

Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Các thuật toán sắp xếp

Bài Toán Sắp Xếp

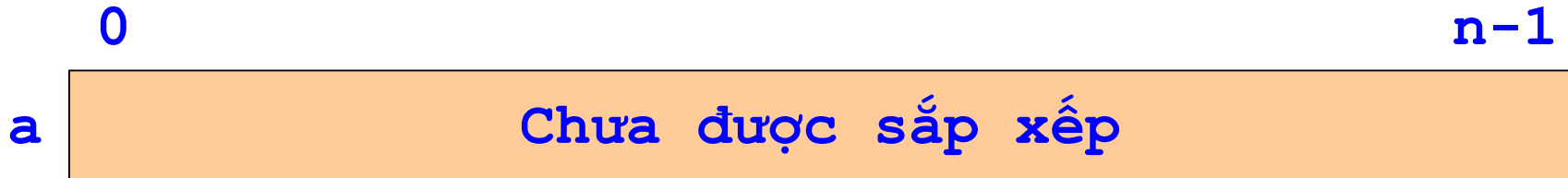
- Bài toán: đặt *các số* vào vị trí
 - Thông thường từ nhỏ nhất đến lớn nhất: *tăng dần*
 - Cũng có thể từ lớn nhất đến nhỏ nhất: *giảm dần*
- Phát biểu bài toán:
 - Cho một mảng **a** có độ dài **n**, sắp xếp lại mảng **a** sao cho
$$\mathbf{a[0] \leq a[1] \leq \dots \leq a[n-1]}$$
 - Quy tắc viết tắt:
 - Ký hiệu **array[i..k]** có nghĩa tất cả các phần tử **array[i]**, **array[i+1]**,..., **array[k]**
 - Cách viết này không đúng trong C
 - Mảng ban đầu có thể viết dưới dạng **a[0..n-1]**

Bài Toán Sắp Xếp

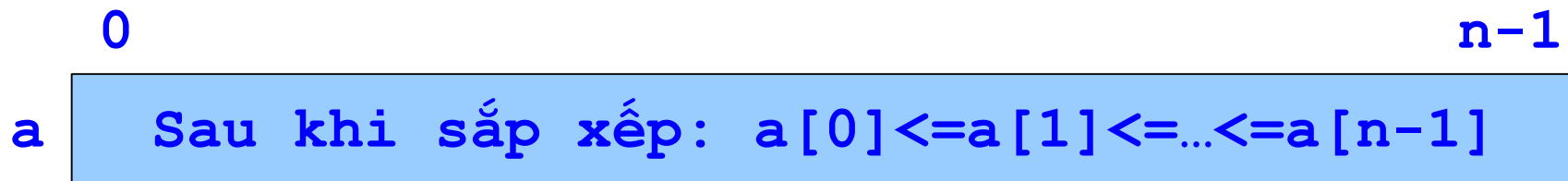
- Có rất nhiều ứng dụng thực tiễn:
 - Sắp xếp số lần truy cập các đường dẫn trong máy tìm kiếm (i.e. Google)
 - Chuẩn bị các danh sách đầu ra
 - Kết nối dữ liệu từ các nguồn khác nhau
 - Là công cụ để giải các bài toán phức tạp hơn
 - Và còn nhiều nữa...
- Các thuật toán sắp xếp đã được nghiên cứu rất nhiều trong nhiều thập kỷ.
- Có rất nhiều cách để sắp xếp lại một mảng, 2 thuật toán nổi bật sẽ được đề cập đến hôm nay.

Bài Toán Sắp Xếp

- Dữ liệu ban đầu:

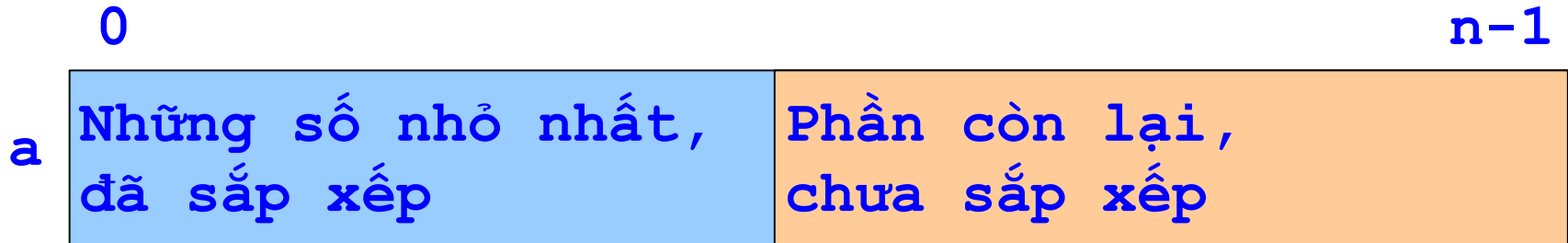


- Mong muốn:
 - Dữ liệu được sắp xếp theo một trật tự nhất định

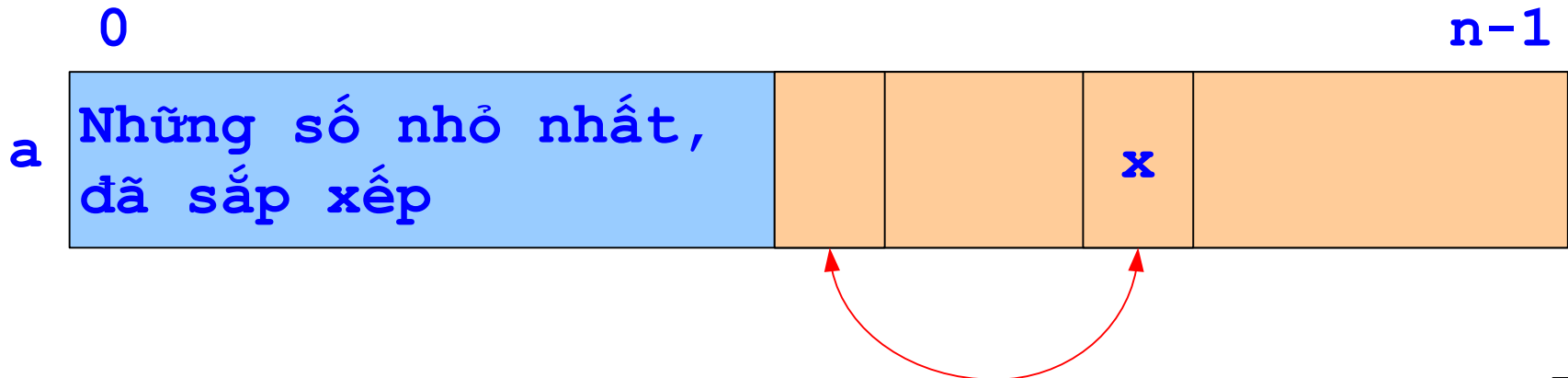


Sắp Xếp Lựa Chọn

- Trạng thái ban đầu:



- Các bước tiến hành:
 - Tìm số nhỏ nhất trong **a[k..n-1]**
 - Hoán đổi vị trí **a[k]** với số nhỏ nhất vừa tìm được



Tìm Số Nhỏ Nhất

```
/* Yield location of smallest element in a[k..n-1] */
/* Assumption: k < n */
/* Returns index of smallest, does not return the
   smallest value itself */

int min_loc (int a[], int k, int n) {
    /* a[pos] is smallest element found so far */
    int j, pos;
    pos = k;
    for (j = k + 1; j < n; j = j + 1)
        if (a[j] < a[pos])
            pos = j;
    return pos;
}
```

Sắp Xếp Lựa Chọn

```
/* Sort a[0..n-1] in non-decreasing order (rearrange  
elements in a so that  $a[0] \leq a[1] \leq \dots \leq a[n-1]$  ) */
```

```
int sel_sort (int a[], int n) {  
    int k, m;  
    for (k = 0; k < n - 1; k = k + 1) {  
        m = min_loc(a, k, n);  
        swap(&a[k], &a[m]);  
    }  
}
```

