

# Thuật toán sắp xếp

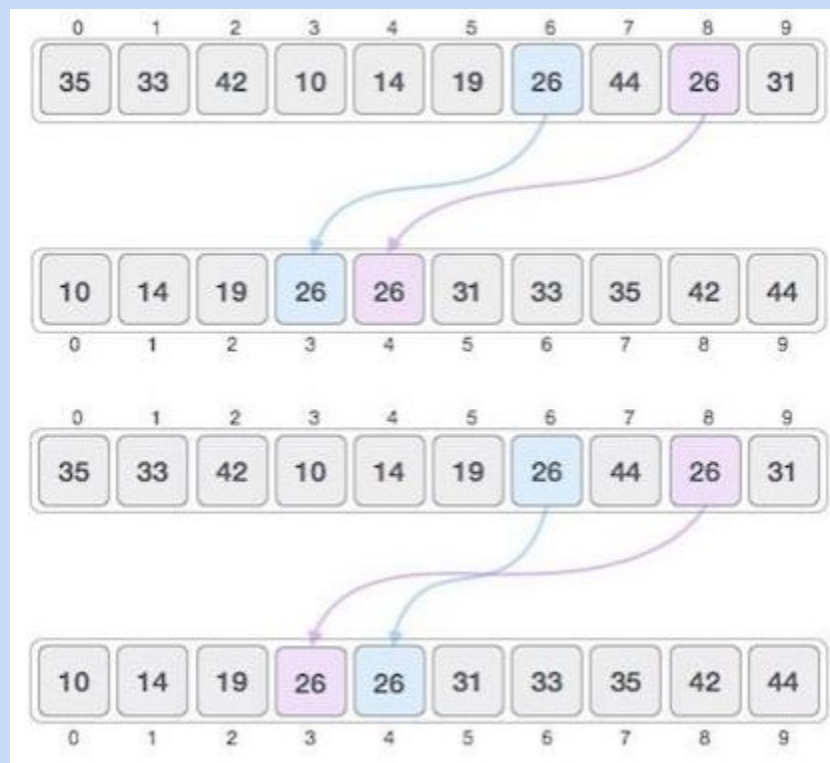
Duc-Minh Vu @ Phenikaa - ORLab

# Sắp xếp [TXNam]

- Bài toán cơ bản của lập trình máy tính
  - Xếp tăng dần một danh sách
  - So sánh theo các khóa ▪
- Được nghiên cứu từ rất sớm, hiện vẫn có vài cải tiến
- Rất nhiều thuật toán đã được phát triển, mỗi thuật toán có ưu / nhược điểm riêng
  - [Sorting algorithm - Wikipedia](#)
- Tính so sánh: thuật toán sắp xếp dựa trên việc so sánh các phần tử với nhau
  - Hầu hết các thuật toán sắp xếp đều thuộc loại này
  - Một vài thuật toán đặc biệt không cần so sánh
- Tính thích ứng (adaptive): thuật toán tận dụng được đặc tính của dữ liệu, chạy nhanh hơn nếu dữ liệu đã sắp sẵn

# Sắp xếp [TXNam]

- Phân loại theo cách làm việc với dữ liệu:
  - Sắp xếp tại chỗ (in-place): làm việc với chính dữ liệu sắp xếp
  - Sắp xếp ngoài (out-place): đẩy kết quả ra ngoài
- Phân loại theo mức độ xáo trộn dữ liệu
  - Sắp xếp ổn định (stable): thứ tự tương đối (trước / sau) giữa các phần tử bằng nhau sẽ được giữ nguyên sau khi thực hiện thuật toán sắp xếp.
  - Sắp xếp không ổn định (unstable): thứ tự tương đối (trước / sau) giữa các phần tử bằng nhau sẽ không được giữ nguyên sau khi thực hiện thuật toán sắp xếp





Sắp xếp ổn định vs sắp xếp không ổn định

# Bài toán

- Cho một dãy  $n$  phần tử, sắp xếp dãy đó tăng dần.
- Các thuật toán sắp xếp:
  - Sắp xếp không nhanh:  $O(n^2)$  - nổi bọt, chèn, chọn, etc.
  - Sắp xếp nhanh:  $O(n \log n)$  - sắp xếp nhanh, sắp xếp trộn, sắp xếp vun đống, etc
  - Sắp xếp “tuyến tính” (khai thác thêm thông tin phụ của dữ liệu): counting sort, radix sort, etc.
- C++ hỗ trợ hai hàm sắp xếp:
  - `sort`: sắp xếp nhanh
  - `stable_sort`: sắp xếp ổn định
    - Cũng là sắp xếp nhưng bảo toàn thứ tự tương đối các phần tử có giá trị bằng nhau, nghĩa là nếu  $a = b$  và  $a$  đứng trước  $b$  trong dãy ban đầu thì cũng đứng trước trong dãy đã sắp xếp.

# Một số hàm liên quan đến dãy sắp xếp trong STL C++

## Sorting:

<b>sort</b>	Sort elements in range (function template )
<b><u>stable_sort</u></b>	Sort elements preserving order of equivalents (function template )
<b>partial_sort</b>	Partially sort elements in range (function template )
<b>partial_sort_copy</b>	Copy and partially sort range (function template )
<b>is_sorted</b> 	Check whether range is sorted (function template )
<b>is_sorted_until</b> 	Find first unsorted element in range (function template )
<b>nth_element</b>	Sort element in range (function template )

# Sắp xếp nổi bọt

- [Thuật toán sắp xếp nổi bọt \(Bubble Sort\) \(tek4.vn\)](https://tek4.vn)
- [Bubble Sort - GeeksforGeeks](https://www.geeksforgeeks.org/bubble-sort/)
- So sánh hai phần tử đứng cạnh nhau và đổi chỗ cho nhau.
- Ở lần lặp thứ  $k$ :
  - sắp xếp tăng dần  $k$  phần tử đầu tiên bằng cách đổi chỗ liên tiếp từ vị trí thứ  $k$  về vị trí thứ nhất.
  - Hoặc sắp xếp phần tử lớn thứ  $n-k+1$  về đúng vị trí của nó

