CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Bài 02: Các thuật toán sắp xếp cơ bản

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

Ngày 15 tháng 9 năm 2025

Nội dung

Bài toán sắp xếp

Định nghĩa

Sắp xếp (*Sorting*) là quá trình tổ chức lại các dữ liệu theo thứ tự giảm dần hoặc tăng dần (*ascending or descending order)*

Bài toán sắp xếp (1)

Bài toán sắp xếp

Cho một mảng a gồm n phần tử $a[0], a[1], ..., a[n-1]$. Hãy sắp xếp mảng a theo thứ tự tăng dần hoặc giảm dần về giá trị.

Ví dụ

Mảng ban đầu: $a = \{4, 1, 3, 5, 6\}$
Mảng được sắp tăng dần: $\{1, 3, 4, 5, 6\}$
Mảng được sắp giảm dần: $\{6, 5, 4, 3, 1\}$

Độ quan trọng của thuật toán sắp xếp

40% thời gian hoạt động của máy tính là dành cho việc sắp xếp
– D.Knuth -

Bài toán sắp xếp (2)

Phân loại

- **Sắp xếp trong** (*internal sorting*): các bản ghi lưu trữ trong bộ nhớ trong (hay còn gọi là bảng)
- Sắp xếp ngoài (*external sorting*): các bản ghi lữu trữ ở bộ nhớ ngoài (tệp tin)

Đặc trưng của các thuật toán sắp xếp

- Tại chỗ (*in place*): nếu không gian nhớ phụ mà thuật toán đòi hỏi là O(1), nghĩa là bị chặn bởi hằng số không phụ thuộc vào độ dài của dãy cần sắp xếp
- Ổn định (*stable*): nếu các phần tử có cùng giá trị vẫn giữ nguyên thứ tự tương đối của chúng như trước khi sắp xếp

Bài toán sắp xếp (3)

Hai phép toán cơ bản mà các thuật toán sắp xếp sử dụng

- Đổi chỗ (swap): thời gian thực hiện là O(1)

Bài toán sắp xếp (3)

Hai phép toán cơ bản mà các thuật toán sắp xếp sử dụng

- Đổi chỗ (swap): thời gian thực hiện là O(1)

```
void swap (int a[], int i, int j)
{
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

- So sánh (compare): hàm compareTo(a,b)

Nội dung

Sắp xếp nổi bọt

- Là thuật toán sắp xếp đơn giản nhất
- Nguyên tắc hoạt động
  - Bắt đầu với phần tử đầu tiên, so sánh với phần tử thứ hai. Nếu phần tử đầu tiên lớn hơn thì đổi chỗ nó với phần tử thứ hai
  - Quá trình tiếp tục cho các cặp phần tử liền kề trong dãy
  - Bắt đầu lại với hai phần tử đầu tiên, lặp lại cho đến khi không thể đổi chỗ các cặp phần tử được nữa

Sắp xếp nổi bọt

```
void bubbleSort(int a[]) {
    int i, j, n = a.length;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                // Swap a[j] and a[j + 1]
                swap(a,j,j+1);
            }
        }

