

CONFIDENTIAL

C Programming Basic – week 11

Sắp xếp (Sorting) II

Lecturer :

Do Quoc Huy

Dept of Computer Science

Hanoi University of Technology



Các chủ đề tuần này

- Giải thuật sắp xếp tiên tiến
 - Sắp xếp nhanh (Quick sort)
 - Sắp xếp gộp (Merge sort)
 - Xử lý đệ quy (Recursive processing)
- Exercises



Thuật toán sắp xếp nhanh

Cho một mảng n phần tử (ví dụ: các số nguyên):

- Nếu mảng chỉ chứa 1 phần tử, trở về
- Nếu không
 - Chọn một phần tử để sử dụng như là *phần tử chốt (pivot)*.
 - Chia các phần tử vào 2 mảng con:
 - Các phần tử nhỏ hơn hoặc bằng phần tử chốt
 - Các phần tử lớn hơn phần tử chốt
 - Sắp xếp nhanh 2 mảng con
 - Trả về các kết quả



Phần tử chốt (pivot)

- Chọn phần tử đứng đầu hoặc đứng cuối làm phần tử chốt.
- Chọn phần tử đứng giữa danh sách làm phần tử chốt.
- Chọn phần tử trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối làm phần tử chốt.
- Chọn phần tử ngẫu nhiên làm phần tử chốt.
(Cách này có thể dẫn đến khả năng rơi vào các trường hợp đặc biệt)

Ví dụ

- Cho mảng n số nguyên để sắp xếp:

| | | | | | | | | |
|----|----|----|----|----|----|---|----|-----|
| 40 | 20 | 10 | 80 | 60 | 50 | 7 | 30 | 100 |
|----|----|----|----|----|----|---|----|-----|

Sắp xếp nhanh(Hoare)

- Cho $(R_0, R_1, \dots, R_{n-1})$

K_i : giá trị *khóa (key)* của phần tử chốt

Nếu K_i được đặt trong $S(i)$,

thì $K_j \leq K_{S(i)}$ với $j < S(i)$,

$K_j \geq K_{S(i)}$ với $j > S(i)$.

