ĐẠI NAM
UNIVERSITY

Phần trình bày của:

ĐẬU HẢI PHONG

Giảng viên

Đại Nam, ngày 01 tháng 12 năm 2022

LƯU Ý

KHÔNG NÓI CHUYỆN RIÊNG

KHÔNG SỬ DỤNG ĐIỆN THOẠI

KHÔNG NGỦ GẬT

GHI CHÉP ĐẦY ĐỦ

ĐẠI NAM
UNIVERSITY

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

CHƯƠNG 4

THUẬT TOÁN SẮP XẾP

NỘI DUNG

- Bài toán sắp xếp
- Sắp xếp chèn - Insertion sort
- Sắp xếp lựa chọn – Selection sort
- Sắp xếp nổi bột – Bubble sort
- Sắp xếp trộn – Merge Sort
- Sắp xếp nhanh – Quick Sort
- Bài tập

Bài toán sắp xếp

- Là cách sắp xếp lại các phần tử theo chiều tăng hay giảm dần theo tiêu chí
  - VD: Sắp xếp danh sách sinh viên theo mã sinh viên, ĐTB, tên,… theo chiều tăng dần.
- Giúp có cái nhìn tổng quát về dữ liệu, dễ dàng tìm kiếm.
  - VD: Tìm 3 sinh viên có ĐTB cao nhất lớp