# Thuật toán sắp xếp cơ bản

Dùng để giới thiệu thuật toán sắp xếp  
Dùng để sắp xếp dữ liệu kích thước  
nhỏ



# Sắp xếp nổi bọt

- [Thuật toán sắp xếp nổi bọt \(Bubble Sort\) \(tek4.vn\)](https://tek4.vn)
- Duyệt toàn bộ danh sách: nếu hai phần tử liên tiếp không đúng thứ tự (tăng dần) thì đổi chỗ chúng cho nhau.
  - Nổi bọt
- Lặp lại bước duyệt cho đến khi không xảy ra đổi chỗ nữa
- Thuật toán có vẻ khá tệ, nhưng chạy tốt trong vài tình huống đặc biệt:
  - Dãy đã sắp xếp gần hết
- [Bubble Sort visualize | Algorithms | HackerEarth](#)

```
// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

Thuật toán nổi bọt có tính chất:

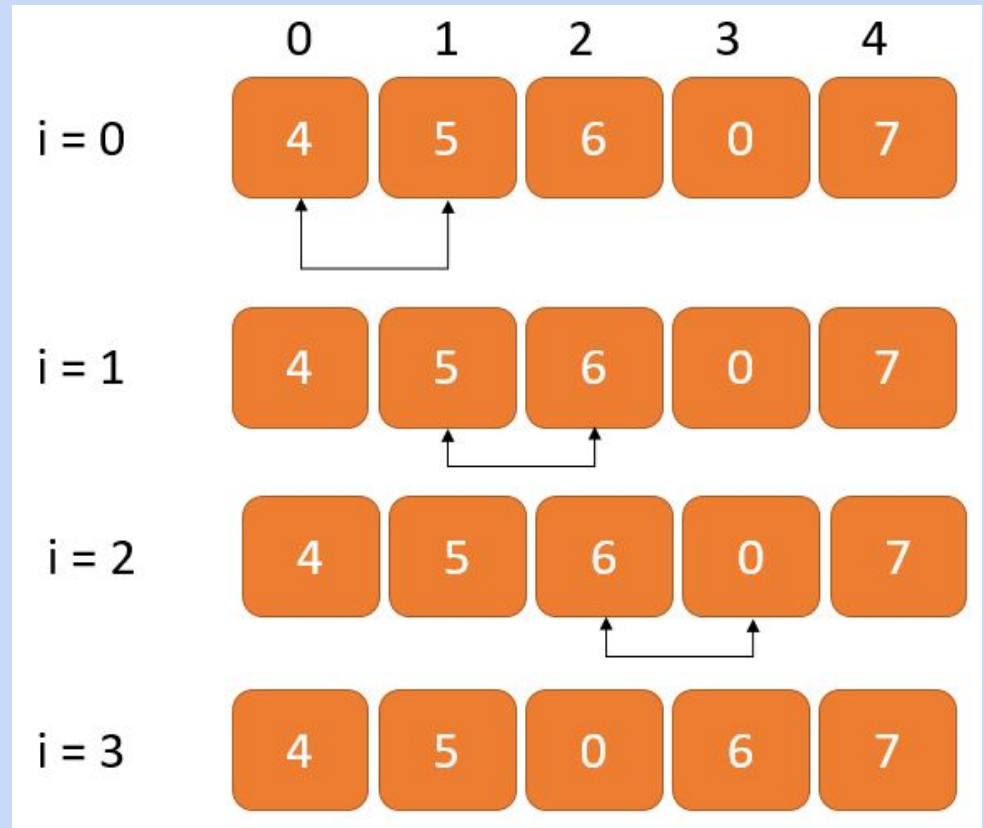
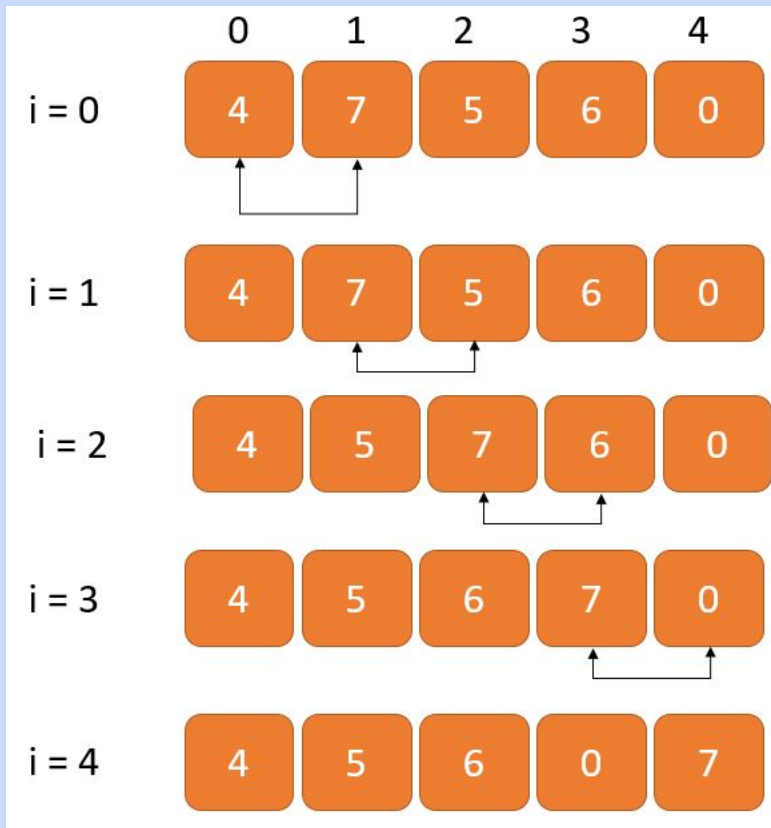
Sau lần lặp thứ  $i$ :  $i$  phần tử lớn nhất của mảng đã ở đúng vị trí.

Lần lặp 1: đưa số lớn nhất về đúng vị trí.

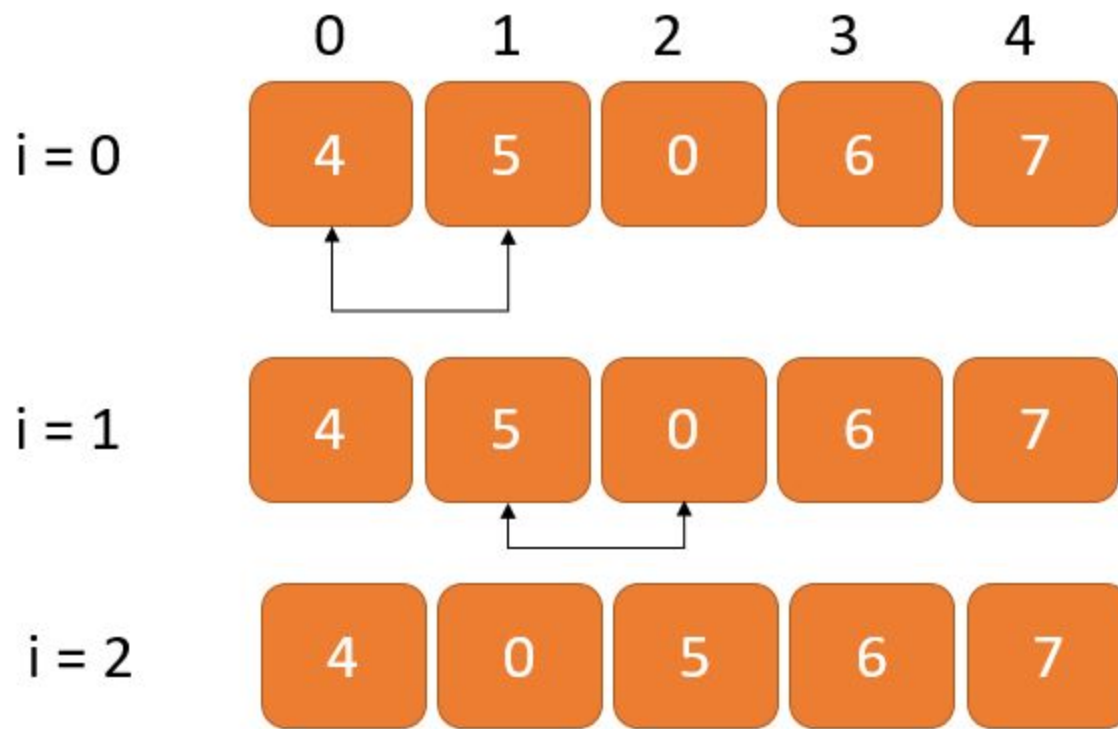
Lần lặp 2: đưa số lớn nhì về đúng vị trí.

...

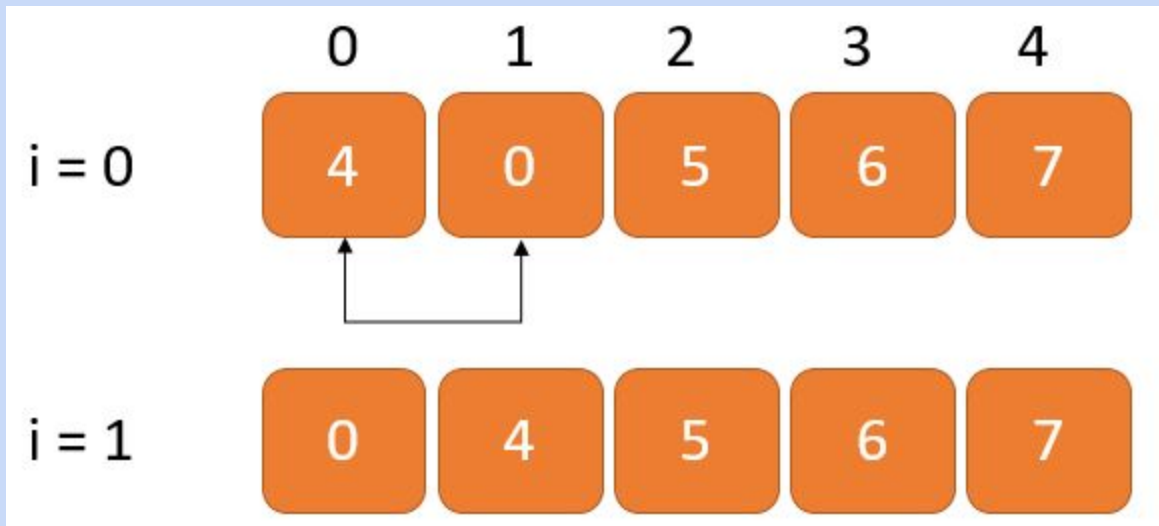
## Sắp xếp nổi bọt



Vòng lặp 1 và 2 của sắp xếp nổi bọt



Vòng lặp 3 của sắp xếp nổi bọt



Vòng lặp 4 của sắp xếp nổi bọt

# Sắp xếp chọn

- [Thuật toán sắp xếp chọn \(Selection Sort\) \(tek4.vn\)](https://tek4.vn)
- Sắp xếp chọn hay sắp xếp lựa chọn là một thuật toán chọn phần tử nhỏ nhất từ danh sách chưa được sắp xếp trong mỗi lần lặp và đặt phần tử đó ở đầu danh sách chưa được sắp xếp.
- Ở lần thứ  $k$ , số bé thứ  $k$  được đưa về đúng vị trí.
- Ở lần thứ  $k$ , tìm số bé nhất trong khoảng  $A[k..n]$  và đổi chỗ cho  $A[k]$ .
- [Selection Sort visualize | Algorithms | HackerEarth](#)

