CMC UNIVERSITY

CHƯƠNG 3. SẮP XẾP

©CMC Univ. 2022

CMC UNIVERSITY

Nội dung

©CMC Univ. 2022

CMC UNIVERSITY

BÀI TOÁN SẮP XẾP

©CMC Univ. 2022

CMC UNIVERSITY

3.1 Bài toán sắp xếp

D. Knuth: "40% thời gian hoạt động của các máy tính là dành cho sắp xếp"

- **Sắp xếp** (Sorting) là quá trình tổ chức lại các phần tử của một tập đối tượng nào đó theo một thứ tự ấn định. Chẳng hạn thứ tự tăng dần hoặc giảm dần (ascending or descending order) của một dãy số, thứ tự từ điển đối với dãy xâu kí tự, ...
- Yêu cầu về sắp xếp thường xuyên xuất hiện trong các ứng dụng tin học, với các mục đích khác nhau: sắp xếp dữ liệu lưu trữ trong máy tính để thuận lợi cho tìm kiếm, sắp xếp các kết quả xử lý để in ra trên bảng biểu, …
- Dữ liệu sắp xếp có thể xuất hiện dưới nhiều dạng khác nhau. Ta qui ước tập đối tượng sắp xếp là tập các bản ghi (records), mỗi bản ghi gồm một số trường (fields) dữ liệu.
- **Khóa** sắp xếp (sort key) là một trường (hoặc một số trường) dữ liệu trong bản ghi xác định thứ tự sắp xếp của bản ghi. Ta cần sắp xếp các bản ghi theo thứ tự của các khoá.

©CMC Univ. 2022

3.1 Bài toán sắp xếp (tiếp)

- Giới hạn bài toán sắp xếp có dạng sau đây:
  - Input: Dãy gồm n số $a_1, a_2, .., a_n$
  - Output: Một hoán vị (sắp xếp lại) $a'_1, a'_2, \ldots a'_n$ của dãy số đã cho thỏa mãn điều kiện: $a'_1 \leq a'_2 \leq \ \ldots \leq a'_n$
- Ứng dụng của sắp xếp:
  - ✓ Quản trị cơ sở dữ liệu (Database management)
  - ✓ Trong các bài toán khoa học kỹ thuật
  - ✓ Các thuật toán lập lịch (Scheduling algorithms)
  - ✓ Máy tìm kiếm Web (Web search engine)
  - ✓ Nhiều ứng dụng khác…

©CMC Univ. 2022

3.1 Bài toán sắp xếp (tiếp)

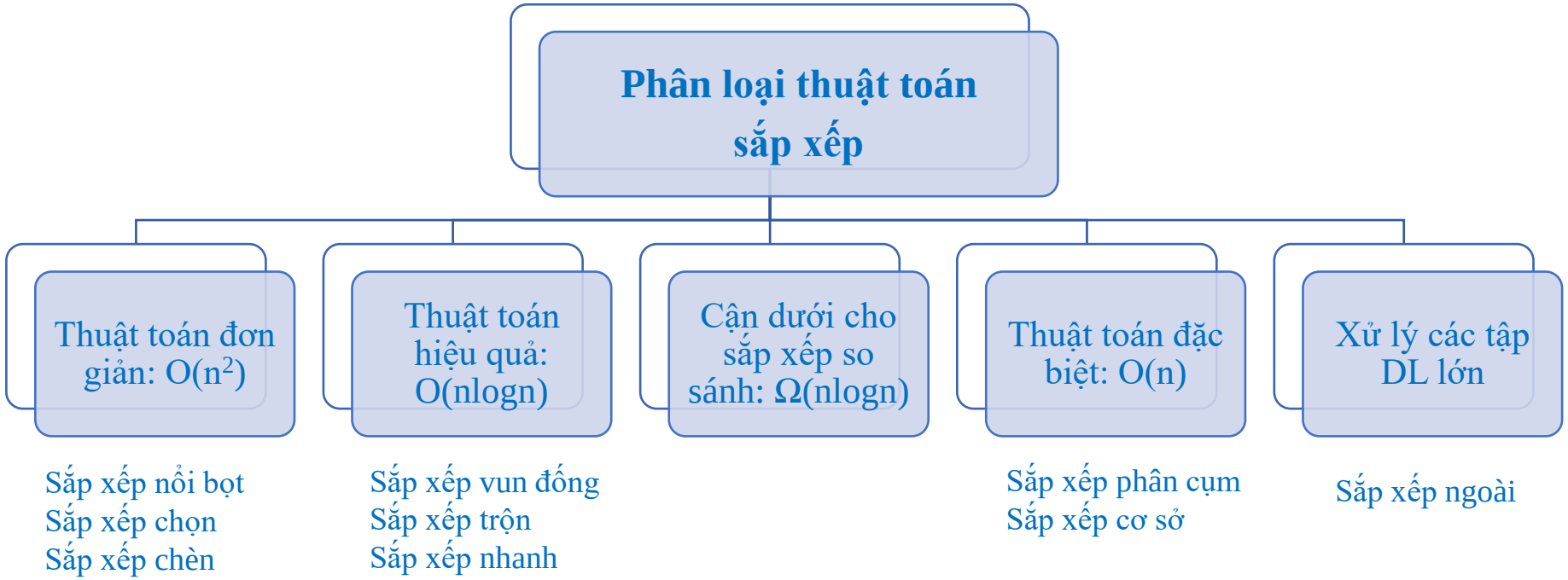
- Các đặc trưng của một thuật toán sắp xếp
  - Tại chỗ (in place): nếu không gian nhớ phụ mà thuật toán đòi hỏi là O(1) thì nghĩa là bị chặn bởi hằng số không phụ thuộc vào độ dài của dãy cần sắp xếp.
  - Ổn định (stable): nếu các phần tử có cùng giá trị vẫn giữ nguyên thứ tự tương đối của chúng như trước khi sắp xếp.

