**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

*Trà Vinh, ngày ….. tháng …… năm ……*

**Giáo viên hướng dẫn**

*(Ký tên và ghi rõ họ tên)*

*Trà Vinh, ngày ….. tháng …… năm ……*

**Giáo viên hướng dẫn**

*(Ký tên và ghi rõ họ tên)*

# LỜI CẢM ƠN

**NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày ….. tháng …… năm ……*

**Thành viên hội đồng**

*(Ký tên và ghi rõ họ tên)*

Đầu tiên em xin cảm ơn thầy Lê Minh Tự đã cho em chọn đề tài này, đề tài này giúp em rèn luyện rất nhiều kỹ năng mà hong có được, qua đề tài này đã giúp em hiểu sâu hơn về các thuật toán và danh sách liên kết đơn

# MỤC LỤC

[LỜI CẢM ƠN 2](#_Toc16549)

[MỤC LỤC 4](#_Toc3487)

[DANH MỤC HÌNH ẢNH – BẢNG BIỂU 5](#_Toc111)

[TÓM TẮT NIÊN LUẬN/ĐỒ ÁN CƠ SỞ NGÀNH/ĐỒ ÁN CHUYÊN NGÀNH 6](#_Toc24883)

[MỞ ĐẦU 7](#_Toc32766)

[CHƯƠNG 1: TỔNG QUAN 8](#_Toc23115)

[CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT 11](#_Toc31112)

[2.1 Tổng quan về thuật toán sắp xếp 11](#_Toc23252)

[2.1.1 Tổng quan 11](#_Toc23953)

[2.1.2 Ưu điểm và nhược điểm 13](#_Toc30313)

[2.1.3 Cách thức hoạt động 14](#_Toc18918)

[2.2 Tổng quan về danh sách liên kết đơn 16](#_Toc12753)

[2.2.1 Tổng quan 16](#_Toc23961)

[2.2.2 Ưu điểm và nhược điểm 16](#_Toc20379)

[2.2.3 Cách thức hoạt động 17](#_Toc31158)

[2.3 Tổng quan về C++ 18](#_Toc6769)

[2.3.1 Tổng quan 18](#_Toc15561)

[2.3.2 Ưu và nhược điểm 19](#_Toc25363)

[2.3.3 Cách thức hoạt động 21](#_Toc14192)

[CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU 22](#_Toc2694)

[3.1 Các bước thực hiện 22](#_Toc16120)

[3.1.1 Áp dụng QuickSort, BubbleSort, HeapSort vào danh sách liên kết đơn 22](#_Toc25519)

[CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU 25](#_Toc25807)

[CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 28](#_Toc13139)

[DANH MỤC TÀI LIỆU THAM KHẢO 29](#_Toc13068)

[PHỤ LỤC 30](#_Toc3904)

# DANH MỤC HÌNH ẢNH – BẢNG BIỂU

*[Bảng](#_Toc16971)* [1](#_Toc16971) *[: Danh sách ban đầu](#_Toc16971)* [26](#_Toc16971)

*[Bảng](#_Toc8737)* [2](#_Toc8737) *[: Danh sách sắp xếp theo BubbleSort](#_Toc8737)* [26](#_Toc8737)

*[Bảng](#_Toc18441)* [3](#_Toc18441) *[: Danh sách sắp xếp theo QuickSort](#_Toc18441)* [27](#_Toc18441)

*[Bảng](#_Toc18419)* [4](#_Toc18419) *[: Danh sách sắp xếp theo HeapSort](#_Toc18419)* [27](#_Toc18419)

*[Hình](#_Toc32335)* [1](#_Toc32335) *[: Tổng hợp các thuật toán sắp xếp](#_Toc32335)* [8](#_Toc32335)

*[Hình](#_Toc21865)* [2](#_Toc21865) *[: Sắp Xếp Chèn](#_Toc21865)* [11](#_Toc21865)

*[Hình](#_Toc23926)* [3](#_Toc23926) *[: Sắp xếp chọn](#_Toc23926)* [11](#_Toc23926)

*[Hình](#_Toc17844)* [4](#_Toc17844) *[: Sắp xếp nổi bọt](#_Toc17844)* [12](#_Toc17844)

*[Hình](#_Toc10307)* [5](#_Toc10307) *[: Khai báo thư viện](#_Toc10307)* [22](#_Toc10307)

*[Hình](#_Toc2429)* [6](#_Toc2429) *[: Áp dụng BubbleSort](#_Toc2429)* [22](#_Toc2429)

*[Hình](#_Toc4808)* [7](#_Toc4808) *[: Các mã lệnh cần thiết](#_Toc4808)* [23](#_Toc4808)

*[Hình](#_Toc27927)* [8](#_Toc27927) *[: Áp dụng QuickSort](#_Toc27927)* [23](#_Toc27927)

*[Hình](#_Toc135)* [9](#_Toc135) *[: Áp dụng HeapSort](#_Toc135)* [23](#_Toc135)

*[Hình](#_Toc18860)* [10](#_Toc18860) *[: Mã lệnh tích hợp HeapSort](#_Toc18860)* [24](#_Toc18860)

*[Hình](#_Toc14994)* [11](#_Toc14994) *[: Khai báo ỏ hàm main](#_Toc14994)* [24](#_Toc14994)

*[Hình](#_Toc3199)* [12](#_Toc3199) *[: Kết thúc hàm main](#_Toc3199)* [24](#_Toc3199)

*[Hình](#_Toc20720)* [13](#_Toc20720) *[: Danh sách trước khi sắp xếp](#_Toc20720)* [26](#_Toc20720)

*[Hình](#_Toc17267)* [14](#_Toc17267) *[: Sắp xếp theo thuật toán BubbleSort](#_Toc17267)* [26](#_Toc17267)

*[Hình](#_Toc11075)* [15](#_Toc11075) *[: Sắp xếp theo thuật toán QuickSort](#_Toc11075)* [26](#_Toc11075)

*[Hình](#_Toc21201)* [16](#_Toc21201)  *[Sắp cếp theo thuật toán HeapSort](#_Toc21201)* [27](#_Toc21201)

# TÓM TẮT NIÊN LUẬN/ĐỒ ÁN CƠ SỞ NGÀNH/ĐỒ ÁN CHUYÊN NGÀNH

**Tóm tắt:**

* Nghiên cứu về các thuật toán sắp xếp như: QuickSort, HeapSort, BubbleSort,……
* Áp dụng các thuật toán vào danh sách liên kết đơn

**Cách giải quyết:**

* Sử dụng ngôn ngữ C++
* Áp dụng thuât toán vào danh sách liên kết đơn

**Kết quã đạt được**

* Thuật toán chạy dược và kết quả thực hiện từng bước của phương pháp sắp xếp
* Ấp dụng vào danh sách liên kết đơn chạy và chọn từng chương trình

# MỞ ĐẦU

**Lý do chọn đề tài:**

* Muốn tìm hiều chuyên sâu và nghiên cứu về danh sách liên kết đơn và các thuật toán sắp xếp

**Mục đích:**

* Thực hiện được các phương pháp sắp xếp vào danh sách liên kết đơn

**Đối tượng và phạm vi nghiên cứu:**

* Nghiên cứu lý thuyết: Tìm các thuật toán sắp xếp
* Nghiên cứu thực nghiệm: Xây dựng chương trình demo

# CHƯƠNG 1: TỔNG QUAN

Danh sách liên kết đơn (Singly Linked List) là một cấu trúc dữ liệu linh hoạt, bao gồm một chuỗi các nút, mỗi nút chứa dữ liệu và một liên kết đến nút tiếp theo. Sắp xếp danh sách liên kết đơn là một nhiệm vụ quan trọng và thách thức, bởi vì nó khác biệt so với sắp xếp mảng do tính chất của các liên kết giữa các nút.

Ứng dụng các phương pháp sắp xếp trên danh sách liên kết đơn (singly linked list) là một chủ đề quan trọng trong lĩnh vực cấu trúc dữ liệu và thuật toán. Trong một danh sách liên kết đơn, mỗi nút chứa dữ liệu và một liên kết (thường là một con trỏ) đến nút tiếp theo trong danh sách. Việc sắp xếp một danh sách liên kết đơn đòi hỏi các phương pháp tiếp cận khác biệt so với sắp xếp mảng hay danh sách.

Đây là một chuỗi các "nút" (nodes), trong đó mỗi nút chứa dữ liệu và một liên kết (thường là một con trỏ) đến nút tiếp theo trong chuỗi.

