**TRƯỜNG ĐẠI HỌC KINH DOANH VÀ CÔNG NGHỆ HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**MÔN CẤU TRÚC DỮ LIỆU**

Đề tài 3: Tìm hiểu thuật toán sắp xếp

**Insertion sort và Bubble sort**

Nhóm 4 - Lớp: TH28.27

Mai Quang Hải

Ngô Tuấn Việt

Nguyễn Ngọc Sơn

Trần Văn Tú

Giáo viên hướng dẫn: Phạm Thị Ngừng

**Mục Lục**

[1. Giới thiệu 3](#_Toc182031317)

[1.1. Mục tiêu đề tài 3](#_Toc182031318)

[1.2. Lý do chọn đề tài 3](#_Toc182031319)

[1.3. Phạm vi đề tài 4](#_Toc182031320)

[2. Thuật toán Insertion Sort 4](#_Toc182031321)

[2.1. Nguyên lý hoạt động 4](#_Toc182031322)

[2.2. Độ phức tạp thuật toán 5](#_Toc182031323)

[2.3. Chương trình Insertion Sort bằng Python 5](#_Toc182031324)

[2.4. Ví dụ minh họa 6](#_Toc182031325)

[Bước 1: Chia mảng thành hai phần 6](#_Toc182031326)

[Bước 2: Sắp xếp phần chưa sắp xếp 6](#_Toc182031327)

[2.5. Ưu điểm và nhược điểm 8](#_Toc182031328)

[3. Thuật toán Bubble Sort 8](#_Toc182031329)

[3.1. Mô tả thuật toán 8](#_Toc182031330)

[3.2. Độ phức tạp thuật toán 9](#_Toc182031331)

[3.3. Chương trinh Bubble Sort bằng Python 9](#_Toc182031332)

[3.4. Ví dụ minh họa 10](#_Toc182031333)

[3.5. Ưu điểm và nhược điểm 11](#_Toc182031334)

[4. So sánh Insertion Sort và Bubble Sort 12](#_Toc182031335)

[5. Kết luận 12](#_Toc182031336)

# 1. Giới thiệu

Sắp xếp là một trong những bài toán cơ bản và quan trọng trong khoa học máy tính, với mục tiêu sắp xếp một dãy các phần tử ( hoặc bất kỳ dãy dữ liệu nào ) theo một thứ tự nhất định, chẳng hạn như thứ tự tăng dần hoặc giảm dần. Các thuật toán sắp xếp có thể được phân loại theo độ phức tạp và cách thức hoạt động. Trong báo cáo này, chúng ta sẽ tìm hiểu về hai thuật toán sắp xếp đơn giản nhưng phổ biến là Insertion Sort và Bubble Sort. Cả hai thuật toán này đều thuộc nhóm thuật toán sắp xếp so sánh và có đặc điểm dễ hiểu và dễ triển khai.

## 1.1. Mục tiêu đề tài

Mục tiêu của đề tài là nghiên cứu và phân tích hai thuật toán sắp xếp cơ bản **Bubble Sort** và **Insertion Sort**, nhằm:

* Hiểu rõ về cách thức hoạt động, nguyên lý và các bước của từng thuật toán.
* Đánh giá và so sánh độ phức tạp thời gian, không gian của hai thuật toán trong các tình huống khác nhau.
* Tìm hiểu ưu nhược điểm của mỗi thuật toán và ứng dụng của chúng trong thực tế.
* Cải thiện khả năng viết mã và tối ưu thuật toán sắp xếp cho các bài toán thực tế.

## 1.2. Lý do chọn đề tài

Thuật toán sắp xếp là một trong những thuật toán cơ bản và quan trọng nhất trong lĩnh vực khoa học máy tính. Dưới đây là những lý do chính để chọn đề tài này:

* **Tính phổ biến**: Các thuật toán sắp xếp như Bubble Sort và Insertion Sort là những thuật toán cơ bản mà bất kỳ lập trình viên nào cũng cần phải hiểu rõ. Chúng không chỉ có trong các bài học cơ bản mà còn là nền tảng cho việc học các thuật toán sắp xếp phức tạp hơn.
* **Đơn giản và dễ hiểu**: Cả hai thuật toán này đều có cấu trúc đơn giản, dễ hiểu, giúp người học dễ dàng tiếp cận và nghiên cứu cách hoạt động của thuật toán sắp xếp.
* **Đánh giá hiệu suất**: Mặc dù là các thuật toán đơn giản, nhưng **Bubble Sort** và **Insertion Sort** có các đặc điểm khác nhau về hiệu suất trong các tình huống khác nhau (mảng đã được sắp xếp, mảng ngược hoàn toàn), giúp người học có cái nhìn sâu sắc hơn về các yếu tố ảnh hưởng đến hiệu suất thuật toán.
* **Lý thuyết và thực hành**: Việc thực hiện các thuật toán này trong mã nguồn thực tế sẽ giúp học viên nắm vững các khái niệm lý thuyết về thuật toán sắp xếp, đồng thời cải thiện kỹ năng lập trình của mình.

