

Sắp xếp (Sorting)

Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin

Trường Đại học Thủy Lợi

Nội dung

1. Sắp xếp chọn
2. Sắp xếp nổi bọt
3. Sắp xếp chèn
4. Sắp xếp vun đống
5. Sắp xếp trộn
6. Sắp xếp nhanh

1. Sắp xếp chọn (selection sort)

Sắp xếp chọn

- Dãy A gồm n phần tử a_0, a_1, \dots, a_{n-1} .
- Mỗi bước xét một danh sách con chưa sắp xếp (unsorted sublist - USL).
- Có n-1 bước:
 - Bước 0: $USL_0 = \{a_0, a_1, \dots, a_{n-1}\}$
 - Bước 1: $USL_1 = \{a_1, \dots, a_{n-1}\}$
 - ...
 - Bước n-2: $USL_{n-1} = \{a_{n-2}, a_{n-1}\}$

Sắp xếp chọn (tiếp)

- Mỗi bước:
 - Tìm phần tử nhỏ nhất a_{\min} trong USL.
 - Đổi chỗ a_{\min} và phần tử đầu tiên của USL.
 - Dịch chuyển biên trái của USL sang phải một vị trí.


Ví dụ sắp xếp chọn

- Ban đầu: 64, 25, 12, 22, 11 ($11 \leftrightarrow 64$)
- Sau bước 0: 11, 25, 12, 22, 64 ($12 \leftrightarrow 25$)
- Sau bước 1: 11, 12, 25, 22, 64 ($22 \leftrightarrow 25$)
- Sau bước 2: 11, 12, 22, 25, 64 ($25 \leftrightarrow 25$)
- Sau bước 3: 11, 12, 22, 25, 64

(Danh sách con chưa sắp xếp được gạch chân)

Cài đặt sắp xếp chọn

Ta dùng lớp vector trong thư viện chuẩn của C++



```
void selectionSort(vector<int> & a) {  
    for (int i = 0; i < a.size() - 1; i++) {  
        int vt = i; // Vị trí của min  
        for (int j = i + 1; j < a.size(); j++)  
            if (a[vt] > a[j])  
                vt = j; // Cập nhật vị trí của min  
        if (vt != i) { // Đổi chỗ min và phần tử đầu USL  
            int tg = a[vt];  
            a[vt] = a[i];  
            a[i] = tg;  
        }  
    }  
}
```

Phân tích sắp xếp chọn

- Đếm số phép so sánh $a[v] > a[j]$.
- Vòng for bên trong lặp với j từ $i+1$ đến $n-1$, tức là có $n-1-i$ phép so sánh.
- Vòng for bên ngoài lặp với i từ 0 đến $n-2$.

$$\begin{aligned} t(n) &= \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \cdots + 1 \\ &= \frac{n(n-1)}{2} = O(n^2) \end{aligned}$$

2. Sắp xếp nổi bọt (bubble sort)

Sắp xếp nổi bọt

- Mỗi bước duyệt qua các phần tử a_0, a_1, \dots, a_k .
- Tại mỗi phần tử a_i ($i < k$):
 - So sánh a_i với a_{i+1}
 - Đổi chỗ nếu chúng chưa đúng thứ tự
- Sau mỗi bước, phần tử lớn nhất sẽ được đặt (“nổi bọt”) xuống cuối dãy (a_k là max).

Ví dụ sắp xếp nổi bọt

- Ban đầu: 64, 25, 12, 22, 11 ($k = n-1-0$)
- Sau bước 0: 25, 12, 22, 11, 64 ($k = n-1-1$)
- Sau bước 1: 12, 22, 11, 25, 64 ($k = n-1-2$)
- Sau bước 2: 12, 11, 22, 25, 64 ($k = n-1-3$)
- Sau bước 3: 11, 12, 22, 25, 64

(Danh sách con chưa sắp xếp được gạch chân)