```

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of
    // unsorted subarray
    for (i = 0; i < n-1; i++)
    {

        // Find the minimum element in
        // unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

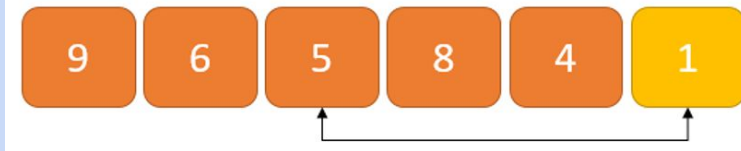
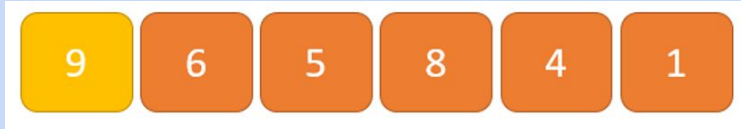
        // Swap the found minimum element
        // with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

```

Sắp xếp mảng  $A[1..n]$  bằng thuật toán sắp xếp chọn:

1. Lặp  $n-1$  lần.
2. Lần thứ  $k$  tìm số bé nhất trong đoạn  $[k,n]$  và đổi chỗ cho  $A[k]$

Sắp xếp chọn

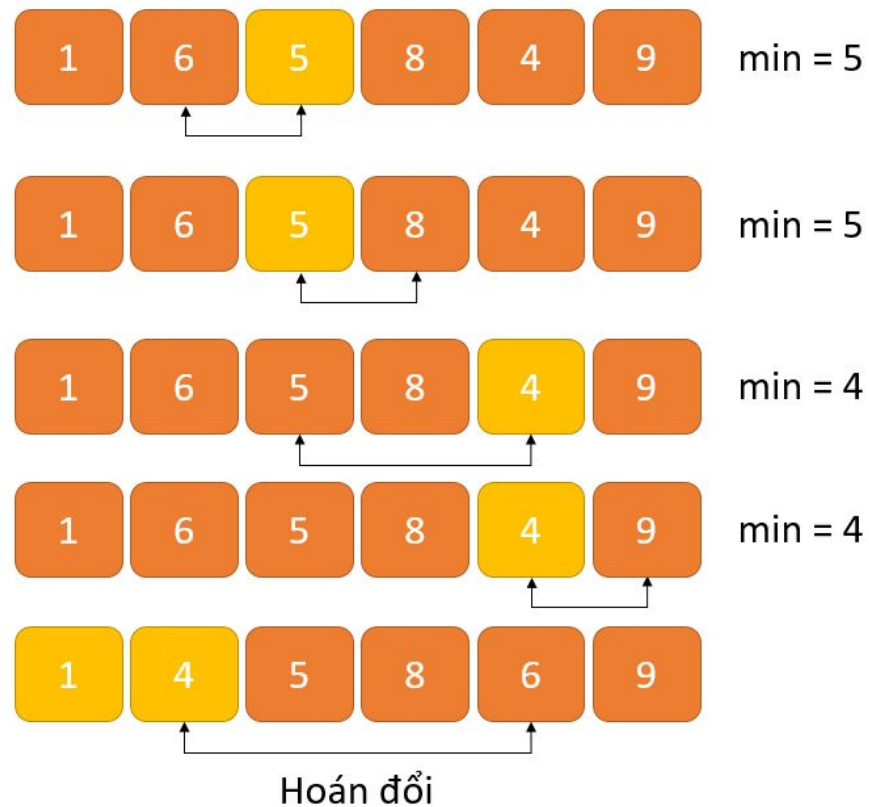


Sắp xếp chọn

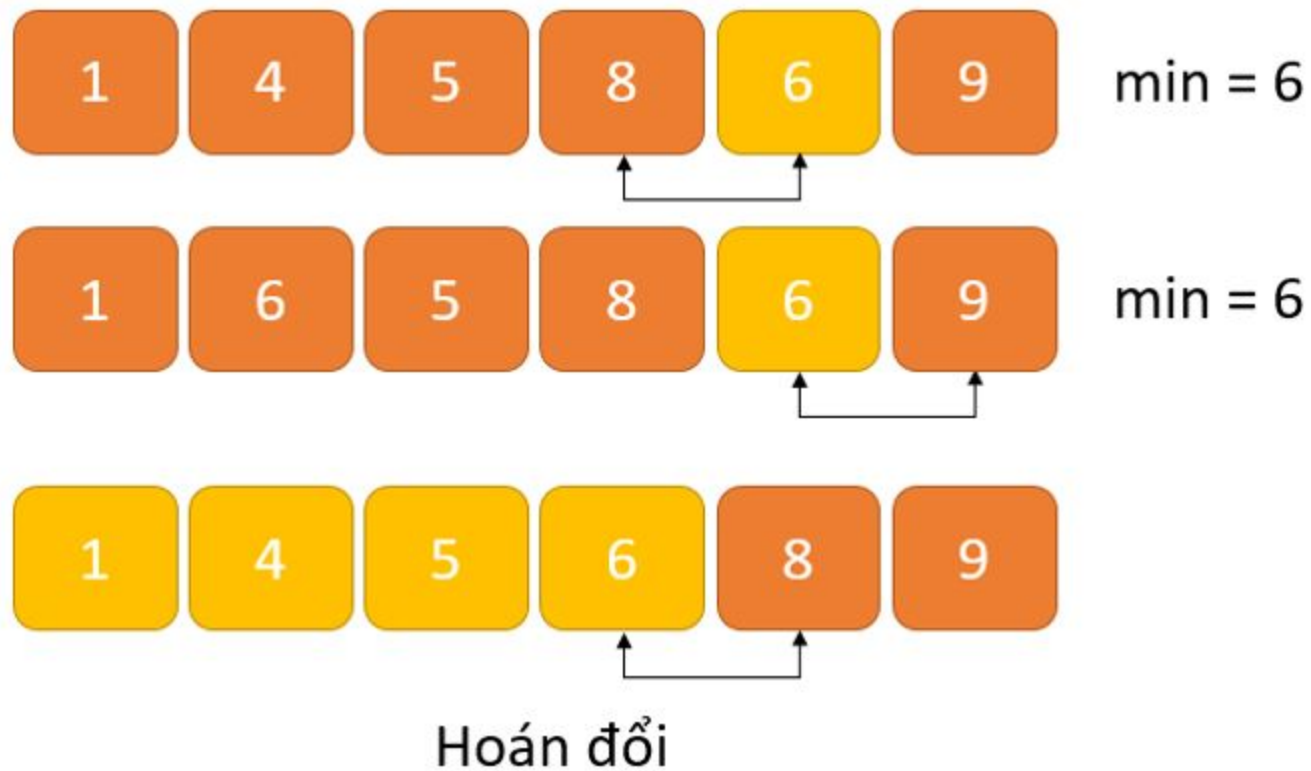


Thiếu cập nhật vị trí bé nhất ở vị trí ứng với số 4, trước khi tìm thấy vị trí bé nhất ứng với số 1.





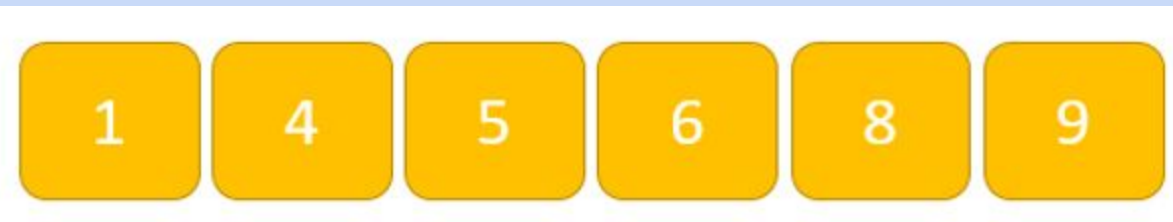
Sắp xếp chọn



Sắp xếp chọn



Giữ nguyên



Sắp xếp chọn

# Thuật toán sắp xếp chèn

- [Thuật toán sắp xếp chèn \(Insertion Sort\) \(tek4.vn\)](https://tek4.vn)
