Insertion sort:

Ý tưởng:

Để thực hiện sắp xếp theo insertion sort, ta sẽ duyệt lần lượt các phần tử tính từ vị trí thứ 2 bên trái của mảng. Sau đó xét lần lượt các phần tử đứng trước phần tử đang duyệt. Nếu phần tử tại đó có giá trị lớn hơn giá trị duyệt thì ta sẽ “đẩy” vị trí của phần tử đang xét lên 1 đơn vị. Việc xét này sẽ lặp lại liên tục cho đến khi vị trí đang xét là bằng 0 thì dừng (trong trường hợp không có giá trị nào lớn hơn) hoặc gặp giá trị lớn hơn giá trị đang duyệt. Ta tiếp tục thực hiện cho đến khi ta duyệt đến phần tử cuối cùng của mảng thì hoàn tất.

Các bước thực hiện:

Ví dụ ta có mảng đầu vào như sau:

𝑎 = {9, 12, 3, 7, 1}

Ở đây tôi dùng ký tự “|” để phân cách 2 phần đã sắp xếp và chưa sắp xếp của mảng. Phần tử có màu đỏ chính là phần tử nhỏ nhất của phần **chưa được sắp xếp**.

Dưới đây là các bước thực hiện thuật toán:

|  |  |  |
| --- | --- | --- |
| Giai đoạn | Mảng 𝑎 | Giải thích |
| 1 | {\*, 9, 12 | **3**, 7, 1} | Ta chạy từ phần tử thứ 2 bên phải của mảng a. Hai phần tử đầu đã sắp xếp cho đến phần tử thứ 3. Ta sẽ chèn các số lớn hơn 3 về sau và đưa 3 về đầu mảng. |
| 2 | {3, \* 9, 12 | **7**, 1} | Bây giờ, ta sẽ đưa phần tử 7 lên trước, và vị trí thích hợp là trước số 9. |
| 3 | {\*, 3, 7, 9, 12 | **1**} | Bây giờ, ta sẽ đưa phần tử 1 lên trước, và vị trí thích hợp là trước số 3. |
| 4 | {1, 3, 7, 9, 12 |} | Tới đây thì thuật toán dừng, ta không cần đưa phần tử 4 lên trước vì nó đã ở đúng vị trí. |

Độ phức tạp thuật toán:

* Theo thời gian (time complexity):
* Trường hợp xấu nhất (worst case): O(n^2)
* Trường hợp trung bình (average case): O(n^2)
* Trường hợp tốt nhất (best case): O(n)
* Bộ nhớ bổ sung (auxiliary space):
* O(1) cho mọi trường hợp

Shaker sort:

**Shaker Sort:**

**Ý tưởng:**

Vì shaker sort là một cải tiến của bubble sort, nên cách thức hoạt động của nó sẽ có phần tương tự giống với bubble sort.

Shaker sort sẽ sử dụng hai vòng lặp để sắp lần lượt từ trái sang phải và từ phải sang trái theo cơ chế sau:

* Ở vòng lặp thứ nhất duyệt mảng từ trái sang phải, ta sẽ so sánh các phần tử liền kề và nếu phần tử bên trái lớn hơn phần tử bên phải thì hai giá trị sẽ hoán chỗ cho nhau. Cứ thế tiếp tục cho tới vòng lặp cuối cùng, số lớn nhất sẽ nằm ở cuối mảng.
* Ở vòng lặp thứ hai duyệt mảng từ phải sang trái, và cách của nó sẽ tương tự như vòng lặp thứ nhất. Phần tử nhỏ nhất sẽ nằm ở đầu mảng khi kết thúc vòng lặp.

**Các bước thực hiện:**

Ví dụ ta có mảng đầu vào như sau:

𝑎 = {9, 12, 3, 7, 1}

Ở đây, tôi đánh dấu phần tử có màu đỏ chính là hai phần tử đang được so sánh. Tôi dùng ký tự “|” để phân cách phần đã sắp xếp và chưa sắp xếp của mảng, sẽ có 2 ký tự “|” vì sẽ có 2 phần đã sắp xếp (là phần đầu mảng và cuối mảng).

