**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO**

**HỌC PHẦN:** CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Bài tập sắp xếp

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực hiện** | **MSSV** | **Nhiệm vụ** |
| Huỳnh Phạm Hoàng Sơn | 3121411184 | Câu hỏi và bài tập cơ sở |
| Vũ Bình Phước | 3121411171 | Câu hỏi và bài tập ứng dụng |

Giảng viên môn học: ĐỖ NHƯ TÀI

TP.HCM ngày 14 tháng 3 năm 2025

**Mục lục**

[Câu hỏi 1](#_Toc22972)

[Trả lời 1](#_Toc15545)

[Bài tập ứng dụng 4](#_Toc27354)

[Bài tập cơ sở 4](#_Toc15579)

[Câu 1 4](#_Toc5445)

[Câu 2 8](#_Toc4616)

[Câu 3 10](#_Toc3838)

**Bài tập sắp xếp**

# Câu hỏi

1. Trình bày tư tưởng của các thuật toán sắp xếp?
2. Trong các thuật toán sắp xếp, bạn thích nhất thuật toán nào? Thuật toán nào bạn không thích nhất? Tại sao?
3. Trình bày và cài đặt tất cả các thuật toán sắp xếp nội, ngoại theo thứ tự giảm dần. Cho nhận xét về các thuật toán này.
4. Hãy trình bày những ưu điểm và nhược điểm của mỗi thuật toán sắp xếp? Theo bạn, cách khắc phục những nhược điểm này là gì?

## Trả lời:

1. Các thuật toán sắp xếp có nhiều loại khác nhau, nhưng chúng lun dựa theo một số tư tưởng chính để sắp xếp dữ liệu theo thứ tự mong muốn:
   1. Insertion Sort: Duy trì một danh sách con đã sắp xếp, sau đó chèn từng phần tử mới vào đúng vị trí của nó.
   2. Selection Sort: Lặp qua danh sách, mỗi lần tìm phần tử nhỏ nhất (hoặc lớn nhất) rồi đưa nó về đúng vị trí.
   3. Bubble Sort: So sánh từng cặp phần tử liền kề, đổi chỗ nếu sai thứ tự và lặp lại cho đến khi danh sách được sắp xếp hoàn toàn
   4. Quick Sort: Chọn một phần tử làm "pivot", chia danh sách thành hai phần (nhỏ hơn và lớn hơn pivot), sau đó tiếp tục sắp xếp hai phần này đệ quy.
   5. Merge Sort: Chia danh sách thành các phần nhỏ hơn, sắp xếp từng phần, rồi trộn chúng lại theo đúng thứ tự.
   6. Heap Sort: Xây dựng một **heap** để tìm phần tử lớn nhất (hoặc nhỏ nhất), sau đó đưa nó về đúng vị trí và lặp lại.
   7. Radix Sort: Sắp xếp dần theo từng chữ số (hoặc chữ cái) từ thấp đến cao, dựa vào một thuật toán sắp xếp khác như Counting Sort.
   8. Interchange sort: Là một thuật toán sắp xếp đơn giản dựa trên việc hoán đổi các phần tử. Nó hoạt động bằng cách so sánh từng cặp phần tử trong danh sách và hoán đổi vị trí của chúng nếu cần thiết để sắp xếp theo thứ tự mong muốn.
2. Thuật toán sắp xếp yêu thích nhất là Quick Sort, vì:

- Hiệu suất làm việc tốt hơn, nhanh hơn nhiều so với 1 số thuật toán khác như Bubble Sort hay Selection Sort

- Các tiếp cận đệ quy giúp thuật toán dễ hiểu hơn và dễ triển khai hơn

- Tối ưu bộ nhớ, không cần phải dùng mảng phụ như Merge Sort, tiết kiệm được phần lớn thời gian

Thuật toán sắp xếp không thích là Bubble Sort, vì:

- Độ phức tạp của thuật toán này làm chậm đi, kém hiểu quả đối với các danh sách lớn cần giải quyết

- So sánh và đổi vị trí liên túc. Điều này làm lãng phí tài nguyên và thời gian hoạt động

- Vì độ hiểu quả kém nên ít được tìm hiểu sâu và áp dụng rộng rãi. Kém phát triển hơn so với các thuật toán khác

Cài đặt: file code

1. Ưu điểm, nhược điểm và cách khắc phục:

## ***Insertion Sort (Sắp xếp chèn)***

### **Ưu điểm:**

Hiệu quả với mảng nhỏ hoặc gần như đã sắp xếp.  
 Ít phép hoán đổi so với Bubble Sort.  
 Sử dụng trong các ứng dụng yêu cầu sắp xếp ổn định.

### **Nhược điểm:**

Chậm với dữ liệu lớn do O(n²).

