BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC ĐẠI NAM

BÀI TẬP LỚN

TÊN HỌC PHẦN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ TÀI: …15-CÀI ĐẶT GIẢI THUẬT SẮP XẾP 2…….

Giáo viên hướng dẫn: **Trần Quý Nam**

Sinh viên thực hiện: ***1. Vũ Anh Quân***

2. Lê Tiến Thành

3. Nguyễn Xuân Hải

Hà Nội, 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC ĐẠI NAM

BÀI TẬP LỚN

TÊN HỌC PHẦN: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỀ TÀI: ……15-CÀI ĐẶT GIẢI THUẬT SẮP XẾP 2……

STT	Mã Sinh Viên	Họ và Tên	Ngày Sinh	Điểm	
				Bằng Số	Bằng Chữ
1	1451020086	Vũ Anh Quân	01/07/2006		
2	1871020540	Lê Tiến Thành	05/05/2006		
3	1871020207	Nguyễn Xuân Hải	18/09/2006		

CÁN BỘ CHẤM THI 1 **CÁN BỘ CHẤM THI 2**

Trần Quý Nam

Hà Nội, 2024

LỜI NÓI ĐẦU

Môn học Cấu trúc dữ liệu và giải thuật là một môn quan trọng trong chương trình đào tạo về Ngành công nghệ thông tin. Môn học này cung cấp cho sinh viên những kiến thức về cài đặt cũng như cách hoạt động của các hàm, hiểu rõ về các thuật toán. Nắm vững kiến thức cơ bản về cấu trúc dữ liệu, giải thuật để triển khai các ứng dụng có hiệu quả. Sử dụng được cấu trúc dữ liệu trong việc tổ chức quản lý dữ liệu trong chương trình. Cài đặt được các giải thuật xử lý các cấu trúc dữ liệu và một số giải thuật quan trọng khác trong lập trình. Thể hiện khả năng làm việc độc lập và hợp tác trong các nhóm.

Để hoàn thiện bài tập lớn với đề tài: "Cài đăt giải thuật sắp xếp 2". Em xin bày tỏ lòng biết ơn đến giảng viên đã truyền kiến thức kĩ năng cần thiết để làm bài tập.

Đặc biệt, em xin gửi lời cảm ơn chân thành đến giáo viên hướng dẫn, quan tâm và cho em lời động viện cũng như kinh nghiệm giúp nhóm em hoàn thiện bài tập lớn.

Bài tập lớn của nhóm em gồm 2 chương:

CHƯƠNG I: CƠ SỞ LÝ THUYẾT

CHƯƠNG II: CÀI ĐẶT GIẢI THUẬT SẮP XẾP 2

MỤC LỤC

BẢNG CÁC TỪ VIẾT TẮT

STT	TỪ VIẾT TẮT	VIẾT ĐẦY ĐỦ
1	**FIFO**	First In First Out
2	**LIFO**	Last In First Out

CHƯƠNG 1. CƠ SỞ LÍ THUYẾT

1.1.Cấu trúc dữ liệu

Định nghĩa: Cấu trúc dữ liệu là một cách lưu dữ liệu trong máy tính sao cho chúng có thể được sử dụng một các hiệu quả. Một cấu trúc dữ liệu được thiết kế tốt cho phép thực hiện nhiều phép toán,sử dụng càng ít tài nguyên, thời gian xử lý và không gian bộ nhớ càng tốt .

Hình1.1. Cấu trúc dữ liệu

Các cấu trúc dữ liệu thường dùng: Mảng; Ngăn xếp; Hàng đợi; Bảng băm; Danh sách liên kết; Cây(tree); Đồ thị (graph).

1.2.Giải thuật

Định nghĩa: Là dãy các câu lệnh chặt chẽ và rõ ràng xác định một trình tự các thao tác trên một số đối tượng nào đó, sao cho sau một số hữu hạ bước thực hiện ta đạt được kết quả mong muốn.

Mỗi thuật toán có một dữ liệu vào (Input) và một dữ liệu ra(output);

a) Phân tích thuật toán

Thuật toán là các chỉ dẫn rõ ràng từng bước để giải quyết một vấn đè nào đó.Mục tiêu của việc phân tích thuật toán là để so sánh các thuật toán với nhau và ý nghĩa của việc so sánh này liên quan đến thời gian chạy. Phân tích thời gian chạy là quá trình xác định thời gian sử lý tăng thế nào khi kích thước của vấn đề (kích cỡ đàu vào) tăng.

- Kích cỡ đầu vào là số lượng các phần tử đầu vào, phụ thuộc vào loại vấn đề và có thể có nhiều dạng.

b) So sánh các thuật toán

So sánh dựa trên thời gian thực thi (Execution times): Đây không phải giải pháp tốt vì thời gian thực thi trên các máy tính khác nhau là khác nhau.

So sánh dựa trên số lượng câu thực hiện: Đây cũng không phải là biện pháp tốt do số lượng câu lệnh có thể thay đổi tùy vào ngôn ngữ lập trình...

Giải pháp:

Biểu thị thời gian của một thuật toán cho trước như một hàm số f(n) với n là kích cỡ đàu vào.

Với nhiều thuật toán, sẽ có nhiều hàm số f(n) tương ứng. Ta so sánh các hàm này với nhau, điều này đồng nghĩa với việc so sánh các thời gian chạy.

So sánh bằng cách này độc lập với thời gian thực thi máy tính, phong cách lập trình.

c) Kết quả phân tích

Có 3 dạng kết quả phân tích:

- Trường hợp tệ nhất (Worse case): Xác định đầu vào làm cho thuật toán tốn nhiều thời gian để hoàn thành nhất.
- Trường hợp trung bình (Average case): Cung cấp một dự báo về thời gian chạy của thuật toán. Chạy thuật toán nhiều lần, sử dụng các đầu vào ngẫu nhiên khác nhau, tính tổng thời gian chạy rồi chia cho số lần chạy thử nghiệm.
- Trường hợp tốt nhất (Best case): Xác định đàu vào làm cho thuật toán tốn ít thời gian để hoàn thành nhất.

1.3.Đệ quy

Đệ quy: Đệ quy là hàm mà hàm đó tự gọi chính mình. Một hàm đệ quy giải quyết bài toán đưa ra bằng cách chạy nội dung chương trình của chính nó trên bài toán nhỏ hơn được gọi là bước đệ quy. Để đảm bảo đệ quy có thể kết thúc, dãy cá bài toán nhỏ dần, cuối cùng hội tụ về một trường hợp cơ sở nào đó. Đệ quy rất hưu ích với bài toán mà chuỗi các nhiệm vụ con (nhiệm vụ con của nhiệm vụ con) đều thực hiện tương tự một vấn đề.

Công thức đệ quy

Định nghĩa: là công thức biểu diễn a_n dưới dạng một hoặc nhiều thành phần trước nó trong dãy $a_0\, a_1 \ldots a_{(n-1)}$ với mọi số nguyên. Một dãy số được gọi là một lời giải của công thức đệ quy nếu các thành phần trong dãy số đó thỏa mãn công thức đệ quy.

Nếu cả công thức đệ quy và các điều kiện đầu đều được xác định, thì dãy số lời giải của công thức đệ quy sẽ được xác định duy nhất. Công thức đệ quy là công thức cho phép tính giá trị của các đại lượng theo từng bước, dựa vào các giá trị tính ở các bước trước và một số giá trị đầu.

Một số phương pháp giải:

- Phương trình đặc trưng giải Công thức đệ quy tuyến tính thuần nhất hằng số(sẽ viết tắt là CTDQ TTTNHSH)
- Phương pháp thế xuôi
- Phương pháp thế ngược
- Cây đệ quy

Cấu trúc hàm đệ quy

- Cấu trúc 1

  IF(điều kiện){

  Thực hiện hành động

  }**ELSE**{

  -Gọi hàm đệ quy này thường dùng với hàm đệ quy là hàm có gái trị trả về.
- Cấu trúc 2

  IF(chưa gặp điều kiện dừng){

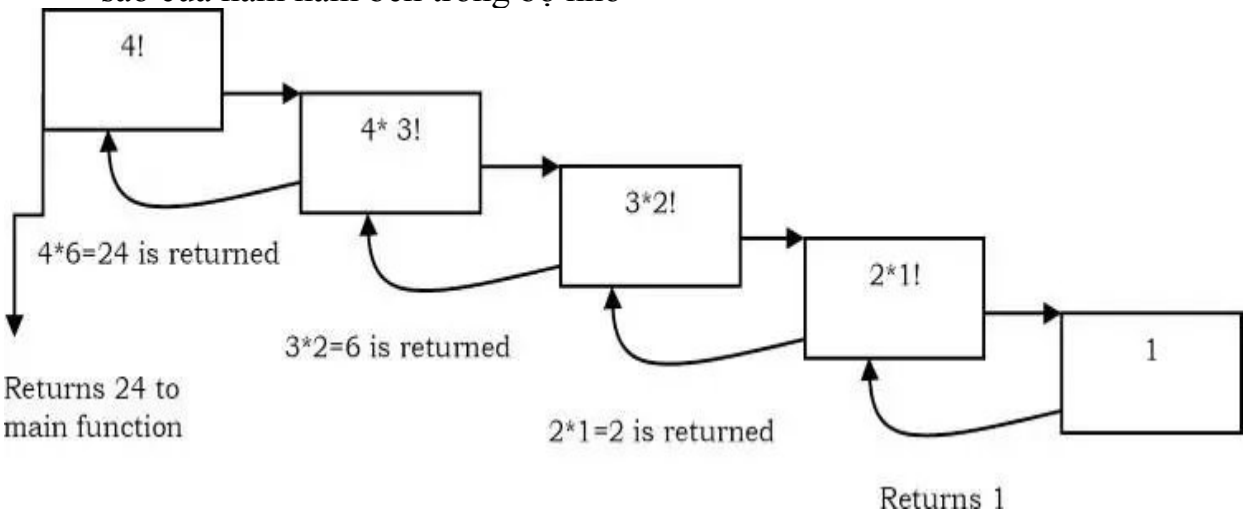
  Gọi hàm đệ quy trên bài toán tương tự nhưng nhỏ hơn.