## 1.3. Phạm vi đề tài

Đề tài này sẽ nghiên cứu và thực hiện thuật toán sắp xếp **Bubble Sort** và **Insertion Sort** với phạm vi như sau:

* **Phạm vi lý thuyết**: Tìm hiểu về lý thuyết thuật toán sắp xếp, cách thức hoạt động của các thuật toán Bubble Sort và Insertion Sort, so sánh độ phức tạp thời gian và không gian của chúng.
* **Phạm vi thực hành**: Triển khai hai thuật toán Bubble Sort và Insertion Sort trong một ngôn ngữ lập trình cụ thể (ví dụ: Python, C++). Đánh giá hiệu suất của từng thuật toán trên các mảng có kích thước khác nhau và các tình huống khác nhau như mảng đã sắp xếp, mảng ngược hoàn toàn, mảng ngẫu nhiên.
* **Phạm vi ứng dụng**: Đưa ra các bài toán thực tế và các ứng dụng mà thuật toán Bubble Sort và Insertion Sort có thể áp dụng. Chú trọng đến các bài toán có dữ liệu nhỏ hoặc yêu cầu tính ổn định trong quá trình sắp xếp.
* **Giới hạn nghiên cứu**: Đề tài sẽ không đi sâu vào các thuật toán sắp xếp phức tạp hơn như **Quick Sort**, **Merge Sort**, hoặc **Heap Sort**, mà sẽ tập trung vào phân tích và so sánh hai thuật toán cơ bản này.

# 2. Thuật toán Insertion Sort

## 2.1. Nguyên lý hoạt động

Thuật toán Insertion Sort hoạt động theo nguyên lý giống như cách mà chúng ta sắp xếp các lá bài trong một ván bài. Khi bắt đầu, ta coi phần tử đầu tiên là đã được sắp xếp. Sau đó, lần lượt thêm các phần tử tiếp theo vào "dãy đã sắp xếp" sao cho dãy luôn giữ được tính chất đã sắp xếp.

Cụ thể, thuật toán thực hiện các bước sau:

1. Bắt đầu từ phần tử thứ hai ( vì phần tử đầu tiên coi như đã được sắp xếp ), so sánh với các phần tử trước đó trong dãy đã sắp xếp.

2. Nếu phần tử hiện tại nhỏ hơn phần tử trước đó, ta dịch chuyển phần tử trước đó sang vị trí tiếp theo.

3. Lặp lại bước trên cho đến khi tìm được vị trí thích hợp cho phần tử hiện tại.

4. Lặp lại các bước cho tất cả các phần tử trong mảng.

## 2.2. Độ phức tạp thuật toán

- Thời gian thực thi trung bình và xấu nhất: O(n²), khi các phần tử cần phải di chuyển rất nhiều.

- Thời gian thực thi tốt nhất: O(n), khi mảng đã được sắp xếp sẵn.

- Không gian lưu trữ: O(1), vì thuật toán này là sắp xếp tại chỗ (in-place).

## 2.3. Chương trình Insertion Sort bằng Python

from os import system

n = int(input("Nhap so phan tu: "))

arr = []

def insertionSort(arr):

for i in range(1, n):

print(f"Vong lap {i}: {arr}")

key = arr[i]

j = i - 1

while j >= 0 and key < arr[j]:

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

for i in range(n):

arr.append(int(input(f"Nhap phan tu thu {i + 1}: ")))

system("cls")

print(f"Truoc khi sort: {arr}\n")

insertionSort(arr)

print(f"\nKet qua: {arr}")

## 2.4. Ví dụ minh họa

Giả sử chúng ta có một dãy số chưa sắp xếp:

[5, 2, 9, 1, 5, 6]

### Bước 1: Chia mảng thành hai phần

* **Phần đã sắp xếp**: Ban đầu chỉ có phần tử đầu tiên, tức là [5].
* **Phần chưa sắp xếp**: Phần còn lại của mảng là [2, 9, 1, 5, 6].

### Bước 2: Sắp xếp phần chưa sắp xếp

Bắt đầu từ phần tử thứ hai, thuật toán sẽ so sánh từng phần tử với các phần tử trong phần đã sắp xếp, di chuyển các phần tử trong phần đã sắp xếp để tìm vị trí thích hợp cho phần tử hiện tại.

**Lượt 1** (Phần tử 2):

* **So sánh**: 2 với 5 (phần tử trong phần đã sắp xếp).
* **Kết quả**: 2 < 5, nên ta di chuyển 5 sang vị trí tiếp theo và chèn 2 vào đầu dãy.
* Mảng sau bước này: [2, 5, 9, 1, 5, 6].

