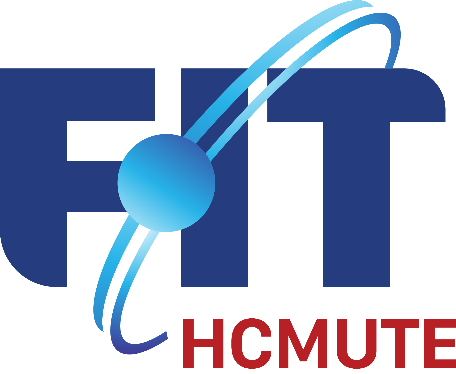
**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**



**HUỲNH THANH QUY MSSV: 21110622**

**CÁP LÊ HỮU TÂN MSSV: 21110920**

**PHẠM HỮU TUẤN MSSV: 21110713**

**Đề Tài:**

**MÔ PHỎNG CÁC THUẬT TOÁN SẮP XẾP BẰNG GRAPHICS.H**

**TIỂU LUẬN KẾT THÚC HỌC PHẦN**

**Học kỳ 1 /2022-2023**

**GVHD: TS.HUỲNH XUÂN PHỤNG**

|  |  |
| --- | --- |
| **ĐH SƯ PHẠM KỸ THUẬT TP.HCM**  **KHOA CNTT**  \*\*\*\*\*\*\* | **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**  **Độc lập – Tự do – Hạnh Phúc**  \*\*\*\*\*\*\* |

**PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

Họ và tên sinh viên 1: ………………………………………MSSV 1:……………………

Họ và tên sinh viên 2: ………………………………………MSSV 2:……………………

Họ và tên sinh viên 3: ………………………………………MSSV 3:……………………

Ngành: Công nghệ Thông tin

Tên đề tài: ……………………………………………………………………………………

………………………………………………………………………………………………

Họ và tên Giáo viên hướng dẫn: ……………………………………………………………

**NHẬN XÉT:**

1. Về nội dung đề tài khối lượng thực hiện:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Ưu điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Khuyết điểm:

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Đánh giá loại:
2. Điểm:

Tp. Hồ Chí Minh, ngày tháng năm 2021

Giáo viên hướng dẫn

(Ký & ghi rõ họ tên)

**MỤC LỤC**

[**Phần 1: MỞ ĐẦU 3**](#_Toc121185298)

[**I. Lý do chọn đề tài 3**](#_Toc121185299)

[**II. Mục đích của đề tài 4**](#_Toc121185300)

[**III. Phương pháp nghiên cứu 4**](#_Toc121185301)

[**Phần 2: NỘI DUNG 4**](#_Toc121185302)

[**I. Giải thích thuật toán 4**](#_Toc121185299)

1. **Thuật toán Comparison Counting Sort**
2. **Thuật toán Shell Sort**
3. **Thuật toán Quick Sort**
4. **Thuật toán Selection Sort**
5. **Thuật toán Two-Way Merge Sort**
6. **Thuật toán Straight Insertion Sort**
7. **Thuật toán Bubble Sort**

[**II. Chạy tay ví dụ các thuật toán 8**](#_Toc121185300)

1. **Thuật toán Comparison Counting Sort**
2. **Thuật toán Shell Sort**
3. **Thuật toán Quick Sort**
4. **Thuật toán Selection Sort**
5. **Thuật toán Two-Way Merge Sort**
6. **Thuật toán Straight Insertion Sort**
7. **Thuật toán Bubble Sort**

[**Phần 3: KẾT LUẬN 25**](#_Toc121185309)

**Phần 1: MỞ ĐẦU**

1. **Lý do chọn đề tài**

* Đối với ngành công nghệ thông tin nói chung thì kiến thức cơ bản là các kiến thức liên quan đến kiểu dữ liệu, câu lệnh điều khiển, câu lệnh điều kiện… Những kiên thức này không là của riêng 1 ngôn ngữ lập trình nào. Và dĩ nhiên, cấu trúc dữ liệu và giải thuật cũng như thế. Đó là kiến thức liên quan đến cách tổ chức giữa dữ liệu và các giải thuật trên từng tổ chức đó. Và 1 chương trình máy tính muốn chạy được phải có dữ liệu và các thuật toán, việc đảm bảo được dữ liệu tổ chức theo cấu trúc phù hợp và được thực thi với một thuật toán hợp lý sẽ tăng được hiệu suất lên rất nhiều. Trong đó, một trong các loại thuật toán quan trọng và cơ bản nhất đó là thuật toán sắp xếp. Qua bộ môn cấu trúc dữ liệu và giải thuật, chúng ta được học rất nhiều thuật toán liên quan tới sắp xếp một cách trực quan, và qua đề tài này mà chúng ta cũng hiểu rõ được thuật toán sắp xếp đó được chạy như thế nào, thông qua đó mà chúng ta có thể rút ra cho chính mình kinh nghiệm trong lập trình rằng lúc nào cần dùng thuật toán nào cho phù hợp với vấn đề cần giải quyết để đạt được hiệu suất cao nhất có thể.

1. **Mục đích của đề tài**

* Ngoài việc chúng ta biết thêm về các thuật toán sắp xếp, chúng ta có thể hiểu sâu hơn về cách mà thuật toán sắp xếp đó chạy thông qua các chuyển động dễ hiểu, nhằm mục đích nâng cao sự thú vị trong việc học, nhưng chủ yếu là sẽ giúp chúng ta hiểu được rằng thuật toán sẽ được chạy như thế nào để áp dụng vào từng bài toán cho phù hợp, đồng thời rút ra được ưu và nhược điểm của thuật toán sắp xếp đó.

1. **Phương pháp nghiên cứu**

* Làm việc theo nhóm.
* Đưa ra kịch bản, hình dung ra cách mà bài toán sẽ vận hành.
* Viết code sườn cho thuật toán để biết rằng thuật toán đó có chạy đúng hay không.
* Bổ sung thêm màu sắc, chuyển động cho thuật toán để người xem có thể hiểu thuật toán 1 cách trực quan hơn.

**Phần 2: NỘI DUNG**

1. **Giải thích thuật toán**
2. **Thuật toán Comparison Counting Sort**

* Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ).
* Bước 2: Tạo 1 mảng có kiểu dữ liệu int đặt tên là Count có n phần tử được đặt như sau ( **int Count[n]** ), ban đầu ta xét Count[0] = Count[1] = … = Count[n-1] = 0.
* Bước 3: Tạo 1 vòng lặp đối với biến **int i**. Vòng lặp được bắt đầu áp dụng từ bước 4 cho tới bước 5, biến **i** sẽ được lặp từ **n-1** cho tới **i=1**.
* Bước 4: Tạo 1 vòng lặp cho biến **int j** bên trong vòng lặp cho biến **i**. **j** đi từ **i-1** cho tới khi **i=0**.
* Bước 5: Chúng ta so sánh **rc[i].Key và rc[j].Key**, nếu **rc[i].Key < rc[j].Key** thì **Count[j]+=1**. Ngược lại, nếu **rc[i].Key rc[j].Key** thì **Count[i]+=1**.
* Bước 6: Sau khi vòng lặp biến **i** ở trên kết thúc. Chúng ta đã thu được một mảng **Count[n]** có các giá trị đã được thay đổi. Cuối cùng, tạo 1 Record **result[n]** có **n** phần tử. Tạo 1 vòng lặp cho **k** từ **0 -> n**. Ta đặt **result[Count[k]] = rc[k]**.
* Bước 7: Kết thúc thuật toán, ta thu được 1 mảng **result[n]** đã được sắp xếp từ mảng **rc[n]** ban đầu.
* Thuật toán này sắp xếp dựa theo 1 mảng phụ, mảng phụ này có chức năng sẽ lưu lại vị trí xuất hiện của các phần tử trong mảng gốc dựa theo giá trị của chúng so với nhau.

1. **Thuật toán Shell Sort**

* Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ).
* Bước 2: Tạo biến **int x= n/2**. Sau đó tạo 1 mảng **int h[]** có **n/2** phần tử sao cho **h[0] =1, h[1]=2,…h[x-1]=n/2**. Mảng **h** này sẽ lưu các khoảng cách của các phần tử sẽ được so sánh với nhau trong mảng **rc[]**.
* Bước 3: Tạo vòng lặp với biến **s**, **s** đi từ (**= x**) cho tới **0**.
* Bước 4: Đặt biến **temp = h[s]**. Và tiếp theo ta sẽ tạo 1 vòng lặp với biến **j**, trong đó **j** đi từ temp cho tới **n-1**.
* Bước 5: Trong vòng lặp cho biến **j**, ta đặt biến **int i=j-temp**, **int K = rc[j].Key, Record R = rc[j]**.
* Bước 6: Nếu **rc[i].Key K**, ta thực hiện trực tiếp bước 8.
* Bước 7: Đặt **rc[i+temp] = rc[i]**. Nếu **i>0** thì quay lại bước 6.
* Bước 8: Đặt **rc[i+temp] = R**. Sau đó quay lại bước 5.
* Thuật toán này hoạt động bằng việc hoán đổi các phần tử cách nhau 1 khoảng cách nào đó. Sau khi hoàn thành 1 lượt sắp xếp, khoảng cách sẽ giảm đi 1 và nếu phù hợp điều kiện sẽ lại hoán đổi các phần tử theo khoảng cách mới. Khi sắp xếp xong thì khoảng cách sẽ trở về 0.