**Danh Sách Liên Kết Đơn**

Cấu Trúc: Mỗi nút trong danh sách liên kết đơn chứa dữ liệu và một liên kết đến nút tiếp theo. Danh sách kết thúc bằng một nút liên kết đến null.

Ưu Điểm: Linh hoạt trong quản lý bộ nhớ, hiệu quả trong việc chèn và xóa các nút.

Nhược Điểm: Truy cập tuần tự, không hỗ trợ truy cập ngẫu nhiên nhanh như mảng.

Phương Pháp Sắp Xếp



*Hình 1**: Tổng hợp các thuật toán sắp xếp*

**Sắp Xếp Chèn (Insertion Sort):**

* Mô tả: Chèn từng nút vào vị trí thích hợp trong danh sách đã sắp xếp.
* Hiệu quả: Tốt với danh sách nhỏ, nhưng không hiệu quả với danh sách lớn.

**Sắp Xếp Chọn (Selection Sort):**

* Mô tả: Tìm nút nhỏ nhất trong phần chưa sắp xếp và chuyển nó lên đầu danh sách.
* Hiệu quả: Đơn giản nhưng không hiệu quả với danh sách lớn.

**Sắp Xếp Nổi Bọt (Bubble Sort):**

* Mô tả: So sánh và hoán đổi các nút liên tiếp nếu không đúng thứ tự.
* Hiệu quả: Đơn giản nhưng hiệu suất thấp, thích hợp cho danh sách nhỏ.

**Sắp Xếp Nhanh (Quick Sort):**

* Mô tả: Sử dụng phương pháp chia để trị, chọn một nút làm pivot và phân loại các nút khác.
* Hiệu quả: Hiệu quả cao với danh sách lớn, nhưng phức tạp hơn.

**Sắp Xếp Hợp Nhất (Merge Sort):**

* Mô tả: Chia danh sách thành các phần nhỏ, sắp xếp và hợp nhất chúng.
* Hiệu quả: Hiệu suất tốt, ổn định và không phụ thuộc vào dữ liệu đầu vào.

**Sắp Xếp Xếp Đóng (HeapSort):**

* Mô tả: Sử dụng cấu trúc dữ liệu heap để sắp xếp.
* Hiệu quả: Hiệu suất tốt, ổn định và không phụ thuộc vào dữ liệu đầu vào.

**Lựa Chọn Phương Pháp Sắp Xếp**

**Kích Thước Dữ Liệu:** Đối với dữ liệu lớn, nên chọn các thuật toán có độ phức tạp thời gian thấp như Merge Sort, Quick Sort hoặc HeapSort.

**Yêu Cầu Về Ổn Định:** Nếu cần bảo toàn thứ tự tương đối của các phần tử giống nhau, chọn thuật toán ổn định như Merge Sort.

**Bộ Nhớ và Môi Trường:** Trong môi trường có hạn chế về bộ nhớ, các thuật toán không cần bộ nhớ phụ như Quick Sort hoặc HeapSort có thể phù hợp hơn.

**Tính Đơn Giản và Dễ Hiểu:** Đối với các ứng dụng đơn giản hoặc giáo dục, Insertion Sort hoặc Bubble Sort có thể là lựa chọn tốt do tính đơn giản và dễ hiểu.

Để lựa chọn phương pháp sắp xếp phụ thuộc vào yếu tố như kích thước danh sách, yêu cầu về hiệu suất và độ phức tạp mong muốn. Sắp xếp chèn và nổi bọt thích hợp cho các danh sách nhỏ, trong khi sắp xếp nhanh và hợp nhất phù hợp hơn với danh sách lớn và yêu cầu hiệu suất cao. Sắp xếp chọn, mặc dù đơn giản nhưng thường không hiệu quả so với các phương pháp khác.

Tóm lại, danh sách liên kết đơn là một cấu trúc dữ liệu linh hoạt và hữu ích, phù hợp cho nhiều loại ứng dụng, đặc biệt là khi kích thước dữ liệu không cố định hoặc cần thực hiện các thao tác chèn và xóa hiệu quả.

Qua phần tổng quan về danh sách liên kết đơn và các phương pháp sắp xếp thì em đã chọn ra những phương pháp: QuickSort, BubbleSort, HeapSort để tiến hành vào quá trình nghiên cứu.

# **CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT**

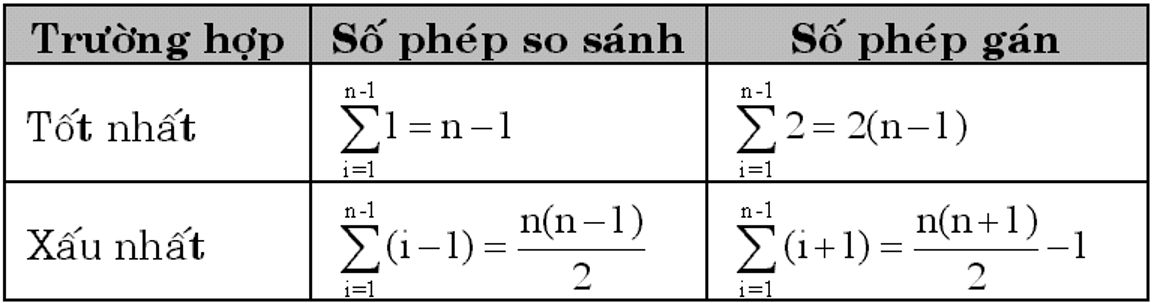
## Tổng quan về thuật toán sắp xếp

### Tổng quan

Thuật toán sắp xếp là một phần quan trọng trong lĩnh vực khoa học máy tính và kỹ thuật phần mềm, với nhiều ứng dụng từ cơ sở dữ liệu đến giao diện người dùng. Các thuật toán sắp xếp có mục đích sắp xếp một dãy số liệu theo một trật tự nhất định (thường là tăng dần hoặc giảm dần). Dưới đây là tổng quan về một số thuật toán sắp xếp phổ biến:

**Sắp Xếp Chèn (Insertion Sort):**

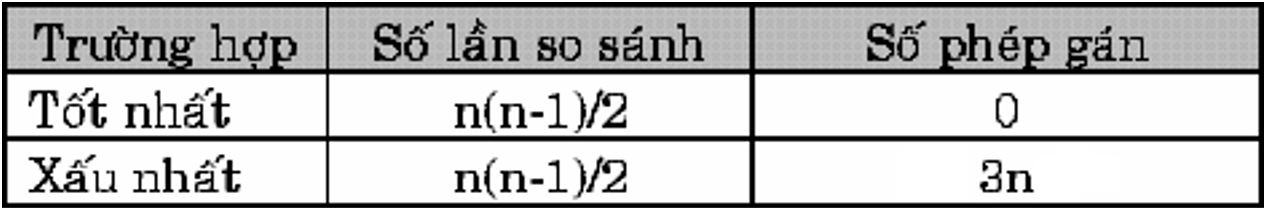
* Mô tả: Chèn từng phần tử vào vị trí phù hợp.
* Độ phức tạp: O(n^2) trong trường hợp xấu nhất.
* Ứng dụng: Hiệu quả với dữ liệu nhỏ, gần như đã được sắp xếp.



*Hình 2**: Sắp Xếp Chèn*

**Sắp Xếp Chọn (Selection Sort)**

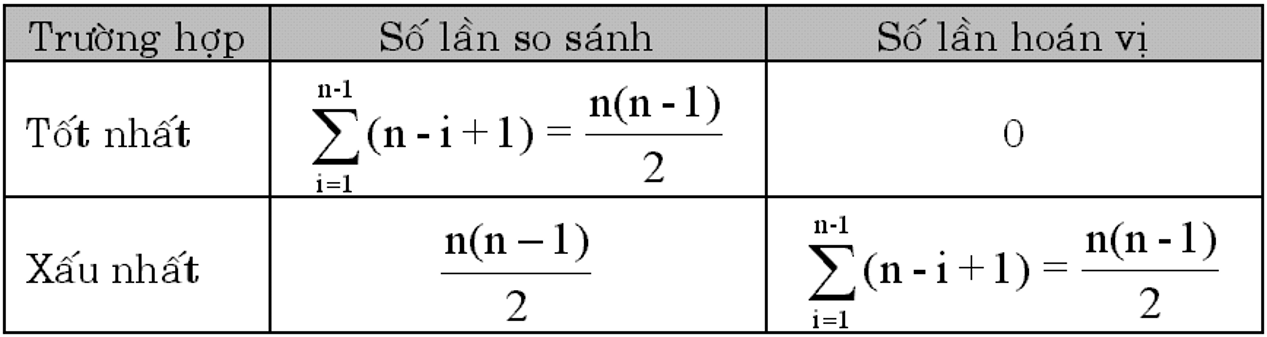
* Mô tả: Tìm phần tử nhỏ nhất (hoặc lớn nhất) và đặt nó vào đầu danh sách.
* Độ phức tạp: Luôn là O(n^2), không phụ thuộc vào dữ liệu đầu vào.
* Ứng dụng: Đơn giản nhưng không hiệu quả cho dữ liệu lớn.



