Trường Đại Học Khoa Học Tự Nhiên
Khoa Công Nghệ Thông Tin

TÀI LIỆU LÍ THUYẾT CTDL & GT

# Thuật Toán Sắp Xếp

HCM City – Winter 2014

# Nội dung

- Sắp xếp là gì?
- Tại sao cần sắp xếp?
- Ứng dụng sắp xếp
- Phân loại
- Thực thi
- Một số thuật toán sắp xếp

# Sắp xếp là gì?

- Cần sắp xếp nhóm người theo thứ tự chiều cao tăng dần, hãy hình dung cần thực hiện như thế nào và kết quả ra sao?

# Sắp xếp là gì? (tt)

Nó có giống như bạn nghĩ?

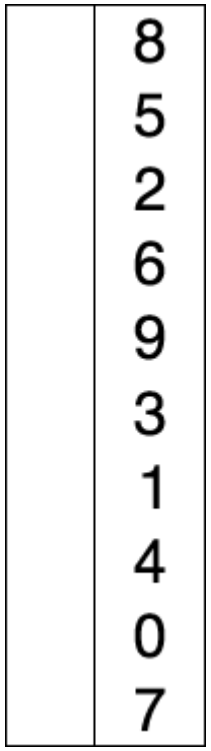
Bạn đã làm điều đó như thế nào?

Bao nhiêu bước để bạn có thể hoàn thành?

# Sắp xếp là gì? (tt)

- ***Sắp xếp*** là tiến trình đặt các phần tử của một danh sách theo thứ tự xác định.

6 5 3 1 8 7 2 4



# Tại sao cần sắp xếp?

- *Cần quan tâm đến sắp xếp vì:*
  - Sắp xếp là một khối cơ bản trong rất nhiều thuật toán khác.
  - Trong lịch sử phát triển, máy tính phải tốn nhiều thời gian dành cho sắp xếp hơn là làm bất kì việc khác. Theo [Knu73b], một phần tư tất cả chu kỳ chạy của máy mainframe được sử dụng cho sắp xếp.
  - Hầu hết các ý tưởng tuyệt vời trong quá trình thiết kế thuật toán có được đều bắt nguồn từ việc sắp xếp như chia để trị (divide-and-conquer), các cấu trúc dữ liệu, thuật toán ngẫu nhiên…

Knu73b: D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, Reading MA, 1973.

# Tại sao cần sắp xếp? (tt)

| $n$ | $n^2/4$ | $n \lg n$ |
|---|---|---|
| 10 | *25* | 33 |
| 100 | 2,500 | *664* |
| 1,000 | 250,000 | *9,965* |
| 10,000 | 25,000,000 | *132,877* |
| 100,000 | 2,500,000,000 | *1,660,960* |

*Thuật toán sắp xếp có độ phức tạp khác nhau có thể thực hiện với thời gian rất khác nhau.*

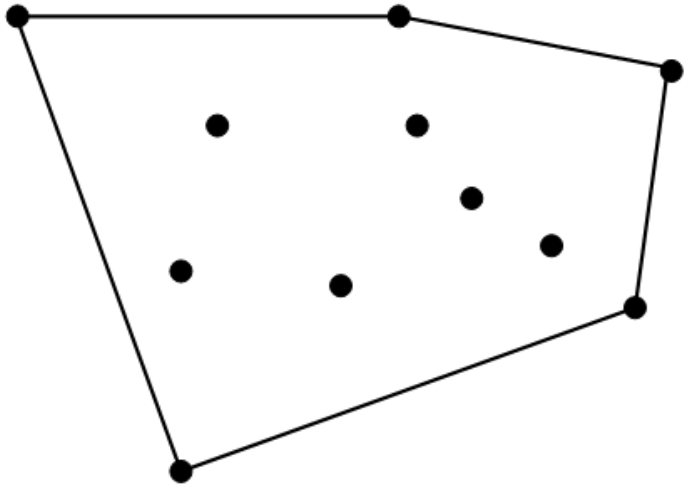
# Ứng dụng sắp xếp

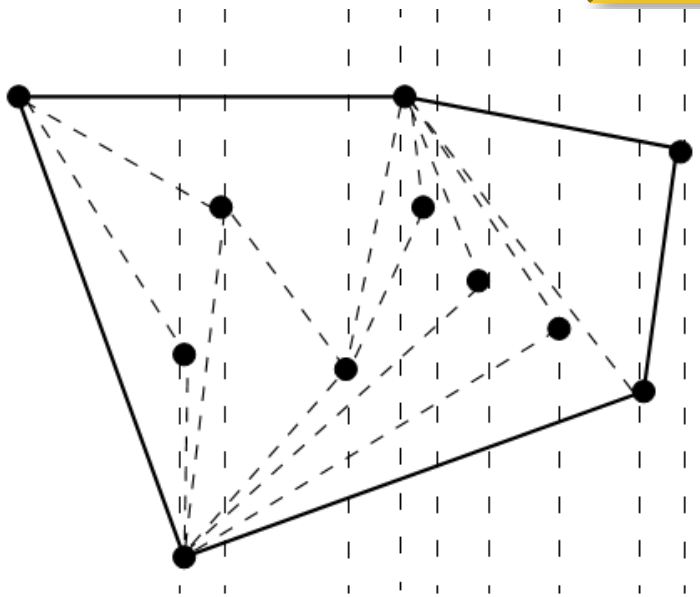
- Tìm kiếm:
  - Tìm kiếm nhị phân cho phép tìm kiếm một phần tử trong danh sách với độ phức tạp $O(\log n)$ khi mảng đã sắp xếp. Trong khi đó tìm kiếm tuần tự phải mất $O(n)$.
- Cặp gần nhau nhất:
  - Cho n số, làm cách nào để tìm cặp số có hiệu khoảng cách là nhỏ nhất? Khi đã sắp xếp, cặp gần nhau nhất này sẽ nằm kề nhau trong danh sách, do đó khi tìm tuần tự, độ phức tạp $O(n \log n)$ bao gồm cả thao tác sắp xếp.

# Ứng dụng sắp xếp (tt)

- Phần tử trùng nhau:
  - Cần kiểm tra liệu có trùng nhau trong danh sách n phần tử? Thuật toán hiệu quả nhất là sắp xếp chúng và duyệt tuần tự để kiểm tra cặp kề nhau có khoảng cách bằng 0.
- Độ phổ biến của phần tử:
  - Cho n phần tử, hãy xác định số lần xuất hiện của mỗi phần tử? Tương tự trên.
- Phần tử lớn thứ k:
  - Tìm phần tử lớn thứ k trong mảng?

# Ứng dụng sắp xếp (tt)





- Vùng bao:
  - Cho n điểm trong không gian hai chiều, hãy xác định đa giác nhỏ nhất để chứa tất cả chúng?
  - Sắp xếp các phần tử tăng dần theo tọa độ x, phần tử trái nhất và phải nhất chắc chắn nằm trên đa giác. Các điểm tiếp theo được duyệt dựa trên các điểm này.

# Ứng dụng sắp xếp (tt)

- Hãy kể các ứng dụng thực tế mà bạn thấy có sử dụng sắp xếp?
  - Danh sách lớp theo MSSV, hay họ tên
  - Sắp xếp các quốc gia theo dân số
  - Sắp xếp kết quả tìm kiếm trong công cụ tìm kiếm
  - …

# Cấu trúc giải thuật sắp xếp

- Input:
  - Mảng A gồm n phần tử
- Output:
  - Một hoán vị của A sao cho: $A_0 \leq A_1 \leq \cdots \leq A_{n-1}$ (sắp xếp tăng dần)
- Thao tác cơ bản:
  - So sánh
  - Hoán vị (đổi vị trí hai phần tử)

# Phân loại sắp xếp

| In memory sorting | | | External sorting |
|---|---|---|---|
| Comparison sorting $\Omega(N \log N)$ | | Specialized Sorting | |
| $O(N^2)$ | $O(N \log N)$ | $O(N)$ | # of tape accesses |
| • Bubble Sort<br>• Selection Sort<br>• Insertion Sort<br>• Shell Sort | • Merge Sort<br>• Quick Sort<br>• Heap Sort | • Bucket Sort<br>• Radix Sort | • Simple External Merge Sort<br>• Variations |

# Thực thi của một số thuật toán

| Algorithm | Worst-case running time |
|---|---|
| Insertion sort | $\Theta(n^2)$ |
| Merge sort | $\Theta(n \lg n)$ |
| Heapsort | $O(n \lg n)$ |
| Quicksort | $\Theta(n^2)$ |
| Counting sort | $\Theta(k + n)$ |
| Radix sort | $\Theta(d(n + k))$ |
| Bucket sort | $\Theta(n^2)$ |

# Thực thi của một số thuật toán

| | | | | | |
|---|---|---|---|---|---|
| Bubble sort | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | Yes | Exchanging |
| Cocktail sort | — | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | Yes | Exchanging |
| Comb sort | — | — | $\mathcal{O}(1)$ | No | Exchanging |
| Gnome sort | — | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | Yes | Exchanging |
| Selection sort | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | No | Selection |
| Insertion sort | $\mathcal{O}(n^2)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(1)$ | Yes | Insertion |
| Shell sort | — | $\mathcal{O}(n \log^2 n)$ | $\mathcal{O}(1)$ | No | Insertion |
| Binary tree sort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n)$ | Yes | Insertion |
| Library sort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | Yes | Insertion |
| Merge sort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n)$ | Yes | Merging |
| -place merge sort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(1)$ | No | Merging |
| Heapsort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(1)$ | No | Selection |
| Smoothsort | — | $\mathcal{O}(n \log n)$ | $\mathcal{O}(1)$ | No | Selection |
| Quicksort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(\log n)$ | No | Partitioning |
| Introsort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n \log n)$ | $\mathcal{O}(\log n)$ | No | Hybrid |
| Patience sorting | — | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | No | Insertion & Selection |
| Strand sort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | Yes | Selection |

*Wikipedia*

# Một số vấn đề khác

- *Nên sắp xếp tăng hay giảm?*
- *Sắp xếp chỉ giá trị khóa (key value) hay toàn bộ một record?*
  - Ví dụ một record: name, address, phone number, …
- *Làm gì với giá trị trùng nhau?*
  - Liệu có thể xem như một khóa độc lập và sắp xếp như bình thường hay gom lại thành một nhóm.
- *Nếu dữ liệu không phải là số?*
  - Chuỗi kí tự sắp theo alphabet?

