**TRƯỜNG ĐẠI HỌC NHA TRANG**

**BỘ MÔN MẠNG MÁY TÍNH & TRUYỀN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO THỰC TẬP CƠ SỞ**

SO SÁNH THỜI GIAN CHẠY CỦA MỘT SỐ GIẢI THUẬT SẮP XẾP

Giáo viên hướng dẫn: Cấn Thị Phượng

CÁC THÀNH VIÊN TRONG NHÓM

|  |  |
| --- | --- |
| Thành viên | Thuật toán |
| Võ Đinh Chí | Merge Sort, Heap Sort |
| Cáp Minh Hoàng | Radix sort, Insert Sort |
| Phạm Hoàng Huy | Bubble sort, Quicksort |
| Lê Thị Mỹ Lê | Selection Sort ,Tree Sort |
| Hồ Đại Phương | Couting sort, Timsort |

**Sơ lược**

**I.Thông tin chung về đề tài:**

**II.Nội dung**

1.Tổng quan

2.Công cụ làm việc

2.1.Giới thiệu

2.2.Chi tiết

3.Chi tiết từng thuật toán

4.Lập trình

**I.THÔNG TIN CHUNG VỀ ĐỀ TÀI**

**1. Tên đề tài:** So sánh thời gian chạy của một số giải thuật sắp xếp (Selection Sort và Tree Sort)

**2. Nhóm sinh viên thực hiện:**

- Họ và tên: Võ Đình Chí, Cáp Minh Hoàng, Phạm Hoàng Huy, Lê Thị Mỹ Lê, Hồ Đại Phương

- Lớp: 60-CNTT 3

- Khoa: Công nghệ Thông tin

**3. Giáo viên hướng dẫn:** Cấn Thị Phượng

**4. Thời gian thực hiện:** từ 18-12-2020 đến 19-01-2021

**5. Cơ sở thực tập:** Bộ môn Mạng máy tính và Truyền thông, Khoa Công nghệ Thông Tin, Trường Đại học Nha Trang.

**II. NỘI DUNG**

**1.Tổng quan**

*1.1. Lý do lựa chọn đề tài:* So sánh thời gian chạy của một số giải thuật để chọn ra giải thuật sắp xếp nào nhanh nhất với số liệu dữ liệu khác nhau

*1.2. Mục đích của đề tài:*

Vận dụng ngôn ngữ lập trình C++, C#, phương pháp hướng đối tượng, so sánh được thời gian chạy của một số giải thuật sắp xếp. Rèn luyện kỹ năng giải quyết vấn đề, làm việc nhóm, quản lý mã nguồn

*1.3. Các nội dung của đề tài:*

**Liệt kê các nội dung thực hiện của đề tài ở đây**

-Trình bày được thuật toán chi tiết và cụ thể.

-Tổ chức dữ liệu và cài đặt chương trình hoàn chỉnh.

- Sử dụng đầu vào kích thước khác nhau n=10, n=100, n=500… Chạy 5-10 lần cho mỗi thuật toán cho mỗi đầu vào và tính được thời gian chạy trung bình.

- Hiển thị thời gian chạy so sánh giữa các thuật toán

*1.4.Dự kiến kết quả đạt được:*

Sử dụng các dữ liệu đầu vào từ các file Notepad, Excel,… để thực hiện giải thuật.

So sánh thời gian chạy và tìm được những thuật toán có thời gian chạy tối ưu nhất trong các thuật toán sắp xếp đã học.

**2.Công cụ làm việc:**

*2.1.Sơ Lược*

GitHub là một dịch vụ cung cấp [kho lưu trữ mã nguồn](https://vi.wikipedia.org/w/index.php?title=Kho_l%C6%B0u_tr%E1%BB%AF_m%C3%A3_ngu%E1%BB%93n&action=edit&redlink=1) [Git](https://vi.wikipedia.org/wiki/Git_(ph%E1%BA%A7n_m%E1%BB%81m)) dựa trên nền web cho các dự án phát triển phần mềm. GitHub cung cấp cả phiên bản trả tiền lẫn miễn phí cho các tài khoản. Các dự án [mã nguồn mở](https://vi.wikipedia.org/wiki/Ph%E1%BA%A7n_m%E1%BB%81m_ngu%E1%BB%93n_m%E1%BB%9F) sẽ được cung cấp kho lưu trữ miễn phí. Tính đến tháng 4 năm 2016, GitHub có hơn 14 triệu người sử dụng với hơn 35 triệu kho mã nguồn, làm cho nó trở thành máy chủ chứa mã nguồn lớn trên thế giới.

Github đã trở thành một yếu tố có sức ảnh hưởng trong cộng đồng phát triển mã nguồn mở. Thậm chí nhiều nhà phát triển đã bắt đầu xem nó là một sự thay thế cho sơ yếu lý lịch và một số nhà tuyển dụng yêu cầu các ứng viên cung cấp một liên kết đến tài khoản Github để đánh giá ứng viên.[[](https://vi.wikipedia.org/wiki/GitHub#cite_note-gitresume-5)

*2.2.Chi tiết:*

Có các cách để làm việc với github như: Web hoặc app nhưng đều cần có tài khoản để tạo dự án ( Project) và chia nhánh.

2.2.1. Làm việc với git

- Sử dụng lệnh git init để khởi tạo một kho mới

- git clone <đường dẫn> để tạo bản sao từ Server Git (online)

- cd <tên kho/tên thư mục> để di chuyển đến kho hoặc thư mục

- touch <tên file> để tạo file

- git add hello.html để đưa file vào chế độ sẵn sàng commit

- git commit -m “ Câu commit” để lưu lại những thay đổi với file

- git push để push các file lên online.

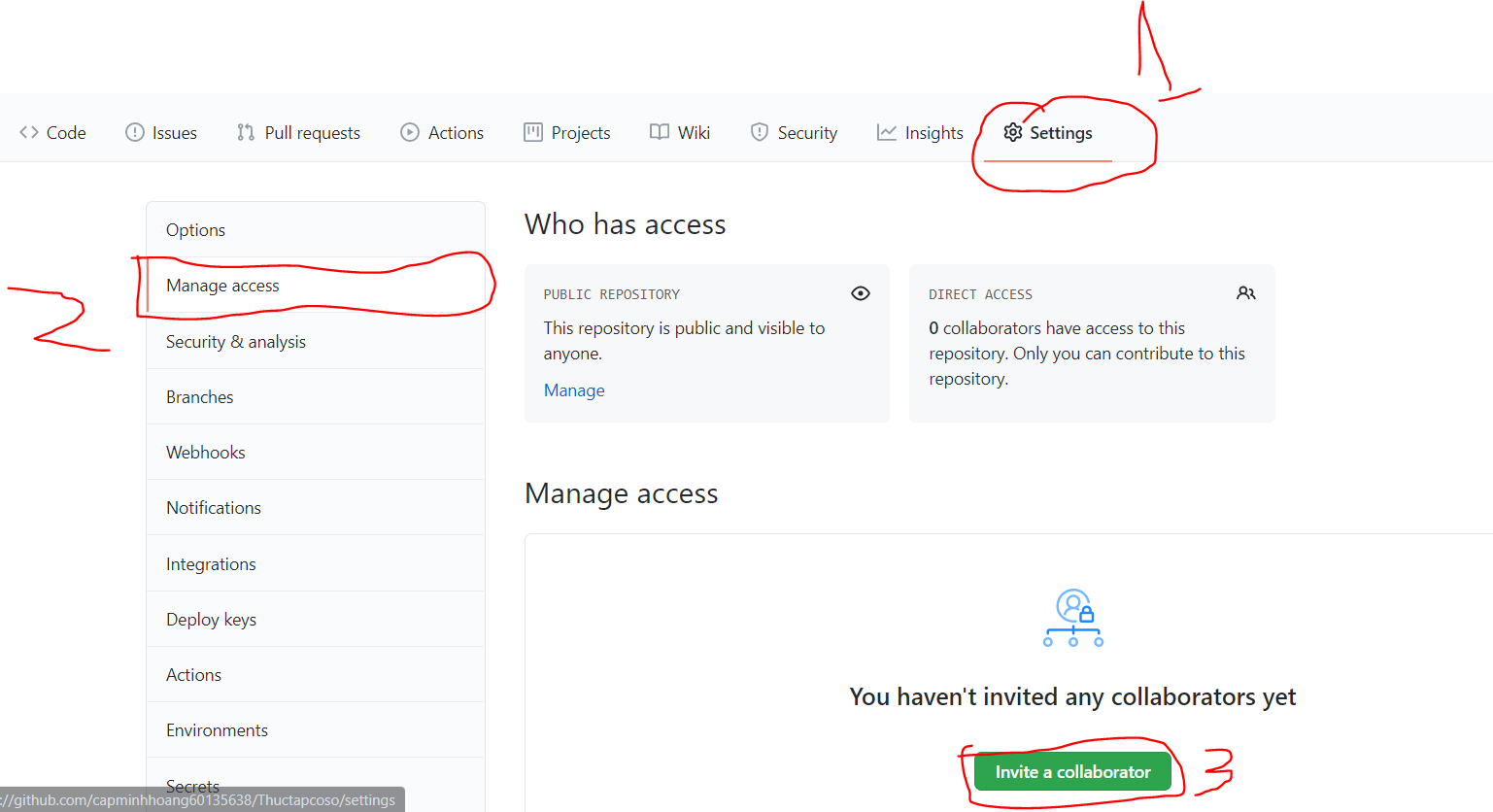
- git pull để lấy từ online về local

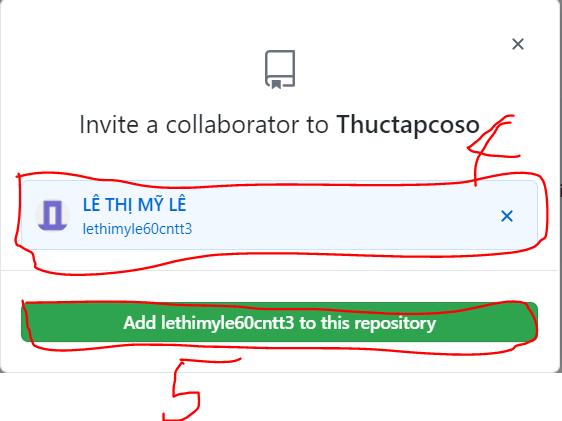
Sau khi chuẩn bị đủ các thành viên và tạo 1 project ta cần tạo các branch (Nhánh) để làm việc cho từng cá nhân như sau:

BƯỚC 1: Sử dụng git checkout -b <ten nhanh moi> để tạo nhánh trong dự án

Ví dụ: git checkout –b Hoang

BƯỚC 2: Vào git online >> Tên dự án >> Setting >> Manage Access >> Invite a collaborator >> Nhap tên thành viên trong nhóm >> Add để mời thành viên vào dự án





**3.Vận dụng lập trình hướng đối tượng và kỹ thuật lập trình**

**3.1 Vận dụng lập trình hướng đối tượng**

Chương trình gồm 2 lớp chính sau; Source và Sort

Source có 3 lớp con là: Source100, Source250, Source500

Source có các phương thức sau:

xuatmang(int arr[],int n) để in ra mảng arr có độ lớn n

taoarr(int arr[],int n) để tạo ra một mảng arr có các phần tử ngẫu nhiên có độ lớn n

laymangfile(char\* s,int arr[]) để lấy các phần tử của mảng arr trong file theo đường dẫn s

Source100, Source250, Source500 có:

Các thuộc tính là:

mảng dulieu100(250,500) để chứa các phần tử sắp xếp

số nguyên n là độ lớn của mảng

Các phương thức được kế thừa từ lớp Source

Code:

class Source

{

public:

void xuatmang(int arr[],int n)

{

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

{

cout<<arr[i]<<"\n";

}

}

void taoarr(int arr[],int n)

{

srand(time(NULL));

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

{

arr[i]=rand();

}

}

void laymangfile(char\* s,int arr[])

{

fstream fsFile;

fsFile.open(s, ios::in);

if (fsFile.eof())

{

cout << "Read done";

}

else

{

int tam ;

int i = 0;

while (fsFile>>tam)

{

//cout<<tam<<"\n";

arr[i]=tam;

i++;

}

}

fsFile.close();

}

};

class Source100: public Source{

public:

int dulieu100[100];

int n=100;

Source100():Source()

{

}

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

class Source250:public Source{

public:

int dulieu250[250];

int n=250;

Source250():Source()

{

}

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

class Source500: public Source{

public:

int dulieu500[500];

int n=500;

Source500():Source()

{

}

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

Sort có các lớp con sau: merge, heap, radix, insert, bubble, quick, selection, tree, counting, tim

Sort có phương thức:

void inmang(int arr[],int n) để in mảng arr sau khi được sắp xếp có độ dài n

void hinhchunhat(int x1,int y1,int x2,int y2) để vẽ hình chữ nhật có tọa độ ở điểm có (x1,y1) ở trên trái và (x2,y2) ở góc dưới phải

void screen(int arr[], int n) để vẽ các hình chữ nhật có độ cao dựa vào các phần tử trong mảng

Code:

class Sort{

public:

void inmang(int arr[],int n)

{

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

cout << arr[i] << " ";

}

void hinhchunhat(int x1,int y1,int x2,int y2)

{

line(x1,y1,x1,y2);

line(x1,y2,x2,y2);

line(x2,y2,x2,y1);

line(x2,y1,x1,y1);

}

//in ra man hinh day so theo do hoa

void screen(int arr[], int n)

{

int xstart=0,ystart=0;

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

{

hinhchunhat(xstart+i\*800/n, 0, xstart+(i+1)\*800/n, arr[i]/100);

}

} };

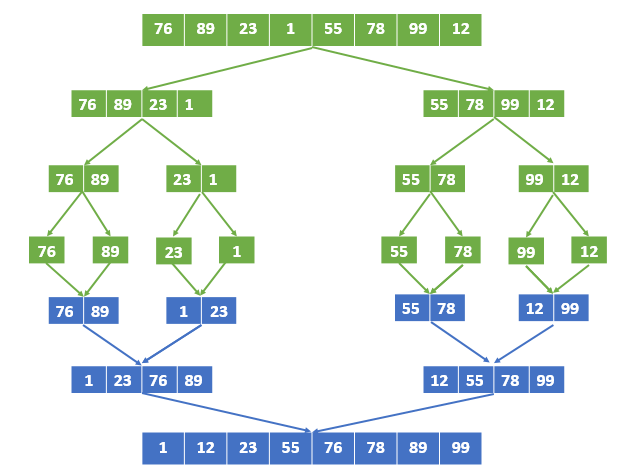
**4.Chi tiết thuật toán**

*3.1. Thuật Toán MergeSort (Sắp xếp trộn)*

3.1.1Giới thiệu về thuật toán MergeSort:

Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp.

3.1.2.Minh họa cách sắp xếp Merge:



3.1.3.Ý tưởng thực hiện:

* Dữ liệu đầu vào sẽ được lưu trữ ở dạng mảng (Array) để thực hiện chương trình đơn giản hơn. Cụ thể là thuật toán Merge sử dụng phương pháp chia để trị nên dùng mảng sẻ tiện cho việc chia mảng thành các mảng con.

Sử dụng giải thuật đệ qui: Việc chia mảng của thuật toán merge sort sẽ lặp đi lặp lại nhiều lần đến khi không chia được nữa nên việc áp dụng đệ qui sẽ làm code gọn gàng và tối ưu thời gian thực hiện hơn.

3.1.4.Ưu nhược điểm của thuật toán:

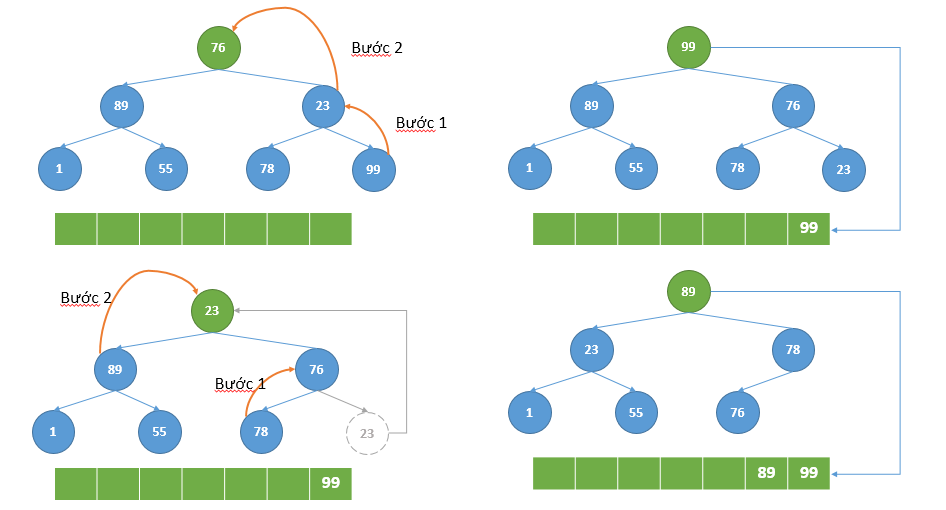
* Ưu điểm: Sắp xếp nhanh hơn so với các thuật toán cơ bản (Insertion Sort, Selection Sort, Interchange Sort). Nhanh hơn Quick Sort trong một số trường hợp.
* Nhược điểm: thuật toán khó cài đặt, không nhận dạng được mảng đã sắp xếp.