- Sắp xếp chèn là một thuật toán sắp xếp có nhiệm vụ đặt một phần tử chưa được sắp xếp vào vị trí thích hợp của nó trong mỗi lần lặp.
- Ở bước thứ  $k$ , chèn  $a[k]$  vào dãy tăng dần  $a[1..k-1]$  sao cho thu được  $a[1..k]$  là một dãy tăng dần.
  - Tìm vị trí chèn = tìm tuyến tính:  $O(k)$  cho  $k = 1..n$
  - Tìm vị trí chèn = tìm nhị phân:  $O(\log k)$  cho  $k = 1..n$
  - Chèn:  $O(k)$
  - Do đó độ phức tạp là  $O(1 + 2 + \dots + n) = O(n^2)$ .
- [Insertion Sort visualize | Algorithms | HackerEarth](#)

# Thuật toán sắp xếp chèn

Hàm sắp xếp chèn với tham số đầu vào là mảng cần sắp xếp

Đánh dấu phần tử đầu tiên là đã được sắp xếp

Sử dụng vòng lặp for cho mỗi phần tử chưa được sắp xếp x

Lấy ra phần tử x

Sử dụng vòng lặp for với  $j =$  chỉ số của phần tử cuối cùng được sắp xếp tới giá trị 0

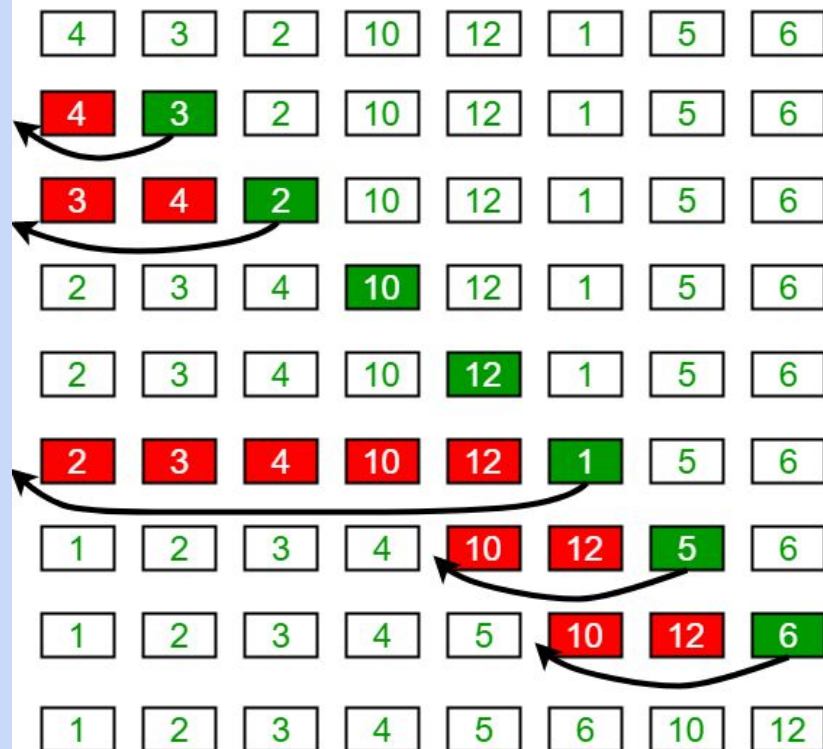
Nếu phần tử hiện tại  $j > x$

Di chuyển phần tử đã sắp xếp sang bên cạnh từ trái qua phải

Dừng vòng lặp và chèn x vào

Kết thúc hàm sắp xếp

### Insertion Sort Execution Example