# Nội dung

- Sắp xếp là gì?
- Tại sao cần sắp xếp?
- Ứng dụng sắp xếp
- Phân loại
- Thực thi
- Một số thuật toán sắp xếp

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Selection Sort – Chọn trực tiếp

Ý tưởng:

- Tìm phần tử thỏa yêu cầu (*nhỏ nhất,lớn nhất*…) từ phần tử đang xét đến cuối mảng.
- Hoán vị hai phần tử này.

Các bước:

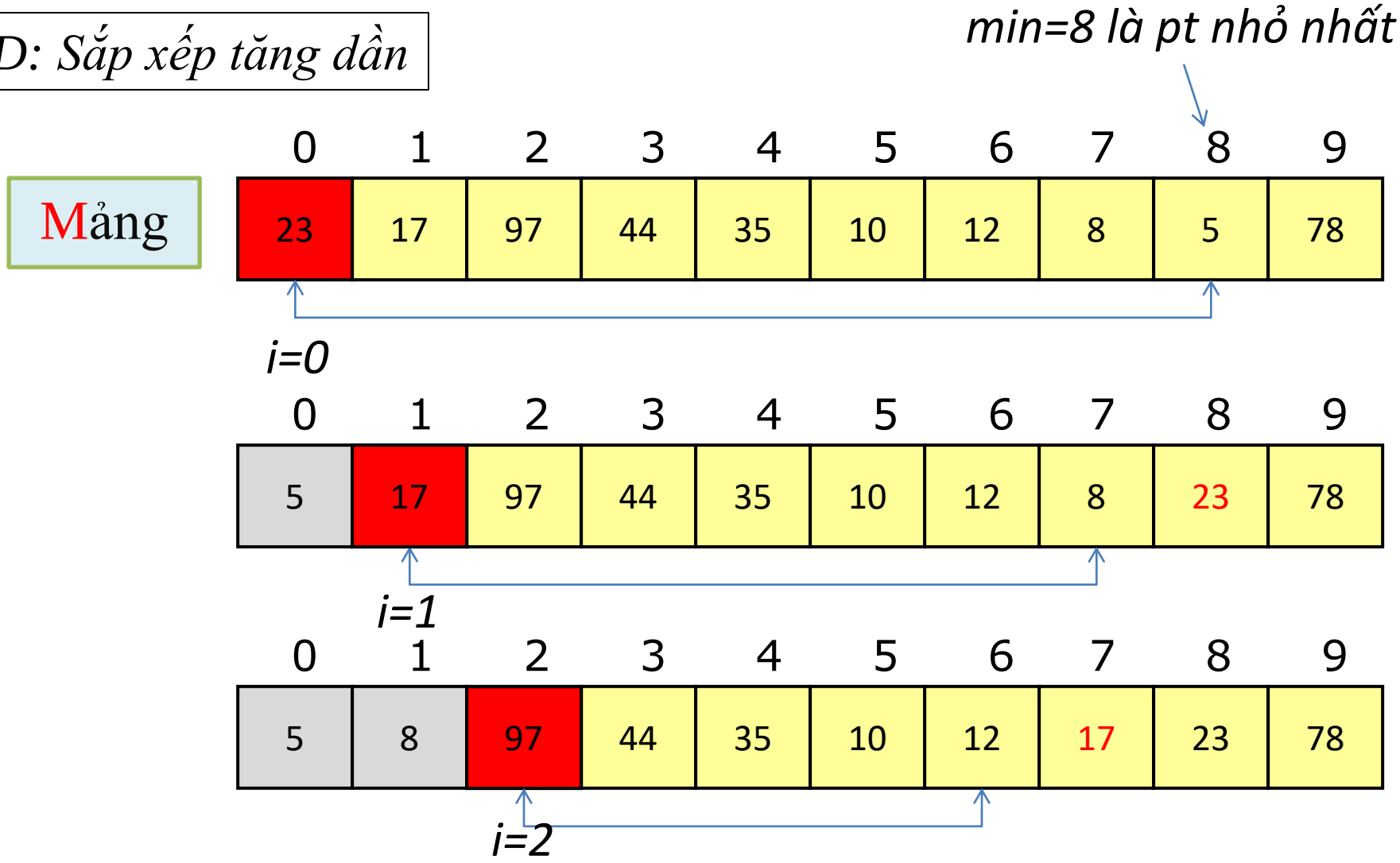
- B1: i = 0;
- B2: Tìm vị trí min của phần tử thỏa yêu cầu trong dãy từ i đến n-1;
- B3: Hoán vị.
- B4: i = i + 1.

  Nếu i < n-1 quay lại B2.

  Ngược lại, kết thúc.

# Selection Sort – Chọn trực tiếp

*VD: Sắp xếp tăng dần*



# Selection Sort – Chọn trực tiếp

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

# Nhận xét Selection Sort

- Ưu điểm:
  - Dễ thực thi
  - Sắp xếp kiểu in-place (không đòi hỏi thêm không gian chứa)
- Khuyết điểm:
  - Độ phức tạp cao: $O(n^2)$

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Insertion Sort – Chèn trực tiếp

Ý tưởng:

Giả sử từ $a_0,\ldots, a_i$ đã có thứ tự, tìm vị trí để chèn phần tử $a_{i+1}$ vào trong dãy đó.

Các bước:

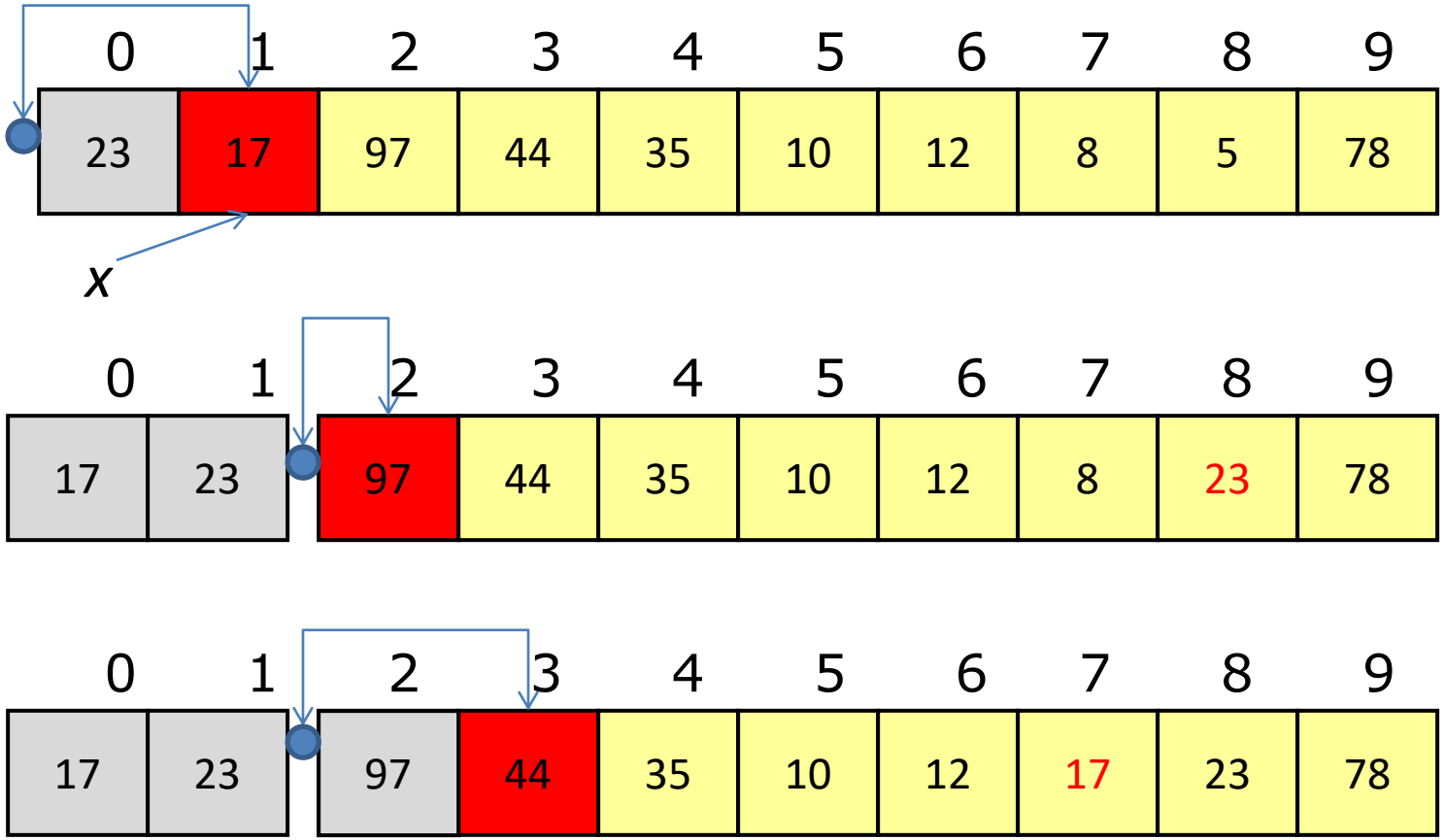
- B1: i = 1; (a[0] đã được sắp vì chỉ có 1 phần tử).
- B2: Đặt x = a[i];
- B3: Tìm vị trí pos để chèn x vào trong đoạn từ a[0] đến a[i-1];
- B4: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chỗ chèn x vào vị trí pos.
- B5: a[pos] = x;
- B6: i = i + 1 ; Nếu i < n thì quay lại B2.

  Ngược lại, kết thúc.

# Insertion Sort – Chèn trực tiếp

*VD: Sắp xếp tăng dần*

Mảng



# Insertion Sort – Chèn trực tiếp

*VD: Sắp xếp tăng dần*

```
for (i ←1 to n-1) do
    x ← a[i]; //lưu lại giá trị a[i]
    pos ← i -1;
    //tìm vị trí chèn x, kết hợp với dời chỗ
    while ( pos ≥ 0 && a[pos] > x) do
            a[pos + 1] = a [pos];
            pos ← pos – 1;
    end while
    a[pos + 1] = x;
end for
```

# Nhận xét Insertion Sort

- Ưu điểm:
  - Dễ thực thi
  - Sắp xếp kiểu in-place (không đòi hỏi thêm không gian chứa)
  - Khả năng sắp xếp realtime, nghĩa là dữ liệu có thể chưa hoàn chỉnh hoặc đang đến nhưng vân có thể sắp xếp được.
- Khuyết điểm:
  - Độ phức tạp cao: $O(n^2)$

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Interchange Sort – Đổi chỗ trực tiếp

Ý tưởng:

- Với mỗi vị trí trong mảng a, tiến hành đổi chỗ với các phần tử sau nó nếu xảy ra nghịch thế.

