BỘ GIÁO DỤC ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**---🙠**🕮**🙢---**



**THỰC TẬP CƠ SỞ**

**GIẢI THUẬT SẮP XẾP**

**(Bubble Sort, Quick Sort, Merge Sort, Insertion Sort)**

**Giảng viên hướng dẫn: Nguyễn Hải Triều**

**Sinh viên thực hiện: Trương Thị Diễm Quỳnh**

**Mã số sinh viên: 611****36433**

Nha Trang 12/2021

# LỜI NÓI ĐẦU

**Lời cảm ơn:**

Lời đầu tiên, cho phép em gửi lời cảm ơn sâu sắc và chân thành nhất đến quý Thầy/Cô và các bạn học đã tạo điều kiện giúp em trong suốt quá trình học tập và thực hiện đề tài. Sự quan tâm và giúp đỡ của quý Thầy/Cô và các bạn học là nguồn động viên rất lớn giúp em hoàn thành tốt đề tài này.

Với lòng biết ơn sâu sắc nhất, em xin gửi đến quý Thầy/Cô ở Khoa Công Nghệ Thông Tin đã truyền đạt vốn kiến thức quý báu cho chúng em trong suốt thời gian học tập tại trường. Nhờ có những lời hướng dẫn, dạy bảo của các Thầy/Cô nên đề tài nghiên cứu của em mới có thể hoàn thiện tốt đẹp.

Một lần nữa, em xin chân thành cảm ơn những Thầy/Cô – người đã trực tiếp giúp đỡ, quan tâm, hướng dẫn chúng em hoàn thành tốt bài báo cáo này trong thời gian qua.

Bước đầu đi vào thực tế của em còn nhiều hạn chế và bỡ ngỡ nên không tránh khỏi những thiếu sót, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy/Cô để kiến thức của em trong bài báo cáo này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Em xin chân thành cảm ơn!

**Mục tiêu cần đạt được:**

Giới thiệu về khái niệm và giải thuật của một số thuật toán sắp xếp cơ bản (Quick sort, Bubble sort, Insertion sort, Merge sort).Phân tích, đánh giá độ phức tạp của các giải thuật sắp xếp. Giải thuật và cài đặt thuật toán (trên ngôn ngữ C). Các ứng dụng các thuật toán trong thực tế.

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc92203842)

[Chương 1: TỔNG QUAN ĐỀ TÀI 4](#_Toc92203843)

[1. LÝ DO CHỌN ĐỀ TÀI 4](#_Toc92203844)

[2. MÔI TRƯỜNG CÀI ĐẶT 4](#_Toc92203845)

[3. PHƯƠNG PHÁP NGHIÊN CỨU 4](#_Toc92203846)

[Chương 2: CƠ SỞ LÝ THUYẾT 5](#_Toc92203847)

[I. ĐỊNH NGHĨA SẮP XẾP 5](#_Toc92203848)

[II. THUẬT TOÁN 5](#_Toc92203849)

[1. THUẬT TOÁN SẮP XẾP NỔI BỌT (BUBBLE SORT) 5](#_Toc92203850)

[2. THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT) 6](#_Toc92203851)

[3. THUẬT TOÁN SẮP XẾP CHÈN (INSERTION SORT) 8](#_Toc92203852)

[4. THUẬT TOÁN SẮP XẾP NHANH (QUICK SORT) 9](#_Toc92203853)

[Chương 3: KẾT QUẢ NGHIÊN CỨU 11](#_Toc92203854)

[I. CÀI ĐẶT THUẬT TOÁN 11](#_Toc92203855)

[1. Thuật Toán Sắp Xếp Nổi Bọt (Bubble Sort) 11](#_Toc92203856)

[2. Thuật Toán Sắp Xếp Trộn (Merge Sort) 14](#_Toc92203857)

[3. Thuật Toán Sắp Xếp Chèn (Insertion Sort) 18](#_Toc92203858)

[4. Thuật Toán Sắp Xếp Nhanh (Quick Sort) 20](#_Toc92203859)

[II. ĐÁNH GIÁ ĐỘ PHỨC TẠP, THỜI GIAN THỰC HIỆN CỦA CÁC THUẬT TOÁN 22](#_Toc92203860)

[1. Thuật toán sắp xếp Chèn (Insertion Sort) 22](#_Toc92203861)

[2. Thuật toán sắp xếp Trộn (Merge Sort) 25](#_Toc92203865)

[3. Thuật toán sắp xếp Nổi Bọt (Bubble Sort) 27](#_Toc92203869)

[4. Thuật toán sắp xếp Nhanh (Quick Sort) 29](#_Toc92203873)

[5. Thời gian thực hiện các thuật toán 31](#_Toc92203877)

[III. CÁC VÍ DỤ, ỨNG DỤNG THỰC TẾ CỦA CÁC THUẬT TOÁN 33](#_Toc92203878)

[1. Thuật Toán Sắp Xếp Nổi Bọt (Bubble Sort) 33](#_Toc92203879)

[2. Thuật Toán Sắp Xếp Trộn (Merge Sort) 34](#_Toc92203880)

[3. Thuật Toán Sắp Xếp Chèn (Insertion Sort) 34](#_Toc92203881)

[4. Thuật Toán Sắp Xếp Nhanh (Quick Sort) 35](#_Toc92203882)

[Chương 4: KẾT LUẬN 37](#_Toc92203883)

[TÀI LIỆU THAM KHẢO 38](#_Toc92203884)

DANH MỤC HÌNH

[Hình 2. 1: Mô tả thuật toán Bubble Sort 6](#_Toc92203909)

[Hình 2. 2 Mô tả thuật toán Merge Sort 7](#_Toc92203910)

[Hình 2. 3 Mô tả thuật toán Insertion Sort 9](#_Toc92203911)

[Hình 2.4 Mô tả thuật toán Quick Sort 10](#_Toc92203912)

[Hình 3. 1.1: Thuật Toán Sắp Xếp Nổi Bọt 11](#_Toc92203913)

[Hình 3. 1.2: Thuật Toán Sắp Xếp Trộn 14](#_Toc92203914)

[Hình 3. 1.3:Thuật Toán Sắp Xếp Chèn 18](#_Toc92203915)

[Hình 3. 1.4: Thuật Toán Sắp Xếp Nhanh 20](#_Toc92203916)

[Hình 3.2.1 Thuật toán sắp xếp chèn 23](#_Toc92203917)

[Hình 3.2.2 Thuật toán sắp xếp trộn 26](#_Toc92203919)

[Hình 3.2.3 Thuật toán sắp xếp nổi bọt 28](#_Toc92203921)

[Hình 3.2.4 Thuật toán sắp xếp nhanh 30](#_Toc92203923)

[Hình 3.5 Đồ thị so sánh thời gian 32](#_Toc92203927)