Cài đặt sắp xếp nổi bọt

```
void bubbleSort(vector<int> & a) {  
    for (int i = 0; i < a.size()-1; i++) {  
        // Bước i  
        for (int j = 0; j < a.size()-1-i; j++) {  
            if (a[j] > a[j+1]) {  
                int tg = a[j];  
                a[j] = a[j+1];  
                a[j+1] = tg;  
            }  
        }  
    }  
}
```

Phân tích sắp xếp nổi bọt

- Đếm số phép so sánh $a[j] > a[j+1]$.
- Vòng for bên trong lặp với j từ 0 đến $n-2-i$, tức là có $n-1-i$ phép so sánh.
- Vòng for bên ngoài lặp với i từ 0 đến $n-2$.

$$\begin{aligned} t(n) &= \sum_{i=0}^{n-2} (n-1-i) \\ &= (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} \\ &= O(n^2) \end{aligned}$$

3. Sắp xếp chèn (insertion sort)

Sắp xếp chèn

- Có $n-1$ bước ứng với $p = 1, 2, \dots, n-1$.
- Ở bước p :
(Khi bắt đầu bước p , các vị trí $0, \dots, p-1$ đã được sắp xếp rồi)
 - Xác định vị trí phù hợp trong các vị trí $0, \dots, p-1$ cho phần tử a_p đang nằm ở vị trí p .
 - Chèn a_p vào vị trí đã xác định được, vì vậy các vị trí từ 0 đến p được sắp xếp.

Ví dụ sắp xếp chèn

Original	34	8	64	51	32	21	Positions Moved
After $p = 1$	8	34	64	51	32	21	1
After $p = 2$	8	34	64	51	32	21	0
After $p = 3$	8	34	51	64	32	21	1
After $p = 4$	8	32	34	51	64	21	3
After $p = 5$	8	21	32	34	51	64	4

Cài đặt sắp xếp chèn

```
void insertionSort(vector<int> & a) {  
    int j;  
    for (int p = 1; p < a.size(); p++) {  
        int tmp = a[p]; // Lấy ra phần tử cần chèn  
        for (j = p; j > 0; j--) { // j: vị trí đón nhận  
            if (tmp < a[j-1])  
                a[j] = a[j-1]; // Dịch về phía sau  
            else  
                break;  
        }  
        a[j] = tmp; // Đặt phần tử cần chèn vào đúng chỗ  
    }  
}
```

Phân tích sắp xếp chèn

- Đếm số phép so sánh $tmp < a[j-1]$.
- Trong trường hợp tồi nhất, vòng for bên trong lặp với j từ p giảm dần về 1, tức là có p phép so sánh.
- Vòng for bên ngoài lặp với p từ 1 đến $n-1$.

