Chương 5 : Các thuật toán sắp xếp

Trịnh Anh Phúc [1]

[1]Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 10 tháng 8 năm 2015

Giới thiệu

1. Bài toán sắp xếp
2. Ba thuật toán sắp xếp cơ bản
3. Sắp xếp trộn
4. Sắp xếp nhanh
5. Sắp xếp vun đống
6. Cận dưới cho bài toán sắp xếp
7. Các phương pháp sắp xếp đặc biệt
8. Tổng kết

Bài toán sắp xếp

Định nghĩa bài toán sắp xếp

Sắp xếp (Sorting) là quá trình tổ chức lại họ các dữ liệu theo thứ tự giảm dần hoặc tăng dần (ascending or descending order). Dữ liệu cần sắp xếp có thể là :

- Số nguyên (Intergers)
- Xâu ký tự (String)
- Đối tượng (Object)

Ta cần có khóa sắp xếp (sort key) dùng để phân biệt các dữ liệu với nhau. Khóa này duy nhất cho từng dữ liệu và được dùng để sắp xếp. Lưu ý, không có khóa trùng lặp cho hai dữ liệu phân biệt chính bởi vậy các giải thuật sắp xếp mới thực hiện được.

Bài toán sắp xếp

Lưu ý khi biểu diễn bài toán sắp xếp trong máy tính

- Việc sắp xếp tiến hành trực tiếp trên bản ghi dữ liệu đòi hỏi các thao tác di chuyển tốn kém.
- Vì vậy người ta thường xây dựng một bảng khóa gồm các bản ghi chỉ gồm hai trường là (khóa, con trỏ) :
  - trường "khóa" chứa giá trị khóa
  - trường "con trỏ" chứa địa chỉ trỏ đến các bản ghi dữ liệu tương ứng
- Việc sắp xếp theo khóa trong bảng khóa trên không đòi hỏi di chuyển các bản ghi dữ liệu - trong bảng chính, nhưng trình tự các bản ghi trong bảng khóa cho phép xác định trình tự các bản ghi dữ liệu trong bản chính.

Bài toán sắp xếp

Mô tả giải thuật của bài toán sắp xếp

- **Đầu vào** : Dãy gồm n khóa $A = (a_1, a_2, \cdots, a_n)$
- **Đầu ra** : Một hoán vị của dãy A là dãy $A' = (a'_1, a'_2, \cdots, a'_n)$ sao cho dãy thỏa mãn

$$a'_1 \leq a'_2 \leq \cdots \leq a'_n$$

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 (tiếp)

Phân loại

- Sắp xếp trong (internal sort) : Đòi hỏi họ dữ liệu đc đưa toàn bộ vào bộ nhớ trong của máy tính
- Sắp xếp ngoài (external sort) : Họ dữ liệu không thể cũng lúc đưa toàn bộ vào bộ nhớ trong, nhưng có thể đọc vào từng bộ phận từ bộ nhớ ngoài

Đặc trưng

- Tại chỗ (in place) : nếu không gian nhớ phụ mà thuật toán đòi hỏi là O(1), nghĩa là 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 (tiếp)

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

- Đổi chỗ (swap) : thời gian thực hiện là O(1), ví dụ mã nguồn cài đặt trên C

```
void swap(datatype &a, datatype &b){
    datatype temp = a;
    a=b;
    b=temp;
}
```

- So sánh (compare) : hàm compare(a,b) trả lại true nếu a ở vị trí trước b theo thứ tự cần sắp xếp và false nếu trái lại.

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

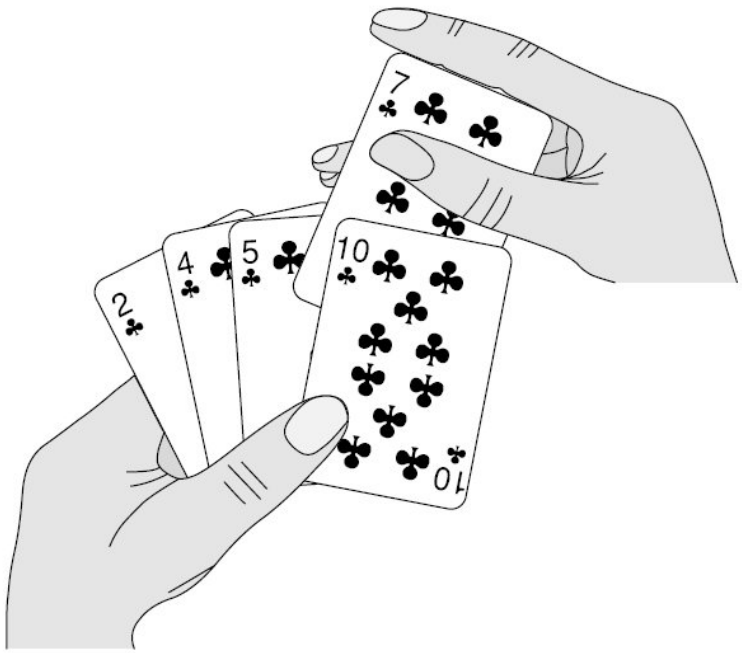
7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Ba thuật toán sắp xếp cơ bản

Sắp xếp chèn - insertion sort

Phỏng theo cách làm của người chơi bài, mỗi khi có một quân bài mới người chơi sẽ tìm vị trí thích hợp trong bộ bài đang cầm trên tay để chèn lá bài mới này vào sao cho giá trị quân bài tăng dần đều.



Ba thuật toán sắp xếp cơ bản

Sắp xếp chèn (tiếp)

Thuật toán

- Tại bước thứ $k = 1, 2, \cdots, n$ đưa phần tử thứ k trong mảng A đã cho vào đúng vị trí trong dãy gồm k phần tử.
- Kết quả sau mỗi bước k là k phần tử đầu tiên được sắp xếp đúng thứ tự.

Ba thuật toán sắp xếp cơ bản

Sắp xếp chèn (tiếp)

Mã giả của giải thuật sắp xếp chèn

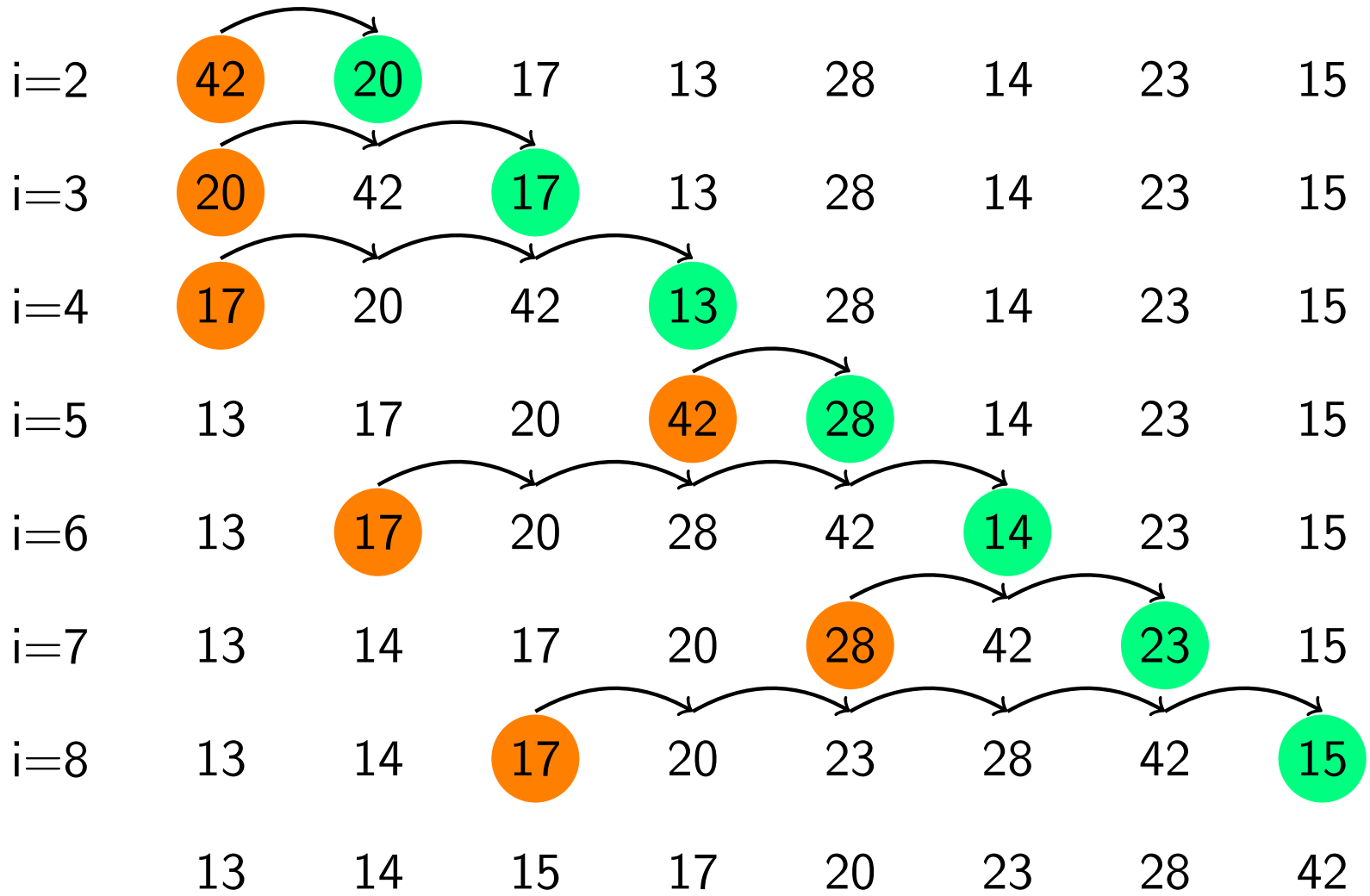
Procedure Insertion-Sort(A,n)

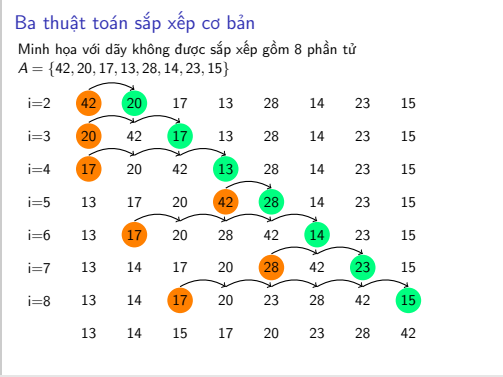
1. **for** i $\leftarrow$ 2 **to** n **do**
2. last $\leftarrow$ A[i]
3. j $\leftarrow$ i
4. **while** (j>1 **and** A[j-1] > last) **do**
5. A[j] $\leftarrow$ A[j-1]
6. j $\leftarrow$ j-1
7. **endwhile**
8. A[j] $\leftarrow$ last
9. **endfor**

End

Ba thuật toán sắp xếp cơ bản

Minh họa với dãy không được sắp xếp gồm 8 phần tử
$A = \{42, 20, 17, 13, 28, 14, 23, 15\}$





Các phép gán A[j] $\leftarrow$ A[j-1] được biểu diễn bằng các mũi tên một chiều.
Giá trị last $\leftarrow$ A[i] được tô mầu xanh sẽ được gán cuối cùng tại vị trí A[j] được tô mầu cam

Ba thuật toán sắp xếp cơ bản

Sắp xếp chèn (tiếp)

Các đặc tính của sắp xếp chèn

- Sắp xếp chèn là tại chỗ và ổn định. Nói cách khác nó luôn đúng và kết thúc.
- Thời gian của thuật toán
  - Trường hợp tốt nhất : 0 có hoán đổi hay dãy cho vào đã được sắp xếp rồi.
  - Trường hợp tồi nhất : Có $n^2/2$ hoán đổi và so sánh, khi dãy đầu vào có thứ tự ngược với chiều cần sắp xếp.
  - Trường hợp trung bình : Cần $n^2/4$ hoán đổi và so sánh.
- Rõ ràng thuật toán có thời gian tính tốt nhất trong trường hợp tốt nhất
- Là thuật toán tốt với dãy đã *gần được sắp xếp*, nghĩa là phần tử đưa vào gần với vị trí cần sắp xếp.

Ba thuật toán sắp xếp cơ bản

Sắp xếp lựa chọn - selection sort

Thuật toán lặp gồm đúng $i = 1, 2, \cdots, n-1$ vòng lặp

1. Tìm phần tử nhỏ nhất đưa vào vị trí 1
2. Tìm phần tử nhỏ thứ hai đưa vào vị trí 2
3. Tìm phần tử nhỏ thứ ba đưa vào vị trí 3 ....

