**ĐẠI HỌC QUỐC GIA TP HCM**

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

🙡🕮🙣

**BÁO CÁO SEMINAR CUỐI KỲ**



ĐỀ BÀI: SỐ 1

**MINH HỌA**

**CÁC THUẬT TOÁN SẮP XẾP**

Lớp: Cấu trúc dữ liệu và giải thuật – **IT003.L21.HTCL**

Nhóm thực hiện: **Nhóm 9**  Viết bằng ngôn ngữ: **C++**

Gồm các thành viên:

20521228 Bùi Đức Duy

20520296 Tôn Nữ Tú Quyên

20520322 Nguyễn Thị Mỹ Trân

# **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT ĐÃ SỬ DỤNG**

1. **Cấu trúc dữ liệu mảng động**

**Tại sao sử dụng cấu trúc dữ liệu mảng động ? Ý nghĩa của việc sử dụng.**



Trong chương trình, người dùng yêu cầu sắp xếp lại các phần tử trong mảng theo chiều tăng hoặc giảm dần, thì việc sử dụng cấu trúc dữ liệu mảng động giúp truy xuất nhanh đến các phần tử và giúp sử dụng bộ nhớ linh hoạt hơn.

**Ưu điểm:**

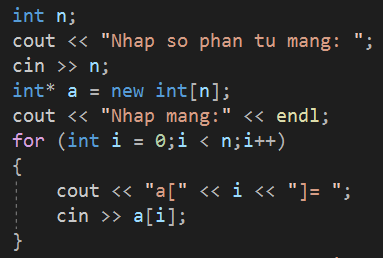
* Dùng mảng động, giúp tính toán được kích thước của mảng động trước khi cấp phát. tránh tình trạng lãng phí bộ nhớ.
* Dễ hiểu, dễ sử dụng, dễ sửa chữa
* Truy cập nhanh đến các phần tử.
* Dễ dàng tìm kiếm và sắp xếp

**Nhược điểm:**

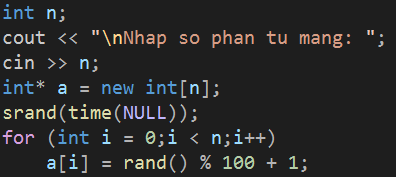
* Khó khăn trong quá trình thêm, sửa, xóa.

Ở bài báo cáo này sử dụng hai loại tạo mảng động là**: Mảng nhập thủ công** và **Mảng được tạo ngẫu nhiên** để tạo mảng số nguyên thực hiện các thuật toán sắp xếp.

**Nhập mảng thủ công:** Mảng sẽ được nhập bằng tay.



**Mảng được tạo ngẫu nhiên:** chương trình sẽ tạo ra một mảng số nguyên bất kì.



1. **Thuật toán sắp xếp đổi chỗ trực tiếp:**

**Giải thuật chính của sắp xếp đổi chỗ trực tiếp (Interchange sort) là:** Xuất phát từ đầu dãy, tìm tất các các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy

Việc sử dụng thuật toán sắp xếp Interchange sort: Số lượng các phép so sánh xảy ra không phụ thuộc vào tình trạng của dãy số ban đầu. Số lượng phép hoán vị thực hiện tùy thuộc vào kết quả so sánh. Trong chương trình, thuật toán được áp dụng sắp xếp dãy số tang hoặc giảm theo thuật toán sắp xếp đổi chỗ trực tiếp:

**Cài đặt thuật toán:**

void InterchangeSort(int a[], int N ) {

int i, j;  
for (i = 0 ; i<N-1 ; i++)

for (j =i+1; j < N ; j++)  
if(a[j ]< a[i]) *//Thỏa 1 cặp nghịch thế* Swap(a[i], a[j]);

}

**Đánh giá thuật toán:**

Trường hợp xấu nhất: số lần so sánh : n(n-1)/2, số lần hoán vị: 0.

Trường hợp tốt nhất: số lần so sánh n(n-1)/2, số lần hoán vị: 0.

Độ phức tạp O(n).