  }

  -Cấu trúc này thường dùng với hàm đệ quy là hàm không có giá trị trả về.

Đệ quy và bộ nhớ

- Mỗi khi thực hiện một lời gọi đệ quy, một bản sao của phương thức đệ quy sẽ được tạo ra trong bộ nhớ.
- Bản sao này sẽ chỉ tồn tại trong bộ nhớ cho đến khi phương thức đệ quy kết thúc.
- Chính vì vậy khi gọi đến một hàm đệ quy, sẽ có thể tồn tại rất nhiều bản sao của hàm nằm bên trong bộ nhớ



Hình1.2. Đệ quy

1.4.Quay lui

Giới thiệu quay lui

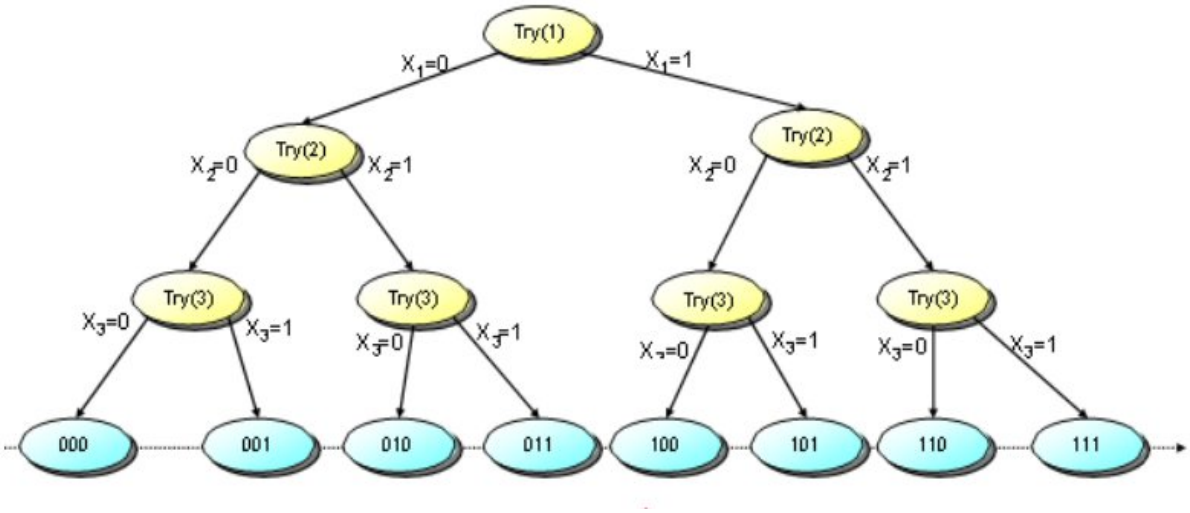
Quay lui là một chiến lược tìm kiếm lời giải cho các bài toán thỏa mãn các ràng buộc, các bài toán này có một lời giải đầy đủ. Các bài toán này bao gồm một tập các biến mà mỗi biến cần được gán một giá trị tùy theo các ràng buộc cụ thể của bài toán. Việc quay lui là để thử tất cả các tổ hợp có thể có để tìm một lời giải.

Quay lui: Quay lui (Backtracking) là một cải tiến của cách tiếp cận vét cạn để tìm trạng thái đầu ra mong muốn của bài toán. Mỗi trạng thái của bài toán được tạo nên bằng cách xây dựng từng phần tử.

Ví dụ: Trạng thái cần tìm kiếm có dạng x[n](một danh sách n phần tử). Lúc này thuật toán quay lui sẽ thực hiện các bước

- Thử tất cả giá trị mà x[1] có thể thực hiện.
- Với mỗi giá trị thử x[1], ta thử tất cả các giá trị mà x[2] có thể nhận.
- ...
- Với mỗi giá trị thử x[n-1], ta thử tất cả các giá trị mà x[n] có thể nhận.
- Với mỗi giá trị thử x[n],bài toán có trạng thái là "x[1], x[2],...,x[n]".

Ví dụ minh họa



Hình1.3. Quay lui

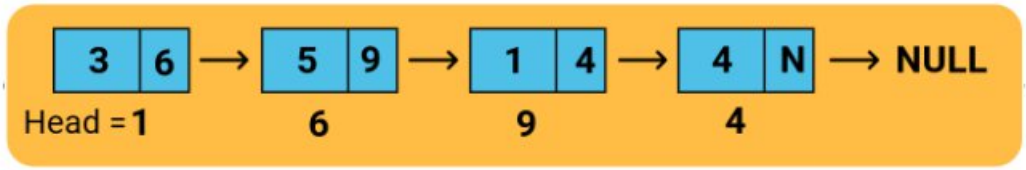
1.5.Các cấu trúc dữ liệu cơ bản

a)Danh sách liên kết đơn

Danh sách liên kết đơn - Singly linked list là một cấu trúc dữ liệu, nó tương tự như một mảng động với những tính chất quan trọng như :

- Mở rộng và thu hẹp một cách linh hoạt
- Các phần tử trong DSLK gọi là node và được cấp phát động khi cần
- Số lượng phần tử trong DSLK phụ thuộc vào bộ nhớ heap
- Dễ dàng chèn và xóa phần tử
- Các phần tử trong DSLK không có thứ tự
- Truy cập phần tử trong DSLK cần truy cập tuần tự không thể truy cập qua chỉ số
- Mỗi node trong DSLK cần có thêm 1 con trỏ để lưu liên kết

Mỗi phần tử trong DSLK được gọi là một node hay nút, node sẽ lưu thông tin dữ liệu (ví dụ như một số nguyên, 1 chuỗi ký tự, 1 sinh viên...) và ngoài ra cần có phần liên kết, phần liên kết này giúp các node có thể liên lạc với nhau. Mỗi node sẽ lưu thêm địa chỉ của node phía sau nó trong DSLK thông qua 1 thuộc tính con trỏ. Node cuối cùng trong danh sách liên kết thì phần liên kết của nó sẽ lưu con trỏ NULL



Hình1.4. Cấu trúc dữ liệu

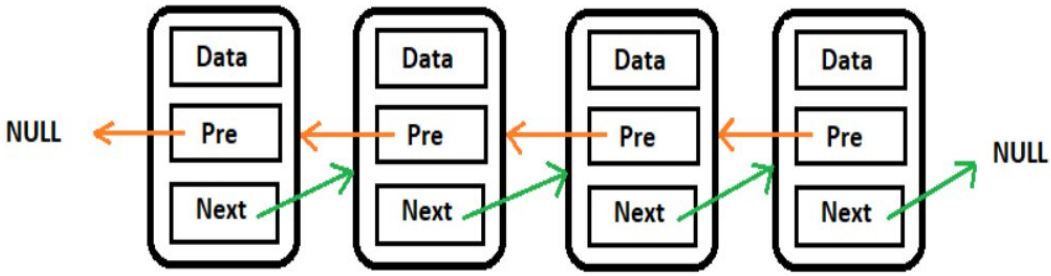
b)Danh sách liên kết đôi

Danh sách liên kết đôi (Doubly Linked List) là một tập hợp các Node được phân bố, được sắp xếp theo cách sap cho mỗi Node chứa:

Một giá trị (Data)

Một con trỏ(Next) sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó, nếu con trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của doulist.

Một con trỏ(Pre) sẽ trỏ đến phần tử trước của danh sách liên kết đó, nếu con trỏ mà trỏ đến NULL, nghĩa là đó là phần tử đầu tiên của doulist.



Hình1.5. Danh sách liên kết đôi

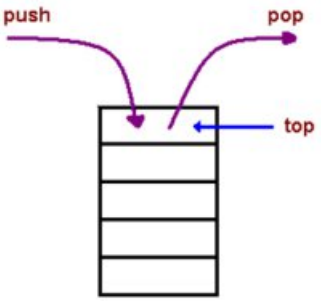
1.6.Stack và Queue

a)Ngăn xếp

Ngăn xếp(Stack): là một danh sách các phần tử đồng nhất, trong đó việc thêm hay bớt các phần tử chỉ xuất hiện ở một đầu, được gọi là đỉnh của ngăn xếp. Phần tử được thêm vào ngăn xếp cuối cùng thì sẽ được lấy ra đầu tiên. Do đó, ngăn xếp còn được gọi là cấu trúc dữ liệu LIFO (Last In First Out).

Một số ứng dụng của ngăn xếp:

- Đảo ngược xâu
- Đổi 1 số sang số nhị phân
- Kiểm tra các khối lồng nhau:{}, [],(),...
- Tính giá trị của biểu thức hậu tố (postfix)



Một số thao tác với ngăn xếp:

- Khởi tạo Stack rỗng
- Kiểm tra Stack rỗng
- Kiểm tra Stack đầy
- Thêm phần tử vào Stack
- Lấy phần tử đầu Stack nhưng không xóa
- Loại bỏ phần tử đầu Stack
- Nhập Stack
- Xuất Stack

b)Hàng đợi

Hàng đợi (Queue) là một danh sách phần tử đồng nhất, trong đó việc thêm phần tử được thực hiện ở một đầu danh sách và bớt (loại bỏ) phần tử được thực hiện ở đầu còn lại. Chính vì vậy, phần tử được thêm đầu tiên vào hàng đợi thì cũng sẽ được lấy ra đầu tiên. Do đó, hàng đợi còn được gọi là cấu trúc dữ liệu FIFO (First In First Out)

Ứng dụng hàng đợi

- Ứng dụng trong thực tế:
  - Mua vé xem phim, thanh toán tại siêu thị, quầy thu ngân
- Ứng dụng trong máy tính:
  - Khử đệ qui, tổ chức lưu vết trong tìm kiếm theo chiều rộng và quay lui, vét cạn, tổ chức quản lý và phân phối tiến trình trong các hệ điều hành, tổ chức bộ đệm.