Ví Dụ

a	3	12	-5	6	142	21	-17	45
---	---	----	----	---	-----	----	-----	----

a	-17	12	-5	6	142	21	3	45
---	-----	----	----	---	-----	----	---	----

a	-17	-5	12	6	142	21	3	45
---	-----	----	----	---	-----	----	---	----

Ví Dụ

a	-17	-5	3	6	142	21	12	45
---	-----	----	---	---	-----	----	----	----

a	-17	-5	3	6	142	21	12	45
---	-----	----	---	---	-----	----	----	----

a	-17	-5	3	6	12	21	142	45
---	-----	----	---	---	----	----	-----	----

Ví Dụ

a	-17	-5	3	6	12	21	142	45
---	-----	----	---	---	----	----	-----	----

a	-17	-5	3	6	12	21	45	142
---	-----	----	---	---	----	----	----	-----

Phân Tích Thuật Toán

- Cần thực hiện bao nhiêu phép toán để sắp xếp **n** số?
 - Trước mỗi lần thực hiện việc hoán đổi vị trí ta cần phải duyệt qua toàn bộ phần mảng chưa được sắp xếp.
 - Độ dài cần duyệt tỷ lệ với độ dài mảng ban đầu là **n** .
 - Cần lặp **n** lần chu kỳ duyệt/hoán đổi.
 - Tổng số bước tỷ lệ với **n^2** .
- Kết luận: sắp xếp lựa chọn khá chậm đối với mảng lớn.

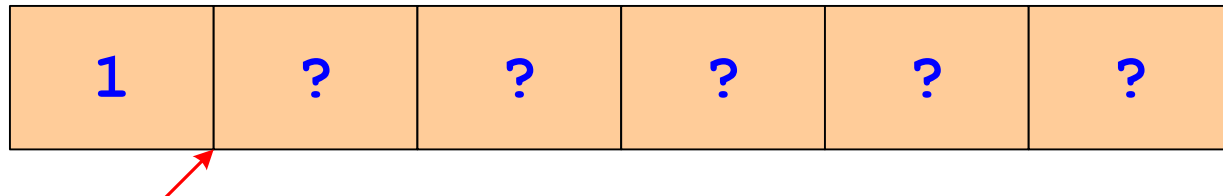
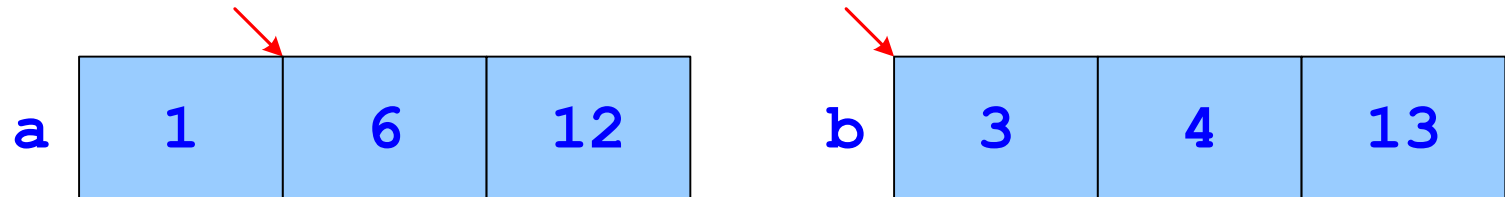
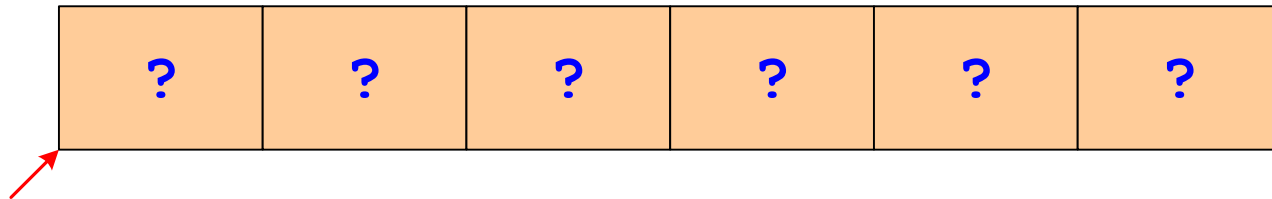
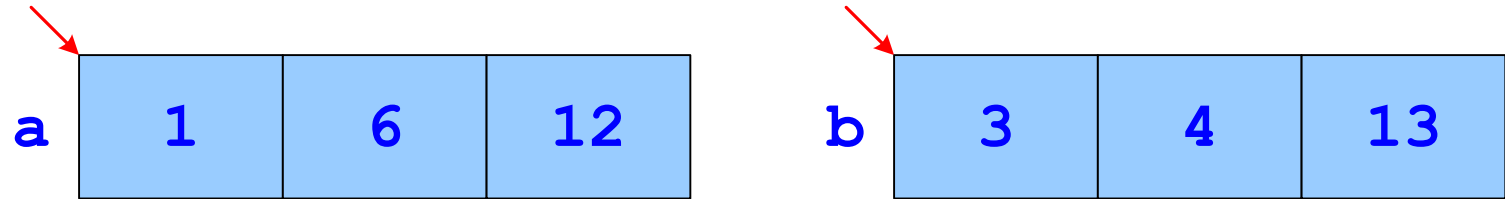
Thuật Toán Ưu Việt Hơn?

