

Chương 5 : Các thuật toán sắp xếp

Trịnh Anh Phúc ¹

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

Ngày 19 tháng 9 năm 2020

Giới thiệu



- 1 Bài toán sắp xếp
- 2 Sắp xếp chèn
- 3 Sắp xếp trộn
- 4 Sắp xếp nhanh
- 5 Sắp xếp vun đống
- 6 Tổng kết

Mô tả giải thuật của bài toán sắp xếp

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

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

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

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;  
}
```

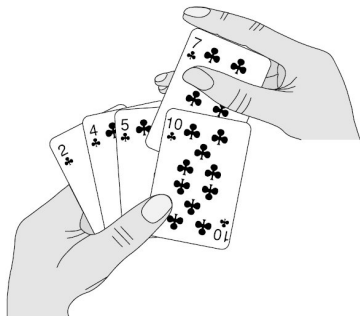
- So sánh (compare) : hàm `compare(a,b)` trả lại `true` nếu `a` ở vị trí trước `b` theo thứ tự cần sắp xếp và `false` nếu trái lại.

- 1 Bài toán sắp xếp
- 2 Sắp xếp chèn**
- 3 Sắp xếp trộn
- 4 Sắp xếp nhanh
- 5 Sắp xếp vun đống
- 6 Tổng kết

Sắp xếp chèn

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.



Sắp xếp chèn (tiếp)

Thuật toán

- Tại bước thứ $k = 1, 2, \dots, n$ 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)

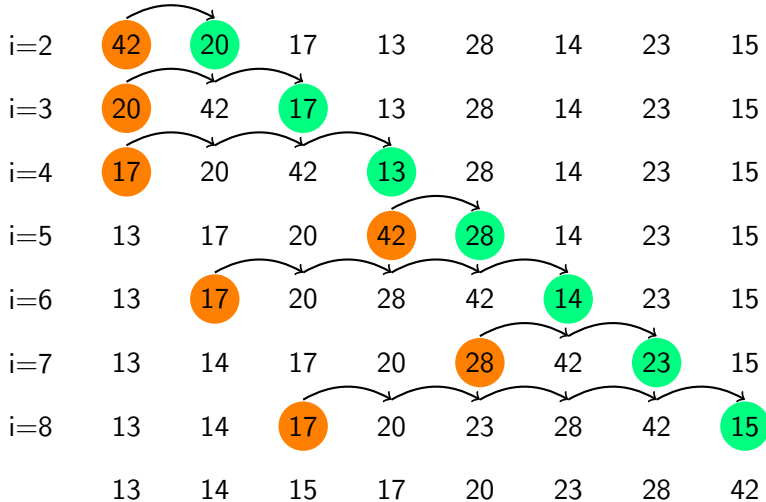
- ① **for** $i \leftarrow 2$ **to** n **do**
- ② $\text{last} \leftarrow A[i]$
- ③ $j \leftarrow i$
- ④ **while** $(j > 1 \text{ and } A[j-1] > \text{last})$ **do**
- ⑤ $A[j] \leftarrow A[j-1]$
- ⑥ $j \leftarrow j-1$
- ⑦ **endwhile**
- ⑧ $A[j] \leftarrow \text{last}$
- ⑨ **endfor**

End

Sắp xếp chèn

Minh họa với dãy không được sắp xếp gồm 8 phần tử

$$A = \{42, 20, 17, 13, 28, 14, 23, 15\}$$



Cấu trúc dữ liệu và giải thuật

└ Sắp xếp chèn

└ Sắp xếp chèn

Sắp xếp chèn

Mình 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 $\text{last} \leftarrow A[i]$ được tô màu xanh sẽ được gán cuối cùng tại vị trí $A[j]$
 được tô màu cam

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.

- 1 Bài toán sắp xếp
- 2 Sắp xếp chèn
- 3 Sắp xếp trộn**
- 4 Sắp xếp nhanh
- 5 Sắp xếp vun đống
- 6 Tổng kết

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-đế-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 (tiếp)

Mã giả của giải thuật đệ qui sắp xếp trộn

MERGE-SORT(A,first,last)

if first < last **then**

 mid \leftarrow (first+last)/2

 MERGE-SORT(A,first, mid)

 MERGE-SORT(A,mid+1, last)

 MERGE(A,first,mid,last)

endif

End

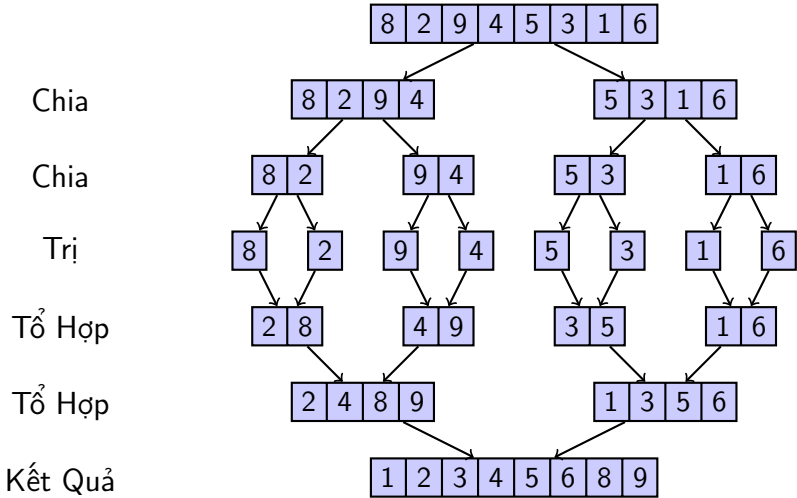
Procedure MERGE(A,first,mid,last)

- 1 Tính $i \leftarrow \text{first}$ và $j \leftarrow \text{mid}+1$ sao cho i trở vào phần tử đầu tiên mảng trái $L[1..n1]$ và j trở vào phần tử đầu tiên mảng bên phải $R[1..n2]$ còn $n1 \leftarrow \text{mid}$ và $n2 \leftarrow \text{last}$. Thêm $L[n1+1] \leftarrow \infty$ và $R[n2+1] \leftarrow \infty$