*Hình 3**: Sắp xếp chọn*

**Sắp Xếp Nổi Bọt (Bubble Sort)**

* Mô tả: So sánh các cặp phần tử liền kề và hoán đổi nếu không đúng thứ tự.
* Độ phức tạp: O(n^2) trong trường hợp xấu nhất.
* Ứng dụng: Dễ hiểu, nhưng ít hiệu quả.



*Hình 4**: Sắp xếp nổi bọt*

**Sắp Xếp Hợp Nhất (Merge Sort)**

* Mô tả: Chia danh sách thành nửa, sắp xếp từng nửa và hợp nhất chúng.
* Độ phức tạp: O(n log n) trong mọi trường hợp.
* Ứng dụng: Hiệu quả và ổn định, tốt cho dữ liệu lớn.

**Sắp Xếp Nhanh (Quick Sort)**

* Mô tả: Chọn một phần tử làm chốt và phân loại các phần tử khác tương đối với chốt.
* Độ phức tạp: O(n log n) trung bình, nhưng O(n^2) trong trường hợp xấu nhất.
* Ứng dụng: Nhanh và hiệu quả, phổ biến nhất.

**Sắp Xếp Heap (Heap Sort)**

* Mô tả: Sử dụng cấu trúc dữ liệu heap để sắp xếp.
* Độ phức tạp: O(n log n) trong mọi trường hợp.
* Ứng dụng: Hiệu quả cho dữ liệu lớn, không cần bộ nhớ phụ.

**Các Yếu Tố Cần Xem Xét Khi Chọn Thuật Toán:**

* Kích Thước Dữ Liệu: Đối với dữ liệu lớn, chọn thuật toán có độ phức tạp thời gian thấp.
* Ổn Định: Cần giữ nguyên thứ tự của các phần tử giống nhau không?
* Bộ Nhớ: Có hạn chế về không gian bộ nhớ không?
* Đơn Giản và Dễ Hiểu: Đối với việc giảng dạy hoặc khi không yêu cầu hiệu suất cao.

Thuật toán sắp xếp được chọn phụ thuộc vào nhu cầu cụ thể của ứng dụng và các hạn chế về môi trường và nguồn lực.

### Ưu điểm và nhược điểm

Thuật toán sắp xếp mỗi loại đều có ưu và nhược điểm riêng, phù hợp với các tình huống và yêu cầu sử dụng khác nhau. Dưới đây là ưu và nhược điểm của một số thuật toán sắp xếp phổ biến:

**Sắp Xếp Chèn (Insertion Sort)**

**Ưu điểm:**

* Hiệu quả với danh sách nhỏ hoặc gần như đã sắp xếp.
* Đơn giản và dễ cài đặt.
* Không cần không gian bổ sung (in-place sorting).
* Ổn định (bảo toàn thứ tự của các phần tử giống nhau).

**Nhược điểm:**

* Không hiệu quả với danh sách lớn.
* Độ phức tạp O(n^2) khiến nó kém hiệu quả khi dữ liệu tăng lên.

**Sắp Xếp Chọn (Selection Sort)**

**Ưu điểm:**

* Dễ hiểu và cài đặt.
* Không cần không gian bổ sung.
* Hiệu suất không thay đổi với dữ liệu đầu vào.

**Nhược điểm:**

* Không hiệu quả với dữ liệu lớn (O(n^2)).
* Không ổn định.

**Sắp Xếp Nổi Bọt (Bubble Sort)**

**Ưu điểm:**

* Cực kỳ đơn giản để hiểu và cài đặt.
* Ổn định.

**Nhược điểm:**

* Không hiệu quả với danh sách lớn (O(n^2)).
* Thường bị thay thế bởi thuật toán sắp xếp khác hiệu quả hơn.

**Sắp Xếp Hợp Nhất (Merge Sort)**

**Ưu điểm:**

* Hiệu suất ổn định O(n log n).
* Hiệu quả với danh sách lớn.
* Ổn định.

**Nhược điểm:**

* Cần không gian bổ sung.
* Có thể không hiệu quả cho danh sách nhỏ.

**Sắp Xếp Nhanh (Quick Sort)**

**Ưu điểm:**

* Hiệu suất cao trung bình O(n log n).
* Không cần không gian bổ sung.
* Linh hoạt với lựa chọn pivot.

**Nhược điểm:**

* Không ổn định.
* Có thể kém hiệu quả (O(n^2)) với dữ liệu đặc biệt.

**Sắp Xếp Heap (Heap Sort)**

**Ưu điểm:**

* Hiệu suất ổn định O(n log n).
* Không cần không gian bổ sung.
* Hiệu quả cho dữ liệu lớn.

**Nhược điểm:**

* Không ổn định.
* Có thể kém hiệu quả hơn so với Quick Sort trong một số trường hợp.

### Cách thức hoạt động

Mỗi thuật toán sắp xếp có cách thức hoạt động riêng biệt, phản ánh cách tiếp cận khác nhau để sắp xếp dữ liệu. Dưới đây là cách thức hoạt động của một số thuật toán sắp xếp phổ biến:

**Sắp Xếp Chèn (Insertion Sort)**

* Bắt đầu từ phần tử thứ hai của danh sách.
* So sánh phần tử hiện tại với các phần tử trước đó và chèn nó vào vị trí thích hợp sao cho phần danh sách đã xét luôn ở trạng thái sắp xếp.
* Lặp lại quá trình cho mỗi phần tử.

**Sắp Xếp Chọn (Selection Sort)**

* Tìm phần tử nhỏ nhất (hoặc lớn nhất) trong danh sách.
* Hoán đổi phần tử này với phần tử đầu tiên.
* Lặp lại quá trình với phần còn lại của danh sách, bỏ qua phần tử đầu tiên đã sắp xếp.

**Sắp Xếp Nổi Bọt (Bubble Sort)**

* Lặp qua danh sách, so sánh các cặp phần tử liên tiếp.
* Hoán đổi chúng nếu chúng không theo thứ tự mong muốn (ví dụ: phần tử trước lớn hơn phần tử sau khi sắp xếp tăng dần).
* Tiếp tục quá trình cho đến khi không cần hoán đổi thêm, có nghĩa là danh sách đã được sắp xếp.

**Sắp Xếp Hợp Nhất (Merge Sort)**

* Chia danh sách thành nửa đến khi mỗi phần chỉ còn một phần tử.
* Hợp nhất các phần này một cách có thứ tự, kết hợp từng cặp danh sách con đã sắp xếp thành một danh sách lớn hơn.
* Tiếp tục quá trình hợp nhất cho đến khi tất cả các phần tử được kết hợp thành một danh sách hoàn chỉnh.

**Sắp Xếp Nhanh (Quick Sort)**

* Chọn một phần tử làm chốt (pivot).
* Phân loại các phần tử còn lại thành hai nhóm: những phần tử nhỏ hơn chốt và những phần tử lớn hơn chốt.
* Áp dụng đệ quy cho cả hai nhóm và chốt.
* Lặp lại quá trình cho đến khi mỗi phần chỉ chứa một phần tử.

**Sắp Xếp Heap (Heap Sort)**

* Xây dựng một heap từ dữ liệu đầu vào.
* Lặp lại quá trình loại bỏ phần tử đỉnh của heap (là phần tử lớn nhất trong max-heap hoặc nhỏ nhất trong min-heap) và tái cấu trúc heap.
* Sau khi loại bỏ hết các phần tử, danh sách sẽ được sắp xếp.

Mỗi thuật toán sắp xếp sử dụng một chiến lược riêng biệt, từ đơn giản đến phức tạp, và chúng có hiệu suất khác nhau tùy thuộc vào kích thước và loại dữ liệu đầu vào.