- $R_0, \dots, R_{S(i)-1}, R_{S(i)}, R_{S(i)+1}, \dots, R_{S(n-1)}$

2 phần



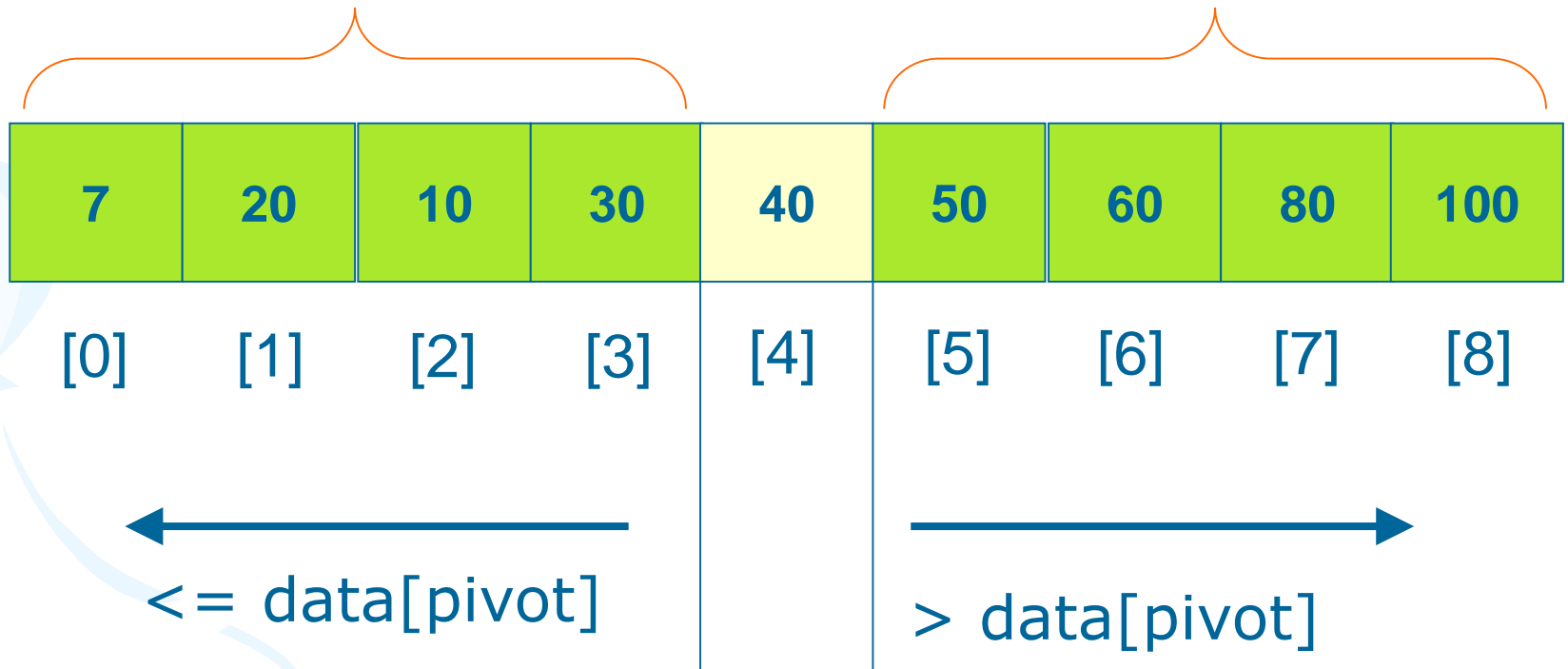
Phân chia mảng

- Cho 1 phần tử chốt, chia các phần tử của mảng để mảng kết quả bao gồm:
 1. Một mảng con chứa các phần tử \geq phần tử chốt
 2. Các mảng con khác chứa các phần tử $<$ phần tử chốt
- Các mảng con được lưu trong mảng dữ liệu ban đầu.
- Lặp lại việc phân chia các phần tử, hoán đổi các phần tử trên/dưới phần tử chốt.

Kết quả phân chia

| | | | | | | | | |
|--------------------|-----|-----|-----|-----|--------------------|-----|-----|-----|
| 7 | 20 | 10 | 30 | 40 | 50 | 60 | 80 | 100 |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
| ← ≤ data[pivot] | | | | | → > data[pivot] | | | |

Đệ quy: Sắp xếp nhanh các mảng con



Ví dụ áp dụng Quick Sort

| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | left | right |
|------|----|----|------|-----|----|------|-----|----|-------|------|-------|
| { 26 | 5 | 37 | 1 | 61 | 11 | 59 | 15 | 48 | 19} | 0 | 9 |
| { 11 | 5 | 19 | 1 | 15} | 26 | { 59 | 61 | 48 | 37} | 0 | 4 |
| { 1 | 5} | 11 | { 19 | 15} | 26 | { 59 | 61 | 48 | 37} | 0 | 1 |
| 1 | 5 | 11 | 15 | 19 | 26 | { 59 | 61 | 48 | 37} | 3 | 4 |
| 1 | 5 | 11 | 15 | 19 | 26 | { 48 | 37} | 59 | { 61} | 6 | 9 |
| 1 | 5 | 11 | 15 | 19 | 26 | 37 | 48 | 59 | { 61} | 6 | 7 |
| 1 | 5 | 11 | 15 | 19 | 26 | 37 | 48 | 59 | 61 | 9 | 9 |
| 1 | 5 | 11 | 15 | 19 | 26 | 37 | 48 | 59 | 61 | | |

Quick Sort

```
void quicksort(element list[], int left,  
int right)  
{  
    int pivot, i, j;  
    element temp;  
    if (left < right) {  
        i = left;      j = right+1;  
        pivot = list[left].key;  
        do {  
            do i++; while (list[i].key < pivot);  
            do j--; while (list[j].key > pivot);  
            if (i < j) SWAP(list[i], list[j], temp);  
        } while (i < j);  
        SWAP(list[left], list[j], temp);  
        quicksort(list, left, j-1);  
        quicksort(list, j+1, right);  
    }  
}
```



Exercise 11-1: Quick sort

- Giả sử, bạn tạo danh bạ điện thoại.
- Tạo mảng chứa 100 phần tử danh bạ.
- Viết chương trình
 - đọc khoảng 10 dữ liệu từ 1 file vào mảng.
 - Ghi dữ liệu vào file đầu ra sau khi sắp xếp theo trật tự tăng dần của tên.
- Sử dụng Quick sort để sắp xếp.