- 2 $i \leftarrow 1; j \leftarrow 1;$
- 3 **for** $k \leftarrow \text{first}$ **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 (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 (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 Sắp xếp chèn
- 3 Sắp xếp trộn
- 4 Sắp xếp nhanh**
- 5 Sắp xếp vun đống
- 6 Tổng kết

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

- ❶ 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.
- ❷ 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.
- ❸ Trị (Conquer) : lặp lại một cách đệ qui thuật toán đối với hai dãy con L và R .
- ❹ Tổ hợp (Combine) : Dãy được sắp xếp là LpR

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

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 đầu, cuối hoặc giữa.
- Chọn ngẫu nhiên một phần tử

Thao tác phân đoạn (tiếp)

Ta xây dựng hàm $\text{Partition}(a, \text{left}, \text{right})$ như sau :

- **Đầu vào** : Mảng $a[\text{left}..\text{right}]$
- **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 :
 - ▶ $a[\text{jpivot}]$ chứa giá trị chốt p
 - ▶ $a[i] \leq a[\text{jpivot}]$ với mọi $\text{left} \leq i < p$
 - ▶ $a[j] > a[\text{jpivot}]$ với mọi $p < j \leq \text{right}$

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

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

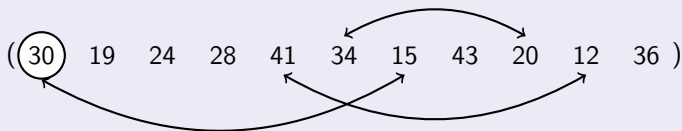
Function partition(a,left,right)

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

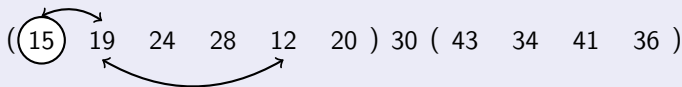
End

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

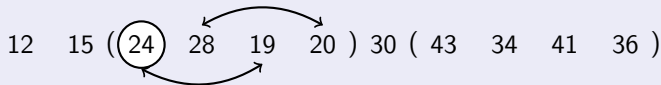
• Bước 1 :



• Bước 2 :



• Bước 3 :



Cấu trúc dữ liệu và giải thuật

- Sắp xếp nhanh
- Thao tác phân đoạn
- Sắp xếp nhanh

▼ 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.

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

- Bước 4 :

12 15 (19) 20) 24 28 30 (43 34 41 36)



- Bước 5 :

12 15 19 20 24 28 30 ((43) 34 41 36)



- Bước 6 :

12 15 19 20 24 28 30 ((36) 34 41) 43



- Bước kết thúc :

12 15 19 20 24 28 30 34 36 41 43

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 .

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
 $T(n) = \Theta(n^2)$

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

$$T(n) = \Theta(n \log n)$$

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 đồng 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.

