# **Thuật toán sắp xếp nhanh (Quick sort)**

## **I. Làm quen với thuật toán**

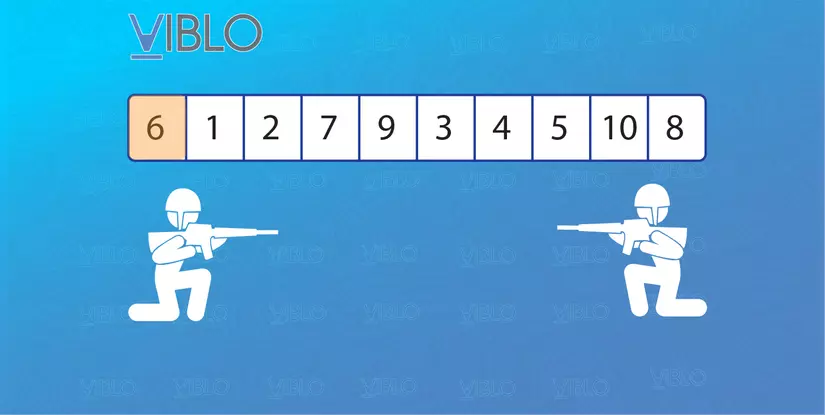
So với thuật toán sắp xếp nổi bọt (bubble sort) thì thuật toán sắp xếp nhanh có tốc độ nhanh hơn. Thay vì đi theo sắp xếp từng cặp như bubble sort, chúng ta có thể chia dữ liệu ra thành 22 danh sách, rồi so sánh từng phần tử của danh sách với một phần tử được chọn (gọi là phần tử chốt) và mục đích của chúng ta là đưa phần tử chốt về đúng vị trí của nó.

## **II. Miêu tả thuật toán**

Chắc hẳn bạn vẫn còn khá mông lung với thuật toán, để giúp bạn hiểu rõ hơn, chúng ta hãy cùng đến với một trò chơi "hành quân" sau:

Xét một dãy số như sau:

6,1,2,7,9,3,4,5,10,86,1,2,7,9,3,4,5,10,8

Yêu cầu là sắp xếp dãy trên theo thứ tự không giảm từ trái qua phải.

Chọn phần tử chốt là số 6, xét hai "quân lính" là quân lính A*A* và quân lính B*B* lần lượt đặt ở hai đầu của dãy số (quân A*A* ở vị trí đầu tiên bên trái, quân B*B* ở vị trí cuối cùng bên phải). Luật hành quân như sau: quân B*B* đi trước, bắt đầu di chuyển về bên trái, đến khi gặp được phần tử có giá trị nhỏ hơn giá trị của phần tử chốt thì dừng lại, ở đây quân B*B* dừng ở vị trí của số 55; Tiếp theo đến lượt quần A*A*, bắt đầu di chuyển về bên phải, đến khi gặp được phần từ có giá trị lớn hơn giá trị của phần tử chốt thì dừng lại, ở đây quân A*A* dừng ở vị trí số 77. Lúc này ta đổi chỗ 22 số ở vị trí của quân A*A* và B*B* cho nhau, sau đó hai quân A*A* và B*B* trở về vị trí như lúc đầu, ta thu được dãy số sau:

6,1,2,5,9,3,4,7,10,86,1,2,5,9,3,4,7,10,8

Tiếp tục cuộc hành quân như trên, lượt này ta sẽ cần đổi chỗ hai số 44 và 99 cho nhau, ta được dãy số:

6,1,2,5,4,3,9,7,10,86,1,2,5,4,3,9,7,10,8

Đến với lượt hành quân tiếp theo, ta thấy quân B*B* sẽ dừng lại ở vị trí của số 33, tuy nhiên quân A*A* chưa tìm được số nào lớn hơn 66 đã "đụng mặt" quân B, như vậy ta coi lượt hành quân này là thất bại, và ta tiến hành đổi chỗ số 33 (số mà quân B*B* đang dừng lại) với phần tử chốt là số 66. Ta thu được:

3,1,2,5,4,6,9,7,10,83,1,2,5,4,6,9,7,10,8

Lúc này, chúng ta hãy quan sát phần tử chốt (số 66): sau loạt hành quân đầu tiền thì tất cả những phần tử nằm bên trái phần tử chốt đều nhỏ hơn nó, và tất cả những phần tử nằm bên phải phần tử chốt đều lớn hơn nó. Như vậy ta đã đưa số 66 về đúng vị trí của nó.

Tiếp theo dãy được chia thành 22 dãy nhỏ hơn là dãy bên trái số 66 và dãy bên phải số 66. Ta tiếp tục thực hiện luật hành quân như trên đối với hai dãy này và sẽ thu được thêm các phần tử chốt khác ở đúng vị trí và xuất hiện thêm các dãy con độ dài ngắn hơn. Thực hiện đến cuối ta thu được dãy có thứ tự như mong muốn.