Một số thao tác chính trên hàng đợi:

- Tạo một hàng đợi rỗng (hàng đợi không dữ liệu)
- Kiểm tra xem hàng đợi có rỗng hay không
- Lấy giá trị của phần tử đầu tiên trong hàng đợi
- Lấy giá trị của phần tử cuối cùng trong hàng đợi
- Thêm một phần tử vào phần sau của hàng đợi
- Loại bỏ phần tử nằm phía trước hàng đợi

1.7.Cấu trúc cây

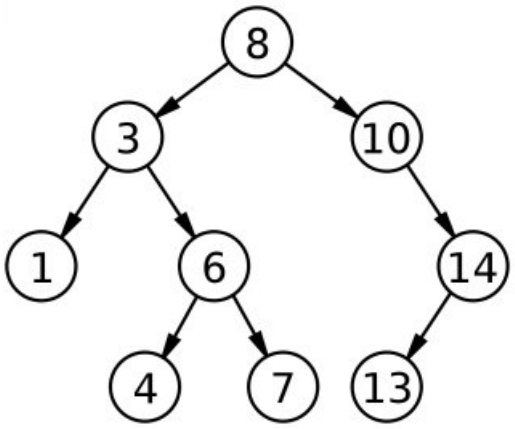
a)Cây nhị phân – Binary tree

Khái niệm:

- Bậc của nút là số cây con của nút đó
- Nút gốc là nút không có nút cha
- Nút là nút có bậc bằng 0
- Nút nhánh là nút có bậc khác 0 và không phải nút gốc
- Đọ dài đường đi từ nút gốc đến nút x là số nhánh cần đi qua kể từ nút gốc

đến x.

- Độ cao của cây là độ dì đường đi từ gốc đến nút lá ở mức thấp nhất



Hình1.6. Cấu trúc cây

Một số khái niệm

Node gốc (Root): Node đầu tiên định hình cây.

Node cha (Father): Node A là node cha của node B nếu B hoặc là node con bên trái của node A (left son) hoặc B là node con bên phải của node A (right son).

Node lá (Leaf): Node không có node con trái, không có node con phải.

Node trung gian (Internal Node): Node hoặc có node con trái, hoặc có node con phải, hoặc cả hai hoặc cả hai.

Node trước (Ancestor): Node A gọi là node trước của node B nếu cây con node gốc là A chứa node B.

Node sau trái (left descendent): node B là node sau bên trái của node A nếu cây con bên trái của node A chứa node B.

Node sau phải (right descendent): node B là node sau bên phải của node A nếu cây con bên phải của node A chứa node B.

Node anh em (brother): A và B là anh em nếu cả A và B là node con trái và node con phải của cùng một node cha.

Bậc của node (degree of node): Số cây con tối đa của node.

Mức của node (level of node): mức node gốc có bậc là 0, mức của các node khác trên cây bằng mức của node cha cộng thêm 1.

Chiều sâu của cây (depth of tree): mức lớn nhất của node lá trong cây. Như vậy, độ sâu của cây bằng đúng độ dài đường đi dài nhất từ node gốc đến node lá.

Các loại cây

Cây lệch trái: Cây chỉ có node con bên trái.

Cây lệch phải: Cây chỉ có node con bên phải.

Cây nhị phân đúng: Node gốc và tất cả các node trung gian có đúng 2 node con.

Cây nhị phân đầy: Cây nhị phân đúng và tất cả node lá đều có mức là d.

Cây nhị phân gần đầy: Cây nhị phân gần đầy có chiều sâu d là cây nhị phân thỏa mãn:

- Tất cả node con có mức không nhỏ hơn d-1 đều có 2 node con.
- Các node ở mức d đầy từ trái qua phải
- Cây nhị phân hoàn toàn cân bằng: Cây nhị phân có số node thuộc nhánh cây con trái và số node thuộc nhánh cây con phải chênh lệch nhau không quá 1.

Cây nhị phân tìm kiếm: Cây nhị phân thỏa mãn điều kiện:

- Hoặc là rỗng hoặc có một node gốc.
- Mỗi node gốc có tối đa hai cây con. Nội dung node gốc lớn hơn nội dung con bên trái và nhỏ hơn nội dung node con bên phải.
- Hai cây con bên trái va bên phải cũng hình thành nên hai cây tìm kiếm.

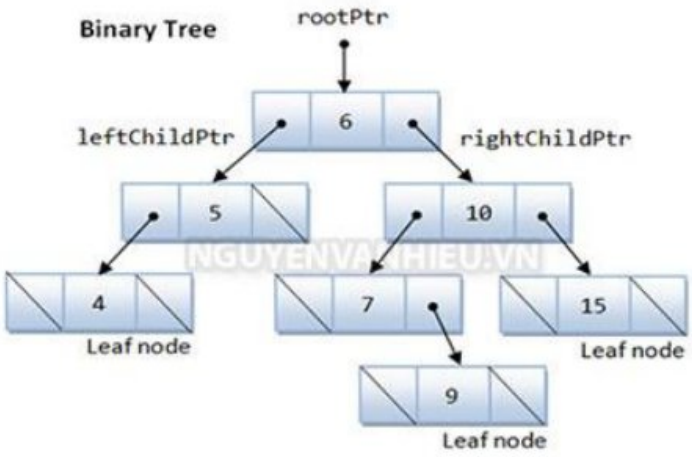
Cây nhị phân tìm kiếm hoàn toàn cân bằng: Cây nhị phân tìm kiếm có chiều sâu cây con bên trái và chiều sâu cây con bên phải chênh lệch nhau không quá 1

Đặc điểm cây nhị phân

- Là cây có số con lớn nhất là 2

- Giá trị của nút bất kỳ luôn lớn giá trị của tất cả các nút bên trái và nhỏ hơn giá trị của các nút bên phải.
- Nút có giá trị nhỏ nhất nằm ở trái nhất của cây
- Nút có giá trị lớn nhất nằm ở phải nhất của cây

```
typedef int Item;
struct TNode
{
    Item Key;
    TNode *Left, *Right;
};
typedef TNode *Tree;
```



b) AVL Tree

Khái niệm: Cây nhị phân tìm kiếm cân bằng là cây mà tại mỗi nút của nó độ cao của cây con trái và của cây con phải chênh lệch không quá một.

Lịch sử cây cân bằng(AVL Tree) là viết tắt của các tác giả người nga đã đưa ra định nghĩa của cây cân bằng Adelson-Velskii và Landis (1962). Từ cây AVL người ta đã phát triển thêm nhiều loại CTDL, hữu dụng như đỏ-đen(Red-Back Tree), B-tree,..

Chỉ số cân bằng của một nút là chỉ số cân bằng của một nít là hiệu của chiều cao cây con phải và cây con trái của nó. Đối với một cây cân bằng, chỉ số cân bằng (CSCB) của mỗi nút chỉ có thể mang một trong các giá trị

- CSCB(p) = 0 ⇔ Độ cao cây trái (p) = Độ cao cây phải (p)
- CSCB(p) = 1 ⇔ Độ cao cây trái (p) < Độ cao cây phải (p)
- CSCB(p) =-1 ⇔ Độ cao cây trái (p) > Độ cao cây phải (p)

**Biểu diễn*

```
#defineLH -1        /* Cây con trái cao hơn */
#defineEH 0         /* Hai cây con bằng nhau */
```

```
#defineRH 1          /* Cây con phải cao hơn */
Struct AVLNode{
        Char   balFactor;//Chỉ số cân bằng
        DataType     data;
        tagAVLNode*       pLeft;
        tagAVLNode*       pRight;
};
Typedef AVLNode* AVLTree;
```

Trường hợp thêm hay hủy một phần tử trên cây có thể làm cây tăng hay giảm chiều cao, khi đó phải cân bằng lại cây.

Việc cân bằng lại một cây sẽ phải thực hiện sao cho chỉ ảnh hưởng tối thiểu đến cây nhằm giảm thiểu chi phí cân bằng.

Các thao tác đặc trưng của cây AVL:

- Thêm một phần tử vào cây AVL
- Hủy một phần tử trên cây AVL
- Cân bằng lại một cây vừa bị mất cân bằng

Các trường hợp mất cân bằng:

- Ta sẽ không khảo sát tính cân bằng của 1 cây nhị phân bất kỳ mà chỉ quan tâm đến các khả năng mất cân bằng xảy ra khi thêm hoặc hủy một nút trên cây AVL. Như vậy, khi mất cân bằng, độ lệch chiều cao giữa 2 cây con sẽ là 2
- Có 6 khả năng sau:

- Trường hợp 1- Cây T lệch về bên trái : 3 khả năng
- Trường hợp 2- Cây T lệch về bên phải: 3 khả năng

Các trường hợp mất cân bằng:

Các trường hợp lệch về bên phải hoàn toàn đối xứng với các trường hợp lệch về bên trái. Vì vậy, chỉ cần khảo sát trường hợp lệch về bên trái.

Trong 3 trường hợp lệch về bên trái, trường hợp T1 lệch phải là phức tạp nhất. Các trường hợp còn lại giải quyết rất đơn giản.

1.8.Thuật toán tìm kiếm

Bài toán tìm kiếm

Cho dãy gồm n đối tượng r1 , r2 , .., r_N . Mỗi đối tượng r_I được tương ứng với một khóa ki ($1 \leq i \leq n$).