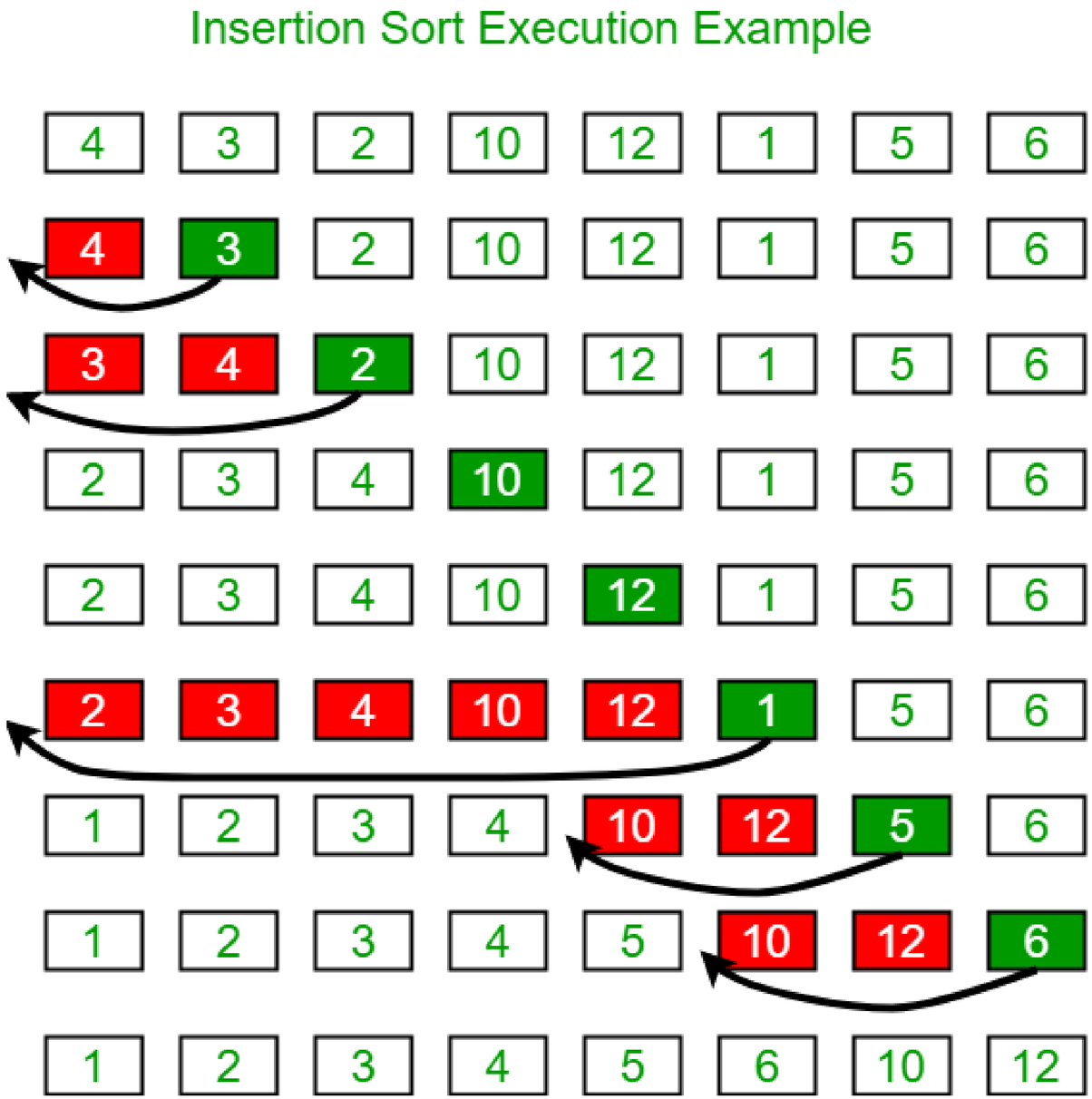
Sắp xếp chèn – Insertion sort

- **Ý tưởng:** Duyệt từng PT và chèn từng PT đó vào đúng vị trí trong mảng con (đã sắp xếp)
- Hình minh họa:

6 5 3 1 8 7 2 4

Sắp xếp chèn – Insertion sort

- Ví dụ minh họa



Sắp xếp chèn – Insertion sort

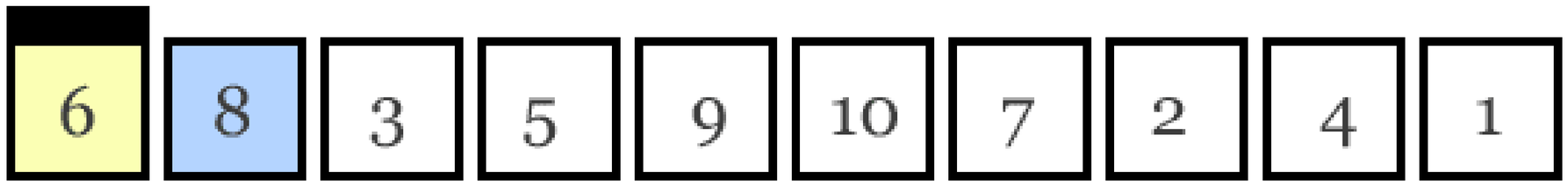
```
4  void insertionSort(int arr[], int n)
5  {
6      int i, key, j;
7      for (i = 1; i < n; i++)
8      {
9          key = arr[i];
10         j = i-1;
11
12         /* Di chuyển các phần tử có giá trị lớn hơn giá trị
13         key về sau một vị trí so với vị trí ban đầu
14         của nó */
15         while (j >= 0 && arr[j] > key)
16         {
17             arr[j+1] = arr[j];
18             j = j-1;
19         }
20         arr[j+1] = key;
21     }
22 }
```

Sắp xếp chèn – Insertion sort

- Độ phức tạp thuật toán
  - Trường hợp tốt: O(n)
  - Trung bình: $O(n^2)$
  - Trường hợp xấu: $O(n^2)$