Ba thuật toán sắp xếp cơ bản

Sắp xếp lựa chọn (tiếp)

Mã giả của giải thuật sắp xếp chọn

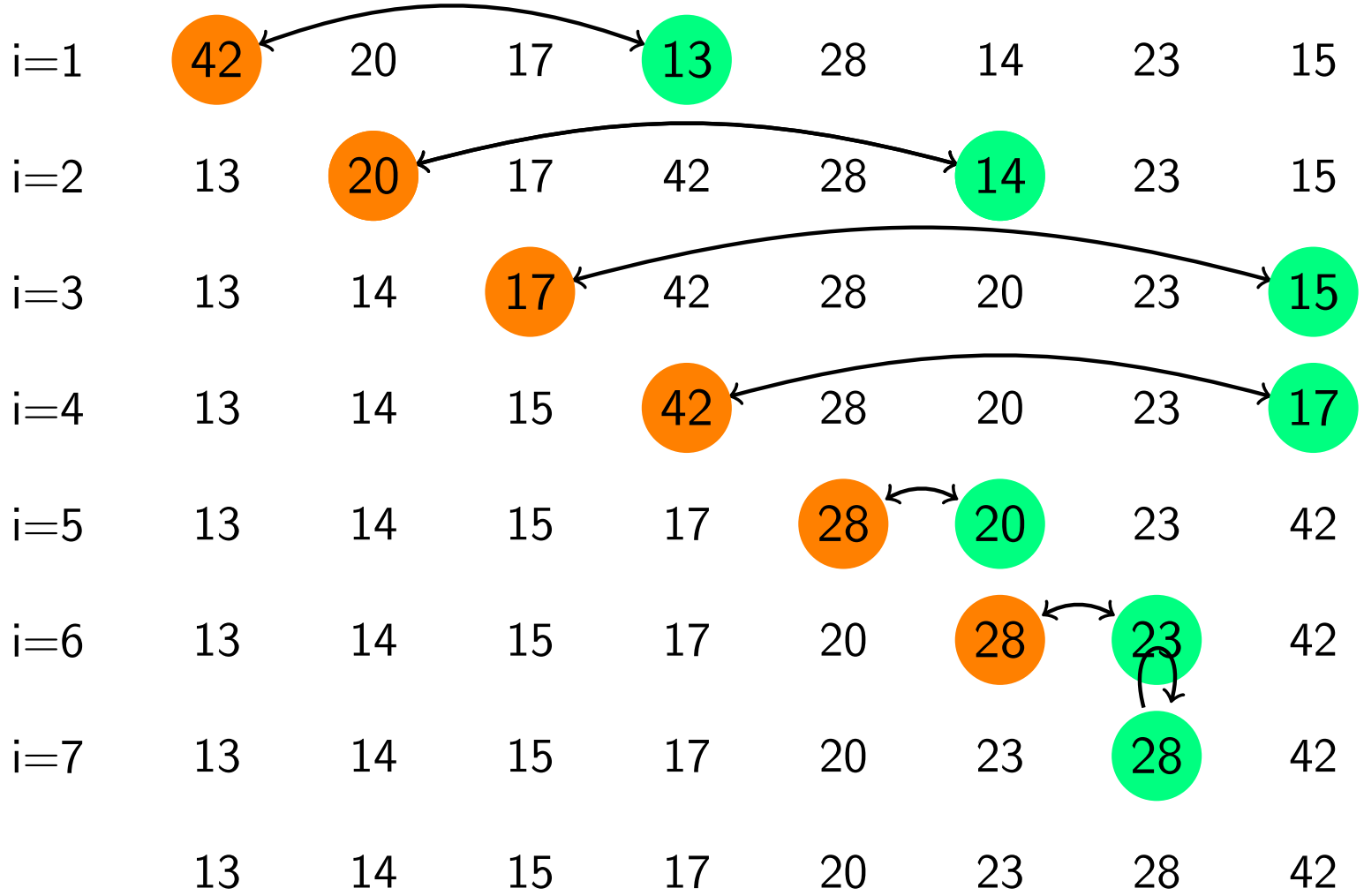
Procedure Selection-Sort(A,n)

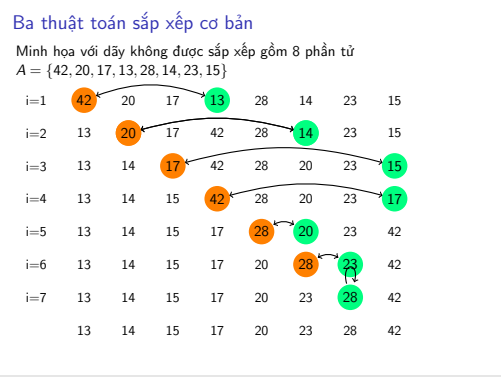
1. **for** i $\leftarrow$ 1 **to** n-1 **do**
2. min $\leftarrow$ i
3. **for** j $\leftarrow$ i+1 **to** n **do**
4. **if** (A[j]<A[min]) **then** min $\leftarrow$ j **endif**
5. **endfor**
6. swap(A[i],A[min]) /* Đổi chỗ */
7. **endfor**

End

Ba thuật toán sắp xếp cơ bản

Minh họa với dãy không được sắp xếp gồm 8 phần tử
$A = \{42, 20, 17, 13, 28, 14, 23, 15\}$





Các A[i] được tô mầu cam sẽ hoán đổi vị trí cho các A[min] tô mầu xanh tại mỗi bước lặp i. Mũi tên hai chiều chỉ phép đổi chỗ swap(A[i],A[min]) theo giải thuật.

Ba thuật toán sắp xếp cơ bản

Sắp xếp lựa chọn (tiếp)

Phân tích thuật toán

- Trường hợp tốt nhất : 0 đổi chỗ, $n^2/2$ phép so sánh
- Trường hợp tồi nhất : $n-1$ phép đổi chỗ và $n^2/2$ phép so sánh
- Trường hợp trung bình : $O(n)$ phép đổi chỗ và $n^2/2$ phép so sánh

Ưu điểm của sắp xếp lựa chọn là đổi chỗ ít.

Ba thuật toán sắp xếp cơ bản

Sắp xếp nổi bọt - bubble sort

Sắp xếp nổi bọt là phương pháp sắp xếp đơn giản thường được sử dụng như trong giáo trình nhập môn công nghệ thông tin. Thuật toán được trình bầy như sau :

1. Bắt đầu duyệt từ đầu dãy, ta so sánh phần tử tại vị trí hiện tại với phần tử ở vị trí kế tiếp đi sau nó, nếu chúng không đúng thứ tự thì đổi chỗ cho nhau.
2. Tiếp tục duyệt cho tới khi không còn phải đổi chỗ trong một lần duyệt, hay dãy đã đc sắp xếp xong.

Tuy đơn giản nhưng đây là *thuật toán sắp xếp kém hiệu quả nhất* trong số ba thuật toán sắp xếp cơ bản.

Ba thuật toán sắp xếp cơ bản

Sắp xếp nổi bọt (tiếp)

Mã giả của giải thuật

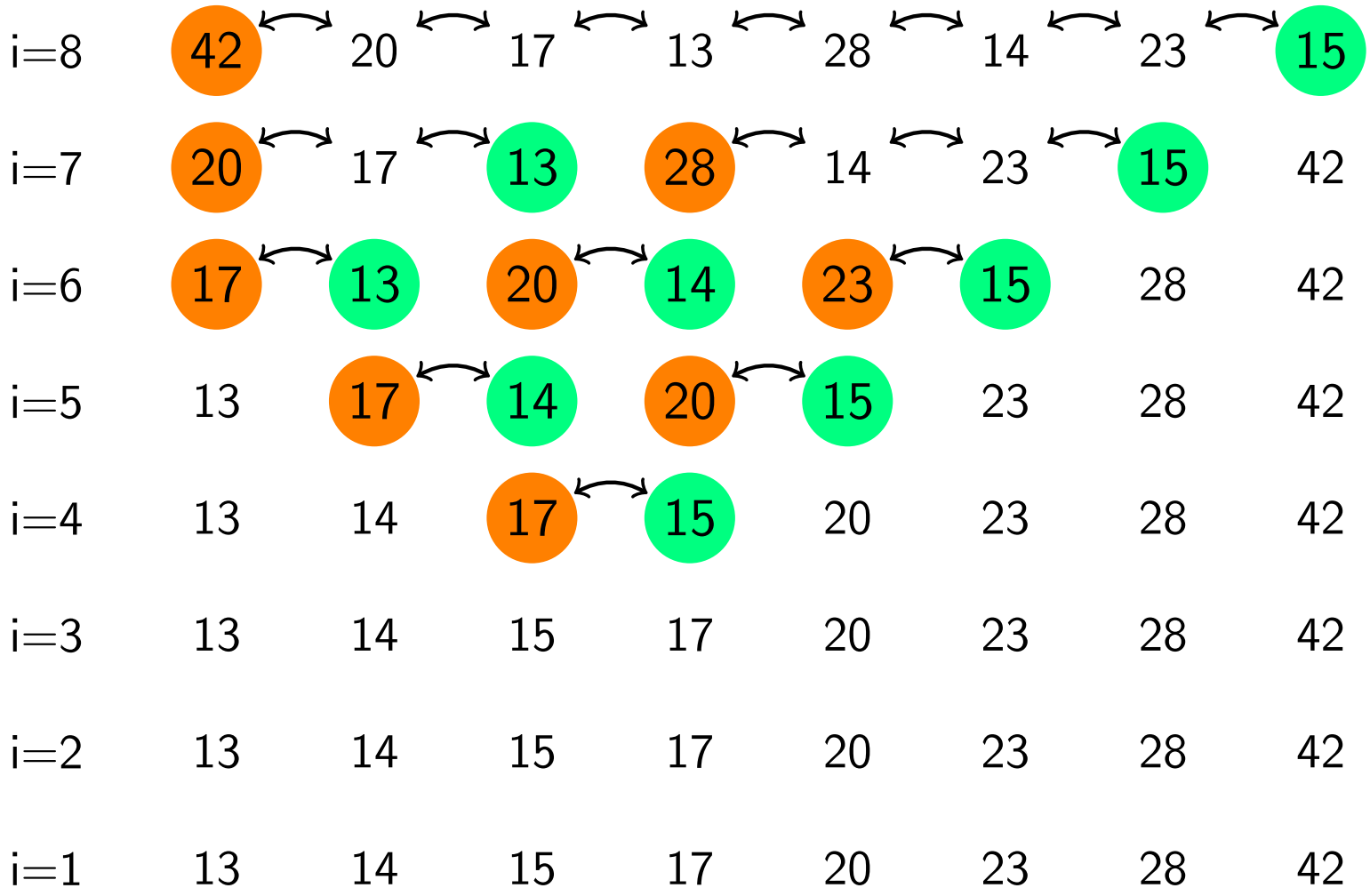
Procedure Bubble-Sort(A,n)

1. **for** i $\leftarrow$ n **to** 1 **do**
2. **for** j $\leftarrow$ 2 **to** i **do**
3. **if** (A[j-1] $>$ A[j]) **then**
4. swap(A[j-1],A[j])
5. **endif**
6. **endfor**
7. **endfor**

End

Ba thuật toán sắp xếp cơ bản

Minh họa với dãy không được sắp xếp gồm 8 phần tử
$A = \{42, 20, 17, 13, 28, 14, 23, 15\}$



Tất cả các phép hoán đổi swap(A[j],A[j-1]) được tiến hàng từ trái qua phải đc thể hiện bởi mũi tên hai chiều. Mỗi bước lặp giá trị lớn sẽ "nổi" trái sang phải từ vị trí tô mầu cam sang vị trí tô mầu xanh. Chú ý, cùng bước lặp i có thể có một vài giá trị cùng "nổi". Giải thuật thực ra đã kết thúc ở bước i=3 tuy nhiên ta chưa cài đặt thuật toán cải tiến để nó dừng tại đó.

Ba thuật toán sắp xếp cơ bản

Sắp xếp nổi bọt (tiếp)

Phân tích thuật toán

- Trường hợp tốt nhất : 0 đổi chỗ, $n^2/2$ so sánh
- Trường hợp tồi nhất : $n^2/2$ so sánh và đổi chỗ
- Trường hợp trung bình : $n^2/4$ đổi chỗ, $n^2/2$ so sánh

Tổng kết ba thuật toán sắp xếp cơ bản

Trường hợp	Chèn	Nổi bọt	Lựa chon
Số lần so sánh			
Tốt nhất	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Trung bình	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Tồi nhất	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Số lần đổi chỗ			
Tốt nhất	0	0	$\Theta(n)$
Trung bình	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Tồi nhất	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Sắp xếp trộn

Sắp xếp trộn - merge sort

Bài toán : Cần sắp xếp mảng $A[1..n]$, thuật toán trộn được phát triển dựa vào phương pháp chia-để-trị (đã được giới thiệu trong chương đệ qui) bao gồm các thao tác sau :

- Neo đệ qui (Base case) : Nếu dãy chỉ có một phần tử được coi là dãy đã được sắp xếp
- Chia (Divide) Chia dãy ban đầu n thành hai dãy có $n/2$ phần tử.
- Trị (Conquer)
  - Sắp xếp mỗi dãy con một cách đệ qui 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 xếp để thu được dãy được sắp xếp gồm tất cả các phần tử của hai dãy con.