## Tổng quan về danh sách liên kết đơn

### Tổng quan

Danh sách liên kết đơn (Single Linked List) là một cấu trúc dữ liệu cơ bản trong lĩnh vực khoa học máy tính. Nó bao gồm một chuỗi các nút, trong đó mỗi nút chứa dữ liệu và một liên kết (thường là một con trỏ) tới nút tiếp theo trong chuỗi. Đây là những đặc điểm nổi bật của danh sách liên kết đơn:

**Cấu trúc Nút (Node Structure):** Mỗi nút trong danh sách liên kết đơn bao gồm hai phần:

**Dữ liệu (Data):** Phần này chứa thông tin hoặc giá trị mà nút đang lưu trữ.

**Con trỏ (Pointer):** Phần này chứa địa chỉ của nút tiếp theo trong danh sách. Trong nút cuối cùng của danh sách, con trỏ này sẽ chỉ đến null hoặc không có giá trị, biểu thị rằng đó là nút cuối của danh sách.

**Thêm (Insertion):** Có thể thêm một nút mới vào đầu danh sách, cuối danh sách, hoặc giữa danh sách bằng cách điều chỉnh các liên kết của nút liền kề.

**Xóa (Deletion):** Có thể loại bỏ một nút khỏi danh sách bằng cách thay đổi liên kết của nút trước đó để trỏ tới nút tiếp theo của nút bị xóa.

**Tìm kiếm (Search):** Có thể tìm kiếm một nút trong danh sách bằng cách duyệt qua từng nút và so sánh dữ liệu của chúng với dữ liệu cần tìm.

Danh sách liên kết đơn là một công cụ hữu ích trong nhiều tình huống như quản lý bộ nhớ động, triển khai các cấu trúc dữ liệu khác (như hàng đợi và ngăn xếp), và trong các thuật toán như sắp xếp và tìm kiếm.

### Ưu điểm và nhược điểm

Danh sách liên kết đơn (Single Linked List) có cả ưu và nhược điểm khi so sánh với các cấu trúc dữ liệu khác như mảng. Dưới đây là chi tiết về các ưu điểm và nhược điểm của nó:

**Ưu điểm:**

**Động và Linh hoạt trong Quản lý Bộ nhớ:** Không giống như mảng, danh sách liên kết đơn không yêu cầu kích thước cố định hoặc cấp phát bộ nhớ trước. Bạn có thể thêm hoặc xóa các nút một cách linh hoạt mà không cần lo lắng về kích thước.

**Chèn và Xóa Hiệu quả:** Thêm hoặc xóa một nút trong danh sách liên kết đơn chỉ cần thay đổi liên kết của nút liền kề, điều này nhanh hơn nhiều so với việc phải dịch chuyển các phần tử như trong mảng.

**Không Cần Phải Di chuyển Phần tử:** Khi thêm hoặc xóa các phần tử, không cần di chuyển các phần tử khác (trừ khi cần di chuyển qua các nút để tìm vị trí chèn hoặc xóa).

**Phù hợp cho Kích thước không Xác định hoặc Thay đổi:** Đối với các trường hợp mà số lượng phần tử không thể xác định trước hoặc thay đổi thường xuyên, danh sách liên kết đơn là lựa chọn tốt.

**Nhược điểm:**

**Truy cập Tuyến tính:** Khác với mảng cho phép truy cập ngẫu nhiên, danh sách liên kết đơn yêu cầu duyệt qua từng nút để tìm một phần tử cụ thể, điều này có thể mất nhiều thời gian với danh sách dài.

**Tiêu tốn Bộ nhớ Hơn:** Mỗi nút trong danh sách liên kết đơn không chỉ chứa dữ liệu mà còn có thêm bộ nhớ cho con trỏ, điều này nghĩa là nó tiêu tốn bộ nhớ hơn so với mảng.

**Không Có Lợi Ích từ Bộ nhớ Cache:** Do các nút không nhất thiết phải được lưu trữ liên tục trong bộ nhớ, danh sách liên kết thường không tận dụng được hiệu quả từ bộ nhớ cache như mảng.

**Phức tạp hơn trong Triển khai:** Việc lập trình và quản lý danh sách liên kết đơn thường phức tạp hơn so với việc sử dụng mảng đơn giản.

Như vậy, việc chọn sử dụng danh sách liên kết đơn hay một cấu trúc dữ liệu khác phụ thuộc vào yêu cầu cụ thể của bài toán và ưu tiên về hiệu suất, bộ nhớ, và độ phức tạp của mã nguồn.

### Cách thức hoạt động

Danh sách liên kết đơn hoạt động dựa trên một cấu trúc dữ liệu linh hoạt, cho phép lưu trữ một dãy các phần tử theo một cách không liên tục trong bộ nhớ. Cách thức hoạt động cụ thể của danh sách liên kết đơn bao gồm các khía cạnh sau:

**Cấu Trúc Nút (Node Structure):**

* Mỗi nút trong danh sách liên kết đơn chứa hai thành phần chính: dữ liệu và một con trỏ.

**Truy cập và Duyệt qua Danh sách:**

* Để truy cập hoặc duyệt qua danh sách, bạn bắt đầu từ nút đầu tiên (được gọi là "đầu" của danh sách).
* Tiếp tục di chuyển từ nút này sang nút kế tiếp bằng cách sử dụng con trỏ cho đến khi bạn tìm thấy nút cần thiết hoặc đến nút cuối cùng (null).

**Thêm Nút (Insertion):**

* Thêm vào đầu danh sách: Tạo một nút mới và làm cho nó trỏ đến nút hiện tại đầu tiên, sau đó cập nhật "đầu" của danh sách thành nút mới này.
* Thêm vào giữa danh sách: Tìm nút trước nút bạn muốn chèn, cập nhật con trỏ của nó để trỏ tới nút mới, và làm cho nút mới trỏ đến nút tiếp theo.
* Thêm vào cuối danh sách: Duyệt qua danh sách cho đến nút cuối cùng, sau đó làm cho con trỏ của nút cuối cùng trỏ đến nút mới này.

**Xóa Nút (Deletion):**

* Để xóa một nút, tìm nút trước nút cần xóa và cập nhật con trỏ của nó để bỏ qua nút cần xóa và trỏ đến nút tiếp theo của nút cần xóa.

**Hiệu Quả và Mục Đích Sử Dụng:**

* Danh sách liên kết đơn cung cấp sự linh hoạt trong quản lý bộ nhớ và thích hợp cho các ứng dụng cần thêm hoặc xóa phần tử một cách thường xuyên.
* Tuy nhiên, việc truy cập phần tử trong danh sách liên kết không nhanh như mảng do cần phải duyệt qua các nút.

Danh sách liên kết đơn là một lựa chọn tốt khi bạn cần một cấu trúc dữ liệu có kích thước thay đổi động và không yêu cầu truy cập ngẫu nhiên nhanh chóng đến các phần tử

## Tổng quan về C++

### Tổng quan

C++ là một ngôn ngữ lập trình được thiết kế bởi Bjarne Stroustrup, ra mắt lần đầu vào năm 1983. Đây là một ngôn ngữ lập trình đa mô hình, hỗ trợ cả lập trình hướng thủ tục, hướng đối tượng, và lập trình chức năng. C++ được xem là một ngôn ngữ lập trình trung cấp, cung cấp sự kết hợp giữa lập trình cấp thấp và cấp cao. Dưới đây là một số điểm nổi bật về C++:

**Đặc Điểm:**

* Hiệu Suất Cao: C++ cung cấp quản lý bộ nhớ thủ công và tính năng cấp thấp, làm cho nó trở nên phù hợp với các ứng dụng cần hiệu suất cao như game, hệ thống nhúng, và phần mềm hệ thống.
* Hướng Đối Tượng: C++ hỗ trợ lập trình hướng đối tượng (OOP), bao gồm các khái niệm như kế thừa, đa hình, đóng gói, và trừu tượng.
* Thư Viện Chuẩn Mạnh Mẽ: C++ đi kèm với Thư viện Chuẩn C++ (STL), bao gồm nhiều chức năng sẵn có như các container, algorithme, và các hàm chuẩn.
* Tương Thích với C: C++ tương thích ngược với C, điều này có nghĩa là hầu hết chương trình C có thể được biên dịch với một trình biên dịch C++.
* Template: C++ cung cấp các template, cho phép lập trình viên viết mã tổng quát mà vẫn duy trì hiệu suất.

**Ứng Dụng:**

* Phát Triển Phần Mềm Hệ Thống và Hệ Điều Hành: Do hiệu suất cao và quyền kiểm soát bộ nhớ.
* Ứng Dụng Máy Tính để Bàn và Game: Cung cấp hiệu suất tốt và hỗ trợ đồ họa.
* Phát Triển Phần Mềm Nhúng và IoT: Kích thước nhỏ gọn và hiệu suất tối ưu.