### **Khắc phục:**

Nếu dữ liệu lớn, có thể sử dụng **Quick Sort hoặc Merge Sort**.

## ***Selection Sort (Sắp xếp chọn)***

### **Ưu điểm:**

### Hoạt động tốt trên tập dữ liệu nhỏ. Số lần hoán đổi ít hơn Bubble Sort.

### **Nhược điểm:**

Chậm với dữ liệu lớn.  
 Không ổn định.

### **Khắc phục:**

Dùng **Heap Sort** nếu cần tìm phần tử lớn nhất/nhỏ nhất nhanh hơn.

## **Bubble Sort (Sắ xếp nổi bọt)**

### **Ưu điểm:**

Cực kỳ dễ hiểu, dễ triển khai.  
 Có thể tối ưu bằng cách dừng sớm nếu danh sách đã sắp xếp.

### **Nhược điểm:**

Rất chậm, tốn nhiều phép hoán đổi.

### **Khắc phục:**

Dùng **Quick Sort hoặc Merge Sort** nếu dữ liệu lớn.

## ***Quick Sort (Sắp xếp nhanh)***

### **Ưu điểm:**

Rất nhanh với dữ liệu ngẫu nhiên.  
 Phù hợp với dữ liệu lớn.

### **Nhược điểm:**

Nếu chọn pivot xấu, có thể chậm như Bubble Sort (O(n²)).  
 Không ổn định.

### **Khắc phục:**

Chọn pivot tốt hơn bằng **Median-of-Three** hoặc **chọn pivot ngẫu nhiên**.

## ***Heap Sort (Sắp xếp vun đống)***

### **Ưu điểm:**

Luôn đảm bảo hiệu suất O(n log n).  
 Tốt khi cần tìm phần tử lớn nhất/nhỏ nhất nhiều lần.

### **Nhược điểm:**

Không ổn định.  
 Không hiệu quả bằng Quick Sort trong hầu hết các trường hợp.

### **Khắc phục:**

Dùng **Merge Sort nếu cần sắp xếp ổn định**.

## ***Interchange Sort (Sắp xếp đổi chỗ trực tiếp)***

### **Ưu điểm:**

Dễ hiểu, dễ triển khai.

### **Nhược điểm:**

Cực kỳ chậm, tốn nhiều phép hoán đổi.  
 Thậm chí còn kém hơn cả Selection Sort.

### **Khắc phục:**

Không nên sử dụng, thay thế bằng **Quick Sort hoặc Merge Sort**.

Bài tập ứng dụng: Trong cá file code

# Bài tập cơ sở

## Câu 1:

1. Mô phỏng như sau:
   * 1. Sắp xếp đổi chổ trực tiếp

Ở thuật toán này, ta sẽ dùng 1 phần từ so sánh từng phần tử phia sau nó

Sẽ hoán đổi nếu phần tử phía sau nhỏ hơn

|  |  |
| --- | --- |
| Số lần lặp | Dãy số được sắp xếp theo từng bước |
| 1 | 8, **39 → 5**, 1, 3, 6, 9, 12, 4, 7, 10 |
| 2 | 8, 5, **39 → 1**, 3, 6, 9, 12, 4, 7, 10 |
| 3 | 8, 5, 1, **39 → 3**, 6, 9, 12, 4, 7, 10 |
| 4 | 8, 5, 1, 3, **39 → 6**, 9, 12, 4, 7, 10 |
| 5 | 8, 5, 1, 3, 6, **39 → 9**, 12, 4, 7, 10 |
| 6 | 8, 5, 1, 3, 6, 9, **39 → 12**, 4, 7, 10 |
| 7 | 8, 5, 1, 3, 6, 9, 12, **39 → 4**, 7, 10 |
| 8 | 8, 5, 1, 3, 6, 9, 12, 4, **39 → 7**, 10 |
| 9 | 8, 5, 1, 3, 6, 9, 12, 4, 7, **39 → 10** |
| 10 | 8, 5, 1, 3, 6, 9, 12, 4, 7, 10, **39** |

Ta cứ làm vậy với các số khác, kết quả cho ra được:

1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39

* + 1. Sắp xếp chọn trực tiếp

Ở thuật toán này, sẽ đi tìm phần tử có giá trị nhỏ nhất và đem giá trị đó so sánh với các phần tử đầu tiên

|  |  |
| --- | --- |
| Số lần lặp | Dãy số được sắp xếp theo từng bước |
| 1 | **1**, 8, 5, 39, 3, 6, 9, 12, 4, 7, 10 |
| 2 | 1, **3**, 5, 39, 8, 6, 9, 12, 4, 7, 10 |
| 3 | 1, 3, **4**, 39, 8, 6, 9, 12, 5, 7, 10 |
| 4 | 1, 3, 4, **5**, 8, 6, 9, 12, 39, 7, 10 |
| 5 | 1, 3, 4, 5, **6**, 8, 9, 12, 39, 7, 10 |
| 6 | 1, 3, 4, 5, 6, **7**, 9, 12, 39, 8, 10 |
| 7 | 1, 3, 4, 5, 6, 7, **8**, 12, 39, 9, 10 |
| 8 | 1, 3, 4, 5, 6, 7, 8, **9**, 39, 12, 10 |
| 9 | 1, 3, 4, 5, 6, 7, 8, 9, **10**, 12, 39 |
| 10 | 1, 3, 4, 5, 6, 7, 8, 9, 10, **12**, 39 |