[Hình 4. Xếp bài 33](file:///E:\TRUONGTHIDIEMQUYNH_61136433_DE04\BAOCAO.docx#_Toc92203928)

[Hình 4.4 a Các mệnh giá chưa được sắp xếp 35](file:///E:\TRUONGTHIDIEMQUYNH_61136433_DE04\BAOCAO.docx#_Toc92203929)

[Hình 4.4 b Chia tiền theo mệnh giá <10000 và >20000 36](file:///E:\TRUONGTHIDIEMQUYNH_61136433_DE04\BAOCAO.docx#_Toc92203930)

[Hình 4.4 c Sắp xếp mệnh giá theo nhóm 36](file:///E:\TRUONGTHIDIEMQUYNH_61136433_DE04\BAOCAO.docx#_Toc92203931)

[Hình 4.4 d Mệnh giá tiền đã được sắp xếp theo chiều tăng 36](file:///E:\TRUONGTHIDIEMQUYNH_61136433_DE04\BAOCAO.docx#_Toc92203932)

**DANH MỤC BẢNG**

[Bảng 3.2.1 Độ phức tạp của thuật toán chèn 23](#_Toc92203894)

[Bảng 3.2.2 Độ phức tạp của thuật toán trộn 26](#_Toc92203896)

[Bảng 3.2.4 Độ phức tạp của thuật toán nhanh 30](#_Toc92203900)

[Bảng 3.5.1 Bảng đo thực thiện thời gian của các giải thuật sắp xếp 32](#_Toc92203901)

[Bảng 3.5.2 Thời gian thực hiện các thuật toán sắp xếp 32](#_Toc92203902)

# Chương 1: TỔNG QUAN ĐỀ TÀI

1. **LÝ DO CHỌN ĐỀ TÀI**

Công nghệ ngày càng phát triển, thông tin ngày càng đa dạng việc quản lý dữ liệu, thao tác tìm kiếm thông tin cần phải thực hiện một cách nhanh chóng và hiệu quả (ví dụ như: tra cứu từ điển, tìm tin tức đáng chú ý hằng ngày, …) và muốn cho việc tìm kiếm nhanh chóng thì dữ liệu phải được sắp xếp ngăn nắp, đúng trật tự, hệ thống nhất định sẽ giúp việc tìm kiếm dễ dàng và hiệu quả hơn.

Do đó khi xây dựng một hệ thống quản lý thông tin trên máy cần có các thuật toán sắp xếp, thuật toán tìm kiếm. Hiện nay đã có rất nhiều thuật toán tìm kiếm cũng như thuật toán sắp xếp với mức độ hiệu quả của từng thuật toán phụ thuộc vào tính chất và cấu trúc của dữ liệu mà nó tác động. Trong khoa học máy tính và trong toán học, một số thuật toán sắp xếp là thuật toán sắp xếp các phần tử của một mảng hoặc một danh sách theo chiều tăng dần hoặc giảm dần. Và thường họ sắp xếp các phần tử ở đây là một dãy số, hầu hết các bài toán khác nhau có thể có nhiều cách giải quyết khác nhau. Để biết rõ hơn về hiệu quả của thuật toán sắp xếp ta đi vào các thuật toán sắp xếp điển hình như là: Sắp xếp nổi bọt (Bubble sort), sắp xếp chèn (Insertion sort), sắp xếp nhanh (Quicksort), sắp xếp trộn (Merge sort). Đề tài thực hiện là “GIẢI THUẬT SẮP XẾP” của các thuật toán trên.

1. **MÔI TRƯỜNG CÀI ĐẶT**

Sử dụng ngôn ngữ lập trình C để cài đặt các thuật toán sắp xếp nêu trên (Bubble sort, Insertion sort, Merge sort, Quick sort).

1. **PHƯƠNG PHÁP NGHIÊN CỨU**

Phân tích thuật toán, cách giải thuật của các thuật toán.

Tìm hiểu giải thuật, so sánh kết quả giữa các giải thuật.

Sử dụng ngôn ngữ C để cài đặt các thuật toán sắp xếp.

# Chương 2: CƠ SỞ LÝ THUYẾT

1. **ĐỊNH NGHĨA SẮP XẾP**

Sắp xếp là một quá trình biến đổi một danh sách các đối tượng thành một danh sách thỏa mãn một thứ tự xác định nào đó. Sắp xếp đóng vai trò quan trọng trong tìm kiếm dữ liệu. Chẳng hạn, trong thực tế là các ứng dụng quản lý danh bạ điện thoại thì có sắp xếp theo số, theo tên. Quản lý học sinh thì có sắp xếp theo điểm, theo lớp, theo trường, ...nếu danh sách đã được sắp xếp theo thứ tự tăng dần (hoặc giảm dần), ta có thể sử dụng kỹ thuật tìm kiếm nhị phân hiệu quả hơn nhiều so với việc tìm kiếm tuần tự… Trong thiết kế thuật toán, ta cũng thường xuyên cần đến việc sắp xếp, nhiều thuật toán được thiết kế dựa trên ý tưởng xử lý các đối tượng theo một thứ tự xác định.

Các thuật toán sắp xếp được chia làm 2 loại: sắp xếp trong và sắp xếp ngoài. Sắp xếp trong được thực hiện khi mà các đối tượng cần sắp xếp được lưu ở bộ nhớ trong của máy tính dưới dạng mảng. Do đó sắp xếp trong còn được gọi là sắp xếp mảng. Khi các đối tượng cần sắp xếp quá lớn cần lưu ở bộ nhớ ngoài dưới dạng file, ta cần sử dụng các phương pháp sắp xếp ngoài, hay còn gọi là sắp xếp file.

1. **THUẬT TOÁN**
2. **THUẬT TOÁN SẮP XẾP NỔI BỌT (BUBBLE SORT)**
   1. **Khái niệm**

Sắp xếp nổi bọt là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

* 1. **Phân tích bài toán**

Duyệt từ cuối mảng về đầu mảng, trong quá trình duyệt nếu phần tử ở dưới (đứng phía sau) nhỏ hơn phần tử đứng ngay trên (trước) nó thì theo nguyên tắc của bọt khí phần tử nhẹ sẽ bị “trồi” lên phía trên phần tử nặng (hai phần tử này sẽ được đổi chỗ cho nhau). Kết quả là phần tử nhỏ nhất (nhẹ nhất) sẽ được đưa lên (trồi lên) trên bề mặt (đầu mảng) rất nhanh.

Sau mỗi lần đi chúng ta đưa được một phần tử trồi lên đúng chỗ. Do vậy, sau n–1 lần đi thì tất cả các phần tử trong mảng M sẽ có thứ tự tăng.