Sắp xếp trộn

Sắp xếp trộn (tiếp)

Mã giả của giải thuật đệ qui sắp xếp trộn

```
MERGE-SORT(A,first,last)
    if first < last then
        mid ← (first+last)/2
        MERGE-SORT(A,first, mid)
        MERGE-SORT(A,mid+1, last)
        MERGE(A,first,mid,last)
    endif
End
```

Sắp xếp trộn

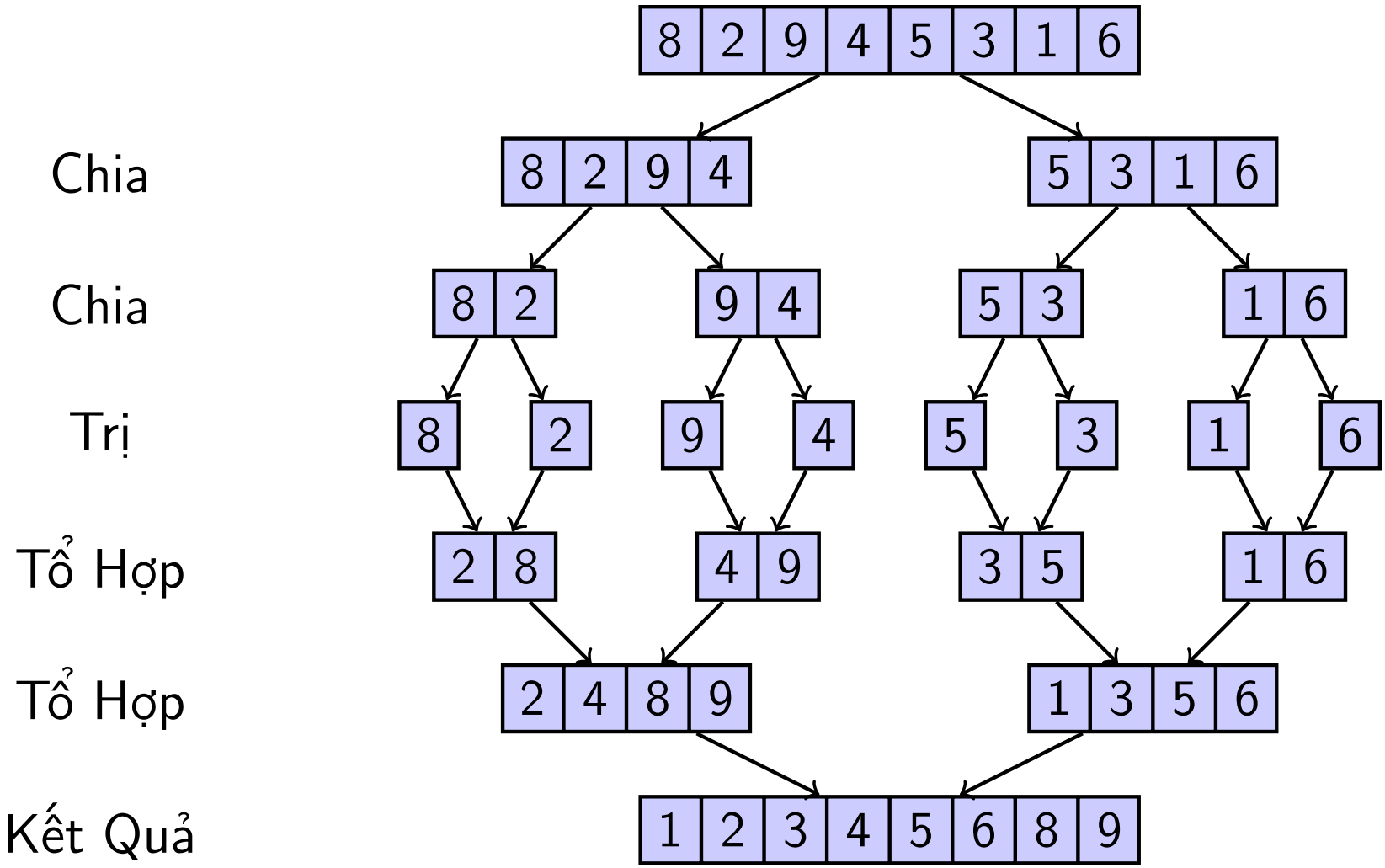
Procedure MERGE(A,first,mid,last)

1. Tính i $\leftarrow$ first và j $\leftarrow$ mid+1 sao cho i trỏ vào phần tử đầu tiên mảng trái L[1..n1] và j trỏ vào phần tử đầu tiên mảng bên phải R[1..n2] còn n1 $\leftarrow$ mid và n2 $\leftarrow$ last. Thêm L[n1+1] $\leftarrow \infty$ và R[n2+1] $\leftarrow \infty$
2. i $\leftarrow$ 1; j $\leftarrow$ 1;
3. **for** k$\leftarrow$ first **to** last **do**
4. **if** (L[i] $\leq$ R[j]) **then**
5. A[k] $\leftarrow$ L[i]
6. i $\leftarrow$ i+ 1
7. **else** A[k] $\leftarrow$ R[j]
8. j$\leftarrow$ j+1
9. **endif**
10. **endfor**

End

Sắp xếp trộn

Minh họa sắp xếp trộn của dãy {8,2,9,4,5,3,1,6}.



Sắp xếp trộn

Sắp xếp trộn (tiếp)

Thời gian tính của phép trộn - merge()

- Khởi tạo hai mảng tạm thời : $\Theta(n_1 + n_2) = \Theta(n)$
- 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 $\Theta(n)$

Sắp xếp trộn

Sắp xếp trộn (tiếp)

Thời gian tính của sắp xếp trộn - merge-sort()

- Chia : Tính mid như là giá trị trung bình của first và last : $\Theta(1)$
- Trị : Giải đệ qui hai bài toán con, mỗi bài kích thước $n/2 \Rightarrow 2T(n/2)$
- Tổ hợp : Trộn **MERGE** trên các mảng có kích thước n phần tử đòi hỏi thời gian $\Theta(n)$

Do đó ta có công thức đệ qui :

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

Suy ra : $T(n) = \Theta(n \log n)$ (Chứng minh bằng qui nạp)

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Sắp xếp nhanh

Sắp xếp nhanh - quick sort

Sơ đồ tổng quát

Thuật toán sắp xếp nhanh được phát triển bởi C.A.R.Hoare vào năm 1960. Theo thông kê tính toán, đây là giải thuật sắp xếp tính nhanh nhất hiện nay. Thuật toán cũng được phát triển dựa theo phương pháp chia để trị

1. Neo đệ qui (Base Case) : Nếu dãy chỉ còn không quá một phần tử thì nó là dãy đã được sắp xếp và trả ngay dãy mà không phải làm gì cả.
2. Chia (Divide) :
   - Chọn một phần tử trong dãy làm chốt p (pivot)
   - Chia dãy đã cho thành hai dãy con : Dãy con trái (L) gồm các phần tử nhỏ hơn chốt, ngược lại các phần tử thuộc dãy con phải (R) gồm các phần tử lớn hơn chốt. Thao tác gọi là phân đoạn - Partition.
3. Trị (Conquer) : lặp lại một cách đệ qui thuật toán đối với hai dãy con L và R.
4. Tổ hợp (Combine) : Dãy được sắp xếp là LpR

Sắp xếp nhanh

Sắp xếp nhanh (tiếp)

Khác với sắp xếp trộn, trong giải thuật sắp xếp nhanh thao tác chia là phức tạp, nhưng thao tác tổ hợp lại đơn giản. Điểm mấu chốt của thuật toán chính là thao tác chia.

Quick-Sort(A,left,right)

1. **if** (left $<$ right)
2. p = Partition(A,left,right)
3. Quick-Sort(A,left,p-1) // dãy con trái
4. Quick-Sort(A,p+1,right) // dãy con phải
5. **endif**

End

Sắp xếp nhanh

Sắp xếp nhanh (tiếp)

Một cải tiến mà D.Knuth đề nghị là nên dùng giải thuật sắp xếp khác khi số phần tử không quá lớn $n_0 = 9$, ví dụ khi áp dụng giải thuật chèn

Quick-Sort(A,left,right)

1. **if** (left - right $< n_0$)
2. insertionSort(A,left,right)
3. **else**
4. p = Partition(A,left,right)
5. Quick-Sort(A,left,p-1) // dãy con trái
6. Quick-Sort(A,p+1,right) // dãy con phải
7. **endif**

end

Sắp xếp nhanh

Thao tác phân đoạn

Thao tác phân đoạn bao gồm hai công việc

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

Thao tác có thể cài đặt tại chỗ với thời gian $\Theta(n)$, hiệu quả của nó phụ thuộc rất nhiều vào việc chọn chốt p. Người ta thường có các cách chọn chốt như sau :

- Chọn phần tử trái nhất
- Chọn phần tử phải nhất
- Chọn phần tử giữa
- Chọn phần tử trung vị (median) trong 3 phần tử đâu, cuối hoặc giữa.
- Chọn ngẫu nhiên một phần tử

Sắp xếp nhanh

Thao tác phân đoạn (tiếp)

Ta xây dựng hàm Partition(a,left,right) 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 *pivot* và trả lại chỉ số *jpivot* sao cho :
  - $a[jpivot]$ chứa giá trị chốt p
  - $a[i] \leq a[jpivot]$ với mọi $left \leq i < p$
  - $a[j] > a[jpivot]$ với mọi $p < j \leq right$

  trong đó p là giá trị chốt được chọn trước đó.

Sắp xếp nhanh

Thao tác phân đoạn : Phần tử chốt là đứng đầu

Sau đây là đoạn mã giả của thao tác phân đoạn với phần tử chốt là đứng đầu dãy

Function partition(a,left,right)

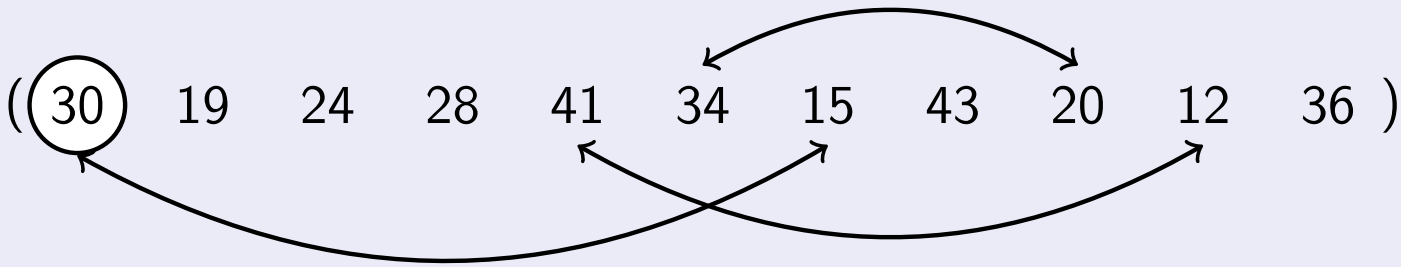
1. i $\leftarrow$ left; pivot $\leftarrow$ a[left]; j $\leftarrow$ right;
2. **while** (i $<$ j) **do**
3. **while** (i $\leq$ right **and** a[i] $\leq$ pivot) **do** i $\leftarrow$ i $+$ 1 **endwhile**
4. **while** (j $\geq$ left **and** a[j] $>$ pivot) **do** j $\leftarrow$ j - 1 **endwhile**
5. **if** (i<j) **then** swap(a[i],a[j]) **endif**
6. **endwhile**
7. swap(a[left],a[j])
8. **return** j

End

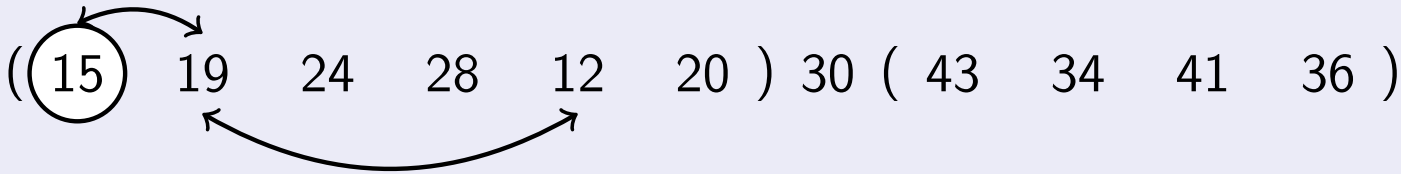
Sắp xếp nhanh

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

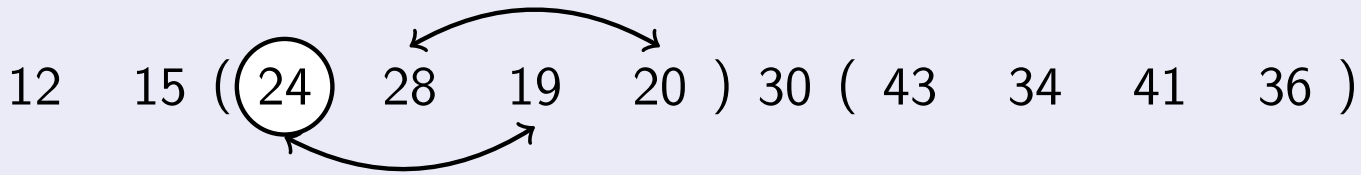
- Bước 1 :

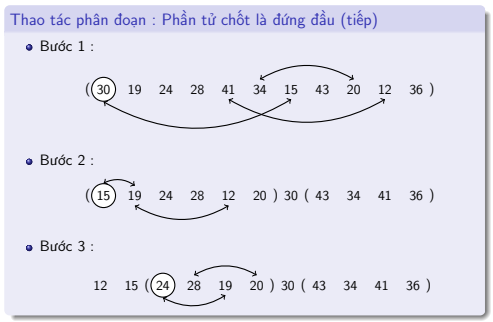


- Bước 2 :



- Bước 3 :



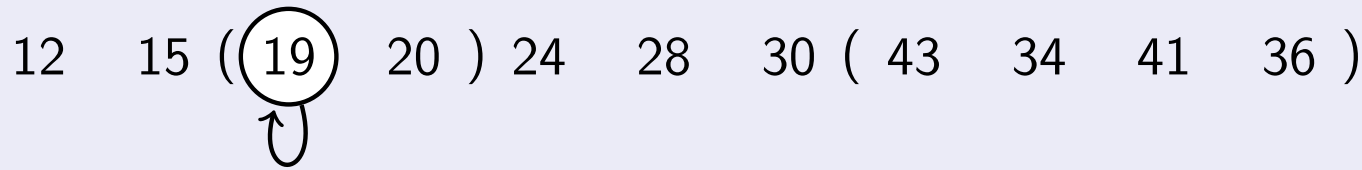


Các phần tử chốt đc minh họa trong vòng tròn. Các dãy phần tử trong dấu ngoặc đơn chỉ dãy chưa được sắp xếp có nhiều hơn một phần tử. Cuối mỗi bước phần tử chốt được chuyển về đúng vị trí của nó trong dãy số. Với các dãy chỉ còn một phần tử cũng vậy, phần tử đó cũng đã được đưa về đúng vị trí.