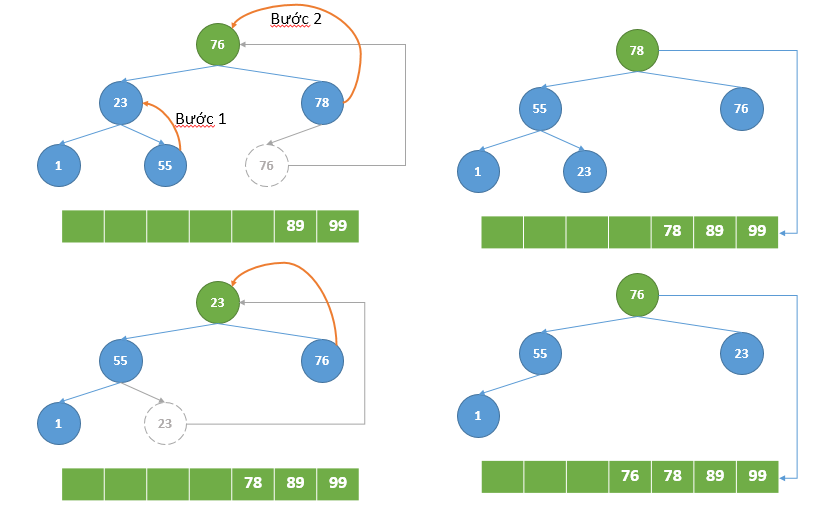
*3.2.Thuật Toán HeapSort (Sắp xếp vun đống)*

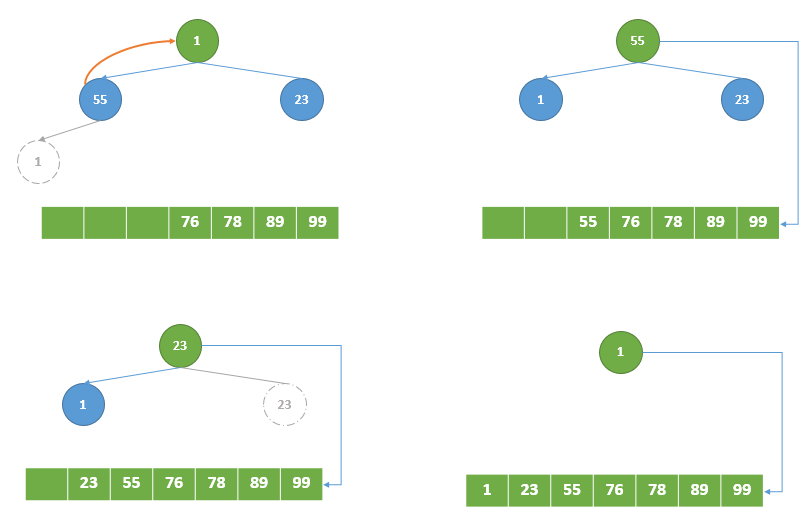
3.2.1Giới thiệu về thuật toán HeapSort:

Heap sort là một kỹ thuật sắp xếp dựa trên so sánh dựa trên cấu trúc dữ liệu Binary Heap. Khi giải thuật bắt đầu thực hiện, phần tử lớn nhất sẽ được lưu trữ ở gốc của Heap sau đó phần tử đó sẽ được lấy đưa vào cuối danh sách sắp xếp đã sắp xếp. Đồng thời thay thế nó ở cây bằng mục cuối cùng của Heap, sau đó giảm kích thước của đống đi 1 đơn vị. Cuối cùng, chất đống gốc của cây. Tiếp tục thực hiện giải thuật đến khi kích thước Heap bằng 1.

3.2.2Minh họa cách sắp xếp Heap:







3.2.3Ý tưởng thực hiện:

* Ở thuật toán này dữ liệu được lưu trữ ở dạng cây phị phân, nhưng để thuận tiện cho việc viết code được dễ dàng thì em xin giải quyết theo kiểu lưu trữ của mảng cho dễ hiểu hơn.
* Sử dụng giải thuật đệ qui: Sau mỗi lần thực hiện tìm phần tử lớn nhất trong Heap đưa lên làm gốc (Root), đồng thời ta phải tiếp tục tìm phần tử mang giá trị gốc tiếp theo đến khi chỉ còn 1 phần tử duy nhất. Việc tìm kiếm lặp đi lặp lại này thì ta sẽ sử dụng đệ qui để tối ưu code.

3.2.4 Ưu nhược điểm của thuật toán:

* Ưu điểm: Chạy nhanh, hiệu suất của thuật toán cao. Sẽ chạy nhanh hơn nếu mảng đầu vào đã được sắp xếp 1 phần.
* Nhược điểm: Chạy chậm hơn MergeSort, QuickSort trong một vài trường hợp. Code phức tạp.

*3.3.Radix Sort*

3.3.1Trình bày thuật toán

Radix Sort là thuật toán sắp xếp dựa vào cơ số. cơ sở để sắp xếp của Radix Sort là dựa theo nguyên tắc phân loại thư của bưu điện.

Radix sort không quan tâm đến việc so sánh giá trị của 2 phần tử mà bản thân việc phân loại và thứ tự phân loại sẽ tạo ra thứ tự cho các phần tử.

3.3.2Ý tưởng cài đặt

Giả sử mỗi phần tử ai trong dãy a0, a1, …, an-1 là một số nguyên có tối đa m chữ số. Phân loại các phần tử này lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, …

Ví dụ: Dãy cần sắp xếp là: 964, 354, 368, 128, 495, 121

Bước 1: Phân theo hàng đơn vị:

**1**: 12**1**

**2**:

**3**:

**4**: 96**4**, 35**4**

**5**: 49**5**

**6**:

**7**:

**8**: 36**8**, 12**8**

**9**:

Mảng A sau khi phân theo hàng đơn vị: 121, 964, 354, 495, 368, 128

Bước 2: Phân theo hàng chục:

**1**:

**2**: 1**2**1,1**2**8

**3**:

**4**:

**5**: 3**5**4

**6**: 9**6**4,3**6**8

**7**:

**8**:

**9**: 4**9**5

Mảng A sau khi phân theo hàng chục: 121,128,354,964,368,495

Bước 3: Phân theo hàng trăm:

**1**: **1**21,**1**28

**2**:

**3:** **3**54**,3**68

**4**: **4**95

**5:**

**6**:

**7**:

**8**:

**9**: **9**64

Mảng A sau khi phân theo hàng trăm: 121,128,354,368,495,964

*3.4. Insert Sort*

3.4.1Trình bày thuật toán

Thuật toán sắp xếp chèn (Insertion Sort) thực hiện sắp xếp các phần tử theo cách duyệt từng phần tử. Và chèn từng phần tử đó vào đúng vị trí trong mảng con. Phần tử được chuyền vào vị trí thích hợp sao cho mảng con vẫn đảm bảo sắp xếp theo đúng thứ tự.

3.4.2.Ý tưởng cài đặt

Xét danh sách con gồm k phần tử đầu a1 … ak. Với k = 1, danh sách gồm một phần tử đã được sắp xếp thành mảng tăng dần. Giả sử trong danh sách k - 1 phần tử đầu a1 … ak - 1 đã được sắp xếp.

Để sắp xếp phần tử ak = x, tìm vị trí thích hợp của nó trong mảng a1 … ak - 1. Vị trí thích hợp cần tìm là vị trí đứng trước phần tử lớn hơn nó và sau phần tử nhỏ hơn hoặc bằng nó.

Ví dụ: Dãy cần sắp xếp là:

A= [ 121, 354, 368, 128, 495, 964 ]

Duyệt từng phần tử, ta thấy rằng từ phần tử thứ 1 đến phần tử thứ 3 không có phần tử đứng trước nhỏ lớn hơn phần tử đang xét nên dãy đã được sắp xếp

Xét phần tử thứ A[4] = 128, ta được vị trí phù hợp là ở giữa A[1]=121 và A[2]=354. Nên mảng sẽ được sắp xếp lại thành:

A= [ 121, 128, 354, 368, 495, 964 ]

Xét phần tử thứ A[5] = 495 và A[6], ta thấy rằng không có phần tử đang xét nào nhỏ hơn phần tử đứng trước nên dãy đã được sắp xếp xong.

Dãy sau khi sắp xếp là:

A= [ 121, 128, 354, 368, 495, 964 ]

3.5. **Bubble sort.**

**3.5.1.Giới thiệu**

**Bubble sort** là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử **kề** nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ . Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang). Sắp xếp nổi bọt còn có tên là *sắp xếp bằng so sánh trực tiếp*.

3.5.2.Ý tưởng cài đặt

* Bước 1: Xét các phần tử từ a[n] đến a[2],với mỗi phần tử a[j] so sánh nó và khóa của phần tử a[j-1]. Nếu khóa của a[j] nhỏ hơn khóa của a[j-1] thì đổi chổ a[j] với a[j-1].
* Bước 2: Xét các phần tư từ a[n] đến a[3],làm tương tự bước 1.
* Bước i: Xét các phần tử từ a[n] đến a[i+1],làm tương tự.
* Sau n bước ta được dãy đã có thứ tự.

**Ví dụ**

**Lần lặp đầu tiên:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Ở đây, thuật toán sẽ so sánh hai phần tử đầu tiên, và đổi chỗ cho nhau do 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Đổi chỗ do 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Đổi chỗ do 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Ở đây, hai phần tử đang xét đã đúng thứ tự

(8 > 5), vậy ta không cần đổi chỗ.