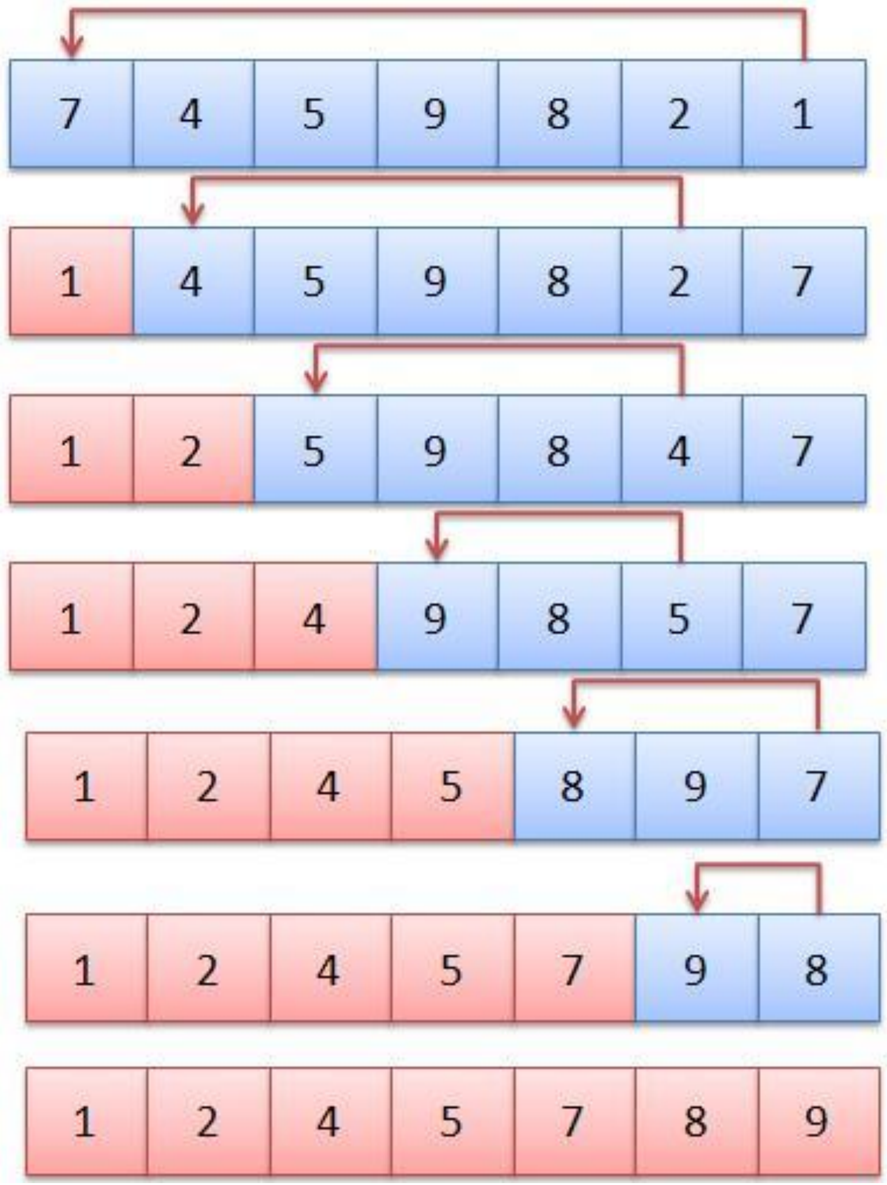
Sắp xếp lựa chọn – Selection sort

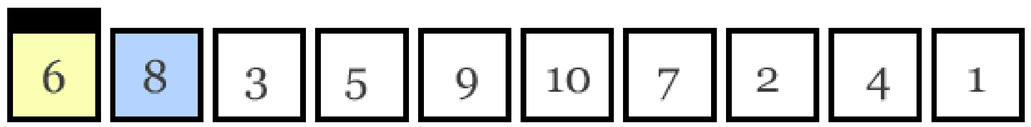
- **Ý tưởng:**
  - Tìm kiếm phần tử nhỏ nhất và đổi chỗ nó với phần tử đầu tiên.
  - Tìm kiếm phần tử nhỏ nhất trong phần còn lại và đổi chỗ nó với phần tử thứ hai
  - Tiếp tục cho đến khi toàn bộ đã được sắp xếp.