}
```

Sắp xếp nổi bọt

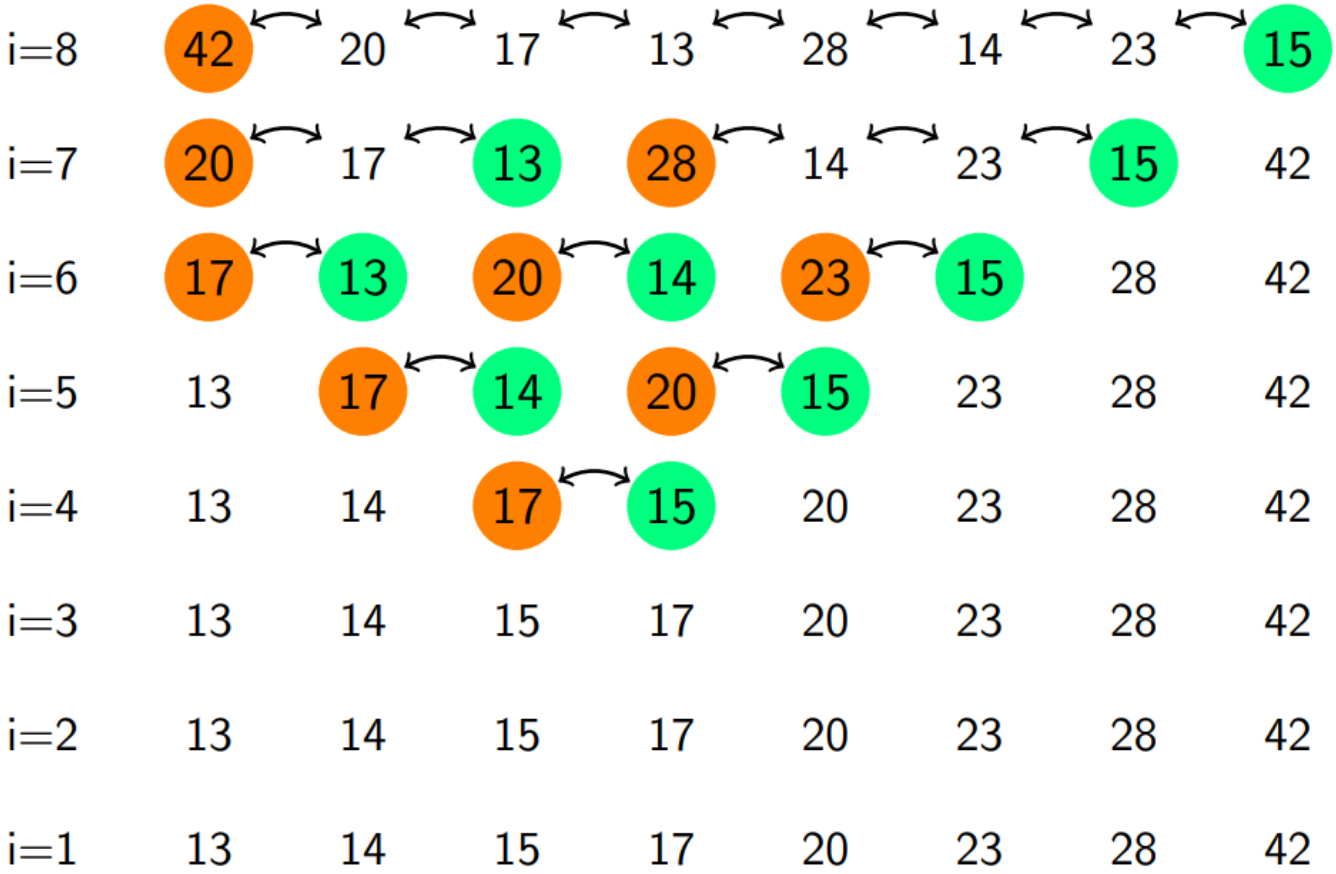
Ví dụ minh hoạ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$

Sắp xếp nổi bọt

Ví dụ minh hoạ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$



Sắp xếp nổi bọt

Phân tích thuật toán

- Trường hợp tốt nhất: 0 đổi chỗ, $\frac{n^2}{2}$ phép so sánh
- Trường hợp tồi nhất: $\frac{n^2}{2}$ phép so sánh và đổi chỗ
- Trường hợp trung bình: $\frac{n^2}{4}$ phép đổi chỗ, $\frac{n^2}{2}$ phép so sánh

Yêu cầu: sửa lại chương trình để trong trường hợp tốt nhất độ phức tạp là O(n)

Sắp xếp nổi bọt

Cải tiến

```
void bubbleSort(int a[]) {
    int i, j, n = a.length;
    for (i = 0; i < n - 1; i++) {
        boolean swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                swap(a,j,j+1);
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }
}
```

Nội dung

Sắp xếp chọn

- Là thuật toán so sánh tại chỗ
- Nguyên tắc hoạt động
  - Tìm giá trị nhỏ nhất của mảng
  - Hoán đổi nó với phần tử ở vị trí đầu tiên
  - Lặp lại quá trình này cho phần còn lại của mảng

Sắp xếp chọn

Cách 1

```
void selectionSort (int a[])
{
    int i, j, n = a.length;
    for (i = 0; i < n-1; i++){
        1) find j: a[j] = min {a[i+1],....,a[n-1]}
        2) swap(a, i, j);
    }


}
```

Sắp xếp chọn

Cách 2

```
void selectionSort (int a[])
{
    int i, j, n = a.length;
    for (i = 0; i < n-1; i++){
        for (int j =i+1; j<n; j++){
            if (a[i] > a[j]){
                swap(a, i, j);
            }
        }
    }