©CMC Univ. 2022

3.1 Bài toán sắp xếp (tiếp)

❖ Phân loại các thuật toán sắp xếp:

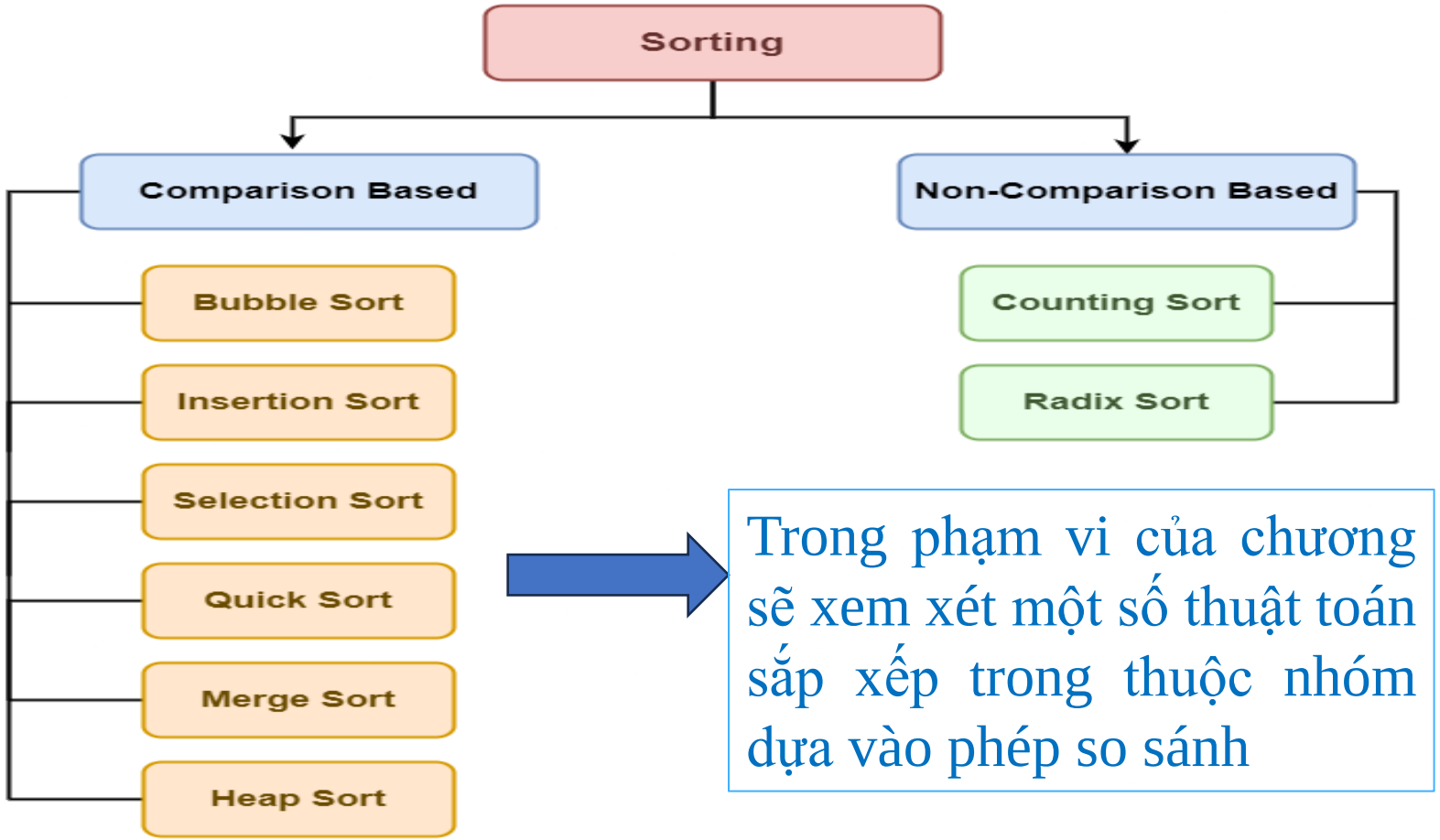
- ➢ Sắp xếp trong (internal sort): đòi hỏi tập dữ liệu được đưa toàn bộ vào bộ nhớ trong
- ➢ Sắp xếp ngoài (external sort): tập dữ liệu không thể cùng lúc đưa toàn bộ vào bộ nhớ trong, nhưng có thể đọc vào từng phần từ bộ nhớ ngoài.



©CMC Univ. 2022

3.1 Bài toán sắp xếp (tiếp)

Phân loại các thuật toán sắp xếp dựa vào việc sử dụng phép toán so sánh



Trong phạm vi của chương sẽ xem xét một số thuật toán sắp xếp trong thuộc nhóm dựa vào phép so sánh

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP NỔI BỌT

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp nổi bọt (Bubble)

- Ý tưởng: Cho một danh sách gồm n phần tử không có thứ tự
  - So sánh các phần tử liền kề trong danh sách, đổi chỗ các phần tử liền kề nếu chúng không đúng thứ tự. Quá trình này được lặp lại n-1 lần cho danh sách n phần tử.
  - Trong mỗi lần lặp, phần tử lớn nhất của danh sách sẽ được chuyển xuống cuối danh sách. Sau lần lặp thứ hai, phần tử lớn thứ hai sẽ được đặt ở vị trí thứ hai tính từ vị trí cuối cùng trong danh sách. Quá trình tương tự được lặp lại cho đến khi danh sách được sắp xếp.