Nhiệm vụ của tìm kiếm là xây dựng thuật toán tìm đối tượng có giá trị khóa là X cho trước.

Công việc tìm kiếm bao giờ cũng hoàn thành bởi một trong hai tình huống:

- Nếu tìm thấy đối tượng có khóa X trong tập các đối tượng thì ta nói phép tìm kiếm thành công (successful)
- Nếu không tìm thấy đối tượng có khóa X trong tập các đối tượng thì ta nói phép tìm kiếm không thành công (unsuccessful)

Tìm kiếm nhị phân

Thuật toán tìm kiếm nhị phân là phương pháp định vị phần tử x trong một danh sách A[] gồm n phần tử đã được sắp xếp.

Quá trình tìm kiếm bắt đầu bằng việc chia danh sách thành hai phần. Sau đó, so sách x với phần từ ở giữa.

Khi đó có 3 trường hợp có thể xảy ra:

- **Trường hợp 1**: nếu x bằng phần tử ở giữa A[mid], thì mid chính là vị trí của x trong danh sách A[].
- **Trường hợp 2**: Nếu x lớn hơn phần tử ở giữa thì nếu x có mặt trọng dãy A[] thì ta chỉ cần tìm các phần tử từ mid+1 đến vị trí thứ n.
- **Trường hợp 3**:Nếu x nhở hơn A[mid] thì x chỉ có thể ở dãy con bên trái của dãy A[].

Lặp lại quá trình trên cho đến khi cận dưới vượt cận trên của dãy A[] mà vẫn chưa tìm thấy x thì ta kết luận x không có mặt trong dãy A[].

Độ phức tạp thuật toán là: O(log(n)).

Đầu vào: Mảng A gồm n phần tử:A[0],...,A[n-1] đã được sắp xếp theo thứ tự tăng dần. Giá trị target có kiểu dữ liệu giống như các phần tử của mảng A

Đầu ra: Chỉ số phần tử mảng có giá trị target nếu target tìm thấy trong A,-1 nếu target không có trong A

Thuật toán: Lấy phần tử giữa A[middle] với middle – n/2

If target< A[middle]:

- Target không xuất hiện ở nửa bên phải dãy gồm các phần tử A[middle+1]...A[n-1], vì vậy có thể bỏ qua dãy nửa bên phải trong quá trình tìm kiếm
- Lặp lại thao tác này cho nửa bên trái gồm A[0]...A[middle-1]
  Else If target>A[middle]:

- Target không xuất hiện ở nửa bên trái gồm các phần tử từ A[0]...A[middle], vì vậy có thể bỏ qua nửa bên trái
- Lặp lại thao tác này cho nửa bên phải gồmA[middle+1]...A[n-1]
- Else If target==A[middle], giá trị target tìm thấy ở vị trí middle
- If toàn bộ mảng đã được tìm kiếm mà không thấy target, thì kết luận: target không có trong mảng

Cây nhị phân

Xây dựng cấu trúc để biểu diễn các tập biến động (dynamic sets). Các phần tử có khóa (key) và thông tin đi kèm(satellite data). Tập động cần hỗ trợ các truy vấn (queries):

- **Search(s,k):** Tìm phần tử có khóa k trong tập S
- **Minimum(S), Maximum(S):** Tìm phần tử có khoá nhỏ nhất, lớn nhất trong tập S
- **Predecessor(S,x),Successor(S,x):** Tìm phần tử kế cận trước, kế cận sau

của x trong tập S

- **Insert(S,x):** Bổ sung (chèn) x vò S
- **Delete(S,x):** Loại bỏ (xóa) x khỏi S

Cây nhị phân tìm kiếm là cấu trúc dữ liệu quan trọng để biểu diễn tập động, trong đó tất cả các thao tác đều được thực hiện với thời gian O(h), trong đó h là chiều cao của cây

Cây nhị phân tìm kiếm(viết tắt là BST) có tính chất:

- Mỗi nút x(ngoài thông tin đi kèm) có các trường:
- **Left**: con trỏ đến con trái
- **Right:** con trỏ đến con phải
- **Parent:** con trỏ đến cha(trường này là tùy chọn)
- **Key:** khóa (thường giả thiết là khoá của các nút là khác nhau từng đôi, trái lại nếu có khoá trùng nhau thì cần chỉ rõ thứ tự của hai khoá trùng nhau)
- Mỗi nút có một khóa duy nhất
- Tất cả các khóa trong cây con trái của nút đều nhỏ hơn khoá của nút
- Tất cả các khoá trong cây con phải của nút đều lớn hơn hoặc bằng khoá của nút

Các phép toán

- Tìm kiếm (search)
- Tìm cực đại, cực tiểu(Findmax,Findmin)
- Kế cận trước, kế cận sau(Predecessor, Successor)
- Chèn (Insert)
- Xóa (Delete)

Độ phức tạp trung bình của các thao tác với BST

Cây nhị có n nút có độ cao tối thiểu là log2n. tình huống tốt nhất xảy ra khi ta dựng được BST là cây nhị phân đầy đủ (là cây có độ cao thấp nhất có thể được).

Có hai cách tiếp cận nhằm đảm bảo độ cao của cây là O(log2n): Luôn giữ cho cây là cân bằng tại mọi thời điểm (AVLTrees). Thỉnh thoảng lại kiểm tra lại xem cây

có "quá mất cân bằng" hay không và nếu điều đó xảy ra thì ta cần tìm cách cân bằng lại nó (Splay Trees [Tarjan]).

CHƯƠNG 2. CÀI ĐẶT GIẢI THUẬT SẮP XẾP 2

2.1.Giới thiệu về thuật toán sắp xếp

Bài toán sắp xếp là cách sắp xếp lại các phần tử theo chiều tăng hay giảm dần theo tiêu chí

Vd: Sắp xếp danh sách theo điểm trung bình giảm dần theo tiêu chí

Giúp có cái nhìn tổng quát về dữ liệu dễ dàng tìm kiếm

Thuật toán sắp xếp là các phương pháp hoặc kỹ thuật được sử dụng để sắp xếp các phần tử trong một mảng hoặc danh sách theo một thứ tự nhất định, thường là tăng dần hoặc giảm dần. Mục đích chính của sắp xếp là làm cho thao tác tìm kiếm phần tử trên tập đó được dễ dàng hơn.

Giải thuật sắp xếp được chia thành 2 loại:

Loại 1: là các giải thuật được cài đặt đơn giản, nhưng không hiệu quả(phải sử dụng nhiều thao tác).

Loại 2: là các giải thuật được cài đặt phức tạp, nhưng hiệu quả hơn về mặt tốc độ(dùng ít thao tác hơn).

Đối với các tập dữ liệu ít phần tử, tốt nhất là nên lựa chọn loại thứ nhất.đối với tập có nhiều phần tử, loại thứ hai sẽ mang lại hiệu quả hơn.

Các đối tượng dữ liệu cần sắp xếp thường có nhiều thuộc tính, và ta cần chọn một thuộc tính làm khóa để sắp xếp dữ liệu theo khóa này. Ví dụ, đối tượng về người thường có các thuộc tính cơ bản như họ tên, ngày sinh, giới tính,v.v... tuy nhiên họ thường được chịn làm khóa để sắp xếp.

Một vấn đề nữa cần phải chú ý khi thực hiện sắp xếp, đó là tính ổn định của giải thuật sắp xếp. Một giải thuật được gọi là ổn định nếu sau khi sắp xếp, nó giữ

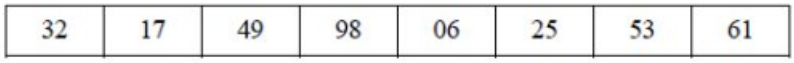
nguyên vị trí của các phần tử có cùng giá trị khóa. Chẳng hạn, với danh sách theo họ tên các sinh viên trong lớp.

2.2.Sắp xếp chọn

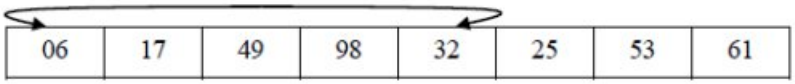
Ý tưởng:

Tìm kiếm phần tử nhỏ nhất và đổi chỗ nó với phần tử đầu tiên. Tìm kiếm phần tử nhỏ nhất trong phần còn lại và đổi chỗ nó với phần tử thứ hai. Tiếp tục cho đến khi toàn bộ đã được sắp xếp.

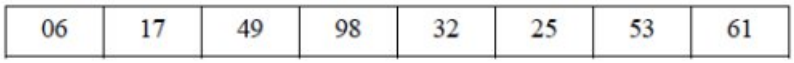
VD:



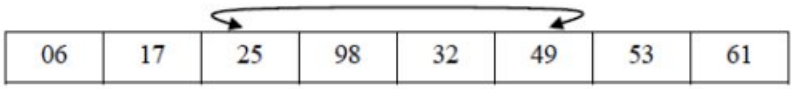
Bước1: Chọn phần tử nhỏ nhất là 06, đổi chỗ cho 32.



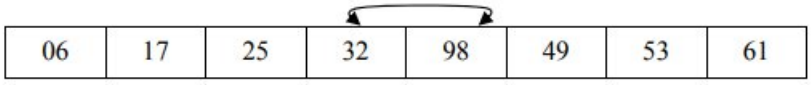
Bước2: Chọn được phần tử nhỏ thứ nhì là 17, đó chính là phần tử thứ 2 nên giữ nguyên.



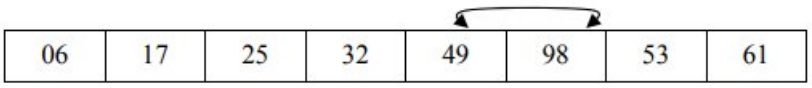
Bước3: Chọn được phần tử nhỏ thứ ba là 25, đổi chỗ cho 49.