Exercise 11-2

- Khởi tạo một mảng ngẫu nhiên **n** số nguyên. n được nhập bởi người dùng.
- Sắp xếp mảng bằng sắp xếp chèn
- Và sử dụng quicksort
- So sánh thời gian chạy của 2 giải thuật.
- Chạy chương trình với các dữ liệu khác nhau của n để xem hiệu quả.

Exercise 11-3 Kết hợp quick sort và insertion sort

- Khi 1 chương trình sắp xếp một số lượng nhỏ dữ liệu, sử dụng sắp xếp chèn sẽ nhanh hơn quick sort.
- Do đó để sắp xếp hiệu quả, chương trình có thể thay đổi thuật toán dựa trên lượng dữ liệu.
- Viết 1 hàm lựa chọn thuật toán sắp xếp
 - Nếu số lượng dữ liệu $> x$, hàm sẽ chọn quick sort.
 - Nếu không, nó sẽ chọn sắp xếp chèn.
- Chú thích: “x” là tham số nhập vào từ chương trình.
- Đọc file text có nhiều hơn 100 ký tự, sắp xếp 100 ký tự đầu tiên, hiển thị kết quả ra màn hình.



Sắp xếp trộn (Merge Sort)

- Bài toán: Cho n phần tử, sắp xếp các phần tử theo trật tự không giảm
- Áp dụng phép *chia để trị* để giải
 - Nếu $n=1$, kết thúc (mọi danh sách một phần tử đã được sắp xếp)
 - Nếu $n>1$, chia các phần tử vào 2 mảng con;
+sắp xếp mỗi mảng;
+kết hợp thành 1 mảng đã sắp xếp duy nhất



Algorithm

MergeSort (E[0 .. N])

if $N < \text{threshold}$

InsertionSort (E[0..N])

else

copy E[0.. N/2] to U[0.. N/2]

copy E[N/2 .. N] to V[0 .. N-N/2]

MergeSort(U[0 .. N/2])

MergeSort(V[0 .. N-N/2])

Merge(U[0 .. N/2], V[0 .. N-N/2}, E[0 .. N])



Merge algorithm

```
Merge (U[0..m] , V[0..n] , E[0..n+m] )
```

```
  i = 0 ,   j = 0
```

```
  k = 0
```

```
  while k <  n+m
```

```
    if U[i] < V [j]
```

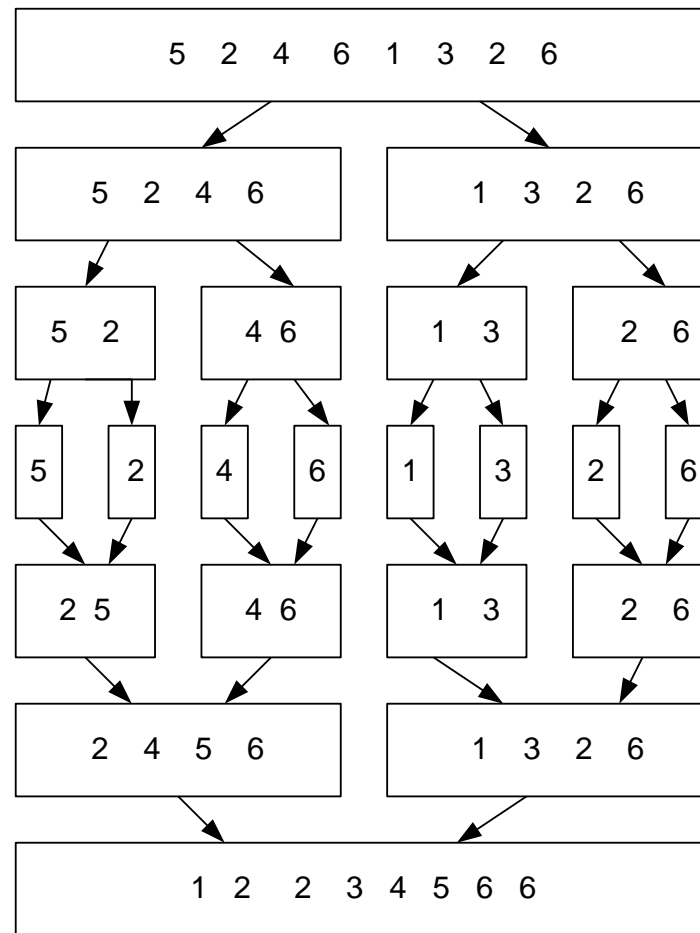
```
      E[k] = U[i] , i++
```

```
    else
```