Sắp xếp lựa chọn – Selection sort

- **Minh họa:**





Sắp xếp lựa chọn – Selection sort

- **Ví dụ: arr[] = 62 24 15 22 1**

// Tìm phần tử nhỏ nhất trong trong arr[0...4] và đổi chỗ nó với phần tử đầu tiên

[1] 24 15 22 62

// Tìm phần tử nhỏ nhất trong trong arr[1...4] và đổi chỗ nó với phần tử đầu tiên của arr[1...4]

1 [15] 24 22 62

// Tìm phần tử nhỏ nhất trong trong arr[2...4] và đổi chỗ nó với phần tử đầu tiên của arr[2...4]

1 15 [22] 24 62

// Tìm phần tử nhỏ nhất trong trong arr[3...4] và đổi chỗ nó với phần tử đầu tiên của arr[3...4]

11 12 22 [24] 62

Sắp xếp lựa chọn – Selection sort

- ```
23 // Hàm selection sort
24 void selectionSort(int arr[], int n)
25 {
26   int i, j, min_idx;
27   // Di chuyển ranh giới của mảng đã sắp xếp và chưa sắp xếp
28   for (i = 0; i < n-1; i++)
29   {
30   // Tìm phần tử nhỏ nhất trong mảng chưa sắp xếp
31   min_idx = i;
32   for (j = i+1; j < n; j++)
33   if (arr[j] < arr[min_idx])
34   min_idx = j;
35
36   // Đổi chỗ phần tử nhỏ nhất với phần tử đầu tiên
37   swap(arr[min_idx], arr[i]);
38   }
39 }
```
```

Sắp xếp lựa chọn – Selection sort

- **Độ phức tạp thuật toán**
  - Trường hợp tốt: $O(n^2)$
  - Trung bình: $O(n^2)$
  - Trường hợp xấu: $O(n^2)$

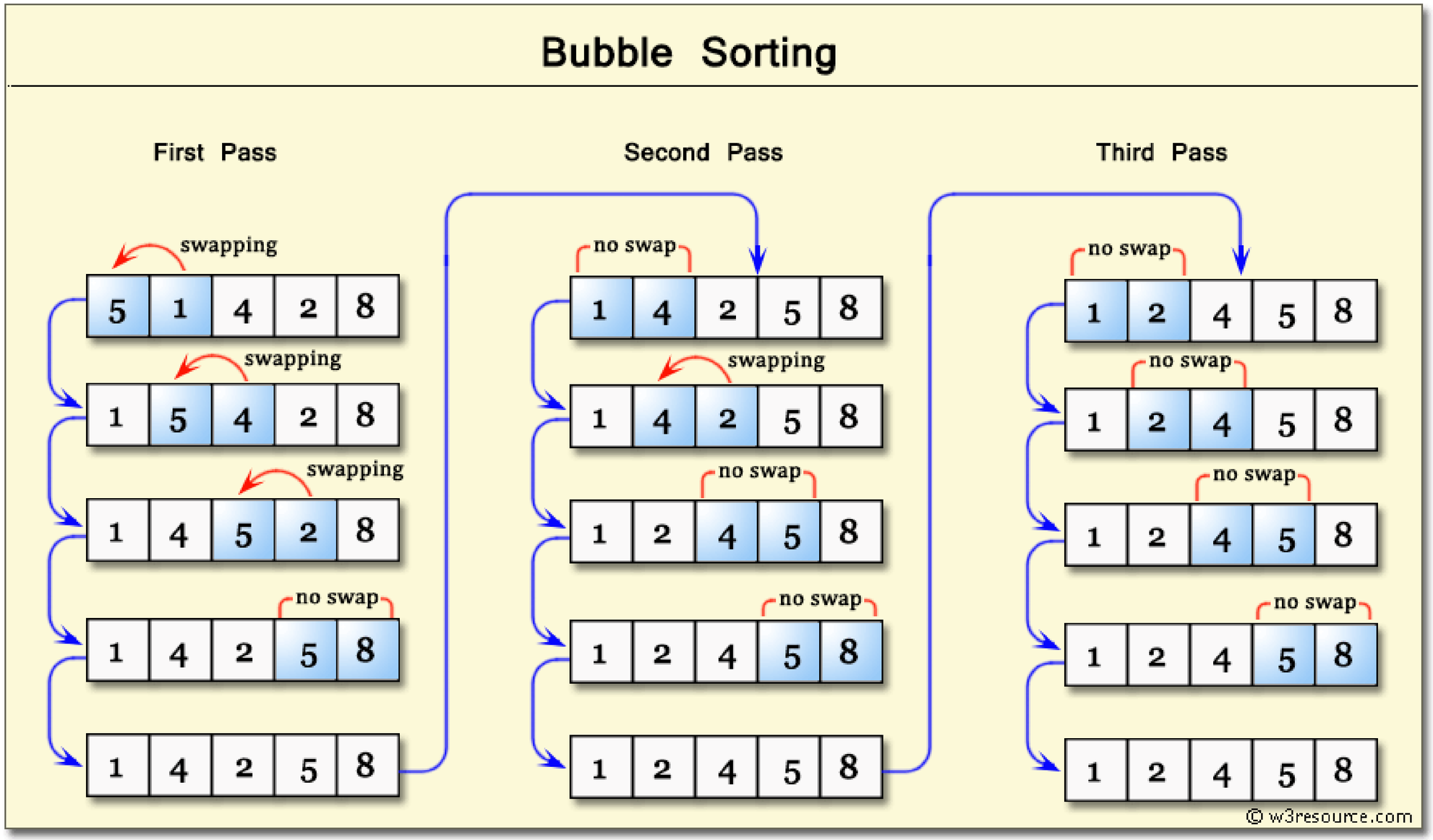
Sắp xếp nổi bột – Bubble sort

- Ý tưởng:
  - Lặp lại đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự
    - Số sau bé hơn số trước với trường hợp sắp xếp tăng dần
  - Cho đến khi dãy số được sắp xếp

6 5 3 1 8 7 2 4

Sắp xếp nổi bột – Bubble sort

- Ví dụ:



Sắp xếp nổi bột – Bubble sort