}
```

Sắp xếp chọn

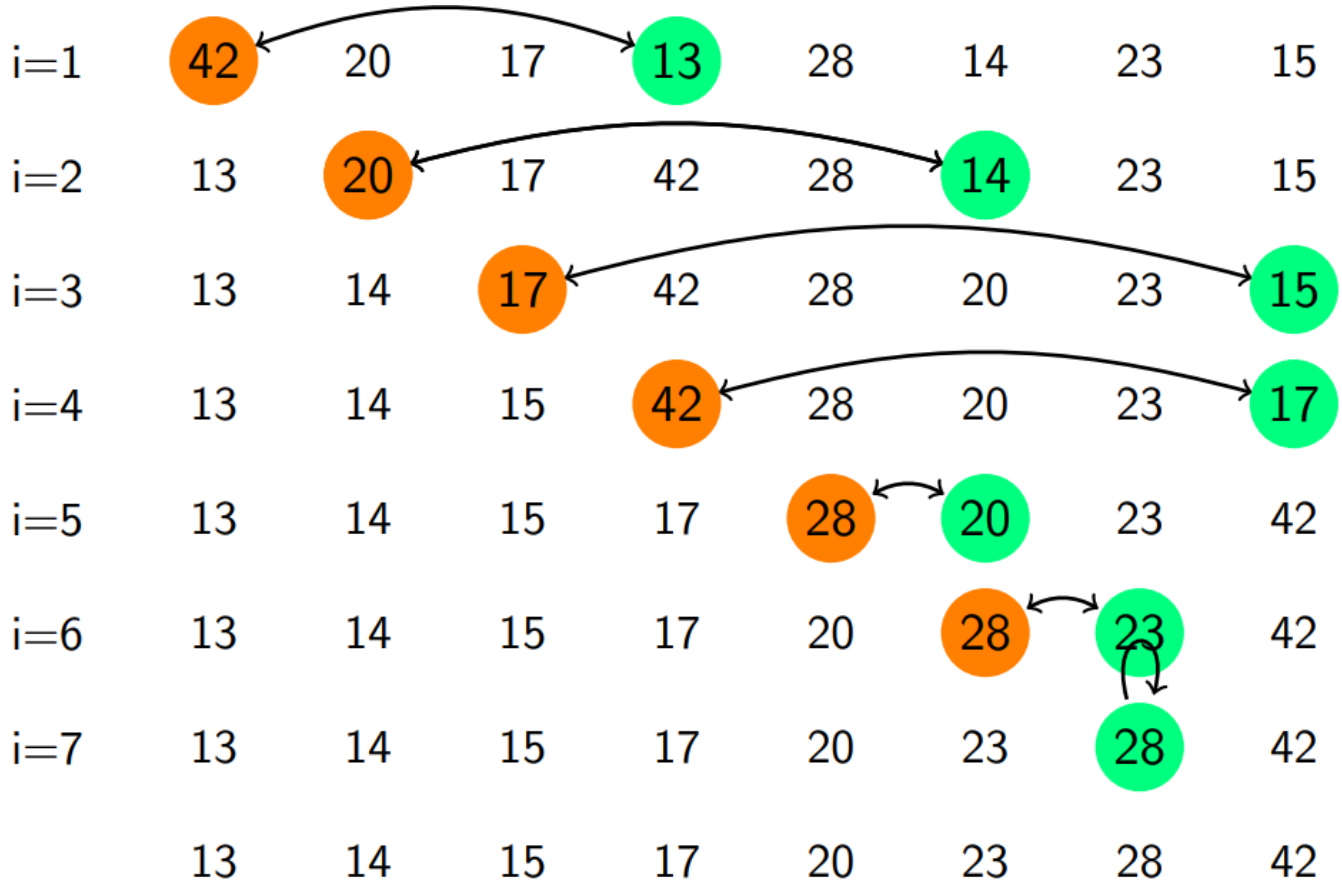
Ví dụ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$

Sắp xếp chọn

Ví dụ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$



Sắp xếp chọn

Phân tích thuật toán

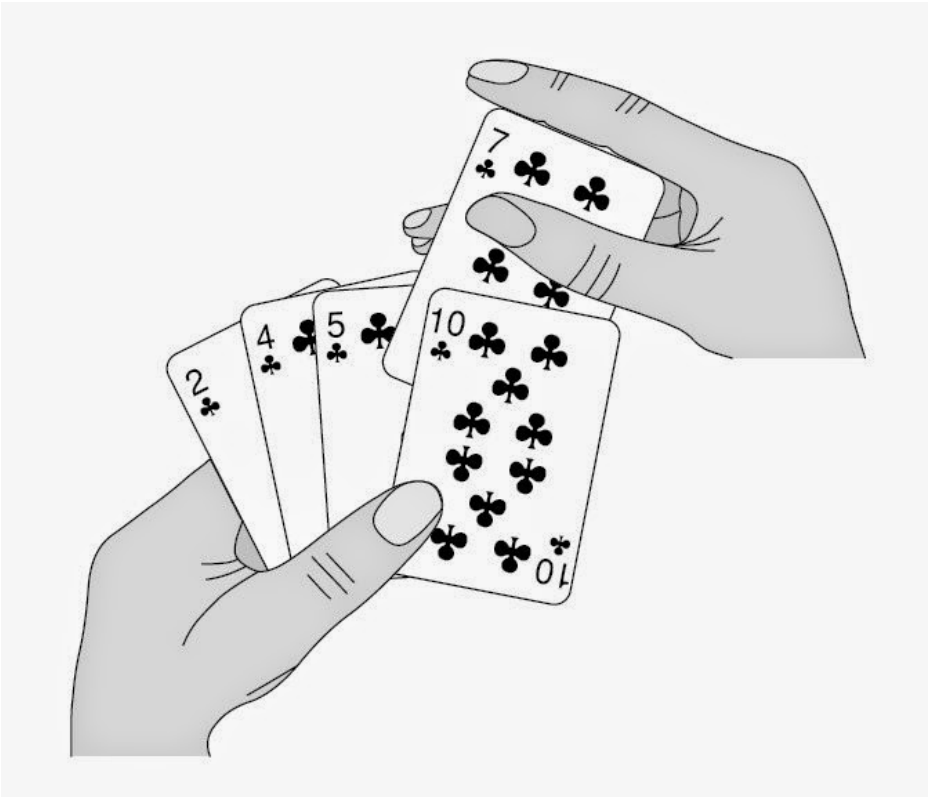
- Trường hợp tốt nhất: 0 đổi chỗ (cách 2), $n-1$ đổi chỗ (cách 1), $\frac{n^2}{2}$ phép so sánh
- Trường hợp tồi nhất: $n-1$ phép đổi chỗ và $\frac{n^2}{2}$ phép so sánh
- Trường hợp trung bình: n phép đổi chỗ và $\frac{n^2}{2}$ phép so sánh

Ưu điểm: số lần đổi chỗ ít

Nội dung

Sắp xếp chèn

- Dựa trên kinh nghiệm của những người chơi bài
- Khi có $i-1$ lá bài được sắp xếp ở trên tay. Nếu rút thêm được lá bài thứ i thì sắp xếp lại thế nào?
- Cách làm là: sẽ so sánh lá bài mới với các lá bài thứ $i-1$, $i-2$,... để tìm ra vị trí " thích hợp" và "chèn vào"



Sắp xếp chèn

Shift Elements

Di chuyển các phần tử bên trái vị trí i để tìm vị trí đúng cho $a[i]$