1. **Thuật toán sắp xếp chọn trực tiếp:**

**Giải thuật chính của sắp xếp chọn trực tiếp (Selection sort) là:** Chọn phần tử nhỏ nhất trong n phần tử của dãy hiện hành ban đầu. Đưa phần tử này về vị trí đầu dãy. Xem dãy hiện hành chỉ còn n-1 phần tử. Bắt đầu từ vị trí thứ 2. Tiếp tục chọn phần tử nhỏ nhất đưa về đầu dãy. Lặp lại quá trình trên cho đến khi dãy hiện hành chỉ còn 1 phần tử.

**Cài đặt thuật toán:**

void SelectionSort(int a[],int n)

{

for (int i = 0; i < n-1; i++) *// i là chỉ số đầu tiên trong dãy hiện hành*

{ int min = i; *// min là chỉ số phần tử nhỏ nhất trong dãy hiện hành, gán min = i*

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

{ if (a[min] > a[j])

min = j; } *// lưu vị trí phần tử hiện nhỏ nhất*

if (min != i)

{ swap (a[min], a[i]); }

}

}

**Đánh giá thuật toán:**

Trường hợp xấu nhất:O(n\*n).

Trường hợp tốt nhất: O(n\*n).

Trường hợp trung bình: O(n\*n).

1. **Thuật toán sắp xếp chèn trực tiếp (Insertion sort).**

**Giải thuật chính của sắp xếp chèn trực tiếp (Selection sort) là:**

Giả sử có một dãy a0, a1, …an trong đó i phần tử đầu tiên a0, a1, …ai-1 đã có thứ tự.

Tìm cách chèn phần tử ai vào vị trí thích hợp của đoạn đã được sắp xếp trước đó để có dãy mới a0, a1, …ai trở nên có thứ tự.

Vị trí thích hợp này chính là vị trí giữa hai phần tử ak -1và ak thỏa ak-1 < ai < ak (1<=k<=i)

**Cài đặt thuật toán:**

void InsertionSort(int a[], int n)

{

int pos, i,

int x; *// Dùng để lưu a[i] để tránh bị ghi đè khi dời chỗ các phần tử*

for (i = 1; i < n; i++) *// Đoạn a[0] đã sắp xếp*

{

x = a[i];

pos = i - 1;

while ((pos >= 0) && (a[pos] > x)) *// Tìm vị trí chèn x*

{

a[pos + 1] = a[pos]; //kết hợp dời chỗ các phần tử sẽ đứng sau x trong dãy mới

pos--;

}

a[pos + 1] = x; // chèn x vào dãy

}

}

**Đánh giá thuật toán:**

Trường hợp xấu nhất:O(n\*n).

Trường hợp tốt nhất: O(n).

Trường hợp trung bình: O(n\*n).

1. **. Thuật toán sắp xếp nổi bọt (Bubble sort):**

**Giải thuật chính của sắp xếp nổi bọt (Bubble sort) là:** Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo. Ở lần xử lý thứ i có vị trí đầu dãy là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét

**Cài đặt thuật toán:**

void BubbleSort(int a[],int n) {

int i, j;  
 for (i = 0 ; i<n-1 ; i++)  
 for (j =n-1; j >i ; j --)  
 if(a[j]< a[j-1*])// nếu sai vị trí thì đổi chỗ*  
 Swap(a[j], a[j-1]);  
}

**Đánh giá thuật toán:**

Trường hợp xấu nhất: số lần so sánh n(n-1)/2, số lần hoán vị : 0.

Trường hợp tốt nhất: số lần so sánh n(n-1)/2, số lần hoán vị: 0.

1. **. Thuật toán sắp xếp chèn nhị phân (Binary Insertion sort):**

**Giải thuật chính của sắp chèn nhị phân (Binary Insertion sort) là:** Binary Insertion Sort là cải tiến của Insertion Sort. Sử dụng tìm kiếm nhị phân để tìm vị trí thích hợp để chèn mục đã chọn ở mỗi lần lặp. Trong trường hợp chèn thông thường, việc sắp xếp lấy O(i) (ở lần lặp thứ i). Chúng ta có thể giảm nó thành O(logi) bằng cách sử dụng tìm kiếm nhị phân.