```
41  void bubbleSort(int arr[], int n)
42  {
43      int i, j;
44      bool haveSwap = false;
45      for (i = 0; i < n-1; i++){
46      // i phần tử cuối cùng đã được sắp xếp
47          haveSwap = false;
48          for (j = 0; j < n-i-1; j++){
49              if (arr[j] > arr[j+1]){
50                  swap(arr[j], arr[j+1]);
51                  haveSwap = true; // Kiểm tra lần lặp này có swap không
52              }
53          }
54          // Nếu không có swap nào được thực hiện => mảng đã sắp xếp.
55          //Không cần lặp thêm
56          if(haveSwap == false){
57              break;
58          }
59      }
60  }
```

Sắp xếp nổi bột – Bubble sort

- Độ phức tạp thuật toán
  - Trường hợp tốt: O(n)
  - Trung bình: $O(n^2)$
  - Trường hợp xấu: $O(n^2)$

Sắp xếp trộn – Merge Sort

- Ý tưởng:
  - Chia mảng sắp xếp thành 2 nửa
  - Tiếp tục lặp lại ở các mảng đã chia
  - Gộp các mảng đó thành mảng đã sắp xếp

6 5 3 1 8 7 2 4

Sắp xếp trộn – Merge Sort

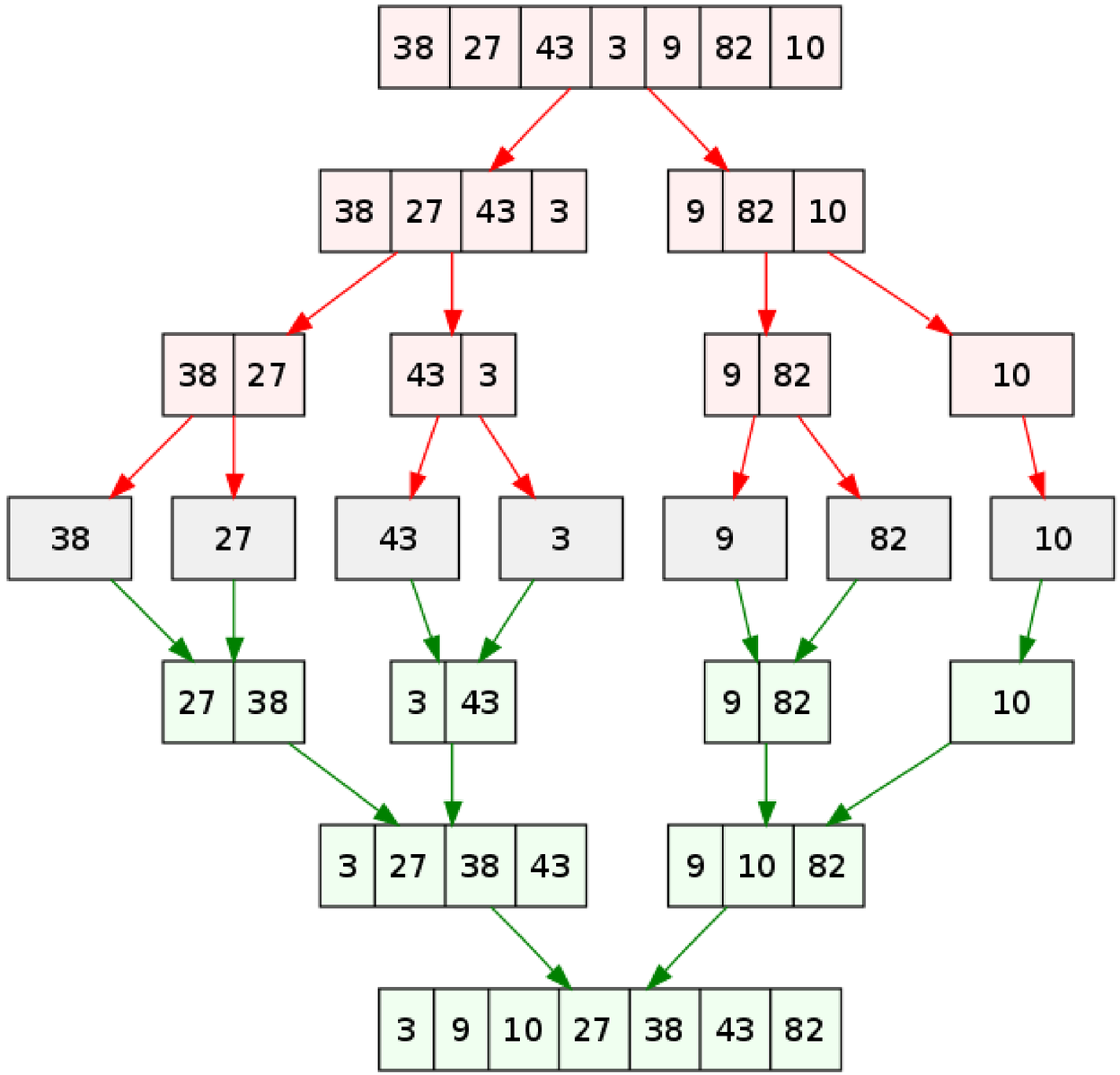
- **Ý tưởng: hàm mergeSort(arr[], l, r)**

Nếu r > l

1. Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:
   middle m = (l+r)/2
2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:
   mergeSort(arr, l, m)
3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:
   mergeSort(arr, m+1, r)
4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):