- Các thuật toán sắp xếp lựa chọn (selection sort), sắp xếp chèn (insertion sort), sắp xếp nổi bọt (bubble sort) đều cần số bước tỷ lệ với n^2 .
- Một số thuật toán sắp xếp chỉ cần số bước tỷ lệ với $n \log n$
 - Sắp xếp trộn (mergesort)
 - Sắp xếp nhanh (quicksort)
 - ...
- Khi kích thước mảng cần được sắp xếp lớn lên, thời gian cần thiết để chạy thuật toán yêu cầu n^2 bước sẽ lớn hơn rất nhiều thuật toán yêu cầu $n \log n$ bước.

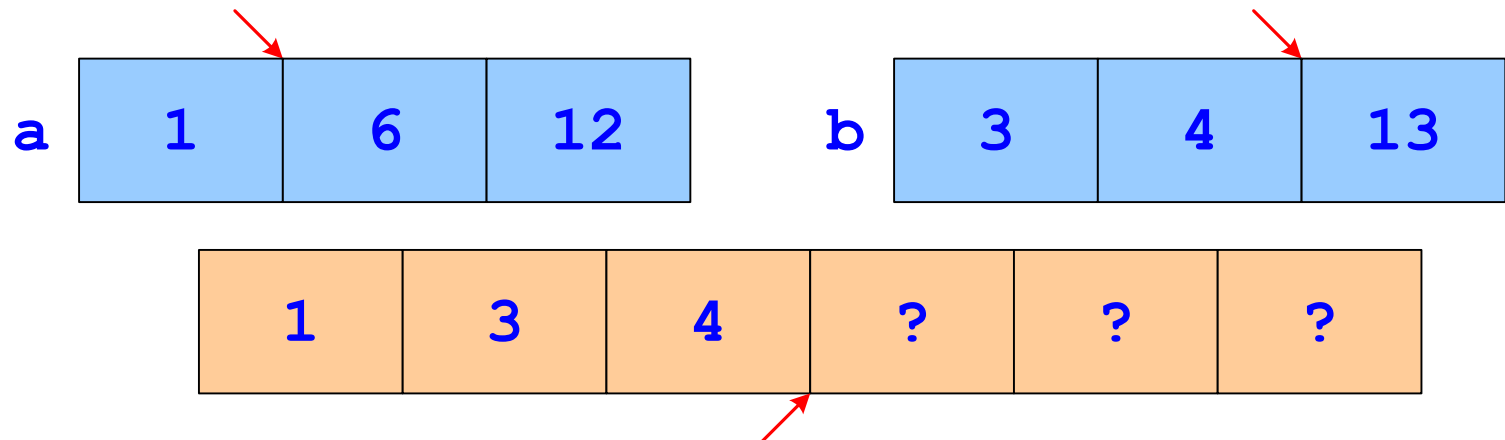
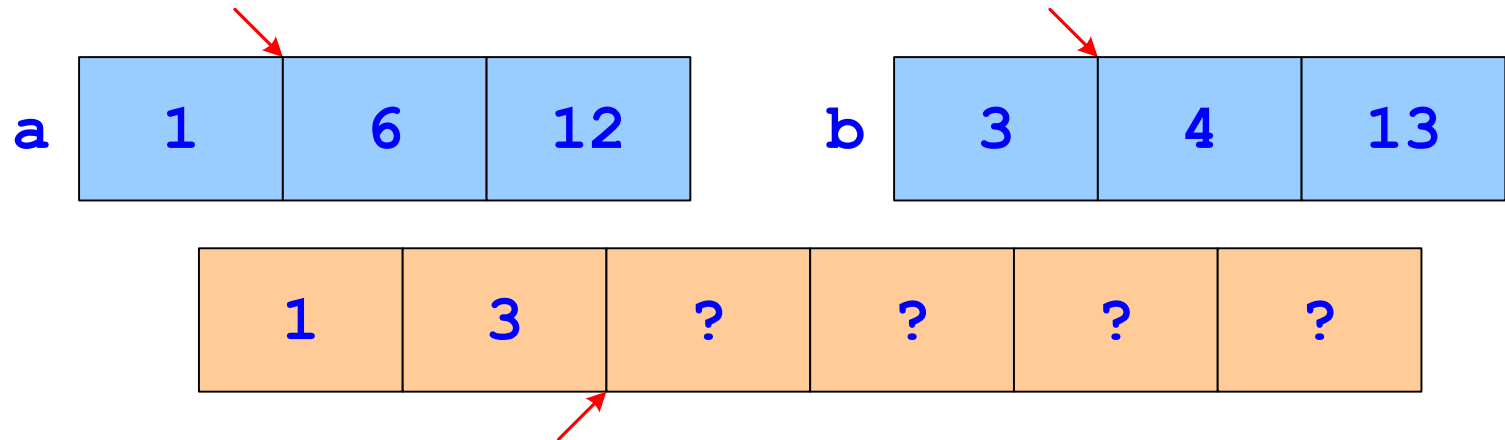
Sắp Xếp Trộn

