**CHUẨN BỊ THỰC HÀNH**

**Chương 1 Lab2**

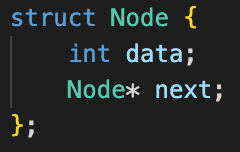
**Họ và tên:** Nguyễn Anh Tuấn

**MSSV:** 20200400

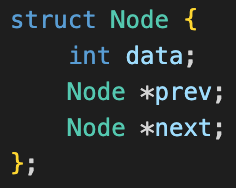
**Lớp:** 2

**Bài làm:**

Impl Singly Linked List:

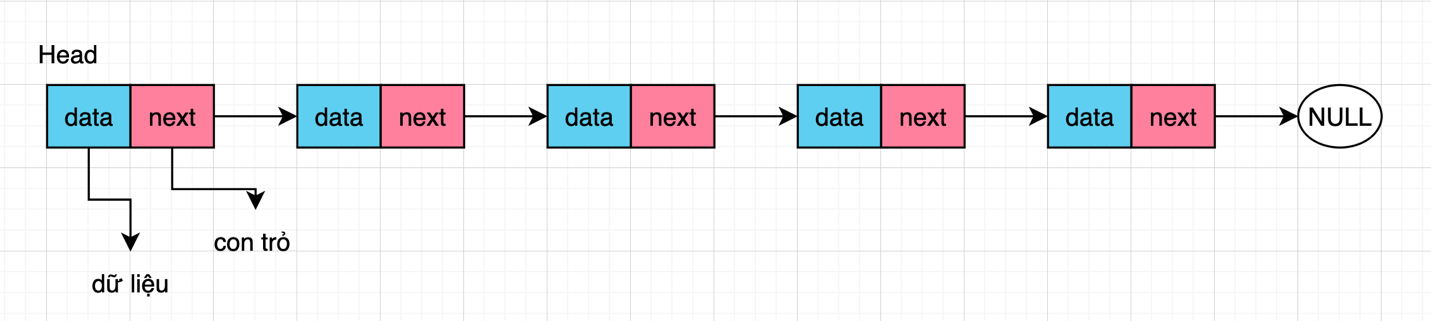
****

Impl Doubly Linked List:

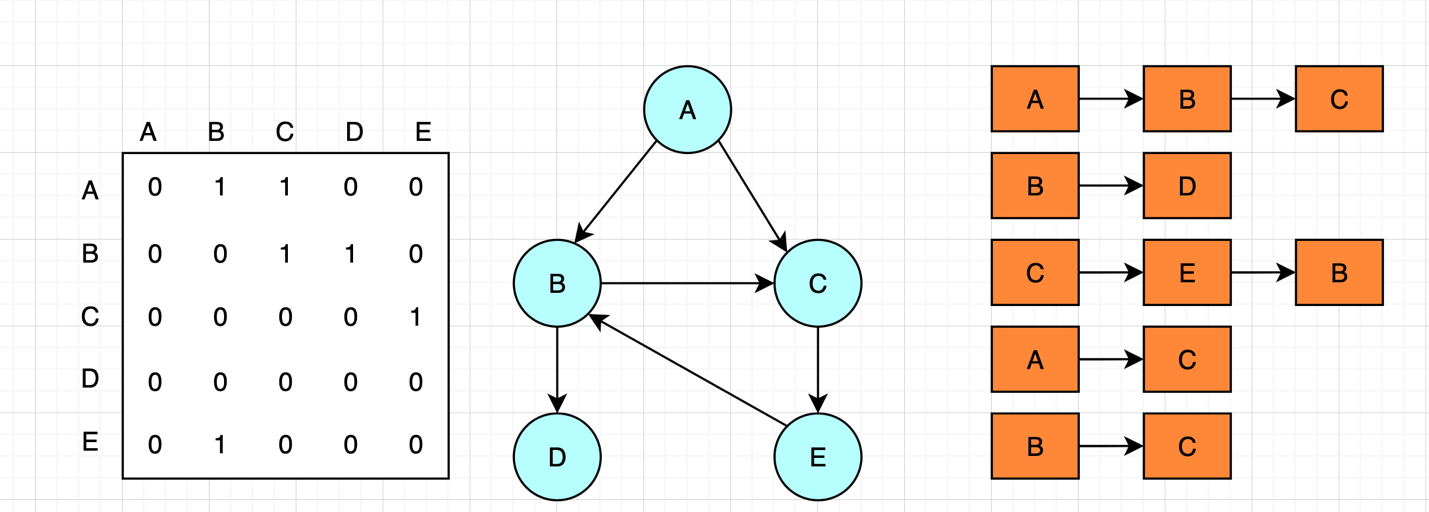
****

1. **Nêu ưu/nhược điểm của CTDL kiểu danh sách động và danh sách kề:**

* Linked List



* Adjacency List ( Graph )



|  |  |  |
| --- | --- | --- |
| **Tính chất** | **Danh sách động** | **Danh sách kề** |
| Mở rộng | Dễ dàng | Khó khăn |
| Truy cập | Chậm | Nhanh |
| Bộ nhớ | Sử dụng nhiều | Tiết kiệm |
| Thêm/Xóa | Dễ dàng | Khó khăn |
| Tìm kiếm cạnh | Khó khăn | Khó khăn |
| Hướng cạnh | Không giữ được | Giữ được |

Cấu trúc dữ liệu danh sách động (Linked List) và danh sách kề (Adjacency List) đều được sử dụng để lưu trữ và thao tác với dữ liệu đồ thị. Dưới đây là một số ưu/nhược điểm của từng loại cấu trúc dữ liệu này:

Ưu điểm của danh sách động:

* Cấu trúc dữ liệu này có thể mở rộng dễ dàng bằng cách thêm nút mới vào đầu hoặc cuối danh sách.
* Không cần phải biết trước kích thước của danh sách, do đó, nó có thể mở rộng và thu hẹp động linh hoạt.
* Các phần tử trong danh sách động có thể được chèn hoặc xóa dễ dàng, chỉ cần thay đổi liên kết giữa các nút.

Nhược điểm của danh sách động:

* Truy cập các phần tử trong danh sách động là chậm hơn so với mảng do phải duyệt qua từng nút trong danh sách để tìm kiếm phần tử cần truy cập.
* Do danh sách động cần phải lưu trữ các liên kết giữa các nút, nó sử dụng bộ nhớ hơn so với mảng.

Ưu điểm của danh sách kề:

* Danh sách kề chỉ lưu trữ các đỉnh và các cạnh kề của chúng, do đó, nó tiết kiệm bộ nhớ hơn so với ma trận kề hoặc danh sách cạnh.
* Truy cập các phần tử trong danh sách kề nhanh hơn so với danh sách động vì chỉ cần truy cập vào phần tử cần tìm trong danh sách kề.