```
void shiftElement(int a[], int j) {
    int value = a[j];
    while ((j > 0) && (a[j-1] > value)) {
        a[j] = a[j-1];
        j--;
    }
    a[j] = value;
}
```

Sắp xếp chèn

```
void insertionSort(int a[], int n) {
    int i;
    for (i = 1; i < n; i++) {
        if (a[i] < a[i-1]) {
            shiftElement(a, i);
        }
    }
}
```

Sắp xếp chèn

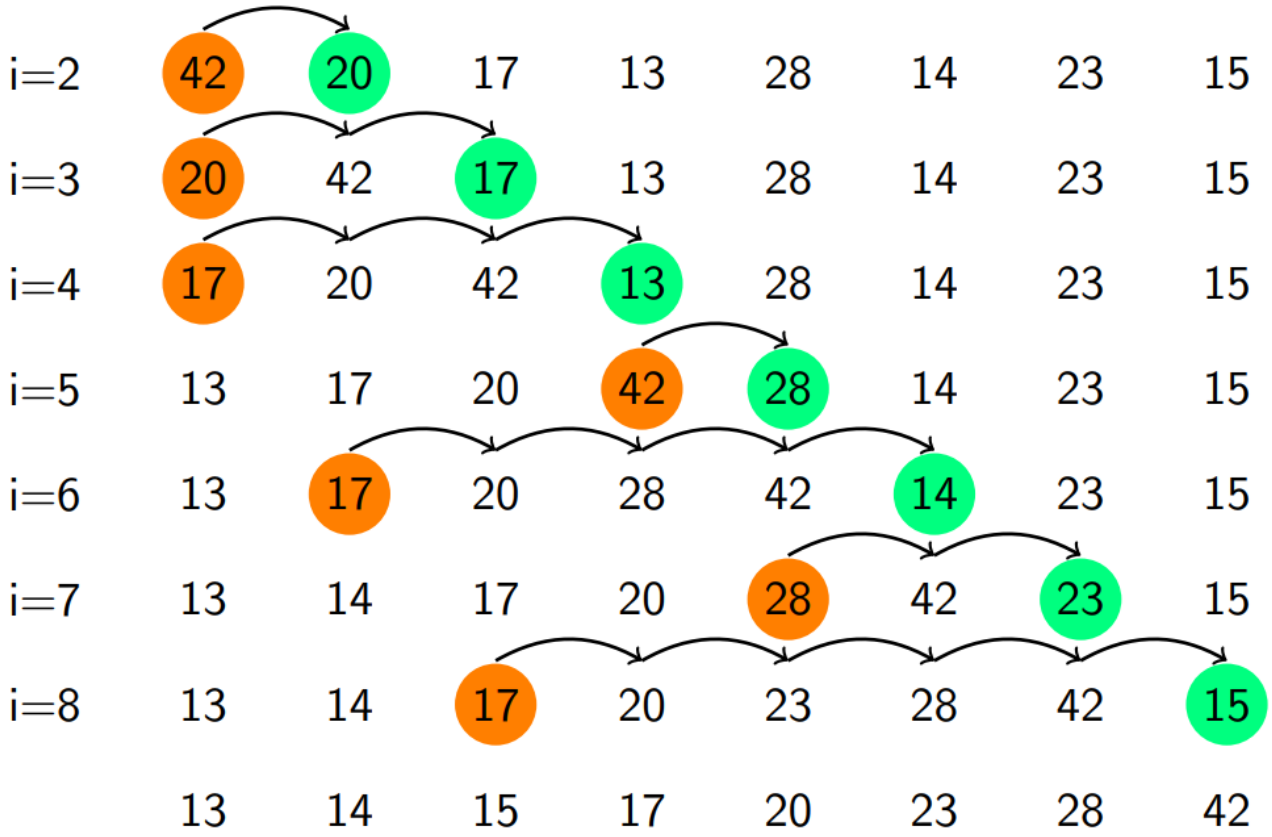
Ví dụ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$

Sắp xếp chèn

Ví dụ

Minh hoạ với dãy phần tử $a = \{42, 20, 17, 13, 28, 14, 23, 15\}$



Sắp xếp chèn

Phân tích thuật toán

- Sắp xếp chèn là tại chỗ và ổn định.
- Thời gian của thuật toán
  - Trường hợp tốt nhất: 0 có hoán đổi (dãy cho vào đã được sắp xếp)
  - Trường hợp tồi nhất: có $\frac{n^2}{2}$ hoán đổi và so sánh, khi dạy đầu vào có thứ tự ngược với chiều cần sắp xếp
  - Trường hợp trung bình: cần $\frac{n^2}{4}$ hoán đổi và so sánh
- Thuật toán tốt với dãy đã *gần được sắp xếp*, nghĩa là phần tử đưa vào gần với vị trí cần sắp xếp

Tổng kết ba thuật toán sắp xếp cơ bản

Thuật toán	**Insertion**	**Bubble**	**Selection**
Số lần so sánh			
Tốt nhất	$O(n)$	$O(n)$	$O(n^2)$
Trung bình	$O(n^2)$	$O(n^2)$	$O(n^2)$
Tệ nhất	$O(n^2)$	$O(n^2)$	$O(n^2)$
Số lần đổi chỗ			
Tốt nhất	0	0	$O(n)$
Trung bình	$O(n^2)$	$O(n^2)$	$O(n)$
Tệ nhất	$O(n^2)$	$O(n^2)$	$O(n)$

Nội dung

Sắp xếp trộn

Thuật toán

- Trường hợp cơ sở (*Base case*): Nếu dãy chỉ có một phần tử được coi là dãy đã được sắp xếp
- Chia (*Divide)*: Chia bài toán ban đầu kích thước n thành hai bài toán con có $n/2$ phần tử
- Trị (*Conquer*):
  - Sắp xếp mỗi dãy con một cách đệ quy sử dụng sắp xếp trộn
  - Khi dãy chỉ còn một phần tử thì trả lại phần tử này
- Tổ hợp (*Combine*): Trộn hai dãy con được sắp để thu được dãy được sắp gồm tất cả các phần tử của cả hai dãy con

Sắp xếp trộn