   merge(arr, l, m, r)

Sắp xếp trộn – Merge Sort

- Ví dụ:



Sắp xếp trộn – Merge Sort

- **Cách trộn 2 mảng con:**
  - Giả sử ta có 2 mảng con lần lượt là:
  - arr1 = [1 9 10 10] , n1 = 4 // chiều dài của mảng con
  - arr2 = [3 5 7 9], n2 = 4
  - sort_arr = [] // Mảng lưu lại tiến trình gộp
  - Khởi tạo i = 0, j = 0 tương ứng là chỉ số bắt đầu của arr1 và arr2
  - Xét thấy arr1[i] < arr2[j] => chèn arr1[i] vào cuối mảng sort_arr, tăng i lên 1 đơn vị
  - => sort_arr = [1], i = 1
  - Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
  - => sort_arr = [1, 3], i = 1, j = 1
  - Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
  - => sort_arr = [1, 3, 5], i = 1, j = 2

Sắp xếp trộn – Merge Sort

- **Cách trộn 2 mảng con:**
  - Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
  - => sort_arr = [1, 3, 5, 7], i = 1, j = 3
  - Xét thấy arr1[i] = arr2[j] => chèn arr1[i] hoặc arr2[j] vào cuối mảng sort_arr
  - Giả sử tôi chọn arr1, tăng i lên 1 đơn vị
  - => sort_arr = [1, 3, 5, 7, 9], i = 2, j = 3
  - Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
  - => sort_arr = [1, 3, 5, 7, 9, 9], i = 1, j = 4
  - Do j >= n2, tiếp tục tăng i chừng nào i < n1 thi thêm phần tử ở arr1[i]ư vào sort_arr.
  - Sau cùng ta được mảng đã sắp xếp là sort_arr = [1, 3, 5, 7, 9, 9, 10, 10]

Sắp xếp trộn – Merge Sort

- Gộp hai mảng con