**Lần lặp thứ 2:**  
( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Đổi chỗ do 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )

**Lần lặp thứ 3:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

Hiệu suất của bubble sort O(n):

- Tốt nhất: n

- Tệ nhất: n^2

Ưu điểm: Code ngắn gọn nhất

Nhược điểm: Hiệu suất thấp nhất

3.6. **Quick Sort**

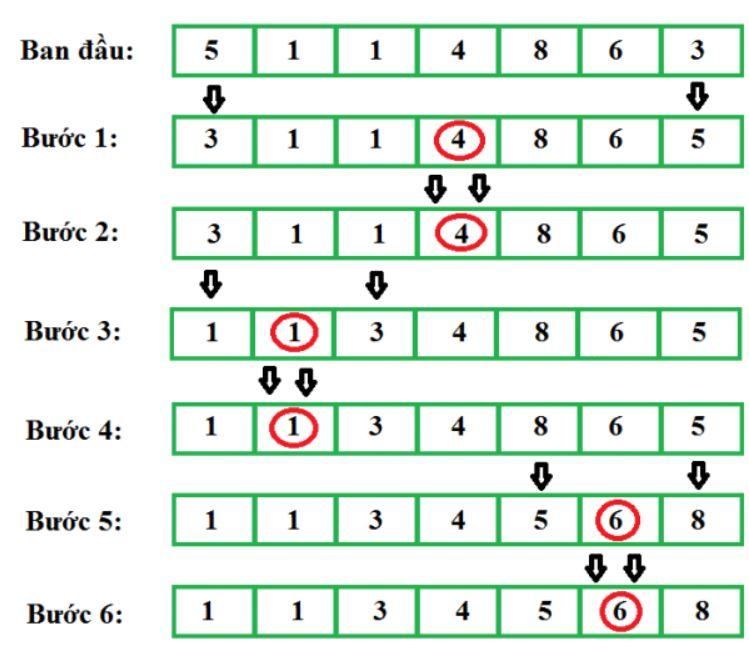
**3.6.1.Giới thiệu**

Quick Sort là một thuật toán sắp xếp hiệu quả dựa trên việc phân chia mảng dữ liệu thành các nhóm phần tử nhỏ hơn. Giải thuật sắp xếp nhanh chia mảng thành hai phần bằng cách so sánh từng phần tử của mảng với một phần tử được gọi là phần tử chốt. Một mảng bao gồm các phần tử nhỏ hơn hoặc bằng phần tử chốt và một mảng gồm các phần tử lớn hơn phần tử chốt. Quá trình phân chia này diễn ra cho đến khi độ dài của các mảng con đều bằng 1. Với phương pháp đệ quy ta có thể sắp xếp nhanh các mảng con sau khi kết thúc chương trình ta được một mảng đã sắp xếp hoàn chỉnh.

3.6.2.Ý tưởng cài đặt

* Xét mảng a có các bản ghi từ a[1],…,a[n].
* Chọn một thành phần của mảng làm chốt(pivot). Phân goạch mảng thành hai phần bằng cách chuyển tất cả các thành phần có khóa lớn hơn chốt sang phải chốt, các thành phần có khóa bé hơn hoặc bằng sang trái chốt.
* Sắp xếp độc lập hai mảng con a[1,…,k-1],a[k+1,…n] bawfgn cách gọi đệ quy thuật toán trên.

Ví dụ:



3.6.3. Độ phức tạp của giải thuật Quick Sort

- Tốt nhất: n log n

- Tệ nhất: n^2

3.6.4 Ưu điểm và nhược điểm

Ưu điểm: Tuỳ cách chọn pivot mà tốc độ của thuật toán nhanh hay chậm

Nhược điểm: Code khá phức tạp

3.7.Couting Sort

3.7.1.Giới thiệu:

Sắp xếp đếm(**Counting Sort**) là một kỹ thuật sắp xếp dựa trên các khóa giữa một phạm vi cụ thể. Nó hoạt động bằng cách đếm số lượng các đối tượng có các giá trị khóa riêng biệt (loại băm). Sau đó, thực hiện một số phép tính để tính toán vị trí của mỗi đối tượng trong chuỗi đầu ra.

3.7.2.Ý tưởng

**Input**: Đầu vào sẽ là n số nguyên trong khoảng [0,k], trong đó *k* là số nguyên, phạm vị giá trị của n và *k = O(n)*

**Ý tưởng**: Với mỗi phần tử x của dãy đầu vào ta xác định hạng (rank) của nó như là số lượng phần tử nhỏ hơn x. Một khi ta đã biết hạng *r* của x, ta có thể xếp nó vào vị trí *r+1*

**Ví dụ**: Nếu có 6 phần tử nhỏ hơn 17, ta có thể xếp 17 vào vị trí thứ 7.

**Lặp**: Khi có 1 loại phần tử có cùng giá trị, ta sẽ xếp chúng theo thứ tự xuất hiện trong dãy ban đầu để có được tính ổn định nhất của sắp xếp

3.7.3.Minh họa:

Để đơn giản, hãy xem xét dữ liệu trong phạm vi từ 0 đến 9.

Dữ liệu đầu vào: 1, 4, 1, 2, 7, 5, 2

1) Lấy một mảng đếm để lưu trữ số lượng của mỗi đối tượng duy nhất.

Chỉ số: 0 1 2 3 4 5 6 7 8 9

Đếm: 0 2 2 0 1 1 0 1 0 0

2) Sửa đổi mảng đếm sao cho mỗi phần tử ở mỗi chỉ mục lưu trữ tổng của các lần đếm trước đó.

Chỉ số: 0 1 2 3 4 5 6 7 8 9

Đếm: 0 2 4 4 5 6 6 7 7 7

Mảng đếm đã sửa đổi cho biết vị trí của từng đối tượng trong trình tự đầu ra.

3) Xuất ra từng đối tượng từ chuỗi đầu vào theo sau là giảm số lượng của nó đi 1.

Xử lý dữ liệu đầu vào: 1, 4, 1, 2, 7, 5, 2. Vị trí của 1 là 2.

3.7.4 Những điểm cần lưu ý

a. Sắp xếp đếm sẽ hiệu quả nếu phạm vi dữ liệu đầu vào không lớn hơn đáng kể so với số đối tượng được sắp xếp. Hãy xem xét tình huống trong đó chuỗi đầu vào nằm trong khoảng từ 1 đến 10K và dữ liệu là 10, 5, 10K, 5K.

b. Nó không phải là sự sắp xếp dựa trên so sánh. Độ phức tạp thời gian chạy của nó là O (n) với không gian tỷ lệ với phạm vi dữ liệu.

c. Nó thường được sử dụng như một quy trình con cho một thuật toán sắp xếp khác như sắp xếp cơ số.

d. Sắp xếp đếm sử dụng băm một phần để đếm sự xuất hiện của đối tượng dữ liệu trong O (1).

e. Sắp xếp đếm có thể được mở rộng để làm việc cho các đầu vào âm.

3.8.Timsort

3.8.1.Giới thiệu

Timsort là một thuật toán sắp xếp hỗn hợp và hiệu quả được tạo ra bằng cách sử dụng khái niệm của cả sắp xếp hợp nhất và sắp xếp chèn. Hơn nữa, Nó được thiết kế để hoạt động tốt trên nhiều loại dữ liệu trong thế giới thực. Tuy nhiên, đó là một trong số ít các thuật toán sắp xếp không xuất hiện từ các phòng học. Nó được Tim Peters triển khai vào năm 2002 để sử dụng trong ngôn ngữ lập trình Python. Thuật toán tìm kiếm hệ quả của dữ liệu đã có thứ tự và sử dụng kết quả này để sắp xếp phần còn lại hiệu quả hơn. Sắp xếp này là sắp xếp mặc định trong ngôn ngữ lập trình python. trong phiên bản python> = 2.3, chỉ sắp xếp (mảng) sẽ áp dụng sắp xếp theo thời gian trên dữ liệu. Mặc dù trọng tâm để phát triển điều này chỉ là ngôn ngữ python, nhưng nó cũng được sử dụng rộng rãi trong các nền tảng khác, chẳng hạn như JAVA, ANDROID, Google Chrome,...

3.8.2.Hoạt động

Timsort được thiết kế để tận dụng lợi thế của chạy *của các yếu* tố thứ tự liên tiếp đã tồn tại trong hầu hết các dữ liệu trong thế giới thực, chạy tự *nhiên.* Nó tuần tự trên các yếu tố thu thập dữ liệu vào chạy và đồng thời đặt những chạy trong một ngăn xếp. Bất cứ khi nào chạy trên đầu trang của ngăn xếp phù hợp với một [tiêu chí hợp](https://en.wikipedia.org/wiki/Timsort#Merge_criteria) nhất, chúng được sáp nhập. Điều này diễn ra cho đến khi tất cả dữ liệu được đi qua; sau đó, tất cả các chạy được sáp nhập hai tại một thời điểm và chỉ có một sắp xếp chạy vẫn còn. Lợi thế của việc sáp nhập thứ tự chạy thay vì sáp nhập cố định kích thước tiểu danh sách (như được thực hiện bởi truyền thống mergesort) là nó làm giảm tổng số so sánh cần thiết để sắp xếp toàn bộ danh sách.

Mỗi lần chạy có kích thước tối thiểu, dựa trên kích thước của đầu vào và nó được định nghĩa ở đầu thuật toán. Nếu chạy nhỏ hơn kích thước chạy tối thiểu này, [sắp xếp chèn được](https://en.wikipedia.org/wiki/Insertion_sort) sử dụng để thêm nhiều yếu tố để chạy cho đến khi đạt đến kích thước chạy tối thiểu.