**Cài đặt thuật toán:**

void BinaryInsertionSort (int a[], int n)

{ int l, r, m, x;

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

{ l = 0;

r = i-1;

x = a[i]; *//lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử*

while (l <= r) *//tìm vị trí chèn x*

{ m = (l+r)/2; *//tìm vị trí thích hợp m*

if (a[m] > x) r = m-1;

else l = m+1;}

for (int j = i; j > l; j--)

a[j] = a[j-1]; *//dời các phần tử sẽ đứng sau x*

a[l] = x; } *//chèn x vào dãy*

}

**Đánh giá thuật toán:**

Trường hợp xấu nhất:O(n\*n).

Trường hợp tốt nhất: O(n).

Trường hợp trung bình: O(n\*n).

1. **Thuật toán sắp xếp Shaker sort:**

**Giải thuật chính của sắp xếp Shaker sort là:** Đây là một thuật toán sắp xếp dựa trên nguyên tắc đổi chỗ trực tiếp như Interchange sort, nhưng nó cải tiến hơn, né hết những “vấp ngã” của “đời trước”.

Trong mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau:

* Lượt đi: Đẩy phần tử nhỏ về đầu mảng.
* Lượt về: Đẩy phần tử lớn về cuối mảng.

Ghi nhận lại những đoạn đã sắp xếp nhằm tiết kiệm các phép so sánh thừa.

**Cài đặt thuật toán:**

void Shaker\_Sort (int a[], int n)

{

int left, right, pos, j;

left = 0;

right = pos = n - 1;

while (left < right)

{

for (j = right; j > left; j--)

{

if (a[j] < a[j - 1])

Swap(a[j], a[j - 1]);

pos = j;

}

left = pos;

for (j = left; j < right; j++)

{

if (a[j] > a[j + 1])

{

Swap(a[j], a[j + 1]);

pos = j;

}

}

right = pos;

}

}

**Đánh giá thuật toán:**

Trường hợp xấu nhất:O(n\*n).

Trường hợp tốt nhất: O(n).

Trường hợp trung bình: O(n\*n).

1. **Thuật toán sắp xếp Shell sort:**

**Giải thuật chính của sắp xếp Shell sort là:** Phân hoạch dãy thành các dãy con. Sắp xếp các dãy con theo phương pháp chèn trực tiếp . Dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy.

**Cài đặt thuật toán:**

void ShellSort (int a[], int n) {

int interval, i, j, temp;

for(interval = n/2; interval > 0; interval /= 2) {

for(i = interval; i < n; i++){

temp = a[i];

for(j = i; j >= interval && a[j - interval] > temp; j -= interval){

a[j] = a[j - interval];

}

a[j] = temp;

}

}

}

**Đánh giá thuật toán:** Giải thuật này khá hiệu quả với các tập dữ liệu có kích cỡ trung bình khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n), với n là số phần tử.

1. **Thuật toán sắp xếp phân phối (Counting sort):**

**Giải thuật chính của sắp xếp phân phối (Counting sort) là:** Giá trị khóa của **ai** là chỉ số của mảng **B** có **k** phần tử (**k** là giá trị khóa lơn nhất của **A**). Quá trình sắp xếp danh sách **A** là đếm số phần tử của mỗi chỉ số của **B** trong **A**. Từ đó tính ra thứ tự của các khóa **ai**trong **A**. Kết quả sắp xếp có được bằng cách lấy vị trí của phần tử **A** được lưu trong **B**.