*Xét bài toán*: Cho dãy các số thực ,,,…, , yêu cầu sắp xếp lại theo trật tự tăng dần của kiểu nổi bọt Ta tiến hành duyệt dãy a [1] …a[n]. Nếu a[i] > a[i+1] (i=1, 2, …, n-1) thì ta đổi chỗ phần tử a[i] với a[i+1]. Lặp lại quá trình duyệt dãy này cho đến khi không có xảy ra việc đổi chỗ của hai phần tử.

* 1. **Bài toán ví dụ**

|  |
| --- |
| Bubble Sort | Codeskulls  Hình 2. 1: Mô tả thuật toán Bubble Sort |

1. **THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT)**
   1. **Khái niệm**

Sắp xếp trộn (Merge Sort) là một giải thuật sắp xếp dựa trên giải thuật Chia để trị (Divide and Conquer). 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. Hàm merge () được sử dụng để gộp hai nửa mảng. Hàm merge (arr, l, m, r) là tiến trình quan

trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

* 1. **Phân tích bài toán**

Cần sắp xếp mảng A[1…n]. Thuật toán sắp xếp trộn được phát triển dựa trên chia để trị, bao gồm các bước sau:

Chia dãy gồm n phần tử cần sắp xếp ra thành hai dãy, mỗi dãy có n/2 phần tử.

Sắp xếp mỗi dãy con một cách đệ quy sử dụng sắp xếp trộn. Khi dãy chỉ còn một phần tử thì trả lại phần tử này.

Trộn (merge) hai dãy con được sắp xếp để thu được dãy sắp xếp gồm tất cả các phần tử của hai dãy con.

*Xét bài toán:* Sắp xếp sắp xếp mảng A [1…n] theo thứ tự tăng dần. Đầu tiên chia dữ liệu thành 2 phần (tức là mỗi dãy gồm n/2), sắp xếp từng phần. Sau đó trộn 2 phần lại với nhau. Để trộn 2 phần, ta làm như sau: Tạo một dãy A rỗng để chứa các phần tử đã sắp xếp; So sánh 2 phần tử đầu tiên của 2 phần. Phần tử nhỏ hơn ta cho vào A và xóa khỏi phần tương ứng; Tiếp tục như vậy đến khi ta cho hết các phần tử vào dãy A.

* 1. **Bài toán ví dụ**

|  |
| --- |
| Merge Sort in C – Algorithm and Program With Explanation  Hình 2. 2 Mô tả thuật toán Merge Sort |

1. **THUẬT TOÁN SẮP XẾP CHÈN (INSERTION SORT)**
   1. **Khái niệm**

Sắp xếp chèn là một giải thuật sắp xếp dựa trên so sánh tại chỗ. Ở đây, một danh sách con luôn luôn được duy trì dưới dạng đã qua sắp xếp. Sắp xếp chèn là chèn thêm một phần tử vào danh sách con đã qua sắp xếp. Phần tử được chèn vào vị trí thích hợp sao cho vẫn đảm bảo rằng danh sách con đó vẫn sắp theo thứ tự

* 1. **Phân tích bài toán**

Thuật toán sắp xếp chèn làm việc cũng giống như tên gọi - nó thực hiện việc quét một tập dữ liệu, với mỗi phân tử, thủ tục kiểm tra và chèn phần tử đó vào vị trí thích hợp trong danh sách đích (chứa các phần tử đứng trước nó đã được sắp xếp) được tiến hành.

Thuật toán được tiến hành bằng cách dịch chuyển phân tử hiện tại đang xét tuần tự qua những phân tử ở vùng dữ liệu phía trước đã được sắp xếp, phép hoán vị nó với phần tử liền kề được thực hiện một cách lặp lại cho tới khi tiến tới được vị trí thích hợp.

Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con (dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

Tức là ban đầu coi dãy khoá chỉ có dãy khoá K1 đã được sắp xếp, khi xét thêm dãy khóa K2, ta phải so sánh K2 với K1 để tìm chỗ thích hợp chèn K2 vào. Dãy K1 và K2 đã được sắp xếp sau đó xét thêm K3 ta phải so sánh K1 và K2 để tìm chỗ chèn K3 vào cho đến Kn được chèn vào đúng chỗ. Khi đó thuật toán kết thúc.

*Xét bài toán:* Sắp xếp một danh sách chứa các phần tử ,,,…, theo thứ tự tăng dần. Các phần tử đưa vào danh sách được đặt đúng vị trí theo trật tự giá trị tăng dần. Chú ý là các ,,,…, được đưa vào cho đến thời điểm thứ i Ta tiến hành khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử(phần tử đầu tiên, phần tử có chỉ số 0); Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử; Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

* 1. **Bài toán ví dụ**

|  |
| --- |
| Insertion Sort - GeeksforGeeks  Hình 2. 3 Mô tả thuật toán Insertion Sort |

1. **THUẬT TOÁN SẮP XẾP NHANH (QUICK SORT)**
   1. **Khái niệm**

Thuật toán Quick Sort là một [thuật toán sắp xếp](https://topdev.vn/blog/cac-thuat-toan-sap-xep/), còn được gọi là sắp xếp kiểu phân chia (Part Sort). Được phát triển dựa trên kỹ thuật chia để trị (kỹ thuật chia để trị: Khi cần giải quyết một bài toán, ta sẽ tiến hành chia bài toán đó thành các bài toán con nhỏ hơn. Tiếp tục chia cho đến khi các bài toán nhỏ này không thể chia thêm nữa, khi đó ta sẽ giải quyết các bài toán nhỏ nhất này và cuối cùng kết hợp giải pháp của tất cả các bài toán nhỏ để tìm ra giải pháp của bài toán ban đầu). Sắp xếp này là phương pháp cải tiến của phương pháp đổi chỗ (Bubble sort).

* 1. **Phân tích bài toán**

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.

*Xét bài toán:* Sắp xếp dãy A[LR] theo chiều tăng dần của dãy. Tìm phần tử chủ chốt p bằng cách (L+R)/2, dãy được làm 2 đoạn A [L…p-1) và A(p+1…R) . Ta tiến hành sắp xếp sao cho:

* Các phần tử trong A(L…p-1)A(p)
* Các phần tử trong A(p+1…R) A(p)
* Sau khi so sánh xong ta tiến hành swap cho 2 vị trí ta vừa so sánh với phần tử chốt. Thực hiện xong các bước, ta thu được 1 dãy đã sắp xếp theo chiều tăng dần.
  1. **Bài toán ví dụ**

|  |
| --- |
| chia se nhung dieu hay va thu thuat: Code C/C++: Thuật toán sắp xếp nhanh ( QuickSort)  Hình 2.4 Mô tả thuật toán Quick Sort |