1. **Thuật toán Selection Sort**

* Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ).
* Bước 2: Tạo 1 vòng lặp với biến **j**, trong đó j đi từ **n-1** tới **1**.
* Bước 3: Tiếp theo, ta tìm số lớn nhất trong khoảng từ **rc[j]** cho tới **0**. Bằng cách đặt 1 biến có tên là **max\_idx =j**.
* Bước 4: Tạo 1 vòng lặp con có biến **i** trong vòng lặp của biến **j**, ( **i = j-1** cho tới **0** ). Nếu ta nhận thấy có **rc[i].Key>rc[max\_idx]** thì đi tới bước 5.
* Bước 5: Hoán đổi **rc[max\_idx]** và **rc[i]**, sau đó đặt **max\_idx = i**.
* Thuật toán này hoạt động dựa trên việc kiểm tra từ vị trí n-1 tới vị trí 0. Trong quá trình kiểm tra đó, nó sẽ liên tục cập nhật vị trí có giá trị cao nhất và sau đó là hoán đổi vị trí đang xét và vị trí có giá trị cao nhất đó. Cứ liên tục làm như thế cho tới khi vị trí được xét trở thành 1.

1. **Thuật toán Quick Sort**

* Bước 1: Tạo 1 mảng có liểu dữ liệu là **Record** có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ), tạo 1 **Stack** có tên là **s**. Stack này sẽ có chức năng lưu trữ các node, mỗi **node** đều có chứa 2 giá trị là ( **int l**, **int r**).
* Bước 2: Đặt biến l=0, r=n-1, s rỗng và M=1.
* Bước 3: Đặt i=l, j = r+1 và K = rc[l].Key
* Bước 4: Chúng ta tăng i lên 1, sau đó kiểm tra nếu rc[i].Key < K thì lặp lại bước này.
* Bước 5: Chúng ta giảm j đi 1, sau đó kiểm tra nếu rc[j].Key > K thì lặp lại bước này.
* Bước 6: Kiểm tra xem nếu j i thì tiến hành hoán đổi rc[l] và rc[j] và sau đó tới bước 8.
* Bước 8: Nếu r-j j-l > M thì tiến hành push(node(j+1,r)) vào stack s và quay lại bước 3. Nếu j-l > r-j > M, tiến hành push(node(l,j-1)) vào stack s và quay lại bước 3. Nếu r-j > M j-l, ta gán l=j+1 rồi quay lại bước 3. Nếu j-l > M r-j thì gán r = j- 1 rồi quay lại bước 3.
* Bước 9. Kiểm tra xem stack có rỗng hay không, nếu có thì thoát khỏi vòng lặp và kết thúc thuật toán.
* Bước 10. Nếu stack không rỗng thì tiến hành việc đặt Node\* p = s.pop() . Đặt l = p->l, r= p->r. sau đó lại quay lại bước 3.
* Thuật toán trên sắp xếp dựa trên việc hoán đổi các giá trị dựa theo vị trí từ các khoảng từ l tới r khác nhau. Trong đó khoảng l và r được cập nhật liên tục. Nếu khoảng nào phù hợp với điều kiện mà chưa được xét thì sẽ được đẩy vào đỉnh stack nhằm phục vụ cho việc hoán đổi trong các khoảng sau này. Khi thuật toán kết thúc thì đồng nghĩa với việc stack sẽ rỗng.

1. **Thuật toán Two-Way Merge Sort**

* Bước 1: Tạo 2 mảng **Record** được đặt tên là **X** và **Y**, lần lượt có **m,n** phần tử. Lưu ý 2 mảng này đã được sắp xếp có trật tự. Đồng thời ta cũng sẽ tạo thêm 1 mảng **Record** được đặt tên là **Z** có **m+n** phần tử, mảng **Z** này sẽ là kết quả sắp xếp được khi trộn 2 mảng **X** và **Y** trên.
* Bước 2: Đặt các biến **i = j = k = 0**.
* Bước 3: So sánh nếu **X[i].Key Y[i].Key** thì đi tới bước 4. Nếu không thì đi tới bước 6.
* Bước 4: Đặt **Z[k] = X[i], k = k+1, i=i+1**, nếu **i m** thì quay lại bước 3.
* Bước 5: Đặt **Z[k], … ,Z[m+n] = Y[j], … , Y[n]** và kết thúc thuật toán.
* Bước 6: Đặt **Z[k] = Y[j], k = k+1, j = j+1**, nếu **j n** thì lại quay lại bước 3.
* Bước 7: Đặt **Z[k], … ,Z[m+n] = X[i], … , X[m]** và kết thúc thuật toán.
* Thuật toán này được thực hiện thông qua 2 mảng đã được sắp xếp sẵn, sau đó trộn 2 mảng trên theo trật tự để tạo nên được 1 mảng mới sẽ được sắp xếp.