Dưới đây là các bước thực hiện thuật toán:

|  |  |  |
| --- | --- | --- |
| Lần lặp | Mảng 𝑎 | Giải thích |
| 1 | { |9, **12**, **3**, 7, 1| } | 12 và 3 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { | 9, 3, **12**, **7**, 1| } | 12 và 7 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { | 9, 3, 7, **12**, **1**| } | 12 và 1 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { | 9, 3, 7, 1 | 12 } | Ta hoàn thành việc duyệt xuôi từ đầu về cuối, lúc này ta thêm 12 vào phần đã sắp xếp ở sau. Và ta bắt đầu duyệt theo thứ tự ngược lại |
| 1 | { | 9, 3, **7**, **1** | 12 } | 7 và 1 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { | 9, **3**, **1**, 7 | 12 } | 3 và 1 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { | **9**, **1**, 3, 7 | 12 } | 9 và 1 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 1 | { 1 | 9, 3, 7 | 12 } | Ta hoàn thành việc duyệt ngược từ cuối về đầu, lúc này ta thêm 1 vào phần đã sắp xếp ở sau. Ta kết thúc lần lặp 1 |

|  |  |  |
| --- | --- | --- |
| 2 | { 1 | **9**, **3**, 7 | 12 } | 9 và 3 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 2 | { 1 | 3, **9**, **7** | 12 } | 9 và 7 đang đứng sai vị trí, ta sẽ hoán vị 2 phần tử này |
| 2 | { 1 | 3, 7 | 9, 12 } | Ta hoàn thành việc duyệt xuôi từ đầu về cuối, lúc này ta thêm 9 vào phần đã sắp xếp ở sau. Và ta bắt đầu duyệt theo thứ tự ngược lại |
| 2 | { 1 | 3, 7 | 9, 12 } | 3 và 7 đã đứng đúng vị trí, ta không làm gì. |
| 2 | { 1, 3 | 7 | 9, 12 } | Ta hoàn thành việc duyệt ngược từ cuối về đầu, lúc này ta thêm 1 vào phần đã sắp xếp ở sau. Ta kết thúc lần lặp 2 |
| 3 | { 1, 3 || 7, 9, 12 } | Ta hoàn thành việc duyệt xuôi từ đầu về cuối, lúc này ta thêm 3 vào phần đã sắp xếp ở sau.  Lúc này ta ngừng thuật toán vì mảng đã được sắp xếp. |

**Độ phức tạp thuật toán:**

* Theo thời gian (time complexity):
* Trường hợp xấu nhất (worst case): O(n)
* Trường hợp trung bình (average case): O(n^2)
* Trường hợp tốt nhất (best case): O(n^2)
* Bộ nhớ bổ sung (auxiliary space):
* O(1) cho mọi trường hợp

**Heap sort:**

**Ý tưởng:**

Thuật toán Heap sort lấy ý tưởng giải quyết từ cấu trúc heap, cụ thể:

* Ta coi dãy cần sắp xếp là một cây nhị phân hoàn chỉnh, sau đó hiệu chỉnh cây thành dạng cấu trúc heap ( vun đống)
* Dựa vào tính chất của cấu trúc heap, ta có thể lấy được ra phần từ lớn nhất hoặc nhỏ nhất của dãy, phần tử này chính là gốc của heap. Giảm số lượng phần tử của cây nhị phân và tái cấu trúc heap.
* Đưa phần tử đỉnh heap về đúng vị trí của dãy ở cuối mảng, sau đó giảm số lượng phần tử của mảng (không xét tới phần tử cuối nữa)
* Tái cấu trúc heap và lặp lại việc lấy phần tử gốc của cấu trúc heap cho tới khi danh sách ban đầu chỉ còn 1 phần tử. Đưa phần tử này về đúng vị trí và kết thúc thuật toán.

Ta phải thực hiện tái cấu trúc heap, vun lại đống bởi vì sau khi lấy ra phần tử gốc heap, cấu trúc heap không còn nữa.