Công Nghệ Máy Tính và AI: Tính linh hoạt và hiệu suất cao trong xử lý dữ liệu.

**Phiên Bản và Tiêu Chuẩn**

C++ liên tục được cập nhật với các tiêu chuẩn mới, như C++11, C++14, C++17, và gần đây nhất là C++20, mỗi phiên bản mang lại các tính năng và cải tiến mới.

C++ là một ngôn ngữ mạnh mẽ với nhiều ứng dụng, từ phát triển phần mềm cấp cao đến hệ thống cấp thấp. Sự linh hoạt và hiệu suất cao của nó làm cho nó trở thành một lựa chọn phổ biến trong nhiều lĩnh vực công nghệ.

### Ưu và nhược điểm

C++ là một ngôn ngữ lập trình mạnh mẽ và linh hoạt, nhưng cũng có những hạn chế. Dưới đây là một số ưu và nhược điểm chính của C++:

**Ưu Điểm:**

* Hiệu Suất Cao: C++ cung cấp quản lý bộ nhớ thủ công và các tính năng cấp thấp, giúp đạt được hiệu suất cao. Điều này rất quan trọng cho các ứng dụng như trò chơi video, chương trình mô phỏng, và phát triển hệ thống.
* Hướng Đối Tượng: C++ hỗ trợ lập trình hướng đối tượng (OOP), một mô hình phát triển phần mềm hiệu quả giúp tái sử dụng mã và tổ chức chương trình tốt hơn.
* Linh Hoạt: C++ cho phép lập trình cả hướng đối tượng và hướng thủ tục, làm cho nó trở nên linh hoạt và thích hợp cho nhiều loại dự án khác nhau.
* Thư Viện Chuẩn Mạnh Mẽ: C++ đi kèm với Thư Viện Chuẩn C++ (STL), bao gồm các tính năng sẵn có như container, algorithm, và thread.
* Tương Thích với C: C++ tương thích tốt với C, điều này cho phép nó sử dụng các thư viện và hệ thống mã đã được viết bằng C.
* Kiểm Soát Tài Nguyên: C++ cung cấp kiểm soát chi tiết đối với tài nguyên hệ thống, làm cho nó trở thành lựa chọn tốt cho các ứng dụng cần quản lý bộ nhớ và tài nguyên hệ thống một cách tinh vi.

**Nhược Điểm:**

* Quản Lý Bộ Nhớ Thủ Công: Quản lý bộ nhớ trong C++ là thủ công và phức tạp, dễ gây ra lỗi như rò rỉ bộ nhớ và truy cập vùng nhớ không hợp lệ.
* Độ Phức Tạp Cao: C++ là một ngôn ngữ lập trình phức tạp với cú pháp khó hiểu đối với người mới học, và nó yêu cầu một sự hiểu biết sâu sắc về các khái niệm như tham chiếu, con trỏ, và quản lý tài nguyên.
* Không Có Quản Lý Bộ Nhớ Tự Động: Khác với ngôn ngữ như Java hay Python, C++ không có bộ thu gom rác tự động, điều này có nghĩa là lập trình viên phải chịu trách nhiệm quản lý và giải phóng bộ nhớ.
* Rủi Ro An Toàn: C++ cho phép truy cập trực tiếp đến bộ nhớ và các hàm cấp thấp, có thể dẫn đến các lỗ hổng bảo mật nếu không được xử lý cẩn thận.
* Không Có Kiểm Tra Thời Gian Chạy (Runtime Checking): C++ ít có khả năng phát hiện lỗi tại thời điểm chạy chương trình (runtime), điều này có thể dẫn đến lỗi phức tạp.
* Khó Gỡ Lỗi và Bảo Trì: Do tính phức tạp của ngôn ngữ, quá trình gỡ lỗi và bảo trì mã C++ có thể trở nên khó khăn.

### Cách thức hoạt động

C++ hoạt động dựa trên một số nguyên tắc cơ bản và quy trình. Dưới đây là mô tả về cách thức hoạt động của C++ từ quá trình viết mã đến thực thi chương trình:

**Viết Mã:** Lập Trình viên viết mã C++ sử dụng một trình soạn thảo văn bản hoặc một Môi trường Phát triển Tích hợp (IDE)

**Biên Dịch:** Biên dịch là quá trình chuyển đổi mã nguồn C++ thành mã máy, thường là dạng tệp nhị phân hoặc thực thi.

**Liên Kết:** Liên kết là quá trình kết hợp các file mã máy (thường là các file đối tượng) thành một chương trình thực thi hoàn chỉnh.

**Thực Thi:** Sau khi quá trình biên dịch và liên kết hoàn tất, chương trình thực thi có thể được chạy trên máy tính.

**Quản Lý Bộ Nhớ:** Trong C++, lập trình viên có trách nhiệm quản lý bộ nhớ, bao gồm cấp phát và giải phóng bộ nhớ.

**Tính năng Khác:** Lập trình Hướng Đối Tượng: C++ hỗ trợ các tính năng OOP như kế thừa, đa hình, và đóng gói.

**Template:** C++ sử dụng template để cung cấp lập trình tổng quát.

**Exception Handling:** C++ cung cấp cơ chế xử lý ngoại lệ để xử lý các tình huống lỗi.

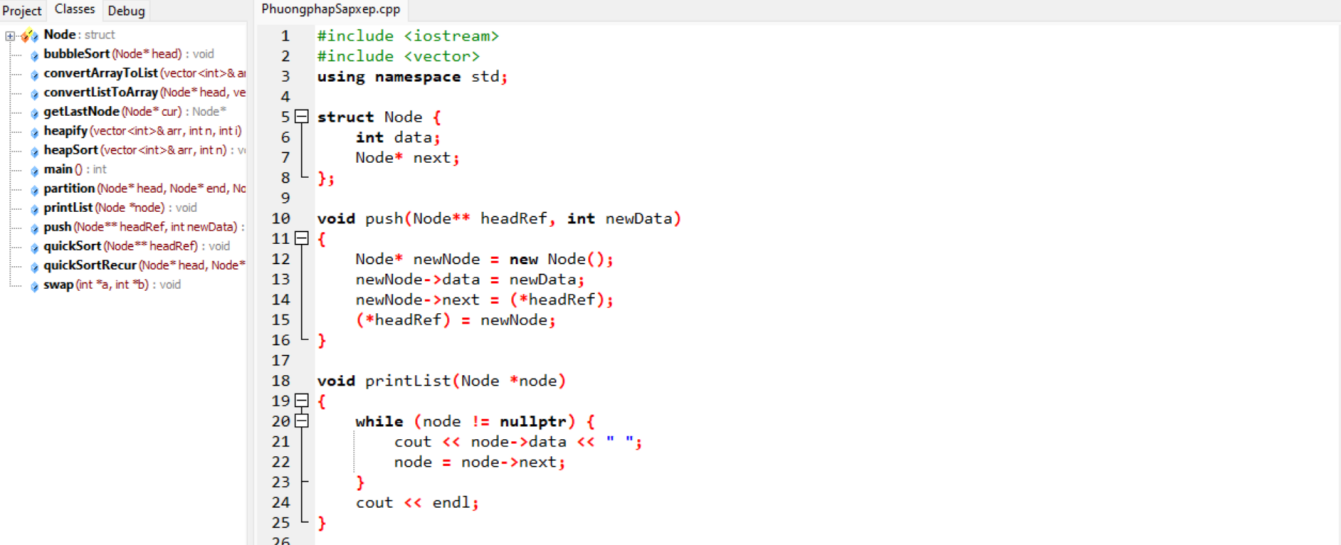
**Tối Ưu Hóa:** C++ cho phép tối ưu hóa mã để cải thiện hiệu suất, bao gồm tối ưu hóa thời gian chạy và bộ nhớ.

Quá trình từ viết mã đến thực thi chương trình trong C++ đòi hỏi một sự hiểu biết về nguyên tắc lập trình, quản lý bộ nhớ, và hoạt động của trình biên dịch. Sự hiểu biết sâu sắc về những nguyên tắc này là cần thiết để viết mã C++ hiệu quả và hiệu suất cao.

# CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU

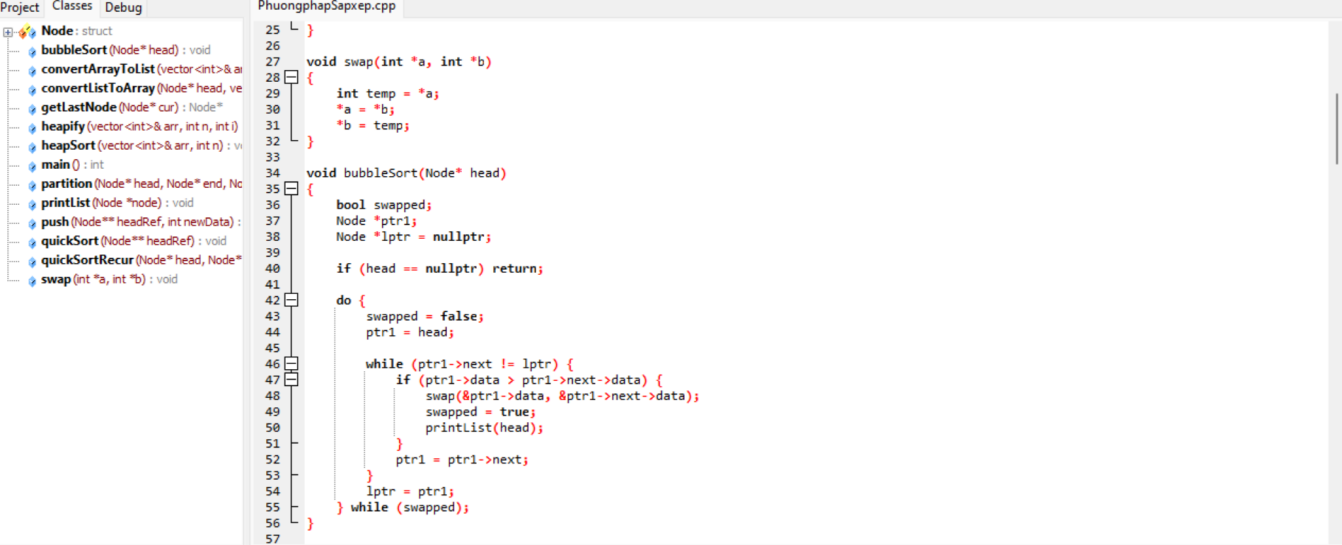
## 3.1 Các bước thực hiện

## 3.1.1 Áp dụng QuickSort, BubbleSort, HeapSort vào danh sách liên kết đơn



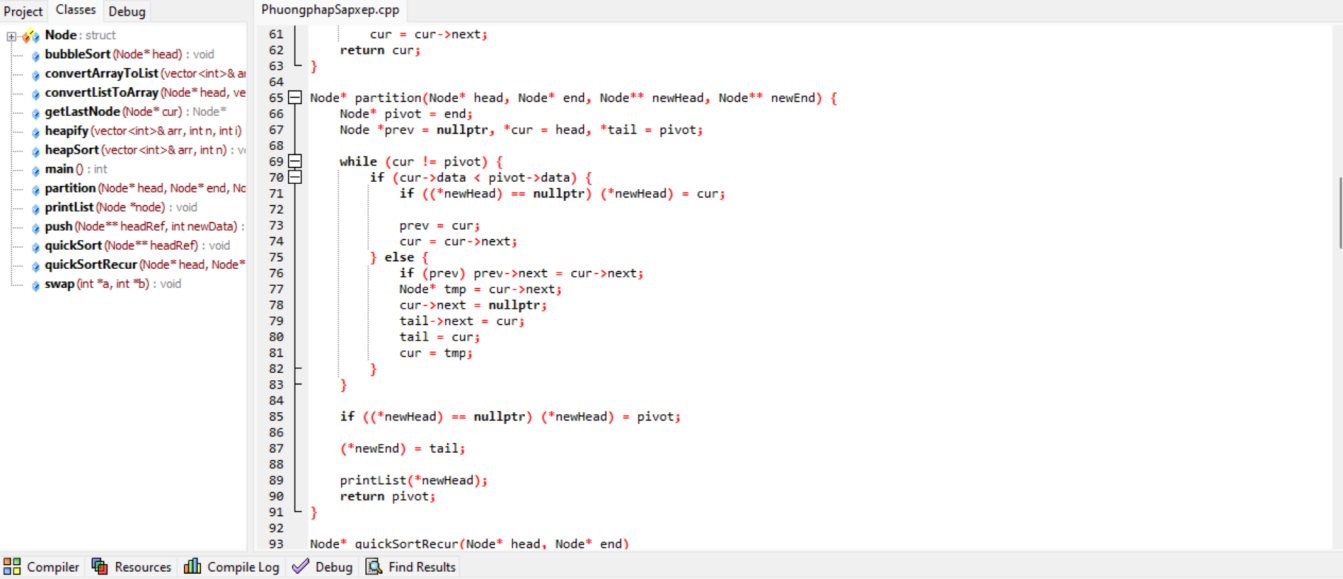
*Hình 5**: Khai báo thư viện*

Sau đó ta áp dụng các thuật toán sắp xếp vào danh sách liên kết đơn



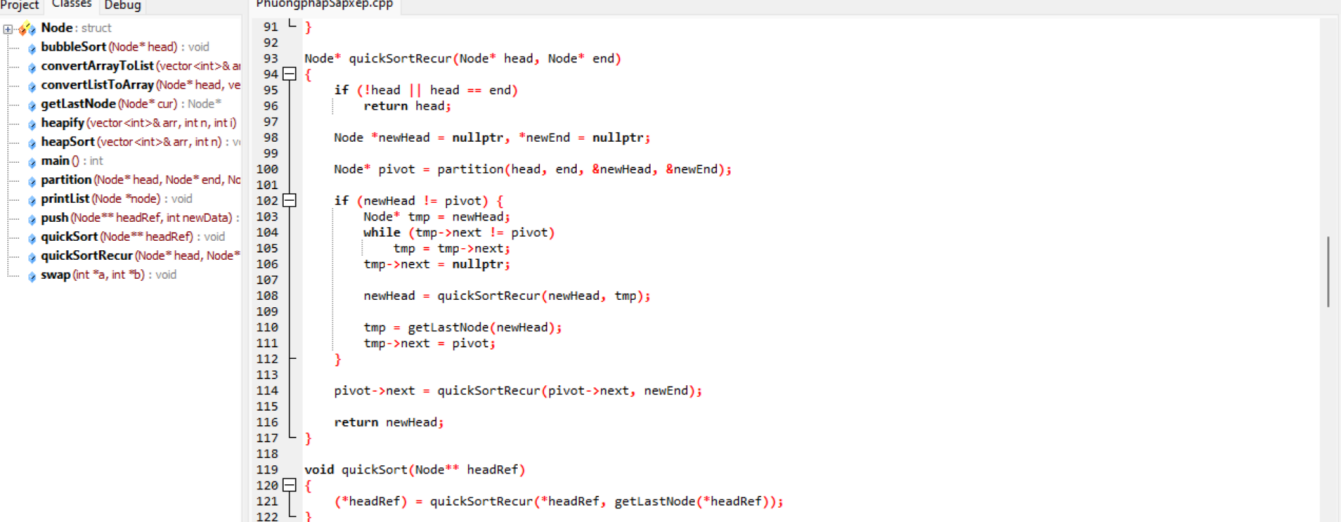
*Hình 6**: Áp dụng BubbleSort*

Các mã lệnh cần thiết dể chương trình có thể được thực thi



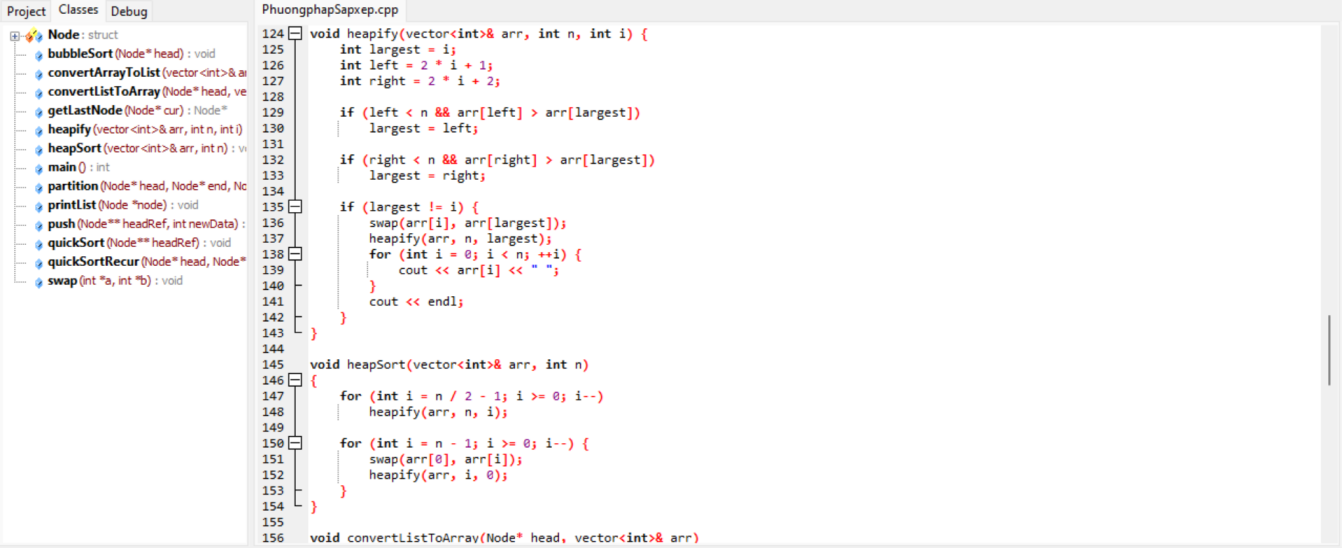
*Hình 7**: Các mã lệnh cần thiết*

Và giờ ta đến QuickSort



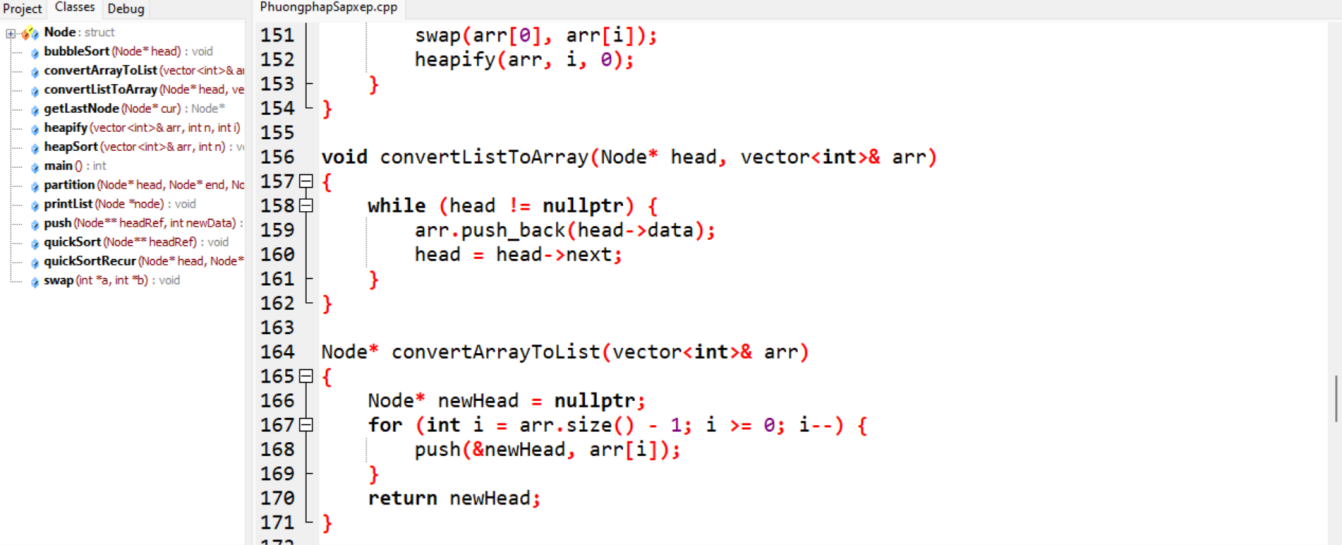
*Hình 8**: Áp dụng QuickSort*

Về thuật toán HeapSort là thuât toán khó áp dụng vào danh sách liên kết đơn nên em khai báo mảng HeapSort ở mảng rồi mới tích hợp vào danh sách liên kết đơn