Các bước:

- B1: i = 0;
- B2: Duyệt qua từng phần tử j liền sau i;
- B3: Nếu a[i] và a[j] vi phạm điều kiện, tiến hành đổi chỗ;
- B4: i = i + 1 ;

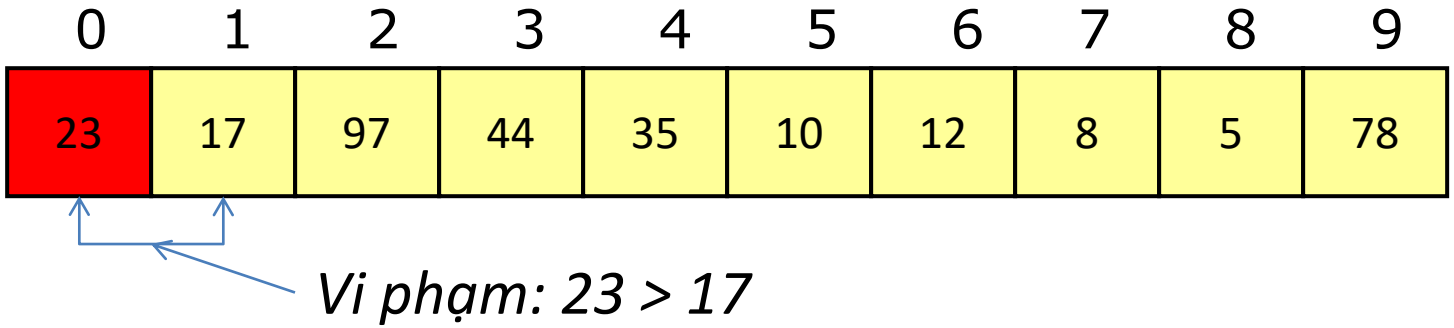
  Nếu i < n-1 thì quay lại B2.

  Ngược lại, kết thúc.

# Interchange Sort – Đổi chỗ trực tiếp

*VD: Sắp xếp tăng dần*

Mảng



# Interchange Sort – Đổi chỗ trực tiếp

*VD: Sắp xếp tăng dần*

```
for (i ←0 to n-2) do
    for (j ←i+1 to n-1) do
            if (a[i] > a[j]) then a[i] ↔ a[j]
    end
end
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Bubble Sort – Nổi bọt

Ý tưởng: phần tử nhẹ nổi lên

- Xuất phát từ cuối mảng, lần lượt đổi chỗ phần tử liền trên (kế cận) nếu chúng nghịch thế với nhau.
- Phần tử nhẹ nhất sẽ nổi lên đầu mảng, bước tiếp không xét đến nó nữa.

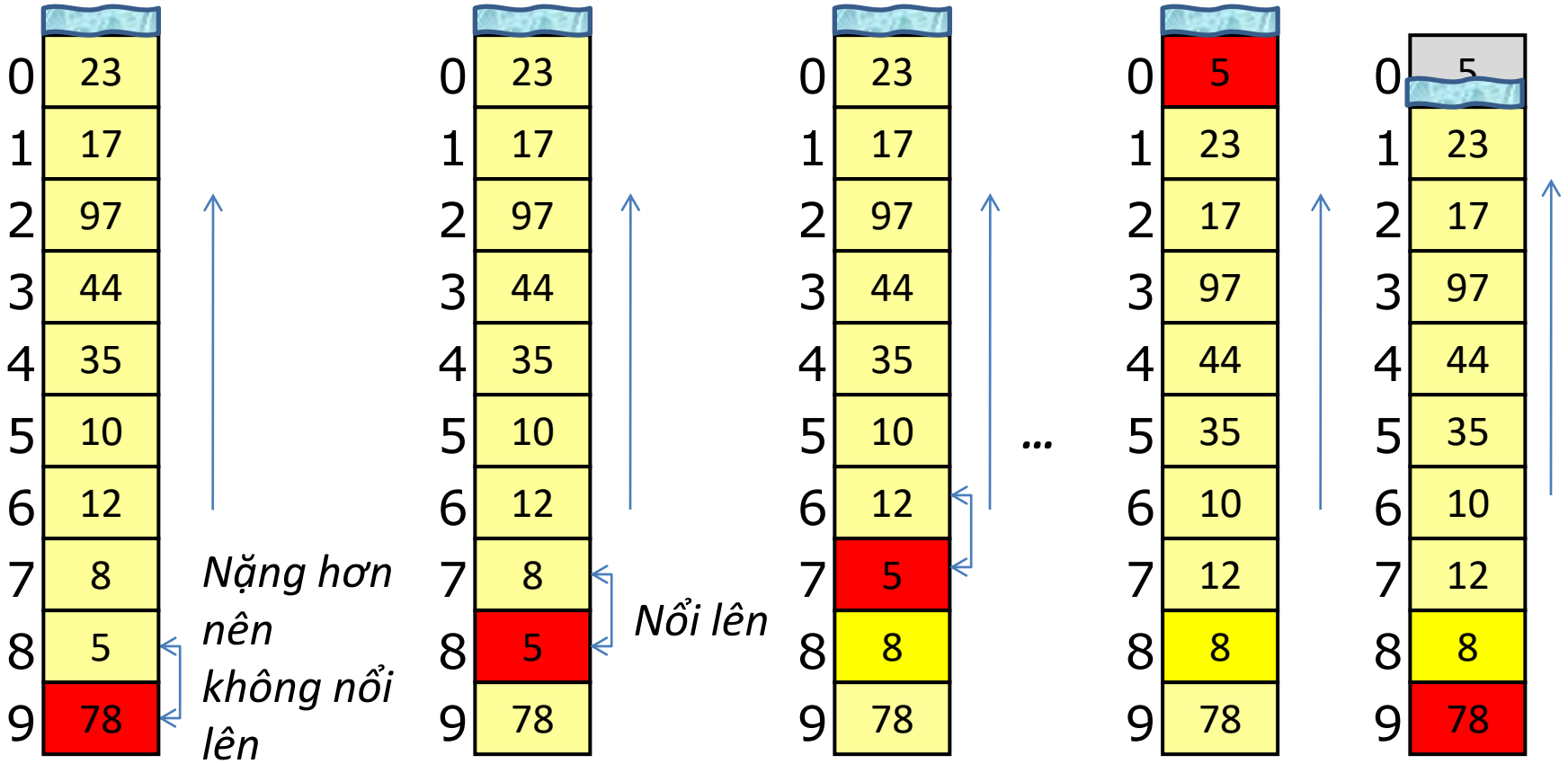
Các bước:

- B1: i = 0; //giới hạn nổi hay bề mặt nổi
- B2: j = n-1; //duyệt từ cuối dãy
- B3: Nếu a[j] và a[j-1] vi phạm điều kiện, tiến hành đổi chỗ; //nổi bọt
- B4: j = j -1; Nếu j > i thì quay lại B3. //nổi bọt cho đến bề mặt
- B5: i = i + 1 ; Nếu i < n-1 thì quay lại B2.

  Ngược lại, kết thúc.

# Bubble Sort – Nổi bọt

*VD: Sắp xếp tăng dần*



# Bubble Sort – Nổi bọt

*VD: Sắp xếp tăng dần*