**Cài đặt thuật toán:**

void CountingSort(int \*a, int n)

{

int max = a[0];

int min = a[0];

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

{

if(a[i] > max) max = a[i];

else if(a[i] < min) min = a[i];

}

int j = max-min+1; *//số lượng phần tử trong mảng đếm*

int \*b = new int [j]; *//tạo mảng đếm để chứa số lượng các khóa từ mảng đã cho*

fill\_n(b,j,0);

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

b[a[i]-min]++; *//lưu số lượng khóa vào mảng đếm*

for(int i = 1; i < j; i++)

b[i] += b[i-1]; *//đổi mảng đếm sao cho chứa vị trí thực của các phần tử cần sắp xếp theo thứ tự tăng dần*

int \*c = new int [n]; *//tạo mảng mới để chứa dãy số sắp xếp*

for(int i = 0; i < n; i++) *//sử dụng vị trí của khóa được lưu trong mảng đếm để sắp xếp mảng mới*

{

c[b[a[i]-min]-1]=a[i];

b[a[i]-min]--;

}

}

**Đánh giá thuật toán:**

* Trong mọi trường hợp, độ phức tạp tính toán của Counting Sort là O(n+k), trong đó k là kích thước của mảng B
* Counting Sort là một trong những thuật toán sắp xếp không dựa vào kết quả so sánh giá trị khóa của các phần tử trong danh sách

1. **Thuật toán sắp xếp phân tích cơ số (Radix sort):**

**Giải thuật chính của sắp xếp phân tích cơ số (Radix sort) là:** Đây là một thuật toán mà nó không đi sắp xếp theo cách thông thường là ta thường lấy phần tử để so sánh xong sắp xếp lại cho đúng vị trí. Radix sort đi theo cách tiếp cận khác đó là sắp xếp không so sánh nó dựa vào việc phân loại thư. Radix Sort không hề quan tâm đến việc so sánh giá trị của phần tử mà bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử.

=> Radix Sort thích hợp với cấu trúc là danh sách liên kết hơn là dùng cấu trúc mảng

**Cài đặt thuật toán:**

void Radix\_Sort(List& l)

{ int Base = 10;

int SoChuSo = Count(l);

List Arr[10];

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

{

for (int i = 0; i < 10; i++)

{

CreateList(Arr[i]);

}

while (l.pHead != NULL)

{int x = Remove\_Head(l);

int sodu = (x % Base) \* 10 / Base;

Add\_Tail(Arr[sodu], CreateNode(x)); }

for (int i = 0; i < 10; i++)

{ Node\* k = Arr[i].pHead;

while (k != NULL)

{ Add\_Tail(l, k);

k = k->pNext;}

}Base \*= 10;

}

}

**Đánh giá thuật toán:**

* Với một dãy n số, mỗi số có tối đa m chữ số, thuật toán thực hiện m lần các thao tác phân lô và ghép lô.
* Trong thao tác phân lô, mỗi phần tử chỉ được xét đúng một lần, khi ghép cũng vậy.
* Như vậy, chi phí cho việc thực hiện thuật toán hiển nhiên là O(2mn) = O(n).
* Radix Sort thích hợp với cấu trúc là danh sách liên kết hơn là dùng cấu trúc mảng.

1. **Thuật toán sắp xếp vun đống (Heapsort) :**

**Giải thuật chính của sắp xếp vun đống (Heapsort) là:** Giải thuật Heapsort còn được gọi là giải thuật vun đống, có thể được xem như bản cải tiến của [Selection Sort](https://www.stdio.vn/giai-thuat-lap-trinh/selection-sort-KQi3U) khi chia các phần tử thành 2 mảng con.

* 1 mảng các phần tử đã được sắp xếp.
* 1 mảng các phần tử chưa được sắp xếp.

Trong mảng chưa được sắp xếp, các phần tử lớn nhất sẽ được tách ra và đưa vào mảng đã được sắp xếp.

**Cài đặt thuật toán:**

void CreateHeap(int a[], int n){

int l=n/2-1;

while(l>=0) {

Shift(a,l,n-1);

l--;

}

}

void Shift(int a[], int l, int r){

int i=l;

int j=2\*i+1;

while(j<=r) {

if(j<r &&a[j]<a[j+1])

j++;

if(a[j]<=a[i])

return;

swap(a[i], a[j]);

i=j;

j=2\*i+1;

}

}

void HeapSort(int a[], int n) {

CreateHeap(a, n);

int r = n -1;

while(r > 0) {

swap(a[0], a[r]);

r--;

if(r > 0)

Shift(a, 0, r); }}

**Đánh giá thuật toán:**

Trường hợp tốt nhất: O(nlog(n)).