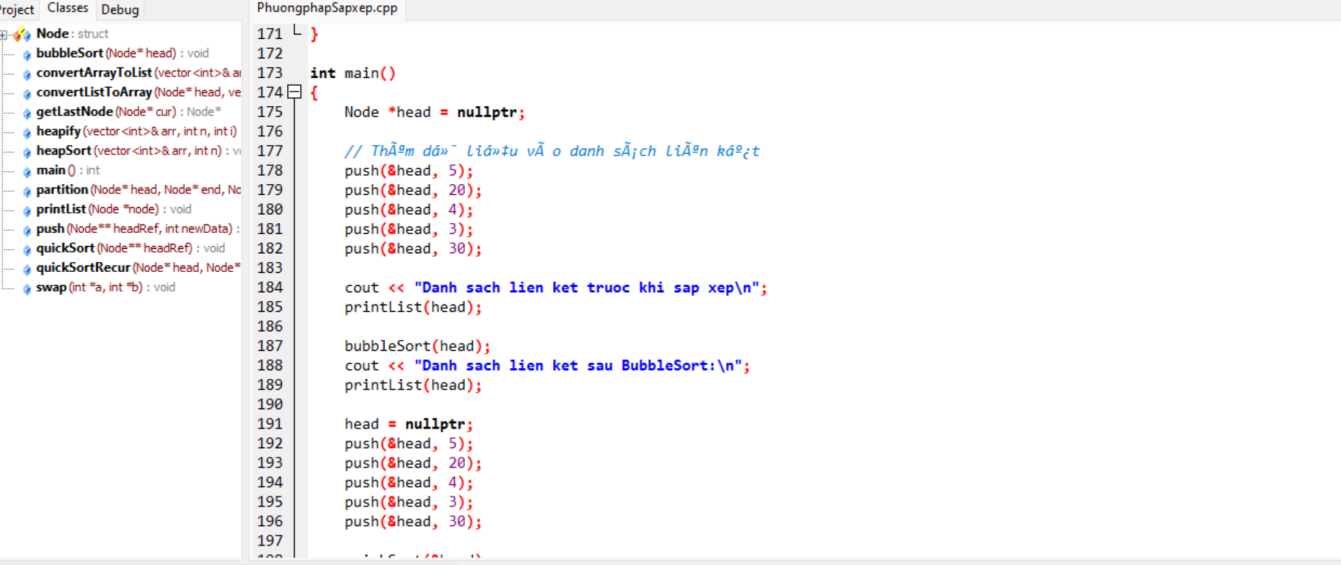
*Hình 9**: Áp dụng HeapSort*

Và đề tích hợp từ bên mảng qua thì càn những mã lệnh sau:

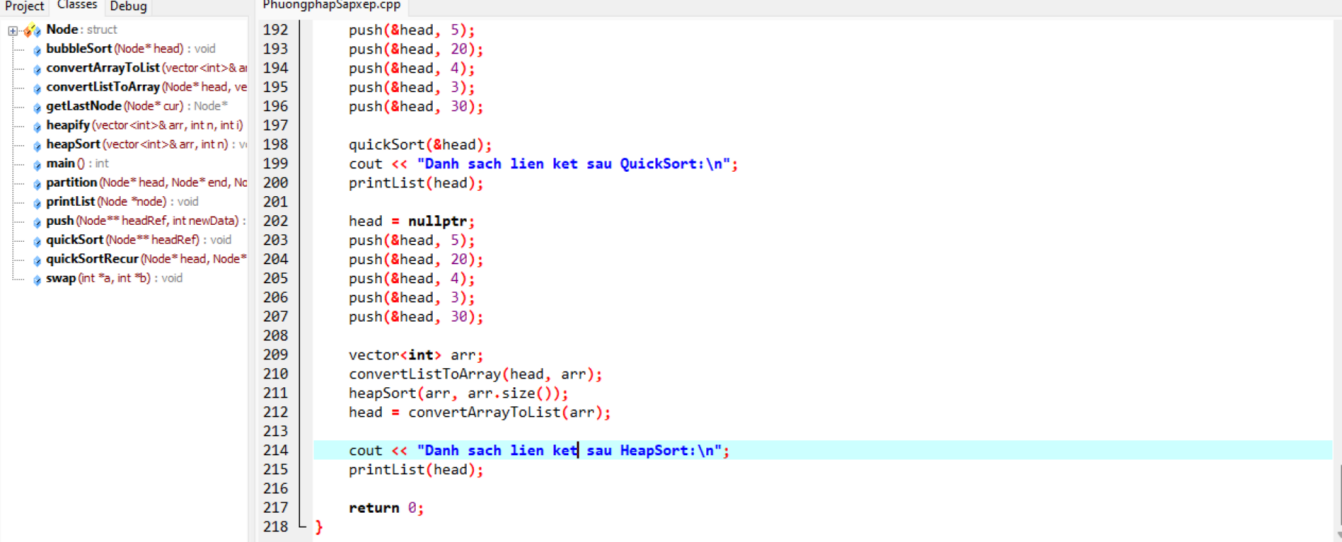


*Hình 10**: Mã lệnh tích hợp HeapSort*

Và đây là khai báo ở hàm main



*Hình 11**: Khai báo ỏ hàm main*



*Hình 12**: Kết thúc hàm main*

# CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

**Về phần hiệu năng:**

**BubbleSort**

* Thời Gian: trong trường hợp trung bình và xấu nhất, trong trường hợp tốt nhất (khi danh sách đã được sắp xếp).
* Không Gian: vì BubbleSort là một thuật toán sắp xếp tại chỗ (in-place).
* Đánh Giá: BubbleSort không hiệu quả cho các danh sách lớn do số lần so sánh và hoán đổi lớn. Tuy nhiên, nó đơn giản và có thể hữu ích cho các danh sách nhỏ hoặc gần như đã sắp xếp.