```
for (i ←0 to n-2) do
    for (j ←n-1 to i+1) do
            if  (a[j-1] > a[j]) then a[j-1] ↔ a[j]
    end
end
```

# Nhận xét Bubble Sort

- Ưu điểm:
  - Dễ thực thi
- Khuyết điểm:
  - Độ phức tạp cao: $O(n^2)$, thậm chí ở cả trường hợp tốt nhất → thuật toán cải tiến: cho bề mặt hạ xuống nơi xảy ra hoán vị cuối cùng.

# Các thuật toán sắp xếp

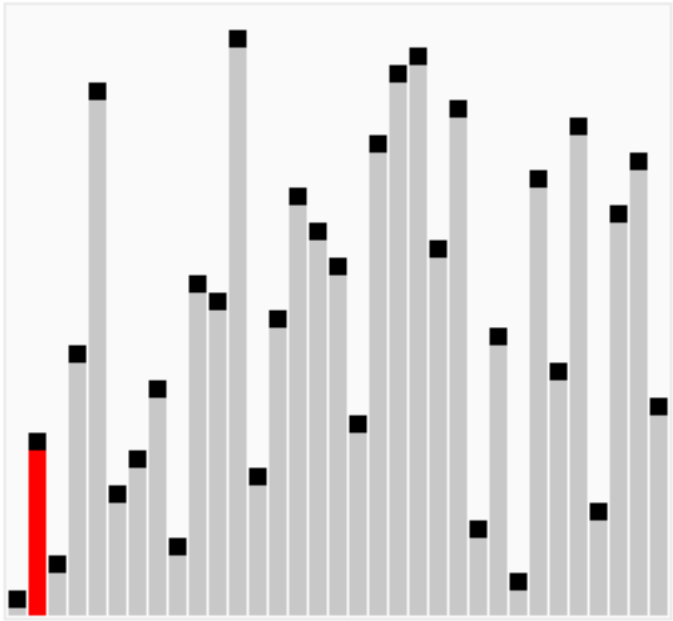
- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Shaker Sort – Nổi bọt và lắng đọng

Có tên gọi khác là Bidirectional Bubble Sort, Cocktail Sort.

Ý tưởng: phần tử nhẹ nổi lên, phần tử nặng chìm xuống

- Tương tự như Bubble sort, nhưng thêm phần tử nặng chìm xuống.
- Ngoài ra còn cải tiến, giảm những phép so sánh thừa bằng cách:
  + Bề mặt nổi cho giai đoạn kế sẽ là nơi xảy ra hoán vị nổi lần cuối.
  + Đáy chìm cho giai đoạn kế sẽ là nơi xảy ra hoán vị chìm lần cuối.



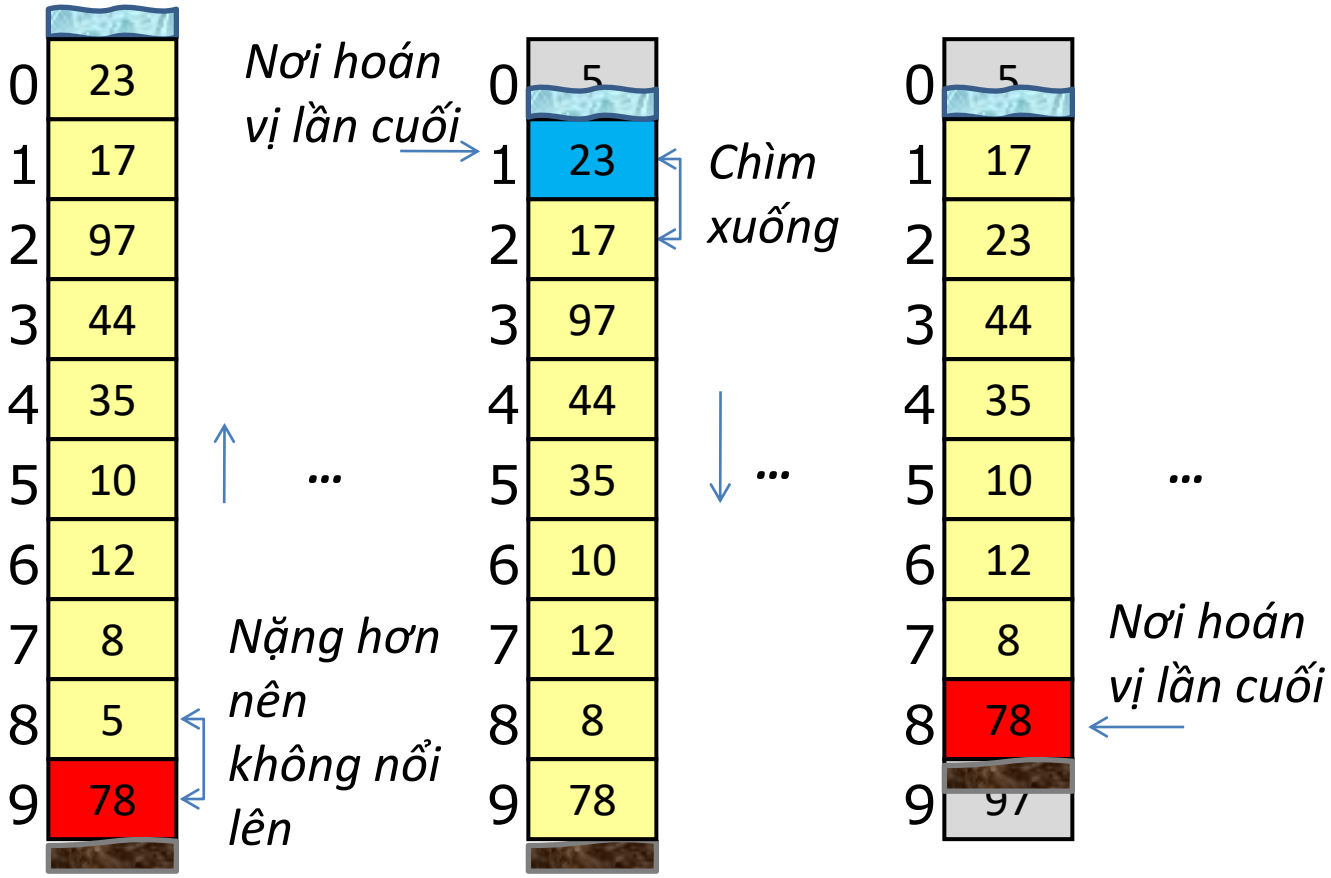
# Shaker Sort – Nổi bọt và lắng đọng

Các bước:

- B1: bm = 0; //giới hạn nổi hay bề mặt nổi
  day = n-1; //giới hạn chìm hay đáy
  k = n-1; //lưu lại vị trí xảy ra hoán vị lần cuối
- B2: j = day; //đẩy phần tử nhẹ từ đáy lên
  - B2a: Nếu a[j] và a[j-1] vi phạm, tiến hành hoán vị; //nổi bọt
    và k = j; // lưu lại nơi xảy ra hoán vị
  - B2b: j = j -1; Nếu j > bm thì quay lại B2. //nổi bọt cho đến bề mặt
- B3: bm = k; //bề mặt mới là nơi hoán vị cuối vì dãy trên nó đã có thứ tự
- B4: j = bm; //đẩy phần tử nặng từ bề mặt xuống.
  - B4a: Nếu a[j] và a[j+1] vi phạm, tiến hành hoán vị; //chìm
    và k = j; // lưu lại nơi xảy ra hoán vị
  - B4b: j = j +1; Nếu j < day thì quay lại B4. //chìm cho đến đáy
- B5: day = k; //đáy mới là nơi hoán vị lần cuối vì dãy dưới đã có thứ tự.
- B6: Nếu bm < day thì quay lại B2. Ngược lại, kết thúc.

# Shaker Sort – Nổi bọt và lắng đọng

*VD: Sắp xếp tăng dần*



# Shaker Sort – Nổi bọt và lắng đọng

*VD: Sắp xếp tăng dần*

```
bm ←0; day ← n-1; k ← n-1;
while (bm < day) do
    for (j ←day to bm+1) do
            if  (a[j-1] > a[j]) then
                        a[j-1] ↔ a[j];
                        k ← j;
            end if
    bm ← k;
     for (j ←bm to day-1) do
            if  (a[j] > a[j+1]) then
                        a[j+1] ↔ a[j];
                        k ← j;
            end if
    day ← k;
end while
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Binary Insertion Sort – Chèn nhị phân

Ý tưởng:

- Trong thuật toán chèn trực tiếp (insertion sort), việc tìm vị trí để chèn là tuần tự. Nhằm cái tiến tốt hơn, ta sử dụng phương pháp tìm kiếm nhị phân (binary search) để tìm vị trí này.

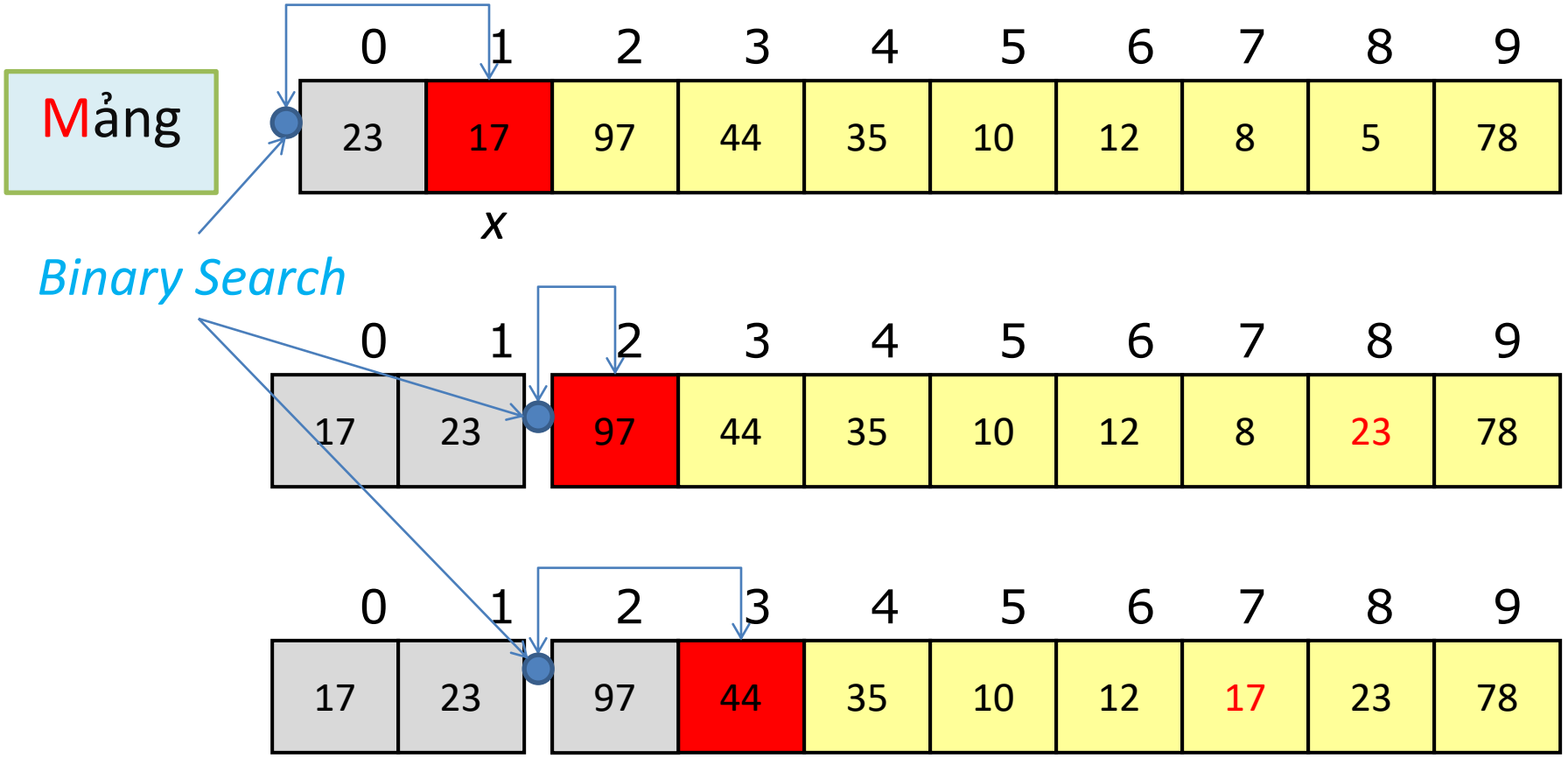
Các bước:

- B1: i = 1; (a[0] đã được sắp vì chỉ có 1 phần tử).
- B2: Đặt x = a[i];
- B3: Tìm vị trí pos để chèn x vào trong đoạn từ a[0] đến a[i-1] *bằng phương pháp tìm kiếm nhị phân (binary search)*
- B4: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chỗ chèn x vào vị trí pos.
- B5: a[pos] = x; //chèn x vào
- B6: i = i + 1 ; Nếu i < n thì quay lại B2.

  Ngược lại, kết thúc.