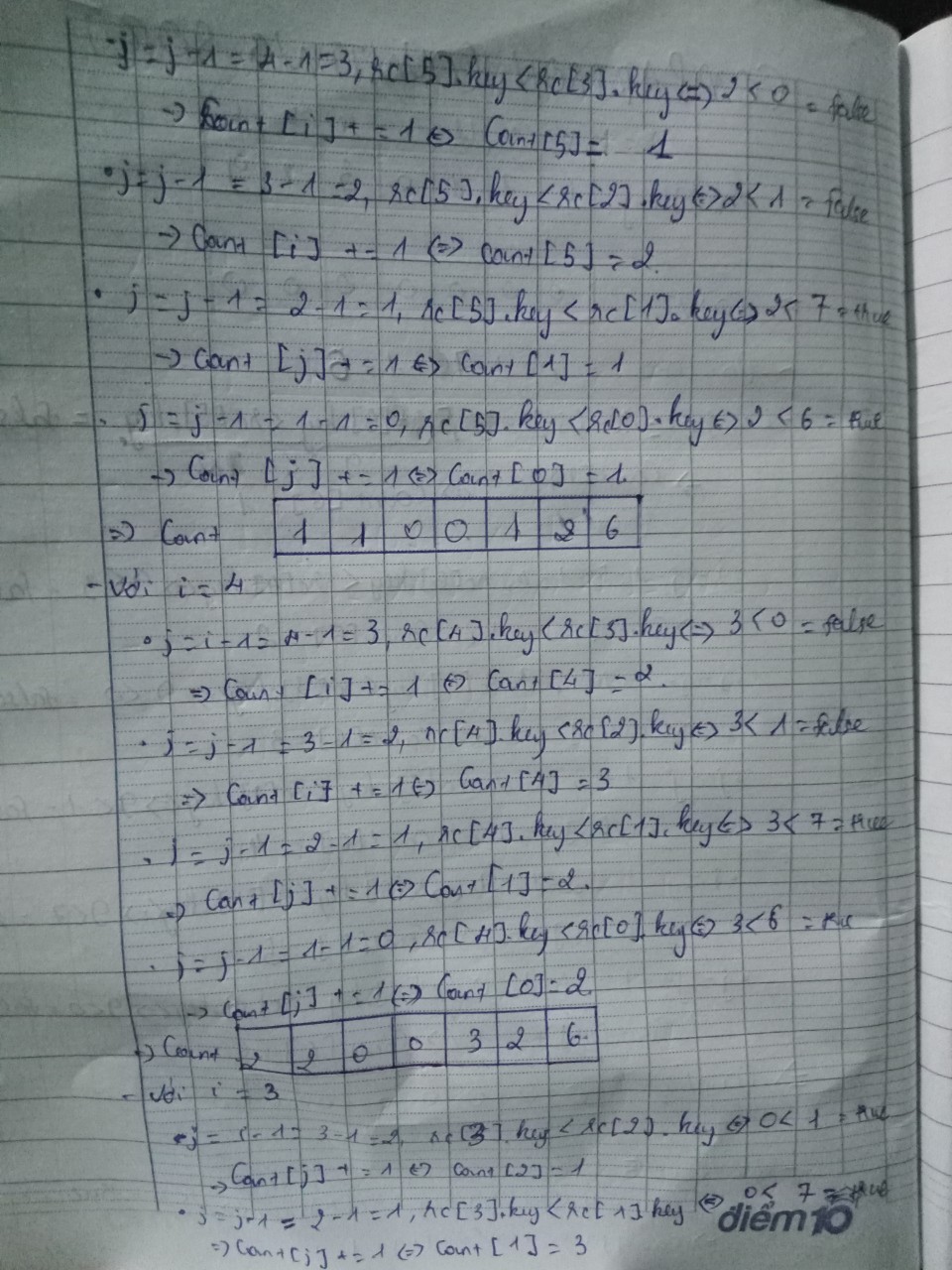
1. **Straight Insertion Sort**

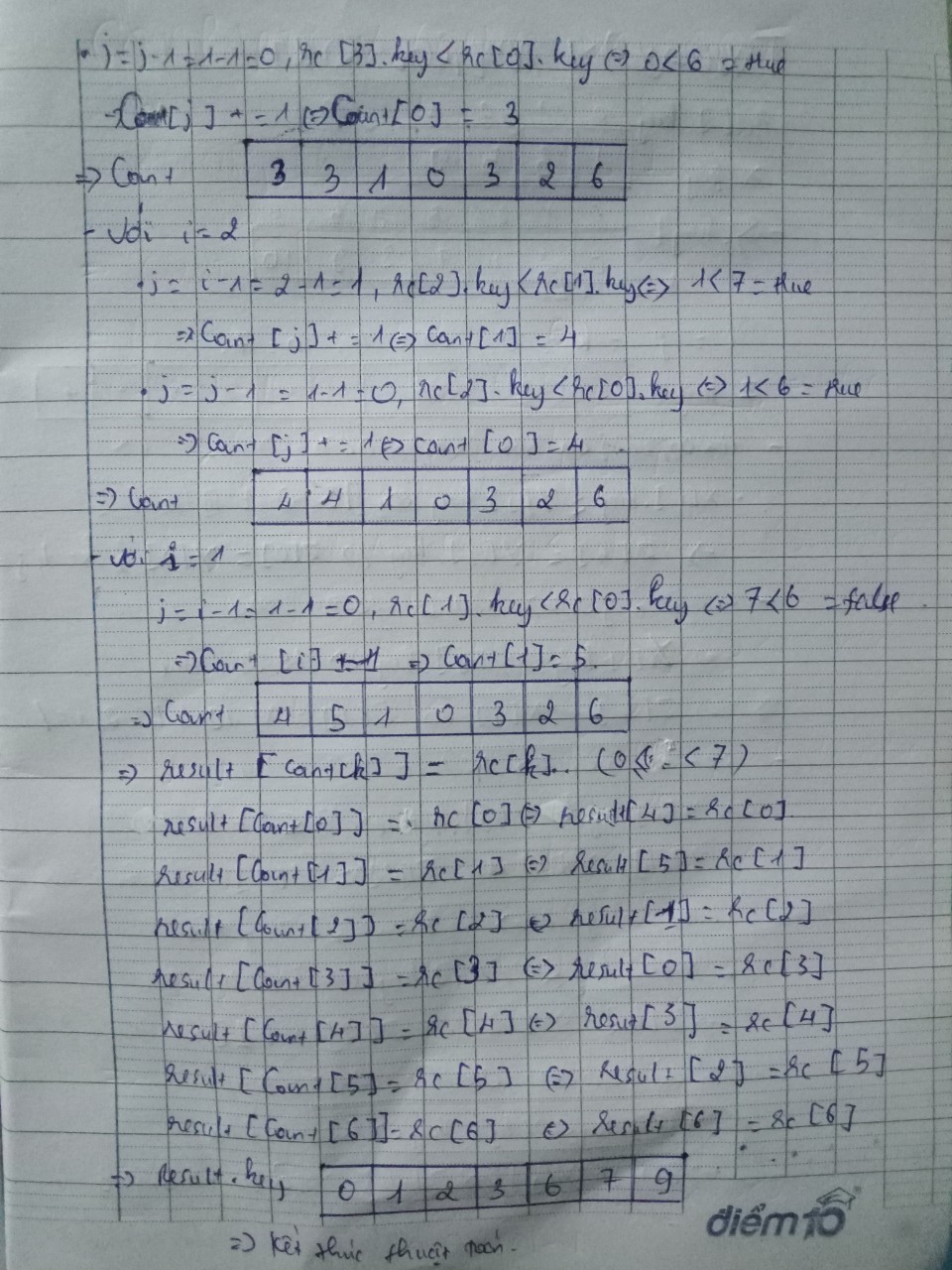
* Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ).
* Bước 2: Tạo 1 vòng lặp cho biến **j**, trong đó **j** đi từ **1** đến **n-1**, sau khi kết thúc vòng lặp thì kết thúc luôn thuật toán. Vòng lặp này áp dụng từ bước 3 tới bước 6.
* Bước 3: Đặt biến **i = j-1, k = rc[j].Key, R = rc[j]**.
* Bước 4: Nếu **k > rc[i].Key** thì chúng ta thực hiện bước 6.
* Bước 5: Đặt **rc[i+1] = rc[i]**, sau đó **i = i-1**, nếu **i>0** thì quay lại bước 4.
* Bước 6: Đặt **rc[i+1] = R**.
* Thuật toán trên hoạt động theo việc chèn vào 1 vị trí nào đó qua 1 cách kiểm tra các phần tử ở trước nó. Nếu tại vị trí cuối cùng nào phù hợp với điều kiện so sánh với vị trí đang xét thì giá trị phần tử đang xét sẽ chèn trực tiếp vào vị trí đó. Thuật toán sẽ kết thúc khi vị trí đang xét là **n-1** đã được hoàn thành.