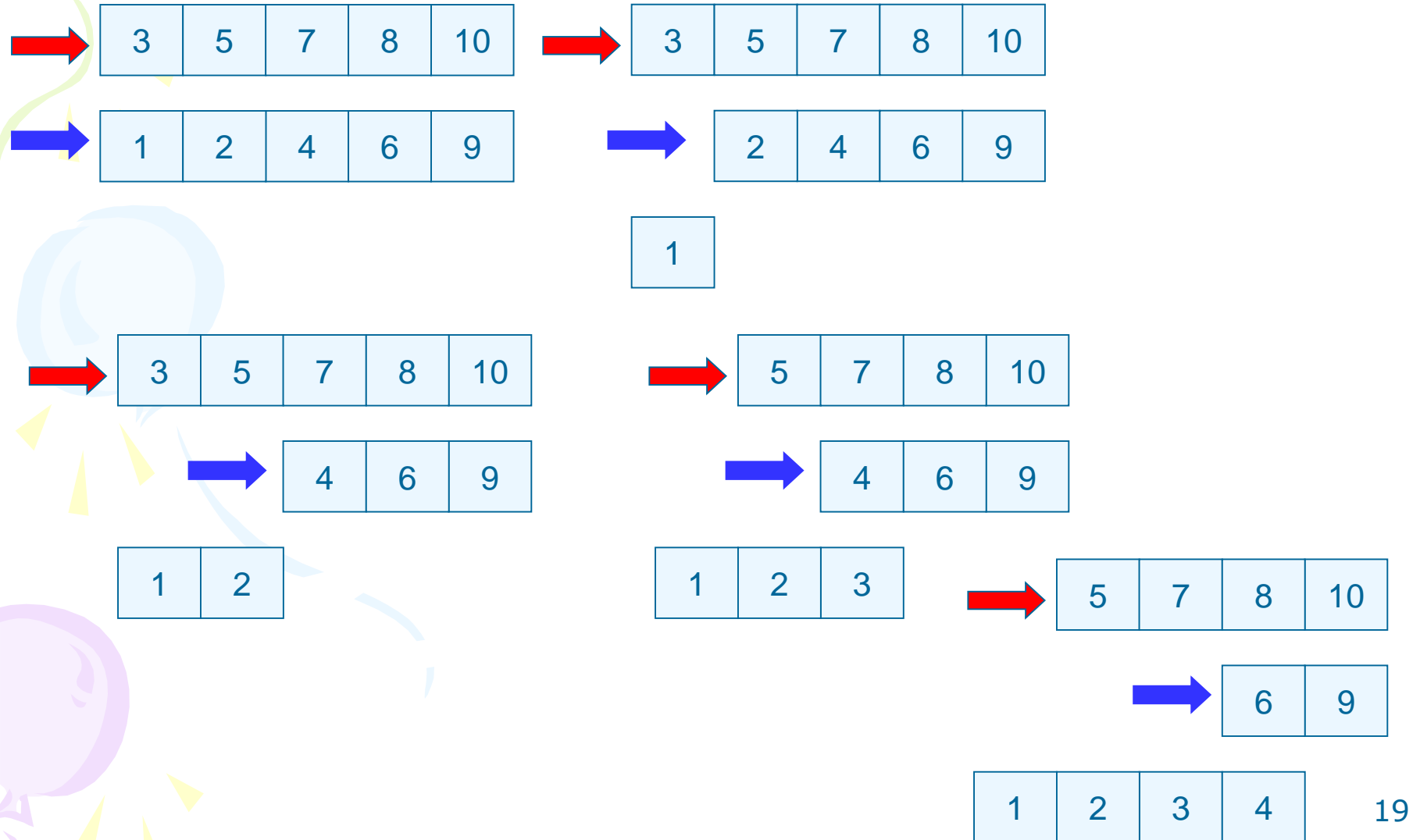
```
      E[k] = V[j] , j++
```

```
    k++
```

Merge Sort: Example



Quá trình trộn





Cài đặt merge sort

```
/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;
        // Same as (l+r)/2, but avoids overflow for large l and h
        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