```
// Gộp hai mảng con arr[l...m] và 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 */
    int L[n1], R[n2];

    /* Copy dữ liệu sang các mảng tạm */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Gộp hai mảng tạm vừa rồi vào mảng arr*/
    i = 0; // Khởi tạo chỉ số bắt đầu của mảng con đầu tiên
    j = 0; // Khởi tạo chỉ số bắt đầu của mảng con thứ hai
    k = l; // IKhởi tạo chỉ số bắt đầu của mảng lưu kết quả
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    /* Copy các phần tử còn lại của mảng L vào arr nếu có */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    /* Copy các phần tử còn lại của mảng R vào arr nếu có */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Sắp xếp trộn – Merge Sort

- Trộn 2 mảng

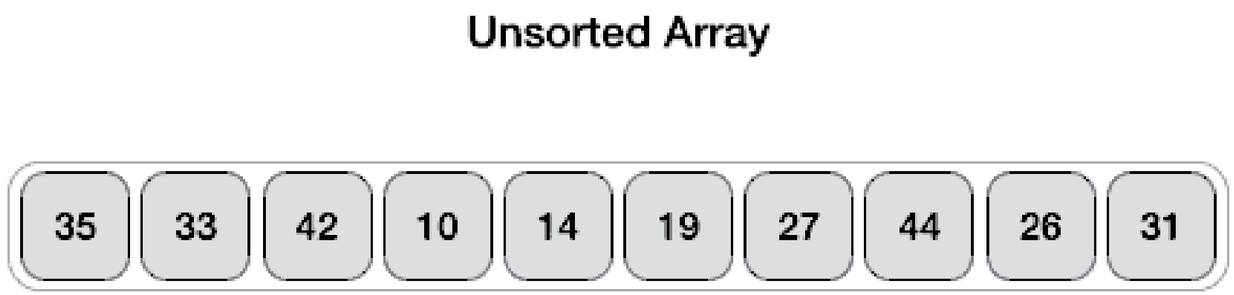
```
53  /* l là chỉ số trái và r là chỉ số phải của mảng cần được sắp xếp */
54  void mergeSort(int arr[], int l, int r)
55  {
56      if (l < r)
57      {
58          // Tương tự (l+r)/2, nhưng cách này tránh tràn số khi l và r lớn
59          int m = l+(r-l)/2;
60
61          // Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng
62          mergeSort(arr, l, m);
63          mergeSort(arr, m+1, r);
64
65          merge(arr, l, m, r);
66      }
67  }
```

Sắp xếp trộn – Merge Sort

- **Độ phức tạp thuật toán:**
  - Trường hợp tốt: O(nlog(n))
  - Trung bình: O(nlog(n))
  - Trường hợp xấu: O(nlog(n))

Sắp xếp nhanh – Quick Sort

- **Ý tưởng:**
  - Chọn một phần tử trong mảng làm điểm đánh dấu(pivot)
  - Chia mảng thành các mảng con dựa vào pivot
  - Một số lựa chọn pivot:
    - Phần tử đầu tiên
    - Phần tử cuối cùng
    - Phần tử ngẫu nhiên
    - Phần tử nằm giữa