**Lượt 2** (Phần tử 9):

* **So sánh**: 9 với 5 (phần tử trong phần đã sắp xếp).
* **Kết quả**: 9 > 5, không cần thay đổi, giữ nguyên thứ tự.
* Mảng sau bước này: [2, 5, 9, 1, 5, 6].

**Lượt 3** (Phần tử 1):

* **So sánh**: 1 với 9 (phần tử trong phần đã sắp xếp).
* **Kết quả**: 1 < 9, di chuyển 9 sang vị trí tiếp theo.
* **So sánh**: 1 với 5.
* **Kết quả**: 1 < 5, di chuyển 5 sang vị trí tiếp theo.
* **So sánh**: 1 với 2.
* **Kết quả**: 1 < 2, di chuyển 2 sang vị trí tiếp theo.
* Sau đó, chèn 1 vào vị trí đầu tiên.
* Mảng sau bước này: [1, 2, 5, 9, 5, 6].

**Lượt 4** (Phần tử 5):

* **So sánh**: 5 với 9.
* **Kết quả**: 5 < 9, di chuyển 9 sang vị trí tiếp theo.
* **So sánh**: 5 với 5 (vị trí tiếp theo).
* **Kết quả**: 5 = 5, không cần thay đổi.
* Mảng sau bước này: [1, 2, 5, 5, 9, 6].

**Lượt 5** (Phần tử 6):

* **So sánh**: 6 với 9.
* **Kết quả**: 6 < 9, di chuyển 9 sang vị trí tiếp theo.
* **So sánh**: 6 với 5 (phần tử liền trước).
* **Kết quả**: 6 > 5, không cần di chuyển nữa.
* Chèn 6 vào vị trí giữa 5 và 9.
* Mảng sau bước này: [1, 2, 5, 5, 6, 9].

Kết quả sau khi sắp xếp xong là: `[1, 2, 5, 5, 6, 9]`.

## 2.5. Ưu điểm và nhược điểm

- Ưu điểm

- Dễ hiểu và dễ triển khai.

- Không cần bộ nhớ bổ sung.

- Hiệu quả khi dữ liệu đã gần như được sắp xếp.

- Nhược điểm:

- Độ phức tạp thời gian O(n²) đối với các mảng có số lượng phần tử lớn, nên không hiệu quả khi số lượng phần tử lớn hoặc trong các ứng dụng cần tối ưu hiệu suất.

# 3. Thuật toán Bubble Sort

## 3.1. Mô tả thuật toán

Bubble Sort là một thuật toán sắp xếp đơn giản hoạt động theo cách "bong bóng nổi lên" . Nó hoạt động bằng cách lặp qua dãy dữ liệu và so sánh các cặp phần tử kề nhau. Nếu phần tử trước lớn hơn phần tử sau, hai phần tử này sẽ được đổi chỗ ( trong trường hợp sắp xếp tăng dần ). Quá trình này được lặp lại cho đến khi không còn phần tử nào cần đổi chỗ nữa.

Cụ thể, thuật toán hoạt động như sau:

1. Thuật toán sẽ duyệt qua dãy dữ liệu, so sánh hai phần tử liền kề trong mảng.

2. Nếu phần tử bên trái lớn hơn phần tử bên phải, đổi chỗ chúng.( sắp xếp tăng dần )

3. Tiếp tục so sánh các cặp phần tử liền kề. Sau mỗi lần duyệt qua dãy, phần tử lớn nhất sẽ được “nổi” đến vị trí cuối cùng.

4. Lặp lại các bước trên cho đến khi dãy đã được sắp xếp.

## 3.2. Độ phức tạp thuật toán

- Thời gian thực thi trung bình và xấu nhất: O(n²), khi mảng không được sắp xếp và cần nhiều lượt hoán đổi.

- Thời gian thực thi tốt nhất: O(n), nếu mảng đã được sắp xếp từ đầu.

- Không gian lưu trữ: O(1), vì thuật toán này cũng là sắp xếp tại chỗ.

## 3.3. Chương trinh Bubble Sort bằng Python

from os import system

n = int(input("Nhap so phan tu: "))

arr = []

def bubbleSort(arr):

for i in range(n):

print(f"Vong lap {i + 1}: {arr}")

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

for i in range(n):

arr.append(int(input(f"Nhap phan tu thu {i + 1}: ")))

system("cls")

print(f"Truoc khi sort: {arr}\n")

bubbleSort(arr)

print(f"\nKet qua: {arr}")

## 3.4. Ví dụ minh họa

* **Vòng lặp ngoài** chạy qua toàn bộ mảng, mỗi vòng lặp sẽ giúp "đẩy" phần tử lớn nhất vào cuối mảng.
* **Vòng lặp trong** sẽ so sánh các phần tử liền kề và hoán đổi chúng nếu chúng không đúng thứ tự.