- Ý tưởng cơ bản:
 - Bắt đầu với các mảng số đã được sắp xếp: các con trỏ chạy nằm ở đầu mảng
 - Sử dụng con trỏ chạy trộn vị trí của từng cặp trong số các mảng đã sắp xếp này để tạo nên một mảng lớn hơn với các thành phần đã được sắp xếp
 - Khi kết thúc trộn hai mảng cuối cùng ta nhận được mảng cần tìm
- Phép toán cơ bản là phép toán trộn.

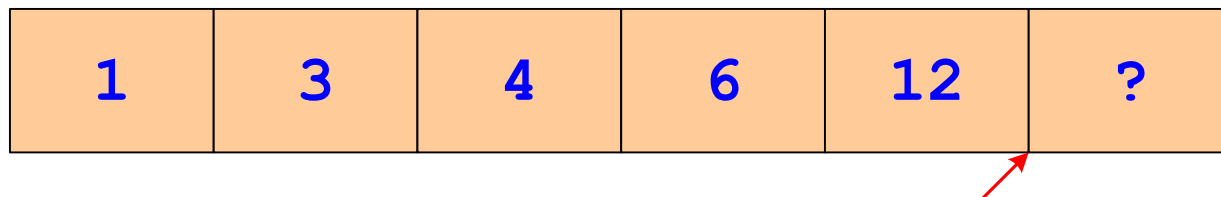
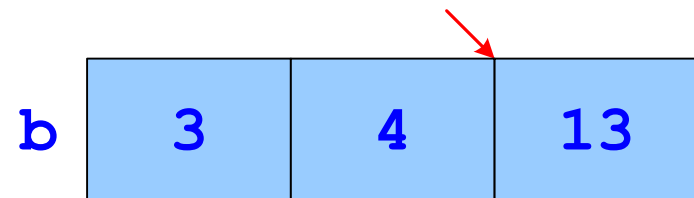
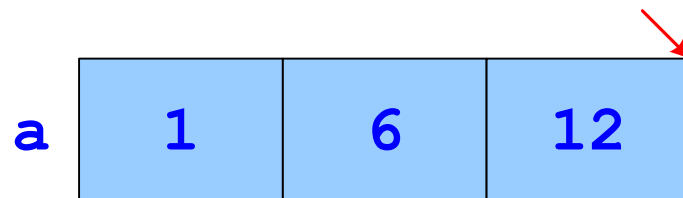
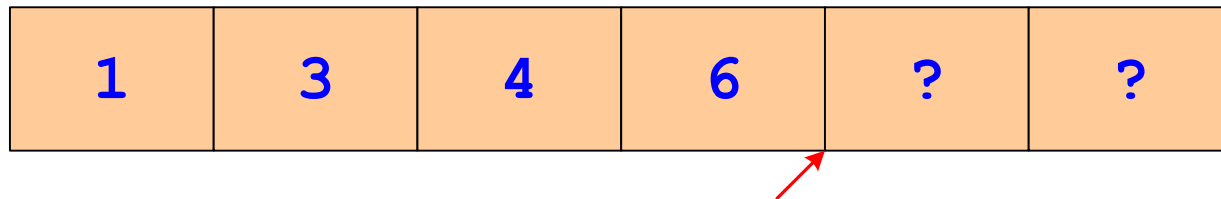
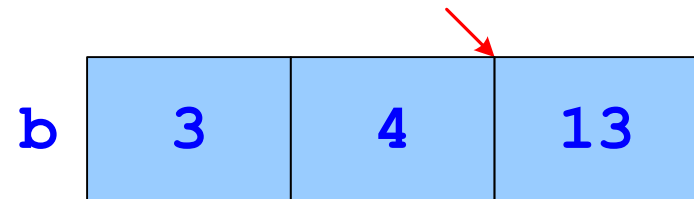
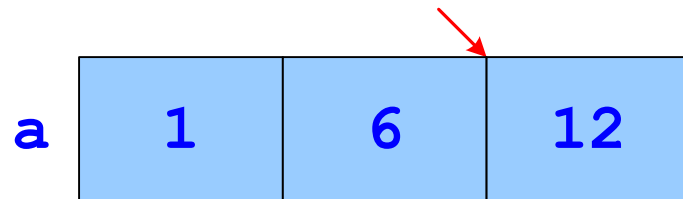
Bài Toán Trộn