**QuickSort**

* Thời Gian: trong trường hợp trung bình và tốt nhất, nhưng O(n^2) trong trường hợp xấu nhất.
* Không Gian: do sử dụng đệ quy, tuy nhiên trên danh sách liên kết, QuickSort có thể được triển khai để sử dụng không gian.
* Đánh Giá: QuickSort thường nhanh hơn BubbleSort và HeapSort, đặc biệt với dữ liệu lớn. Tuy nhiên, nó không ổn định và có thể gặp vấn đề với dữ liệu đã sắp xếp.

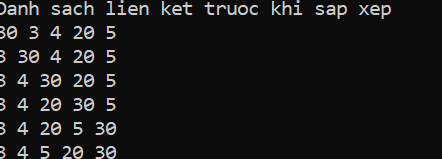
**HeapSort**

* Thời Gian: cho cả trường hợp trung bình và xấu nhất.
* Không Gian: nếu thực hiện trên một mảng; tuy nhiên, nếu bạn cần chuyển từ danh sách liên kết sang mảng, không gian là O(n).
* Đánh Giá: HeapSort cung cấp hiệu năng ổn định, nhưng thường không nhanh bằng QuickSort trong thực tế do overhead trong việc quản lý heap.

**Tổng Kết**

* Hiệu Suất Trong Thực Tế: Hiệu suất của các thuật toán này sẽ phụ thuộc vào môi trường và bộ dữ liệu. QuickSort thường được ưu tiên cho hiệu suất cao, nhưng nó có thể không phù hợp với mọi tình huống, đặc biệt khi bộ dữ liệu đã sắp xếp hoặc rất lớn.
* Lựa Chọn Thuật Toán: Lựa chọn thuật toán phụ thuộc vào yếu tố như kích thước dữ liệu, yêu cầu về ổn định và không gian bộ nhớ. Trong một số trường hợp, sự kết hợp của các thuật toán.

**Kết quả thực hiện:**

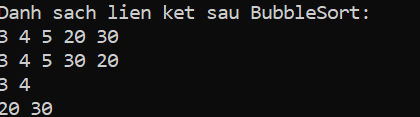


*Hình 13**: Danh sách trước khi sắp xếp*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 30 | 3 | 4 | 20 | 5 |
| 3 | 30 | 4 | 20 | 5 |
| 3 | 4 | 30 | 20 | 5 |
| 3 | 4 | 20 | 30 | 5 |
| 3 | 4 | 20 | 5 | 30 |
| 3 | 4 | 5 | 20 | 30 |

*Bảng 1**: Danh sách ban đầu*

Thực hiện theo thuật toán BubbleSort và hiện từng bước sắp xếp

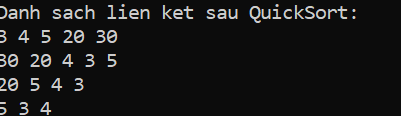


*Hình 14**: Sắp xếp theo thuật toán BubbleSort*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 20 | 30 |
| 3 | 4 | 5 | 30 | 20 |
| 3 | 4 |  |  |  |
| 20 | 30 |  |  |  |

*Bảng 2**: Danh sách sắp xếp theo BubbleSort*

Thực hiện theo thuật toán QuickSort và thực từng bước sắp xếp

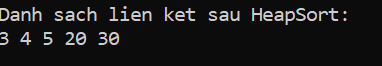
**

*Hình 15**: Sắp xếp theo thuật toán QuickSort*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 20 | 30 |
| 30 | 20 | 4 | 3 | 5 |
| 20 | 5 | 4 | 3 |  |
| 5 | 3 | 4 |  |  |

*Bảng 3**: Danh sách sắp xếp theo QuickSort*

Thực hiện theo thuật toán HeapSort



*Hình 16* *Sắp cếp theo thuật toán HeapSort*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 20 | 30 |

*Bảng 4**: Danh sách sắp xếp theo HeapSort*

# CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

**Những kết quả dạt được:**

* Thực hiện tích hợp các thuât toán sắp xếp vào danh sách liên kết đơn thành công
* Tích hợp các thuật toán sắp xếp vào danh sách lúc đầu còn gặp nhiều khó khăn nhưng trong thời gian tìm hiểu thì đã tích hợp được

**Đóng góp:**

* Kết quả hiện giò vẫn chưa đúng yêu cầu nhưng đó là những gì em đã nghiên cứu
* Trong quá trình nghiên cứu em tự nhận mình chưa làm hết sức mình.
* Hướng phát triển: Kiến nghị về những hướng nghiên cứu tiếp theo.

Kiến nghị về hướng nghiên cứu:

* Em sẽ cố gắng làm hết sức khi còn được thầy hướng dẫn trong những đồ án tiếp theo

# DANH MỤC TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | C. Nguyễn, “Thuật toán sắp xếp C++,” 17 03 2022. [Trực tuyến]. Available: https://vnoi.info/wiki/algo/basic/sorting.md. |
| [2] | C. Nguyễn, “Thuật toán sắp xếp C++,” 12 08 2022. [Trực tuyến]. Available: https://vnoi.info/wiki/algo/basic/sorting.md. |
| [3] | N. Quốc, “Tổng hợp các thuật toán sắp xếp trong C/C++,” 13 08 2021. [Trực tuyến]. Available: https://isinhvien.com/tong-hop-cac-thuat-toan-sap-xep-trong-c-c/#google\_vignette. |
| [4] | Khanh\_48, “Codelearn,” 23 10 2020. [Trực tuyến]. Available: https://codelearn.io/sharing/cac-thuat-toan-sap-xep-trong-cpp. |

# PHỤ LỤC

[LỜI CẢM ƠN 2](#_Toc6445)

[MỤC LỤC 4](#_Toc5768)

[DANH MỤC HÌNH ẢNH – BẢNG BIỂU 5](#_Toc22545)

[TÓM TẮT NIÊN LUẬN/ĐỒ ÁN CƠ SỞ NGÀNH/ĐỒ ÁN CHUYÊN NGÀNH 6](#_Toc1429)

[MỞ ĐẦU 7](#_Toc2524)

[CHƯƠNG 1: TỔNG QUAN 8](#_Toc27903)

[CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT 11](#_Toc28768)

[2.1 Tổng quan về thuật toán sắp xếp 11](#_Toc10703)

[2.1.1 Tổng quan 11](#_Toc5370)

[2.1.2 Ưu điểm và nhược điểm 13](#_Toc9858)

[2.1.3 Cách thức hoạt động 14](#_Toc27774)

[2.2 Tổng quan về danh sách liên kết đơn 16](#_Toc4030)

[2.2.1 Tổng quan 16](#_Toc22424)

[2.2.2 Ưu điểm và nhược điểm 16](#_Toc30640)

[2.2.3 Cách thức hoạt động 17](#_Toc19233)

[2.3 Tổng quan về C++ 18](#_Toc9613)

[2.3.1 Tổng quan 18](#_Toc2109)

[2.3.2 Ưu và nhược điểm 19](#_Toc31166)

[2.3.3 Cách thức hoạt động 21](#_Toc19002)

[CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU 22](#_Toc16513)

[3.1 Các bước thực hiện 22](#_Toc7612)

[3.1.1 Áp dụng QuickSort, BubbleSort, HeapSort vào danh sách liên kết đơn 22](#_Toc7463)

[CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU 25](#_Toc12160)

[CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 28](#_Toc1239)

[DANH MỤC TÀI LIỆU THAM KHẢO 29](#_Toc25882)

[PHỤ LỤC 30](#_Toc21290)