3.8.3.Ràng buộc của phép hợp

Timsort là một thuật toán phân loại ổn định (thứ tự các yếu tố có cùng khóa được giữ) và cố gắng thực hiện hợp nhất cân bằng (hợp nhất do đó hợp nhất các lần chạy có kích thước tương tự).

Để đạt được sự ổn định phân loại, chỉ chạy liên tiếp được sáp nhập. Giữa hai lần chạy không liên tiếp, có thể có một phần tử với cùng một phím của các yếu tố bên trong chạy và sáp nhập hai chạy sẽ thay đổi thứ tự của các phím bằng nhau.

Trong việc theo đuổi hợp nhất cân bằng, Timsort xem xét ba chạy trên đầu trang của ngăn *xếp, X,* *Y,* *Z,*và duy trì các bất biến:

1. | *Z*| > | *Y*| + | *X*|
2. | *Y*| > | *X*| [[11]](https://en.wikipedia.org/wiki/Timsort#cite_note-python_timsort-11)

Nếu bất kỳ bất đẳng thức nào trong số này bị vi phạm, *Y* được hợp nhất với các bất biến *nhỏ hơn X* hoặc *Z* và các bất biến được kiểm tra lại. Một khi bất đẳng thức đúng, việc tìm kiếm một chạy mới trong dữ liệu có thể bắt đầu.Những bất đẳng thức này duy trì sự hợp nhất như là xấp xỉ cân bằng trong khi duy trì sự thỏa hiệp giữa việc trì hoãn sáp nhập để cân bằng, khai thác sự xuất hiện mới của chạy trong bộ nhớ [cache](https://en.wikipedia.org/wiki/CPU_cache) và đưa ra quyết định hợp nhất tương đối đơn giản.

3.8.4.Ý Tưởng

Chúng ta chia Mảng thành các khối được gọi là Run. Chúng ta sắp xếp các chạy bằng cách sử dụng chèn sắp xếp từng cái một và sau đó hợp nhất các chạy bằng cách sử dụng chức năng kết hợp được sử dụng trong sắp xếp hợp nhất. Nếu kích thước của Mảng nhỏ hơn chạy, thì Array được sắp xếp chỉ bằng cách sử dụng Sắp xếp Chèn. Kích thước của chạy có thể khác nhau từ 32 đến 64 tùy thuộc vào kích thước của mảng. Ý tưởng này dựa trên thực tế là sắp xếp chèn hoạt động tốt cho các mảng nhỏ.

Chi tiết thực hiện

* Xây dựng một minrun có kích thước nhỏ hơn 64. Thông thường chúng ta cần lấy một số là lũy thừa của 2.Chúng ta thường lấy 32.
* Tìm kiếm lần chạy dữ liệu trong phút đầu tiên.
* Nếu chúng ta không thể tìm thấy đoạn chạy có độ dài ít nhất là minrun, chúng ta cần sử dụng Insertion Sort để lấy các mục tiếp theo và đẩy chúng vào chạy cho đến khi có kích thước chính xác.
* Chúng ta cần lặp lại cho đến khi toàn bộ mảng được chia thành các phần con đã sắp xếp.
* Bây giờ chúng ta cần sử dụng Merge Sort để nối các mảng có thứ tự.

*3.9. Selection Sort*

3.9.1. Ý tưởng thuật toán

Chọn phần tử nhỏ nhất đưa về vị trí đầu tiên của mảng hiện tại và không cần quan tâm đến nó nữa. Khi đó mảng chỉ còn lại n - 1 phần tử của mảng ban đầu, tiếp tục xét từ phần tử thứ 2 của mảng.

Lặp lại cho đến khi dãy hiện tại chỉ còn 1 phần tử.

3.9.2.Minh họa sắp xếp Selection Sort

**62**

**24**

**15**

**1**

**22**

**1**

**24**

**15**

**22**

**62**

**1**

**15**

**24**

**22**

**62**

**1**

**15**

**22**

**24**

**62**

**1**

**15**

**22**

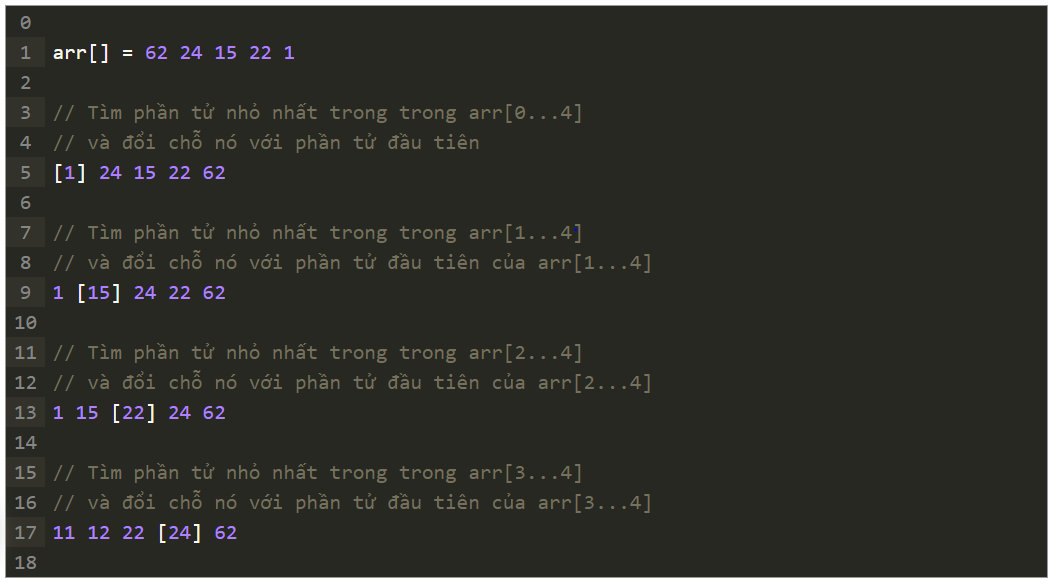
**24**

**62**

3.9.3 Ý tưởng cài đặt

* Bước 1: i = 0.
* Bước 2: Tìm phần tử a[iMin] trong dãy hiện hành từ a[i] đến a[n-1].
* Bước 3: Đổi chỗ a[i] và a[iMin].
* Bước 4: Nếu i < n - 1 thì lặp lại bước 2 với i++ - Ngược lại thì dừng.

Ví dụ:



## 3.9.4 Ưu và nhược điểm

* Số lần so sánh trong trường hợp tốt nhất là n(n-1)/2
* Số lần so sánh trong trường hợp xấu nhất là 3n(n-1)/2

### Ưu điểm

* Thuật toán đơn giản, dễ hiện thực.
* Có số lần hoán đổi các vị trí ít.

### Nhược điểm

* Chỉ được áp dụng trong các trường hợp có số lượng phần tử cần so sánh ít.
* Không nhận biết được mảng đã được sắp xếp.

*3.10.Tree sort*

3.10.1 Giới thiệu

Sắp xếp cây là một thuật toán sắp xếp trực tuyến xây dựng một cây tìm kiếm nhị phân từ các phần tử được sắp xếp, sau đó duyệt qua cây (theo thứ tự) để các phần tử xuất hiện theo thứ tự được sắp xếp.

3.10.2.Ý tưởng cài đặt

Bước 1: Lấy các phần tử đầu vào trong một mảng.

Bước 2: Tạo cây tìm kiếm nhị phân bằng cách chèn các mục dữ liệu từ mảng vào cây tìm kiếm nhị phân.

Bước 3: Thực hiện duyệt theo thứ tự trên cây để lấy các phần tử theo thứ tự đã sắp xếp.

* Phức tạp
* -Độ phức tạp thời gian của trường hợp xấu nhất: Θ (N log N) sử dụng cây tìm kiếm nhị phân cân bằng; Θ (N ^ 2) sử dụng cây tìm kiếm nhị phân không cân bằng.
* -Độ phức tạp thời gian trung bình của trường hợp: Θ (N log N)
* -Độ phức tạp của trường hợp tốt nhất: Θ (N log N)
* -Độ phức tạp không gian: Θ (N).

3.10.3.Ví dụ

27

14

35

10

19

31

42

Từ hình ví dụ minh họa trên ta thấy rằng, khóa của nút gốc có giá trị 27 và tất cả khóa bên trái của cây con bên trái đều có giá trị nhỏ hơn 27 này và tất cả các khóa bên phải của cây con bên phải đều có giá trị lớn hơn 27.