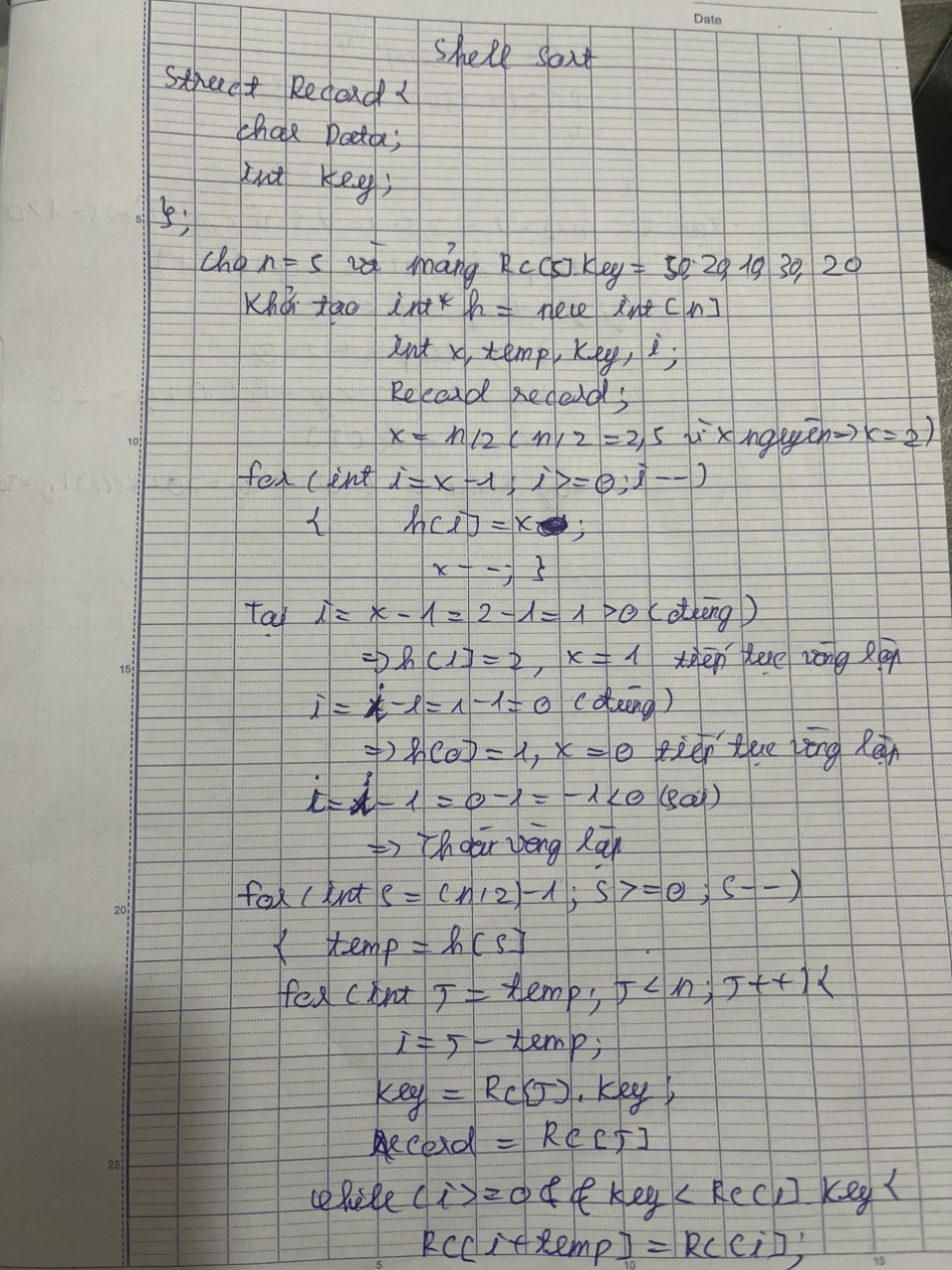
1. **Bubble Sort**

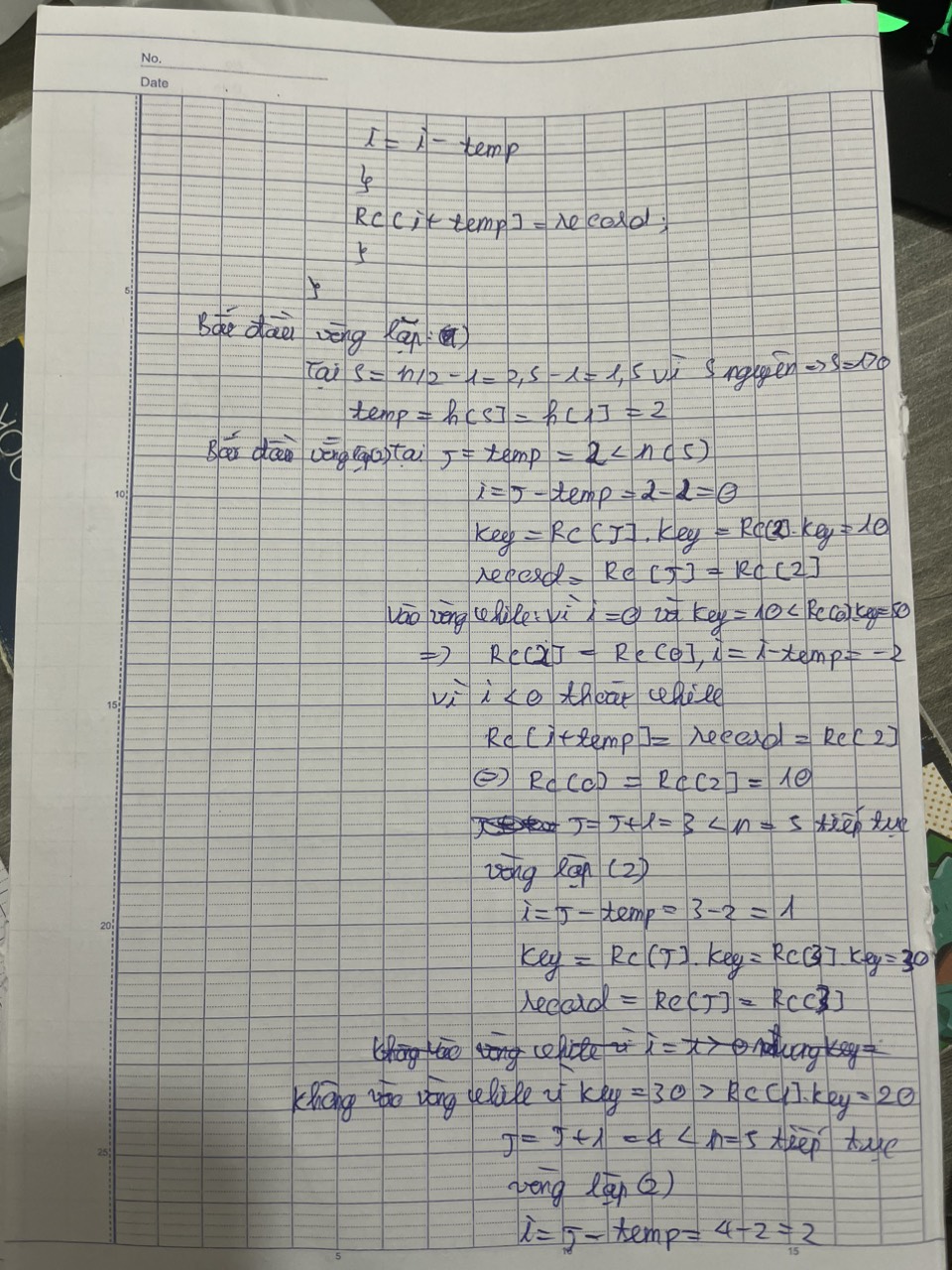
* Bước 1: Tạo 1 mảng có liểu dữ liệu là Record có **n** phần tử được đặt như sau ( **Record rc[n]** ), mỗi phần tử Record chứa 2 kiểu dữ liệu ( **char Data; int Key;** ).
* Bước 2: Tạo biến bound = n-1, (bound là vị trí cao nhất mà Record sẽ được sắp xếp tới, bound sẽ được cập nhật liên tục trong quá trình thực hiện thuật toán).
* Bước 3: Tạo vòng lặp cho biến j, đặt biến t=0, thực hiện vòng lặp này từ bước 4, trong đó j đi từ 0 cho tới bound-1.
* Bước 4. So sánh nếu rc[j].Key > rc[j+1].Key thì hoán đổi rc[j] và rc[j+1], sau đó đặt t=j.
* Bước 5: Kiểm tra xem nếu t=0 thì kết thúc luôn thuật toán, nếu không thì ta gán bound=t và quay lại vòng lặp tại bước 3.
* Thuật toán này thực hiện theo nguyên tắc đưa dần phần tử lớn nhất ra sau vị trí cao nhất, sau đó lại cập nhật lại vị trí cao nhất rồi đưa phần tử lớn nhất khác ra sau vị trí cao nhất mới cập nhật. Cứ tiếp tục làm như thế cho đến khi vị trí cao nhất không còn cập nhật được nữa thì dừng thuật toán.

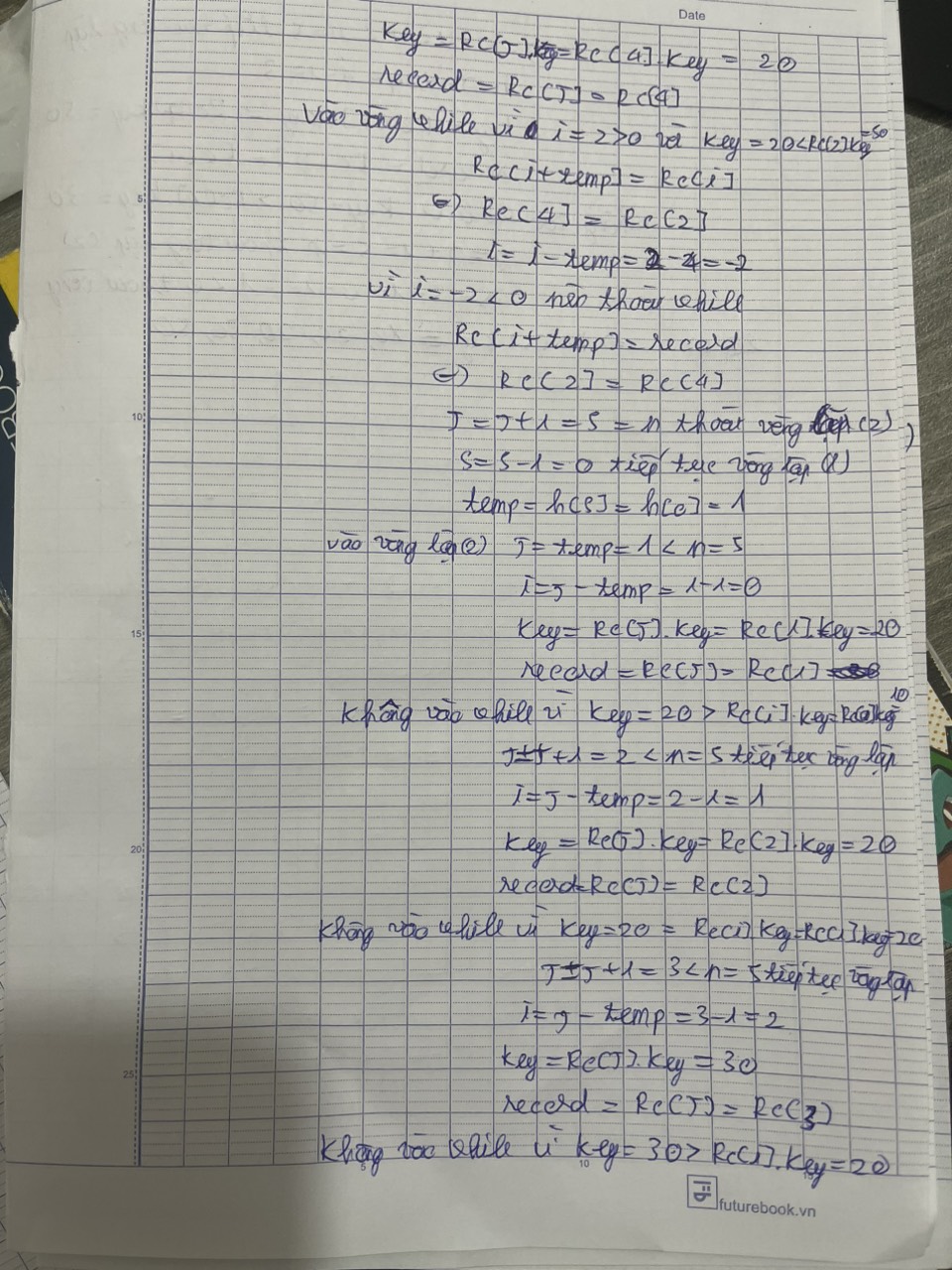
1. **Chạy tay ví dụ các thuật toán**
2. **Thuật toán Comparison Counting Sort**