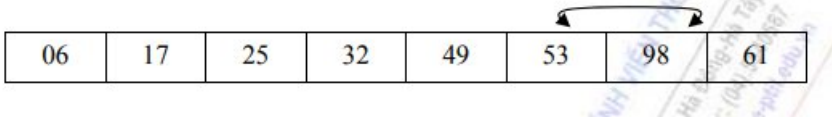
Bước4: Chọn được phần tử nhỏ thứ 4 là 32, đổi chỗ cho 98



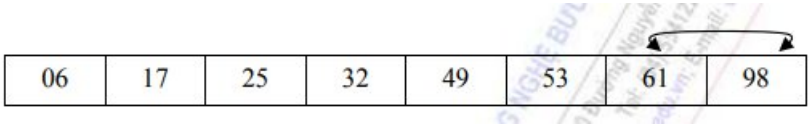
Bước5: Chọn được phần tử nhỏ thứ năm là 49, đổi chỗ cho 98



Bước6: Chọn được phần tử nhỏ thứ sáu là 53, đổi chỗ cho 98



Bước7: Chọn được phần tử nhỏ thứ bảy là 61, đổi chỗ cho 98



Độ phức tạp thuật toán

- Trường hợp tốt: O(n)2
- Trung bình: O(n)2
- Trương hợp xấu: O(n)2

Ứng dụng

Sắp xếp mảng nhỏvkhi mảng cần sắp xếp có kích thước nhỏ, thời gian thực thi của thuật toán sắp xếp chọn không chênh lệch nhiều so với các thuật toán phức tạp hơn.

Khi bộ nhớ có giới hạn do thuật toán sắp xếp chọn chỉ sử dụng không gian bộ nhớ bổ sung O(1) (tức là không cần thêm bộ nhớ để lưu trữ dữ liệu tạm thời), nên nó có thể hữu ích trong các tình huống mà bộ nhớ là một tài nguyên khan hiếm. Thuật toán sắp xếp chọn đơn giản và dễ hiểu, là một công cụ tốt để giảng dạy các khái niệm cơ bản về thuật toán sắp xếp trong các lớp học lập trình và khoa học máy tính.

Ứng dụng trong một số hệ thống nhúng với tài nguyên hạn chế, thuật toán sắp xếp chọn có thể là một lựa chọn phù hợp do tính đơn giản và yêu cầu bộ nhớ thấp. Khi cần tạo danh sách sắp xếp dựa trên một mảng không sắp xếp, thuật toán sắp xếp chọn có thể được sử dụng.

Ưu điểm

- Dễ hiểu và cài đặt
- Có thể sử dụng khi số lượng phần tử nhỏ hoặc danh sách gần như đã được sắp xếp.

Nhược điểm

- Độ phức tạp cao với các danh sách lớn.
- Sắp xếp lựa chọn có độ phức tạp về thời gian là $O(N^2)$ trong trường hợp xấu nhất và trung bình.
- Không hoạt động tốt trên các tập dữ liệu lớn.
- Không bảo toàn thứ tự tương đối của các mục có khóa bằng nhau nghĩa là nó không ổn định.

2.3.Sắp xếp trộn

Ý tưởng

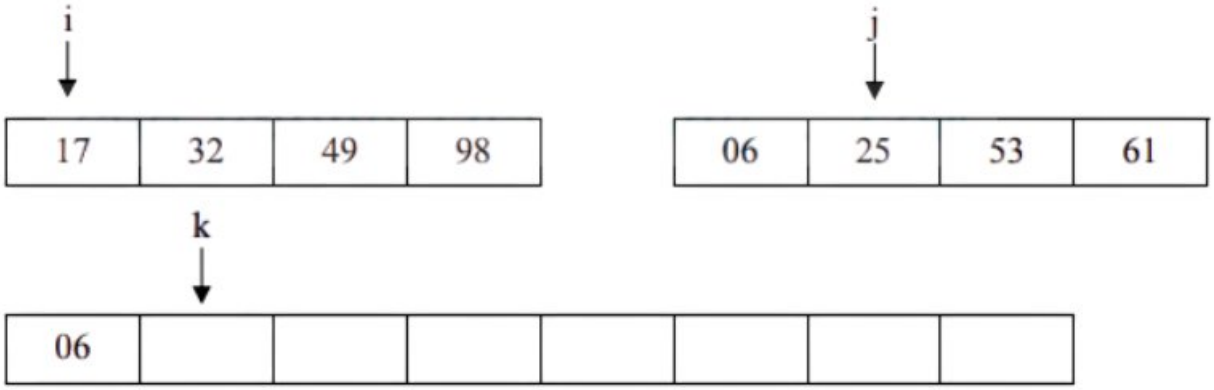
Giải thuật này bắt nguồn từ việc trộn 2 danh sách đã được sắp xếp thành 1 danh sách mới cũng được sắp. Rõ ràng việc trộn 2 dãy đã sắp thành 1 dãy mới được sắp có thể tận dụng đặc điểm đã sắp của 2 dãy con.

Để thực hiện giải thuật sắp xếp trộn đối với 1 dãy bất kỳ, bắt đầu, coi mỗi phần tử của dãy là một danh sách con gồm 1 phần tử được sắp. Tiếp theo, tiến hành trộn từng cặp 2 dãy con 1 phần tử kề nhau để tạo thành các dãy con 2 phần tử được sắp. Các dãy con 2 phần tử được sắp này lại trộn với nhau tạo thành 4 phần tử được sắp. Quá trình tiếp tục đến khi chỉ còn 1 dãy con duy nhất được sắp , đó là dãy ban đầu.

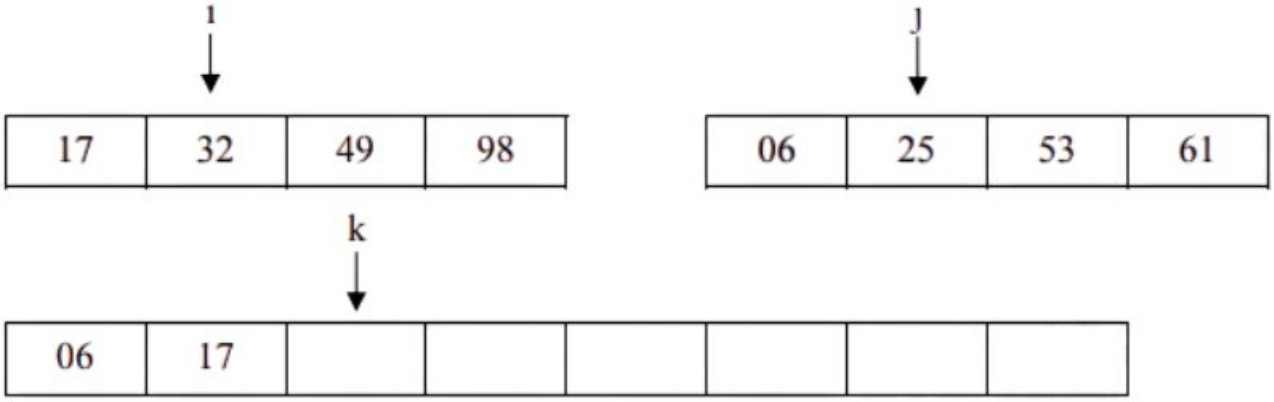
Thao tác đầu tiên cần thực hiện khi sắp xếp trộn là việc tiến hành trộn 2 dãy đã sắp thành 1 dãy mới cũng được sắp. Để làm việc này, ta sử dụng 2 biến duyệt từ đầu mỗi dãy. Tại mỗi bước, tiến hành so sánh giá trị của 2 phần tử tại vị trí của 2 biến duyệt. Nếu phần tử nào có gí trị nhỏ hơn, ta đưa phần tử đó xuống dãy mới và tăng biến duyệt tương ứng lên 1. Quá trình lặp lại cho tới khi tất cả các phần tử của 2 dãy được duyệt và xét.

Để trộn 2 dãy, sử dụng một dãy thứ 3 để chứa các phần tử của dãy tổng. Một biến k dùng để lưu giữ vị trí cần chèn tiếp theo trong dãy mới.

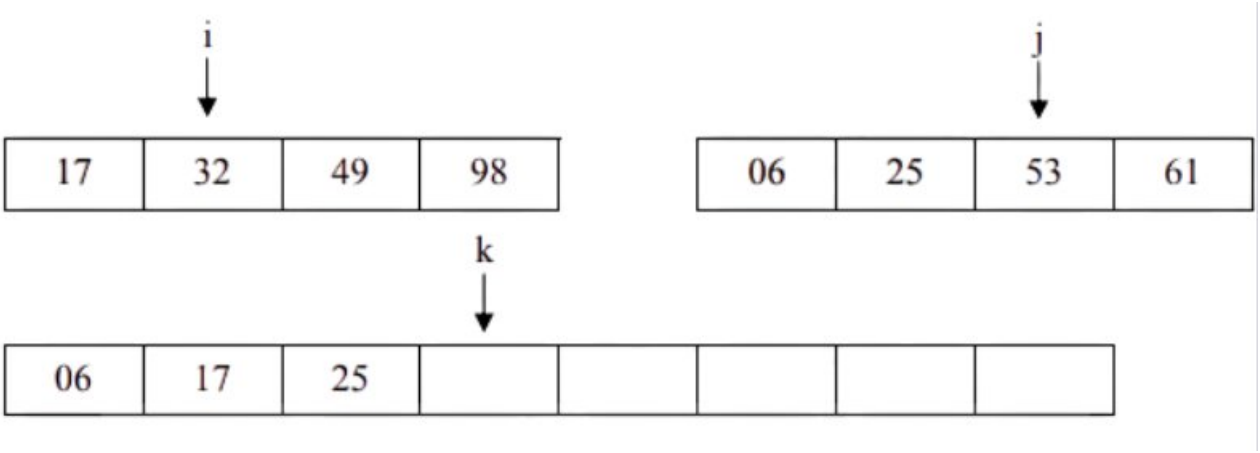
Bước 1: Phần tử tại vị trí biến duyệt j là 06 nhỏ hơn phần tử tại vị trí biến duyệt i là 17 nên ta đưa 06 xuống dãy mới và tăng j lên 1. Đồng thời, biến duyệt l cũng tăng lên 1