```
void merge_sort (int a[], int first, int last){
    if (first < last) {
        int mid = (first + last)/2; // Chia (Divide)
        merge_sort(a, first, mid); // Tri (Conquer)
        merge_sort(a, mid + 1, last); // Tri (Conquer)
        merge(a, first, mid, last); // To hop (Combine)
    }
}
```

Sắp xếp trộn

```
void merge(int a[], int first, int mid, int last){
    int n1 = mid - first + 1, n2 = last - mid;
    int aL[] = new int[n1];
    int aR[] = new int[n2];
    for (int i = 0; i < n1; ++i)
        aL[i] = a[first + i];
    for (int j = 0; j < n2; ++j)
        aR[j] = a[mid + 1 + j];
    int i = 0, j = 0;
    int k = first;
    // xem trang tiep
```

Sắp xếp trộn

```
    while (i < n1 && j < n2) {
        if (aL[i] <= aR[j]) {
            a[k] = aL[i]; i++;
        }
        else {
            a[k] = aR[j]; j++;
        }
        k++;
    }
    while (i < n1) {
        a[k] =aL[i];
        i++; k++;
    }
    while (j < n2) {
        a[k] = aR[j];
        j++; k++;
    }
}
```

Bài tập

Hãy chứng minh thuật toán sắp xếp trộn có độ phức tạp là *nlogn*

Độ phức tạp

Thời gian tính của phép trộn - merge

- Khởi tạo hai mảng tạm thời: $\Theta(n1 + n2) = \Theta(n)$
- Đưa các phần tử đã trộn đúng thứ tự vào mảng kết quả: có n lần lặp, mỗi lần đòi hỏi thời gian hằng số, do đó thời gian cần thiết để thực hiện là $\Theta(n)$
- Tổng cộng thời gian là $\Theta(n)$

Độ phức tạp (tiếp)

Thời gian tính của sắp xếp trộn - mergeSort

- Chia: Tính mid mất $\Theta(1)$
- Trị: Giải đệ quy hai bài toán con, mỗi bài kích thước $n/2 \Rightarrow 2T(n/2)$
- Tổ hợp: $\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{nếu } n = 1 \\ 2T(n/2) + \Theta(n) & \text{nếu } n > 1 \end{cases}$$

Định nghĩa

Định lý thợ rút gọn: Giả sử $a \geq 1, b > 1, c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ quy:

$$T(n) = aT(\tfrac{n}{b}) + cn^k$$

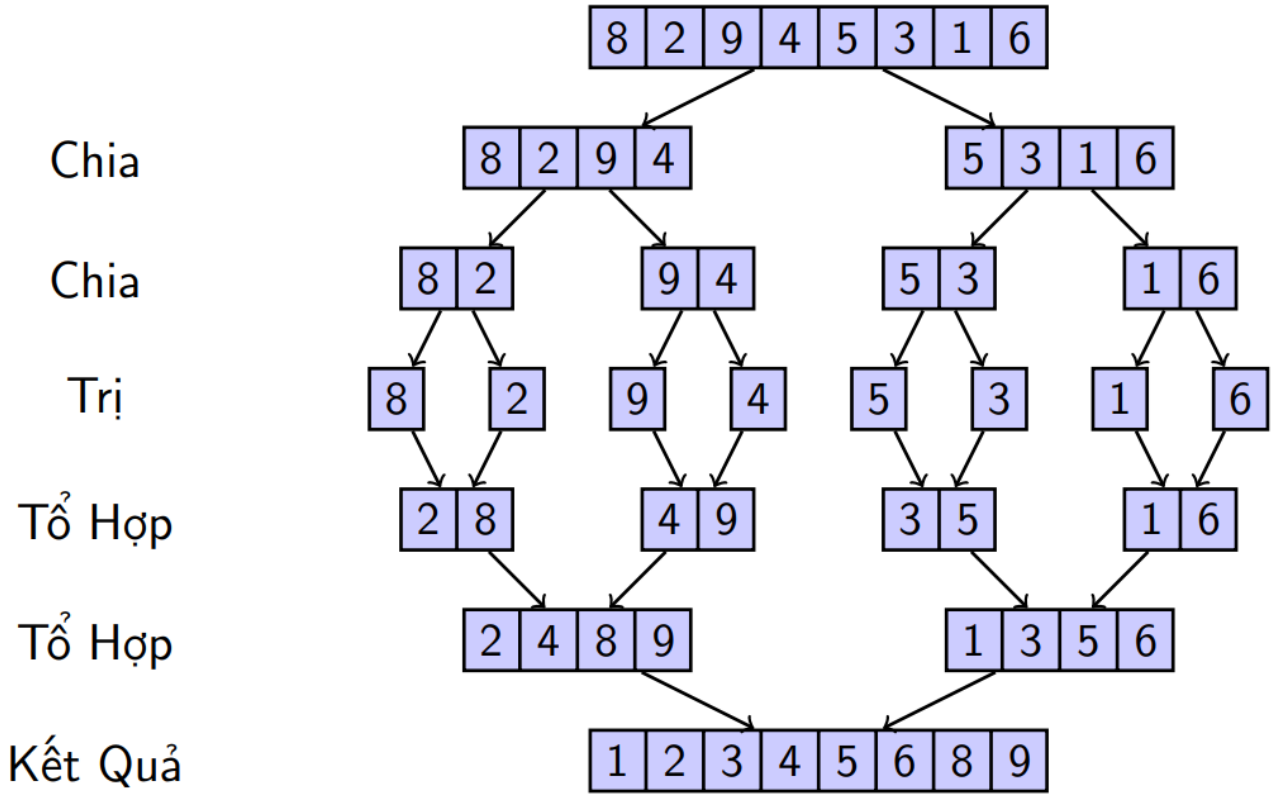
Xác định với $n \geq 0$

1. *nếu $a > b^k$ thì $T(n) = \Theta(n^{log_b^a})$*
2. *nếu $a = b^k$ thì $T(n) = \Theta(n^k log(n))$*
3. *nếu $a < b^k$ thì $T(n) = \Theta(n^k)$*

Sắp xếp trộn

Minh hoạ

Minh hoạ sắp xếp trộn của dãy $\{8, 2, 9, 4, 5, 3, 1, 6\}$



Nội dung

Sắp xếp nhanh

Thuật toán

Thuật toán sắp xếp nhanh được phát triển bởi C.A.R.Hoare vào năm 1960.
Thuật toán cũng được phát triển dựa theo phương pháp chia để trị

- Trường hợp cơ sở (Base Case): Nếu dãy có một phần tử thì nó là phần tử đã được sắp xếp.
- Chia (Divide):
  - Chọn một phần tử trong dãy làm chốt p (pivot)
  - Chia dãy thành hai dãy con: Dãy con trái L gồm các phần tử nhỏ hơn chốt, ngược lại, dãy con phải R gồm các phần tử lớn hơn chốt. Thao tác này gọi là phân đoạn - partition.
  - Trị (Conquer): Lặp lại đệ quy thuật toán với hai dãy con L và R
  - Tổ hợp (Combine): Dãy được sắp xếp là LpR

Sắp xếp nhanh