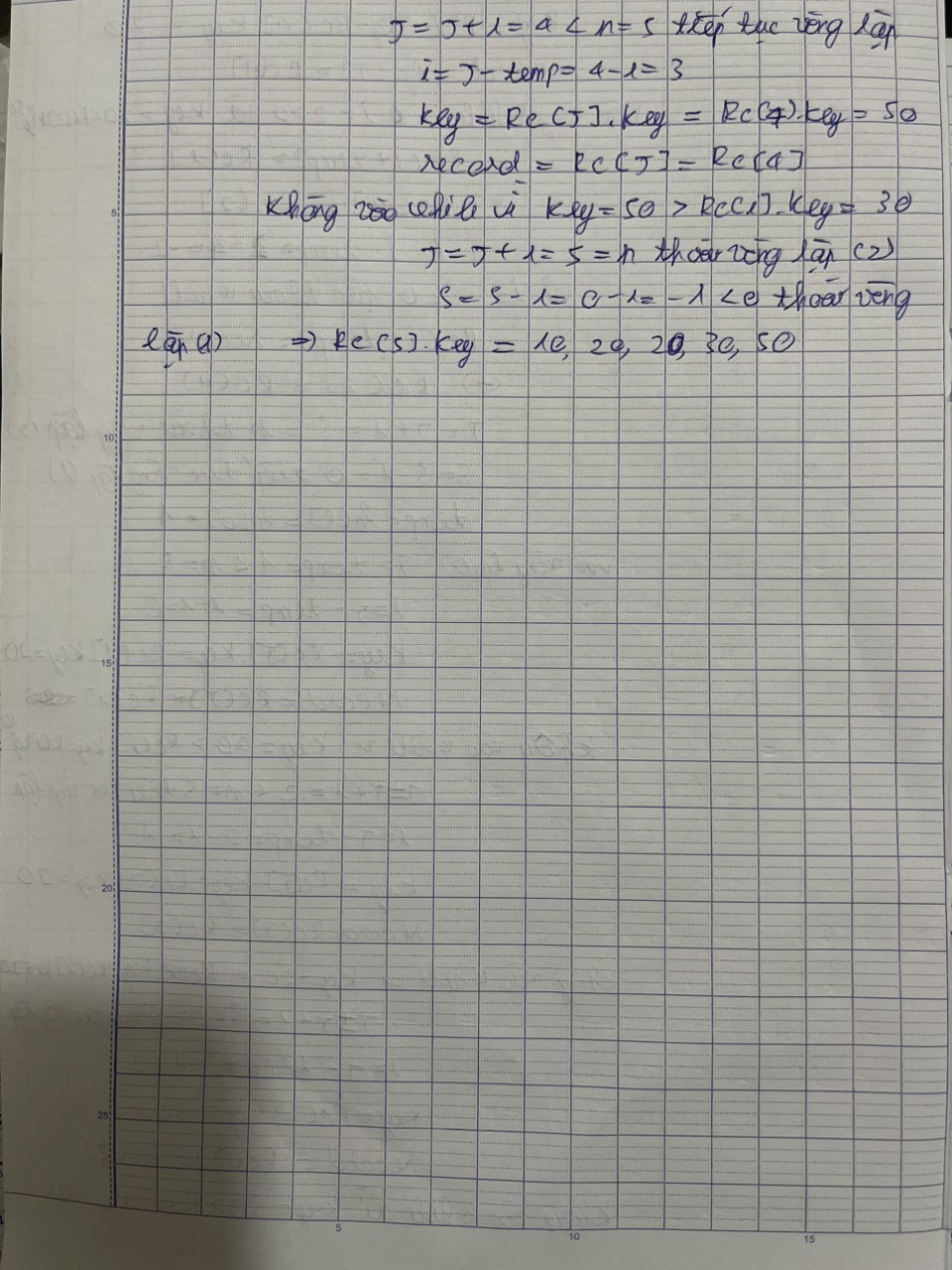




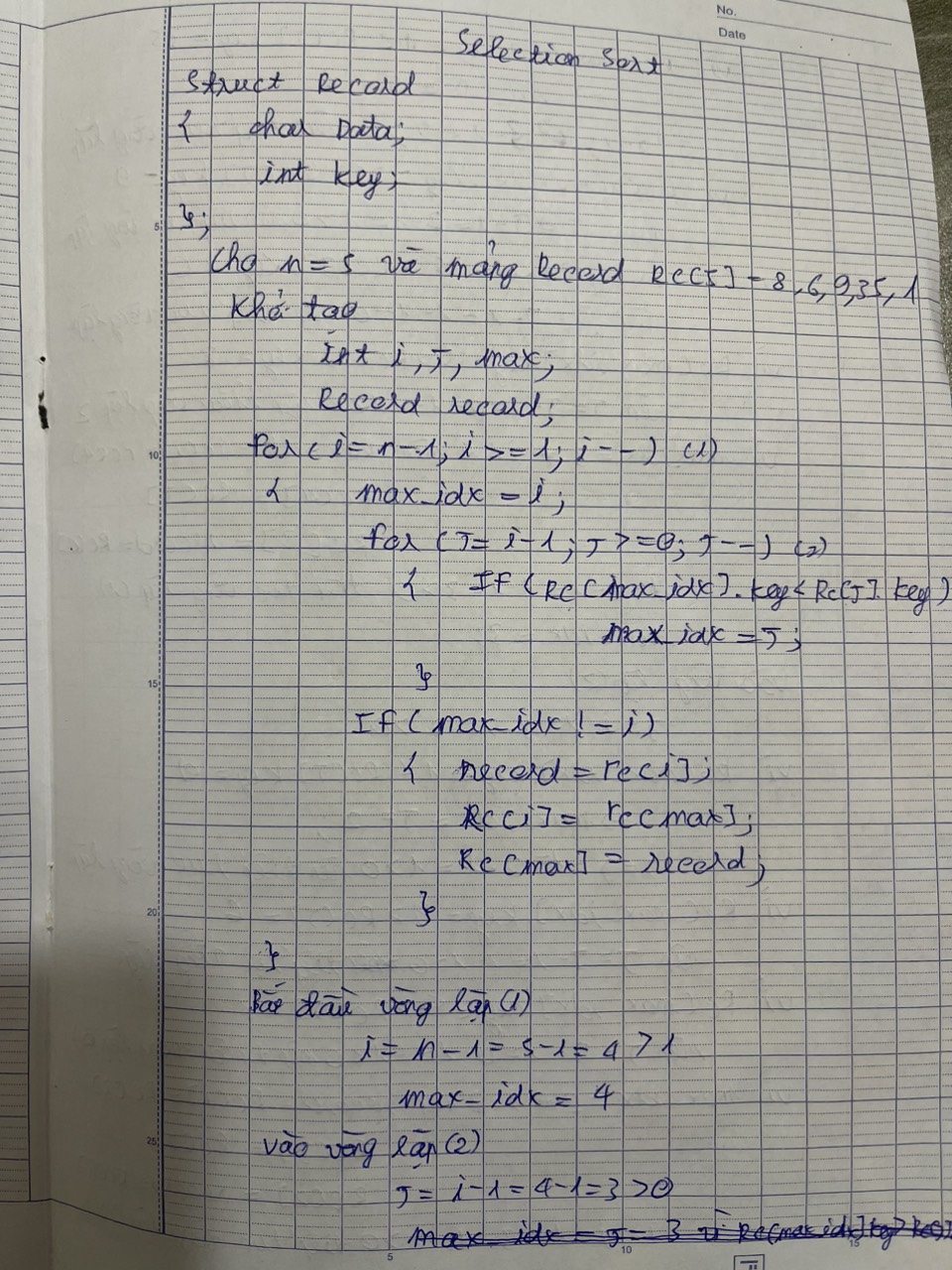
1. **Thuật toán Shell Sort**

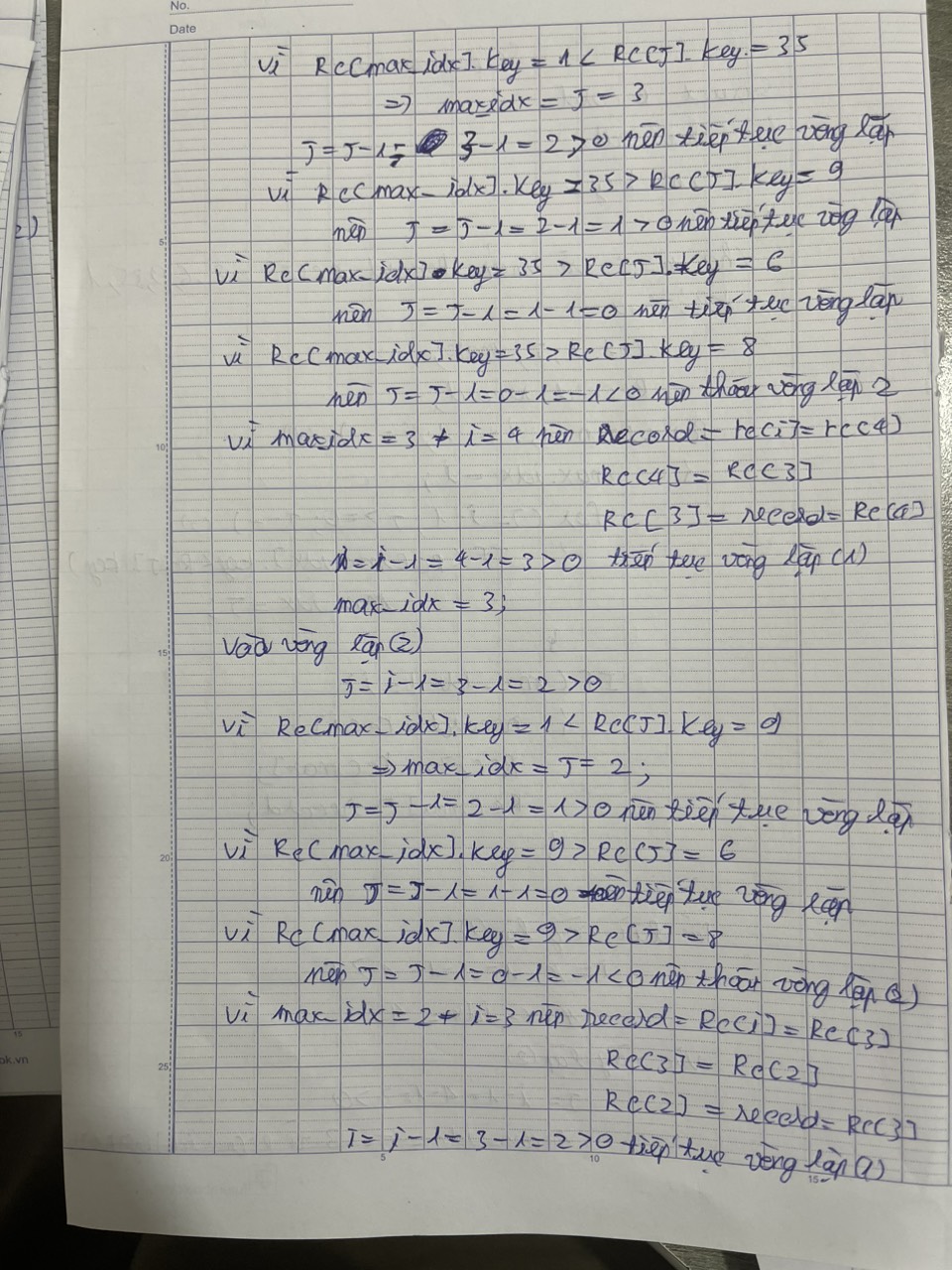
****

****



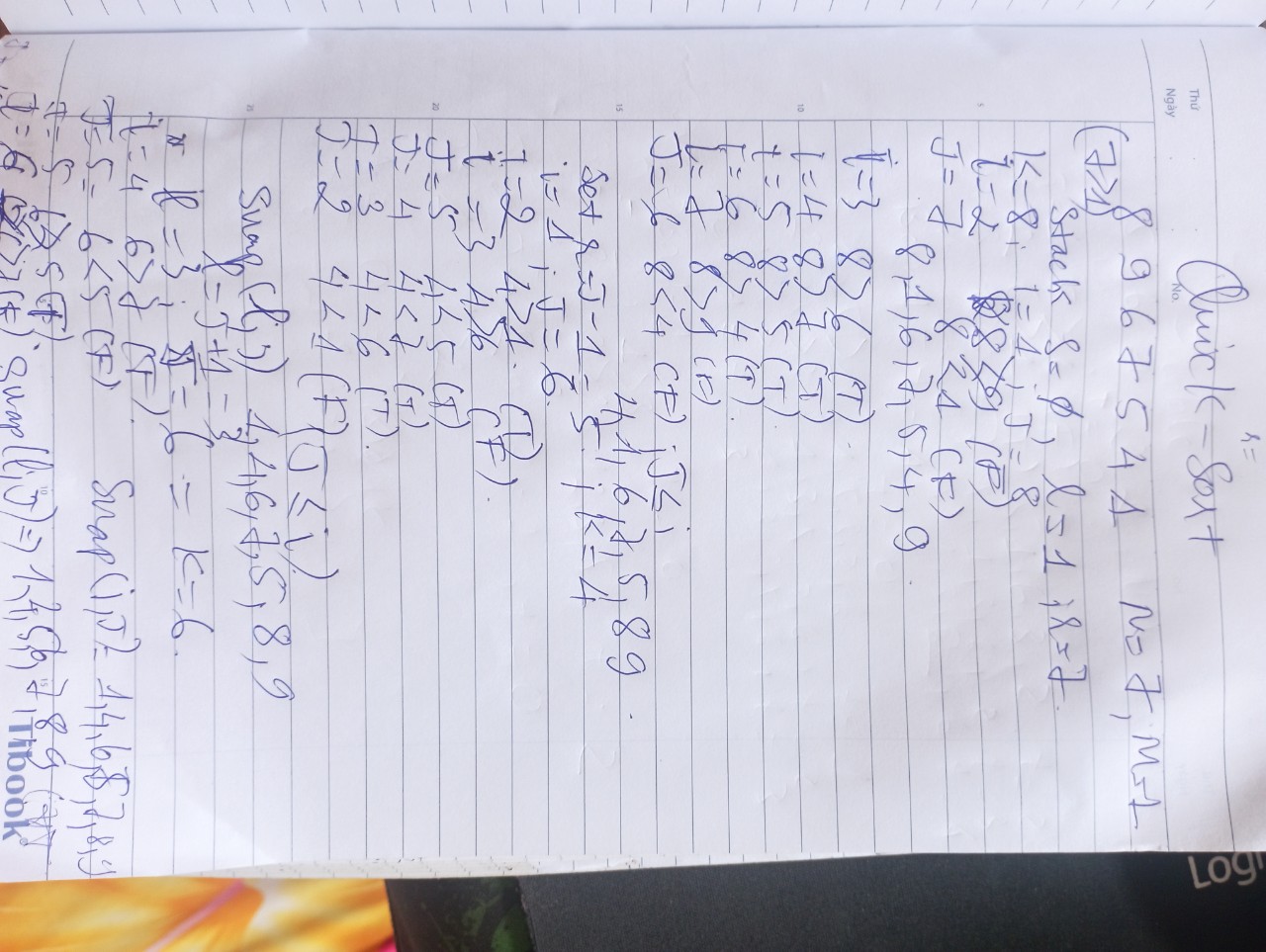
1. **Thuật toán Selection Sort**



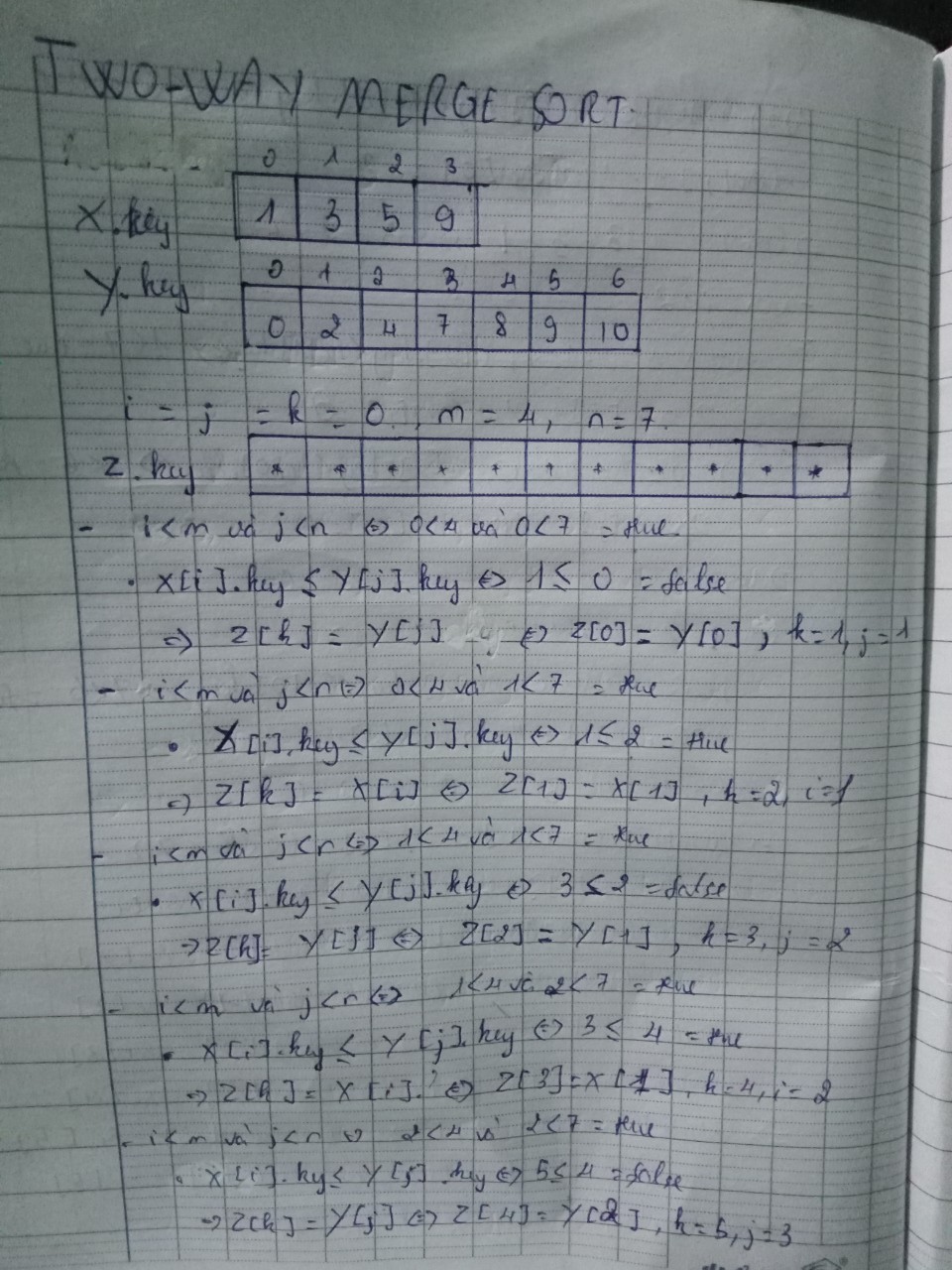