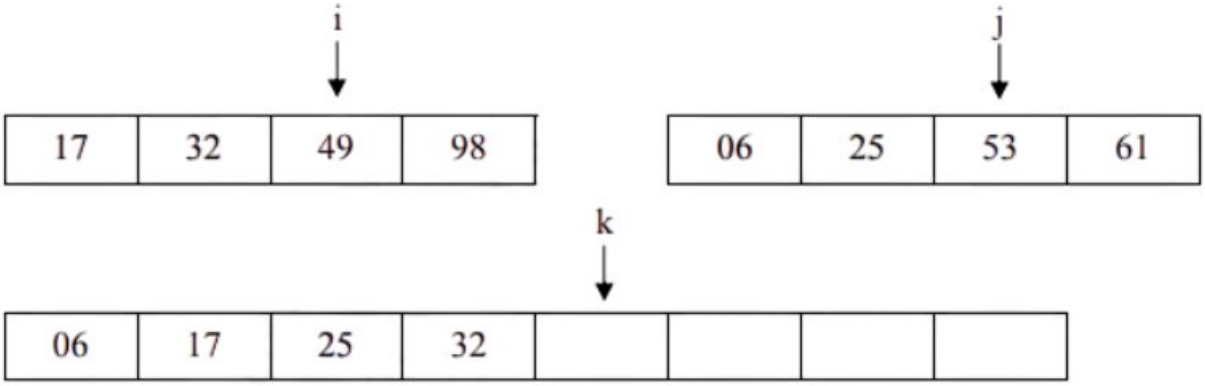
Bước 2: Phần tử tại vị trí i là 17 nhỏ hơn phần tử tại vị trí j là 25 nên ta đưa 17 xuống dãy mới và tăng i lên 1, biến duyệt k cũng tăng lên 1.



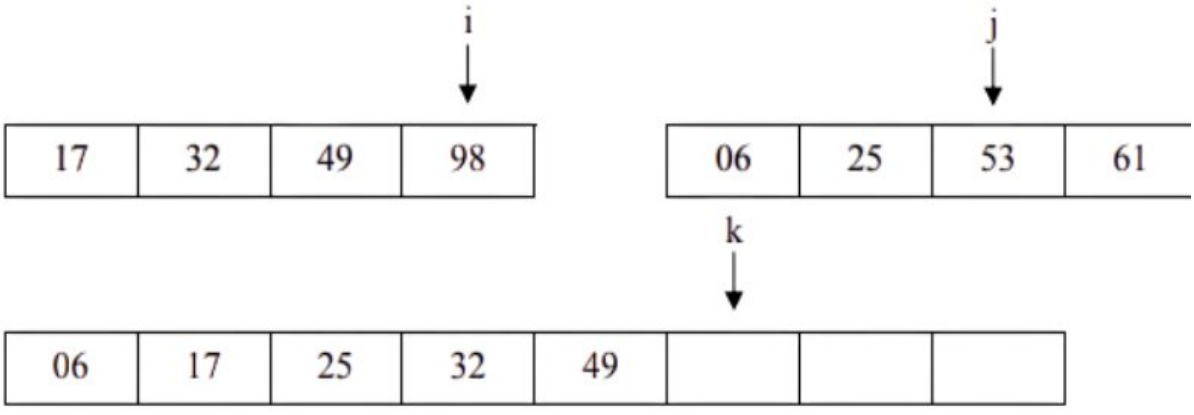
Bước 3: Phần tử tại vị trí j là 25 nhỏ hơn phần tử tại vị trí i là 32 nên ta đưa 25 xuống dãy mới và tăng j lên 1, biến duyệt k cũng tăng lên 1.



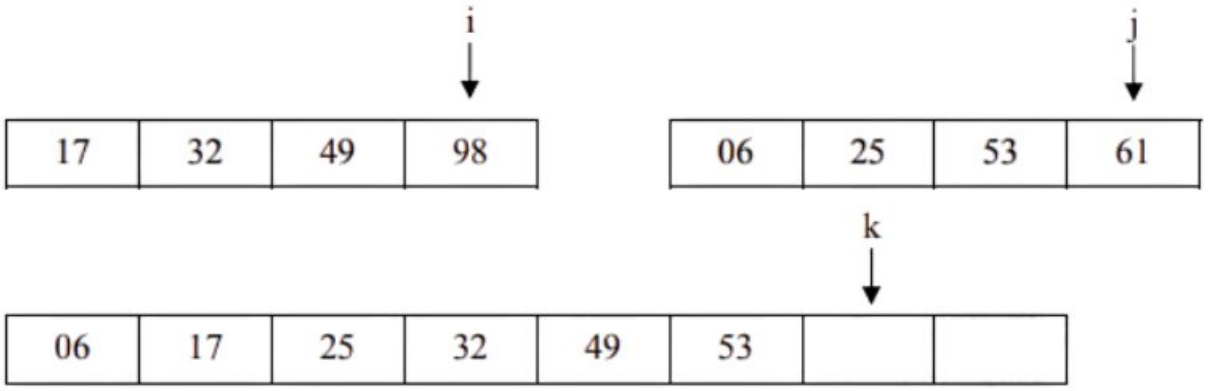
Bước 4: Phần tử tại vị trí i là 32 nhỏ hơn phần tử tại vị trí j là 53 nên ta đưa 32 xuống dãy mới và tăng i lên 1, biến duyệt k cũng tăng lên 1.



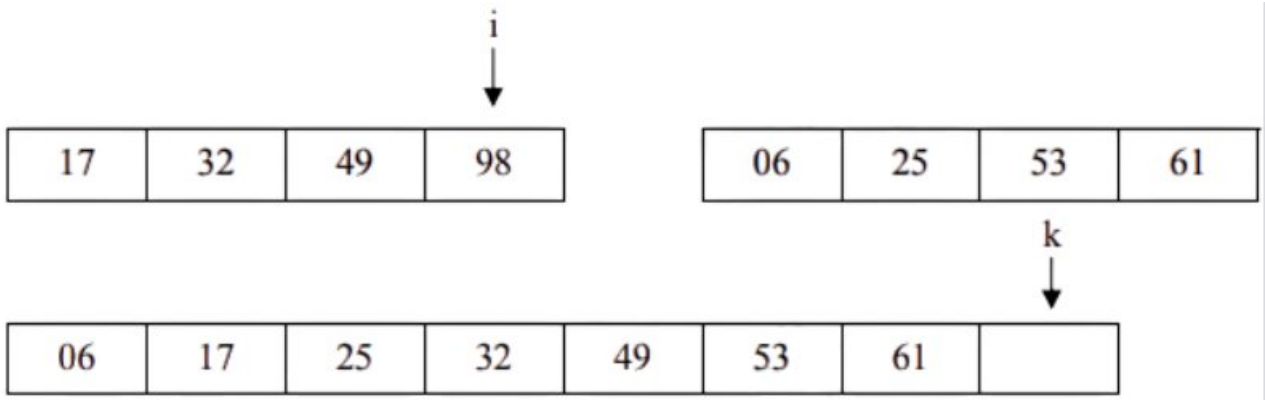
Bước 5: Phần tử tại vị trí i là 49 nhỏ hơn phần tử tại vị trí i là 53 nên ta đưa 49 xuống dãy mới và i lên 1, biến duyệt k cũng tăng lên 1.



Bước 6: Phần tử tại vị rí j là 53 nhỏ hơn phần tử tại vị trí i là 98 nên ta đưa 53 xuống dãy mới và tăng j lên 1, biến duyệt k cũng tăng lên 1



Bước 7: Phần tử tại vị trí j là 61 nhỏ hơn phần tự tại vị trí i là 98 nen ta đưa 61 xuống dãy mới và tăng j lên 1, biến duyệt k cũng tăng lên 1.



Bước 8: Biến duyệt j đã duyệt hết dãy thứ 2. Khi đó, ta tiến hành đưa toàn bộ phần còn lại của dãy 1 xuống dãy mới.

Độ phức tạp thuật toán:

- Trường hợp tốt:O(nlog(n))
- Trung bình: O(nlog(n))
- Trường hợp xấu: O(nlog(n))

Ứng dụng

Khi dữ liệu không thể chứa trong bộ nhớ (RAM) và cần được sắp xếp trên đĩa cứng, sắp xếp trộn (Merge sort) được sử dụng do khả năng xử lý các phần dữ liệu lớn

từng phần một và sau đó hợp nhất chúng lại với nhau. Merge Sort được sử dụng để sắp xếp các chuỗi ký tự hoặc văn bản, giúp tối ưu hóa và tăng tốc độ tìm kiếm và truy vấn trong các ứng dụng liên quan đến văn bản. Merge Sort có thể được sử dụng để sắp xếp các tập tin trong hệ thống tập tin, giúp tăng cường hiệu quả truy xuất và quản lý tập tin. Merge Sort đảm bảo rằng các phần tử có giá trị bằng nhau giữ nguyên thứ tự ban đầu của chúng sau khi sắp xếp, điều này rất quan trọng trong một số ứng dụng đặc thù. Merge Sort dễ dàng áp dụng trong các hệ thống xử lý song song và phân tán, nhờ khả năng phân chia và hợp nhất dữ liệu.