### **Hoạt động cơ bản trên cây tìm kiếm nhị phân**

Dưới đây là một số hoạt động cơ bản có thể được thực hiện trên cây tìm kiếm nhị phân:

* ***Hoạt động tìm kiếm****: tìm kiếm một phần tử trong một cây.*
* ***Hoạt động chèn****: chèn một phần tử vào trong một cây.*
* ***Hoạt động duyệt tiền thứ tự****: duyệt một cây theo cách thức duyệt tiền thứ tự.*
* ***Hoạt động duyệt trung thứ tự****: duyệt một cây theo cách thứ duyệt trung thứ tự.*
* ***Hoạt động duyệt hậu thứ tự****: duyệt một cây theo cách thức duyệt hậu thứ tự.*

**4.Code:**

#include<iostream>

#include <bits/stdc++.h>

#include <ctime>

#include <cstdio>

#include<fstream>

#include <time.h>

#include <cstdlib>

#include<string>

#include <unistd.h>

#include<graphics.h>

using namespace std;

int time\_delay=500;

float thoigian[6];

float thoigiantb[6];

class Source

{

public:

void xuatmang(int arr[],int n)

{

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

{

cout<<arr[i]<<"\n";

}

}

void taoarr(int arr[],int n)

{

srand(time(NULL));

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

{

arr[i]=rand();

}

}

void laymangfile(char\* s,int arr[])

{

fstream fsFile;

fsFile.open(s, ios::in);

if (fsFile.eof())

{

cout << "Read done";

}

else

{

int tam ;

int i = 0;

while (fsFile>>tam)

{

//cout<<tam<<"\n";

arr[i]=tam;

i++;

}

}

fsFile.close();

}

};

class Source100: public Source{

public:

int dulieu100[100];

int n=100;

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

class Source250:public Source{

public:

int dulieu250[250];

int n=250;

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

class Source500: public Source{

public:

int dulieu500[500];

int n=500;

void laymangfile(char \*s,int arr[]){

Source::laymangfile(s,arr);

}

void taomang(int dl[],int n)

{

Source::taoarr(dl,n);

}

void xuatmang(int arr[],int n)

{

Source::xuatmang(arr,n);

}

};

class Sort{

public:

void inmang(int arr[],int n)

{

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

cout << arr[i] << " ";

}

void hinhchunhat(int x1,int y1,int x2,int y2)

{

line(x1,y1,x1,y2);

line(x1,y2,x2,y2);

line(x2,y2,x2,y1);

line(x2,y1,x1,y1);

}

//in ra man hinh day so theo do hoa

void screen(int arr[], int n)

{

int xstart=0,ystart=0;

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

{

hinhchunhat(xstart+i\*800/n, 0, xstart+(i+1)\*800/n, arr[i]/100);

}

}

};

class radix: public Sort{

public:

//Lay so lon nhat trong day

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

//sap xep day so

void countSort(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = { 0 };

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

{

count[(arr[i] / exp) % 10]++;

// delay(10);

}

for (i = 1; i < 10; i++)

{

count[i] += count[i - 1];

//delay(10);

}

for (i = n - 1; i >= 0; i--) {

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

//delay(10);

}

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

{

arr[i] = output[i];

//delay(10);

}

}

/\* void countSort\_dohoa(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = { 0 };

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

count[(arr[i] / exp) % 10]++;

for (i = 1; i < 10; i++)

{

count[i] += count[i - 1];

}

for (i = n - 1; i >= 0; i--) {

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

cleardevice();

Sort::screen(output,n);

//delay(10);

}

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

arr[i] = output[i];

cleardevice();

Sort::screen(arr,n);

delay(10);

}\*/

/\*float radixsort\_dohoa(int arr[],int n)

{

clock\_t begin = clock();

initwindow(800,1000);

// Tim so lon nhat de lay so chu so nhieu nhat

int m = getMax(arr, n);

// Thuc hien sap xep theo vi tri chu so, exp la vi tri chu so, i la so hien tai

for (int exp = 1; m / exp > 0; exp \*= 10)

{

countSort\_dohoa(arr, n, exp);

}

getch();

clock\_t end = clock();

//cout<<begin<<"\t"<<end<<endl;

return (float)((end-begin)/CLOCKS\_PER\_SEC)\*1000;

}\*/

float radixsort(int arr[], int n)

{

//auto start = high\_resolution\_clock::now();

clock\_t t,t2;

// Tim so lon nhat de lay so chu so nhieu nhat

t = clock();

int m = getMax(arr, n);

// Thuc hien sap xep theo vi tri chu so, exp la vi tri chu so, i la so hien tai

for (int exp = 1; m / exp > 0; exp \*= 10)

{

countSort(arr, n, exp);

}

sleep(1);

t2=clock();

float time\_taken = ((float)t)/CLOCKS\_PER\_SEC; // in seconds

float time\_taken2 = ((float)t2)/CLOCKS\_PER\_SEC;

return (((float)time\_taken2-time\_taken-1)/CLOCKS\_PER\_SEC)\*1000;

}

// In mang sau khi sap xep

void inmang(int arr[], int n)

{

Sort::inmang(arr,n);

}

};

class insert:public Sort{

public:

float insertSort(int arr[], int n)

{

clock\_t t1,t2;

t1 = clock();

int i, key, j;

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

{

key = arr[i];

j = i - 1;

// Dich chuyen phan tu len truoc neu phan tu lon hon key

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

sleep(1);

t2 = clock();

float time\_taken1 = ((float)t1)/CLOCKS\_PER\_SEC; // in seconds

float time\_taken2 = ((float)t2)/CLOCKS\_PER\_SEC;

return (((float)time\_taken2-time\_taken1-1)/CLOCKS\_PER\_SEC)\*1000;

}

/\* float insertionSort\_dohoa(int arr[], int n)

{

int i, key, j;

initwindow(800,1000);

clock\_t begin = clock();

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

{

cleardevice();

key = arr[i];

j = i - 1;

// Dich chuyen phan tu len truoc neu phan tu lon hon key

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

screen(arr,n);

delay(10);

}

clock\_t end = clock();

return (float)((end-begin)/CLOCKS\_PER\_SEC);

//closewindow();

}\*/

// ham in mang

void inmang(int arr[], int n)

{

Sort::inmang(arr,n);

}

};

class SelectionSort: public Sort

{

// Ham doi cho hai so nguyen

public: void swap(int\* xp, int\* yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// Hàm selection sort

public: void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// di chuyen ranh gioi cua mang da sap xep va chua sap xep

for (i = 0; i < n - 1; i++)

{

// tim phan tu nho nhat trong mang chua sap xep

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

// doi cho phan tu nho nhat voi phan tu dau tien

swap(&arr[min\_idx], &arr[i]);

}

}

/\* ham xuat mang \*/

public: void printArray(int arr[], int size)

{

Sort::inmang(arr,size);

}

double t\_SelectionSort(int arr[],int n)

{

clock\_t star = clock();

selectionSort(arr,n);

// printArray(arr,n);

sleep(1);

clock\_t end = clock();

double d = (double)((end-star)-1)/CLOCKS\_PER\_SEC;

return d;

}

};

class TreeSort: public Sort {

public: int key;

TreeSort\* left, \* right;

public: TreeSort\* newNode(int item) {

TreeSort\* temp = new TreeSort;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

public:

Node() {

}

public: void storeSorted(TreeSort\* root, int arr[], int& i) {

if (root != NULL) {

storeSorted(root->left, arr, i);

arr[i++] = root->key;

storeSorted(root->right, arr, i);

}

}

// Mot chuc nang tien ích de chèn mot

// Nút voi khóa nhat dinh

public: TreeSort\* insert(TreeSort\* node, int key) {

//Neu cây trong == null, tra ve mot Node moi

if (node == NULL) return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

return node;

}

// Hàm này sap xep arr [0..n-1] bang cách su dung Tree Sort

public: void treeSort(int arr[], int n) {

struct TreeSort\* root = NULL;

// xay dung

root = insert(root, arr[0]);

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

root = insert(root, arr[i]);

// luu tru truyen tai

int i = 0;

storeSorted(root, arr, i);

}

void inmang(int arr[],int n)

{

Sort::inmang(arr,n);

}

double t\_TreeSort(int arr[],int n)

{

clock\_t star = clock();

//inmang(arr,n);

treeSort(arr,n);

sleep(1);

clock\_t end = clock();

double d = (double)((end-star)-1)/CLOCKS\_PER\_SEC;

return d;

}

};

class MergeSort: public Sort

{

public:

void merge(int arr[], int l, int m, int r)

{

// tao 2 mang con

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

// sao chep du lieu vao 2 mang

for (int i = 0; i < n1; i++)

L[i] = arr[l + i];

for (int j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

int i = 0; // khoi tao chi so bat dau cua mang L

int j = 0; // khoi tao chi so bat dau cua mang R

int k = l; // khoi tao chi so bat dau cua mang luu ket qua

// thuc hien tron code

while (i < n1 && j < n2)

{

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

// sao chep cac phan tu con lai vao mang neu co

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

// Ham de qui sap xep chinh

void mergeSort(int arr[], int l, int r)

{

if(l<r)

{

int m = ( l + r ) / 2;

// de qui tach mang

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

// tron code

merge(arr, l, m, r);

}

}

void InMang(int arr[], int n)

{

Sort::inmang(arr, n);

}

// tinh thoi gian thuc hien thuat toan MergeSort

double t\_MergeSort (int arr[], int n)

{

int l = 0; // arr bat dau tai l

int r = n - 1; // ket thuc tai r

clock\_t star = clock();

mergeSort(arr, l, r);

sleep(1);

clock\_t end = clock();

double d = (double)(end - star - 1) / CLOCKS\_PER\_SEC;

return d;

}

};

class HeapSort : public Sort

{

public:

// tim max heap

void heapify(int arr[], int n, int i)