Sắp xếp nhanh

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

Sắp xếp nhanh

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 lại phụ thuộc vào việc phần tử nào được chọn làm chốt.

1. Phân đoạn không cân bằng (unbalanced partition): thực sự không có phần nào cả, do đó một bài toán con có kích thước $n-1$ còn bài toán kia có kích thước 0.
2. Phân đoạn hoàn hảo (perfect partition) : việc phân đoạn luôn được thực hiện dưới dạng phân đôi, như vậy mỗi bài toán con có kích thước cỡ $n/2$
3. Phân đoạn cân bằng (balanced partition) : việc phân đoạn được thực hiện ở đâu đó quanh điểm giữa, nghĩa là một bài toán con có kích thước $n-k$ còn bài toán có kích thước k.

Sắp xếp nhanh

Phân đoạn không cân bằng - unbalanced partition

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

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

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

Sắp xếp nhanh

Phân đoạn hoàn hảo - perfect partition

Công thức đệ qui 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 công thức đệ qui cho thời gian tính trong tình huống này là $T(n) = \Theta(n \log n)$

Sắp xếp nhanh

Phân đoạn cân bằng - balanced partition

Giả sử chốt pivot đc chọn ngẫu nhiên trong số các phần tử của dãy đầu vào. Các tình huống sau đông khả năng

- pivot là phần tử nhỏ nhất trong dãy
- pivot là phần tử nhỏ nhì trong dãy
- ...
-
- pivot là phần tử lớn nhất trong dãy

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

Sắp xếp nhanh

Phân đoạn cân bằng - balanced partition (tiếp)

Khi đó thời gian tính trung bình sẽ có công thức

$\sum$(thời gian phân đoạn kích thước i)$\times$(xác suất có phân đoạn kích thước i)

Khi dãy 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 $T(n) = T(i) + T(n-i-1) + cn$, áp kỳ vọng vào công thức

$$\begin{aligned}\mathbb{E}(T(n)) &= \sum_{i=0}^{n-1} \frac{1}{n}[\mathbb{E}(T(n)) + \mathbb{E}(T(n-i-1)) + cn] \\ &\leq \sum_{i=0}^{n-1} \frac{2}{n}[\mathbb{E}(T(n)) + cn]\end{aligned}$$

Giải công thức đệ qui ta thu đc : $\mathbb{E}(T(n)) = O(n \log n)$

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Sắp xếp vun đống

Cấu trúc dữ liệu đống - heap

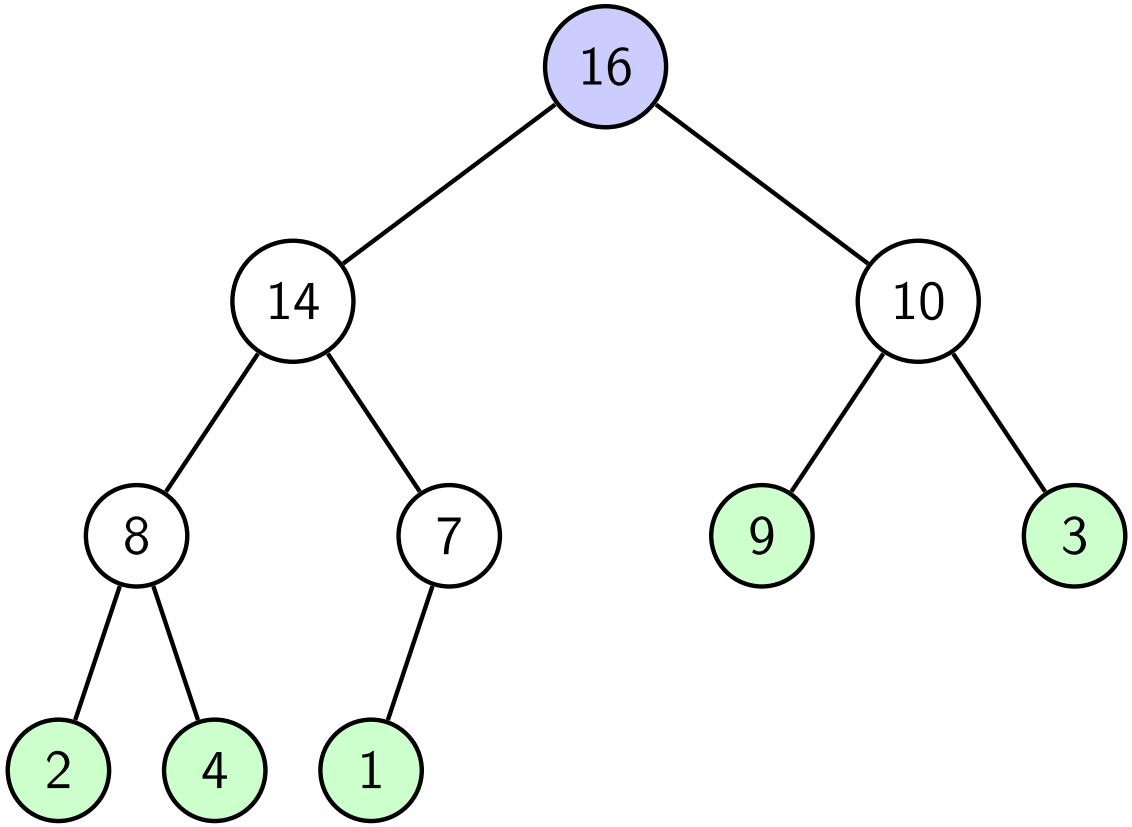
Định nghĩa : Đống (heap) là cây nhị phân *gần hoàn chỉnh* có hai tính chất

- Tính cấu trúc (structural property) : tất cả các mức đều đầy, ngoại trừ mức cuối cùng, mức cuối được điền từ trái sanh phải.
- Tính có thứ tự hay tính chất đống (heap property) : với mọi nút x thì giá trị của nút cha lớn hơn giá trị của nút con $parent(x) \geq x$.

Đống được cài đặt bởi mảng $A[i]$ có độ dài $length[A]$ như vậy gốc của đống có giá trị lớn nhất.

Sắp xếp vun đống

Minh họa đống



16	14	10	8	7	9	3	2	4	1

Sắp xếp vun đống

Cấu trúc dữ liệu đống (tiếp)

Như vậy ta có các giá trị như sau

- Gốc của cây $A[1]$
- Con trái của $A[i]$ là $A[2 * i]$
- Con phải của $A[i]$ là $A[2 * i + 1]$
- Cha của $A[i]$ là $A[i/2]$
- Các phần tử có chỉ số từ $n/2 + 1, \cdots, n$ trong mảng A là các lá.

Như vậy các hàm cơ bản được cài đặt như sau

- parent(i) $=$ i$/2$
- left-child(i) $=$ 2i
- right-child(i) $=$ 2i $+$ 1

Sắp xếp vun đống

Cấu trúc dữ liệu đống (tiếp)

Các phép toán đối với đống

- Bổ sung và loại bỏ nút
  - Nút mới được bổ sung vào mức đáy (từ trái sang phải)
  - Các nút được loại bỏ khỏi mức đáy (từ phải sang trái)
- Phép toán đối với đống
  - Khôi phục tính chất đống (vun lại đống) : Max-Heapify
  - Xây dựng đống (tạo đống ban đầu) : Build-Max-Heap

Sắp xếp vun đống

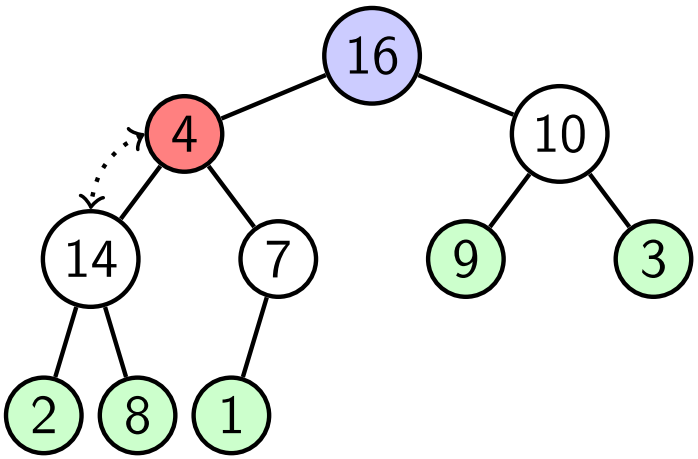
Khôi phục tính chất đống

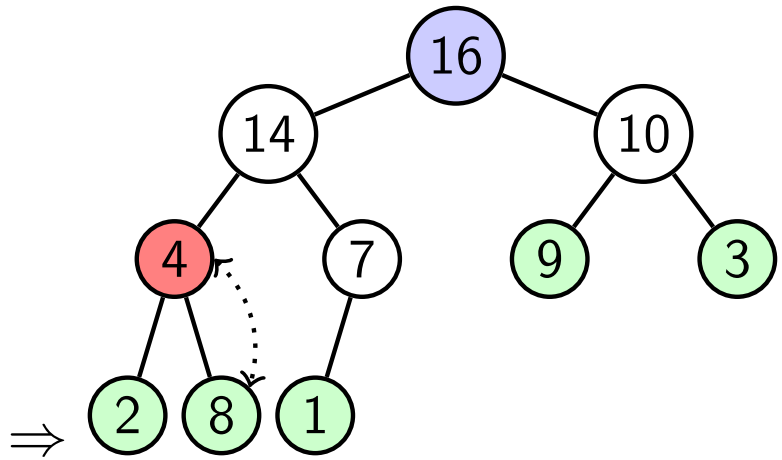
Giả sử nút thứ i có giá trị bé hơn con của nó. Giả thiết là cây con trái và cây con phải của i đều có phần tử lớn nhất đã ở gốc.

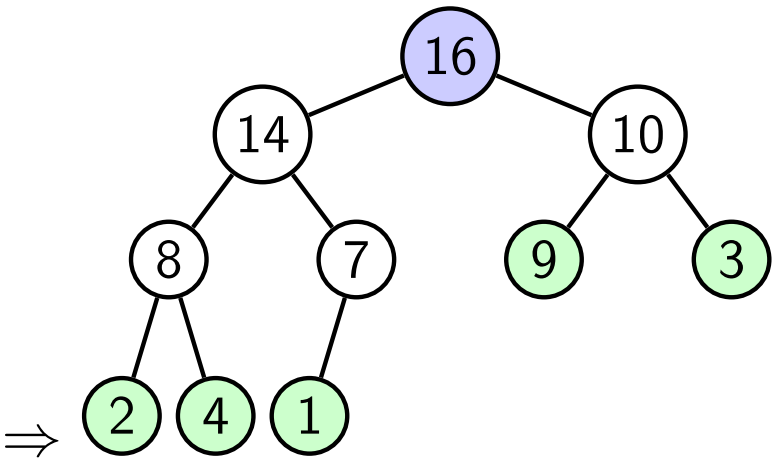
- Đổi chỗ với con lớn hơn
- Di chuyển xuống theo cây
- Tiếp tục qúa trình cho đến khi nút không có nút con lớn hơn

Sắp xếp vun đống

Ví dụ minh họa, nút đỏ là nút vi phạm tính chất đống. Nét đứt có mũi tên chỉ việc tráo đổi giá trị giữa hai nút trên cây. Thứ tự hình minh họa là trái qua phải, trên xuống dưới theo hình mũi tên







Sắp xếp vun đống

Khôi phục tính chất đống (tiếp)

Chú ý giả thiết là hai cây con đều có phần tử lớn nhất là gốc. Vậy mã giả của giải thuật như sau

1. **Max-Heapify**(A,i,n)
2. left $\rightarrow$ left-child(i); right $\rightarrow$ right-child(i);
3. **if** (left $\leq$ n **and** A[left] $>$ A[i]) **then** largest $\leftarrow$ left
   else largest $\leftarrow$ i **endif**
4. **if** (right $\leq$ n **and** A[right] $>$ A[largest]) **then** largest $\leftarrow$ right
5. **if** (largest **not** i) **then**
   swap(A[i],A[largest]); Max-Heapify(A,largest,n);
   endif
6. **End**

trong đó n là số nút của đống

Sắp xếp vun đống

Khôi phục tính chất đống (tiếp)

Thời gian tính của thuật toán đệ qui khôi phục tính chất đống Max-Heapify()

- Từ nút i phải di chuyển theo đường đi xuống phía dưới của cây. Độ dài của đường đi này không vượt quá đường đi từ gốc đến lá.
- Ở mỗi mức phải thực hiện hai phép so sánh
- Do đó tổng số phép so sánh không vượt quá $2h$ với h là chiều cao của cây
- Thời gian tính là $O(h)$ hay $O(\log n)$

Sắp xếp vun đống

Xây dựng đống

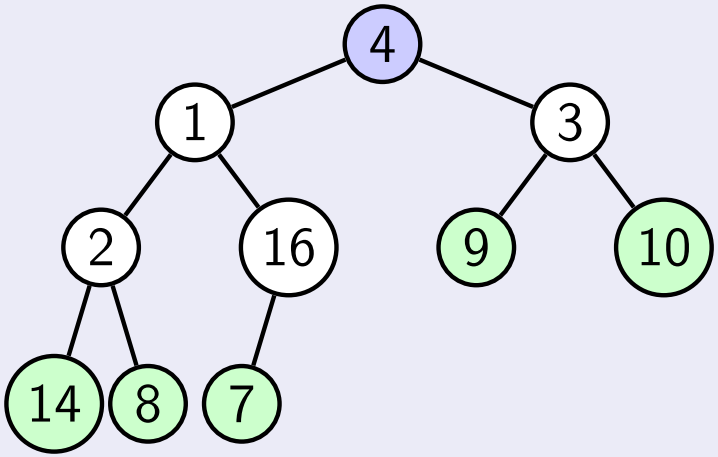
Vấn đề đặt ra là cần biến đổi mảng $A[1 \cdots n]$ thành $max - heap()$, ví các phần tử của mảng con $A[(\lceil n/2 \rceil + 1) \cdots n]$ là các lá, do đó để tạo đống ta chỉ cần áp dụng Max-Heapify() đối với các phần tử từ 1 đến $\lceil n/2 \rceil$.