$$t(n) = \sum_{p=1}^{n-1} p = \frac{n(n-1)}{2} = O(n^2)$$

4. Sắp xếp vun đống (heap sort)

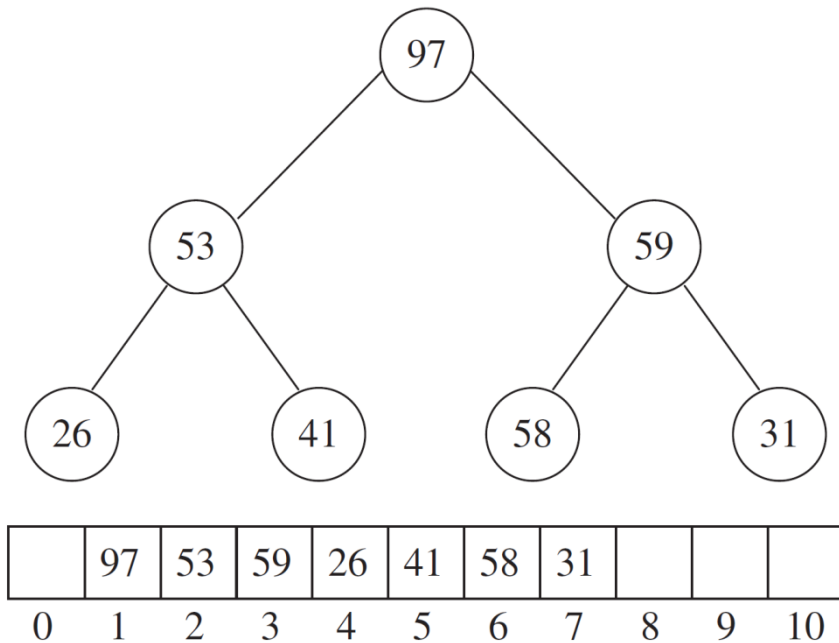
Sắp xếp vun đống

- Các bước thực hiện:
 - Xây dựng đống từ dãy n phần tử đã cho (dùng `buildHeap`) \rightarrow mất thời gian $O(n)$.
 - Thực hiện n lần cặp thao tác `findMin/deleteMin` để lấy ra lần lượt các phần tử từ nhỏ nhất đến lớn nhất \rightarrow mất thời gian $O(n \log n)$.
- Thời gian chạy tổng thể: $O(n \log n)$.
- Nhược điểm: Yêu cầu thêm một vector nữa để lưu trữ các phần tử được rút ra từ đống.

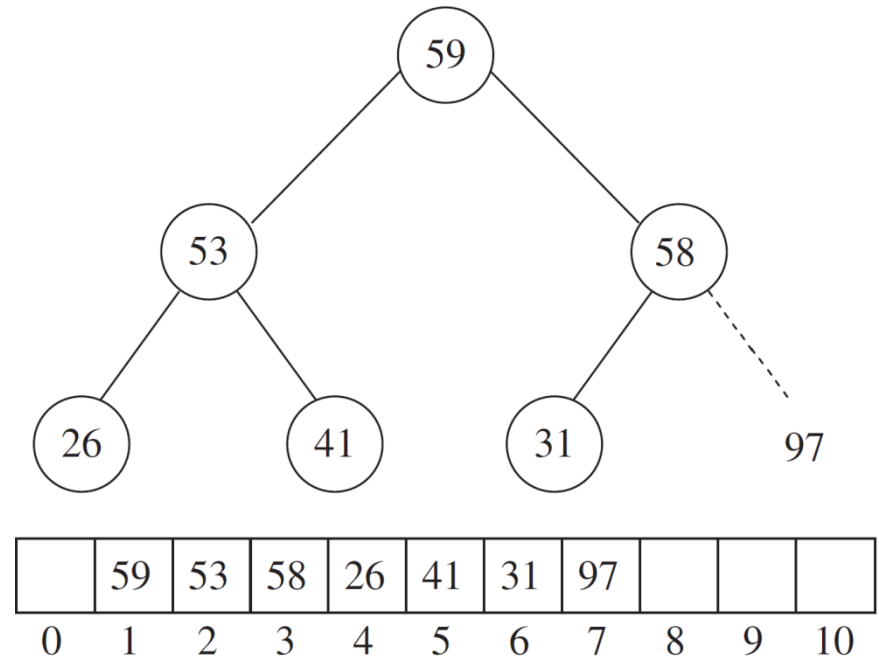
Giải pháp đồng cực đại

- Trên đồng cực đại:
 - Giá trị của nút cha lớn hơn giá trị của các nút con.
 - Phần tử lớn nhất nằm ở nút gốc.
- Khi sắp xếp dùng đồng cực đại:
 - Phần tử được rút ra khỏi đồng được đặt vào ngay sau phần tử cuối cùng của đồng.
 - Chỉ cần một vector cho mục đích lưu trữ.