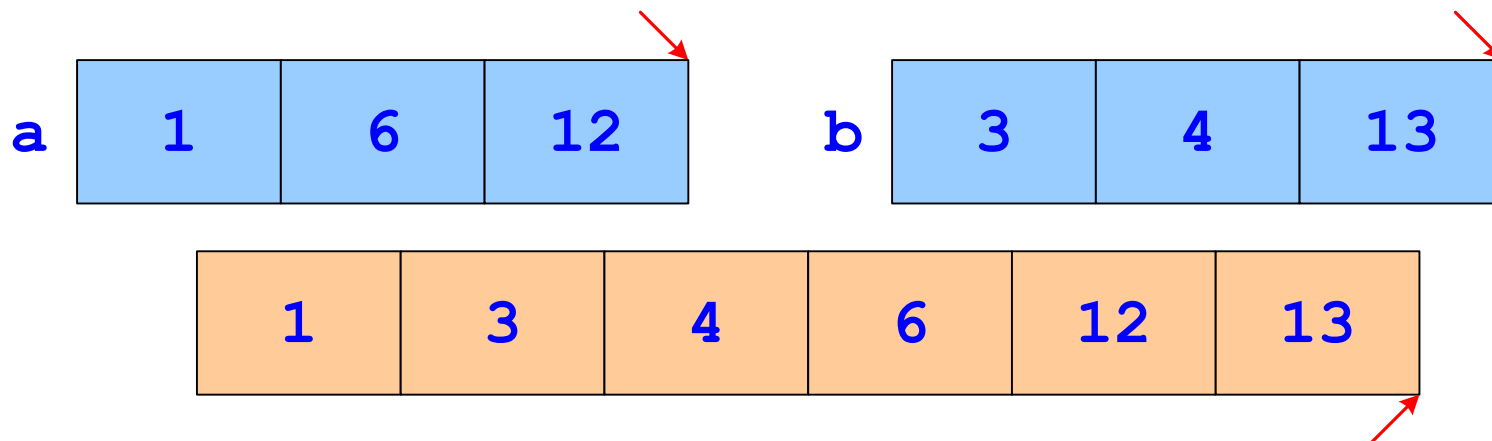
Bài Toán Trộn



Bài Toán Trộn



Bài Toán Trộn

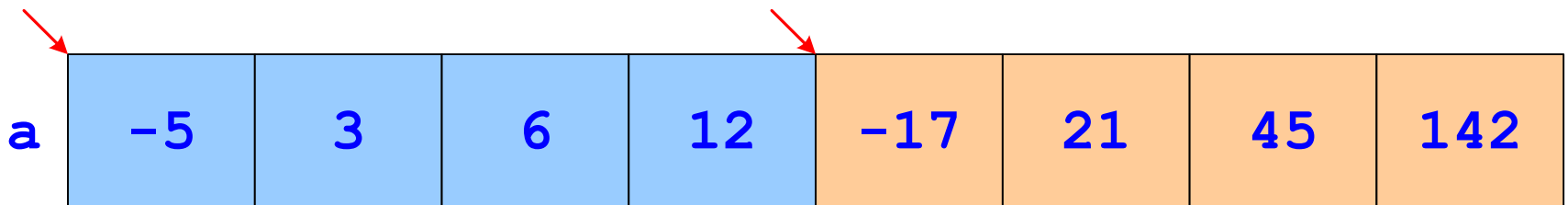
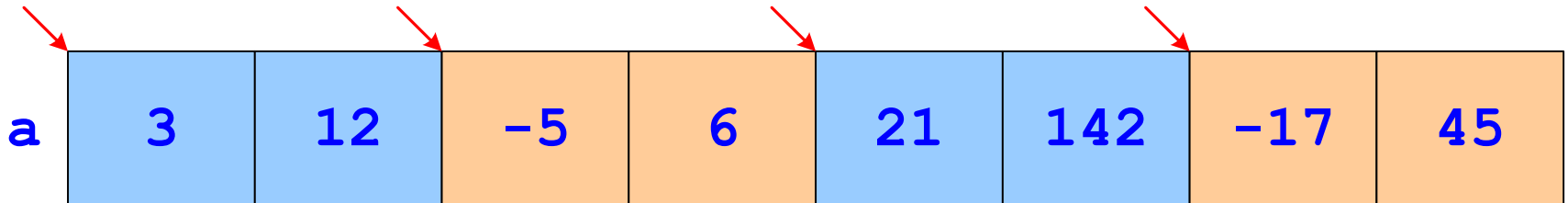
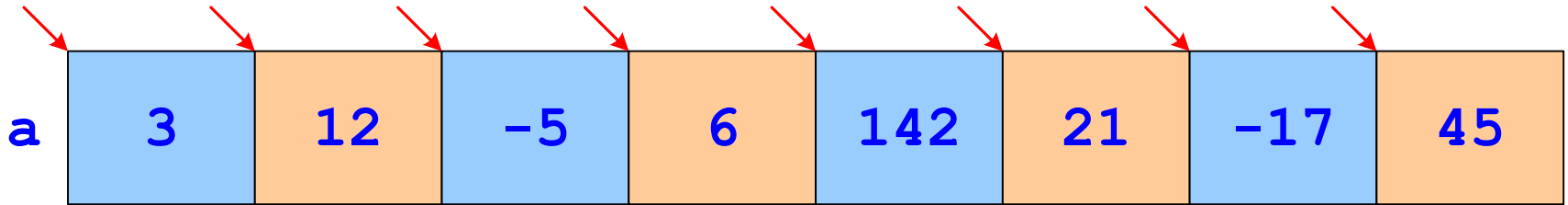


- Ta chỉ cần dùng **n** phép so sánh và **n** lần copy dữ liệu, do vậy khối lượng công việc tỷ lệ với **n**.
- Đây chưa phải là thuật toán sắp xếp.
- Vậy thực hiện như thế nào?


Chuyển Hóa Bài Toán Trộn

- Ta cần vị trí của các con trỏ chạy để trộn các mảng tương ứng với chúng.
- Tại thời điểm bắt đầu, mỗi vị trí trong mảng là một con trỏ chạy.
- Sắp xếp trộn:
 - Trộn các mảng có độ dài 1 thành mảng có độ dài 2
 - Trộn các mảng có độ dài 2 thành mảng có độ dài 4
 - Trộn các mảng có độ dài 4 thành mảng có độ dài 8
 - Tiếp tục đến khi kết thúc