## **III. Thuật toán tham khảo**

* Thuật toán sắp xếp nhanh C++:

void quickSort(int a[], int l, int r){

int p = a[(l+r)/2];

int i = l, j = r;

while (i < j){

while (a[i] < p){

i++;

}

while (a[j] > p){

j--;

}

if (i <= j){

int temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

j--;

}

}

if (i < r){

quickSort(a, i, r);

}

if (l < j){

quickSort(a, l, j);

}}

* Thuật toán sắp xếp nhanh Java:

package quick.sort.algo;

public class QuickSort {

// Hàm nhận phần tử cuối cùng làm chốt,

// đặt các phần tử nhỏ hơn chốt ở trước

// và lớn hơn ở sau nó

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1); // index of smaller element

for (int j = low; j < high; j++) {

// Nếu phần tử hiện tại nhỏ hơn chốt

if (arr[j] < pivot) {

i++;

// swap arr[i] và arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// swap arr[i+1] và arr[high] (hoặc pivot)

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

// arr[] --> Mảng cần được sắp xếp,

// low --> chỉ mục bắt đầu,

// high --> chỉ mục kết thúc

void sort(int arr[], int low, int high) {

if (low < high) {

// pi là chỉ mục của chốt, arr[pi] vị trí của chốt

int pi = partition(arr, low, high);

// Sắp xếp đệ quy các phần tử

// trướcphân vùng và sau phân vùng

sort(arr, low, pi - 1);

sort(arr, pi + 1, high);

}

}

// In các phần tử của mảng

static void printArray(int arr[]) {

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

public static void main(String args[]) {

int arr[] = { 10, 80, 30, 90, 40, 50, 70 };

int n = arr.length;

System.out.println("Mảng ban đầu:");

printArray(arr);

QuickSort ob = new QuickSort();

ob.sort(arr, 0, n - 1);

System.out.println("Mảng sau khi sắp xếp:");

printArray(arr);

}}

* Thuật toán sắp xếp nhanh PHP:

function simple\_quick\_sort($arr){

if(count($arr) <= 1){

return $arr;

}

else{

$pivot = $arr[0];

$left = array();

$right = array();

for($i = 1; $i < count($arr); $i++)

{

if($arr[$i] < $pivot){

$left[] = $arr[$i];

}

else{

$right[] = $arr[$i];

}

}

return (

array\_merge(simple\_quick\_sort($left),

array($pivot), simple\_quick\_sort($right))

);

}}

## **IV. Những điều lưu ý về thuật toán**

### **1. Phần tử chốt.**

Sau khi hiểu về thuật toán, có lẽ bạn sẽ có một nghi vấn nhỏ nảy lên trong đầu: Tại sao chọn phần tử chốt là phần tử đầu tiên bên trái? Và cách chọn phần tử chốt có ảnh hưởng đến độ nhanh chậm của sắp xếp hay không? Thực tế thì kỹ thuật chọn phần tử chốt ảnh hưởng khá lớn đến thuật toán, bởi chúng ta có khả năng bị rơi vào các vòng lặp vô hạn. Một số cách chọn phần tử chốt để bạn tham khảo:

* Chọn phần tử đứng đầu hoặc đứng cuối làm phần tử chốt.
* Chọn phần tử đứng giữa danh sách làm phần tử chốt.
* Chọn phần tử trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt.
* Chọn phần tử ngẫu nhiên làm phần tử chốt. (Cách này có thể dẫn đến khả năng rơi vào các trường hợp đặc biệt)

### **2. Ưu điểm**

* Tốc độ sắp xếp nhanh.
* Được sử dụng trong nhiều thư viện của các ngôn ngữ như C++, Java, ...

### **3. Nhược điểm**

* Phụ thuộc vào cách chọn phần tử chốt.
* Không ổn định.

### **4. Nhận xét**

So với sắp xếp nổi bọt thì sắp xếp nhanh mỗi lần đổi chỗ mang tính nhảy vọt. Từng lượt ta chọn một phần tử chốt, đem những số nhỏ hơn nó đặt bên trái nó, những số lớn hơn nó đặt bên phải phải. Dẫn đến số lượt đổi chỗ được ít hơn so với sự đổi chỗ từng cặp của sắp xếp nổi bọt. Tuy nhiên vào tình huống xấu nhất (việc chọn phần tử chốt chưa đủ tốt) có thể khiến độ phức tạp thuật toán là O(N2)*O*(*N*2). Do tính không ổn định của quick sort nên nó có độ phức tạp trung bình là O(N.logN)*O*(*N*.*logN*).