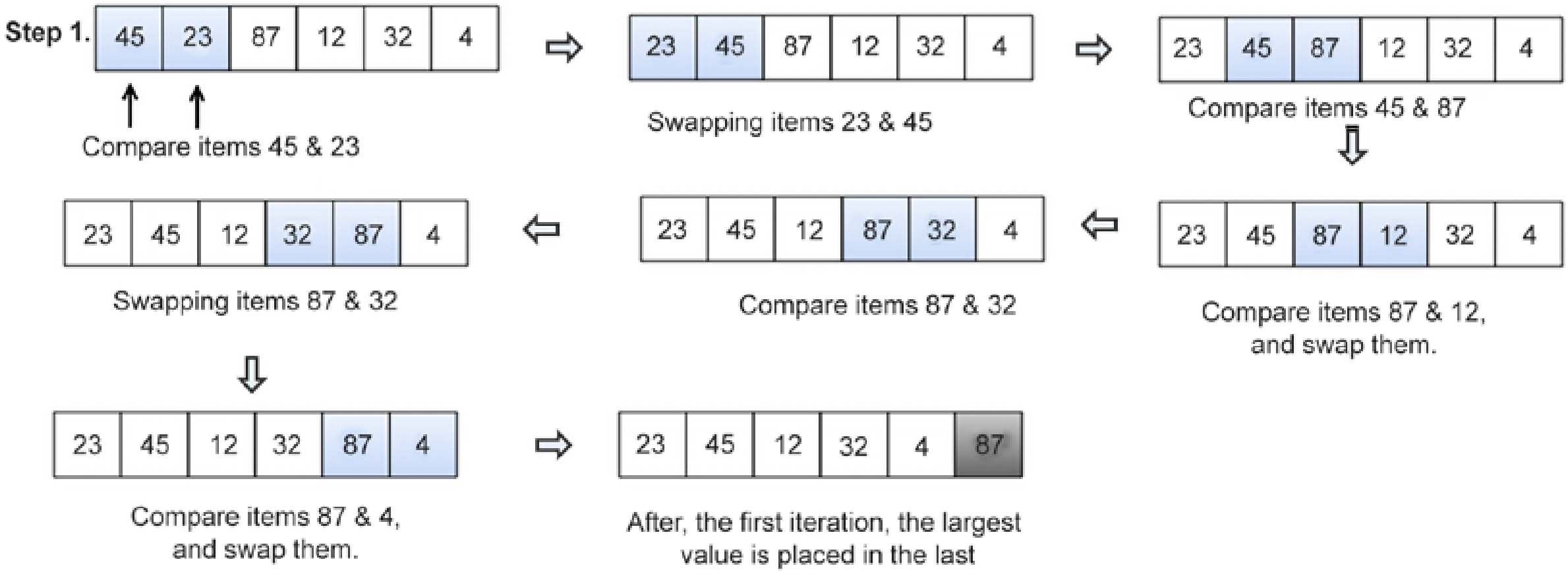
Nếu hình dung dãy khóa được đặt thẳng đứng, thì sau từng lượt sắp xếp các giá trị khóa nhỏ sẽ "*nổi*" dần lên giống như các bọt nước nổi lên trong nồi nước đang sôi.

- ***Ví dụ***: sắp xếp dãy số 45, 23, 87, 12, 32, 4

©CMC Univ. 2022

Ví dụ

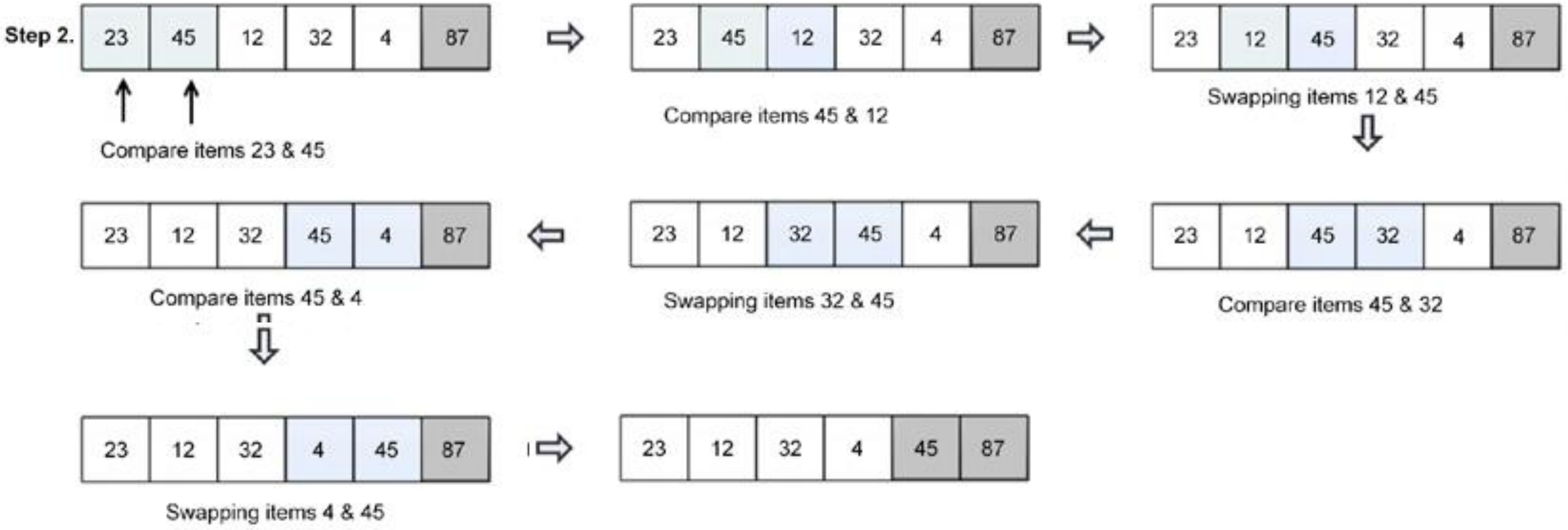
- Sau lần lặp đầu tiên, số lớn nhất 87 được đặt ở cuối danh sách:



©CMC Univ. 2022

Ví dụ

- Sau lần lặp đầu tiên, ta chỉ cần sắp xếp n-1 phần tử còn lại; lặp lại quá trình tương tự bằng cách so sánh các phần tử liền kề với các phần tử còn lại. Sau lần lặp thứ hai, phần tử lớn thứ hai, 45, được đặt ở vị trí thứ hai từ vị trí cuối cùng trong danh sách:

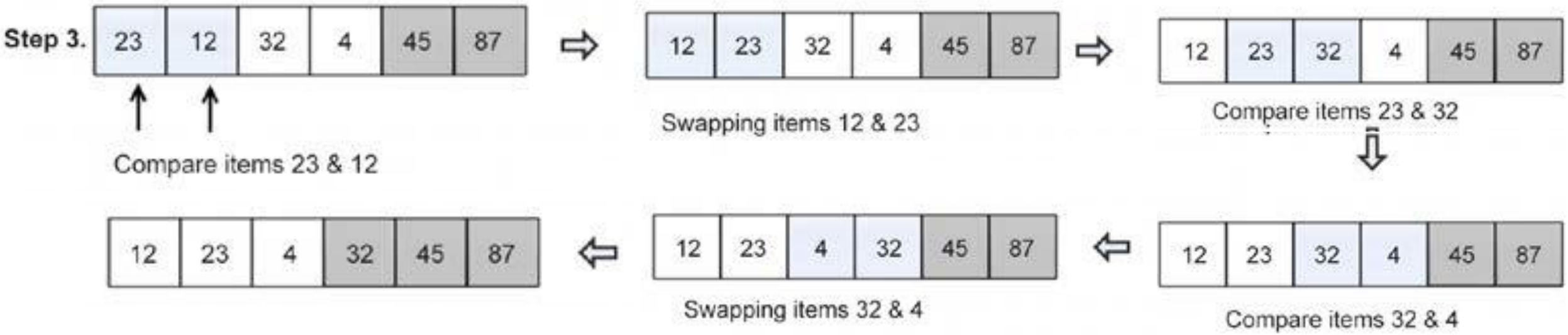


©CMC Univ. 2022

CMC UNIVERSITY

Ví dụ

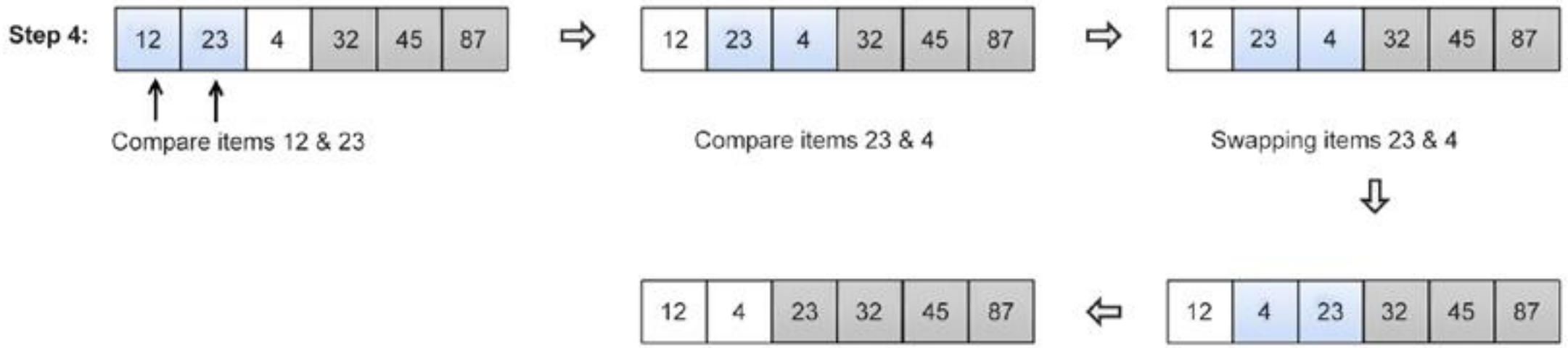
- Tiếp theo, so sánh *(n-2)* phần tử còn lại và sắp xếp chúng:



©CMC Univ. 2022

Ví dụ

Tương tự, sao sánh các phần tử còn lại và sắp xếp chúng:



Đối với hai phần tử cuối cùng còn lại, đặt chúng theo đúng thứ tự để có được danh sách được sắp xếp cuối cùng:

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nổi bọt

```
//Viết bằng C++ dùng mảng

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

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nổi bọt

```
//Viết bằng C++ dùng vector

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;
            }
        }
    }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nổi bọt

```
//Viết bằng Java

private static void bubbleSort(int[] a)
{
      int temp;
      for (int i = 0; i < a.length - 1; i++) {
         for (int j = 0; j < a.length - 1 - i; j++) {
            if (a[j] > a[j + 1]) {
                  temp = a[j];
                  a[j] = a[j + 1];
                  a[j + 1] = temp;
            }
         }
      }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nổi bọt

```
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n;
        do{
            System.out.print("Cho n = ");
            n = input.nextInt();
            if(n <= 0)
                System.out.println("n <= 0. Nhap lai n!");
        }while(n <= 0);
        int[]arr = new int[n];
        System.out.println("Nhap day so:");
        inputArray(arr, n);
        System.out.println("In lai day so:");
        showArray(arr);
        System.out.println("Sap xep theo BubbleSort.");
        bubbleSort(arr);
        System.out.println("Day tang dan:");
        showArray(arr);
}
```

```
Cho n = -2
n <= 0. Nhap lai n!
Cho n = 0
n <= 0. Nhap lai n!
Cho n = 6
Nhap day so:
a[0] = 45
a[1] = 23
a[2] = 87
a[3] = 12
a[4] = 32
a[5] = 4
In lai day so:
45, 23, 87, 12, 32, 4
Sap xep theo BubbleSort.
Day tang dan:
4, 12, 23, 32, 45, 87
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nổi bọt

```
public static void inputArray(int[] arr, int n) {
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; i++){
            System.out.printf("a[%d] = ", i);
            arr[i] = input.nextInt();
        }
}
public static void showArray(int[] arr) {
        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
}
```

Cho n = -2
n <= 0. Nhap lai n!
Cho n = 0
n <= 0. Nhap lai n!
Cho n = 6
Nhap day so:
a[0] = 45
a[1] = 23
a[2] = 87
a[3] = 12
a[4] = 32
a[5] = 4
In lai day so:
45, 23, 87, 12, 32, 4
Sap xep theo BubbleSort.
Day tang dan:
4, 12, 23, 32, 45, 87

©CMC Univ. 2022

CMC UNIVERSITY

Đánh giá

Trong trường hợp tồi nhất, số phép so sánh trong lần lặp đầu tiên là (*n-1*), lần lặp thứ 2 có số phép so sánh là (*n-2*), và lần lặp thứ 3 sẽ là (*n-3*), …. Do vậy, tổng số phép so sánh là:

$$T(n) = \sum_{i=0}^{n-2}(n-1-i) = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} = O(n^2)$$

- Thuật toán sắp xếp nổi bọt không phải là thuật toán sắp xếp hiệu quả:
  - Độ phức tạp thời gian trong trường hợp tồi nhất là $O(n^2)$, khi dãy đã cho có thứ tự ngược với thứ tự cần sắp xếp,
  - Độ phức tạp trường hợp tốt nhất là $O(n)$, khi dãy đã cho đã được sắp xếp (sẽ không cần hoán đổi vị trí)
- Nói chung, không nên sử dụng thuật toán sắp xếp nổi bọt để sắp xếp các danh sách lớn.. Thuật toán sắp xếp nổi bọt hoạt động tốt trên các danh sách tương đối nhỏ.