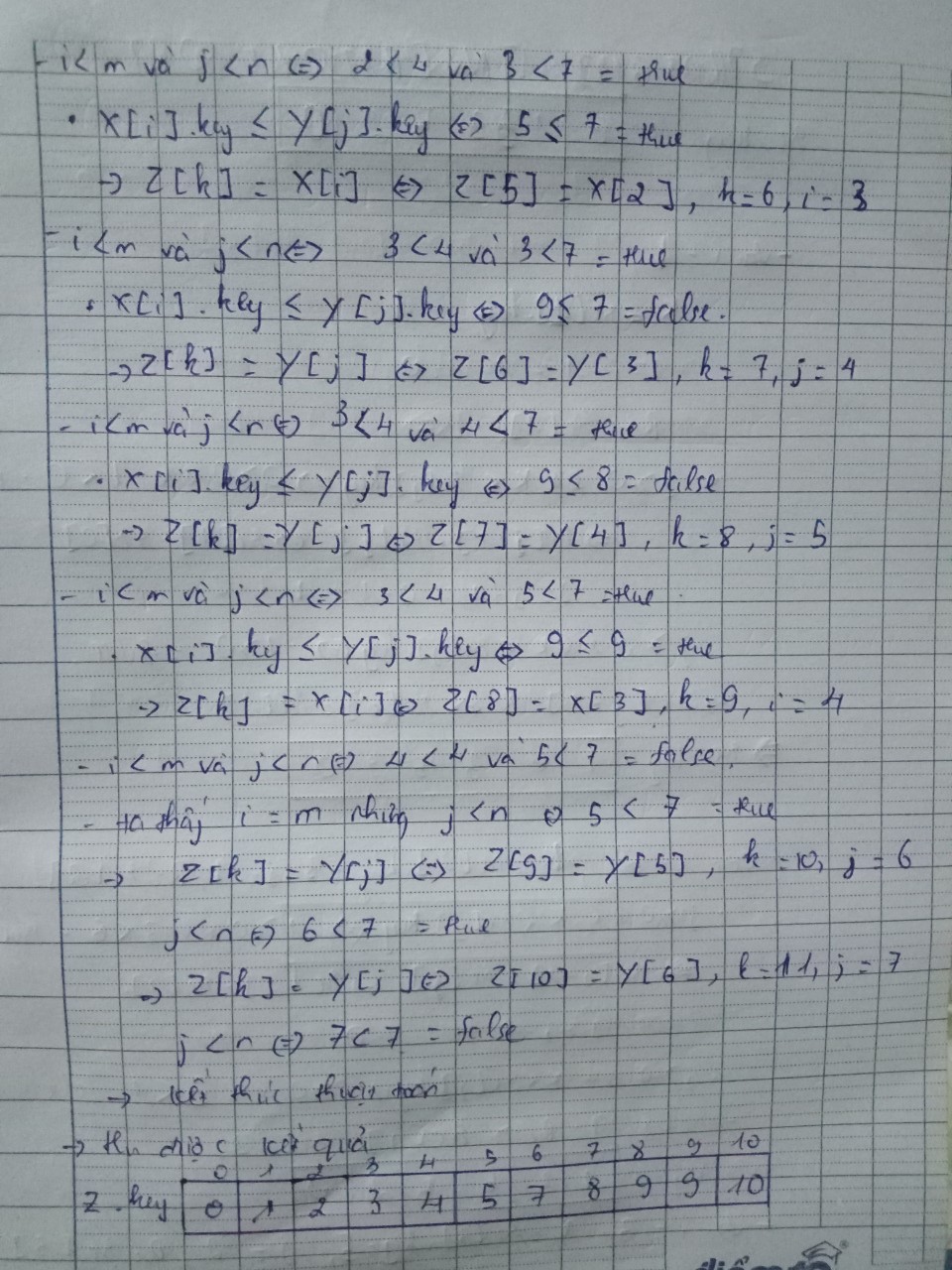


1. **Thuật toán Quick Sort**

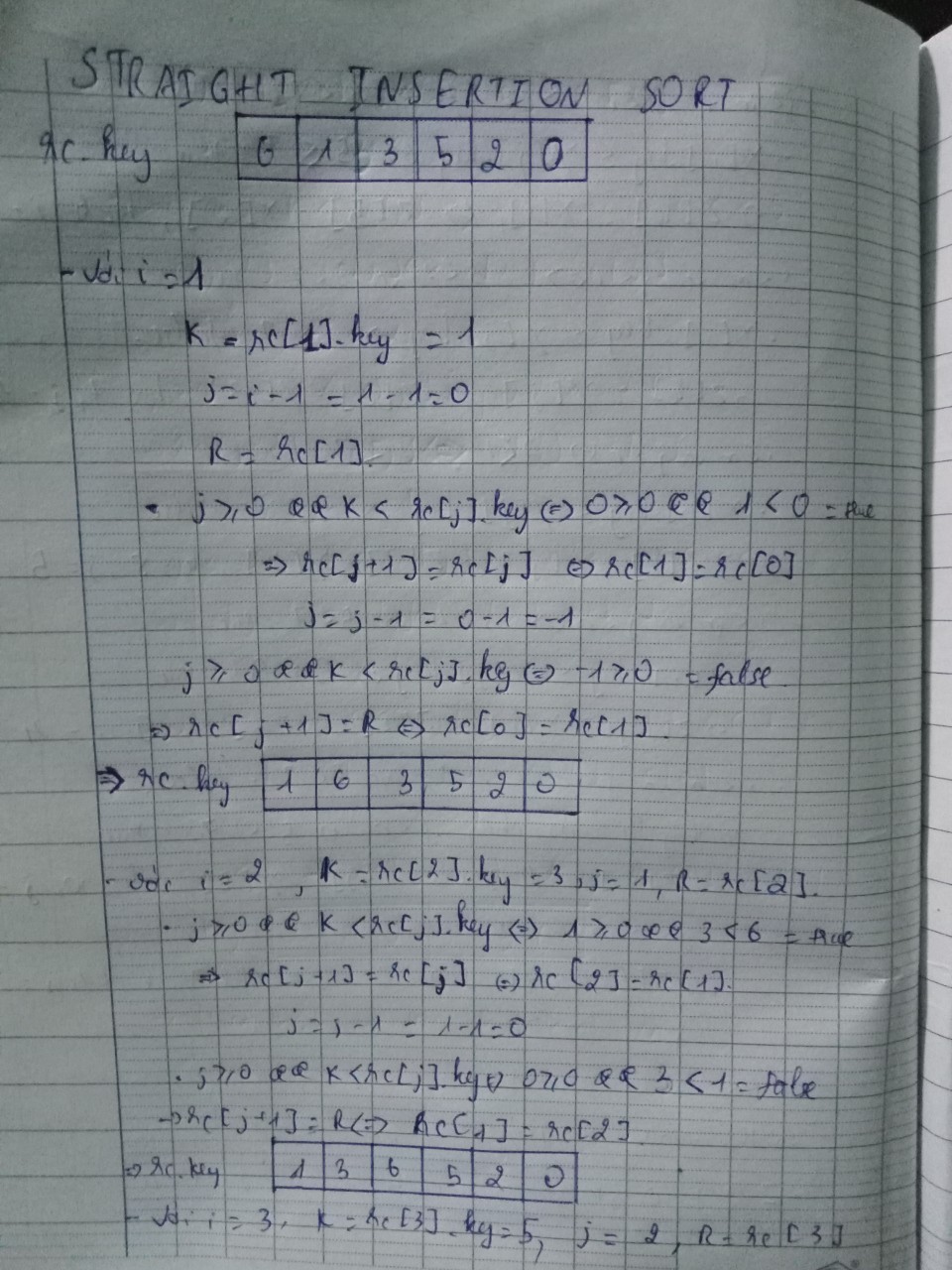
****

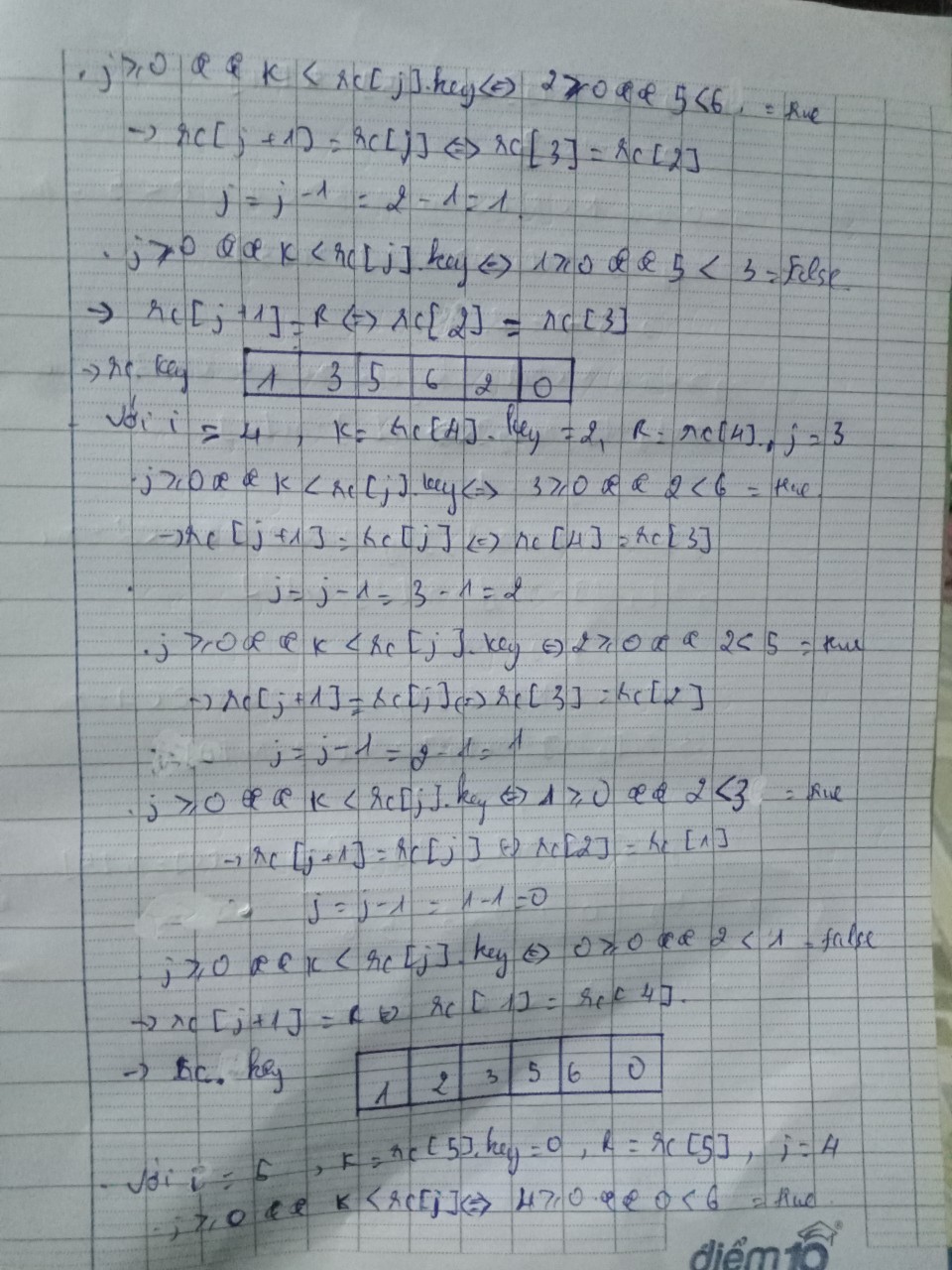
1. **Thuật toán Two-Way Merge Sort**

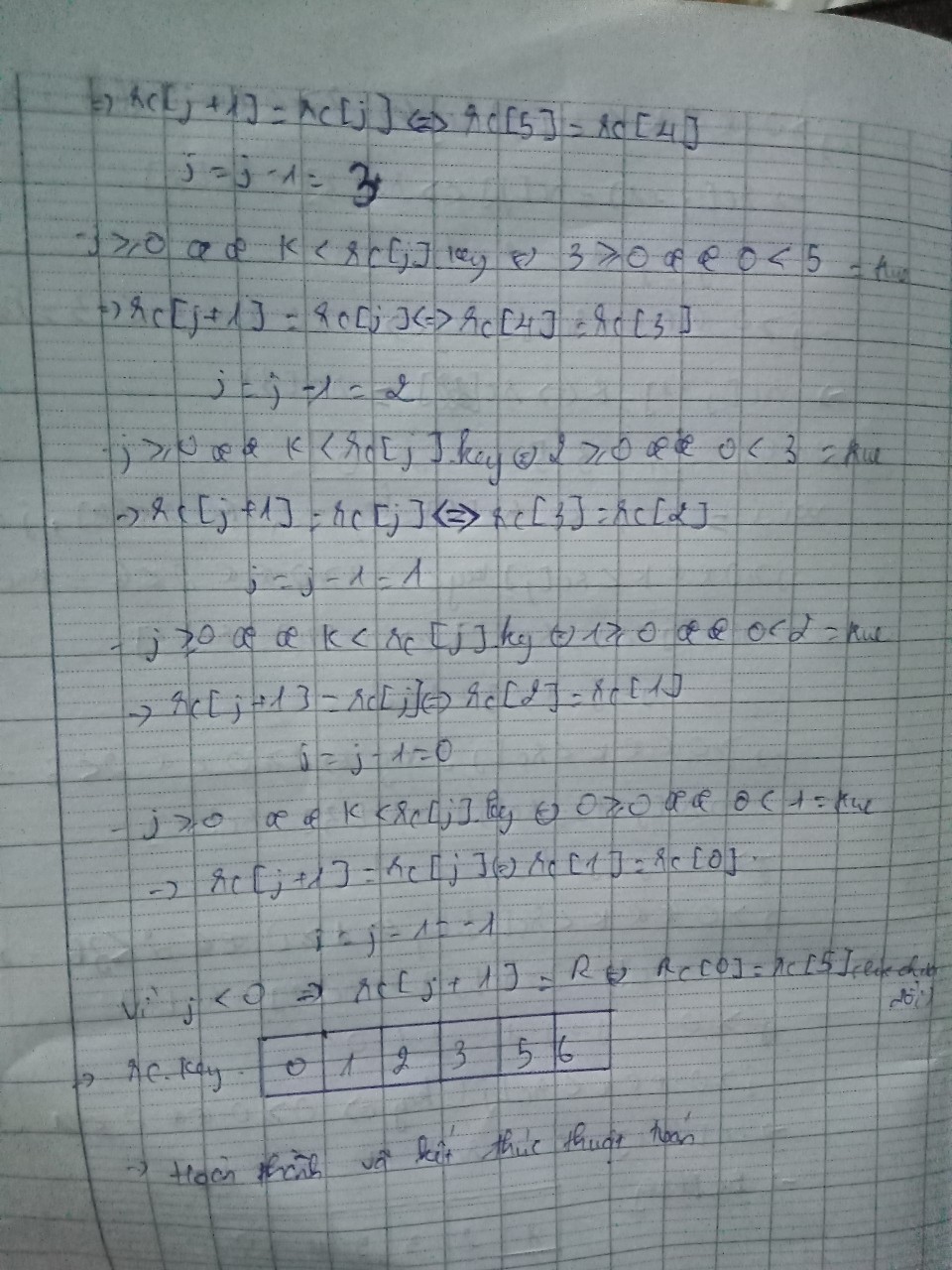
****

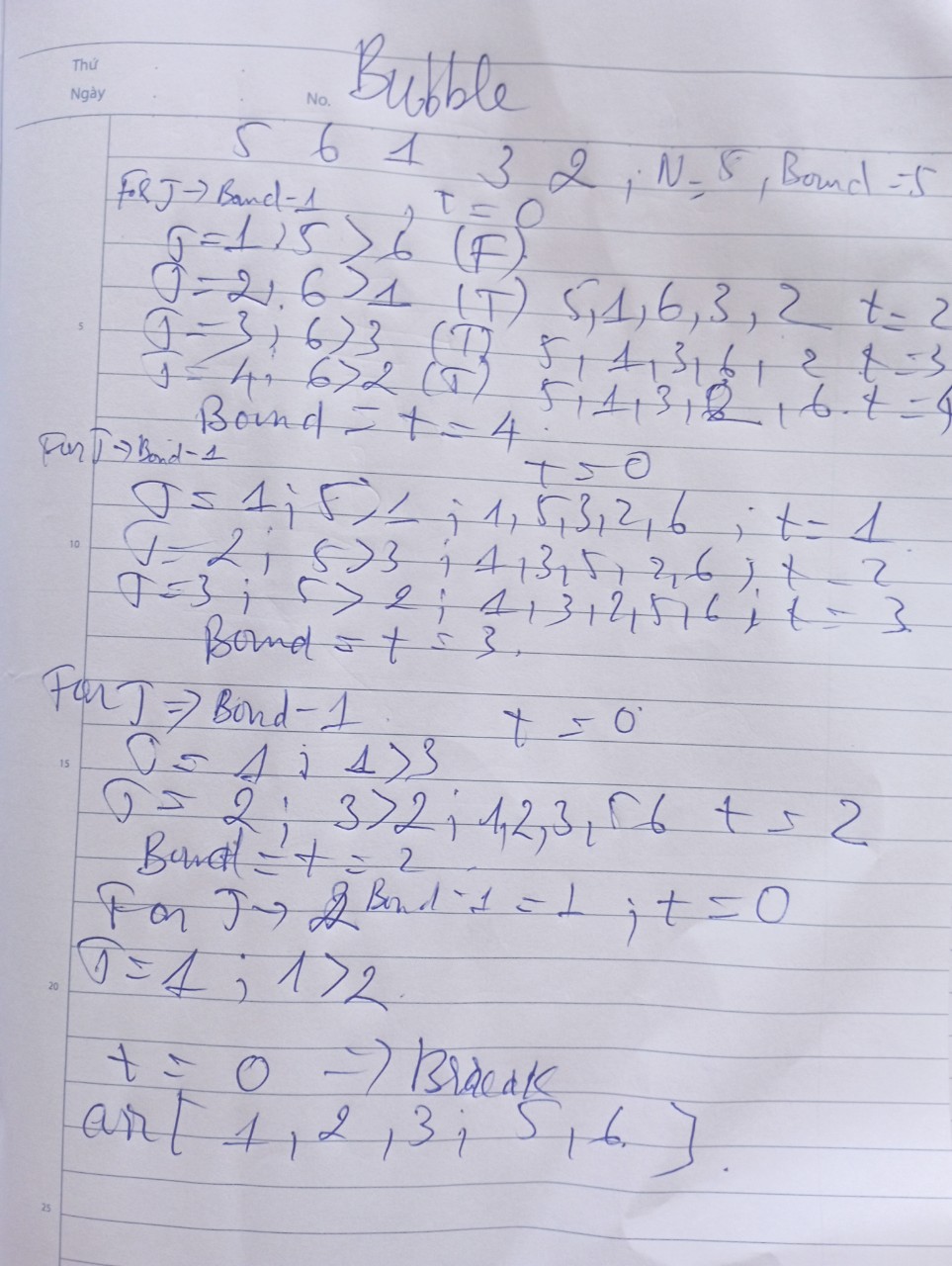


1. **Straight Insertion Sort**







1. **Bubble Sort**

**Phần 3: KẾT LUẬN**

* Qua bộ môn Cấu trúc dữ liệu và giải thuật nói chung, đồng thời qua việc làm đồ án về thuật toán Sort sử dụng graphics.h nói riêng. Nhóm chúng em cũng học được cách làm việc theo nhóm sao cho hiệu quả, thêm vào đó là học được các thuật toán mới và cũng học được các hướng giải quyết khác cho các thuật toán đó, giúp mở mang thêm kiến thức. Ngoài ra, nhóm còn học được cách tự học, và chọn lọc kiến thức thông qua việc tìm kiếm các thông tin mà mình cần được thực hiện.