1. **Build-Max-Heap**(A)
2. n $\leftarrow$ length[A]
3. **for** i $\leftarrow$ $\lfloor n/2 \rfloor$ **downto** 1 **do**
4. Max-Heapify(A,i,n)
5. **endfor**
6. **End**

Sắp xếp vun đống

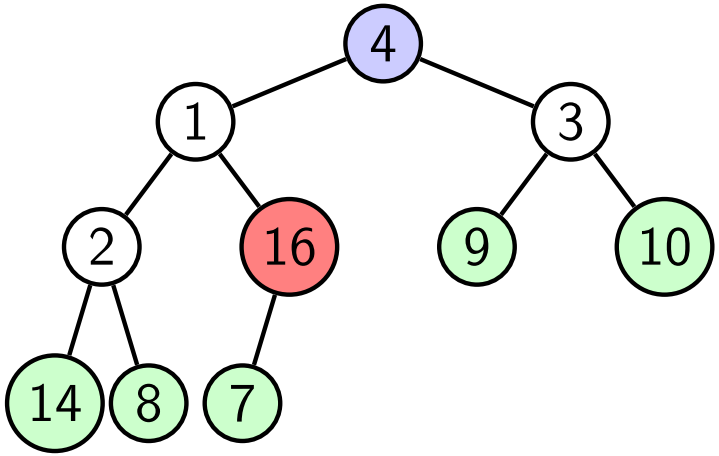
Xây dựng đống (tiếp)

Minh họa dãy ban đầu như sau : (4,1,3,2,16,9,10,14,8,7)



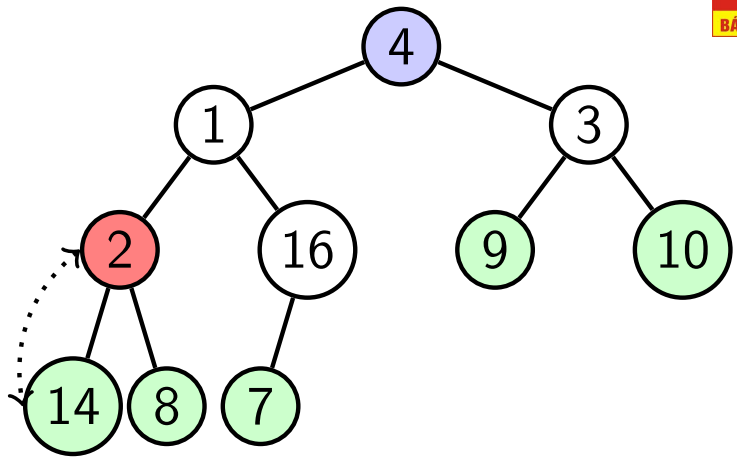
4	1	3	2	16	9	10	14	8	7

Sắp xếp vun đống

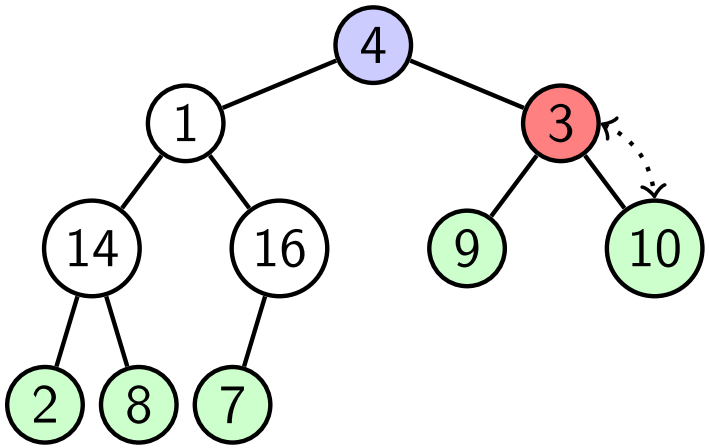


4	1	3	2	16	9	10	14	8	7

$\Rightarrow$

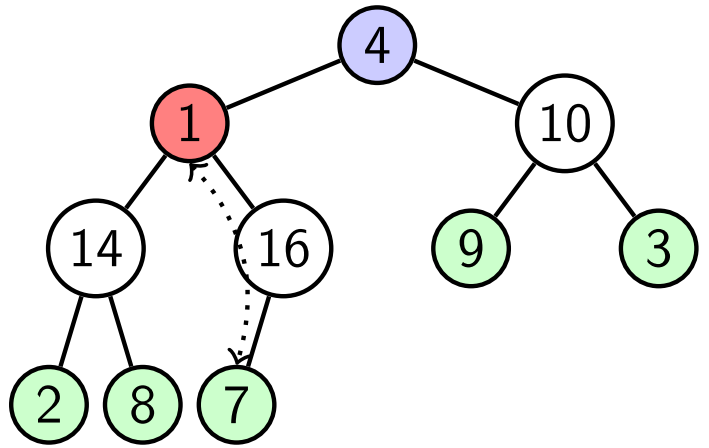


4	1	3	2	16	9	10	14	8	7



$\Rightarrow$

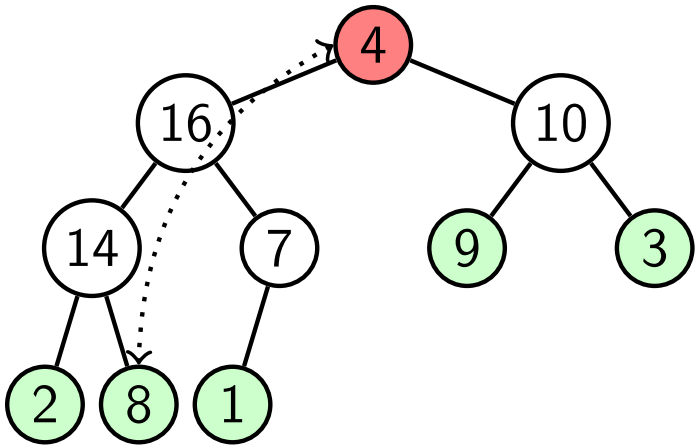
4	1	3	14	16	9	10	2	8	7

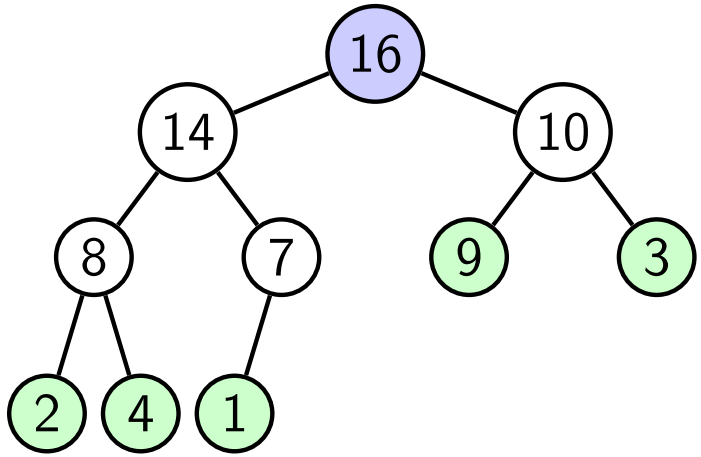


$\Rightarrow$

4	1	10	14	16	9	3	2	8	7

Sắp xếp vun đống





$\Rightarrow$

4	16	10	14	7	9	3	2	8	1

$\Rightarrow$

16	14	10	8	7	9	3	2	4	1

Cuối cùng ta có được đống sau tất cả 6 bước duyệt từ nút $i = 5$ đến $i = 1$

Sắp xếp vun đống

Thời gian của xây dựng đống - Build-Max-Heap

Trong khi Heapify() đòi hỏi thời gian $O(h)$ là chi phí của của Heapify ở nút i là tỉ lệ với chiều cao của nút i trong cây. Thời gian tính của build-max-heap : $T(n) = O(n)$

Sắp xếp vun đống

Giải thuật sắp xếp vun đống

Sử dụng đống ta có thể phát triển thuật toán sắp xếp mảng. Sơ đồ của thuật toán được trình bầy như sau :

- Tạo đống có phần tử gốc có giá trị lớn nhất từ mảng đã cho build-max-heap
- Đổi chỗ gốc (phần tử lớn nhất) với phần tử cuối cung trong mảng
- Loại bỏ nút cuối cung bằng cách giảm kích thước của đống đi một
- Thực hiện vun lại đống với gốc mới
- Lặp lại quá trình đến khi đống chỉ còn một nút

Sắp xếp vun đống

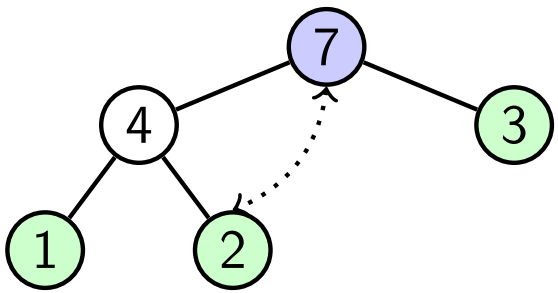
Giải thuật sắp xếp vun đống (tiếp)

Sau đây là mã giả của giải thuật vung đống

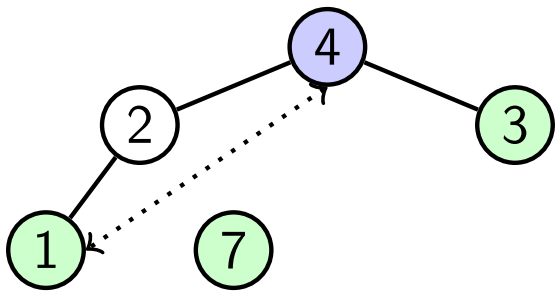
1. **HeapSort**(A)
2. Build-Max-Heap(A)
3. **for** i $\leftarrow$ length[A] **downto** 2 **do**
4. swap(A[1],A[i])
5. Max-Heapify(A,1,i-1)
6. **endfor**

Thời gian tính của dòng 2 là $O(n)$ trong khi thời gian tính của dòng 4 và 5 là $O(\log n)$ và vòng lặp 3 lặp $n-1$ lần. Vậy thời gian tính của HeapSort() là $O(n \log n)$

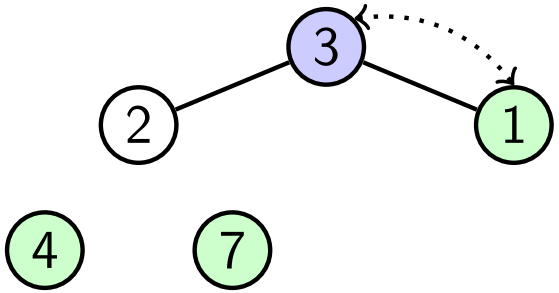
Sắp xếp vun đống



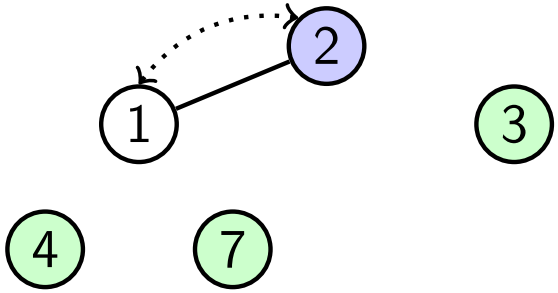
7	4	3	1	2



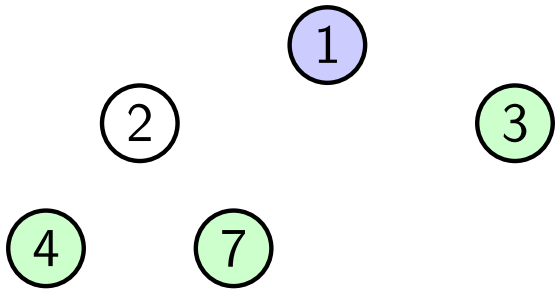
4	2	3	1	7



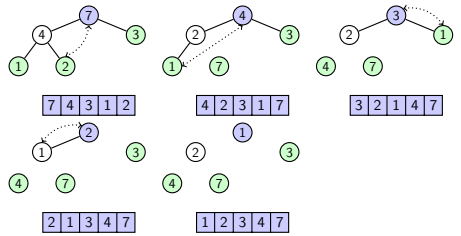
3	2	1	4	7



2	1	3	4	7



1	2	3	4	7



Xét màng A = [7,4,3,1,2] ban đầu không được sắp xếp. Thứ tự các hình là theo chiều từ trái sang phải, trên xuống dưới. Chú ý, ta luôn chạy Max-Heapify() mỗi vòng lặp.