{

int largest = i; // khoi tao largest nhu la root

int l = 2 \* i + 1; // left = 2\*i + 1

int r = 2 \* i + 2; // right = 2\*i + 2

// Neu nut con trai lon hon so voi root

if (l < n && arr[l] > arr[largest])

largest = l;

// Neu nut con phai lon hon so voi root

if (r < n && arr[r] > arr[largest])

largest = r;

// Neu root khong phai la lon nhat

if (largest != i)

{

swap(arr[i], arr[largest]);

// De quy lai ham heapify

heapify(arr, n, largest);

}

}

// Ham vun dong

void heapSort(int arr[], int n)

{

// Tao mot dong (Sap xep lai mang)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--)

{

// Di chuyen root ve cuoi cung

swap(arr[0], arr[i]);

// goi ham heapify

heapify(arr, i, 0);

}

}

void InMang (int arr[], int n)

{

Sort::inmang(arr, n);

}

// tinh thoi gian thuc hien thuat toan HeapSort

double t\_HeapSort(int arr[], int n)

{

clock\_t star = clock();

heapSort(arr, n);

sleep(1);

clock\_t end = clock();

double d = (double)(end - star - 1) / CLOCKS\_PER\_SEC;

return d;

}

};

class QuickSort: public Sort

{ public :

void HoanVi(int& x, int& y)

{

int temp = x;

x = y;

y = temp;

}

void InMang(int arr[], int n)

{

cout << "\nQuick Sort:\n";

Sort::inmang(arr, n);

}

void quickSort(int a[], int left, int right)

{

int i, j, x;

if (left >= right)

{

return;

}

x = a[(left + right) / 2];

i = left;

j = right;

while (i < j)

{

while (a[i] > x)

{

i++;

}

while (a[j] < x)

{

j--;

}

if (i <= j)

{

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

i++; j--;

}

}

quickSort(a, left, j);

quickSort(a, i, right);

}

double t\_QuickSort (int arr[], int n)

{

int j = 1000;

clock\_t start1 = clock();

quickSort(arr, 0,n - 1);

sleep(1);

clock\_t finish1 = clock();

double duration1 = (double)(finish1-1 - start1) / (CLOCKS\_PER\_SEC);

return duration1;

}

};

class BubbleSort : public Sort

{

public:

void HoanVi(int& x, int& y)

{

int temp = x;

x = y;

y = temp;

}

void bubbleSort(int arr[],int n)

{

for (int i = 0; i < n - 1; i++)

{

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

{

if (arr[j] > arr[j - 1])

{

HoanVi(arr[j], arr[j - 1]);

}

}

}

}

void InMang (int arr[], int n)

{

cout << "\nBubble Sort:\n";

Sort::inmang(arr, n);

}

// tinh thoi gian thuc hien thuat toan HeapSort

double t\_BubbleSort(int arr[], int n)

{

clock\_t start2 = clock();

bubbleSort(arr, n);

sleep(1);

clock\_t finish2 = clock();

double duration2 = (double)(finish2 - start2-1) / CLOCKS\_PER\_SEC;

return duration2;

}

};

class CoutingSort : public Sort {

public: void CountSort(int arr[], int n) {

int arr1[40000];

int count\_arr[40000];

int x = arr[0];

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

if (arr[i] > x)

x = arr[i];

}

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

count\_arr[i] = 0;

}

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

count\_arr[arr[i]]++;

}

for (int i = 1; i <= x; i++) {

count\_arr[i] += count\_arr[i - 1];

}

for (int i = n - 1; i >= 0; i--) {

arr1[count\_arr[arr[i]] - 1] = arr[i];

count\_arr[arr[i]]--;

}

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

arr[i] = arr1[i];

}

}

public: void printArray(int arr[], int n)

{

Sort::inmang(arr, n);

}

public: double ThoiGianThucHien(int arr[], int n)

{

clock\_t star = clock();

CountSort(arr,n);

sleep(1);

clock\_t end = clock();

double d = (double)(end - star-1) / (CLOCKS\_PER\_SEC );

return d;

}

};

class timSort : public Sort {

const int RUN = 32;

public: void insertionSort(int arr[], int left, int right)

{

for (int i = left + 1; i <= right; i++)

{

int temp = arr[i];

int j = i - 1;

while (j >= left && arr[j] > temp)

{

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = temp;

}

}

// Chuc nang tron tron cac run da sap xep lai voi nhau

public:void merge(int arr[], int l, int m, int r)

{

// Mang ban dau duoc chia lam hai mang trai va phai

// len1 => chieu dai mang trai && len2 => chieu dai mang phai

int len1 = m - l + 1, len2 = r - m;

int left[len1], right[len2];

for (int i = 0; i < len1; i++)

left[i] = arr[l + i];

for (int i = 0; i < len2; i++)

right[i] = arr[m + 1 + i];

int i = 0;

int j = 0;

int k = l;

//Sau khi so sanh, ta tron 2 mang nay vao mot mang lon hon

while (i < len1 && j < len2)

{

if (left[i] <= right[j])

{

arr[k] = left[i];

i++;

}

else

{

arr[k] = right[j];

j++;

}

k++;

}

//Sao nhung phan tu con lai ben trai, neu co

while (i < len1)

{

arr[k] = left[i];

k++;

i++;

}

//Sao nhung phan tu con lai ben phai, neu co

while (j < len2)

{

arr[k] = right[j];

k++;

j++;

}

}

// Lap lai ham Timsort de sap xep arr[0..n-1] ( tuong tu voi sap xep tron)

public:void Sort(int arr[], int n)

{

// Sap xep tung mang con voi kich thuoc RUN

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

insertionSort(arr, i, min((i + 31),

(n - 1)));

//Bat dau tron tu kich thuoc RUN ( hoac 32)

//No se tron den kich thuoc 64, sau do la 128, den 256 ....

for (int size = RUN; size < n;

size = 2 \* size)

{

//Chon diem bat dau cua mang ben trai.Ta bat dau tron arr[left..left+2\*size-1]

for (int left = 0; left < n;

left += 2 \* size)

{

//Tim diem ket thuc cua mang ben trai

//mid+1 la diem bat dau cua mang ben phai

int mid = left + size - 1;

int right = min((left + 2 \* size - 1),

(n - 1));

// tron nhung mang con arr[left.....mid] & arr[mid+1....right]

merge(arr, left, mid, right);

}

}

}

public: void printArray(int arr[], int n)

{

Sort::inmang(arr, n);

}

public: double ThoiGianThucHien(int arr[], int n)

{

clock\_t star = clock();

Sort(arr,n);

sleep(1);

clock\_t end = clock();

double d = (double)(end - star-1) / (CLOCKS\_PER\_SEC );

return d;

}

};

int main()

{

/\*Source100 s100;

Source250 s250;

Source500 s500;

radix ra100,ra250,ra500;

insert ins100,ins250,ins500;

//s100.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao100.txt",s100.dulieu100);

s100.taomang(s100.dulieu100,s100.n);

//s250.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao250.txt",s250.dulieu250);

s250.taomang(s250.dulieu250,s250.n);

//s500.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao500.txt",s500.dulieu500);

s500.taomang(s500.dulieu500,s500.n);

cout<<"size\t\t100\t\t250\t\t500"<<endl;

for(int i=1;i<10;i++)

{

thoigian[0]=ra100.radixsort(s100.dulieu100,s100.n);

thoigian[1]=ra250.radixsort(s250.dulieu250,s250.n);

thoigian[2]=ra500.radixsort(s500.dulieu500,s500.n);

thoigian[3]=ins100.insertSort(s100.dulieu100,s100.n);

thoigian[4]=ins250.insertSort(s250.dulieu250,s250.n);

thoigian[5]=ins500.insertSort(s500.dulieu500,s500.n);

printf("Radix Sort\t%f\t%f\t%f\n",thoigian[0],thoigian[1],thoigian[2]);

printf("insert Sort\t%f\t%f\t%f",thoigian[3],thoigian[4],thoigian[5]);

if(i==1)

{

thoigiantb[0]=thoigian[0];

thoigiantb[1]=thoigian[1];

thoigiantb[2]=thoigian[2];

thoigiantb[3]=thoigian[3];

thoigiantb[4]=thoigian[4];

thoigiantb[5]=thoigian[5];

}

else

{

thoigiantb[0]=(thoigiantb[0]+thoigian[0])/2;

thoigiantb[1]=(thoigiantb[1]+thoigian[1])/2;

thoigiantb[2]=(thoigiantb[2]+thoigian[2])/2;

thoigiantb[3]=(thoigiantb[3]+thoigian[3])/2;

thoigiantb[4]=(thoigiantb[4]+thoigian[4])/2;

thoigiantb[5]=(thoigiantb[5]+thoigian[5])/2;

}

printf("\n\n");

}

cout<<"Thoi gian trung binh la:\n";

printf("Radix Sort\t%f\t%f\t%f\n",thoigiantb[0],thoigiantb[1],thoigiantb[2]);

printf("insert Sort\t%f\t%f\t%f",thoigiantb[3],thoigiantb[4],thoigiantb[5]);

cout<<"\nXong";\*/

Source100 s100;

Source250 s250;