Ví dụ đồng cực đại



Sau buildHeap



Sau findMax/deleteMax đầu tiên

Cài đặt sắp xếp vun đống

```
// Các phần tử nằm từ vị trí 0 (thay vì 1) trong vector
void heapSort(vector<int> & a) {
    for (int i = a.size()/2 - 1; i >= 0; i--) // buildHeap
        percolateDown(a, i, a.size());

    for (int j = a.size() - 1; j > 0; j--) { // deleteMax
        int max = a[0];
        a[0] = a[j];
        a[j] = max;
        percolateDown(a, 0, j);
    }
}

// Trả về con trái của nút i
int leftChild(int i) {
    return 2 * i + 1;
}
```

Cài đặt sắp xếp vun đống (tiếp)

```
// Thăm thấu xuôi: i là lỗ trống, n là số phần tử đang xét
void percolateDown(vector<int> & a, int i, int n) {
    int tmp = a[i];
    while (leftChild(i) < n) {
        int child = leftChild(i);
        if (child < n - 1 && a[child] < a[child + 1])
            child++; // Cập nhật con lớn hơn
        if (tmp < a[child]) { // Vi phạm tính chất thứ tự đống
            a[i] = a[child];
            i = child;
        }
        else
            break;
    }
    a[i] = tmp;
}
```


5. Sắp xếp trộn (merge sort)

Sắp xếp trộn

- Xét dãy gồm n phần tử.
- Nếu $n = 1$: Dãy đã sắp xếp rồi.
- Nếu $n > 1$:
 - Chia dãy thành hai nửa trái và phải, mỗi nửa có kích thước $n/2$.
 - Sắp xếp trộn đối với mỗi nửa (gọi đệ quy).
 - Trộn hai nửa thành danh sách đầy đủ sao cho danh sách này cũng được sắp xếp.

Thao tác trộn (1)

Đầu vào: Hai dãy A và B đã sắp xếp.

Đầu ra: Dãy C đã sắp xếp, gồm tất cả các phần tử trong A và B.

- Dùng các bộ đếm Actr, Bctr, Cctr để chỉ vị trí hiện hành trong các dãy A, B, C.
- Mỗi bước:
 - So sánh hai phần tử hiện hành trong A và B.
 - Sao chép phần tử nhỏ hơn sang vị trí hiện hành trong C.
 - Tăng các bộ đếm tương ứng.

1	13	24	26
---	----	----	----

↑
Actr

2	15	27	38
---	----	----	----

↑
Bctr

--	--	--	--	--	--	--	--

↑
Cctr

(Sẽ sao chép 1, tăng Actr và Cctr)

Thao tác trộn (2)