Ưu điểm

- Thuật toán sắp xếp ổn định, nghĩa là nó duy trì thứ tự tương đối của các phần tử bằng nhau trong mảng
- Thuật toán sắp xếp trộn có độ phức tạp thời gian trong trường hợp xấu nhất là O(nlog(n)).
- Phương pháp chia để trị rất đơn giản.
- Chúng tôi độc lập hợp nhất các mảng con khiến nó phù hợp cho việc xử lý song song.

Nhược điểm

- Thuật toán sắp xếp trộn yêu cầu thêm bộ nhớ để lưu trữ các mảng con đã hợp nhất trong quá trình sắp xếp.
- Merge Sort không phải là thuật toán sắp xếp tại chỗ, nghĩa là nó cần thêm bộ nhớ để lưu trữ dữ liệu đã sắp xếp. Điều này có thể là một bất lợi trong các ứng dụng mà việc sử dụng bộ nhớ là mối quan tâm.

2.4.Sắp xếp nhanh – Quick Sort

Ý tưởng

Quick Sort là một thuật toán sắp xếp được phát minh làn đầu bởi C.A.Hoare vào năm 1960. Đây có lẽ là thuật toán được nghiên cứu nhiều nhất và được sử dụng rộng rãi nhất trong các lớp thuật toán sắp xếp.

Ý tưởng dựa trên phương pháp chia để trị.

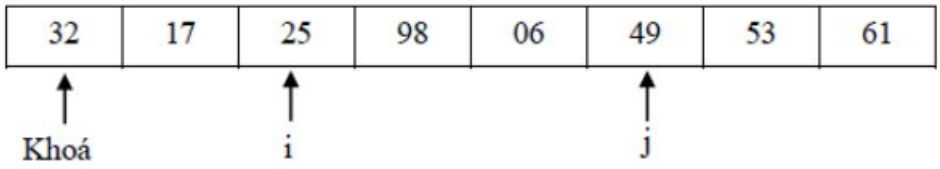
Giải thuật chia dãy cần sắp thành 2 phần, sau đó thực hiện việc sắp xếp cho mỗi phần độc lập với nhau.

Để thực hiện điều này,đầu tiên chọn ngẫu nhiên 1 phần tử nào đó của dãy làm khóa. Trong bước bước tiếp theo,các phần tử nhỏ hơn khóa phải được xếp vào phía trước khóa và các phần tử lớn hơn được xếp vào phía sau khóa. Để có được sự phân loại này, các phần tử sẽ đuốc sánh với khóa và hoán đổi vị trí cho nhau hoặc cho khóa nếu nó lớn hơn khóa mà lại nằm trước hoặc nhỏ hơn khóa mà lại nằm sau. Khi lượt hoán đổi đầu tiên thực hiện xong thì dãy được chia thành 2 đoạn: 1 đoạn bao gồm các phần tử nhỏ hơn khóa, đoạn còn lại bao gồm các phần tử lơn hơn khóa.

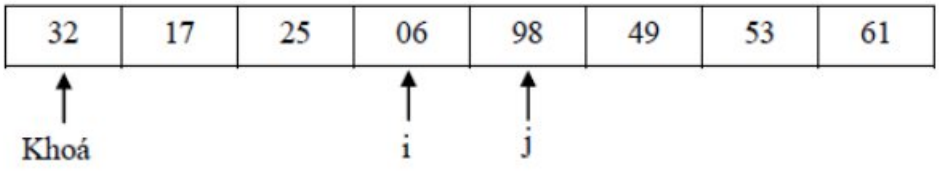
VD:

32	17	49	98	06	25	53	61

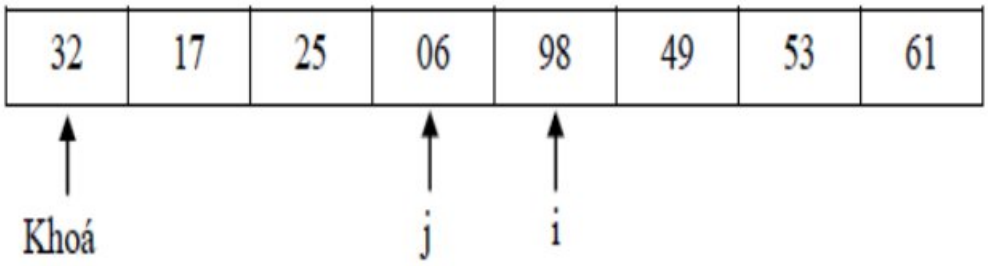
Chọn phần tử đầu tiên của dãy, phần tử 32, làm khóa. Quá trình duyệt từ bên trái với biến duyệt i sẽ dừng lại ở 49, vì đây là phần tử lớn hơn khóa. Quá trình duyệt từ bên phải với biến duyệt j sẽ dừng lại ở 25 vì đây là phần tử mhor hơn khóa. Tiến hành đổi chỗ 2 phần tử cho nhau.



Quá trình duyệt tiếp tục. Biến duyệt i dừng lại ở 98, còn biến duyệt j dừng lại ở 06. Lại tiến hành đổi vị trí 2 phần tử 98 và 06.



Tiếp tục quá trình duyệt. Các biến duyệt i và j gặp nhau và quá trình duyệt dừng lại



Dãy được chia làm 2 nửa. Nửa đầu từ phần tử đầu tiên đến phần phần tử j, bao gồm các phần tử nhỏ hơn hoặc bằn khóa. Nửa sau từ phần tử thứ i đến phần tử cuối, bao gồm các phần tử lơn hơn hoặc bằng khóa.

Quá trình duyệt và đổi chỗ được lặp lại với 2 nửa vừa được tạo ra, và cứ tiếp tục như vậy cho tới khi dãy được sắp hoàn toàn.

Độ phức tạp thuật toán:

- Trường hợp tốt:O(nlog(n))
- Trung bình: O(nlog(n))
- Trường hợp xấu: O(n^2)

Ứng dụng

Quick Sort được sử dụng để sắp xếp các bản ghi trong cơ sở dữ liệu, giúp tối ưu hóa việc tìm kiếm và truy xuất dữ liệu. Khi cần sắp xếp các bảng dữ liệu lớn, Quick Sort giúp giảm thời gian xử lý và tăng hiệu suất. Các ứng dụng yêu cầu xử lý và sắp xếp lượng dữ liệu lớn thường sử dụng Quick Sort để đảm bảo hiệu suất cao. Quick Sort giúp sắp xếp các chuỗi ký tự hoặc văn bản trong các ứng dụng xử lý ngôn ngữ tự nhiên. Quick Sort là một phần quan trọng trong các khóa học thuật toán và cấu trúc dữ liệu, giúp sinh viên hiểu rõ về kỹ thuật chia để trị. Quick Sort được sử dụng trong các hệ thống tập tin để sắp xếp và tổ chức các tập tin, giúp tối ưu hóa việc truy xuất và quản lý tập tin. Quick Sort được sử dụng để sắp xếp các pixel trong ảnh hoặc khung hình trong video, giúp cải thiện chất lượng và hiệu suất xử lý.

Quick Sort giúp sắp xếp các đối tượng trong trò chơi, chẳng hạn như sắp xếp theo điểm số, thời gian, hoặc các thuộc tính khác để tối ưu hóa trải nghiệm người chơi. Quick Sort được sử dụng để sắp xếp các sản phẩm trong danh sách hiển thị, giúp người dùng tìm kiếm sản phẩm nhanh chóng và dễ dàng. Quick Sort giúp sắp xếp các mục trong giỏ hàng theo các tiêu chí như giá cả, số lượng, hoặc tên sản phẩm. Quick Sort giúp sắp xếp các giao dịch tài chính theo thời gian, số tiền, hoặc các tiêu chí khác để tối ưu hóa việc phân tích và quản lý dữ liệu.

Ưu điểm

- Tốc độ sắp xếp nhanh.
- Được sử dụng trong thư viện của các ngôn ngữ như c++,java,...

Nhược điểm

- Phụ thuộc vào cách chịn phần tử chốt.
- Không ổn định.

2.5. Bài tập

Đề bài: Nhóm sinh viên lập trình giải quyết bài toán sắp xếp một tập hợp các sinh viên được nhập đủ thông tin từ file chứa tối thiểu 10 sinh viên. Mỗi sinh viên gồm có các thông tin: mã sinh viên, họ tên, năm sinh, điểm trung bình. Nhóm sinh viên lập trình giải quyết bài toán sắp xếp các sinh viên theo tên sinh viên theo các thuật toán sau:

- Sắp xếp chọn
- Sắp xếp trộn
- Sắp xếp nhanh

Bài làm:

```cpp
#include <iostream>
#include <string>
using namespace std;
const int MAX_SINH_VIEN = 100;
struct SinhVien {
    int MaSV;
    string HoTen;
    int NamSinh;
    float DiemTrungBinh;
};
void NhapThongTin(SinhVien danhSach[], int& n) {
    cout << "Nhap vao so luong sinh vien (toi da " << MAX_SINH_VIEN << "): ";
    cin >> n;
    if (n > MAX_SINH_VIEN) {
        cout << "So luong sinh vien vuot qua gioi han! Gan max la " << MAX_SINH_VIEN << ".\n";
        n = MAX_SINH_VIEN;
    }

    for (int i = 0; i < n; i++) {
        cout << "Nhap vao ma sinh vien thu " << (i + 1) << " : ";
        cin >> danhSach[i].MaSV;

        cout << "Nhap vao ho ten sinh vien thu " << (i + 1) << " : ";
        cin.ignore();
        getline(cin, danhSach[i].HoTen);

        cout << "Nhap vao nam sinh sinh vien thu " << (i + 1) << " : ";
        cin >> danhSach[i].NamSinh;

        cout << "Nhap vao diem trung binh cua sinh vien " << (i + 1) << " : ";
        cin >> danhSach[i].DiemTrungBinh;
    }
}
void HienThiThongTin(const SinhVien danhSach[], int n) {
    cout << "\nDanh sach sinh vien:\n";
    for (int i = 0; i < n; i++) {
        cout << "Ma SV: " << danhSach[i].MaSV << ", Ho Ten: " << danhSach[i].HoTen
             << ", Nam Sinh: " << danhSach[i].NamSinh
             << ", Diem TB: " << danhSach[i].DiemTrungBinh << endl;
    }
}
```