```
// Function to sort an array using
// insertion sort
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key, to one
        // position ahead of their
        // current position
        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Sắp xếp chèn

# Thuật toán sắp xếp đếm phân phối

- [Thuật toán sắp xếp đếm \(Counting Sort\) \(tek4.vn\)](#)
- [Counting Sort - GeeksforGeeks](#)
- Áp dụng với các danh sách mà có khoảng giá trị nhỏ; e.g.  $1..10^6$ .
- Ý tưởng: đếm xem mỗi số xuất hiện bao nhiêu lần rồi sau đó “in” ra các giá trị với số lần in là số lần xuất hiện tương đương.
- Độ phức tạp:  $O(n+m)$  với  $n$  là số giá trị và  $m$  là giá trị lớn nhất.

# Sắp xếp không so sánh



# Đếm phân phối

- [Thuật toán sắp xếp đếm \(Counting Sort\) \(tek4.vn\)](http://tek4.vn)
- Khai thác miền giá trị để sắp xếp, các phần tử  $a[i]$  nằm trong khoảng  $1..M$  (e.g.  $M \leq 10^6$ ).
- Đếm số lần xuất hiện của mỗi giá trị  $x$  trong mảng  $A$  và lưu vào một mảng phụ  $B$ .
- Duyệt mảng phụ  $B$  in ra kết quả, e.g., in ra với mỗi số  $x$ ,  $B[x]$  lần.

# Thuật toán sắp xếp đếm phân phối

Hàm sắp xếp đếm với bộ tham số đầu vào là (Mảng, Kích thước)

Gán max với giá trị lớn nhất trong mảng

Khởi tạo mảng count hay mảng đếm với tất cả các giá trị là 0

Sử dụng vòng lặp for với  $j = 0$  cho đến giá trị của kích thước

    Tìm ra tổng số đếm của mỗi phần tử

    Lưu trữ biến đếm tại chỉ số thứ  $j$  trong mảng đếm

Sử dụng vòng lặp for với  $i = -1$  cho đến giá trị của biến max

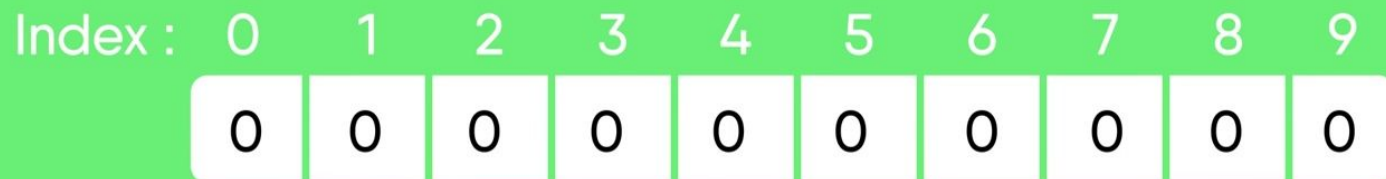
    Tìm ra tổng số tích lũy và lưu trữ nó vào trong mảng đếm

Sử dụng vòng lặp for với  $j =$  giá trị của kích thước cho tới giá trị là 1

    Lưu lại các phần tử vào trong mảng

    Giảm biến đếm của mỗi phần tử được lưu trữ đi 1 đơn vị

For simplicity, consider data in range of 0 to 9

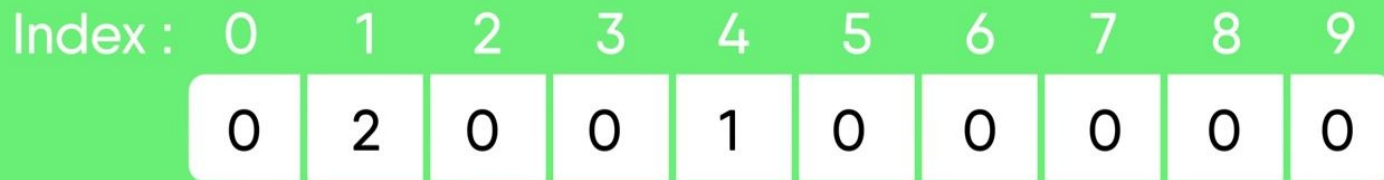


Count each element in the given array and place the count at the appropriate index.

Đếm mỗi số từ 0..9 xuất hiện bao nhiêu lần trong mảng ban đầu

Thuật toán sắp xếp đếm phân phối

For simplicity, consider data in range of 0 to 9



Đếm mỗi số từ 0..9 xuất hiện bao nhiêu lần trong mảng ban đầu

Thuật toán sắp xếp đếm phân phối

For simplicity, consider data in range of 0 to 9

1	4	1	2	7	5	2
---	---	---	---	---	---	---

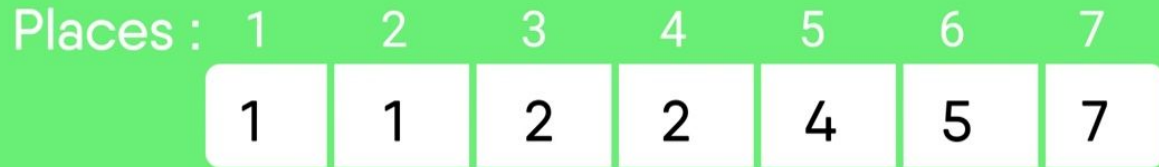
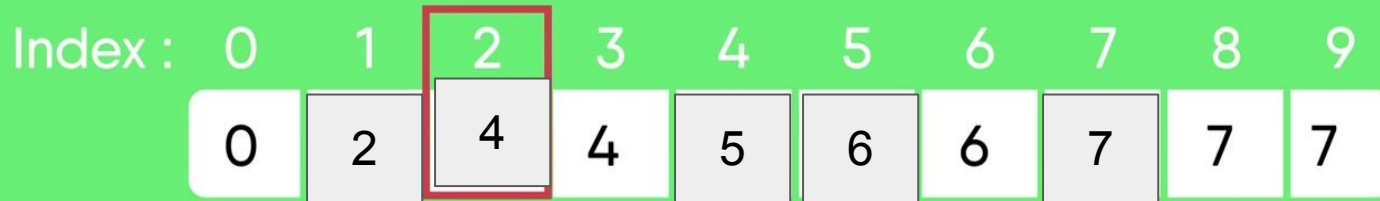
Index : 0 1 2 3 4 5 6 7 8 9

0	2	2	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---

Modify the count array by adding the previous counts.