1	13	24	26
---	----	----	----

↑
Actr

2	15	27	38
---	----	----	----

↑
Bctr

1							
---	--	--	--	--	--	--	--

↑
Cctr

1	13	24	26
---	----	----	----

↑
Actr

2	15	27	38
---	----	----	----

↑
Bctr

1	2						
---	---	--	--	--	--	--	--

↑
Cctr

1	13	24	26
---	----	----	----

↑
Actr

2	15	27	38
---	----	----	----

↑
Bctr

1	2	13					
---	---	----	--	--	--	--	--

↑
Cctr

1	13	24	26
---	----	----	----

↑
Actr

2	15	27	38
---	----	----	----

↑
Bctr

1	2	13	15				
---	---	----	----	--	--	--	--

↑
Cctr

1	13	24	26
---	----	----	----

↑
Actr

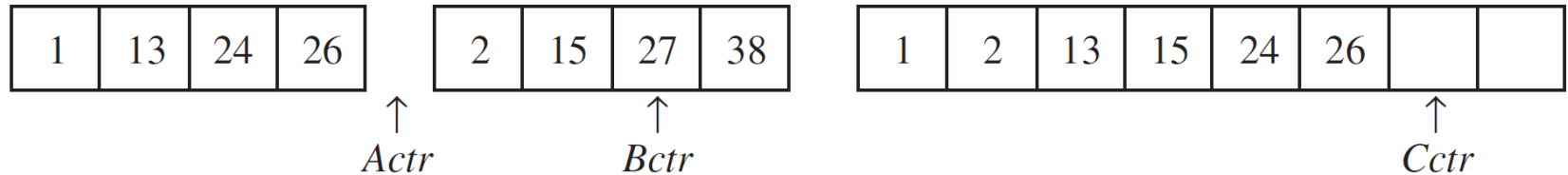
2	15	27	38
---	----	----	----

↑
Bctr

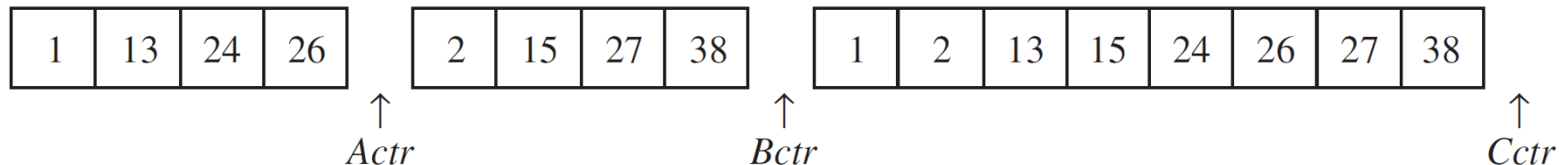
1	2	13	15	24			
---	---	----	----	----	--	--	--

↑
Cctr

Thao tác trộn (3)



A đã hết, sao chép phần còn lại của B sang C:



Phân tích thao tác trộn:

- Có n bước (n là tổng số phần tử của cả A và B).
- Mỗi bước mất thời gian hằng để sao chép một phần tử từ A hoặc B sang C.

⇒ Thời gian chạy của thao tác trộn là $O(n)$.

Cài đặt sắp xếp trộn

```
void mergeSort(vector<int> & a) {  
    vector<int> tmpArray(a.size());  
    mergeSort(a, tmpArray, 0, a.size() - 1);  
}
```

```
// tmpArray là mảng tạm để chứa kết quả trộn.  
// left là vị trí trái cùng của mảng con cần sắp xếp.  
// right là vị trí phải cùng của mảng con cần sắp xếp.  
void mergeSort(vector<int> & a, vector<int> & tmpArray, int  
left, int right) {  
    if (left < right) {  
        int center = (left + right) / 2;  
        mergeSort(a, tmpArray, left, center);  
        mergeSort(a, tmpArray, center + 1, right);  
        merge(a, tmpArray, left, center + 1, right);  
    }  
}
```

Cài đặt thao tác trộn

```
// leftPos là vị trí bắt đầu của nửa trái.
// rightPos là vị trí bắt đầu của nửa phải.
// rightEnd là vị trí cuối cùng của nửa phải.
void merge(vector<int> & a, vector<int> & tmpArray,
int leftPos, int rightPos, int rightEnd) {
    int leftEnd = rightPos - 1; // Vị trí cuối cùng của nửa trái
    int tmpPos = leftPos;        // Vị trí hiện hành trong mảng tạm
    int numElements = rightEnd - leftPos + 1; // Số phần tử của cả 2 nửa
    while (leftPos <= leftEnd && rightPos <= rightEnd)
        if (a[leftPos] <= a[rightPos])
            tmpArray[tmpPos++] = a[leftPos++];
        else
            tmpArray[tmpPos++] = a[rightPos++];
    while (leftPos <= leftEnd) // Sao chép phần còn lại của nửa trái
        tmpArray[tmpPos++] = a[leftPos++];
    while (rightPos <= rightEnd) // Sao chép phần còn lại của nửa phải
        tmpArray[tmpPos++] = a[rightPos++];
    // Sao chép từ mảng tạm về mảng chính
    for (int i = 0; i < numElements; i++, rightEnd--)
        a[rightEnd] = tmpArray[rightEnd];
}
```

Phân tích sắp xếp trộn

- Nếu $n = 1$, không phải làm gì, tức là $t(1) = 1$.
- Nếu $n > 1$, sắp xếp hai nửa mất thời gian $2t(n/2)$, sau đó là trộn hai nửa mất thời gian n , do đó:

$$t(n) = 2t(n/2) + n$$

$$t(n) = 4t(n/4) + 2n \quad (\text{Sau khi thay } t(n/2) = 2t(n/4) + n/2)$$

$$t(n) = 8t(n/8) + 3n \quad (\text{Sau khi thay } t(n/4) = 2t(n/8) + n/4)$$

...