Khai báo thư viện,hằng số và cấu trúc.Nhập thông tin sinh viên, kiểm tra số lượng sinh viên nếu người dùng nhập số lơn hơn hằng số chương trình giới hạn lại mức hoạt động. Dùng cin.ignore() để loại bỏ ký tự xuống dòng dư thừa trước khi nhập chuỗi.Hiển thị thông tin dưới dạng bảng màn hình.

```
42  void SapXepChon(SinhVien danhSach[], int n) {
43      for (int i = 0; i < n - 1; i++) {
44          int maxIndex = i;
45          for (int j = i + 1; j < n; j++) {
46              if (danhSach[j].DiemTrungBinh > danhSach[maxIndex].DiemTrungBinh) {
47                  maxIndex = j;
48              }
49          }
50          swap(danhSach[i], danhSach[maxIndex]);
51      }
52  }
53
54  void Ghep(SinhVien danhSach[], int left, int mid, int right) {
55      int n1 = mid - left + 1;
56      int n2 = right - mid;
57
58      SinhVien leftList[MAX_SINH_VIEN], rightList[MAX_SINH_VIEN];
59
60      for (int i = 0; i < n1; i++)
61          leftList[i] = danhSach[left + i];
62      for (int j = 0; j < n2; j++)
63          rightList[j] = danhSach[mid + 1 + j];
64
65      int i = 0, j = 0, k = left;
66
67      while (i < n1 && j < n2) {
68          if (leftList[i].DiemTrungBinh >= rightList[j].DiemTrungBinh) {
69              danhSach[k++] = leftList[i++];
70          } else {
71              danhSach[k++] = rightList[j++];
72          }
73      }
74
75      while (i < n1) {
76          danhSach[k++] = leftList[i++];
77      }
78
79      while (j < n2) {
80          danhSach[k++] = rightList[j++];
81      }
82  }
83
```

Sắp xếp chọn sẽ sắp xếp sinh viên theo thứ tự giảm dần điểm trung bình, lặp qua từng vị trí trong mảng.Các phần tử có điểm trung bình lớn nhất trong phạm vị từ i đến cuối mảng. Hoán đổi phần tử đó vói phần tử tại vị trí i để đảm bảo phần tử lơn nhất được đưa lên đầu. Sau đó ta dùng hàm ghép để chia mảng thành 2 mảng con. So sánh từng phần tử từ hai mảng con đưa phần tử lớn hơn vào mảng kết quảvaf gộp nốt các phần còn lại.

```
84  void SapXepTron(SinhVien danhSach[], int left, int right) {
85      if (left < right) {
86          int mid = left + (right - left) / 2;
87
88          SapXepTron(danhSach, left, mid);
89          SapXepTron(danhSach, mid + 1, right);
90          Ghep(danhSach, left, mid, right);
91      }
92  }
93
94  int YenCau(SinhVien danhSach[], int left, int right) {
95      SinhVien pivot = danhSach[right];
96      int i = left - 1;
97
98      for (int j = left; j < right; j++) {
99          if (danhSach[j].DiemTrungBinh > pivot.DiemTrungBinh) {
100             i++;
101             swap(danhSach[i], danhSach[j]);
102         }
103     }
104     swap(danhSach[i + 1], danhSach[right]);
105     return i + 1;
106 }
107
108 void SapXepNhanh(SinhVien danhSach[], int left, int right) {
109     if (left < right) {
110         int pi = YenCau(danhSach, left, right);
111         SapXepNhanh(danhSach, left, pi - 1);
112         SapXepNhanh(danhSach, pi + 1, right);
113     }
114 }
115
```

Sắp xếp trộn sẽ sắp xếp danh sách theo thứ tụ giảm dần điểm trung bình bằng thuật toán chia để trị.chia mảng thành hai nửa từ left đến mid và mid +1 đến right.Sắp xếp từng nửa mảng đệ quy rồi gộp hai mảng đã sắp xếp bằng hàm Ghép. Sau đó sẽ khai báo hàm sắp xếp nhanh. Sắp xếpnhanh sẽ sắp xếp theo thứ tự giảm dần điểm trung bìnhbawngf phân chia và chinh phục. Chọn một phần tử pivot rồi phân chia mảng thành hai phần tử. Một là phần bên trái chứa các phần tử pivot lớn hơn, hai là phần bên phải chức các phần tử nhỏ hơn pivot và cuối cùng sẽ gọi đệ quy để sắp xếp hai phần.

```
116 int main() {
117     SinhVien danhSach[MAX_SINH_VIEN];
118     int n;
119 
120     NhapThongTin(danhSach, n);
121 
122     cout << "\nDanh sach sinh vien truoc khi sap xep:\n";
123     HienThiThongTin(danhSach, n);
124 
125     SapXepChon(danhSach, n);
126     cout << "\nDanh sach sinh vien sau khi sap xep theo diem TB (Sap xep chon):\n";
127     HienThiThongTin(danhSach, n);
128 
129     SapXepTron(danhSach, 0, n - 1);
130     cout << "\nDanh sach sinh vien sau khi sap xep theo diem TB (Sap xep tron):\n";
131     HienThiThongTin(danhSach, n);
132 
133     SapXepNhanh(danhSach, 0, n - 1);
134     cout << "\nDanh sach sinh vien sau khi sap xep theo diem TB (Sap xep nhanh):\n";
135     HienThiThongTin(danhSach, n);
136 
137     return 0;
138 }
```

Cuối cùng là tạo hàm để nhập thông tin sinh viên và hiện danh sách lựa chọn của ba thuật toán

2.7. Tổng kết chương

Thuật toán sắp xếp và tìm kiếm với kĩ thuật được sử dụng trong đó được coi là các kĩ thuật cơ sở cho lập trình máy tính. Các thuật toán được xem xét bao gồm các lớp thuật toán đơn giản và cả các thuật toán càu đặt nhưng có thời gian thực hiện tốt ưu.

KẾT LUẬN

Sắp xếp là quá trình bố trí lại các phần tử của 1 tập hợp theo thứ tự nào đó đối với 1 tiêu chí nào đó. Các thuật toán sắp xếp đơn giản dễ dàng trong việc cài đặt nhưng thời gian thực hiện lớn. Các giải thuật sắp xếp phức tạp cài đặt khó khăn hơn nhưng có thời gian chạy nhanh hơn. Các giải thuật sắp xếp đơn gian có thời gian là O(N^2) trong khi các giải thuật sắp xếp phức tạp có thời gian thực hiện là O(NlogN).

Quick Sort là giải thuật sắp xếp dựa trên phương pháp chia để trị. Chia dãy cần sắp thành 2 phần, sau đó thực hiện việc sắp xếp cho mỗi phần độc lập với nhau. Các phần tử trong 1 phần luôn nhỏ hơn 1 giá trị khóa, còn các phần tử trong phần tử trong phần còn lại luôn lơn hơn giá trị khóa.

Merge Sort là phương pháp sắp xếp dựa trên việc trộn 2 danh sách đã sắp thành 1 danh sách cũng được sắp. Quá trình sắp xếp sẽ trộn từng cặp 2 dãy con 1 phần tử kề nhau để tạo thành các dãy con 2 phần tử được sắp. Các dãy con 2 phần tử được sắp này lại được trộn với nhau tạo thành dãy con 4 phần tử được sắp. Quá trình tiếp tục đến khi toàn bộ dãy được sắp.

Selection Sort Phát triển các biến thể của Selection Sort để giảm bớt số lần so sánh và hoán đổi. Sử dụng AI và machine learning để tự động chọn thuật toán sắp xếp phù hợp cho các tập dữ liệu khác nhau, bao gồm cả Selection Sort. Tiếp tục sử dụng Selection Sort trong các hệ thống nhúng và thiết bị hạn chế tài nguyên vì tính đơn giản và ít tốn kém tài nguyên của nó.

Quick Sort nghiên cứu và phát triển các chiến lược chọn pivot hiệu quả hơn để tránh trường hợp xấu nhất (O(n^2)). Phát triển các phiên bản Quick Sort chạy song song trên các hệ thống đa xử lý để tăng hiệu suất. Sử dụng Quick Sort trong các hệ thống xử lý dữ liệu lớn và công nghệ thông tin để sắp xếp và truy xuất dữ liệu hiệu quả.

Merge Sort Phát triển các phiên bản Merge Sort chạy song song để tối ưu hóa hiệu suất trên các hệ thống đa xử lý. Ứng dụng Merge Sort trong các hệ thống cần xử

lý dữ liệu lớn mà không thể lưu trữ hoàn toàn trong bộ nhớ RAM. Nghiên cứu và phát triển các kỹ thuật sử dụng bộ nhớ hiệu quả hơn trong quá trình hợp nhất (merge) để giảm thiểu yêu cầu về bộ nhớ.

DANH MỤC TÀI LIỆU THAM KHẢO

[1]. Trần Thông Quế, *Cấu trúc dữ liệu và thuật toán (phân tích và cài đặt trên C/C++)*, Tập 1, Nhà xuất bản Thông tin và Truyền thông, 2017.

[2]. Wisnu Anggoro, *C++ Data Structures and Algorithms*, Packt Publishing,2018.