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP CHỌN

©CMC Univ. 2022

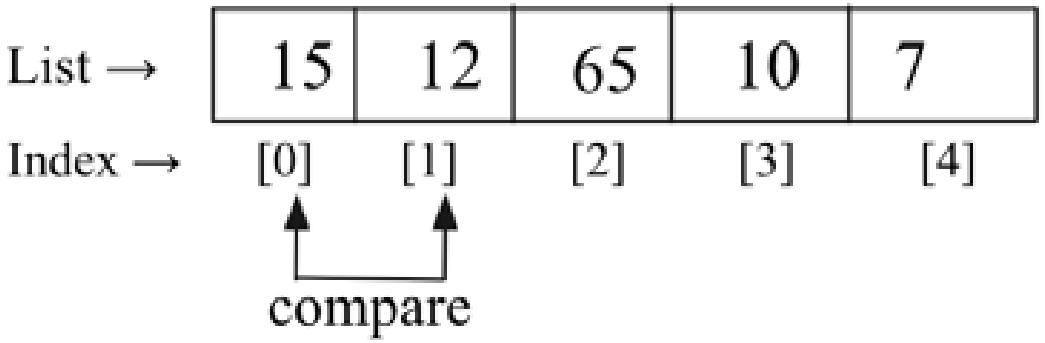
CMC UNIVERSITY

Thuật toán sắp xếp chọn (selection sort)

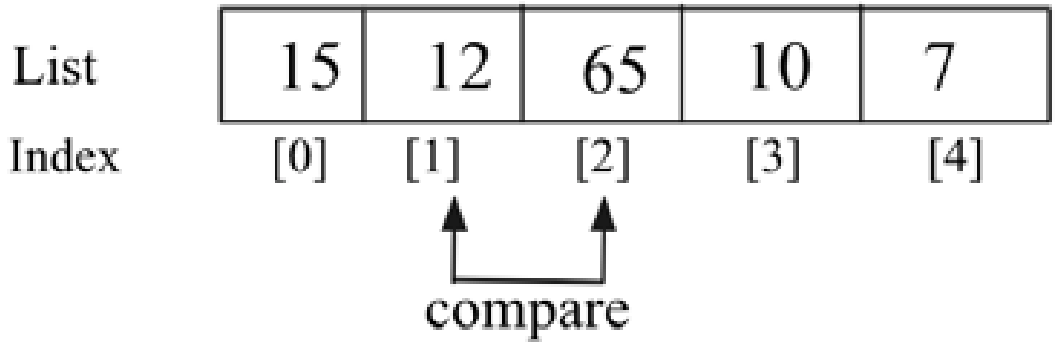
- Ý tưởng: cho dãy gồm n phần tử
  - Thuật toán sắp xếp lựa chọn bắt đầu bằng cách tìm phần tử nhỏ nhất trong danh sách và trao đổi nó với phần tử được lưu trữ ở vị trí đầu tiên trong danh sách. Quá trình này được lặp lại (*n-1*) lần để sắp xếp n phần tử.
  - Tiếp theo, phần tử nhỏ thứ hai, cũng là phần tử nhỏ nhất trong danh sách còn lại, được xác định và hoán đổi với vị trí thứ hai trong danh sách. Điều này làm cho hai phần tử ban đầu được sắp xếp.
  - Quá trình được lặp lại và phần tử nhỏ nhất còn lại trong danh sách được hoán đổi với phần tử trong chỉ mục thứ ba trong danh sách. Điều này có nghĩa là ba phần tử đầu tiên đã được sắp xếp.
- ***Ví dụ***: Sắp xếp dãy gồm các phần tử 15, 12, 65, 10, 7