$$t(n) = 2^k t(n/2^k) + kn$$

Chọn $k = \log n$:

$$t(n) = n t(1) + n \log n = n + n \log n = O(n \log n)$$

6. Sắp xếp nhanh (quick sort)

Sắp xếp nhanh

Ta phải sắp xếp dãy A có n phần tử:

1. Nếu $n = 0$ hoặc $n = 1$ thì kết thúc.
2. Chọn một phần tử $v \in A$ làm chốt (pivot).
3. Phân chia $A - \{v\}$ (những phần tử còn lại trong A) thành hai nhóm A_1 và A_2 :

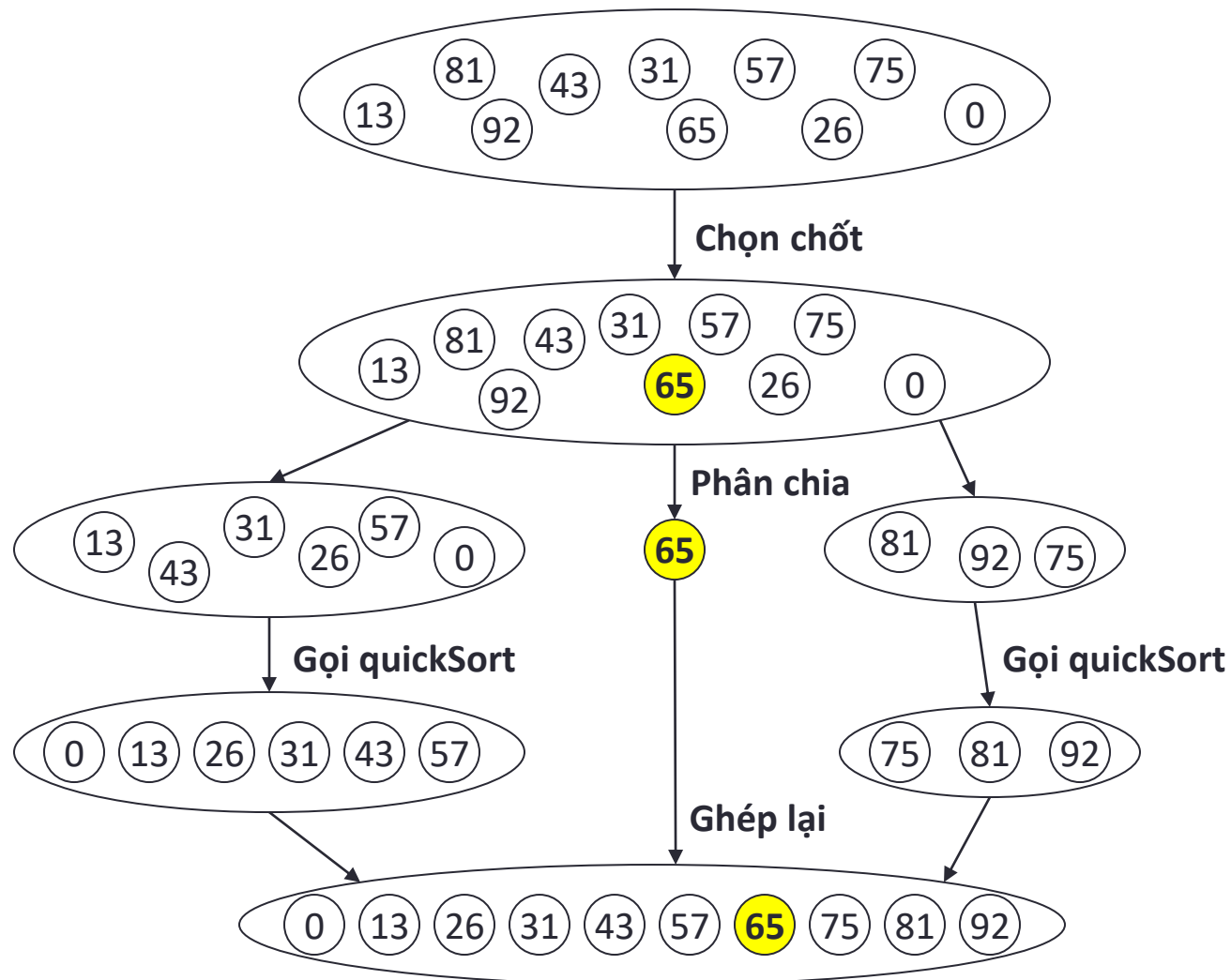
$$A_1 = \{ x \in A - \{v\} \mid x < v \}$$