# Chương 3: KẾT QUẢ NGHIÊN CỨU

1. **CÀI ĐẶT THUẬT TOÁN**
   1. **Thuật Toán Sắp Xếp Nổi Bọt (Bubble Sort)**

|  |
| --- |
| Hình 3. 1.1: Thuật Toán Sắp Xếp Nổi Bọt |

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define MAX 100000  int a[MAX],n;  void Swap(int &a, int &b)  {  int t=a; a=b; b=t;  }  void Doc(){    FILE \*f = fopen("FILE.dat", "r");  fscanf(f, "%d", &n);  for (int i = 0; i < n; i++)  {  fscanf(f, "%d", &a[i]);  }  printf("\nChieu dai mang la: %d",n);  printf("\nDanh sach cac phan tu mang:\t");  for (int i=0; i<n; ++i)  {  printf("%d ",a[i]);  }  printf("\n");  fclose(f);  }  void Ghi()  {  FILE \*fp;  fp=fopen("KQ.dat","w");  for(int i=1; i<n;i++)  {  fprintf(fp,"%d\n",a[i]);  }  fclose(fp);  }  void BubbleSort(int a[], int n)  {  for (int i=0;i<n-1;i++)  for (int j=n-1;j>i;j--)  if (a[j]<a[j-1])  Swap(a[j],a[j-1]);  }  int main()  {  Doc();  BubbleSort(a,n);  Ghi();  } |

* 1. **Thuật Toán Sắp Xếp Trộn (Merge Sort)**

|  |
| --- |
| Hình 3. 1.2: Thuật Toán Sắp Xếp Trộn |

#include <stdio.h>

#include <stdlib.h>

#define MAX 100000

int a[MAX],n;

void Doc()

{

FILE \*f = fopen("FILE.dat", "r");

fscanf(f, "%d", &n);

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

{

fscanf(f, "%d", &a[i]);

}

printf("\nChieu dai mang la: %d",n);

printf("\nDanh sach cac phan tu mang:\t");

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

{

printf("%d ",a[i]);

}

printf("\n");

fclose(f);

}

void Merge(int a[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

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

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

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

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

i=0;

j=0;

k=l;

while (i < n1 && j < n2)

{

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

{

a[k] = L[i];

i++;

}

else

{

a[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

a[k] = L[i];

i++;

k++;

}

while (j < n2)

{

a[k] = R[j];

j++;

k++;

}

}

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

{

if (l<r)

{

int t=(l+r)/2;

MergeSort(a,l,t);

MergeSort(a,t+1,r);

Merge(a,l,t,r);

}

}

void Ghi()

{

FILE \*fp;

fp=fopen("KQ.dat","w");

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

{

fprintf(fp,"%d\n",a[i]);

}

fclose(fp);

}

int main()

{

Doc();

MergeSort(a,0,n-1);

Ghi();

}

* 1. **Thuật Toán Sắp Xếp Chèn (Insertion Sort)**

|  |
| --- |
| Hình 3. 1.3:Thuật Toán Sắp Xếp Chèn |

#include <stdio.h>

#include <stdlib.h>

#define MAX 100000

int a[MAX],n;

void Doc(){

FILE \*f = fopen("FILE.dat", "r");

fscanf(f, "%d", &n);

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

{

fscanf(f, "%d", &a[i]);

}

printf("\nChieu dai mang la: %d",n);

printf("\nDanh sach cac phan tu mang:\t");

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

{

printf("%d ",a[i]);

}

printf("\n");

fclose(f);

}

void Ghi()

{

FILE \*fp;

fp=fopen("KQ.dat","w");

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

{

fprintf(fp,"%d\n",a[i]);

}

fclose(fp);

}

void InsertSort(int a[], int n)

{

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

{

int tam=a[i],j=i-1;

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

if (tam<a[j])

a[j+1]=a[j];

else break;

a[j+1]=tam;

}

}

int main()

{

Doc();

InsertSort(a,n);

Ghi();

}

* 1. **Thuật Toán Sắp Xếp Nhanh (Quick Sort)**

|  |
| --- |
| Hình 3. 1.4: Thuật Toán Sắp Xếp Nhanh |

#include <stdio.h>

#include <stdlib.h>

#define MAX 100000

int a[MAX],n;

void Doc(){

FILE \*f = fopen("FILE.dat", "r");

fscanf(f, "%d", &n);

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

{

fscanf(f, "%d", &a[i]);

}

printf("\nChieu dai mang la: %d",n);

printf("\nDanh sach cac phan tu mang:\t");

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

{

printf("%d ",a[i]);

}

printf("\n");

fclose(f);

}

void Swap(int &a, int &b)

{

int t=a; a=b; b=t;

}

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

{ int i=l; int j=r; int mid= a[(l+r)/2];

do

{

while(a[i]<mid)

i++;

while (mid<a[j])

j--;

if(i<=j)

{

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

i++; j--;

}

} while (i<=j);

if(i<r)

QuickSort(a,i,r);

if(l<j)

QuickSort(a,l,j);

}

void Ghi ()

{

FILE \*fp;

fp=fopen("KQ.dat","w");

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

{

fprintf(fp,"%d\n", a[i]);

}

fclose(fp);

}

int main ()

{

Doc ();

QuickSort(a,0, n-1);

Ghi ();

}

1. **ĐÁNH GIÁ ĐỘ PHỨC TẠP, THỜI GIAN THỰC HIỆN CỦA CÁC THUẬT TOÁN**
   1. **Thuật toán sắp xếp Chèn (Insertion Sort)**
      1. **Ý tưởng thuật toán**

Sắp xếp chèn là thuật toán một thuật toán sắp xếp bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự người chơi rút lần lượt từ quân bài thứ hai so với quân bài bài đứng trước nó để chèn vào vị trí thích hợp.

* + 1. **Giải thuật**
* **Bước 1:** i=1
* **Bước 2:** lần lượt so sánh và đối chỗ (nếu có) từ phần tử kế bên phần tử đang xét đến hết mảng
* **Bước 3:** i=i+1
* **Bước 4**: Nếu i<n thì quay lại bước 2 ngược lại i>n thì kết thúc
  + 1. **Độ phức tạp**

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

{int tam=a[i], j=i-1;

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

if (tam<a[j])

a[j+1] = a[j];

else break;

a[j+1] = tam;}

Hình 3.2.1 Thuật toán sắp xếp chèn

Phân tích độ phức tạp của thuật toán:

|  |  |  |
| --- | --- | --- |
| Trường hợp | Số lần so sánh | Số lần hoán vị |
| Tốt nhất | = n-1 | = 2(n-1) |
| Xấu nhất | = | = |