©CMC Univ. 2022

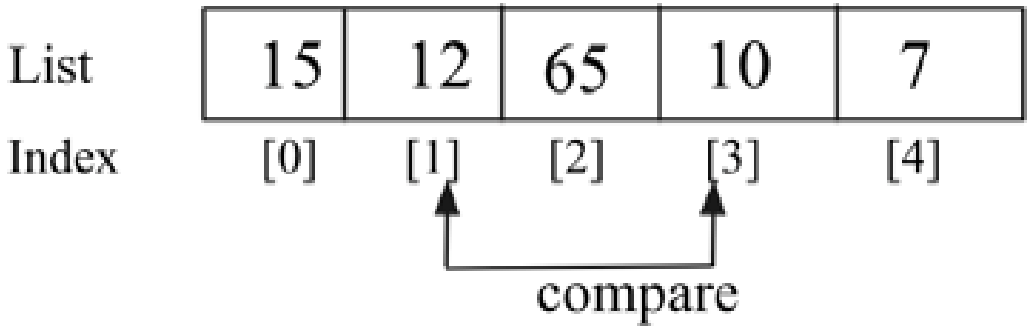
Ví dụ



Min value is at index 1



Min value is at index 1

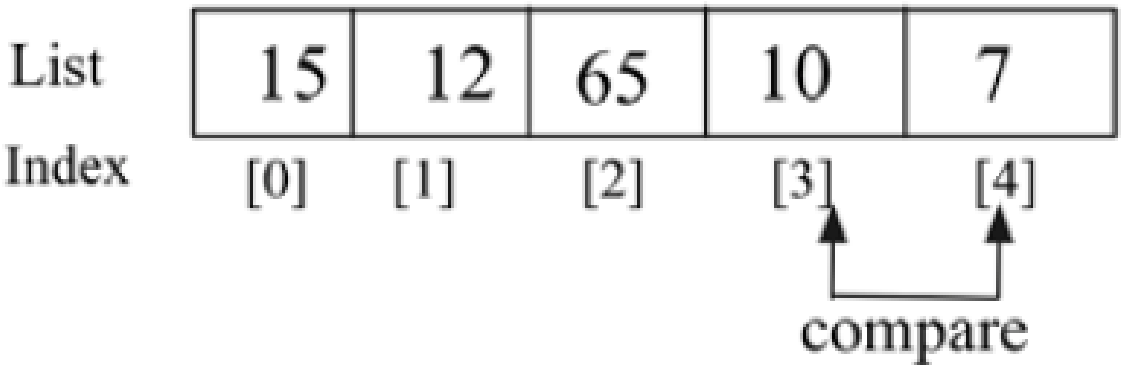


Min value is at index 3

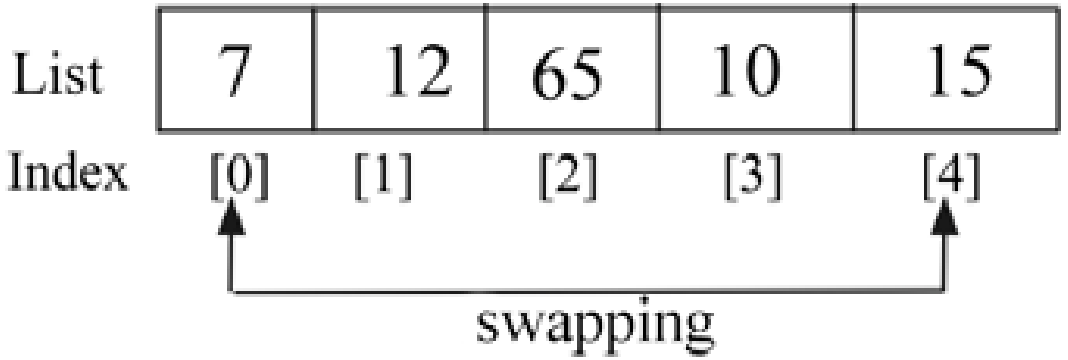
©CMC Univ. 2022

CMC UNIVERSITY

Example



Min value is at index 4

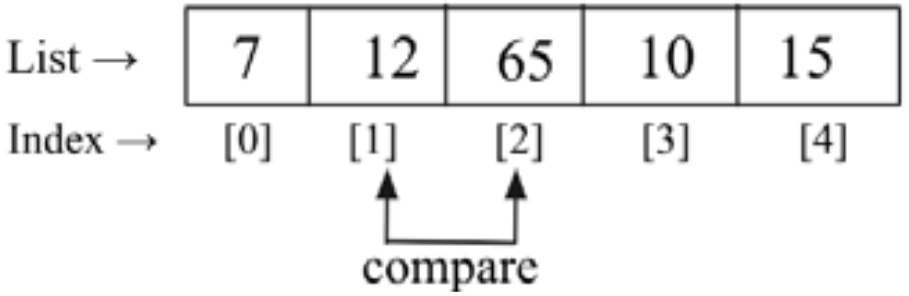


Swapping the min value with the first element

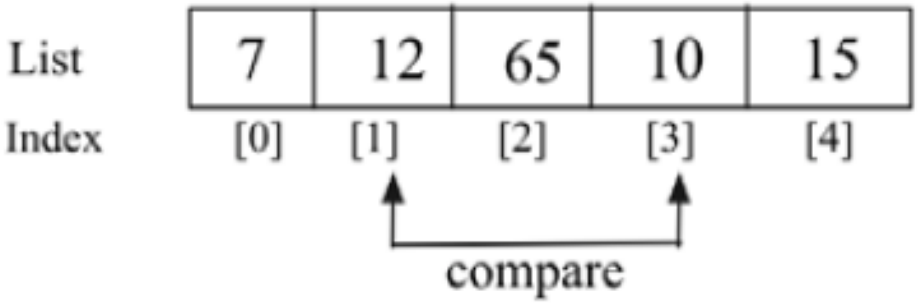
©CMC Univ. 2022

CMC UNIVERSITY

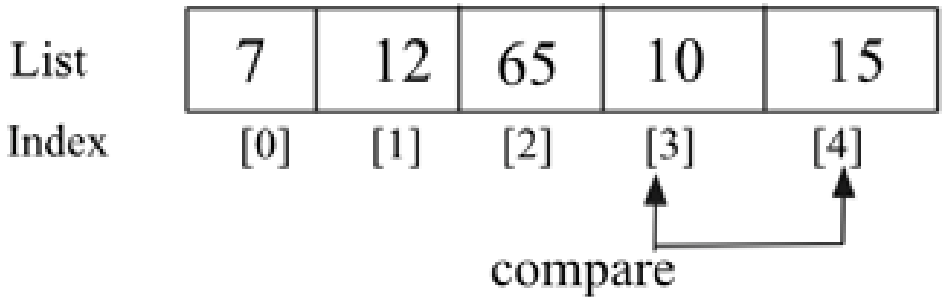
Example



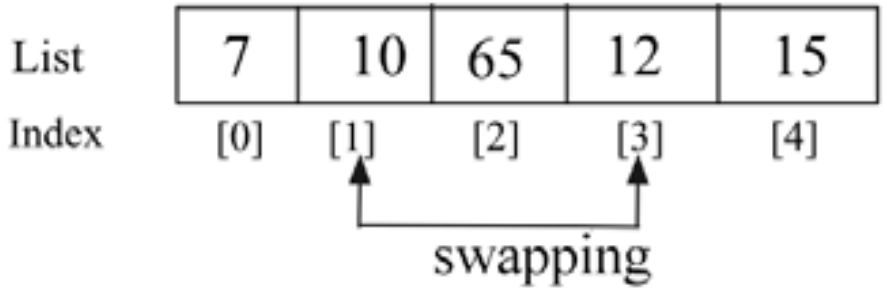
Min value is at index 1



Min value is at index 3



Min value is at index 3



Swapping the min value with the second element

©C

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chọn

Dùng lớp vector trong thư viện chuẩn của C++

```
//Viết bằng 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;
        }
    }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chọn