Sắp xếp nhanh – Quick Sort

- **Thuật toán phân đoạn – Partition:**
  - Left – phần tử trái nhất; Right – phẩn tử phải nhất (bỏ qua pivot)
  - Chừng nào left < right mà arr[left] > pivot và arr[right] < pivot thì đổi chỗ hai phần tử left và right
  - Cuối cùng, ta đổi chỗ hai phần tử left và pivot cho nhau

Sắp xếp nhanh – Quick Sort

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];    // pivot
    int left = low;
    int right = high - 1;
    while(true){
        while(left <= right && arr[left] < pivot) left++; // Tìm phần tử >= arr[pivot]
        while(right >= left && arr[right] > pivot) right--; // Tìm phần tử <= arr[pivot]
        if (left >= right) break; // Đã duyệt xong thì thoát vòng lặp
        swap(arr[left], arr[right]); // Nếu chưa xong, đổi chỗ.
        left++; // Vì left hiện tại đã xét, nên cần tăng
        right--; // Vì right hiện tại đã xét, nên cần giảm
    }
    swap(arr[left], arr[high]);
    return left; // Trả về chỉ số sẽ dùng để chia đôi mảng
}
```

Sắp xếp nhanh – Quick Sort

- **Ví dụ: cho quá trình phân đoạn**
  - arr[] = {10, 80, 30, 90, 40, 50, 70}
  - Indexes: 0 1 2 3 4 5 6
  - pivot = 6, left = 0, right = 5
  - arr[left] = 10 < arr[pivot] = 70 và left <= right, left = 1
  - arr[left] = 80 > arr[pivot] = 70, tạm dừng
  - arr[right] = 50 < arr[pivot] = 70, tạm dừng
  - Do left < right, đổi chỗ arr[left], arr[right]
  - arr[] = {10, 50, 30, 90, 40, 80, 70}
  - left = 2, right = 4

Sắp xếp nhanh – Quick Sort

- **Ví dụ cho quá trình phân đoạn: (tiếp)** arr[] = {10, 50, 30, 90, 40, 80, 70}
  - arr[left] = 30 < arr[pivot] = 70 và left <= right, left = 3
  - arr[left] = 90 > arr[pivot] = 70, tạm dừng
  - arr[right] = 40 < arr[pivot] = 70, tạm dừng
  - Do left < right, đổi chỗ arr[left], arr[right]
  - arr[] = {10, 50, 30, 40, 90, 80, 70}
  - left = 4, right = 3
  - // Do left >= right
  - arr[] = {10, 50, 30, 40, 70, 80, 90}. // Đổi chỗ arr[left] và arr[pivot]
  - Bây giờ, 70 đã nằm đúng vị trí, các phần từ <= 70 nằm phía trước và lớn hơn 70 nằm phía sau.

Sắp xếp nhanh – Quick Sort

- **Quy trình của thuật toán sắp xếp quick sort:**
  - B1. Lấy phần tử chốt là phần tử ở cuối danh sách
  - B2. Chia mảng theo phần tử chốt
  - B3. Sử dụng sắp xếp nhanh một cách đệ qui với mảng con bên trái
  - B4. Sử dụng sắp xếp nhanh một cách đệ qui với mảng con bên phải

Sắp xếp nhanh – Quick Sort

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi là chỉ số nơi phần tử này đã đứng đúng vị trí
         và là phần tử chia mảng làm 2 mảng con trái & phải */
        int pi = partition(arr, low, high);

        // Gọi đệ quy sắp xếp 2 mảng con trái và phải
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Sắp xếp nhanh – Quick Sort

- Độ phức tạp thuật toán của quick sort:
  - Trường hợp tốt: O(nlog(n))
  - Trung bình: O(nlog(n))
  - Trường hợp xấu: O(n^2)

Sắp xếp vun đống – Heap Sort

- Sinh viên tự tìm hiểu

Hỏi - Đáp

Bài tập

- Bài 1.

ĐẠI NAM
UNIVERSITY

Cảm ơn đã lắng nghe!

ĐẬU HẢI PHONG
Giảng viên
dauhaiphong@dainam.edu.vn
0912441435