Hàng đợi có ưu tiên

Hàng đợi có ưu tiên - priority queue

Cho tập S thường xuyên biến động, mỗi phần tử x được gán với một giá trị gọi là khóa (hay độ ưu tiên). Cần một cấu trúc dữ liệu hỗ trợ hiệu quả các thao tác chính như sau :

- Insert(S,x) bổ sung phần tử x vào S.
- Max(S) trả lại phần tử lớn nhất.
- Extract-Max(S) loại bỏ và trả lại phần tử lớn nhất.
- Increase-Key(S,x,k) tăng khóa của x thành k.

Cấu trúc dữ liệu đáp ứng các yêu cầu trên được gọi là hàng đợi có ưu tiên. Hàng đợi có ưu tiên có thể sử dụng cấu trúc dữ liệu đống để cất giữ các khóa.

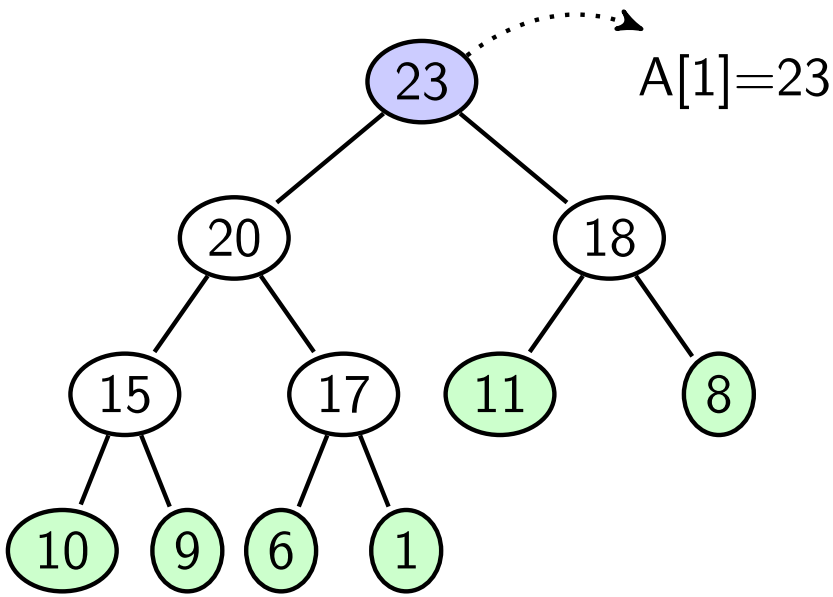
Hàng đợi có ưu tiên

Các phép toán đối với hàng đợi có ưu tiên khi dùng đống - Max

Chức năng : trả lại phần tử lớn nhất của đống

1. **Heap-Max**(A)
2. **return** A[1]

Thời gian tính : $O(1)$



Hàng đợi có ưu tiên

Các phép toán đối với hàng đợi có ưu tiên khi dùng đống ExtractMax

Chức năng : lấy ra phần tử lớn nhất và khôi phục lại tính chất đống
Giải thuật :

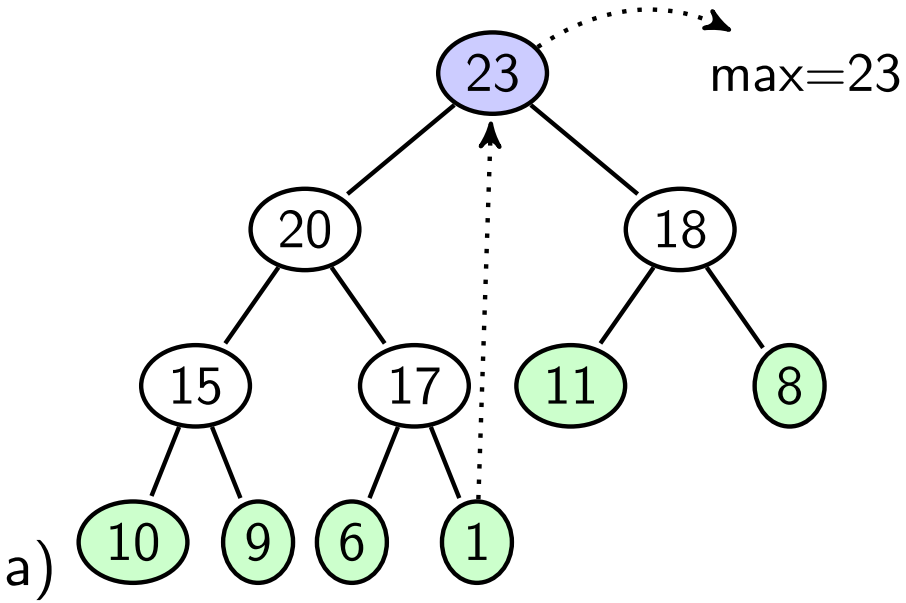
- Hoán đổi giá trị gốc với phần tử cuối cùng
- Giảm kích thước đống đi một
- Gọi Max-Heapify() với gốc mới trên đống kích thước n-1

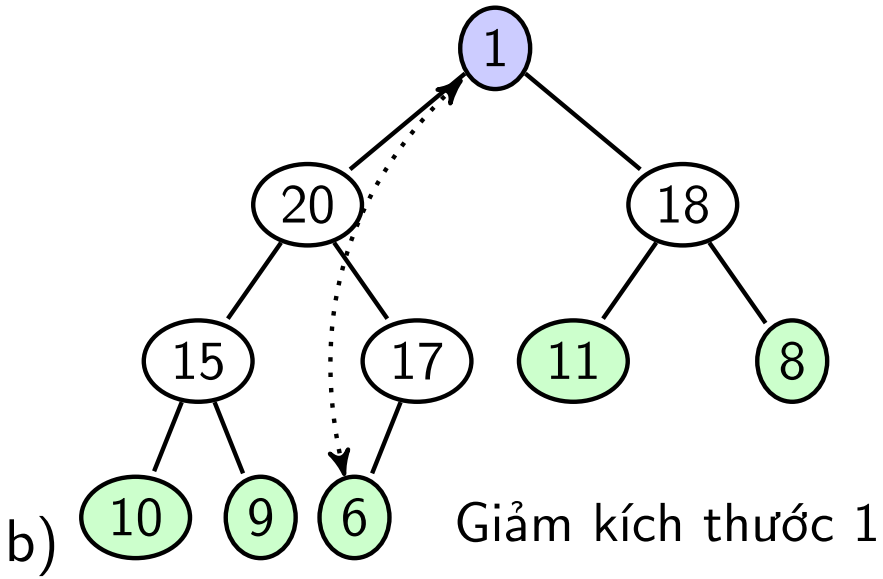
Mã giả :

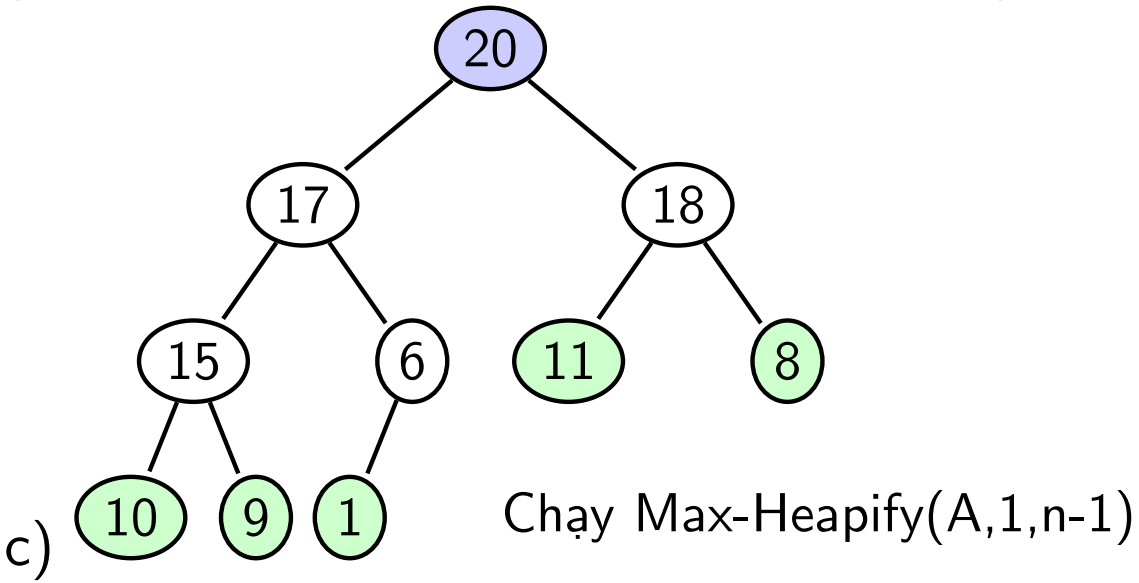
1. **Heap-Extract-Max**(A,n)
2. **if** (n<1) **then** "không có nút trong đống" **return** NULL **endif**
3. max $\leftarrow$ A[1]; A[1] $\leftarrow$ A[n];
4. Max-Heapify(A,1,n-1) // Vun lại đống
5. **return** max

Thời gian tính : $O(\log n)$

Hàng đợi có ưu tiên







Hàng đợi có ưu tiên

Các phép toán đối với hàng đợi có ưu tiên khi dùng đống - IncreaseKey

Chức năng : Tăng giá trị khóa của phần tử *i* trong đống.
Thuật toán :

- Tăng khóa của A[i] thành giá trị mới
- Tính chất của đống - A[parent(i)] $\geq$ A[i]- bị vi phạm : di chuyển theo đường đến gốc để tìm chỗ thích hợp cho khóa mới bị tăng này.

Hàng đợi có ưu tiên

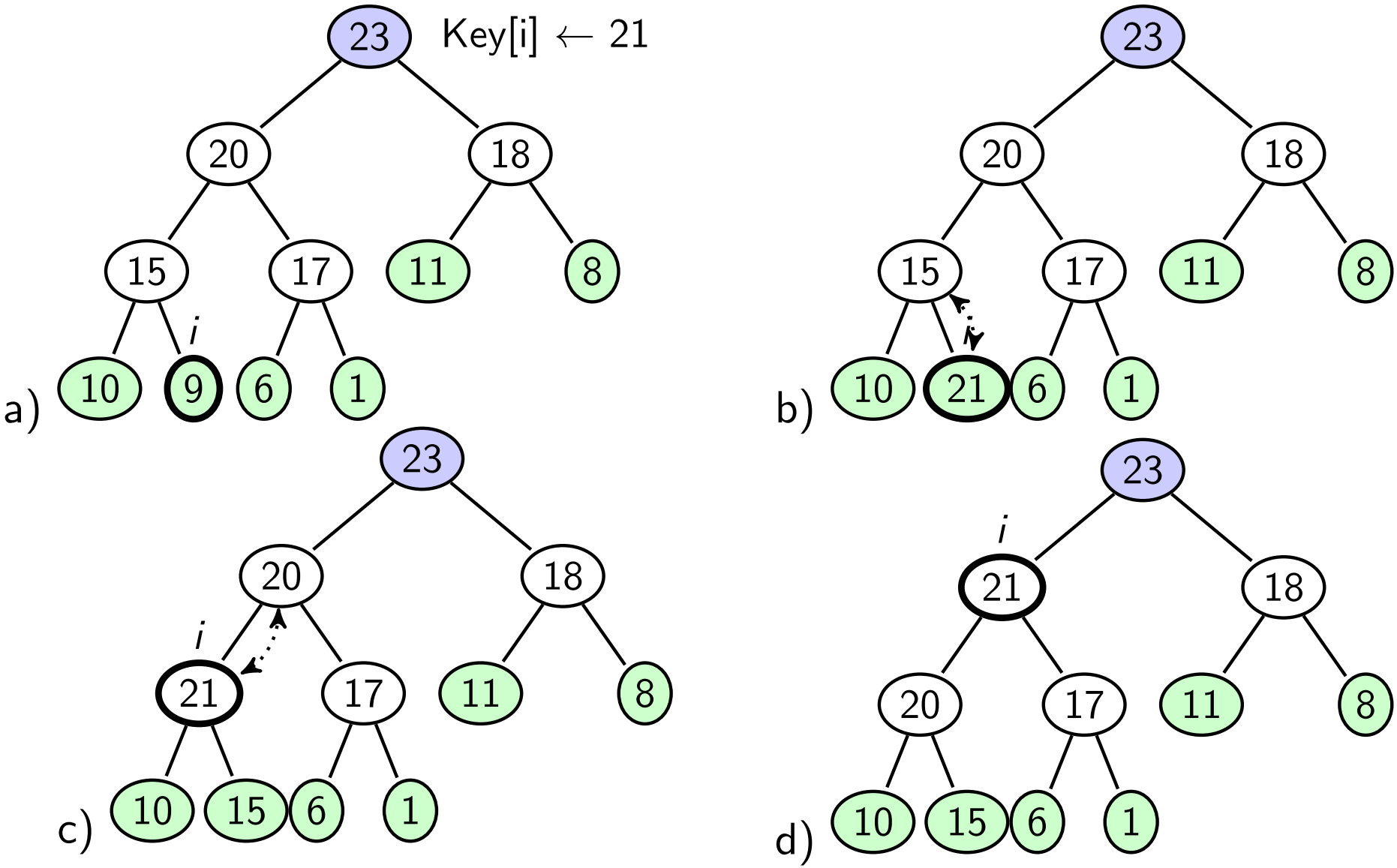
Các phép toán đối với hàng đợi có ưu tiên khi dùng đống - IncreaseKey (tiếp)