# Binary Insertion Sort – Chèn nhị phân

*VD: Sắp xếp tăng dần*



# Binary Insertion Sort – Chèn nhị phân

*VD: Sắp xếp tăng dần*

```
for (i ←0 to n-1) do
    x ← a[i]; //lưu lại giá trị a[i]
    //tìm vị trí chèn x bằng phương pháp tìm kiếm nhị phân
    pos ← BinarySearch(a,i,x);
    //dời các phần tử về sau để trừ khoảng trống
    for (j ← i to pos+1) do
            a[j] = a [j-1];
    end for
    a[pos] = x;
end for
```

# Binary Insertion Sort – Chèn nhị phân

```
function BinarySearch(...,k,x) //k là giới hạn vị trí tìm, x là giá trị cần s2
    left ← 0; right ← k;
    do
            mid ← (left+right)/2;                       //mốc so sánh
            if (x ≤ a[mid]) then right ← mid – 1;       //ưu tiên bên trái hơn
            else      left ← mid + 1;                   //vì có thể bên trái có
            end if                                      //phần tử bằng a[mid]
    while ( left ≤ right);
    return mid;
end function
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Shell Sort – Độ dài bước giảm dần

Ý tưởng:

- Chia dãy ban đầu thành những dãy con gồm các phần tử cách nhau h vị trí.
- Sắp xếp các phần tử trong dãy con.
- Giảm khoảng cách h để tạo thành dãy con mới. Dừng khi h = 1.

Các bước:

- B1: Tạo các khoảng cách $h_0, h_1, \ldots, h_{k-1}$ với $h_i > h_{i+1}$ và $h_{k-1} = 1$;
- B2: i = 0;
- B3: Phân chia dãy ban đầu thành các dãy con cách nhau h[i] khoảng cách. Sắp xếp từng dãy con bằng phương pháp chèn trực tiếp.
- B4: i = i + 1 ; Nếu i < k thì quay lại B2.

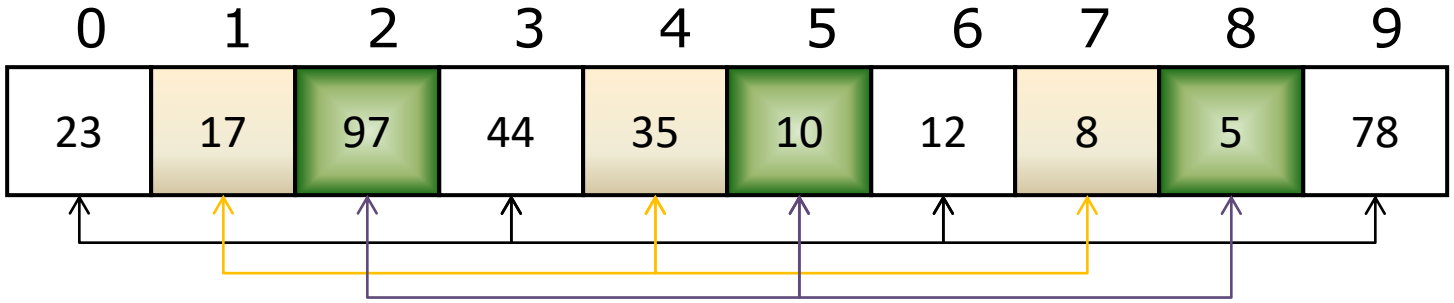
  Ngược lại, kết thúc.

# Shell Sort – Độ dài bước giảm dần

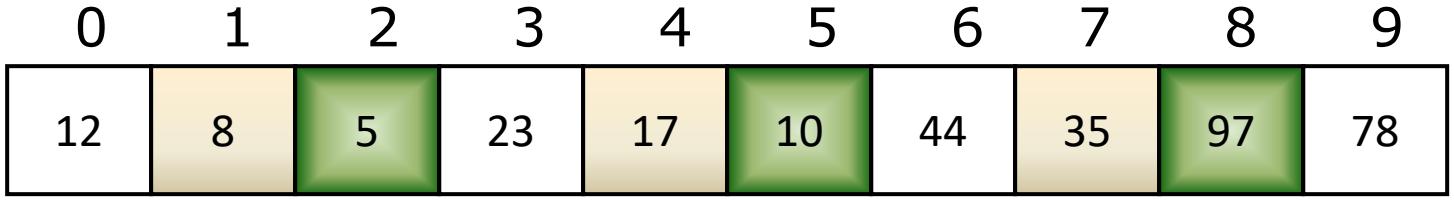
*VD: Sắp xếp tăng dần*

Các khoảng cách là 3, 1

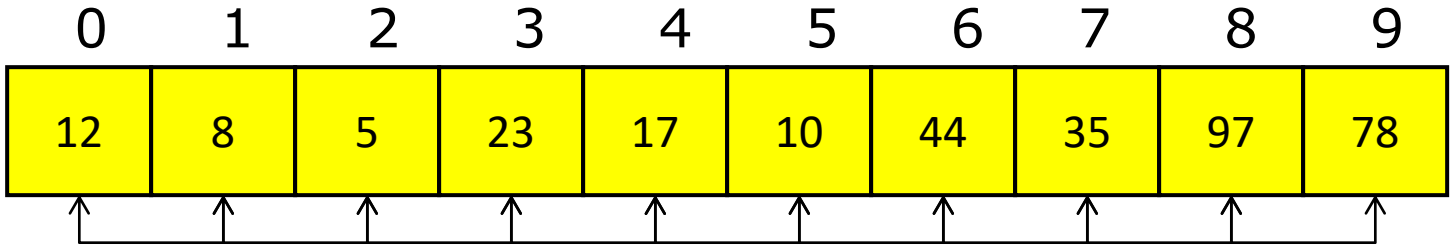
*h=3*




*Sắp xếp từng dãy con*




*h=1*




...

# Shell Sort – Độ dài bước giảm dần

*VD: Sắp xếp tăng dần*

```
Chọn h[0], h[1], …, h[k-1]=1;
for (step ← 0 to k-1) do
    len = h[step];
    for (i ← len to n-1) do
            x = a[i];
            j = i – len; //j là vị trí kề trước trong cùng dãy con
            //sắp xếp dãy con chứa x bằng p2 chèn trực tiếp
            while ( j ≥ 0 && x < a[j]) do
                    a[j+len] = a[j];
                    j = j – len;
            end while
            a[j+len] = x;
    end for
end for
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Heap Sort – Sắp xếp cây/vun đống

Ý tưởng:

- Khi tìm phần tử nhỏ nhất ở bước i, phương pháp sắp xếp chọn trực tiếp không tận dụng các thông tin đã có được do các phép so sánh ở bước i-1.
- Sử dụng cấu trúc dữ liệu cây Heap để giải quyết bài toán trên.

Định nghĩa Heap:

- Heap là cây vun đống nhị phân: nếu B là node con của A thì key(B) ≤ key(A). Thường được gọi là max-heap. So sánh ngược lại gọi là min-heap.
- Xét trường hợp sắp xếp tăng dần và đếm từ 0 thì $a_l$, $a_{l+1}$,…, $a_r$ là một heap khi $\forall i \in [l,r]$:

  + $a_i \geq a_{2i+1}$ (nhánh trái)

  + $a_i \geq a_{2i+2}$ (nhánh phải)

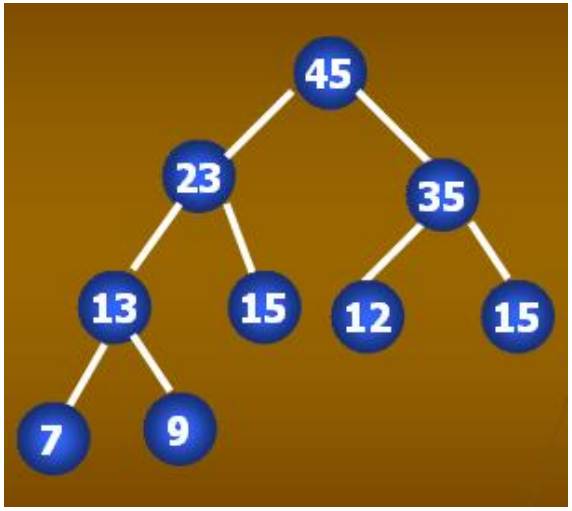
  Lúc đó, $(a_i,a_{2i+1})$ và $(a_i,a_{2i+2})$ là các cặp phần tử liên đới.

# Heap (đống)

- Max-Heap (đống):
  - Mỗi mảng a có thể xem như một cây nhị phân với gốc là phần tử đầu mảng a[0].
  - Con bên trái của đỉnh a[i] là a[2*i+1], con bên phải của đỉnh a[i] là a[2*i+2] nếu 2*i+1<=n.

  → các phần tử có chỉ số $i > \left\lfloor \frac{n}{2} \right\rfloor$ sẽ không có con, gọi là *lá*

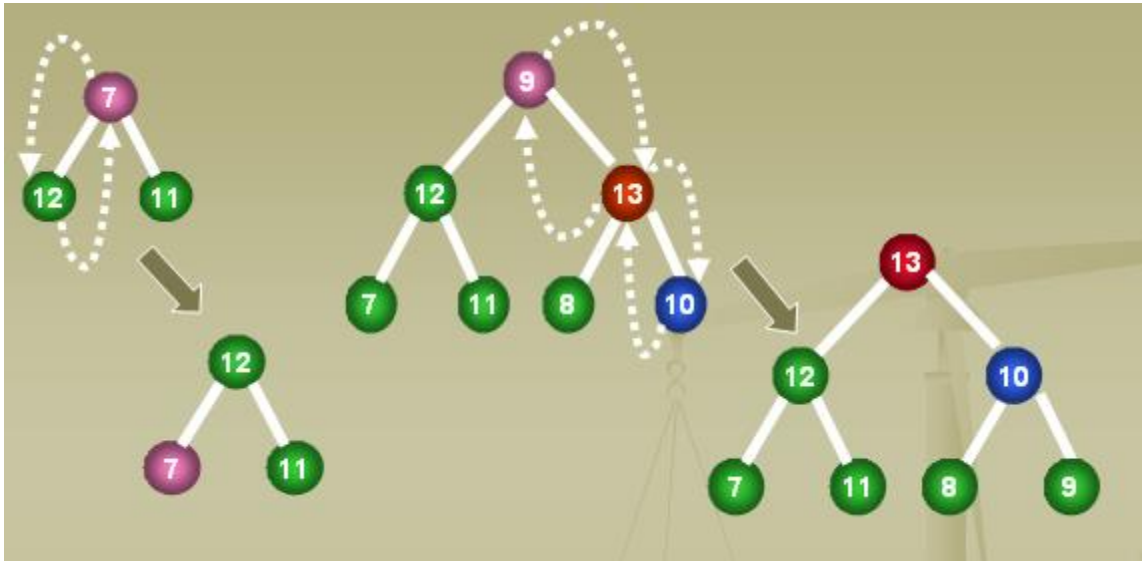
  - Node con luôn có giá trị nhỏ hơn node cha.