Bảng 3.2.1 Độ phức tạp của thuật toán chèn

* Số lần đổi chỗ:
* Tmin= 0, n-1 so sánh (khi dãy đầu vào đã được sắp)
* Tave=/4, hoán đổi và so sánh.
* Tmax= /2, hoán đổi và so sánh (khi dãy đầu vào có thứ tự ngược lại với thứ tự cần sắp xếp).
* Thuật toán này có thời gian tính trong tình huống tốt nhất là tốt nhất. Là thuật toán sắp xếp tốt đối với dãy đã gần được sắp xếp (tức là mỗi phần tử đã đứng gần vị trí rất gần vị trí trong thứ tự cần sắp xếp).
* Trường hợp tốt nhất: khi mảng a ban đầu đã có thứ tự tăng
* Số phép gán: Tmin = 2×(n-1)
* Số phép so sánh: Smin = 1+2+…+(n-1) = n×(n-1)/2
* Số phép hoán vị: Tmin = 0
* Trường hợp xấu nhất, khi mảng M ban đầu luôn có phần tử nhỏ nhất trong n-k phần tử còn lại đứng ở vị trí sau cùng sau mỗi lần hoán vị:
* Số phép gán: Tmax = [2×(n-1)] + [ 1+2+…+(n-1)] = [2×(n-1)] + [n×(n-1)/2]
* Số phép so sánh: Tmax = (n-1)
* Số phép hoán vị: Tmax = 0
* Trung bình: Số phép gán: Tavg = 2×(n-1) + [n×(n-1)/4]
* Số phép so sánh: Tavg = [n×(n-1)/2 + (n-1)]/2 = (n+2) × (n-1)/4
* Số phép hoán vị: Tavg = 0
* Các phép so sánh xảy ra trong mỗi vòng lặp tìm vị trí thích hợp pos mỗi lần xác định vị trí pos đang xét không thích hợp thì dời phần tử M[pos-1] đến vị trí pos:
* Giải thuật thực hiện tất cả n-1 vòng lặp tìm pos do số lượng phép so sánh và dời chổ này phụ thuộc vào tình trạng của dãy số ban đầu nên chỉ có thể ước lượng từng trường hợp cụ thể như sau: Do với mỗi phân tử, insertion sort cần thực hiện so sánh với các phần tử trước nó nên tối đa số lần duyệt dữ liệu là !n . Vì vậy cũng giống như Bubble sort Insertion sort được coi là có độ phức tạp O(n2).
* Mặc dù có cùng độ phức tạp, Insertion sort hiệu quả hơn gần như hai lần so với Bubble sort, tuy nhiên vẫn không hiệu quả với tập dữ liệu lớn.

Xấu nhất: T(n)= O ()

Tốt nhất: O(n).

* Ưu điểm: chạy nhanh khi mảng nhỏ hay được sắp xếp một phần.
* Nhược điểm: Hiệu suất thấp.
  1. **Thuật toán sắp xếp Trộn (Merge Sort)**
     1. **Ý tưởng thuật toán**

Giống như Quick sort, 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. Hàm merge () được sử dụng để gộp hai nửa mảng. Hàm merge (arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

* + 1. **Giải thuật**

Các bước thực hiện thuật toán trộn tự nhiên như sau:

**Bước 1:** // Chuẩn bị

r = 0; // r dùng để đếm số đường chạy

**Bước 2:** Tách dãy a1, a2, ..., an thành 2 dãy b, c theo nguyên tắc luân phiên từng đường chạy:

**Bước 2.1:** Phân phối cho b một đường chạy; r = r+1;

Nếu a còn phần tử chưa phân phối

Phân phối cho c một đường chạy; r = r+1;

**Bước 2.2:** Nếu a còn phần tử: quay lại bước 2.1;

**Bước 3**: Trộn từng cặp đường chạy của 2 dãy b, c vào a.

**Bước 4:** Nếu r >= 2 thì trở lại bước 1;

Ngược lại: Dừng.

* + 1. **Độ phức tạp**

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

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

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

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

i=0; j=0; k=l;

while (i<n1 && j<n2)

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

{ a[k] = L[i];

i++; }

else

{ a[k] = R[j];

j++; }

k++; }

while (i < n1)

{ a[k] = L[i];

i++;

k++; }

while (j<n2)

{ a[k] = R[j];

j++;

k++;}

Hình 3.2.2 Thuật toán sắp xếp trộn

Phân tích độ phức tạp của thuật toán:

|  |  |
| --- | --- |
| Trường hợp | Số lần so sánh |
| Tốt nhất | nn |
| Xấu nhất | nn |

Bảng 3.2.2 Độ phức tạp của thuật toán trộn

* Thời gian tính của trộn:
* Khởi tạo hai mảng con L, R là: O(n1+n2) =O(n)
* Đưa các phần tử vào mảng kết quả (vòng lặp): có số lần lặp và mỗi lần lặp đòi hỏi thời gian hằng số. Do đó thời gian cần thiết để thực hiện là O(n).
* Tổng cộng thời gian trộn: O(n).
* Thời gian tính của sắp xếp trộn:
* Chia: Tính t như giá trị trung bình của l, r: T(n)= O(1).
* Trị: Giải đệ quy 2 bài toán con, mỗi bài có kích thước n/2 T(n)=2O(n/2).
* Tổ hợp: Trộn (Merge) trên các mảng con có n phần tử đòi hỏi thời gian O(n) T(n)=O(n)
* Do đó ta có công thức đệ quy:

T(n)= Suy ra: T(n) = O(nn)

* Ưu điểm: Hiệu suất của merge sort rất cao.
* Nhược điểm: Code thuật toán này khá phức tạp.
  1. **Thuật toán sắp xếp Nổi Bọt (Bubble Sort)**
     1. **Ý tưởng thuật toán:**

Sắp xếp chèn là một thuật toán sắp xếp bắt chước cách sắp xếp quân bài của những người chơi. Muốn sắp một bộ bài theo đúng trật tự người ta chơi bài rút lần lượt từ trái sang phải so với con quân trước nó nhỏ hơn thì chèn ở trước ngược lại lớp hơn thì chèn ở sau.

* + 1. **Giải thuật:**

**Bước 1:** i=1

**Bước 2:** lần lượt so sánh và đổi chỗ (nếu cần) từ phần tử kế bên phần tử đang xét đến hết mảng

**Bước 3:** i=i+1

**Bước 4:** Nếu i<n thi quay ngược lại bước 2, ngược lại i>n thì dừng.

* + 1. **Độ phức tạp**

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

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

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

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

Hình 3.2.3 Thuật toán sắp xếp nổi bọt

Phân tích độ phức tạp của thuật toán:

|  |  |  |
| --- | --- | --- |
| Trường hợp | Số lần so sánh | Số lần hoán vị |
| Tốt nhất | = | 0 |
| Xấu nhất |  | = |

Bảng 3.2.3 Độ phức tạp của sắp xếp nổi bọt

Số lần so sánh: các phép so sánh n-1 sẽ được thực hiện ở lần 1. Tương tự, n-2, n-3 sẽ lần lượt là các lần thứ 2,3, …Vì vậy, tổng số phép so sánh là:

(n-1) +(n-2) +(n-3) +…+3+2+1. T(n)= n\*(n-1)/2.