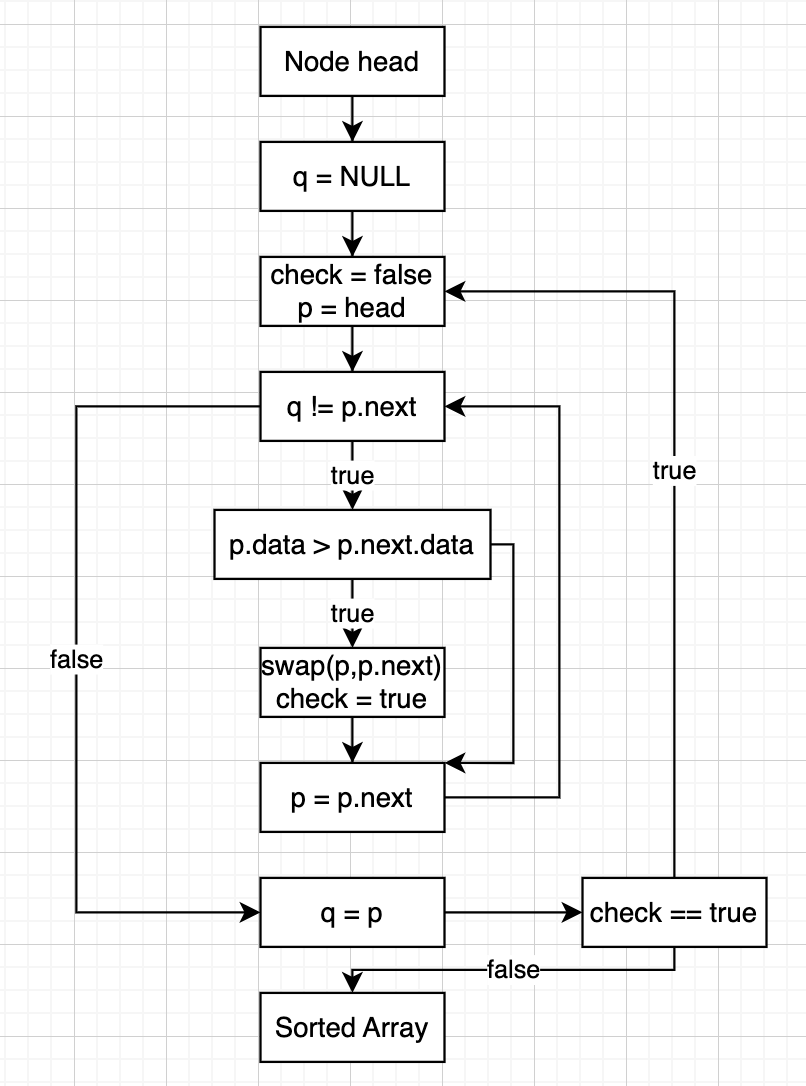
Nhược điểm của danh sách kề:

* Không thể dễ dàng mở rộng danh sách kề nếu số lượng đỉnh trong đồ thị tăng lên.
* Khó khăn trong việc tìm kiếm đến các cạnh không có sẵn trong danh sách kề.
* Không giữ được thông tin về hướng của các cạnh.

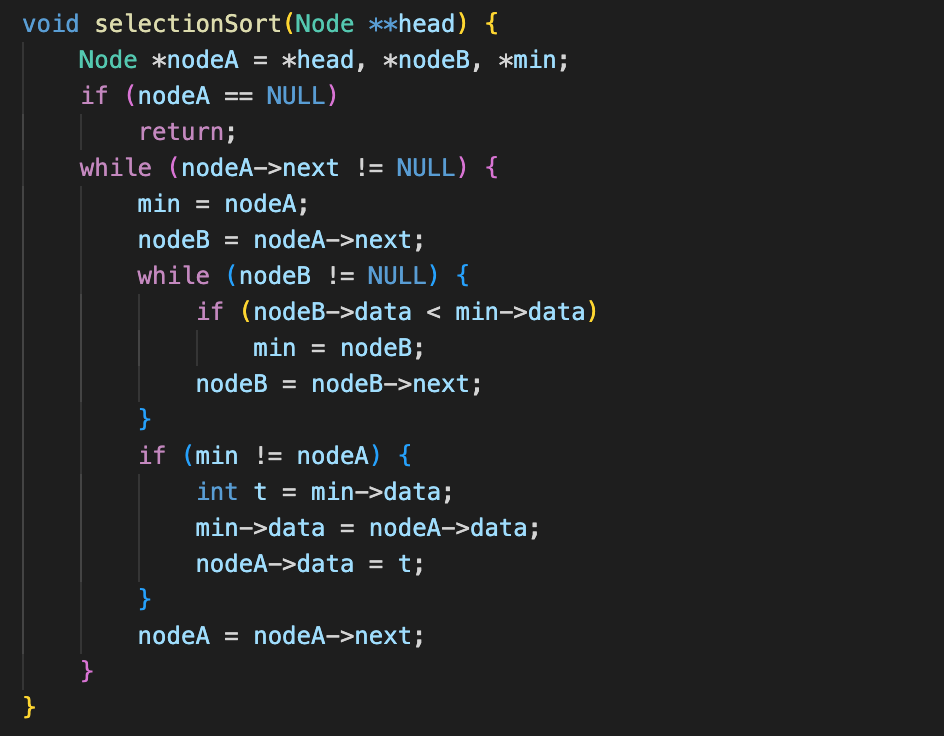
1. **Flowchart Bubble sort và Selection sort danh sách liên kết đơn:**