Ta cứ làm vậy với các số khác, kết quả cho ra được:

1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39

* + 1. Sắp xếp chèn trực tiếp

Ở thuật toán này sẽ giả sử phần đầu đã sắp xếp, chèn phần từ tiếp theo vào các vị trí thích hợp

|  |  |
| --- | --- |
| Số lần lặp | Dãy số được sắp xếp theo từng bước |
| 1 | **8, 39**, 5, 1, 3, 6, 9, 12, 4, 7, 10 |
| 2 | **5, 8, 39**, 1, 3, 6, 9, 12, 4, 7, 10 |
| 3 | **1, 5, 8, 39**, 3, 6, 9, 12, 4, 7, 10 |
| 4 | **1, 3, 5, 8, 39**, 6, 9, 12, 4, 7, 10 |
| 5 | **1, 3, 5, 6, 8, 39**, 9, 12, 4, 7, 10 |
| 6 | **1, 3, 5, 6, 8, 9, 39**, 12, 4, 7, 10 |
| 7 | **1, 3, 5, 6, 8, 9, 12, 39**, 4, 7, 10 |
| 8 | **1, 3, 4, 5, 6, 8, 9, 12, 39**, 7, 10 |
| 9 | **1, 3, 4, 5, 6, 7, 8, 9, 12, 39**, 10 |
| 10 | 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39 |

Ta cứ làm vậy với các số khác, kết quả cho ra được:

1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39

* + 1. Sắp xếp nổi bọt

Thuật toán này lặp qua dãy, hoán đổi các phần tử sai thứ tự, lặp lại liên tục đến khi không còn hoán đổi

|  |  |
| --- | --- |
| Số lần lặp | Dãy số được sắp xếp theo từng bước |
| 1 | 8, 5, 1, 3, 6, 9, 12, 4, 7, 10, 39 |
| 2 | 5, 1, 3, 6, 8, 9, 4, 7, 10, 12, 39 |
| 3 | 1, 3, 5, 6, 8, 4, 7, 9, 10, 12, 39 |
| 4 | 1, 3, 5, 6, 4, 7, 8, 9, 10, 12, 39 |
| 5 | 1, 3, 5, 4, 6, 7, 8, 9, 10, 12, 39 |
| 6 | 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39 |

1. Code nằm trong file BTCS1 trong folder BTcoso
2. Độ phức tapj:
   * 1. Sắp xếp đổi chỗ trực tiếp

Ý tưởng: So sánh từng phần tử với tất cả phần tử phía sau, nếu lớn hơn thì đổi chỗ.

Số lần so sánh:

Vòng lặp ngoài chạy từ i = 0 đến size - 2.

Vòng lặp trong chạy từ j = i + 1 đến size - 1.

Tổng số lần so sánh:

(size−1) + (size−2) +...+ 1 = (size(size−1))2 = O(n2)

Số lần hoán vị: Tối đa cũng là O(n^2)

**Độ phức tạp:** O(n^2)

* + 1. Sắp xếp chọn trực tiếp

Ý tưởng: Tìm phần tử nhỏ nhất trong phần còn lại của mảng và đổi chỗ với phần tử đầu tiên.

Số lần so sánh:

Lần lặp đầu tiên: size - 1 so sánh.

Lần thứ hai: size - 2 so sánh.

Tổng số so sánh:

(size−1) + (size−2) + … + 1 = O(n2)

Số lần hoán vị: Luôn là size−1, tức là O(n).

**Độ phức tạp:**O(n^2)**, nhưng số lần hoán vị ít hơn so với Interchange Sort.**

* + 1. Sắp xếp chèn trực tiếp

Ý tưởng: Giả sử dãy bên trái đã sắp xếp, chèn phần tử tiếp theo vào vị trí đúng.

Số lần so sánh:

Trường hợp xấu nhất (mảng ngược): Mỗi phần tử phải dịch chuyển qua toàn bộ dãy phía trước.

Tổng số lần so sánh:

1 + 2 +...+ (size−1)=O(n^2)

Trường hợp tốt nhất (mảng đã sắp xếp): Mỗi phần tử chỉ so sánh đúng 1 lần, tức là O(n)O(n)O(n).