Thuật toán sắp xếp đếm phân phối

For simplicity, consider data in range of 0 to 9



$\text{Index}[x] = \text{index}[0] + \text{index}[1] + \dots + \text{index}[x]$

Mảng places: số có giá trị x sẽ nằm ở các vị trí từ  $\text{index}[x-1]+1$  cho đến  $\text{index}[x]$ .

# Thuật toán sắp xếp đếm phân phối

```
// assume 1 <= a[i] <= RANGE;
void countSort(int arr[], int n, int RANGE)
{
    int count[RANGE + 1], i;
    memset(count, 0, sizeof(count));

    // Store count of each character
    for (i = 0; arr[i]; ++i)
        ++count[arr[i]];

    //write count[i] number i in the arr
    for (int i = 0, m = 0; i <= RANGE; i++)
        for (int j = 1; j <= count[i]; j++)
            arr[m++] = i;
}
```

Code mẫu

Code mẫu khác: [Counting Sort - GeeksforGeeks](https://www.geeksforgeeks.org/counting-sort/)

(hơi dài)

Số có giá trị  $i$  sẽ nằm ở vị trí  $\text{count}[1] + \dots + \text{count}[i-1] + 1$  cho tới  $\text{count}[1] + \dots + \text{count}[i]$ .

Nếu  $\text{count}[i] = 0$  thì số này không xuất hiện trong dãy kết quả.

```
// đếm số lần xuất hiện
for (int i = 1; i <= N; i++) dem[A[i]]++;
// đưa ra mảng được sắp
for (int i = 1, m = 0; i <= M; i++)
    for (int j = 0; j < dem[i]; j++) // i xuất hiện dem[i] lần
        A[++m] = i;
```

Độ phức tạp :  $O(N+M)$  với  $M$  là giới hạn cho các giá trị  $A[i]$ .

Đếm phân phối



# Quick Sort & Merge Sort

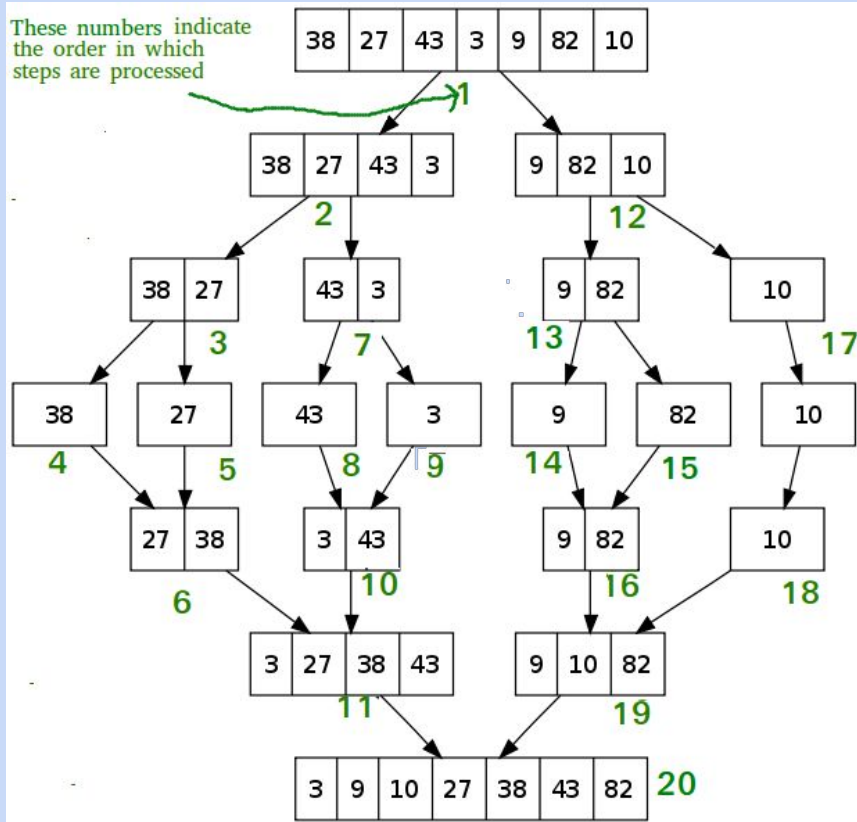
# Quick Sort & Merge Sort

- Có tốc độ sắp xếp tốt  $O(n \log n)$ .
- Luôn sử dụng chia để trị - chia ra làm các bài toán con và giải (trị), sau đó kết hợp kết quả các bài toán con để nhận lời giải bài toán ban đầu
- `Sort(A[], l, r)` // sắp xếp mảng A từ l .. r
  - Nếu  $l == r$  : return // chỉ có 1 phần tử
  - Chia A thành hai mảng con với mốc m (một giá trị nào đó)
  - Sắp xếp nửa trái `A[l, m]`
  - Sắp xếp nửa phải `A[m + 1, r]`
  - Kết hợp hai dãy tăng `A[l..m]` và `A[m+1..r]` để tạo ra một dãy tăng `A[l..r]`
- Quick Sort: chia (hơi) phức tạp, kết hợp lời giải bài toán con đơn giản.
- Merge Sort: chia đơn giản, kết hợp lời giải bài toán con (hơi) phức tạp.

# Sắp xếp trộn - Merge Sort

1. [Thuật toán sắp xếp trộn \(Merge Sort\) \(tek4.vn\)](https://tek4.vn)
2. Mốc “m” được chọn đơn giản:
  - a. Là điểm chia đôi của dãy đang cần sắp xếp
3. Làm sao để trộn hai dãy đã sắp xếp tăng B[p..m] và C[q..n] để thu được một dãy sắp xếp tăng A[...].
  - a. Khai thác giả thiết tăng để có thuật toán tuyến tính
  - b. nếu  $p > m$  hoặc  $q > n$ : đưa toàn bộ dãy còn lại vào mảng A.
  - c. Ngược lại đưa  $\min(B[p], C[q])$  vào mảng A và cập nhật chỉ số  $p=p+1$  hoặc  $q=q+1$
4. Độ phức tạp:  $T(n) = 2T(n/2) + O(n) = O(n \log n)$
5. [Merge Sort visualize | Algorithms | HackerEarth](#)

These numbers indicate the order in which steps are processed



Sắp xếp trộn

## C++ Program to Implement Merge Sort (tutorialspoint.com)

```
void merge(int *array, int l, int m, int r) {
    int i, j, k, nl, nr;
    //size of left and right sub-arrays
    nl = m-l+1; nr = r-m;
    int larr[nl], rarr[nr];
    //fill left and right sub-arrays
    for(i = 0; i<nl; i++)
        larr[i] = array[l+i];
    for(j = 0; j<nr; j++)
        rarr[j] = array[m+1+j];
    i = 0; j = 0; k = l;
    //merge temp arrays to real array
    while(i < nl && j<nr) {
        if(larr[i] <= rarr[j]) {
            array[k] = larr[i];
            i++;
        }else{
            array[k] = rarr[j];
            j++;
        }
        k++;
    }
    while(i<nl) {          //extra element in left array
        array[k] = larr[i];
        i++; k++;
    }
}
```

```
// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return; // Returns recursively

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}
```

