## ****. Ý tưởng sắp xếp trộn****

Các thuật toán sắp xếp đơn giản như [Bubble Sort](https://duongdinh24.com/thuat-toan-sap-xep-noi-bot/), [Insertion Sort](https://duongdinh24.com/thuat-toan-sap-xep-chen/) . . . đều không thể xử lý được dữ liệu lớn. Thuật toán sắp xếp trộn lấy ý tưởng từ việc chia để trị để chia nhỏ bài toán thành các bài toán nhỏ hơn, sau đó giải quyết chúng. Từ đó sẽ giúp xử lý dữ liệu lớn một cách tốt hơn, tối ưu về mặt thời gian.

**Ý tưởng đưa ra như sau:**

* Chia danh sách gồm n phần tử chưa được sắp xếp thành n danh sách con, mỗi danh sách chứa một phần tử (danh sách một phần tử được coi là đã sắp xếp).
* Liên tục hợp nhất các danh sách con để tạo ra các danh sách con được sắp xếp mớ cho đến khi chỉ còn lại một danh sách. Đây sẽ là danh sách được sắp xếp.

**Khi triển khai code, ta sẽ cụ thể hóa bằng các bước:**

**Bước 1:**

* Chia dãy cần sắp xếp thành 2 dãy con
* Từ dãy con thu được lại tiếp tục chia thành 2 dãy con nhỏ hơn nữa
* Quá trình phân chia tiếp tục cho đến khi thu được dãy con chỉ còn duy nhất 1 phần tử.

**Bước 2**:

* Hòa nhập 2 dãy con nhỏ nhất thành dãy con lớn hơn sao cho đúng thứ tự
* Từ hai dãy con lớn hơn lại hòa nhập thành 2 dãy con lớn hơn nữa….
* Quá trình hòa nhập cứ tiếp tục như vậy cho đến khi thu được dãy số ban đầu đã

**mergeSort(arr[], l,  r)**

If r > l

     1. Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:

             middle m = (l+r)/2

     2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:

             mergeSort(arr, l, m)

     3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:

             mergeSort(arr, m+1, r)

     4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):

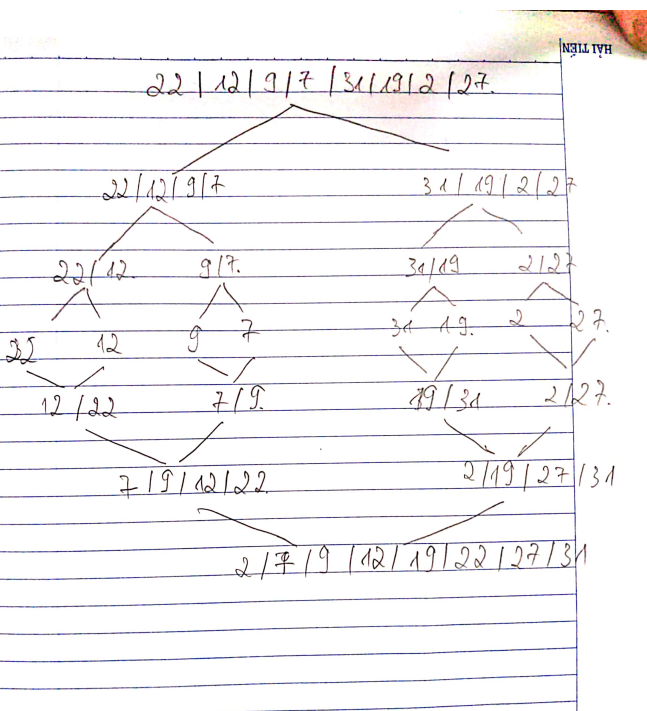
             merge(arr, l, m, r)

được sắp xếp.

