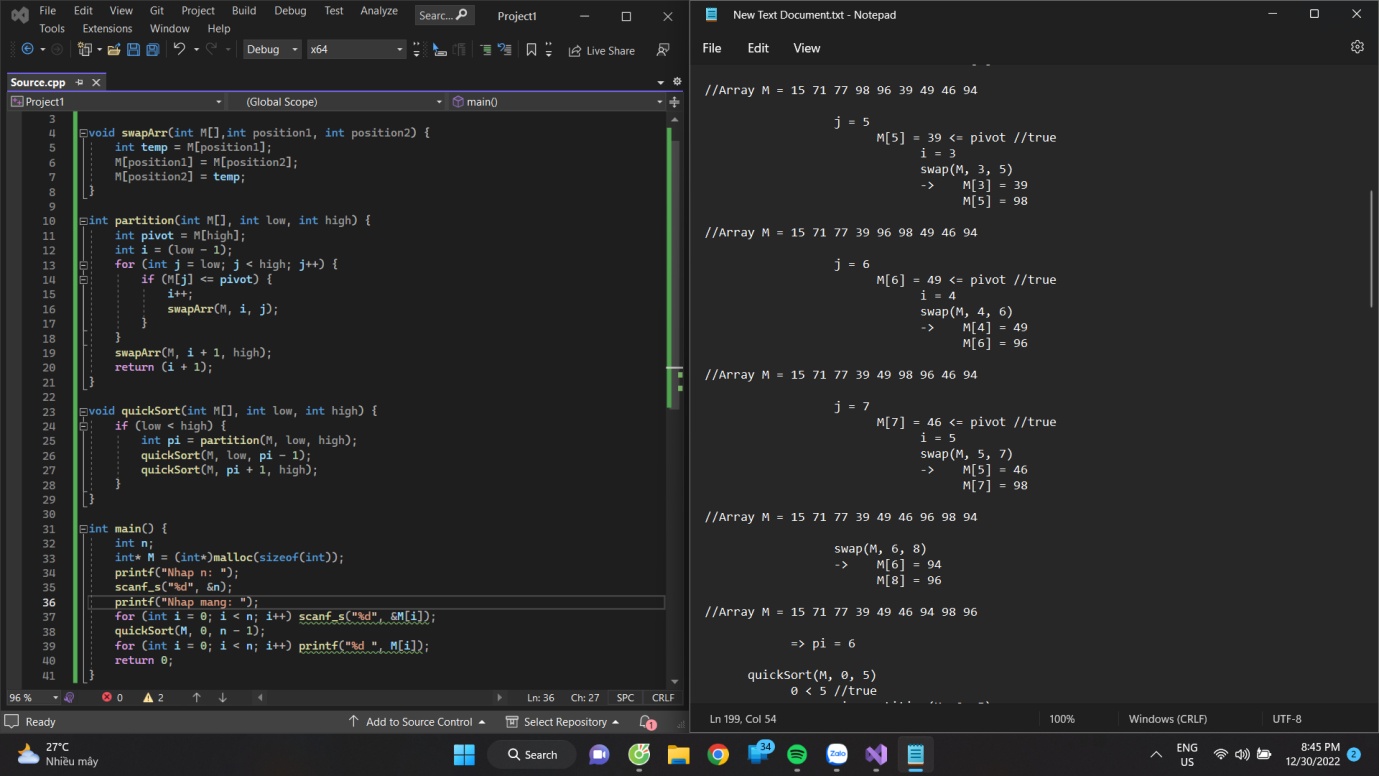
**Bước 5:** Lặp lại bước 1 - 4 với các mảng con có thứ tự ***LOW*** đến i và i + 2 đến ***HIGH*** cho đến khi ***LOW*** ≥ ***HIGH***.

1. **Code:**

****

1. **Chạy tay:**

****

****