Mã giả của phép toán

1. **Heap-Increase-Key**(A,i,key)
2. **if** (key $<$ A[i]) **then** "khóa mới nhỏ hơn khóa hiện tại"; **return** A[i] **endif**
3. A[i] $\leftarrow$ key
4. **while** (i$>$1 **and** A[parent(i)] $<$ A[i]) **do**
5. swap(A[i],A[parent(i)])
6. i $\leftarrow$ parent(i)
7. **endwhile**
8. **return** A[i]

Thời gian tính $O(\log n)$

Hàng đợi có ưu tiên



Hàng đợi có ưu tiên

Các phép toán đối với hàng đợi có ưu tiên khi dùng đống - MaxInsert

Chức năng : Chèn một phần tử mới vào đống
Thuật toán :

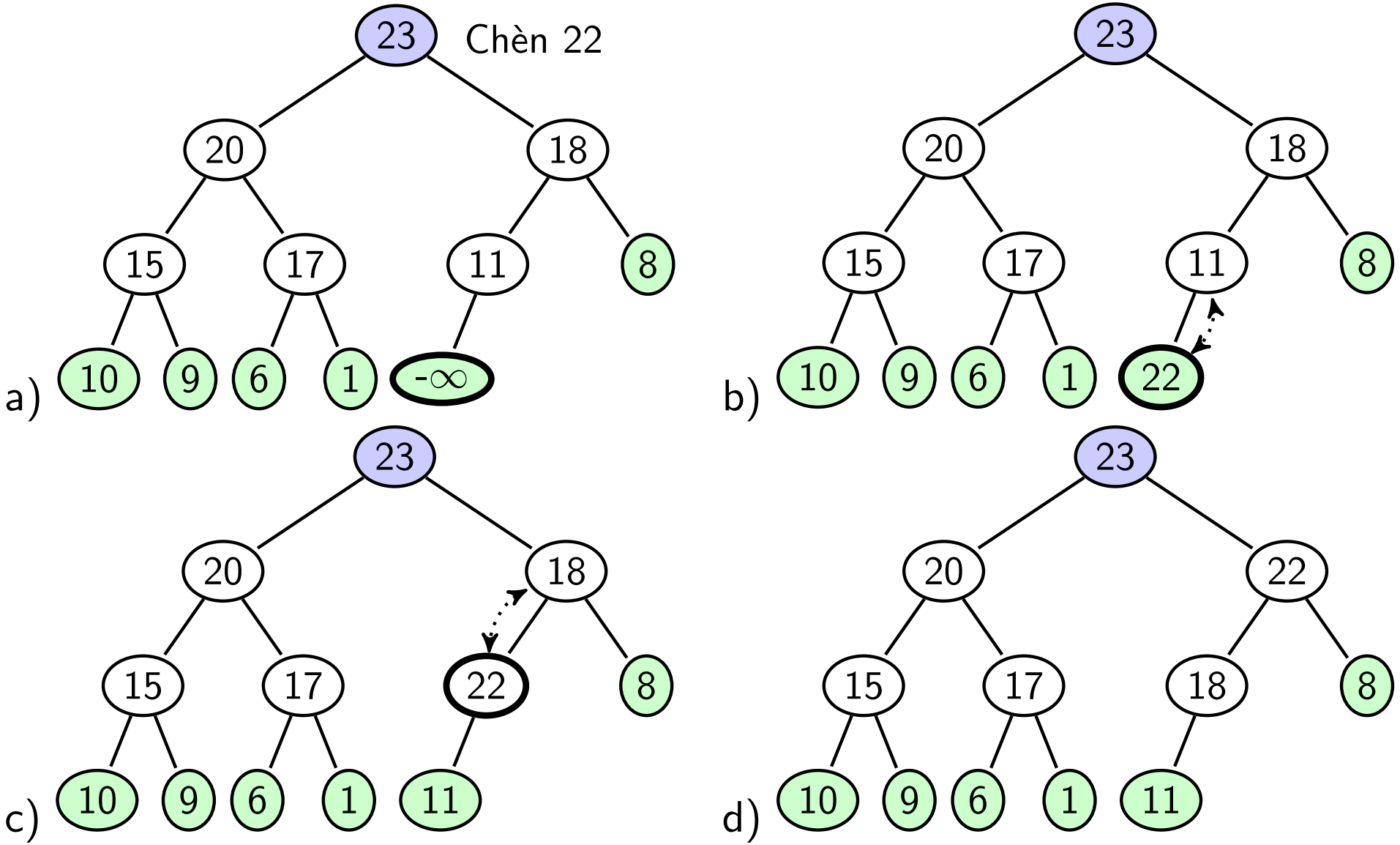
- Mở rộng mảng A với nút mới có khóa chứa giá trị nhỏ nhất có thể $-\infty$
- Gọi Heap-Increase-Key để tăng khóa của nút mới này thành giá trị của phần tử mới và vun lại đống.

Giải thuật có mã giả như sau

1. **Heap-Max-Insert**(A, key, n)
2. heapsize(A) $\leftarrow$ n+1
3. A[n+1] $\leftarrow -\infty$
4. Heap-Increase-Key(A,n+1,key)

Thời gian tính : $O(\log n)$

Hàng đợi có ưu tiên



Tổng kết

Ta có bảng tổng hợp phép toán và thời gian tính của đống như sau

Phép toán	Thời gian tính
Max-Heapify()	$O(\log n)$
Build-Max-Heap()	$O(n)$
Heap-Sort()	$O(n \log n)$
Max-Heap-Insert()	$O(\log n)$
Heap-Extract-Max()	$O(\log n)$
Heap-Increase-Key()	$O(\log n)$
Heap-Maximum()	$O(1)$

Tổng kết (tiếp)

Nhắc lại là ta cũng có thể tạo hàng đợi có ưu tiên bằng danh sách móc nối đơn, sau đây là bảng so sanh thời gian tính của các phép toán sử dụng hai cấu trúc khác nhau

Phép toán	Đống	Móc nối đơn
Max-Heap-Insert()	$O(\log n)$	$O(n)$
Heap-Extract-Max()	$O(\log n)$	$O(1)$
Heap-Increase-Key()	$O(\log n)$	$O(n)$
Heap-Maximum()	$O(1)$	$O(1)$

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Cận dưới cho bài toán sắp xếp

Mô hình sắp xếp

Giả sử rằng, phép toán cơ bản mà ta được sử dụng là *so sánh hai số* ta có thể giảm bớt không giam tìm kiếm đi một nửa sau mỗi lần so sánh hai số. Giả sử là có N phần tử cần sắp xếp và không có hai phần tử trùng nhau. Đối với N phần tử :

- N khả năng chọn vị trí thứ nhất, (N-1) khả năng chọn vị trí thứ hai $\cdots$ 1 khả năng chọn vị trí cuối cùng.
- Vậy có tất cả N(N-1)...(2)(1) = N! thứ tự có thể.

Hoán vị các phần tử khi sắp xếp

Cho dãy có 3 phần tử không trùng nhau : (a, b, c) như vậy khi liệt kê tất cả các hoán vị - trong đó một hoán vị thỏa mãn yêu cầu sắp xếp - thi ta có 6 trường hợp sau

$$(a < b < c)(a < c < b)(b < a < c)(b < c < a)(c < a < b)(c < b < a)$$

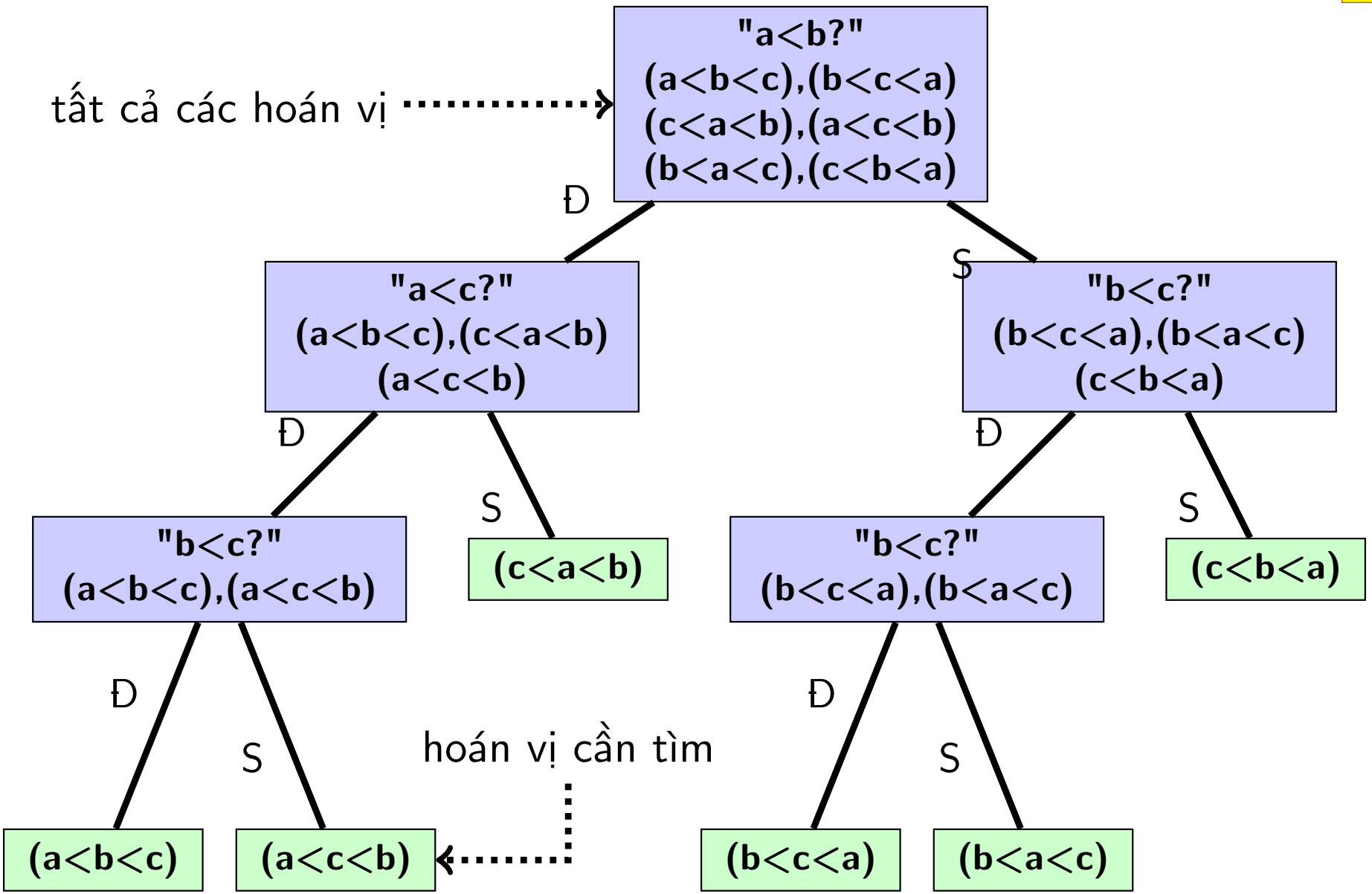
Cận dưới cho bài toán sắp xếp

Mô hình sắp xếp (tiếp)

Cây quyết định : Cây quyết định là cây nhị phân thỏa mãn

- Mỗi nút tương ứng với một phép so sánh "a<b"
  - Cũng có thể coi tương ứng với một không gian con
- Mỗi cạnh tương ứng rẽ nhánh theo câu trả lời (Đúng hay Sai)
- Mỗi lá tương ứng trình tự sắp xếp
  - Cây quyết định sẽ phải có N! lá nếu có N phần tử phân biệt

Cận dưới cho bài toán sắp xếp



Cận dưới cho bài toán sắp xếp

Mô hình sắp xếp (tiếp)

Cây quyết định và sắp xếp

- Mỗi thuật toán sắp xếp đều có thể mô tả bởi một cây quyết định
  - Tìm lá nhờ di chuyển theo cây (tức là chỉ thực hiện phép so sánh)
  - Mỗi quyết định sẽ giảm không gian tìm kiếm đi một nửa
- Thời gian tính trong tình huống tồi nhất $\geq$ số phép so sánh nhiều nhất phải thực hiện
  - Số phép so sánh nhiều nhất cần thực hiện chính là độ dài đường đi dài nhất trên cậy quyết định tức là độ cao của cây

Mệnh đề sau được chứng minh $\log(N!) = \Omega(N \log N)$

Cận dưới cho bài toán sắp xếp

Cận dưới cho độ phức tạp của bài toán sắp xếp

- Thời gian tính của mọi thuật toán sắp xếp chỉ dựa vào phép so sánh là $\Omega(N \log N)$ (theo mệnh đề slice trước)
- Do để giải bài toán sắp xếp, ta có thể sử dụng thuật toán sắp xếp vung đống với thời gian $O(N \log N)$, nên cận trên cho bài toán sắp xếp là $O(N \log N)$
- Suy ra, cận dưới cho độ phức tạp tính toán của bài toán sắp xếp chỉ dựa trên phép so sánh là $\Theta(N \log N)$

Câu hỏi đặt ra

Vậy mọi thuật toán chỉ sử dụng phép so sánh có thời gian tính $\Omega(N \log N)$, nghĩa là giải thuật HeapSort, MergeSort nhanh nhất trong các giải thuật dạng này. Ta có thể *phát triển thuật toán tốt hơn* không nếu không hạn chế là chỉ sử dụng phép so sánh ?