Merge sort

# Sắp xếp nhanh - Merge Sort

- [Thuật toán sắp xếp nhanh \(Quick Sort\) \(tek4.vn\)](#)
- Phân hoạch phức tạp, sử dụng một giá trị pivot  $x = A[q]$  với  $q$  được chọn theo quy tắc nào đó.
  - $q = l$  hoặc  $q=r$ , hoặc một giá trị ngẫu nhiên trong đoạn  $[l..r]$ , etc. (giả sử cần sắp xếp  $A[l..r]$ )
- Sau khi phân hoạch (đổi chỗ các phần tử), thu được mảng  $A$  và một chỉ số  $p$  thỏa điều kiện:
  - $A[p] = x$ .
  - $A[l..p-1]$  gồm các phần tử  $< x$  (hoặc  $\leq x$ )
  - $A[p+1..r]$  gồm các phần tử  $\geq x$  (hoặc  $> x$ ).
- Thực hiện sắp xếp  $A[l..p-1]$  và  $A[p+1..r]$ .
- Sau khi sắp xếp xong,  $A[l..r]$  là một dãy tăng dần.
- [Quick Sort visualize | Algorithms | HackerEarth](#)



Pivot được chọn là  $A[r]$  ( $=4$ )

## Sắp xếp nhanh - Merge Sort



```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

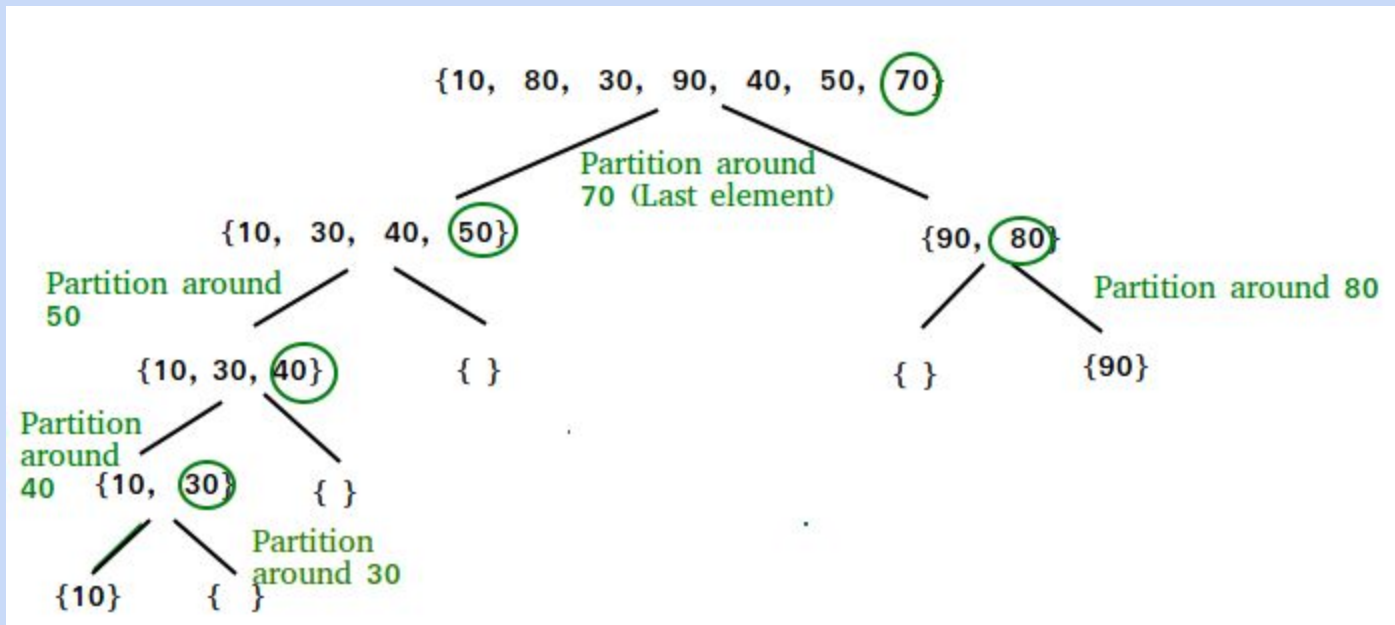
        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
```

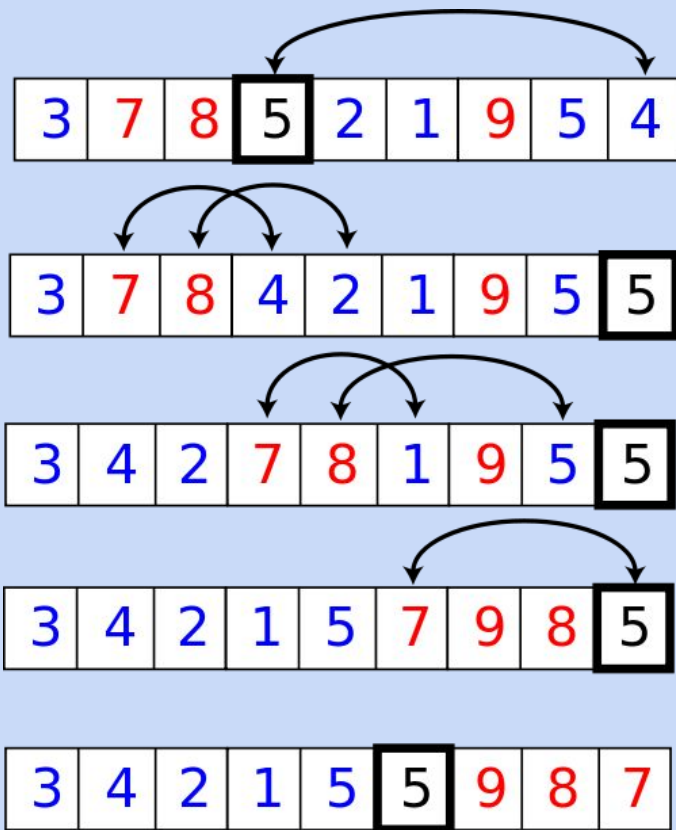
## Quicksort

# Phân hoạch - Partition

- Cho một mảng và xác định phần tử  $X$  là pivot
  - Đặt  $X$  vào đúng vị trí của mảng đã sắp xếp
  - Di chuyển tất cả các phần tử của mảng nhỏ hơn  $X$  sang bên trái và lớn hơn sang bên phải
- Trong một mảng, dãy số cho trước, chúng ta có thể lựa chọn pivot bằng các cách sau:
  - Chọn phần tử đầu tiên của mảng
  - Chọn phần tử cuối cùng của mảng
  - Chọn 1 phần tử ngẫu nhiên của mảng
  - Chọn số trung vị của mảng (Median element)
- Trong tất cả trường hợp:
  - đều có thể đưa về trường hợp chọn phần tử đầu tiên hoặc cuối cùng bằng phép toán đổi chỗ.