$$A_2 = \{ x \in A - \{v\} \mid x > v \}$$

4. Trả về $\{ \text{quickSort}(A_1), \{v\}, \text{quickSort}(A_2) \}$

Ví dụ: Nếu $\text{quickSort}(A_1) = \{1, 3\}$, $v = 4$, $\text{quickSort}(A_2) = \{6, 8, 9\}$ thì trả về $\{1, 3, 4, 6, 8, 9\}$.

Ví dụ sắp xếp nhanh



Cài đặt sắp xếp nhanh

```
void quickSort(vector<int> & a) {  
    if (a.size() < 2)  
        return;  
    vector<int> smaller; // Tập phần tử nhỏ hơn chốt  
    vector<int> same;    // Tập phần tử bằng chốt  
    vector<int> larger;  // Tập phần tử lớn hơn chốt  
    int v = a[a.size() / 2]; // Chốt là phần tử chính giữa  
    for (int i = 0; i < a.size(); i++) {  
        if (a[i] < v)  
            smaller.push_back(a[i]);  
        else if (v < a[i])  
            larger.push_back(a[i]);  
        else  
            same.push_back(a[i]);  
    }  
    ... // Xem tiếp ở slide sau  
}
```

Cài đặt sắp xếp nhanh (tiếp)

```
void quickSort(vector<int> & a) {  
    ...  
    quickSort(smaller); // Gọi đệ quy  
    quickSort(larger);  // Gọi đệ quy  
    // Ghép các dãy con đã sắp xếp  
    a.clear();  
    for (int i = 0; i < smaller.size(); i++)  
        a.push_back(smaller[i]);  
    for (int i = 0; i < same.size(); i++)  
        a.push_back(same[i]);  
    for (int i = 0; i < larger.size(); i++)  
        a.push_back(larger[i]);  
}
```

Phân tích sắp xếp nhanh

- Trường hợp tồi nhất (chốt luôn là phần tử nhỏ nhất của dãy con đang xét): $O(n^2)$
- Trường hợp tốt nhất (chốt luôn là trung vị, tức là chia dãy con đang xét thành hai nhóm có kích thước bằng nhau): $O(n \log n)$
- Trường hợp trung bình: $O(n \log n)$
- *Xem phân tích thời gian chạy chi tiết trong sách.*
- *Xem cách cài đặt tốt hơn trong sách.*

Bài tập

1. Sắp xếp dãy { 3, 1, 4, 1, 5, 9, 2, 6, 5 } dùng thuật toán:
 - a. Sắp xếp chọn
 - b. Sắp xếp nổi bọt
 - c. Sắp xếp chèn
2. Dùng thuật toán sắp xếp vun đống để sắp xếp dãy:
{ 142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102 }
3. Dùng thuật toán sắp xếp trộn để sắp xếp dãy:
{ 3, 1, 4, 1, 5, 9, 2, 6 }
4. Dùng thuật toán sắp xếp nhanh để sắp xếp dãy:
{ 7, 6, 1, 5, 2, 8, 4, 3 }