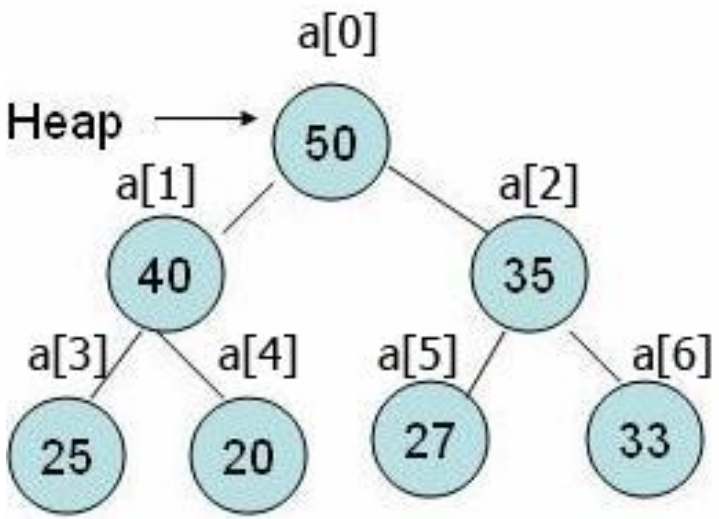

*Mảng (45,23,35,13,15,12,15,7,9) là một đống*

# Vun đống

- Việc sắp xếp lại các phần tử của mảng ban đầu sao cho nó trở thành đống được gọi là vun đống.



# Tính chất Heap




| | Heap |
|---|---|
| 0 | 50 |
| 1 | 40 |
| 2 | 35 |
| 3 | 25 |
| 4 | 20 |
| 5 | 27 |
| 6 | 33 |

Tính chất Heap:

[1] Nếu $a_l, a_{l+1}, \ldots, a_r$ là heap, thì $a_i, a_{i+1}, \ldots, a_j$ $(l \le i \le j \le r)$ cũng là một heap.

[2] Nếu $a_l, a_{l+1}, \ldots, a_r$ là heap, thì $a_l$ luôn là phần tử lớn nhất (xét max-heap).

[3] Mọi dãy $a_l, a_{l+1}, \ldots, a_r$ với $i > \frac{r}{2}$ là một heap.

# Heap Sort – Sắp xếp cây

Giải thuật: gồm 2 giai đoạn

- Giai đoạn 1: Hiệu chỉnh dãy số ban đầu thành heap.
- Giai đoạn 2: Sắp xếp dãy số dựa trên heap.
  + B1: Đưa phần tử lớn nhất về vị trí cuối dãy bằng cách hoán vị;
  + B2: Loại phần tử cuối khỏi heap, hiệu chỉnh phần còn lại thành một heap.
  + B3: Nếu heap còn phần tử thì lặp lại B1.

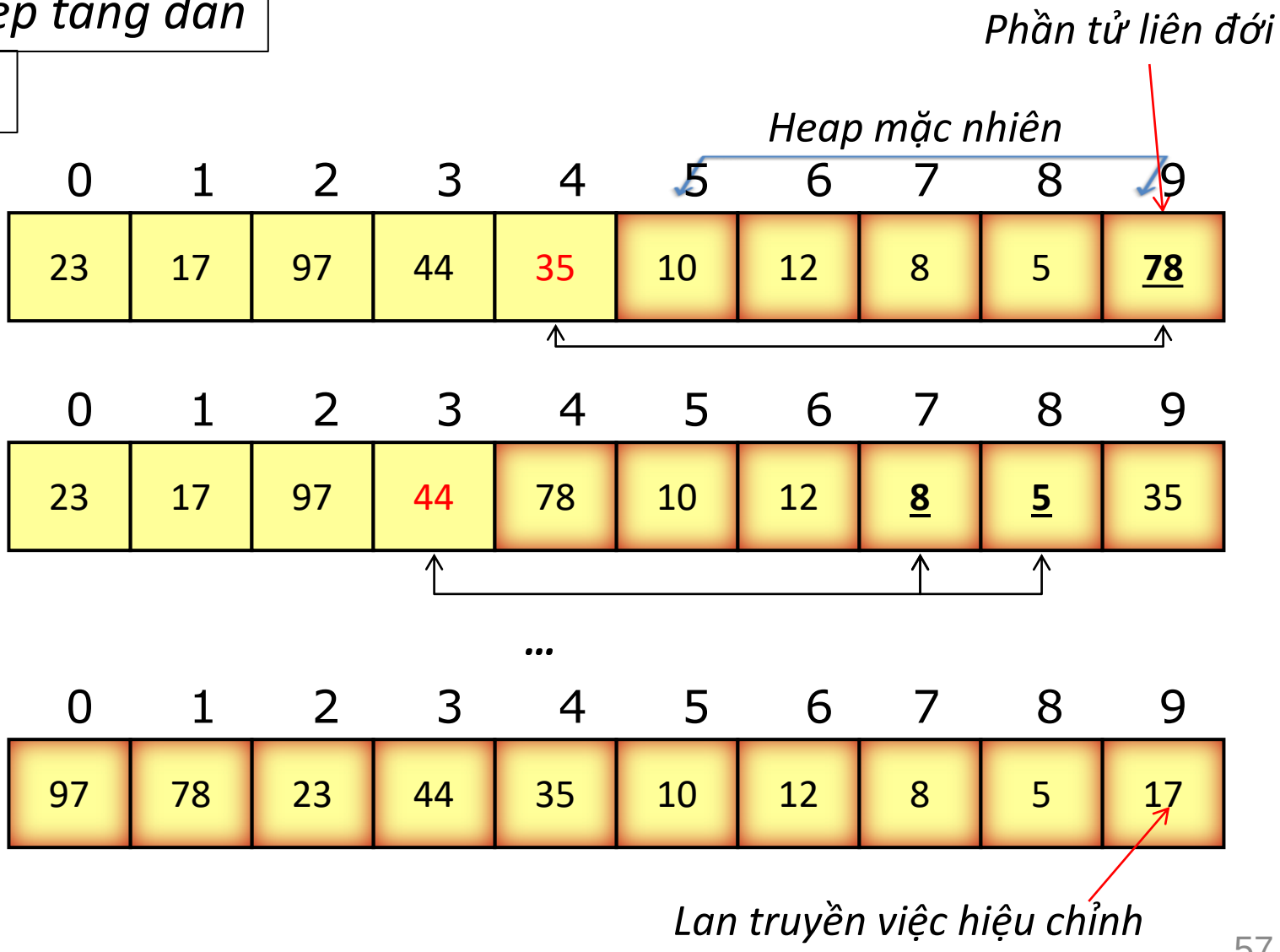
    Ngược lại, dừng.

Nhận xét: Dựa vào tính chất [3], ta suy ra được $a_{(n-1)/2+1}$, …, $a_{n-1}$ luôn là một heap, như vậy lần lượt thêm vào các phần tử $a_{(n-1)/2}$, …, $a_0$ đồng thời hiệu chỉnh lại, ta sẽ được heap theo mong muốn.

# Heap Sort – Sắp xếp cây

*VD 2: Sắp xếp tăng dần*

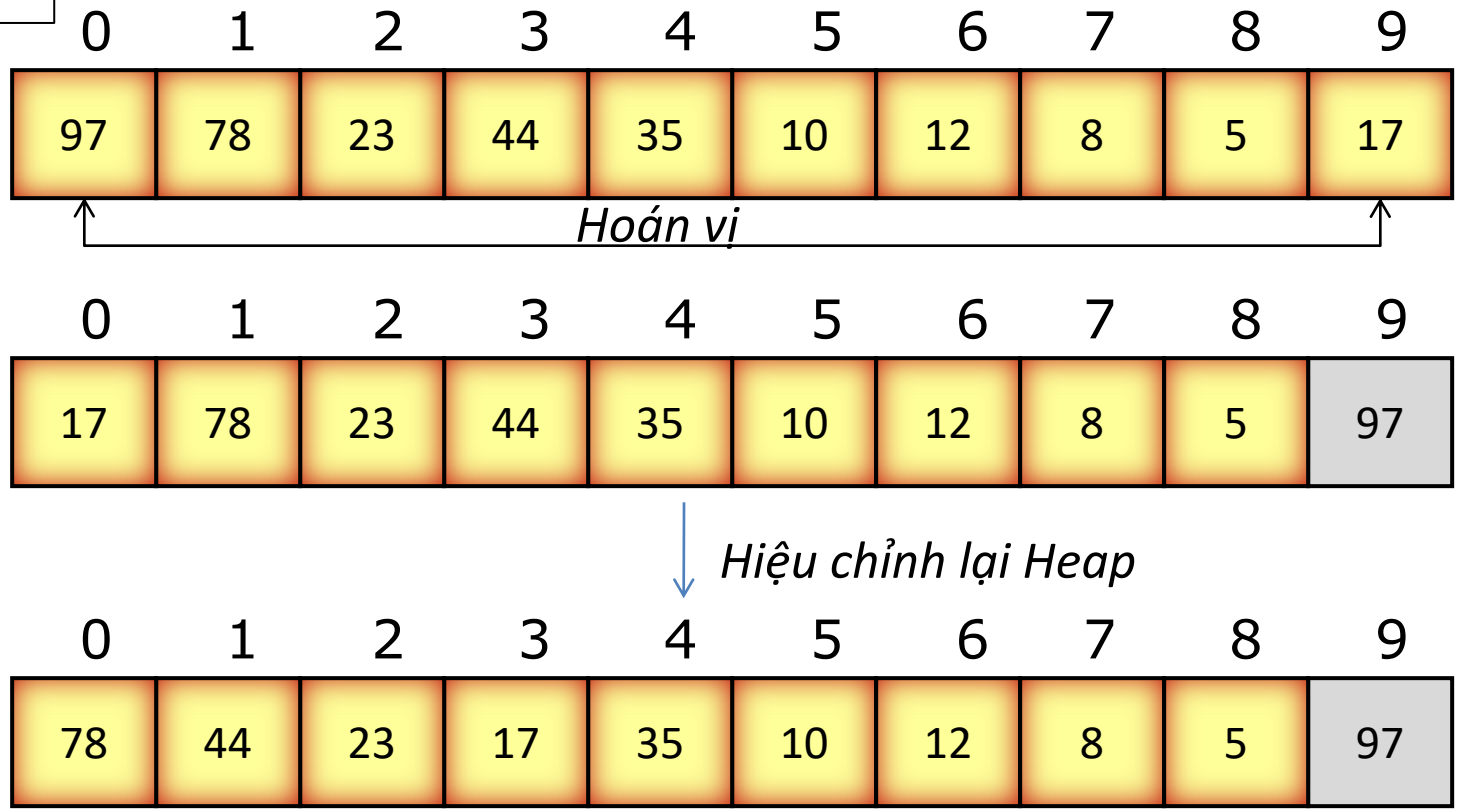
*Giai đoạn 1*



# Heap Sort – Sắp xếp cây

*VD: Sắp xếp tăng dần*

*Giai đoạn 2*



# Heap Sort – Sắp xếp cây

*VD: Sắp xếp tăng dần*

```
function HeapSort(…)
    CreateHeap(…);  //giai đoạn 1
    //giai đoạn 2
    for (i ← n-1 to 1) do
            a[0] ↔ a[i];
            siftdown(a, 0, i-1);          //hiệu chỉnh heap
    end for
end function
```

```
function CreateHeap(…)
    //từ (n-1)/2+1 đến cuối là heap mặc nhiên
    for (i ← (n-1)/2 to 0) do
            siftdown(a, i, n-1);          //hiệu chỉnh heap
    end for
end function
```

# Heap Sort – Sắp xếp cây

*VD: Sắp xếp tăng dần*

```
function siftdown(a, left, right)
    p ← 2*left + 1;                    //phần tử liên đới nhánh trái
    if (p > right) then return;
    end if
    if (a[p] < a[p+1]) then            //liên đới trái nhỏ hơn liên đới phải
            p ← p + 1;
    end if
    if (a[left] < a[p]) then
            a[left] ↔ a[p];            //hoán vị
            siftdown(a, p,right);      //lan truyền hiệu chỉnh
    end if
end function
```

# Nhận xét Heap Sort

- Ưu điểm:
  - Độ phức tạp thấp: $O(nlogn)$
- Khuyết điểm:
  - Mặc dù độ phức tạp thấp hơn so với Quick Sort nhưng thực tế thực thi của Quick Sort lại tốt hơn.