**Các bước thực hiện:**

Ví dụ ta có mảng đầu vào như sau:

𝑎 = {12, 3, 1, 9, 7}

Ở đây, khi cài đặt heap bằng mảng, ta sẽ xem phần tử đứng ở vị trí 𝑖 sẽ có con trái đứng tại vị trí 2 ∗ 𝑖 + 1 và con phải tại vị trí 2 ∗ 𝑖 + 2.

Dưới đây là các bước thực hiện thuật toán ở giai đoạn dựng max-heap. Phần tử màu đỏ chính là phần tử đang được xét, phần tử màu xanh chính là con của phần tử đang xét.

|  |  |  |
| --- | --- | --- |
| **Vị trí đang xét** | Mảng 𝑎 | Giải thích |
| 1 | {12, **3**, 1, **9**, **7**} | Ta sẽ xét các phần tử từ vị trí n / 2 – 1 về trước vì các phần tử sau vị trí này không có con nên ta không cần xét. Phần tử 3 có 2 con là 9 và 7, do ta đang dựng max – heap nên ta cần đưa 9 lên vị trí 3. |
| 0 | {**12**, **9**, **1**, 3, 7} | Ta xét phần tử ở vị trí 0, phần tử này có 2 con có giá trị là 9 và 1 đều nhỏ hơn 12, vì vậy ta không thực hiện gì. Tới đây ta hoàn tất giai đoạn xây dựng max-heap. |

Tiếp theo là giai đoạn 2, ở đây tôi sẽ dùng ký tự “|” để phân tách mảng thành 2 phần, phần max-heap và phần đã được sắp xếp.

|  |  |  |
| --- | --- | --- |
| **Lần lặp** | Mảng 𝑎 | Giải thích |
| 1 | {**12**, 9, 1, 3, 7 |} | Đầu tiên ta đem phần tử 12 về cuối mảng, sau đó đem phần tử 7 lên thay thế phần tử 12. |
| 1 | {**7**, **9**, **1**, 3, | 12} | Ta sẽ đẩy phần tử 7 xuống vị trí thích hợp để đảm bảo max heap. |
| 1 | {9, **7**, 1, **3**, | 12} | Tới đây phần tử 7 đã nằm đúng vị trí, ta dừng lần lặp 1. Lưu ý là lúc này con của phần tử này chỉ có 1, vì phần tử 12 đã nằm ngoài vùng quản lý hiện tại. |
| 2 | {**9**, 7, 1, 3, | 12} | Ta đưa 9 về sau, đem phần tử 3 lên thay 9. |
| 2 | {**3**, **7**, **1** | 9, 12} | Ta cần đưa phần tử 3 tới vị trí thích hợp, ta sẽ thay thế vị trí của phần tử 3 và 7. |
| 2 | {7, **3**, 1 | 9, 12} | Tới đây phần tử 3 đã nằm đúng vị trí, ta dừng lần lặp 2. |
| 3 | {**7**, 3, 1 | 9, 12} | Ta đưa 2 về sau, đem phần tử 0 lên thay 2. |
| 3 | {**1**, **3**, | 7, 9, 12} | Ta cần đưa phần tử 0 tới vị trí thích hợp, ta sẽ thay thế vị trí của phần tử 1 và 3. |
| 3 | {3, **1**, | 7, 9, 12} | Phần tử 1 đã nằm đúng vị trí, ta kết thúc lần lặp 3. |
| 4 | {**3**, 1, | 7, 9, 12} | Ta đưa phần tử 3 về sau, đem phần tử 1 lên thay. |
| 4 | {1, 3 | 7, 9, 12} | Phần tử 1 đã nằm đúng vị trí, ta kết thúc vòng lặp 4 và kết thúc thuật toán. |

**Độ phức tạp thuật toán:**

* Theo thời gian (time complexity):
* Trường hợp xấu nhất (worst case): O(nlogn)
* Trường hợp trung bình (average case): O(nlogn)
* Trường hợp tốt nhất (best case): O(nlogn)
* Bộ nhớ bổ sung (auxiliary space):
* O(1) cho mọi trường hợp

**Quick sort:**

**Ý tưởng:**

Quick sort là một thuật toán chia để trị, thuật toán này hoạt động một cách đệ quy như sau:

* Nếu mảng chỉ có 1 phần tử thì ta không làm gì.
* Ngược lại, ta chọn một phần tử pivot (gọi là phần tử chốt), ta chia mảng ra làm 2 phần, một phần gồm các phần tử bé hơn phần tử chốt (nằm bên trái), và các phần tử còn lớn hơn phần tử chốt (nằm bên phải).