* Số lần đổi chỗ:
* Tmin: 0 đổi chỗ.
* Tave: n^2 /4.
* Tmax: n^2/2 đổi chỗ.

T(n)= O (): độ phức tạp xấu nhất; tối ưu nhất O(n).

* Trong mọi trường hợp: Số phép gán: G = 0
* Số phép so sánh: S = (n-1) + (n-2) + … + 1 = n(n-1)/2.
* Trong trường hợp tốt nhất: Khi mảng ban đầu đã có thứ tự tăng:
* Số phép hoán vị: Tmin = 0
* Trong trường hợp xấu nhất: khi mảng ban đầu đã có thứ tự giảm:
* Số phép hoán vị: Tmin = (n-1) + (n-2) + … + 1 = n(n-1)/2.
* Số phép hoán vị trung bình: Tavg = n(n-1)/4.

T(n)= O(): độ phức tạp xấu nhất; tốt nhất O(n).

* Ưu điểm: Thuật toán dễ cài đặt, code ngắn gọn.
* Nhược điểm: Độ phức tạp lớn, hiệu suất thấp nhất.
  1. **Thuật toán sắp xếp Nhanh (Quick Sort)**
     1. **Ý tưởng thuật toán**

Tìm cách chia đôi dãy ban đầu ban đầu bằng cách chọn ra một phần tử là chốt (pivot). Từ dãy ban đầu, tất cả phần tử nhỏ phần tử nhỏ hơn phần tử chốt thì đưa về bên trái dãy, tất cả các phần tử lớn hơn hoặc bằng phần tử chốt thì đưa về bên phải. Sau bước này ta có phần tử chốt là đứng đúng vị trí. Dãy ban đầu phân chia làm hai dãy con nằm hai bên chốt. Tiếp tục phân chia các dãy con theo cách tương tự đến khi mọi dãy con đều có một độ dài bằng 1. Có thể lựa chọn phần tử chốt nằm đầu, cuối hay giữa. ở đây sẽ lưa ra phần tử chốt nằm giữa dãy nhất.

* + 1. **Giải thuật**

**Bước 1:** pivot = a[(l+r)/2]

**Bước 2:** i=l,j=r

**Bước 3:** Nếu a[i] <pivot thì i = i+1**,** ngược lại nếu a[j]>pivot thì j=j-1

**Bước 4:** nếu i>=j thì đổi chỗ a[i] với a[j] quay về bước 3

**Bước 5:** Lặp lại từ bước 1 đến bước 4 với đoạn a[l] đến a[j] và a[r] cho đến khi tất cả đoạn có độ dài là 1.

* + 1. **Độ phức tạp**

while (i<=j)

{

while (a[i]<a[mid])

i++;

while (a[mid]<a[j])

j--;

if (i<=j)

{

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

i++;j--;

}

}

Hình 3.2.4 Thuật toán sắp xếp nhanh

Phân tích độ phức tạp của thuật toán:

|  |  |
| --- | --- |
| Trường hợp | Số lần hoán vị |
| Tốt nhất | nn |
| Xấu nhất |  |

Bảng 3.2.4 Độ phức tạp của thuật toán nhanh

* Về nguyên tắc, có thể chọn giá trị mốc x là một phần tử tùy ý trong dãy, nhưng để đơn giản, phần tử có vị trí giữa thường được chọn, khi đó p=(l+r)/2.
* Giá trị mốc x được chọn sẽ có tác động đến hiệu quả thực hiện thực toán vì nó quyết định số lần phân hoạch.
* Số lần phân hoạch là ít nhất nếu ta chọn x là phần tử trung vị(median), là nhiều nhất nếu x là cục trị của dãy.
* Tuy nhiên do chi phí chọn phần tử median quá cao nên trong thực tế người ta không chọn phần tử này mà chọn phần tử nằm chính giữa dãy hy vọng nó có thể gần với giá trị median.
* Hiệu quả phụ thuộc vào việc chọn giá trị mốc.
* Trường hợp tốt nhất: Mỗi lần phân hoạch đều chọn phần tử median làm mốc, khi đó dãy được phân chia thành hai phần bằng nhau và cần (n) lần phân hoạ ch thì sắp xếp xong.
* Nếu mỗi lần phân hoạch chọn giá trị cực đại hay cực tiểu là mốc thì dãy sẽ bị phân chia thành 2 phần không đều: một phần chỉ có 1 phần tử, phần còn lại gồm (n-1) phần tử, do vậy cần n lần mới sắp xếp xong.
* Ưu điểm: Tuỳ cách chọn phần tử chốt (pivot) mà tốc độ của thuật toán nhanh hay chậm.
* Nhược điểm: Code khá phức tạp.
  1. **Thời gian thực hiện các thuật toán**

Ta có: n là bộ mẫu thử các số lần lượt là 10000, 50000, 100000, đoạn mẫu thử thuộc đoạn [1;100000]. Cùng thực hiện thời gian tính toán trong cùng một điều kiện: cùng một máy HP, trên hệ điều hành (Windows 10), không gian lưu trữ như nhau. Thực hiện thời gian chạy thu được bảng như sau:

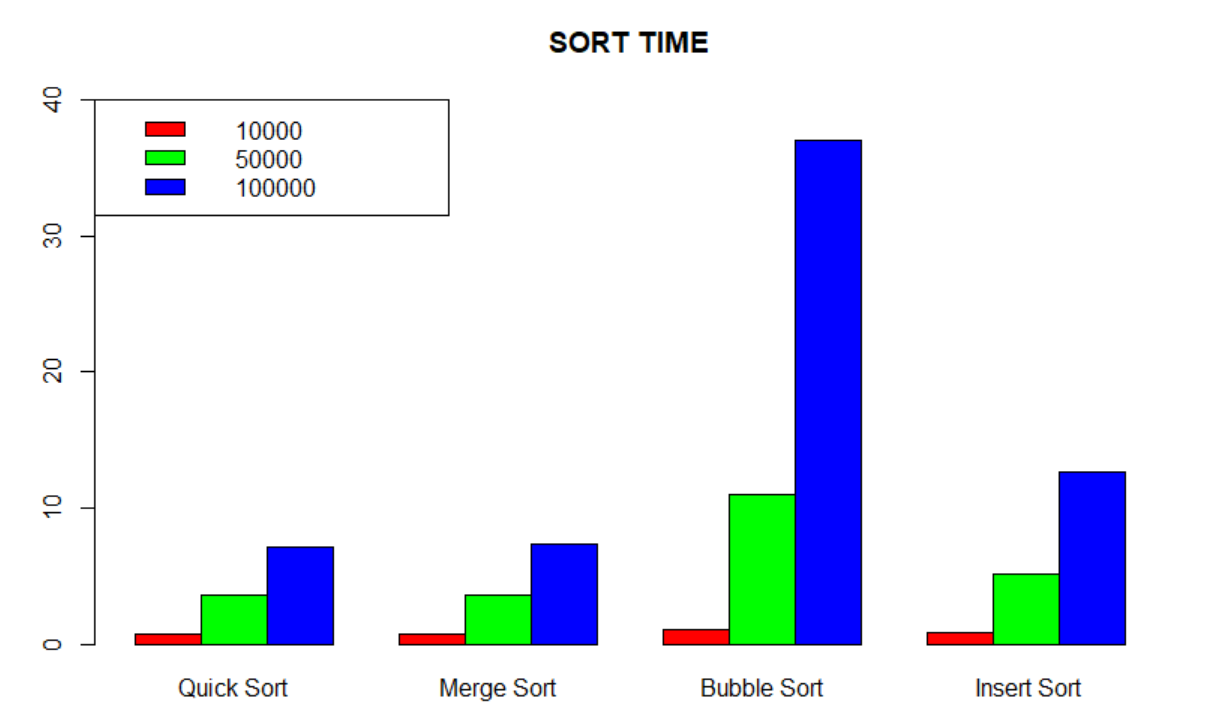
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Đoạn mẫu thử | Bộ thử (n) | Thuật toán | Lần 1 | Lần 2 | Lần 3 | Lần 4 | Lần 5 |
| [1;1000000] | 10000 | Quick sort | 0.673 | 0.689 | 0.696 | 0.73 | 0.71 |
| Merge sort | 0.712 | 0.702 | 0.701 | 0.694 | 0.704 |
| Bubble sort | 1.007 | 1.021 | 1.053 | 1.02 | 1.055 |
| Insertion sort | 0.814 | 0.781 | 0.756 | 0.797 | 0.769 |
| 50000 | Quick sort | 3.535 | 3.563 | 3.636 | 3.643 | 3.627 |
| Merge sort | 3.631 | 3.516 | 3.624 | 3.537 | 3.603 |
| Bubble sort | 10.89 | 11.28 | 10.981 | 11.144 | 10.863 |
| Insertion sort | 4.802 | 5.003 | 5.739 | 5.066 | 4.984 |
| 100000 | Quick sort | 7.065 | 6.995 | 7.201 | 7.235 | 7.184 |
| Merge sort | 7.452 | 7.748 | 7.177 | 7.214 | 7.305 |
| Bubble sort | 37.445 | 36.925 | 38.51 | 37.789 | 36.306 |
| Insertion sort | 12.706 | 12.563 | 12.503 | 12.768 | 12.597 |

Bảng 3.5.1 Bảng đo thực thiện thời gian của các giải thuật sắp xếp

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Đoạn mẫu thử | Bộ thử (n) | Thời gian thực hiện (s) | | | |
| Quick sort | Merge sort | Bubble sort | Insert sort |
| [1;100000] | 10000 | 0.6996 | 0.7026 | 1.0312 | 0.7834 |
| 50000 | 3.6008 | 3.5822 | 11.0316 | 5.1188 |
| 100000 | 7.136 | 7.3792 | 36.995 | 12.6274 |

Bảng 3.5.2 Thời gian thực hiện các thuật toán sắp xếp

Từ bảng thời gian trên, vẽ đồ thị (Đồ thị được vẽ trên ngôn ngữ Rstudio)

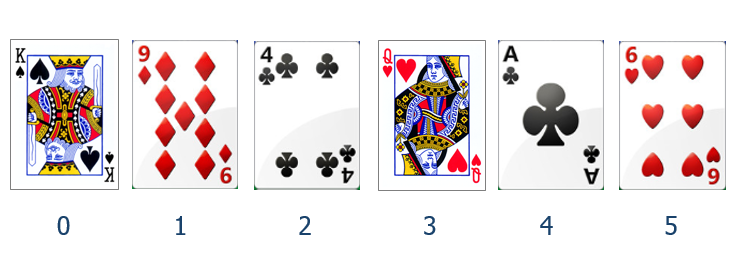


Hình 3.5 Đồ thị so sánh thời gian

Nhìn vào đồ thị có thể nhận thấy thời gian thực hiện được sắp xếp từ nhanh nhất đến chậm nhất lần lượt là: Quick sort, Merge sort, Insertion sort, Bubble sort. Thời gian trên thực hiện gần đúng với thời gian thực hiện trong thực tế. Trong thực tế ta thấy Insertion sort, Bubble sort đều có độ phức tạp n2) do vậy cả hai không thích hợp sử dụng để sắp xếp mảng lớn.

*Bảng so sánh độ phức tạp*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Thời gian | | | Không gian |
|  | Tốt nhất | Xấu nhất | Trung bình |  |
| Quick Sort | O(nlogn) | n2) | O(nlogn) | O(nlogn) |
| Merge Sort | O(nlogn) | O(nlogn) | O(nlogn) | n) |
| Bubble Sort | n) | n2) | n2) | ) |
| Insertion Sort | n) | n2) | n2) | ) |

1. **CÁC VÍ DỤ, ỨNG DỤNG THỰC TẾ CỦA CÁC THUẬT TOÁN**

Hình 4. Xếp bài

* 1. **Thuật Toán Sắp Xếp Nổi Bọt (Bubble Sort)**

Cũng giống như cách sắp xếp dãy số thì ta sẽ lần lượt sắp xếp các lá bài theo chiều tăng dần của các quân bài theo trình tự 3, 4, 5, 6, 7, 8, 9, J, Q, K, A, 2. Để sắp xếp dãy a={K, 9, 4, Q, A, 6} thì đầu tiên ta so sánh 2 phần tử kế nhau (tức là dãy a có i là chỉ số các phần tử (i=, n=13- bài tiến lên) là a[0]= K và a[1]=9, a[2]= 4, a[3]=Q, a[4]=A, a[5]=6. Nếu a [0]>a [1] thì ta hoán đổi vị trí của chúng (hay nói cách khác là phần tử có giá trị nhỏ sẽ trồi lên trước phần tử có giá trị lớn và ngược lại các phần tử có giá trị lớn sẽ chìm về sau) có thứ tự 9, K cứ thế so sánh cho đến khi hết dãy thì thu được 9, K, 4, Q, A, 6 (thay đổi giá trị nhưng chỉ số không đổi). Ta có cặp quân bài liền kề tiếp là K, 4 có thứ tự sắp xếp 4, K, thu được dãy 9, 4, K, Q, A, 6 tiến hành so sánh cặp quân bài 9, 4 và sắp xếp lại 4,9 thu được dãy 4, 9, K, Q, A. Tiếp tục thực hiện đến các cặp phần tử liền kề nhau tiếp cho đến khi hết dãy. Kết thúc quá trình ta sẽ thu được 1 dãy sắp xếp tăng dần có thứ tự là 4, 9, Q, K, A.