```
//Viết bằng Java
private static void selectionSort(int[] arr) {
	int n = arr.length;

	for (int i = 0; i < n-1; i++)
	{
		// Find the minimum element in unsorted array
		int min_idx = i;
		for (int j = i+1; j < n; j++)
			if (arr[j] < arr[min_idx])
				min_idx = j;
		// Swap the found minimum element with the first element
		int temp = arr[min_idx];
		arr[min_idx] = arr[i];
		arr[i] = temp;
	}
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chọn

```
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n;
        do{
            System.out.print("Cho n = ");
            n = input.nextInt();
            if(n <= 0)
                System.out.println("n <= 0. Nhap lai n!");
        }while(n <= 0);
        int[]arr = new int[n];
        System.out.println("Nhap day so:");
        inputArray(arr, n);
        System.out.println("In lai day so:");
        showArray(arr);
        System.out.println("Sap xep theo Selection Sort.");
        selectionSort(arr);
        System.out.println("Day tang dan:");
        showArray(arr);
}
```

Cho n = 0
n <= 0. Nhap lai n!
Cho n = 5
Nhap day so:
a[0] = 15
a[1] = 12
a[2] = 65
a[3] = 10
a[4] = 7
In lai day so:

15, 12, 65, 10, 7

Sap xep theo Selection Sort.
Day tang dan:

7, 10, 12, 15, 65

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chọn

```
public static void inputArray(int[] arr, int n) {
        Scanner input = new Scanner(System.in);
        for(int i = 0; i < n; i++){
            System.out.printf("a[%d] = ", i);
            arr[i] = input.nextInt();
        }
}
public static void showArray(int[] arr) {
        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
}
```

Cho n = 0
n <= 0. Nhap lai n!
Cho n = 5
Nhap day so:
a[0] = 15
a[1] = 12
a[2] = 65
a[3] = 10
a[4] = 7
In lai day so:

15, 12, 65, 10, 7

Sap xep theo Selection Sort.
Day tang dan:

7, 10, 12, 15, 65

©CMC Univ. 2022

Đánh giá

- Trong thuật toán sắp xếp chọn, (*n-1*) phép so sánh được yêu cầu trong lần lặp đầu tiên, (n-2) phép so sánh cho lần lặp thứ 2, và (*n-3*) phép so sánh cho lần lặp thứ 3, …. Do vậy, tổng số phép so sánh là:

$$T(n) = \sum_{i=0}^{n-2} (n-1-i) = (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}$$

→ $\mathrm{T}(n) = O(n^2)$

- Độ phức tạp thời gian trường hợp tồi nhất là $O(n^2)$, khi danh sách đã cho có thứ tự ngược với thứ tự cần sắp xếp.
- Độ phức tạp thời gian trong trường hợp tốt nhất là $O(n^2)$.
- Thuật toán sắp xếp chọn nên được sử dụng khi danh sách các phần tử cần sắp xếp là nhỏ

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP CHÈN

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp chèn (Insertion Sort)

Sắp xếp dãy có n phần tử:

- Có n - 1 bước ứng với p = 1, 2, …, n - 1.
- Ở bước p:

  (Khi bắt đầu bước p, các vị trí 0, …, p - 1 đã được sắp xếp rồi)

  - Xác định vị trí phù hợp trong các vị trí 0, …, p - 1 cho phần tử a_p đang 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.

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp chèn (insertion sort)

- Ý tưởng: cho danh sách gồm n phần tử

Ta duy trì hai danh sách con (danh sách con là một phần của danh sách lớn hơn ban đầu), một danh sách đã được sắp xếp và một danh sách chưa được sắp xếp. Ta lấy các phần tử từ danh sách con chưa sắp xếp và chèn chúng vào đúng vị trí trong danh sách con đã sắp xếp, sao cho danh sách con này vẫn được sắp xếp.

- Thuật toán:
  - Bắt đầu với một phần tử, sắp xếp nó,
  - Lấy từng phần tử từ danh sách con chưa được sắp xếp và đặt chúng vào đúng vị trí (so với phần tử đầu tiên) trong danh sách con đã sắp xếp.

©CMC Univ. 2022

CMC UNIVERSITY

Ví dụ

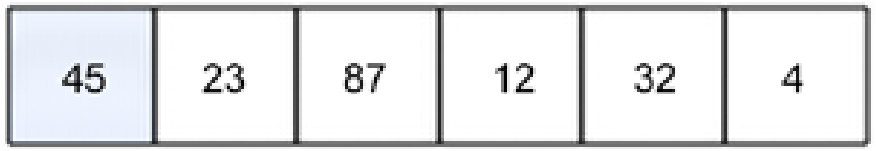
Ví dụ 1: Sắp xếp danh sách {45, 23, 87, 12, 32, 4}

- Bắt đầu với phần tử 45, giả sử nó đã được sắp xếp,
- Sau đó lấy phần tử tiếp theo, 23, từ danh sách con chưa được sắp xếp và chèn nó vào đúng vị trí trong danh sách con đã sắp xếp: 23, 45
- Lần lặp tiếp theo, lấy phần tử thứ ba, 87, từ danh sách con chưa được sắp xếp, và chèn nó vào đúng vị trí trong danh sách con đã sắp xếp: 23, 45, 87
- Tiếp theo lấy 12 từ danh sách con chưa được sắp xếp và chèn vào đúng vị trí trong danh sách con đã sắp xếp: 12, 23, 45, 87
- Tiếp theo lấy 32 và chèn vào đúng vị trí: 12, 23, 32, 45, 87
- Lấy phần tử cuối cùng, 4, từ danh sách con chưa sắp xếp và chèn đúng vị trí trong danh sách con đã được sắp xếp: 4, 12, 23, 32, 45, 87

©CMC Univ. 2022

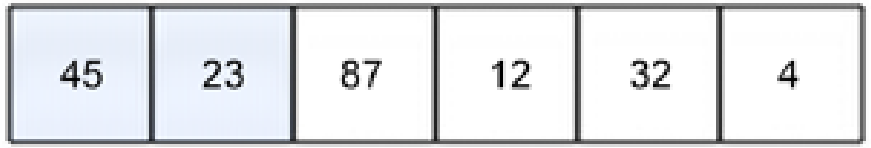
Ví dụ

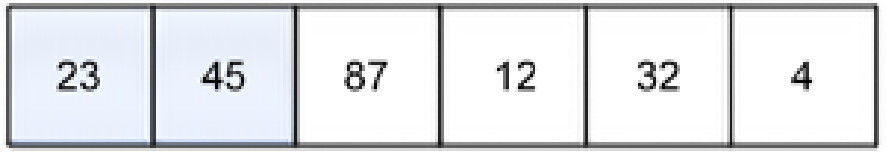
Step 1.