#### Dãy ban đầu: [5, 2, 9, 1, 5, 6]

Vòng lặp 1:

1. So sánh 5 và 2: 5 > 2 → Hoán đổi → [2, 5, 9, 1, 5, 6]
2. So sánh 5 và 9: 5 < 9 → Không hoán đổi → [2, 5, 9, 1, 5, 6]
3. So sánh 9 và 1: 9 > 1 → Hoán đổi → [2, 5, 1, 9, 5, 6]
4. So sánh 9 và 5: 9 > 5 → Hoán đổi → [2, 5, 1, 5, 9, 6]
5. So sánh 9 và 6: 9 > 6 → Hoán đổi → [2, 5, 1, 5, 6, 9]

Sau vòng lặp 1, phần tử lớn nhất (9) đã "bong bóng" lên vị trí cuối cùng của mảng.

Vòng lặp 2:

1. So sánh 2 và 5: 2 < 5 → Không hoán đổi → [2, 5, 1, 5, 6, 9]
2. So sánh 5 và 1: 5 > 1 → Hoán đổi → [2, 1, 5, 5, 6, 9]
3. So sánh 5 và 5: 5 = 5 → Không hoán đổi → [2, 1, 5, 5, 6, 9]
4. So sánh 5 và 6: 5 < 6 → Không hoán đổi → [2, 1, 5, 5, 6, 9]

Sau vòng lặp 2, phần tử lớn thứ hai (6) đã "bong bóng" lên vị trí kế cuối.

Vòng lặp 3:

1. So sánh 2 và 1: 2 > 1 → Hoán đổi → [1, 2, 5, 5, 6, 9]
2. So sánh 2 và 5: 2 < 5 → Không hoán đổi → [1, 2, 5, 5, 6, 9]
3. So sánh 5 và 5: 5 = 5 → Không hoán đổi → [1, 2, 5, 5, 6, 9]

Sau vòng lặp 3, phần tử lớn thứ ba (5) đã "bong bóng" lên vị trí thứ ba.

Vòng lặp 4:

1. So sánh 1 và 2: 1 < 2 → Không hoán đổi → [1, 2, 5, 5, 6, 9]

Sau vòng lặp 4, mảng đã được sắp xếp hoàn toàn.

Kết quả sau khi sắp xếp xong là: `[1, 2, 5, 5, 6, 9]`.

## 3.5. Ưu điểm và nhược điểm

- Ưu điểm:

- Dễ hiểu và dễ triển khai.

- Hoạt động tốt với các mảng nhỏ.

- Không yêu cầu bộ nhớ phụ.

- Nhược điểm:

- Độ phức tạp thời gian O(n²) làm cho thuật toán không hiệu quả đối với mảng có kích thước lớn.

- Các lượt hoán đổi không cần thiết có thể làm giảm hiệu suất.

# 4. So sánh Insertion Sort và Bubble Sort

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Insertion Sort** | **Bubble Sort** |
| Độ phức tạp thời gian | Trung bình và xấu nhất O(n^2), tốt nhất O(n) | Trung bình và xấu nhất O(n^2), tốt nhất O(n) |
| Độ phức tạp không | O(1) | O(1) |
| Khả năng hoạt động khi mảng gần như đã sắp xếp | Tốt, hiệu quả hơn khi mảng gần như đã được sắp xếp | Kém, cần nhiều lượt so sánh dù mảng đã gần như sắp xếp |
| Sự ổn định | Ổn định (không thay đổi thứ tự của các phần tử có giá trị bằng nhau) | Ổn |
| Khả năng áp dụng trong thực tế | Thường dùng khi mảng nhỏ hoặc gần như sắp xếp | Thường dùng khi mảng nhỏ hoặc trong các tình huống học thuật |

# 5. Kết luận

Cả Insertion Sort và Bubble Sort đều là các thuật toán sắp xếp đơn giản và dễ hiểu. Tuy nhiên, chúng đều có độ phức tạp thời gian O(n^2) trong trường hợp xấu nhất nên chúng không phải là lựa chọn tối ưu khi xử lý các dữ liệu lớn.Mặc dù vậy, Insertion Sort có thể hiệu quả hơn khi xử lý dữ liệu gần như đã được sắp xếp, và Bubble Sort thích hợp chô các bài toán nhỏ và dễ tối ưu hóa.

Trong các tình huống thực tế, những thuật toán như Quick Sort, Merge Sort hoặc Heap Sort thường được ưu tiên hơn vì độ phức tạp thấp hơn trong trường hợp mảng lớn. Tuy nhiên, Insertion Sort và Bubble Sort vẫn có giá trị trong việc giảng dạy và nghiên cứu các khái niệm cơ bản của thuật toán sắp xếp.