* 1. **Thuật Toán Sắp Xếp Trộn (Merge Sort)**

Merge sort vẫn thực hiện sắp xếp các lá bài theo chiều tăng dần của các quân bài theo trình tự 3, 4, 5, 6, 7, 8, 9, J, Q, K, A, 2. Xác định phần tử làm mốc n-1/2, để chia dãy thành 2 phần. Sử dụng đệ quy để trộn. Phần 1 gồm các lá bài bên trái, phần 2 là các lá bài bên phải. 2 phần này ta tiếp tục phân tách thành các dãy con (có thể dãy con gồm 1 hoặc 2 phần tử). Ví dụ a= {K, 9, 4, Q, A, 6} n=6, tiến hành chia dãy n-1/2=2, phân tách phần tử thành 2 dãy. Dãy 1 (trái): K, 9. Dãy 2 (phải): 4, Q, A, 6. Tiếp tục phân tách dãy 1, 2 lần lượt thành các dãy con là các dãy con K, 9, 4, Q, A, 6 gồm 1 phần tử. Sau đó, tiến hành trộn, sắp xếp từ trái sang phải, trộn từng cặp (1): 9 - K, (2): 4 – Q, (3): 6 – A.Tiếp tục trộn các cặp (1),(2) và (3), ta thu được cặp (a): 4 – 9 – Q – K cặp (b): 6 – A . Trộn cặp (a), (b) ta thu được dãy sắp xếp tăng dần: 4, 6, 9, Q, K, A.

* 1. **Thuật Toán Sắp Xếp Chèn (Insertion Sort)**

Đối với sắp xếp Insertion sort thì ta cũng thực hiện sắp xếp các lá bài theo chiều tăng dần của các quân bài theo trình tự 3, 4, 5, 6, 7, 8, 9, J, Q, K, A, 2. Như ví dụ có dãy a= {K, 9, 4, Q, A, 6}. Sau khi được chia bài thì ta bốc quân bài đầu tiên có giá trị là a[0]= K – giữ quân bài là mốc (xem như đã sắp xếp, các lá bài còn lại chưa được sắp xếp), ta bốc đến quân bài a[1]= 9, tiến hành so sánh và ta tìm vị trí chèn, vì 9 < K nên a[1] sẽ được chèn trước quân bài thứ nhất khi đó dãy có thứ tự là 9, K dãy được sắp là 9, K, 4, Q, A, 6. Tiếp đến quân bài a[2]= 4 tiếp tục so sánh và tìm vị trí chèn để thu được dãy tăng dần vì 4<9<K nên ta chèn quân bài a[2] trước quân bài a[1] và a[0]. Tiếp tục như vậy đến khi hết quân bài. Khi đó, thu được dãy quân bài theo thứ tự tăng dần 4, 9, Q, K, A.

* 1. **Thuật Toán Sắp Xếp Nhanh (Quick Sort)**

Đối với Quick sort thì cũng giống như các giải thuật trên vẫn mục tiêu là thực hiện sắp xếp các lá bài theo chiều tăng dần của các quân bài theo trình tự 3, 4, 5, 6, 7, 8, 9, J, Q, K, A, 2. Ví dụ a={K, 9, 4, Q, A, 6}, n/2= 2, ta chọn được quân bài làm phần tử chốt có giá trị a[1]=9 tiến hành so sánh các quân bài bên trái quân bài a[1] phải lớn hơn hoặc bằng a[1] và ngược lại các quân bài bên phải a[1] phải nhỏ hơn hoặc bằng a[1] (1). Ta tiến hành so sánh các quân bài khác với quân bài a[1], ta có a[0] > a[1] và a[5] < a[1] ta sẽ hoán đổi vị trí khi đó dãy được sắp xếp 6, 9, 4, Q, A, K. Tiếp tục so sánh a[1]>=a[1] và a[3]<a[1] ta hoán đổi vị trí khi đó dãy được sắp xếp 6, 4, 9, Q, A, K. Vì không có quân bài nào thỏa mãn điều kiện (1) ta sẽ tách dãy. Khi đó, dãy 1: 6, 4, 9; dãy 2: Q, A, K… Tiếp tục tìm phần tử chốt của 2 dãy và tiến hành sắp xếp, lưu ý phải thỏa điều kiện: các phần tử bên trái phần tử chốt sẽ lớn hơn hoặc bằng phần tử chốt. Ngược lại, các phần tử bên phải phần tử chốt phải nhỏ hơn hoặc bằng phần tử chốt. Khi thực hiện xong các bước ta thu được kết quả là một dãy tăng dần: 4, 6, 9, Q, K, A.

***Ví dụ xếp tiền: Có một xấp tiền gồm nhiều tờ mệnh giá khác nhau đang xếp lộn xộn, cần sắp lại theo mệnh giá từ nhỏ đến lớn. (Quick sort)***



Hình 4.4 a Các mệnh giá chưa được sắp xếp



Hình 4.4 b Chia tiền theo mệnh giá <10000 và >20000



Hình 4.4 c Sắp xếp mệnh giá theo nhóm



Hình 4.4 d Mệnh giá tiền đã được sắp xếp theo chiều tăng

# Chương 4: KẾT LUẬN

Các giải thuật sắp xếp luôn được ứng dụng rất nhiều trong đời sống (ví dụ: quản lý sinh viên trong việc thống kê điểm, họ tên, … Các bài toán thống kê, sắp xếp dữ liệu trong máy tính để thuận lợi cho việc tìm kiếm xử lý để in bảng). Sắp xếp trộn: cơ sở dữ liệu sử dụng sắp xếp hợp nhất bên ngoài để sắp xếp các tập dữ liệu quá lớn để có thể tải hoàn toàn vào bộ nhớ nhằm giảm số lượng I/Os của đĩa. Sắp xếp nổi bọt: sử dụng trong lập trình vi điều khiển từ xa của TV để sắp xếp các kênh trên cơ sở thời gian xem lâu hơn. Sắp xếp nhanh: sắp xếp tỷ lệ thắng thua dựa trên điểm đô của trận đấu thể thao.

**Soue code:** <https://github.com/TTDiemQuynh/TTCS>

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | N. s. v. Đ. C. Thơ, 2015. [Online]. Available: http://luanvan.co/luan-van/tieu-luan-cac-thuat-toan-sap-xep-co-ban-66640/. |
| [2] | T. Triệu, "Lập trình không khó," [Online]. Available: https://nguyenvanhieu.vn/thuat-toan-sap-xep-quick-sort/. |
| [3] | T. Triệu, "Lập trình không khó," [Online]. Available: https://nguyenvanhieu.vn/thuat-toan-sap-xep-chen/. |
| [4] | T. Triệu, "Lập trình không khó," [Online]. Available: https://nguyenvanhieu.vn/thuat-toan-sap-xep-merge-sort/. |
| [5] | T. Triệu, "Lập trình không khó," [Online]. Available: https://nguyenvanhieu.vn/thuat-toan-sap-xep-noi-bot/. |