Phân đoạn cân bằng - balanced partition (tiếp)

Khi đó thời gian tính trung bình sẽ có công thức

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

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 dụng 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 quy ta thu được : $\mathbb{E}(T(n)) = O(n \log n)$

- 1 Bài toán sắp xếp
- 2 Sắp xếp chèn
- 3 Sắp xếp trộn
- 4 Sắp xếp nhanh
- 5 Sắp xếp vun đống**
- 6 Tổng kết

Cấu trúc dữ liệu đống - heap

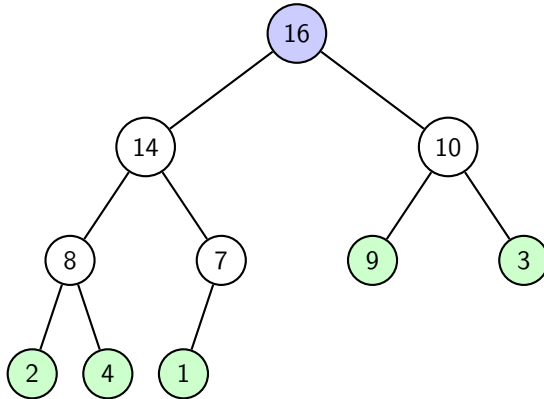
Định nghĩa : Đống (heap) là cây nhị phân *gần hoàn chỉnh* có hai tính chất

- Tính cấu trúc (structural property) : tất cả các mức đều đầy, ngoại trừ mức cuối cùng, mức cuối được điền từ trái sang 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
----	----	----	---	---	---	---	---	---	---

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 $n/2 + 1, \dots, n$ trong mảng A là các lá.

Như vậy các hàm cơ bản được cài đặt như sau

- $\text{parent}(i) = i/2$
- $\text{left-child}(i) = 2i$
- $\text{right-child}(i) = 2i + 1$

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

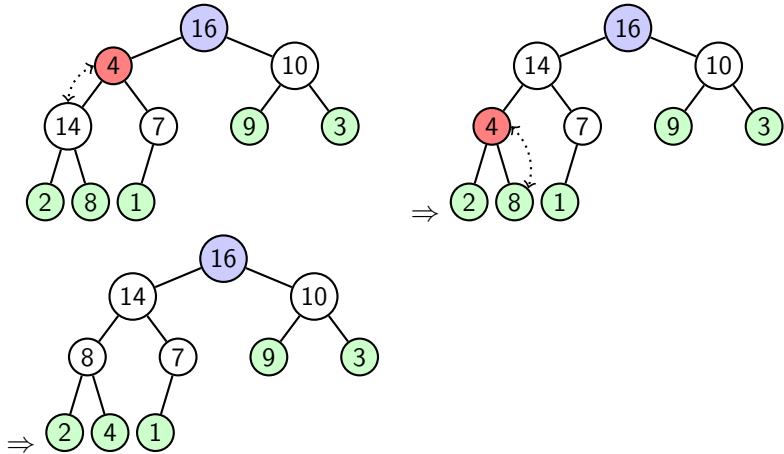
Khôi phục tính chất đống

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 quá 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 trao đổi giá trị giữa hai nút trên cây. Thứ tự hình minh họa là trái qua phải, trên xuống dưới theo hình mũi tên



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

- ❶ **Max-Heapify**(A,i,n)
- ❷ left \rightarrow left-child(i); right \rightarrow right-child(i);
- ❸ **if** (left \leq n **and** A[left] > A[i]) **then** largest \leftarrow left
else largest \leftarrow i **endif**
- ❹ **if** (right \leq n **and** A[right] > A[largest]) **then** largest \leftarrow right
- ❺ **if** (largest **not** i) **then**
 swap(A[i],A[largest]); Max-Heapify(A,largest,n);
endif
- ❻ **End**

trong đó n là số nút của 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 $O(\log n)$

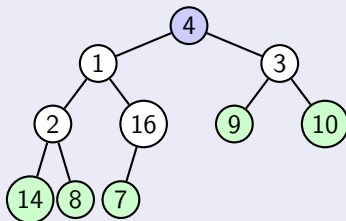
Xây dựng đống

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 $\lceil n/2 \rceil$.

- ❶ **Build-Max-Heap**(A)
- ❷ $n \leftarrow \text{length}[A]$
- ❸ **for** $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1 **do**
- ❹ *Max-Heapify*(A,i,n)
- ❺ **endfor**
- ❻ **End**

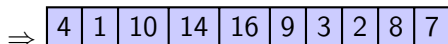
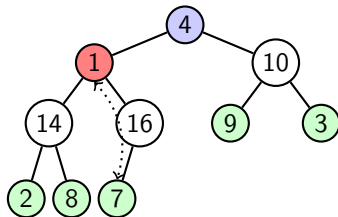
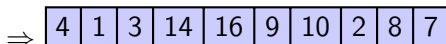
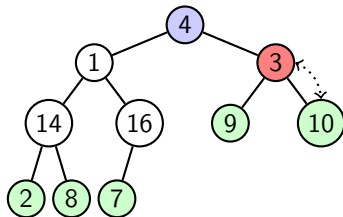
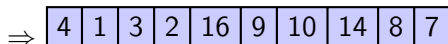