**Độ phức tạp:** O(n2)O(n^2)O(n2) **(trung bình và xấu nhất),** O(n)O(n)O(n) **(tốt nhất).**

* + 1. Sắp xếp nổi bọt

Ý tưởng: So sánh từng cặp phần tử, nếu sai thứ tự thì đổi chỗ, phần tử lớn nhất sẽ "nổi" lên đầu.

Số lần so sánh:

Vòng lặp ngoài chạy từ 0 đến size - 2.

Vòng lặp trong chạy từ 0 đến size - i - 2.

Tổng số lần so sánh:

(size−1) + (size−2) +...+ 1 = O(n2)

Trường hợp tốt nhất (mảng đã sắp xếp): Nếu không có hoán vị nào xảy ra, chỉ mất O(n)O(n)O(n).

**Độ phức tạp:** O(n2)O(n^2)O(n2) **(trung bình và xấu nhất),** O(n)O(n)O(n) **(tốt nhất).**

Câu 2:Mô phỏng như sau:

1. Sắp xếp nhanh

#### ****Bước 1: Chọn pivot****

Chọn pivot = 10 (phần tử cuối dãy).

Chia dãy thành hai phần:

Nhỏ hơn hoặc bằng 10: [8, 5, 1, 3, 6, 9, 4, 7]

Lớn hơn 10: [12]

Đặt 10 vào đúng vị trí.

**Dãy sau bước 1:**  
8, 5, 1, 3, 6, 9, 4, 7, 10, 12

#### ****Bước 2: Đệ quy trên phần nhỏ hơn 10****

Chọn pivot = 7 → Chia thành [5, 1, 3, 6, 4] và [8, 9]

Đặt 7 vào đúng vị trí.

**Dãy sau bước 2:**  
5, 1, 3, 6, 4, 7, 8, 9, 10, 12

#### ****Bước 3: Đệ quy tiếp****

Phần [5, 1, 3, 6, 4]: Pivot = 4 → Chia thành [1, 3] và [5, 6]

Đặt 4 vào đúng vị trí.

**Kết quả**  
1, 3, 4, 5, 6, 7, 8, 9, 10, 12

1. Sắp xếp trộn

#### ****Bước 1: Chia đôi dãy****

(8, 5, 1, 3, 6) và (9, 12, 4, 7, 10)

#### ****Bước 2: Tiếp tục chia nhỏ****

(8, 5) và (1, 3, 6)

(9, 12) và (4, 7, 10)

#### ****Bước 3: Sắp xếp và trộn lại****

Sắp xếp (8, 5) → (5, 8)

Sắp xếp (1, 3, 6) → (1, 3, 6)

Sắp xếp (9, 12) → (9, 12)

Sắp xếp (4, 7, 10) → (4, 7, 10)

Trộn (5, 8) và (1, 3, 6) → (1, 3, 5, 6, 8)

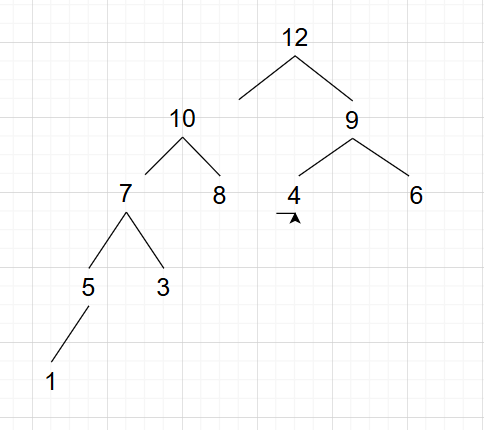
Trộn (9, 12) và (4, 7, 10) → (4, 7, 9, 10, 12)

Trộn hai phần còn lại (1, 3, 5, 6, 8) và (4, 7, 9, 10, 12)

**Kết quả** : 1, 3, 4, 5, 6, 7, 8, 9, 10, 12

1. Sắp xếp cây

Bước 1: dẫy số chuyển về dạng cây



#### ****Bước 2: Loại bỏ phần tử lớn nhất và tái tạo heap****

* Swap 12 với 1, heapify lại → 10 lên đầu.
* Swap 10 với 3, heapify lại → 9 lên đầu.
* Swap 9 với 5, heapify lại → 8 lên đầu.
* Swap 8 với 4, heapify lại → 7 lên đầu.
* Swap 7 với 3, heapify lại → 6 lên đầu.
* Swap 6 với 3, heapify lại → 5 lên đầu.
* Swap 5 với 1, heapify lại → 4 lên đầu.

Kết quả cuối cùng: 1, 3, 4, 5, 6, 7, 8, 9, 10, 12

* 1. Code nằm trong file BTCS2 trong folder BTcoso

Câu 3:code nằm trong file BTCS3 trong folder BTcoso