Ví Dụ



Ví Dụ

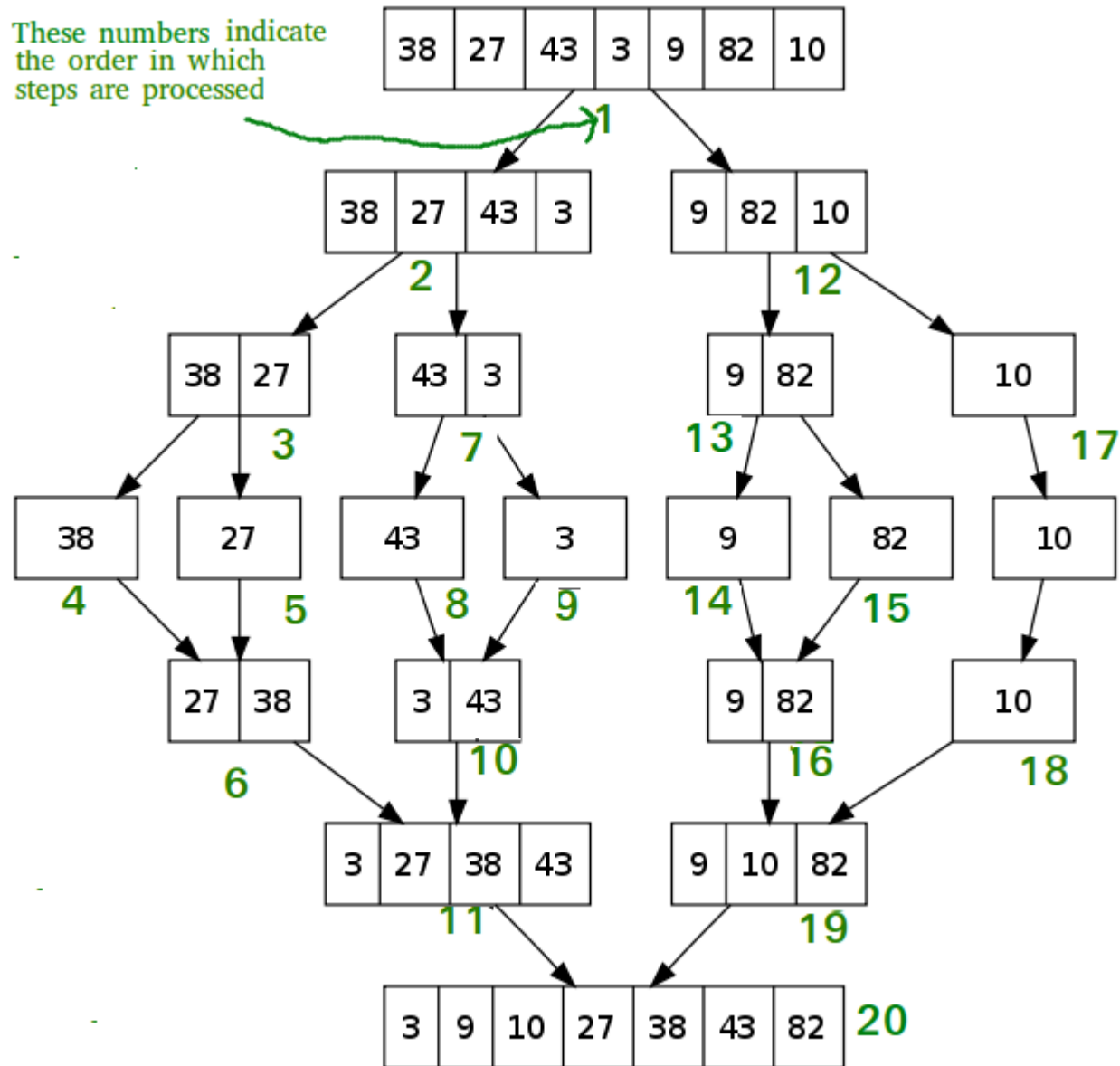


a	-17	-5	3	6	12	21	45	142
---	-----	----	---	---	----	----	----	-----

- Bài toán đã hoàn thành.
- Mỗi bước trộn yêu cầu thời gian tỷ lệ với **n**.
- Vậy có bao nhiêu lần trộn? Trong ví dụ này là 3.
- Trong trường hợp tổng quát, ta cần **$\log_2 n$** bước trộn.
- Tổng thời gian tỷ lệ với **$n \log_2 n$** (hay **$n \log n$**).

Tóm tắt

These numbers indicate the order in which steps are processed



Còn Cách Nào Tốt Hơn **$n \log n$** ?

- Trong trường hợp tổng quát, câu trả lời là không.
- Tuy nhiên trong một số trường hợp cụ thể, ta có thể làm tốt hơn.
- Ví dụ: sắp xếp các bài thi theo kết quả chấm \rightarrow để mỗi bài thi vào một trong số 10 chồng ứng với số điểm của bài \rightarrow thời gian cần thiết tỷ lệ với **n** .
- Tính hiệu quả của thuật toán có thể được tính theo toán học, không cần sử dụng máy tính.
- Phân ngành này của toán học được gọi là lý thuyết xấp xỉ, nó còn rất nhiều vấn đề thú vị chưa được giải quyết.

Tính Hiệu Quả

- Hiệu quả ở đây có nghĩa là thực hiện công việc theo cách có thể tiết kiệm các nguồn lực.
- Thông thường được đo bằng thời gian thực thi và dung lượng bộ nhớ cần thiết.
- Rất nhiều các chi tiết nhỏ trong lập trình có ít hoặc thậm chí không có tác động đáng kể nào đến tính hiệu quả.
- Khác biệt thường đến từ sự lựa chọn đúng đắn của thuật toán và/hoặc cấu trúc dữ liệu.