```
void quick_sort(a, left, right)
{
    if (left < right){
        p = partition (a, left, right);
        quick_sort (a, left, p-1);
        quick_sort (q, p+1, right);
    }
}
```

Sắp xếp nhanh

Một cải tiến mà D.Knuth đề nghị là nên dùng giải thuật sắp xếp khác khi số phần tử không quá lớn $n0 = 9$, ví dụ áp dụng giải thuật sắp xếp chèn

```
void quick_sort(int a[], int left, int right)
{
    if (right - left < n0)
        insertionSort (a, left, right);
    else{
        if (left < right) {
        int p = partition(a, left, right);
        quick_sort(a, left, p - 1);
        quick_sort(a, p + 1, right);
    }
}
```

Sắp xếp nhanh

Thao tác phân đoạn

- Chọn phần tử chốt p
- Chia dãy đã cho thành hai dãy con: Dãy con trái L gồm những phần tử nhỏ hơn chốt và dãy con phải R gồm các phần tử lớn hơn chốt

Thao tác phân đoạn có thể cài đặt tại chỗ, hiệu quả phụ thuộc vào việc chọn chốt p. Có mốt số cách để chọn chốt

- Chọn phần tử trái nhất
- Chọn phần tử phải nhất
- Chọn phần tử ở giữa
- Chọn phần tử trung vị (median) trong ba phần tử đầu, cuối hoặc giữa
- Chọn ngẫu nhiên một phần tử

Sắp xếp nhanh

Thao tác phân đoạn

Hàm `partition (a, left, right)` được xây dựng như sau:

- Đầu vào: mảng a[left...right]
- Đầu ra: Phân bố lại các phần tử của mảng ban đầu dựa vào phần tử chốt p và trả lại chỉ số jp sao cho:
  - $a[jp]$ chứa giá trị chốt p
  - $a[i] \leq a[jp]$ với mọi $left \leq i < jp$
  - $a[j] > a[jp]$ với mọi $jp < j \leq right$

Sắp xếp nhanh

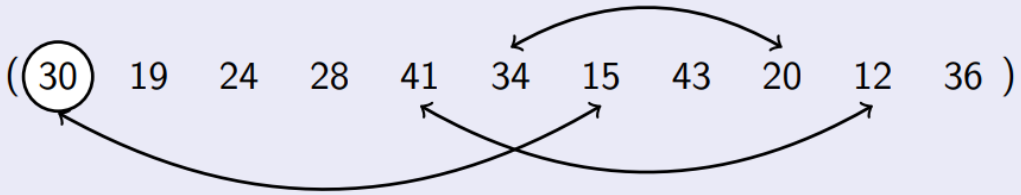
```
int partition(int a[], int left, int right) {
    int i = left, pivot = a[left], j = right;
    while (i < j) {
        while (i <= right && a[i] <= pivot)
            i++;
        while (j >= left && a[j] > pivot)
            j--;
        if (i > j)
            break;
        swap(a, i, j);
    }
    swap(a, left, j);
    return j;
}
```

Sắp xếp nhanh

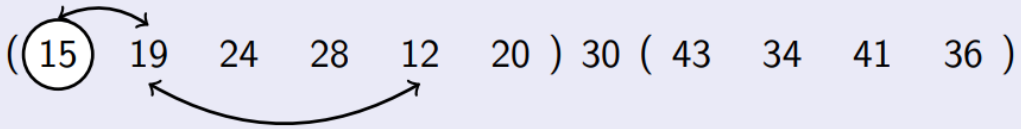
Ví dụ

Thao tác phân đoạn : Phần tử chốt là đứng đầu (tiếp)

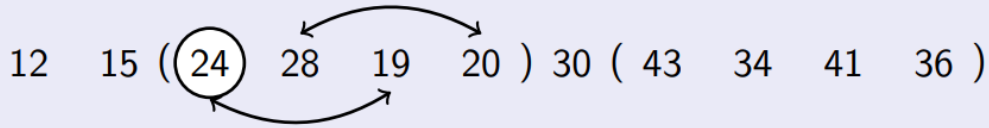
- Bước 1 :



- Bước 2 :



- Bước 3 :

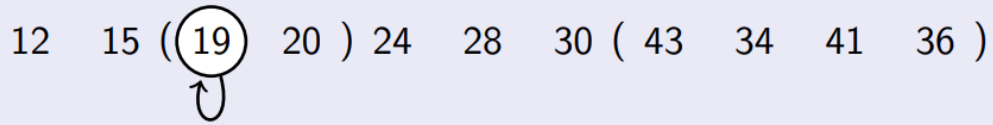


Sắp xếp nhanh

Ví dụ

Thao tác phân đoạn : Phần tử chốt là đứng đầu (tiếp)

- Bước 4 :



- Bước 5 :

- Bước 6 :

- Bước kết thúc :

12 15 19 20 24 28 30 34 36 41 43

Độ phức tạp

Độ phức tạp của sắp xếp nhanh

Thời gian tính của thuật toán sắp xếp nhanh phụ thuộc vào việc phân chia cân bằng (*balanced*) hay không cân bằng (*unbalanced*) và điều đó phụ thuộc vào việc phần tử nào được chọn làm chốt.

- Phân đoạn không cân bằng: một bài toán con có kích thước $n-1$, bài toán còn lại có kích thước 1
- Phân đoạn hoàn hảo: phân đoạn luôn thực hiện dưới dạng phân đôi, như vậy mỗi bài toán con có kích thước khoảng $n/2$