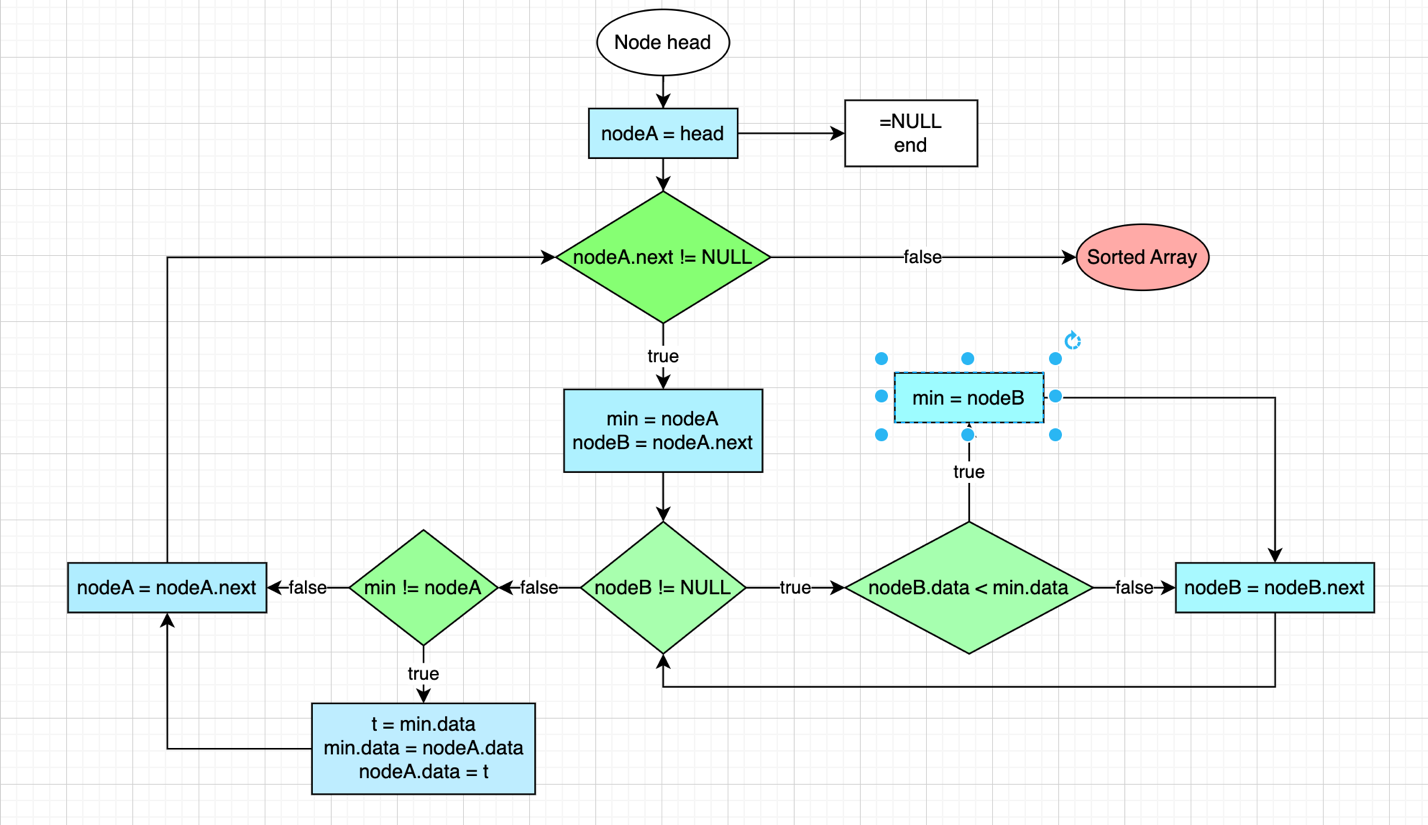
Code implement **Bubble Sort**:



****

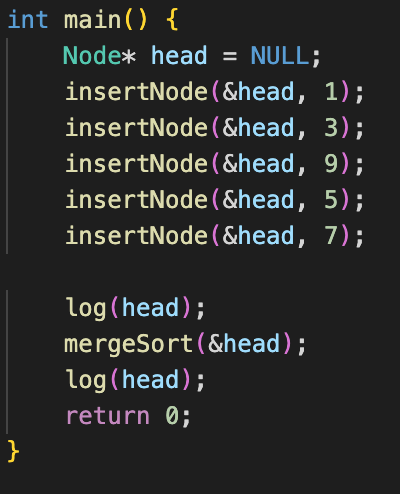
Code implement **Selection Sort**:



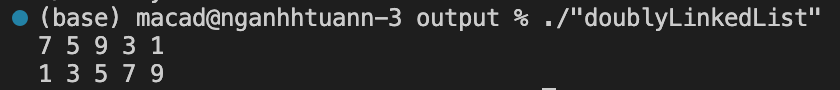


1. **Trình bày Merge sort và Shaker sort bằng danh sách liên kết kép:**

Code implement **Merge Sort**:

****

**Kết quả:**

****

**Trình bày:**

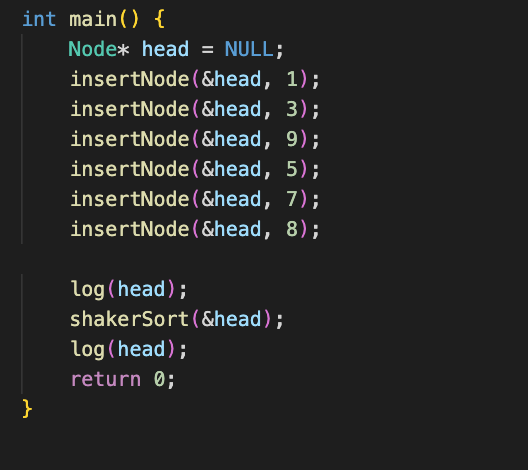
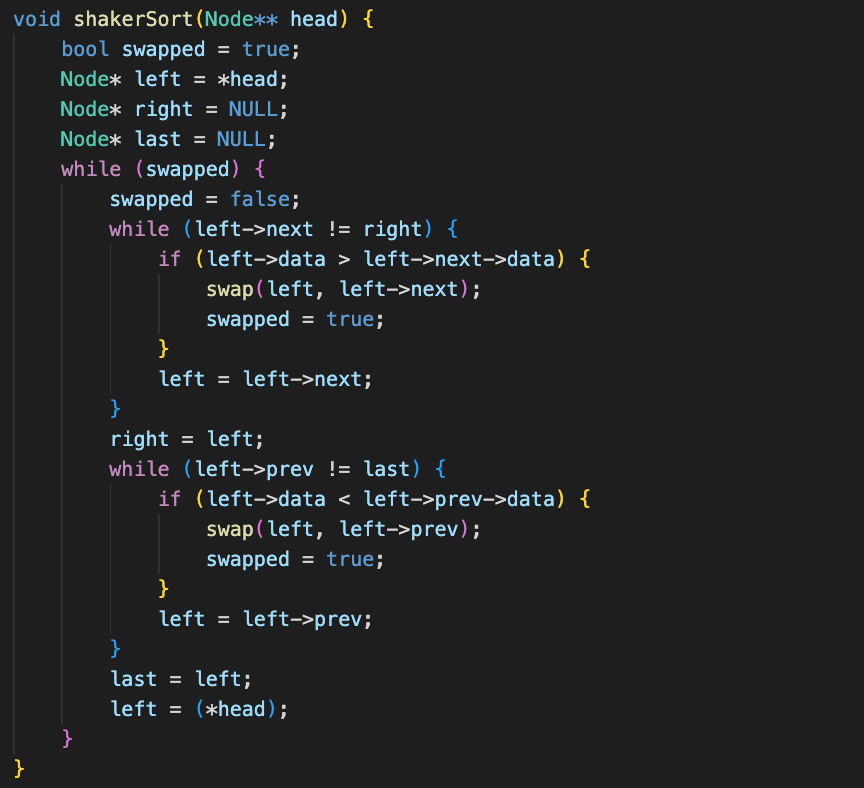
Thuật toán merge sort với danh sách liên kết kép sẽ được thực hiện như sau:

* Hàm mergeSort(head) sẽ được gọi với con trỏ đến đầu danh sách liên kết kép ban đầu.
* Nếu danh sách liên kết kép có ít hơn hai phần tử, không cần sắp xếp, return head\_ref.
* Sử dụng con trỏ slow và fast để tìm giữa danh sách và chia thành hai danh sách con.
* Đệ quy sử dụng mergeSort cho hai danh sách con.
* Gọi hàm merge để trộn hai danh sách con đã được sắp xếp.
* Trả về con trỏ đến danh sách đã được sắp xếp.

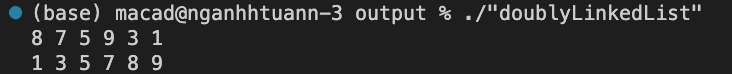
Hàm merge sẽ được thực hiện như sau:

* Tạo một danh sách liên kết kép mới để lưu trữ danh sách đã được trộn.
* Sử dụng hai con trỏ để duyệt qua hai danh sách đã được sắp xếp.
* So sánh hai phần tử được trỏ bởi hai con trỏ, chọn phần tử nhỏ hơn và thêm vào danh sách mới.
* Di chuyển con trỏ tới phần tử tiếp theo trong danh sách được chọn.
* Lặp lại quá trình trên cho đến khi tất cả các phần tử đã được thêm vào danh sách mới.
* Trả về danh sách liên kết kép mới đã được trộn.

Code implement **Shaker Sort**:

****

**Kết quả:**

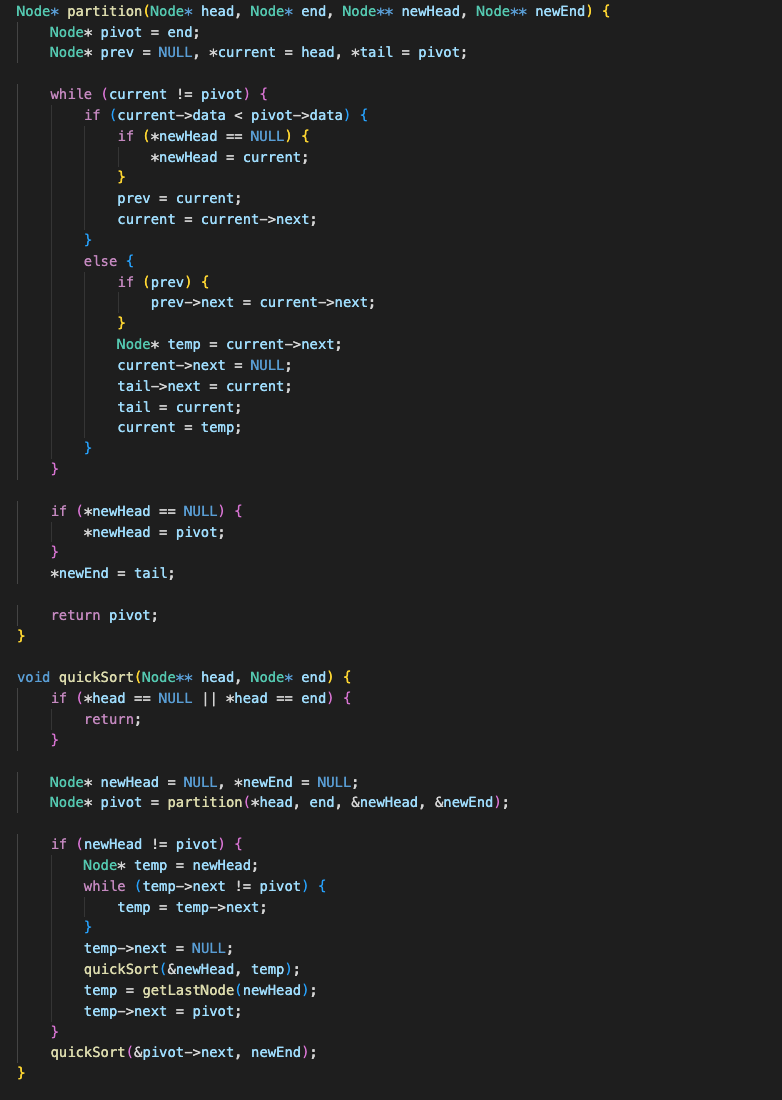
****

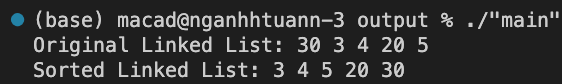
**Trình bày:**

Thuật toán Shaker sort trên danh sách liên kết kép có thể được thực hiện như sau:

* Khởi tạo hai biến left và right, lần lượt trỏ đến đầu và cuối danh sách liên kết.
* Sử dụng một biến check để kiểm tra xem có thực hiện hoán đổi giá trị giữa hai phần tử không. Ban đầu gán check bằng true.
* Lần lượt duyệt từ trái sang phải và từ phải sang trái của danh sách liên kết, so sánh các cặp phần tử kế tiếp và hoán đổi chúng nếu chúng không được sắp xếp đúng thứ tự. Sau mỗi lần hoán đổi, gán check bằng true để đánh dấu việc có thực hiện hoán đổi.
* Nếu sau khi duyệt từ trái sang phải và từ phải sang trái mà không có hoán đổi nào được thực hiện (check vẫn bằng true), thì dãy đã được sắp xếp đúng thứ tự, kết thúc thuật toán.
* Ngược lại, gán check bằng false và tiếp tục lặp lại các bước trên cho đến khi không có hoán đổi nào được thực hiện.

1. **Trình bày QuickSort không đệ quy bằng danh sách liên kết:**





Hàm partition để chia danh sách thành hai phần: một phần chứa các phần tử nhỏ hơn pivot và phần còn lại chứa các phần tử lớn hơn hoặc bằng pivot. Sau đó, chúng ta tiếp tục áp dụng thuật toán quicksort trên cả hai phần đó, bằng cách gọi đệ quy quickSort cho mỗi phần đó.

Hàm partition sử dụng ba con trỏ: prev, cur và tail để xử lý danh sách liên kết. Trong quá trình duyệt danh sách từ đầu đến pivot, các node có giá trị nhỏ hơn pivot sẽ được đưa vào danh sách mới bằng cách cập nhật con trỏ prev và cur. Các node có giá trị lớn hơn hoặc bằng pivot sẽ được di chuyển vào phần còn lại của danh sách bằng cách cập nhật con trỏ prev, cur và tail.

Hàm getLastNode được sử dụng để tìm kiếm và trả về node cuối cùng để xử lí.