# Các thuật toán sắp xếp

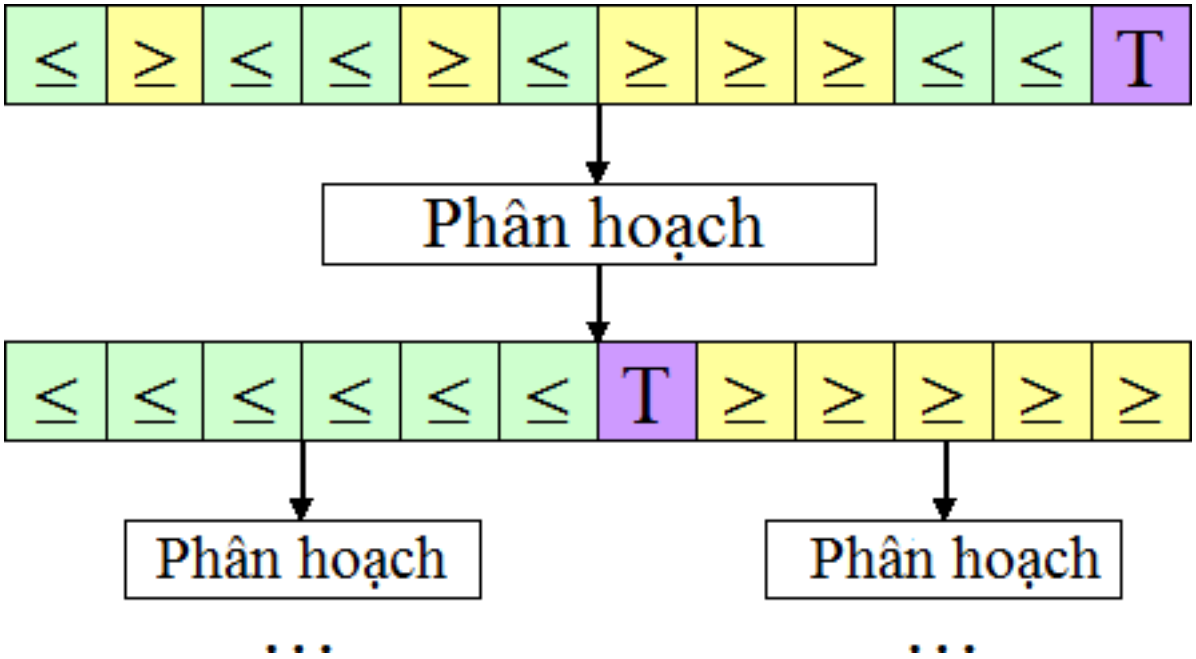
- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Quick Sort – SX dựa trên phân hoạch

Được gọi là phương pháp chia để trị (divide-and-conquer)

Ý tưởng:

- Phân hoạch dãy đầu thành hai dãy: dãy con 1 gồm các phần tử nhỏ hơn x, dãy con 2 gồm các phần tử lớn hơn x. (x là phần tử tùy ý trong dãy)
- Quá trình phân hoạch sẽ được lặp lại trên từng dãy con đến khi dãy con chỉ còn 1 phần tử.



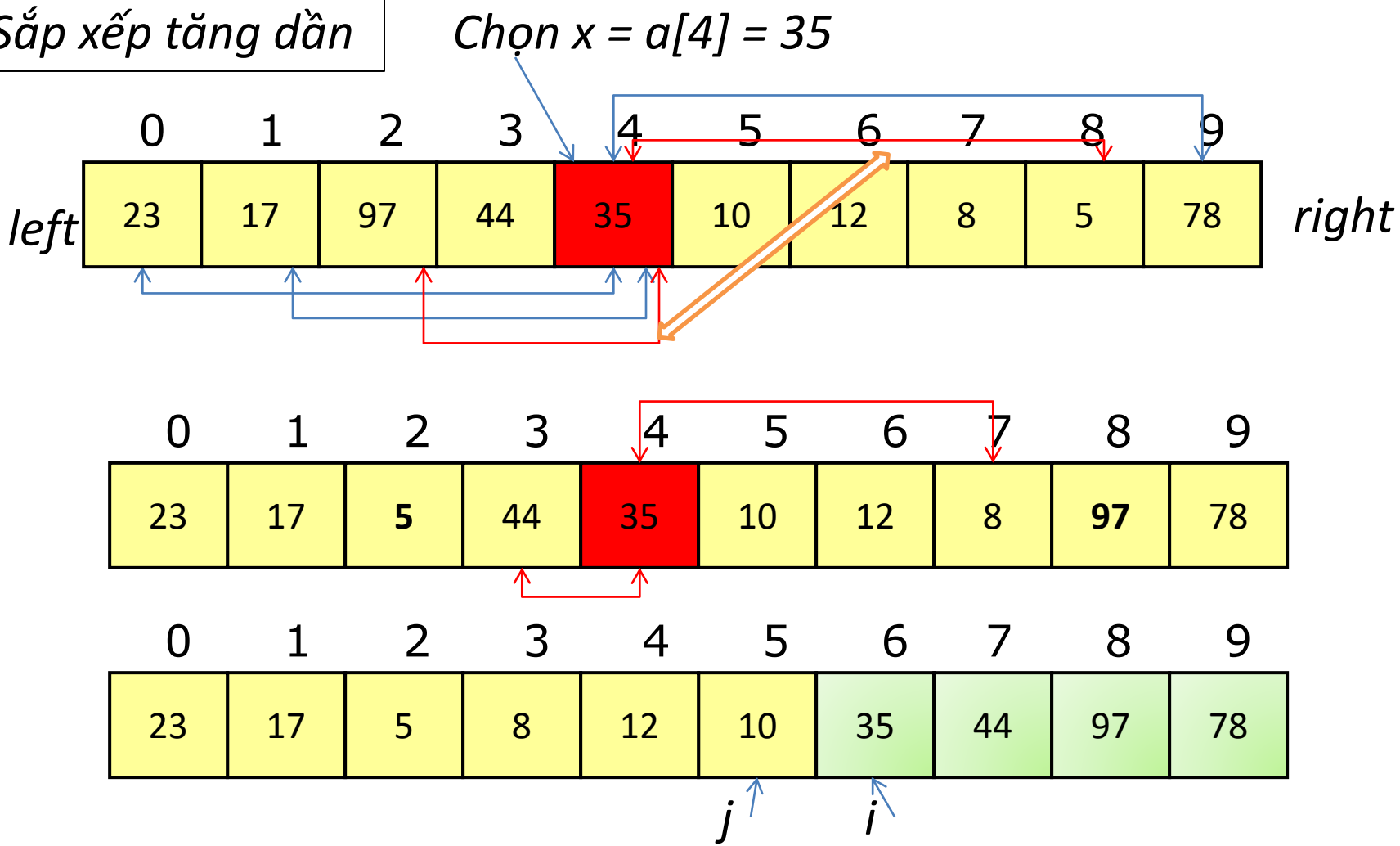
# Quick Sort – SX dựa trên phân hoạch

Các bước:

- B1: Chọn tùy ý một phần tử  x trong dãy làm giá trị mốc. i = left; j = right-1;
- B2: Tìm và hiệu chỉnh các cặp phần tử a[i], a[j] nằm sai chỗ.
  - B2a: Trong khi (a[i] < x) i++;
  - B2b: Trong khi (a[j] > x) j--;
  - B2c: Nếu i ≤ j thì hoán vị (a[i], a[j]);
- B3:  i ≤ j: quay về B2.
- B4: Đệ quy phân hoạch nhánh trái (left, j).
- B5: Đệ quy phân hoạch nhánh phải (i, right).