Source500 s500;

radix ra100,ra250,ra500;

insert ins100,ins250,ins500;

s100.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao100.txt",s100.dulieu100);

//s100.taomang(s100.dulieu100,s100.n);

s250.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao250.txt",s250.dulieu250);

//s250.taomang(s250.dulieu250,s250.n);

s500.laymangfile("D:\\Thuctapcoso\\ThucTapCoSo\\dauvao500.txt",s500.dulieu500);

//s500.taomang(s500.dulieu500,s500.n);

int choose;

menu:

cout<<"-------------------------"<<endl;

cout<<"1.Merge Sort & Heap Sort"<<endl;

cout<<"2.Radix Sort & Insert Sort"<<endl;

cout<<"3.Bubble Sort & Quick Sort"<<endl;

cout<<"4.Selection Sort & Tree Sort"<<endl;

cout<<"5.Counting Sort & Tim Sort"<<endl;

cout<<"0. Thoat chuong trinh"<<endl;

cout<<"-------------------------"<<endl;

cout<<"Nhap lua chon cua ban:"<<endl;

cout<<">>";cin>>choose;

switch(choose)

{

case 0: return -1;

break;

case 1:

{

MergeSort ms100, ms250, ms500;

HeapSort hs100, hs250, hs500;

double sumMS100=0, sumMS250=0, sumMS500=0;

double sumHS100=0, sumHS250=0, sumHS500=0;

cout << "Size\t\t100\t250\t500";

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

{

//cout << "\nMerge Sort\t" << ms100.t\_MergeSort(s100\_1.dulieu100,s100\_1.n) << "\t" << ms250.t\_MergeSort(s250\_1.dulieu250,s250\_1.n) << "\t" << ms500.t\_MergeSort(s500\_1.dulieu500,s500\_1.n);

//cout << "\nHeap Sort\t" << hs100.t\_HeapSort(s100\_2.dulieu100,s100\_2.n) << "\t" << hs250.t\_HeapSort(s250\_2.dulieu250,s250\_2.n) << "\t" << hs500.t\_HeapSort(s500\_2.dulieu500,s500\_2.n);

sumMS100 = sumMS100 + ms100.t\_MergeSort(s100.dulieu100,s100.n);

sumMS250 = sumMS250 + ms250.t\_MergeSort(s250.dulieu250,s250.n);

sumMS500 = sumMS500 + ms500.t\_MergeSort(s500.dulieu500,s500.n);

sumHS100 = sumHS100 + hs100.t\_HeapSort(s100.dulieu100,s100.n);

sumHS250 = sumHS250 + hs250.t\_HeapSort(s250.dulieu250,s250.n);

sumHS500 = sumHS500 + hs500.t\_HeapSort(s500.dulieu500,s500.n);

}

cout << "\nMerge Sort :\t" << sumMS100/10 << "\t" << sumMS250/10 << "\t" << sumMS500/10;

cout << "\nHeap Sort :\t" << sumHS100/10 << "\t" << sumHS250/10 << "\t" << sumHS500/10;

}

case 2:

{

radix ra100,ra250,ra500;

insert ins100,ins250,ins500;

double sumRD100=0, sumRD250=0, sumRD500=0;

double sumIN100=0, sumIN250=0, sumIN500=0;

for(int i=1;i<10;i++)

{

sumRD100 +=ra100.radixsort(s100.dulieu100,s100.n);

sumRD250 +=ra250.radixsort(s250.dulieu250,s250.n);

sumRD500 +=ra500.radixsort(s500.dulieu500,s500.n);

sumIN100+=ins100.insertSort(s100.dulieu100,s100.n);

sumIN250+=ins250.insertSort(s250.dulieu250,s250.n);

sumIN500+=ins500.insertSort(s500.dulieu500,s500.n);

}

cout<<"size\t\t100\t\t250\t\t500"<<endl;

printf("Radix Sort\t%f\t%f\t%f\n",sumRD100/10,sumRD250/10,sumRD500/10);

printf("insert Sort\t%f\t%f\t%f",sumIN100/10,sumIN250/10,sumIN500/10);

}

case 3:

{

QuickSort qs100, qs250, qs500;

BubbleSort bs100, bs250, bs500;

double sqs100=0, sqs250=0, sqs500=0;

double sbs100=0, sbs250=0, sbs500=0;

// hien thi

cout << "Size\t\t100\t250\t500";

// cout << "\nQuick Sort\t" << qs100.t\_QuickSort(s100\_1.dulieu100,s100\_1.n) << "\t" << qs250.t\_QuickSort(s250\_1.dulieu250,s250\_1.n) << "\t" << qs500.t\_QuickSort(s500\_1.dulieu500,s500\_1.n);

//cout << "\nbubble Sort\t" << hs100.t\_BubbleSort(s100\_2.dulieu100,s100\_2.n) << "\t" << hs250.t\_BubbleSort(s250\_2.dulieu250,s250\_2.n) << "\t" << hs500.t\_BubbleSort(s500\_2.dulieu500,s500\_2.n);

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

{ sqs100 = sqs100 + qs100.t\_QuickSort(s100.dulieu100,s100.n);

sqs250 = sqs250 + qs250.t\_QuickSort(s250.dulieu250,s250.n);

sqs500 = sqs500 + qs500.t\_QuickSort(s500.dulieu500,s500.n);

sbs100 = sbs100 + bs100.t\_BubbleSort(s100.dulieu100,s100.n);

sbs250 = sbs250 + bs250.t\_BubbleSort(s250.dulieu250,s250.n);

sbs500 = sbs500 + bs500.t\_BubbleSort(s500.dulieu500,s500.n);

}

cout << "\nQuick Sort :\t" << sqs100/10 << "\t" << sqs250/10 << "\t" << sqs500/10;

cout << "\nBubble Sort :\t" << sbs100/10 << "\t" << sbs250/10 << "\t" << sbs500/10;

}

case 4:

{

SelectionSort ss100, ss250, ss500;

TreeSort ts100, ts250, ts500;

ss100.selectionSort(s100.dulieu100,s100.n);

ss250.selectionSort(s250.dulieu250,s250.n);

ss500.selectionSort(s500.dulieu500,s500.n);

double sumSS100=0, sumSS250=0, sumSS500=0;

double sumTS100=0, sumTS250=0, sumTS500=0;

cout << "\nsize\t\t100\t250\t500";

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

{

// cout<<"\n----------------------------------------------------";

// cout << "\nSelection Sort\t" << ss100.t\_SelectionSort(s100\_1.dulieu100,s100\_1.n) << "\t" << ss250.t\_SelectionSort(s250\_1.dulieu250,s250\_1.n) << "\t" << ss500.t\_SelectionSort(s500\_1.dulieu500,s500\_1.n);

// cout << "\nTree Sort\t" << ts100.t\_TreeSort(s100\_1.dulieu100,s100\_1.n) << "\t" << ts250.t\_TreeSort(s250\_1.dulieu250,s250\_1.n) << "\t" << ts500.t\_TreeSort(s500\_1.dulieu500,s500\_1.n);

sumSS100 = sumSS100 + ss100.t\_SelectionSort(s100.dulieu100,s100.n);

sumSS250 = sumSS250 + ss250.t\_SelectionSort(s250.dulieu250,s250.n);

sumSS500 = sumSS500 + ss500.t\_SelectionSort(s500.dulieu500,s500.n);

sumTS100 = sumTS100 + ts100.t\_TreeSort(s100.dulieu100,s100.n);

sumTS250 = sumTS250 + ts250.t\_TreeSort(s250.dulieu250,s250.n);

sumTS500 = sumTS500 + ts500.t\_TreeSort(s500.dulieu500,s500.n);

}

cout << "\nSelect Sort:\t" << sumSS100/10 << "\t" << sumSS250/10 << "\t" << sumSS500/10;

cout << "\nTree Sort:\t" << sumTS100/10 << "\t" << sumTS250/10 << "\t" << sumTS500/10;

}

case 5:

{

timSort tsort100, tsort250, tsort500;

CoutingSort csort100, csort250, csort500;

double sumTimS100=0, sumTimS250=0, sumTimS500=0;

double sumCountS100=0, sumCountS250=0, sumCountS500=0;

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

// tinh thoi gian trung binh sau 10 lan thuc hien thuat toan

sumTimS100 = sumTimS100 + tsort100.ThoiGianThucHien(s100.dulieu100, s100.n);

sumTimS250 = sumTimS250 + tsort250.ThoiGianThucHien(s250.dulieu250, s250.n);

sumTimS500 = sumTimS500 + tsort500.ThoiGianThucHien(s500.dulieu500, s500.n);

sumCountS100 = sumCountS100 + csort100.ThoiGianThucHien(s100.dulieu100, s100.n);

sumCountS250 = sumCountS250 + csort250.ThoiGianThucHien(s250.dulieu250, s100.n);

sumCountS500 = sumCountS500 + csort500.ThoiGianThucHien(s500.dulieu500,s500.n);

}

cout << "\nMerge Sort :\t" << sumTimS100/10 << "\t" << sumTimS250/10 << "\t" << sumTimS500/10;

cout << "\nHeap Sort :\t" << sumCountS100/10 << "\t" << sumCountS250/10 << "\t" << sumCountS500/10;

}

}

}