1 Bài toán sắp xếp

2 Ba thuật toán sắp xếp cơ bản

3 Sắp xếp trộn

4 Sắp xếp nhanh

5 Sắp xếp vun đống

6 Cận dưới cho bài toán sắp xếp

7 Các phương pháp sắp xếp đặc biệt

8 Tổng kết

Các phương pháp sắp xếp đặc biệt

Các phương pháp sắp xếp sau

- Sắp xếp đếm (counting sort)
- Sắp xếp theo cơ số (radix sort)
- Sắp xếp phân cụm (bunket sort)

Các phương pháp sắp xếp đặc biệt

Mở đầu

Ý tưởng là ta có thể làm tốt hơn chỉ với phép so sánh với thông tin bổ sung từ giả thiết đầu vào

- Thông tin bổ sung/Giả thiết
  - Các số nguyên nằm trong khoảng [0 ... k] trong đó k = O(n).
  - Các số thực phân bố dều trong khoảng [0,1)

Ta sẽ trình bầy ba thuật toán có thời gian sắp xếp tuyến tính :

- Sắp xếp đếm (counting sort)
- Sắp xếp theo cơ số (radix sort)
- Sắp xếp phân cụm (bunket sort)

theo thông tin bổ sung thì các số nguyên đc sắp xếp là số nguyên không âm.

Các phương pháp sắp xếp đặc biệt

Sắp xếp đếm (Counting sort)

Theo giả thuyết đầu vào : n số nguyên không âm cần sắp xếp sẽ ở trong khoảng [0 ... k] trong đó k là số nguyên và $k = O(n)$

Ý tưởng với mỗi phần tử x của dãy đầu vào ta xác định hạng (rank) của nó như là số lượng phần tử nhỏ hơn x. Sau khi đã biết hạng r của x, ta có thể xếp nó vào vị trí $r + 1$.

Lặp khi có một loạt các phần tử có cùng giá trị, ta sắp xếp chúng theo thứ tự xuất hiện trong dãy ban đầu (để có được tính ổn định của sắp xếp)

Các phương pháp sắp xếp đặc biệt

Mã nguồn C

```
void CountSort(int *a,int *b, int *c,int n,int k)
{// Gia thiet la k<<n
    int i;
    // Dem so lan xuat hien [0..k-1] trong a[0..n-1]
    for(i=0;i<k;i++) c[i] = 0;
    for(i=0;i<n;i++) c[a[i]]++;
    // Tinh hang, hay chi so cuoi cua i=[0..k-1]
       trong b[0..n-1]
    for(i=1;i<k;i++) c[i] += c[i-1];
    // Sap xep
    for(i=n-1;i>=0;i--){
      b[c[a[i]]-1] = a[i];
      c[a[i]] -= 1;
    }
}
```

Các phương pháp sắp xếp đặc biệt

Sắp xếp đếm (tiếp)

Phân tích độ phức tạp của sắp xếp đếm

- Vòng lặp for đếm b[i] - số phần tử có giá trị i, đòi hỏi thời gian $\Theta(n+k)$
- Vòng lặp for tính hạng đòi hỏi thời gian $\Theta(n)$
- Vòng lặp for thực hiện sắp xếp đòi hỏi thời gian $\Theta(k)$

Tổng cộng thời gian tính giải thuật là $\Theta(n+k)$

Chú ý giả thiết $k = \Theta(n)$ nên thời gian tính của thuật toán là $\Theta(n+k)$ vậy là trong trường hợp này nó là một trong những thuật toán tốt nhất. Điều đó không thể xảy ra nếu giả thiết $k = \Theta(k)$ là không thực hiện được. Thuật toán sec làm việc rất tồi khi $k >> n$. Sắp xếp đếm không có tính tại chỗ.

Các phương pháp sắp xếp đặc biệt

Sắp xếp theo cơ số

Giả thiết đầu vào gồm *n số nguyên*, mỗi số có *d* chữ số

Ý tưởng của thuật toán Do thứ tự sắp xếp các số cần tìm cũng chính là thứ tự từ diển của các xâu tương ứng với chúng, nên để tiến hành sắp xếp ta sẽ tiến hành *d* bước sau :

- Bước 1 : Sắp xếp các số theo chữ số 1
- Bước 2 : Sắp xếp các số theo chữ số 2
- ....
- Bước d : Sắp xếp các số theo chữ số *d*

Giả sử các số trong hệ đếm cơ số *k*, khi đó mỗi chữ số chỉ có *k* giá trị nên ở mỗi bước ta có thể sử dụng sắp xếp theo cơ số với thời gian tính $O(n+k)$

Các phương pháp sắp xếp đặc biệt

Sắp xếp theo cơ số (tiếp)

Cho mảng A chứa các số có d chữ số
Procedure Radix-Sort(A,d)

1. **for** i $\leftarrow$ 1 **to** d **do**
2. sử dụng sắp xếp ổn định để sắp xếp theo chữ số thứ i
3. **endfor**

End

Các phương pháp sắp xếp đặc biệt

Sắp xếp theo cơ số (tiếp)

Phân tích độ phức tạp :

- Thời gian tính : nếu bước 2 sử dụng *sắp xếp đếm* thì thời gian tính của một lần lặp là $\Theta(n+k)$ do đó thời gian tính của thuật toán Radix Sort là $T(n) = \Theta(d(n+k))$

Các phương pháp sắp xếp đặc biệt

Sắp xếp theo cơ số (tiếp)

Ví dụ : Xét dãy số d = 2, k =10

$$23, 45, 7, 56, 20, 19, 88, 77, 61, 13, 52, 39, 80, 2, 99$$

Việc sắp xếp bao gồm nhiều bước, bắt đầu từ chữ số trái nhất, tiếp đến là chữ số hàng chục, hàng trăm, v.v...

- Bước 1 (i=1) : 20, 80, 61, 52, 2, 23, 13, , 45, 56, 7, 77, 88, 19, 39, 99
- Bước 2 (i=2) : 2, 7, 13, 19, 20, 23, 39, 45, 52, 56, 61, 77, 80, 88, 99

Các phương pháp sắp xếp đặc biệt

Sắp xếp phân cụm (Bucket Sort)

Giả thiết : Đầu vào gồm *n số thực* có phần bố đều trong khoảng [0..1) (các số có xác suất xuất hiện như nhau)

Ý tưởng của thuật toán

- Chia đoạn [0..1) ra làm n cụm (bunkets) như vậy là

$$0, 1/n, 2/n, \cdots, (n-1)/n$$

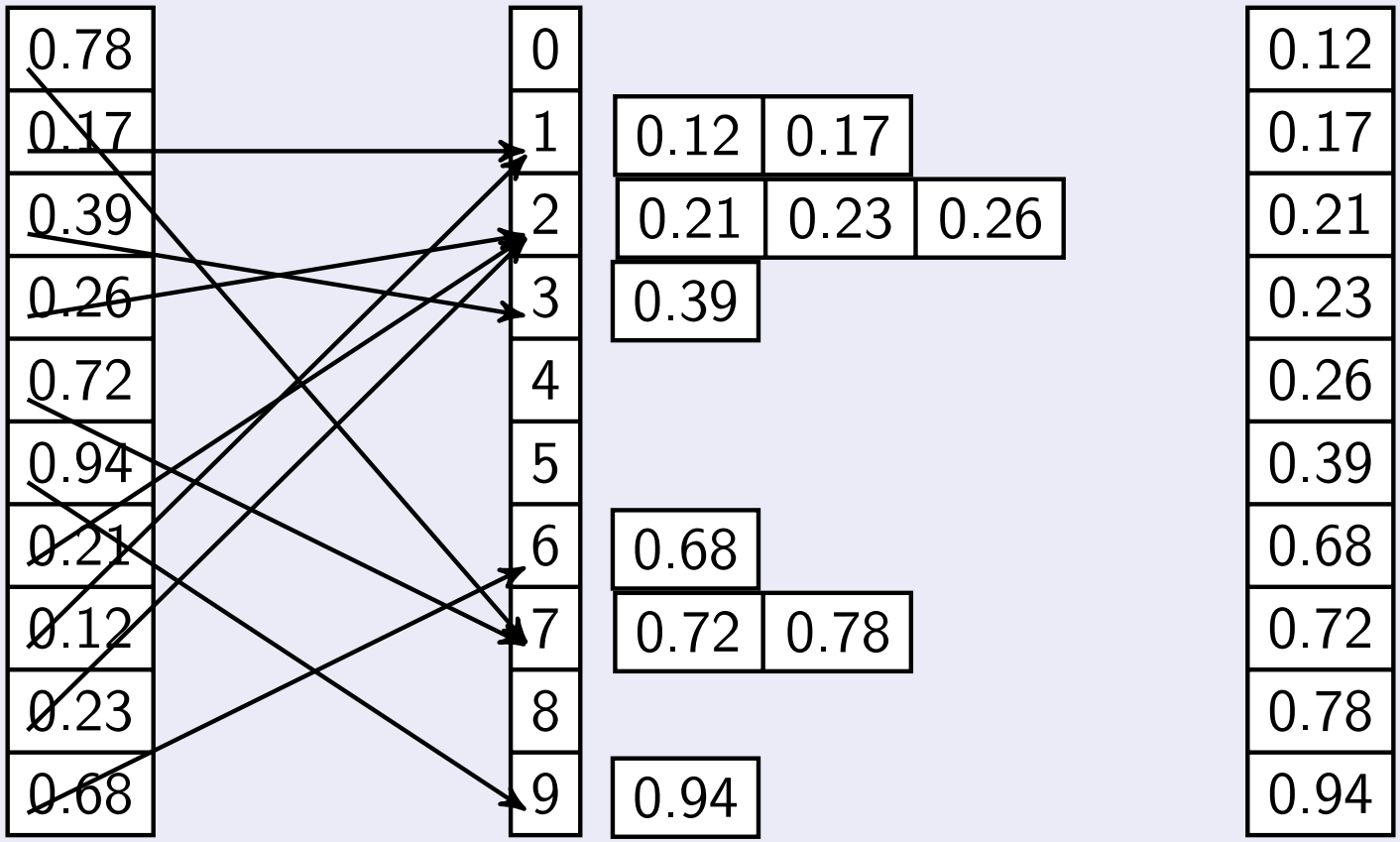
- Đưa mỗi phần tử a_j vào đúng cụm của nó $i/n \leq a_j < (i+1)/n$
- Do các số là phân bố đều nên không có quá nhiều số rơi vào cũng một cụm.
- Nếu ta chèn chúng vào cụm (sử dụng sắp xếp chèn) thì các cụm và các phần tử trong chúng đều được sắp xếp theo đúng thứ tự.

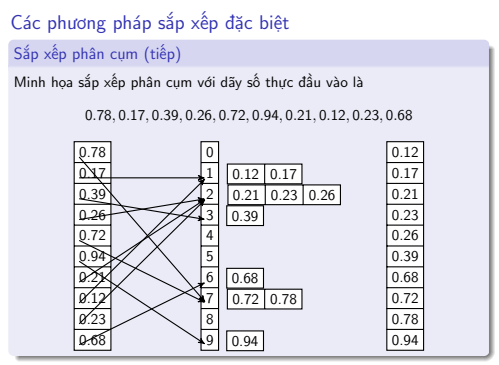
Các phương pháp sắp xếp đặc biệt

Sắp xếp phân cụm (tiếp)

Minh họa sắp xếp phân cụm với dãy số thực đầu vào là

$$0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68$$





Ba cọc số gồm : Cột bên trái là danh sách A các số ban đầu, tính thứ tự trên xuống dưới. Cột giữa là các cụm B tương ứng với mỗi cụm là danh sách các phần tử đã được sắp xếp. Cột bên phải là dãy số thực được sắp xếp.

Các phương pháp sắp xếp đặc biệt

Sắp xếp phân cụm (tiếp)

BucketSort(A)
/* A[0..n-1] là mảng đầu vào, B[0]...B[n-1] là danh sách các cụm */

1. $n \leftarrow$ length(A)
2. **for** i $\leftarrow$ 0 **to** n-1 **do**
3. Bổ sung A[i] vào danh sách B[$\lfloor$ n*A[i] $\rfloor$]
4. **endfor**
5. **for** i $\leftarrow$ 0 **to** n-1 **do**
6. InsertionSort(B[i])
7. **endfor**
8. Nối các danh sách B[0]...B[n-1] theo thứ tự sau khi sắp xếp chèn tại mỗi cụm

End

Các phương pháp sắp xếp đặc biệt

Sắp xếp phân cụm (tiếp)

Phân tích thời gian tính của BucketSort

- Tất cả các dòng trong giải thuật, ngoại trừ dòng 6, đòi hỏi thời gian tính toán là $O(n)$ trong tình huống tồi nhất.
- Trong *tình huống tồi nhất*, O(n) số được đưa vào cùng một cụm, do đó thuật toán có thời gian tính là $O(n^2)$ trong tình huống tồi nhất.
- Trong *tình huống trung bình*, chỉ có một lượng hằng số phần tử của dãy cần sắp xếp rơi vào trong mỗi cụm và vì thế thuật toán có thời gian tính trung bình là O(n)

Tổng kết các phương pháp sắp xếp

Bảng tổng kết độ phức tạp tính toán của các giải thuật sắp xếp đã học

Phương pháp sắp xếp	Trung bình	Tồi nhất	Tại chỗ	Ổn định
Nổi bọt	-	$O(n^2)$	Có	Có
Lựa chọn	$O(n^2)$	$O(n^2)$	Có	Không
Chèn	$O(n+d)$	$O(n^2)$	Có	Có
Trộn	$O(n \log n)$	$O(n \log n)$	Không	Có
Vun đống	$O(n \log n)$	$O(n \log n)$	Có	Không
Nhanh	$O(n \log n)$	$O(n^2)$	Không	Không