swssssss

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STT** | **Thuật toán** | **Tốt nhất** | **Trung bình** | **Xấu nhất** | **Bộ nhớ** | **Stable** |
| 1 | Bubble Sort | O(n) | O(n²) | O(n²) | O(1) | Có |
| 2 | Insertion Sort | O(n) | O(n²) | O(n²) | O(1) | Có |
| 3 | Merge Sort | O(nlogn) | O(nlogn) | O(nlogn) | O(n) | Có |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Merge sort | Insertion Sort | Bubble sort |
| độ hiệu quả | so sánh các phần tử của hai phần tử đã sắp xếp để hợp nhất chúng thành mảng được sắp xếp cuối cùng. | so sánh các phần tử để quyết định vị trí của một phần tử trong mảng đã được sắp xếp một phần. | so sánh các phần tử để đặt các phần tử lớn nhất vào vị trí cuối cùng. |
| độ phức tạp về thời gian | O(nlogn) | O(n^2) | O(n^2) |
| thao tác so sánh | phân bổ thêm bộ nhớ và sao chép dữ liệu | hoán đổi | hoán đổi |
| Ưu Điểm | * Chạy nhanh, độ phức tạp O(N∗logN)O(N∗logN). * Ổn định | * Nếu danh sách đã gần đúng thứ tự, Insertion Sort sẽ chạy rất nhanh. Ví dụ bạn cần sắp xếp Highscore trong game. | * Code đơn giản, dễ hiểu * Không tốn thêm bộ nhớ |
| Nhược Điểm | * Cần dùng thêm bộ nhớ để lưu mảng A. | * Độ phức tạp O(N2)O(N2), không đủ nhanh với dữ liệu lớn. | * Độ phức tạp O(N2)O(N2), không đủ nhanh với dữ liệu lớn. |

* Với mảng đã được sắp xếp, thì Bubble Sort cho tốc độ nhanh nhất do chi phí để biết được đây là mảng có thứ tự của 2 thuật toán trên là O(n).
* Với mảng gần như đã được sắp xếp thì Insertion Sort và Binary Insertion Sort là những sự lựa chọn tốt nhất do số phép hoán đổi phải thực hiện ít.
* Merge Sort có tốc độ ổn định xuyên suốt cả 4 loại dữ liệu đầu vào.
* 

P(x) = xmxm+am-1xm-1+ …+a1x+a0

Q(x) = bnxn+bn-1xn-1+…+b1x+b0

if (m<n) p = m; else p =n;

for (i=0;i<=p;i++)

            c[i]=a[i] + b[i];

if (p<m)

            for (i=p+1;i<=m;i++) c[i] = a[i];

else

            for (i=p+1;i<=n;i++) c[i] = b[i];

while (p>0 && c[p] = 0) p = p-1;

**Quy tắc lấy max**:

            T(n) = O(f(n)+ g(n)) = O(max(f(n), g(n)))

**Quy tắc cộng**:

            T1(n) = O(f(n))                     T2(n) = O(g(n))

            T1(n) + T2(n) = O(f(n) + g(n))

Xét biểu thức:

Phép toán của tính C[i] bao gồm:

c[i]=a[i] + b[i]; //thực hiện p lần (p =min(n,m)

if (p<m)

            for (i=p+1;i<=m;i++) c[i] = a[i];

else

            for (i=p+1;i<=n;i++) c[i] = b[i];

// thực hiện max(n,m)-p lần

* + Tổng số lần tính c[i] là p + (max(n,m) -p) lần
  + Độ phức tạp của phép toán tính c[i] là
  + T1 = O(max(n,m))

Phép toán while (p>0 && c[p] = 0) trải qua p lần duyệt hay min(n,m) lần duyệt

* + T2 = O(min(n,m)

Áp dụng quy tắc cộng:

* + T = T1 + T2 = O(max(n,m)+min(n,m))

Áp dụng quy tắc lấy max:

* + T = O(max(n,m)+min(n,m))

= O(max(max(n,m),min(n,m))

= O(max(n,m)) (max(n,m),min(n,m): chính là n or m))