```
// Trộn 2 mảng con vào arr[].  
// Mảng con thứ 1 là arr[l..m]  
// Mảng con thứ 2 là arr[m+1..r]  
void merge(int arr[], int l, int m, int r)  
{
```

```
    int i, j, k;
```

```
    int n1 = m - l + 1;
```

```
    int n2 = r - m;
```

```
    /* tạo các mảng tạm thời*/
```

```
    int L[n1], R[n2];
```

```
    /* Chép dữ liệu vào các mảng tạm thời arrays L[] và R[] */
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[l + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[m + 1 + j];
```

```
/* Trộn các mảng tạm thời trở lại vào arr[l..r]*/  
i = 0; // Khởi tạo chỉ số cho mảng con thứ 1  
j = 0; // Khởi tạo chỉ số cho mảng con thứ 2  
k = l; // Khởi tạo chỉ số cho mảng con được trộn  
while (i < n1 && j < n2)  
{  
    if (L[i] <= R[j])  
    {  
        arr[k] = L[i];  
        i++;  
    }  
    else  
    {  
        arr[k] = R[j];  
        j++;  
    }  
    k++;  
}
```



```
/* Chép các phần tử còn lại của L[], nếu còn */
```

```
while (i < n1)
```

```
{
```

```
    arr[k] = L[i];
```

```
    i++;
```

```
    k++;
```

```
}
```

```
/* Chép các phần tử còn lại của R[], nếu còn */
```

```
while (j < n2)
```

```
{
```

```
    arr[k] = R[j];
```

```
    j++;
```

```
    k++;
```

```
}
```

```
}
```



Exercise: 11-3 Merge sort

- Tạo danh bạ điện thoại
- Sử dụng merge sort để sắp xếp.

A decorative graphic on the left side of the slide featuring a green balloon at the top, a blue balloon in the middle, and a purple balloon at the bottom, all with yellow streamers and triangular flags.

Exercise: Xử lý đệ quy

- Viết thuật toán đệ quy để xử lý một bộ bài. Tham số là:
- (i) các lá bài chưa chia, và
- (ii) người nào sẽ nhận lá bài tiếp theo. Giả định:
- Người chơi ngồi xung quanh một bàn tròn;
- Bắt đầu chia bài cho người bên trái người chia;
- Mỗi lần chia 1 lá bài cho 1 người chơi sau đó chuyển sự chú ý sang người bên trái; và
- Chia cho đến khi hết bài.

Exercise: Xử lý đệ quy

- Viết 1 hàm đệ quy **void recurTriangle (int n, char ch)** in ra 1 tam giác ngược.
- Tham số *ch* là ký tự để in ra tam giác và *n* số ký tự trên hàng đầu.
- Ví dụ, nếu *n* là 7 và *ch* là '+', thì đầu ra của hàm sẽ có dạng sau:

```
+++++++  
+++++++  
+++++++  
+++++++  
++++  
+++  
++  
+
```



Exercise 11-4: Sắp xếp xâu

- Viết chương trình sắp xếp các xâu bằng sắp xếp quick sort theo trật tự alphabetical dựa theo các chỉ dẫn sau.

I. So sánh các chuỗi ký tự

- Viết hàm "preceding()" để xem ký tự nào đứng trước trong thứ tự alphabet.

```
int preceding(char *first, char *second)
```

- Giá trị trả về theo thứ tự alphabet
 - Trường hợp chuỗi ký tự tham số "first" là trước chuỗi ký tự của tham số "second" : **1**
 - Trường hợp chuỗi ký tự tham số "first" là bằng với chuỗi ký tự của tham số "second" : **0**
 - Trường hợp chuỗi ký tự tham số "first" là trước chuỗi ký tự của tham số "second" : -1

II. Nhập xâu ký tự vào từ file

- Viết hàm `setup_nameList()` để đọc tên của từ 2 đến 25 người từ file và lưu vào mảng `nameList[]` của các xâu ký tự (thực tế, đây là một mảng các con trỏ trỏ đến xâu ký tự)

```
int setup_nameList(char *namelist[], char *filename)
```



III. Cài đặt Quicksort

- Viết hàm “qsort_name()” để sắp xếp các chuỗi ký tự trong mảng “namelist[]” theo thứ tự alphabetical bằng quick sort, sử dụng các hàm bạn đã xây dựng.