Trường hợp xấu nhất:O(nlong(n)).

Trường hợp trung bình: O(nlog(n)).

Heapsort là thuật toán in-place, nghĩa là không cần thêm bất cứ cấu trúc dữ liệu phụ trợ trong quá trình chạy thuật toán. Tuy nhiên, giải thuật này không có độ ổn định (stability).

1. **Thuật toán sắp xếp nhanh (Quick sort):**

**Giải thuật chính của sắp xếp nhanh (Quick sort) là:**



**\* Trường hợp thứ nhất:**

- Đoạn thứ 2 đã có thứ tự.

- Nếu các đoạn 1 và 3 chỉ có 1 phần tử: đã có thứ tự

🡪 Khi đó dãy con ban đầu đã được sắp.



**\* Trường hợp thứ hai:**

- Đoạn thứ 2 đã có thứ tự.

- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.

- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày…

**Cài đặt thuật toán:**

void QuickSort(int a[], int l, int r)

{ int i, j, x;

x = a[(l+r)/2];

i = l; j = r;

while(i < j)

{ while(a[i] < x) i++;

while(a[j] > x) j--;

if(i <= j)

{ swap(a[i],a[j]);

i++; j--; }

}

if(l < j) QuickSort(a, l, j);

if(i < r) QuickSort(a, i, r);

}

**Đánh giá thuật toán:**

Trường hợp tốt nhất: O(n\*n).

Trường hợp xấu nhất:O(nlog(n)).

Trường hợp trung bình: O(nlog(n)).

1. **Thuật toán sắp xếp trộn (Merge sort)**

**Giải thuật chính của sắp xếp trộn (Merge sort) là:**

Cũng là thuật toán theo cơ chế phân hoạch/ chia để trị.

* Chia mảng cần sắp xếp thành nhiều mảng con:

✓Hoặc thành 2 mảng con theo phân phối bằng cách xen kẻ, luân phiên :

✓Hoặc thành 2 mảng con từ phần tử chính giữa ✓Hoặc phân chia thành các mảng con theo đường chạy tự nhiên (mảng con tăng/giảm ngặt, mảng con không giảm/tăng)

* Tiếp tục lặp lại việc này ở các nửa mảng đã chia.
* Sau cùng gộp các nửa đó thành 1 mảng đã sắp xếp

**Cài đặt thuật toán:**

void MergeSort (int a[], int Left, int Right)

{

if (Left < Right)

{

int Mid = (Left + Right) / 2;

*// Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng*

MergeSort\_Tang(a, Left, Mid);

MergeSort\_Tang(a, Mid + 1, Right);

Merge\_Tang(a, Left, Mid, Right);

}

}

void Merge (int a[], int Left, int Mid, int Right)

{ int\* temp = new int[Right - Left + 1]; *//Tạo mảng tạm temp*

int m;

int i = Left;

int j = Mid + 1;

m = 0;

*//Vừa sắp xếp vừa gộp mảng vào mảng tạm temp*

while (!(i > Mid && j > Right))

{ if ((i <= Mid && j <= Right && a[i] < a[j]) || j > Right)

temp[m++] = a[i++];

else

temp[m++] = a[j++];

}

}

**Đánh giá thuật toán:**

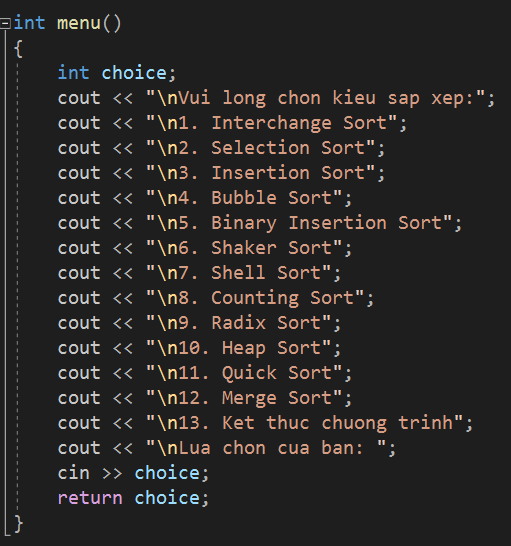
Trường hợp tốt nhất: O(1).