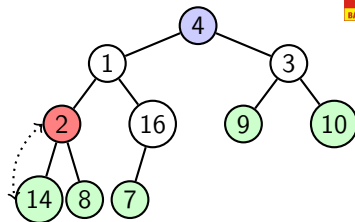
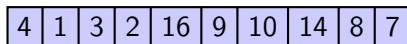
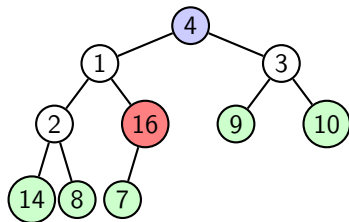
Xây dựng đống (tiếp)

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

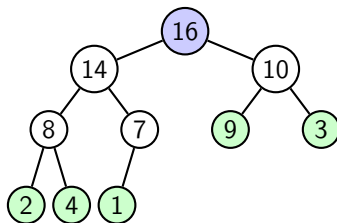
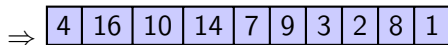
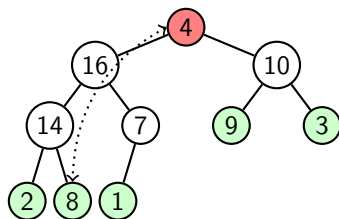


4	1	3	2	16	9	10	14	8	7
---	---	---	---	----	---	----	----	---	---

Sắp xếp vun đống



Sắp xếp vun đống



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

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 : $T(n) = O(n)$

Cấu trúc dữ liệu và giải thuật

- Sắp xếp vun đống
- Cấu trúc dữ liệu đống
 - Sắp xếp vun đống

Gọi chiều cao cây mỗi lần gọi i là h thì chi phí để `Max-Heapify` $O(h)$ Vậy chi phí toàn bộ vòng lặp for là

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left\langle n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h} \right\rangle$$

Theo công thức, $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$ với $x = 1/2$ và k thay chỗ h thì tổng phía trong của công thức trên

$$\sum_{h=0}^{\infty} h\left(\frac{1}{2}\right)^h = \frac{1/2}{(1 - 1/2)^2} = 2$$

Vậy thời gian chạy $T(n) = O(n)$

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 bày 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 cùng trong mảng
- Loại bỏ nút cuối cùng 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

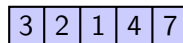
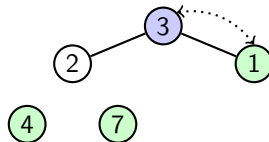
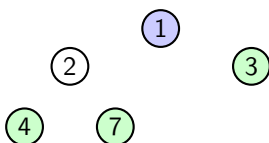
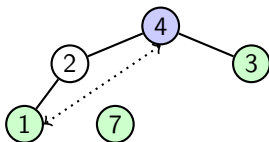
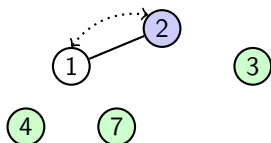
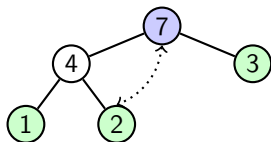
Giải thuật sắp xếp vun đống (tiếp)

Sau đây là mã giả của giải thuật vun đống

- ❶ **HeapSort(A)**
- ❷ **Build-Max-Heap(A)**
- ❸ **for** $i \leftarrow \text{length}[A]$ **downto** 2 **do**
- ❹ $\text{swap}(A[1], A[i])$
- ❺ $\text{Max-Heapify}(A, 1, i-1)$
- ❻ **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 $\text{HeapSort}()$ là $O(n \log n)$

Sắp xếp vun đống

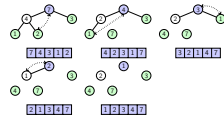


Cấu trúc dữ liệu và giải thuật

└ Sắp xếp vun đống

└ Sắp xếp vun đống

└ Sắp xếp vun đống



Xét mảng $A = [7, 4, 3, 1, 2]$ 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 $\text{Max-Heapify}()$ mỗi vòng lặp.

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

Phương pháp sắp xếp	Trung bình	Tối 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 đồng	$O(n \log n)$	$O(n \log n)$	Có	Không
Nhanh	$O(n \log n)$	$O(n^2)$	Không	Không