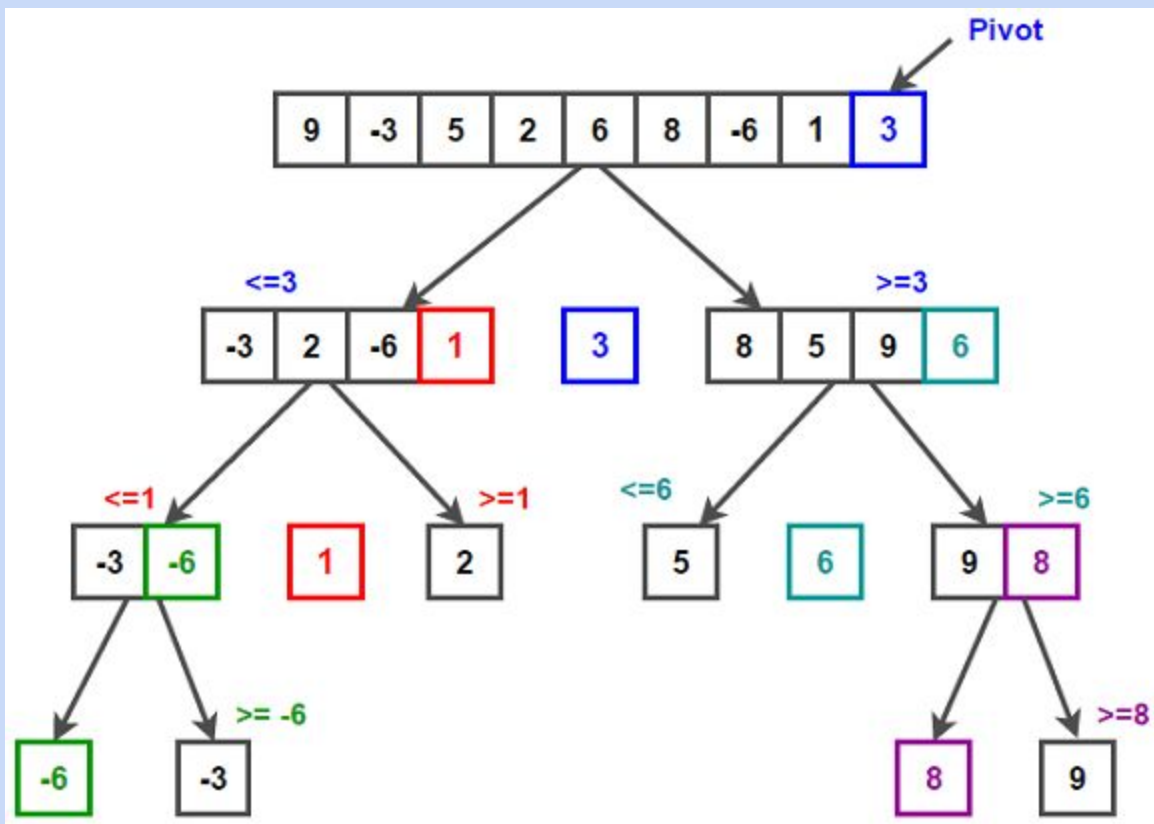
Sắp xếp nhanh với pivot là phần tử cuối cùng



Đưa về trường hợp phần tử cuối cùng bằng cách đổi chỗ với phần tử cuối cùng.

Sau khi chọn  $A[4] = 5$  làm pivot, đưa 5 về cuối mảng và thực hiện phân hoạch.

Pivot là phần tử ngẫu nhiên của đoạn



Quicksort

# Một số cách phân hoạch phổ biến

# Một số cách phân hoạch phổ biến

- Phân hoạch Hoare
- Phân hoạch Lumoton

# Phân hoạch Hoare

- Ý tưởng: duyệt từ 2 bên và đổi hai giá trị  $x \geq \text{pivot}$  và  $y \leq \text{pivot}$  cho nhau nếu  $x$  đứng trước  $y$ .
- Sau khi phân hoạch, các phần tử  $\geq \text{pivot}$  sẽ đứng sau các phần tử  $\leq \text{pivot}$ 
  - a.  $i = l$ : duyệt từ trái qua phải tìm phần tử  $A[i]$  đầu tiên lớn hơn hoặc bằng pivot  $x$ , e.g.  $A[i] \geq A[r]$
  - b.  $j = r-1$ : duyệt từ phải qua trái tìm phần tử đầu tiên nhỏ hơn pivot  $x$ ; e.g.  $A[j] < A[r]$
  - c. đổi chỗ hai phần tử này:  $\text{swap}(A[i], A[j])$
  - d.  $i++$ ,  $j--$ .
  - e. Lặp lại cho đến khi  $i \geq j$ .
  - f.  $\text{swap}(A[r], A[i])$
  - g. return  $i$  // điểm pivot
- [Quick Sort\(Hoare's Partition\) Visualization using JavaScript - GeeksforGeeks](#)



```
// Partition using Hoare's Partitioning scheme
int partition(int a[], int low, int high)
{
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;
    while (1)
    {
        do {
            i++;
        } while (a[i] < pivot);

        do {
            j--;
        } while (a[j] > pivot);

        if (i >= j) {
            return j;
        }

        swap(a[i], a[j]);
    }
}
```

- Ý tưởng: duyệt từ 2 bên và đổi hai giá trị  $x \geq \text{pivot}$  và  $y \leq \text{pivot}$  cho nhau nếu  $x$  đứng trước  $y$ .
- Lặp lại bước trên cho đến khi không còn cặp nào để hoán đổi.

<https://www.techiedelight.com/quick-sort-using-hoare-partitioning-scheme/>

## Phân hoạch Hoare

# Phân hoạch Lomuto

- Ý tưởng: duyệt từ trái qua phải và dồn các phần tử nhỏ hơn pivot về bên trái
  - Đặt pivot là phần tử cuối cùng của mảng arr.
  - $p = l - 1$
  - `for(int i = l; i < r; i++)`
    - `if (a[i] < a[r])`
      - `swap(a[i], a[++p]);` // dồn các số nhỏ hơn về bên trái.
  - `swap(a[r], a[p]);`
  - `return p;` // điểm pivot
- [\(25\) Thuật toán sắp xếp nhanh \(Quick sort\) \(viblo.asia\)](#)
- [Quick Sort\(Lomuto Partition\) Visualization using JavaScript - GeeksforGeeks](#)

```

1 // Partition using the Lomuto partition scheme
2 int partition(int a[], int start, int end)
3 {
4     int pivot = a[end];
5     int pIndex = start;
6
7     for (int i = start; i < end; i++)
8     {
9         if (a[i] <= pivot)
10        {
11            swap(a[i], a[pIndex]);
12            pIndex++;
13        }
14    }
15    swap (a[pIndex], a[end]);
16    return pIndex;
17 }

```

Duyệt từ trái qua phải,  
nếu gặp phần tử  $\leq$   
pivot thì dồn nó về bên  
trái.

Phân hoạch Lomuto

## Độ phức tạp

- Độ phức tạp trung bình:  $O(n \log n)$
- Độ phức tạp xấu nhất:  $O(n^2)$  - e.g. xảy ra khi dãy con bên trái hoặc bên phải không có phần tử nào.
- Đọc: 5.4.3. Cấu trúc dữ liệu và thuật toán - Nguyễn Đức Nghĩa.

# Hàm sort trong STL

- [sort - C++ Reference \(cplusplus.com\)](http://cplusplus.com)
- [stable\\_sort - C++ Reference \(cplusplus.com\)](http://cplusplus.com)
- Cho phép sắp xếp theo quan hệ so sánh < thông thường hoặc sort theo quan hệ so sánh tự tạo.