Trường hợp xấu nhất:O(nlog(n)).

Trường hợp trung bình: O(nlog(n)).

1. **Quản lý chương trình bằng cách dùng menu.**

Thông qua Menu() với giao diện trực quan, gọn, dễ hiểu và thông qua các lựa chọn để thực hiện chương trình dễ dàng**.**



# **KIỂM THỬ CHƯƠNG TRÌNH VÀ HƯỚNG DẪN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **HƯỚNG DẪN** | | | **KẾT QUẢ** |
| 1 | Chọn 1 trong 13 lựa chọn sau :  1. Interchange Sort  2. Selection Sort  3. Insertion Sort  4. Bubble Sort  5. Binary Insertion Sort  6. Shaker Sort  7. Shell Sort  8. Counting Sort  9. Radix Sort  10. Heap Sort  11. Quick Sort  12. Merge Sort  13. Ket thuc chuong trinh | | |  |
| 2 | Bởi các lựa chọn trên đều có chung cách thực hiện.  Nên sau khi chọn 1 trong các lựa chọn trên: VD chọn: 1.Interchange Sort  Chọn tiếp 1 trong 2 lựa chọn:   1. Thu cong (Nhập bằng tay) 2. Ngau nhien (Tự động cho ra dãy số bất kì) | | |  |
| 3 | Nếu chọn  1.Thu cong | STT |  |  |
| 1 | Nhập số nguyên n phần tử mảng số nguyên muốn nhập. VD: 5 |  |
| 2 | Nhập các phần tử của mảng số nguyên cần tạo từ phần tử a[0] đến phần tử a[n-1]. |  |
| 3 | Sau khi nhập xong mảng.  Chọn tiếp 1 trong 2 lựa chọn.   1. Tang ( sắp mảng theo chiều tăng dần) 2. Giam (sắp mảng theo chiều giảm dần) |  |
| 4 | Tiếp tục chọn 1 trong 3 tốc độ xử lý sau:   1. Nhanh. 2. Trung bình. 3. Chậm |  |
| 5 | Sau khi chọn.  VD chọn: 1. Tang ở bước 3 và 1. Nhanh ở bước 4  Chương trình sẽ xuất ra từng quá trình mà mảng sắp xếp theo chiều tang cho đến khi hoàn thành dãy đã được sắp tang với tốc độ xử lý nhanh. |  |
| Nếu chọn  2. Ngau nhien | STT |  |  |
| 1 | Nhập số nguyên n phần tử muốn mảng số nguyên tạo ngẫu nhiên. VD: 5 |  |
| 2 | Sau đó chương trình sẽ tự tạo một mảng số nguyên bất kì có n (5) phần tử bất kì.  Sau đó tiếp tục chọn 1 trong 2 lựa chọn.   1. Tang 2. Giam |  |
| 3 | Tiếp tục chọn 1 trong 3 tốc độ xử lý sau:  1. Nhanh.  2. Trung bình.  3. Chậm |  |
| 4 | Sau khi chọn.  VD chọn: 1. Tang  Chương trình sẽ xuất ra từng quá trình mà mảng sắp xếp theo chiều tăng cho đến khi hoàn thành dãy đã được sắp tăng (Với lựa chọn 2 cũng tương tự) |  |
| 4 | Màn hình lựu chọn thuật toán quay trở lại. Tiếp tục chọn 1 trong các thuật toán sắp xếp để thực hiện tương tự như trên. | | |  |
| 5 | Sau khi thực hiện hiện các thuật toán sắp xếp xong.  Chọn: 13. Ket thuc chuong trinh  Để kết thúc. | | |  |