# Quick Sort – SX dựa trên phân hoạch

*VD: Sắp xếp tăng dần*



# Quick Sort – SX dựa trên phân hoạch

*VD: Sắp xếp tăng dần*

```
function QuickSort(a,left,right)
    i ← left; j ← right;
    while (i ≤ j) do
            while (a[i] < x) do i ← i+1; end while
            while (a[j] > x) do j ← j-1; end while
            if (i ≤ j) then
                    a[i] ↔ a[j];
                    i ← i+1; j ← j-1;
            end if
    end while
    if (left < j) then QuickSort(a,left,j); end if
    if (i < right) then QuickSort(a,i,right); end if
end function
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Merge Sort – Trộn trực tiếp

Ý tưởng:

- Phân hoạch dãy ban đầu thành hai dãy con. Lặp lại phân hoạch trên dãy con cho đến khi dãy còn 1 phần tử. (top-down)
- Trộn từng cặp dãy con của 2 dãy thành một dãy theo nguyên tắc và lặp lại lại cho hai dãy cha của nó, cho đến khi đạt đến dãy lúc đầu. (bottom-up)

6 5 3 1 8 7 2 4

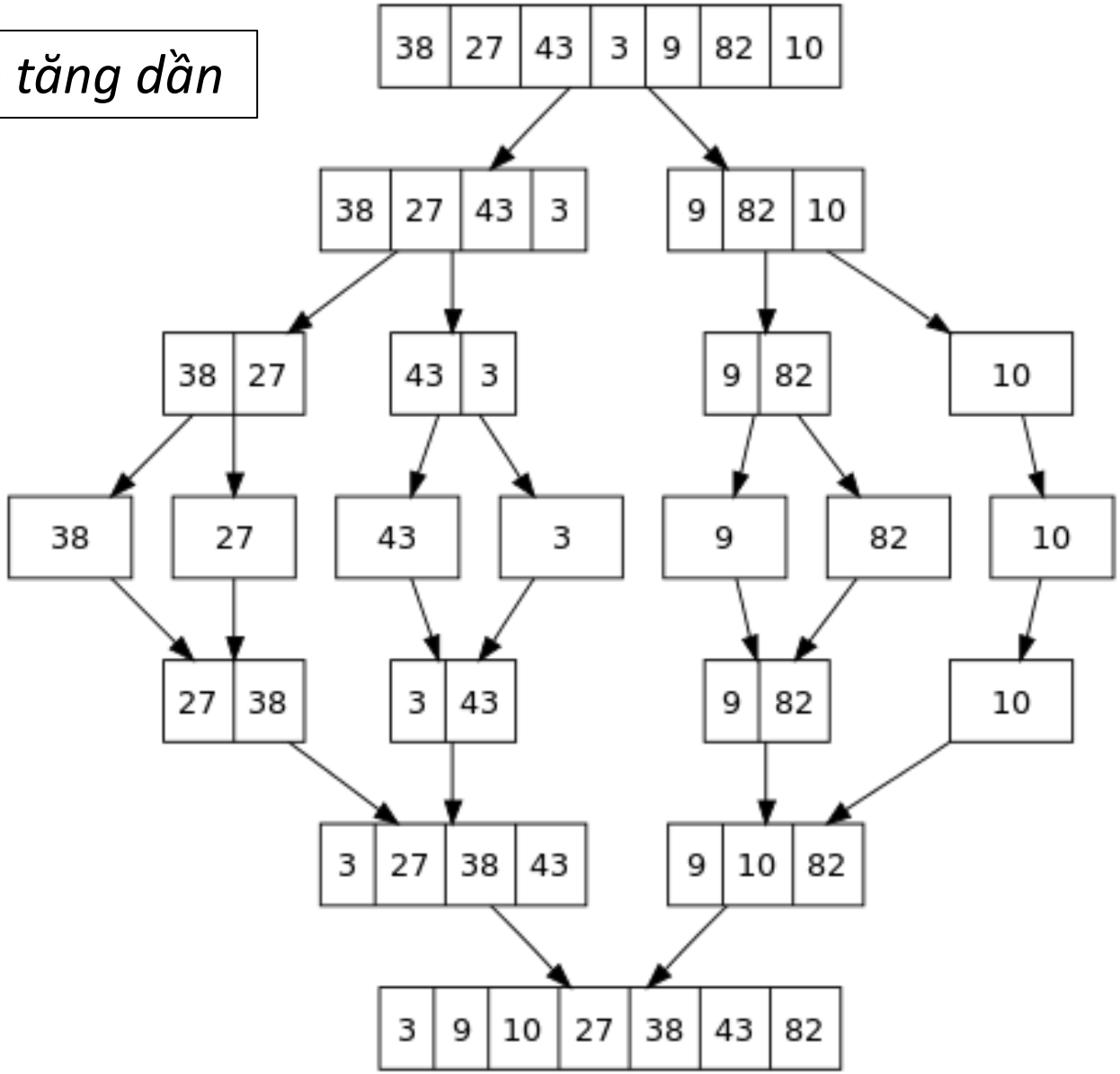
# Merge Sort – Trộn trực tiếp

Các bước:

- B1: mid = (l+r)/2; //chọn vị trí để tách
- B2:Tách dãy a thành 2 dãy con b,c
- B3: Nếu dãy con b,c có trên 2 phần tử thì lặp lại B2.
- B4: Trộn hai dãy con để được dãy cha có thứ tự.
  Lặp lại cho đến khi được mảng đầy đủ phần tử như ban đầu.

# Merge Sort – Trộn trực tiếp

*VD: Sắp xếp tăng dần*



# Merge Sort – Trộn trực tiếp

*VD: Sắp xếp tăng dần*

```
function MergeSort(a[],l,r,temp[])
    if (r-l <2) return;
    mid = (l+r)/2;
    MergeSort(a,l,mid,temp);          //chia và trộn dãy trái
    MergeSort(a,mid, r, temp);        //chia và trộn dãy phải
    Merge(a, l, mid, r, temp);        //trộn hai nửa đã chia
    CopyArray(temp, l,r,a);           //sao chép lại mảng a
end function
```

# Merge Sort – Trộn trực tiếp

*VD: Sắp xếp tăng dần*

```
function Merge(a[], l, mid, r, temp)
    iLeft ← l;        iRight ← mid;      //phần tử bắt đầu ở mỗi dãy con
    for (j ← l; j < r; j++) do
        if ( iLeft < mid && (iRight >= r || a[iLeft] <= a[iRight])) then
                temp[j] ← a[iLeft];
                iLeft++;
        else
                temp[j] ← a[iRight];
                iRight++;
        end if
    end for
end function
```

# Các thuật toán sắp xếp

- Selection Sort
- Insertion Sort
- Interchange Sort
- Bubble Sort
- Shaker Sort
- Binary Insertion Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Merge Sort
- Radix Sort

# Radix Sort – Sắp theo cơ số

Ý tưởng:

- Giả sử mỗi phần tử a trong dãy $a_0,\ldots, a_{n-1}$ là số nguyên có tối đa m chữ số.
- Phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm…của nó.

Các bước:

- B1: k = 0; //phân loại theo hàng đơn vị
- B2: Khởi tạo 10 lô $B_0,\ldots,B_9$ rỗng (cơ chế giống stack).
- B3: Lần lượt đặt các phần tử trong mảng a vào các lô $B_t$ với t là chữ số ở hàng k của nó.
- B4: Nối các số trong lô B theo đúng thứ tự thành a.
- B5: k = k+1; Nếu k<m thì quay lại B2;
  Ngược lại, dừng.

# Radix Sort – Sắp theo cơ số

*Hàng đơn vị*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 701 | 1725 | 999 | 9170 | 3252 | 4518 | 7009 | 1424 | 428 | 1239 | 8425 | 7013 |

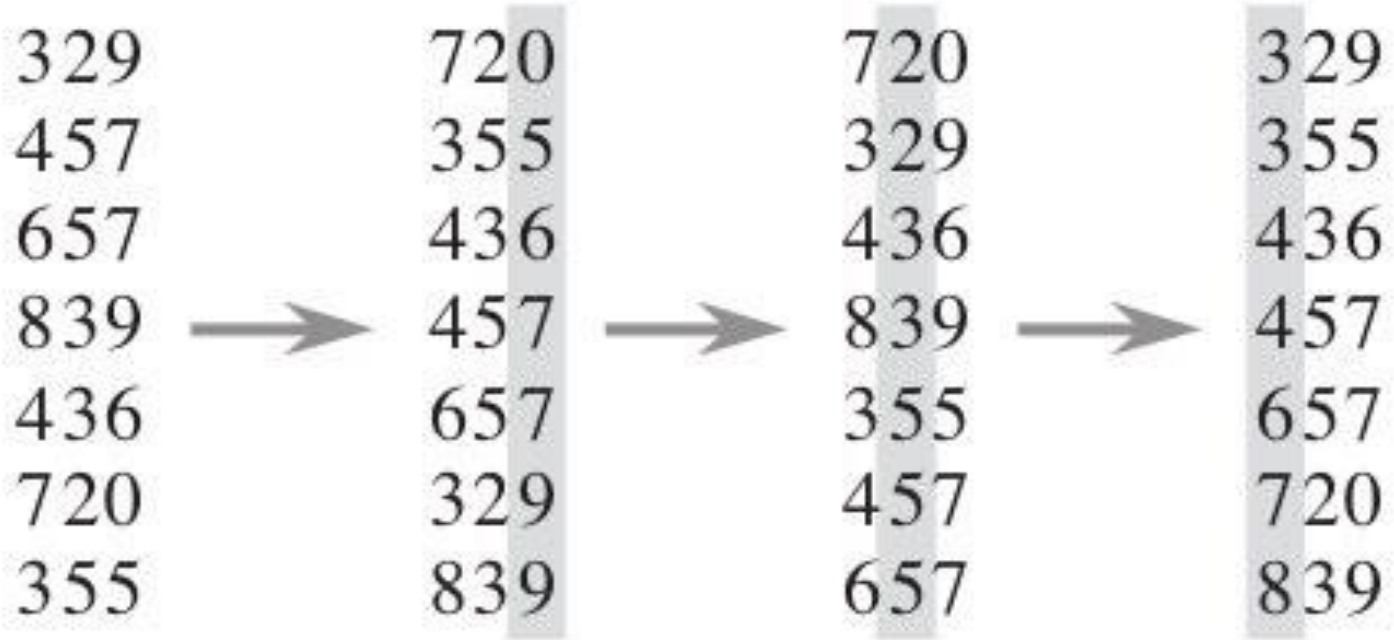
*Lô B*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | 0999 |
| | | | | | 1725 | | | 4518 | 7009 |
| 9170 | 0701 | 3252 | 7013 | 1424 | 8425 | | | 0428 | 1239 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9170 | 0701 | 3252 | 7013 | 1424 | 8425 | 1725 | 0428 | 4581 | 1239 | 7009 | 0999 |

*Hàng ...*

# Radix Sort – Sắp theo cơ số



# Radix Sort – Sắp theo cơ số

*VD: Sắp xếp tăng dần*

```
Khởi tạo B[0,…9];
for (t ← 0 to m-1) do
   for (i ← 0 to n-1) do
         Thêm a[i] vào B[Digit(a[i],t)];
   end for
    for (j ← 0 to 9) do
         Lấy ngược các phần tử từ B[j] đưa vào a;
   end for
end for
```

# Chủ đề nâng cao

- Phân tích độ phức tạp của các thuật toán trên
- Phân tích ưu và nhược điểm của mỗi phương pháp.

# Tài liệu trích dẫn


[1] http://en.wikipedia.org/wiki/Selection_sort

[2] http://en.wikipedia.org/wiki/Heap_(data_structure)

[3] http://www.tech-faq.com/heaps.shtml

[4] http://www.math.hcmuns.edu.vn/~ptbao/DataStructure/01.pdf

[5] Dương Anh Đức, Trần Hạnh Nhi, 2003, "Cấu Trúc Dữ Liệu và Thuật Toán", trường Đại Học Khoa Học Tự Nhiên Tp.HCM.

The End.

# DSA War

- Sử dụng tất cả các thuật toán sắp xếp, không giới hạn trong slide.