**Các bước thực hiện:**

Ví dụ ta có mảng đầu vào như sau:

𝑎 = {12, 3, 1, 9, 7}

Ở đây ta dùng các cặp ngoặc “[]” để thể hiện các mảng đang được gọi đệ quy. Màu đỏ là phần tử chốt.

Dưới đây là các bước thực hiện thuật toán:

|  |  |
| --- | --- |
| Mảng 𝑎 | Giải thích |
| [12, 3, 1, 9, **7**] | Chọn 7 làm phần tử chốt. Phân làm mảng [3, 1] và  [12, 9] |
| [[**3**, 1], 7, [12, 9]] | Ở [3, 1] ta chọn [3] làm phần tử chốt. Phân ra 2 mảng [1] và [3], tới đây ta thực hiện xong một phần. |
| [1, 3, 7, [**12**, 9]] | Ta chọn 12 làm phần tử chốt, phân làm 2 mảng [9] và [12]. Ở đây hai mảng này đều chỉ còn 1 phần tử nên ta không làm gì. |
| [1, 3, 7, 9, 12] | Phần tử 12 đã nằm đúng vị trí. Mảng sắp xếp hoàn tất. Kết thúc thuật toán. |

**Độ phức tạp thuật toán:**

* Theo thời gian (time complexity):
* Trường hợp xấu nhất (worst case): O(n^2)
* Trường hợp trung bình (average case): O(n log(n))
* Trường hợp tốt nhất (best case): O(n log(n))
* Bộ nhớ bổ sung (auxiliary space):
* O(n log(n)) cho mọi trường hợp

**Counting Sort:**

**Ý tưởng:**

Thuật toán counting sort là một thuật toán đếm số lượng phần tử từ một mảng cần sắp xếp, từ mảng đếm đó ta có thể tính toán và trả về một mảng đã sắp xếp

**Các bước thực hiện:**

Ví dụ ta có mảng đầu vào như sau:

𝑎 = {9, 3, 1, 9, 7}

Đầu tiên ta lấy giá trị lớn nhất của mảng – giá trị bé nhất của mảng + 1 (vì thứ tự trong mảng tính từ 0) chính là số lượng phần tử mà mảng đếm cần

Ta sẽ có mảng đếm: count[9 – 1 + 1] = 9 và khởi tạo cho output giá trị 0 ở mọi phần tử

count[0] = 1; count[2] = 1; count[6] = 1; count[8] = 2

Từ đó ta có count = {1, 0, 2, 0, 0, 0, 1, 0, 2}

Sau đó ta lấy tổng các phần tử đứng trước cộng vào phần tử đứng sau trong mảng:

count = {1, 1, 3, 3, 3, 3, 4, 4, 6}

Bây giờ tạo một mảng mới là output để sắp xếp các phần tử trong mảng, các giá trị được sắp xếp bằng cách: các phần tử trong mảng a sẽ truyền vào đúng vị trí là count[a[i] – giá trị min] - 1 với i là vị trí của phần tử trong a.

Từ đó ta có được mảng sắp xếp:

𝑎 = {1, 3, 7, 9, 9}

**Độ phức tạp thuật toán:**

* Theo thời gian (time complexity):
* Trường hợp xấu nhất (worst case): O(n + k)
* Trường hợp trung bình (average case): O(n + k)
* Trường hợp tốt nhất (best case): O(n + k)
* Bộ nhớ bổ sung (auxiliary space):
* O(k) cho mọi trường hợp