- Phân đoạn cân bằng: việc phân đoạn được thực hiện ở đâu đó quanh điểm giữa, nghĩa là một bài toán con có kích thước $n-k$, bài toán còn lại có kích thước k

Độ phức tạp

Phân đoạn không cân bằng - unbalanced partition

Công thức đệ quy cho thời gian tính trong tình huống này là:

- $T(n) = T(n-1) + T(1) + \Theta(n)$
- $T(0) = T(1) = 1$

Vậy công thức đệ quy cho thời gian tính trong tình huống này là $T(n) = \Theta(n^2)$

Độ phức tạp

Phân đoạn hoàn hảo - perfect partition

Công thức đệ quy cho thời gian tính trong tình huống này là:

- $T(n) = T(n/2) + T(n/2) + \Theta(n) = 2T(n/2) + \Theta(n)$

Vậy độ phức tạp thời gian trong tình huống này là $T(n) = \Theta(nlogn)$

Độ phức tạp

Phân đoạn cân bằng - balanced partition

Giả sử phần tử chốt được chọn ngẫu nhiên trong số các phần tử của dãy đầu vào.

- chốt là phần tử nhỏ nhất trong dãy
- chốt là phần tử nhỏ thứ nhì trong dãy
- ....
- chốt là phần tử lớn nhất trong dãy

Điều đó cũng đúng khi chốt luôn được chọn là phần tử đầu tiên, với giả thiết dãy đầu vào hoàn toàn ngẫu nhiên

Độ phức tạp

Phân đoạn cân bằng - balanced partition

Khi đó thời gian tính trung bình sẽ có công thức
$\Sigma(\textit{thời gian phân đoạn kích thước } i) \times (\textit{xác suất có phân đoạn kích thước } i)$
Khi dãy đầu vào ngẫu nhiên, tất cả các kích thước đồng khả năng xảy ra, xác suất đều $1/n$. Do thời gian phân đoạn kích thước i là:
$T(n) = T(i) + T(n-i-1) + cn$

Độ phức tạp

Phân đoạn cân bằng - balanced partition

$$T(n) = \frac{1}{n}\Sigma_{i=0}^{n-1}(T(i) + T(n-i-1)) + cn \quad (1)$$
$$= \frac{2}{n}\Sigma_{i=0}^{n-1} T(i) + cn$$

Nhân cả hai vế với n được

$$nT(n) = 2\Sigma_{i=0}^{n-1} T(i) + cn^2 \quad (2)$$

Tương tự với $(n-1)$

$$(n-1)T(n-1) = 2\Sigma_{i=0}^{n-2} T(i) + c(n-1)^2 \quad (3)$$

Lấy (2) - (3)

Độ phức tạp

Phân đoạn cân bằng - balanced partition

$$nT(n) - (n-1)T(n-1) = 2\Sigma_{i=0}^{n-1} T(i) + cn^2 - \Sigma_{i=0}^{n-2} T(i) - c(n-1)^2 \quad (4)$$
$$= 2T(n-1) + cn$$

Vậy $nT(n) = (n+1)T(n-1) + cn$ Chia $n * (n+1)$ được

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{c}{n+1} \quad (5)$$
$$= \frac{T(n-2)}{n-1} + \frac{c}{n} + \frac{c}{n+1}$$
$$= ...$$
$$= O(1) + c\Sigma_{i=3}^{n} \frac{1}{i}$$

(6)

Độ phức tạp

Phân đoạn cân bằng - balanced partition

Tổng chuỗi $\sum_{i=1}^{n} \frac{1}{i} \approx ln(n) + \gamma$
với γ là hằng số Euler–Mascheroni[a] $\gamma \approx 0.577$

$$\begin{aligned} \frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{c}{n+1} \\ &= \frac{T(1)}{2} + c\Sigma_{i=3}^{n} \frac{1}{i} \\ &= O(1) + c\Sigma_{i=3}^{n} \frac{1}{i} \\ &= O(1) + O(clogn) \end{aligned} \tag{7}$$

Vậy $T(n) = O(nlogn)$

[a] https://en.wikipedia.org/wiki/Euler%27s_constant