Sublist 1 is sorted.

Step 2.



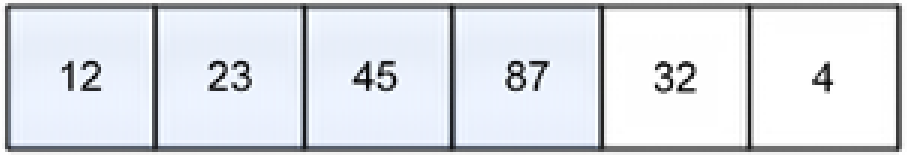


Insert 23 at correct position in sub-list 1.

Step 3.

Insert 87 in correct position in sorted sub-list.

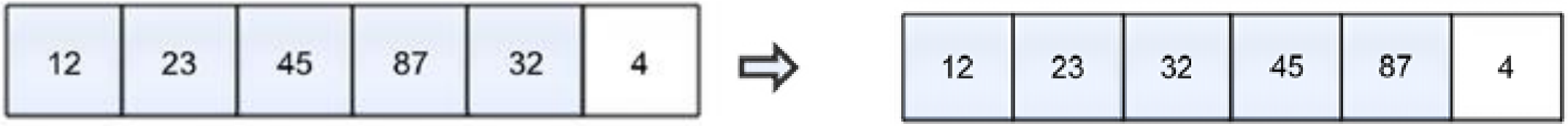
Step 4.



Insert 87 in correct position in sorted sub-list.

Ví dụ

Step 5.



Insert 32 in correct position in sorted sub-list.

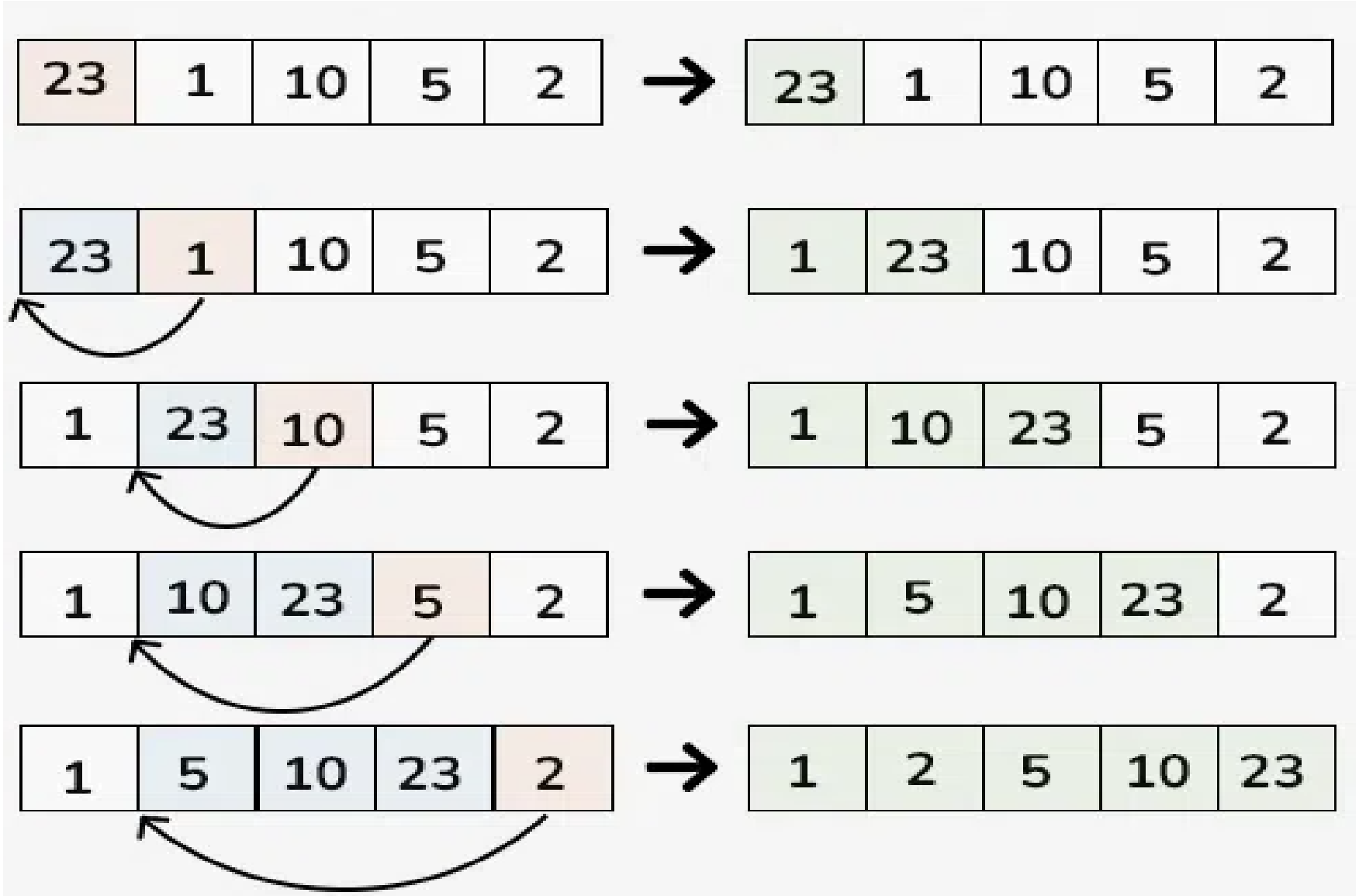
Step 6.

Insert 4 in correct position in sorted sub-list.

©CMC Univ. 2022

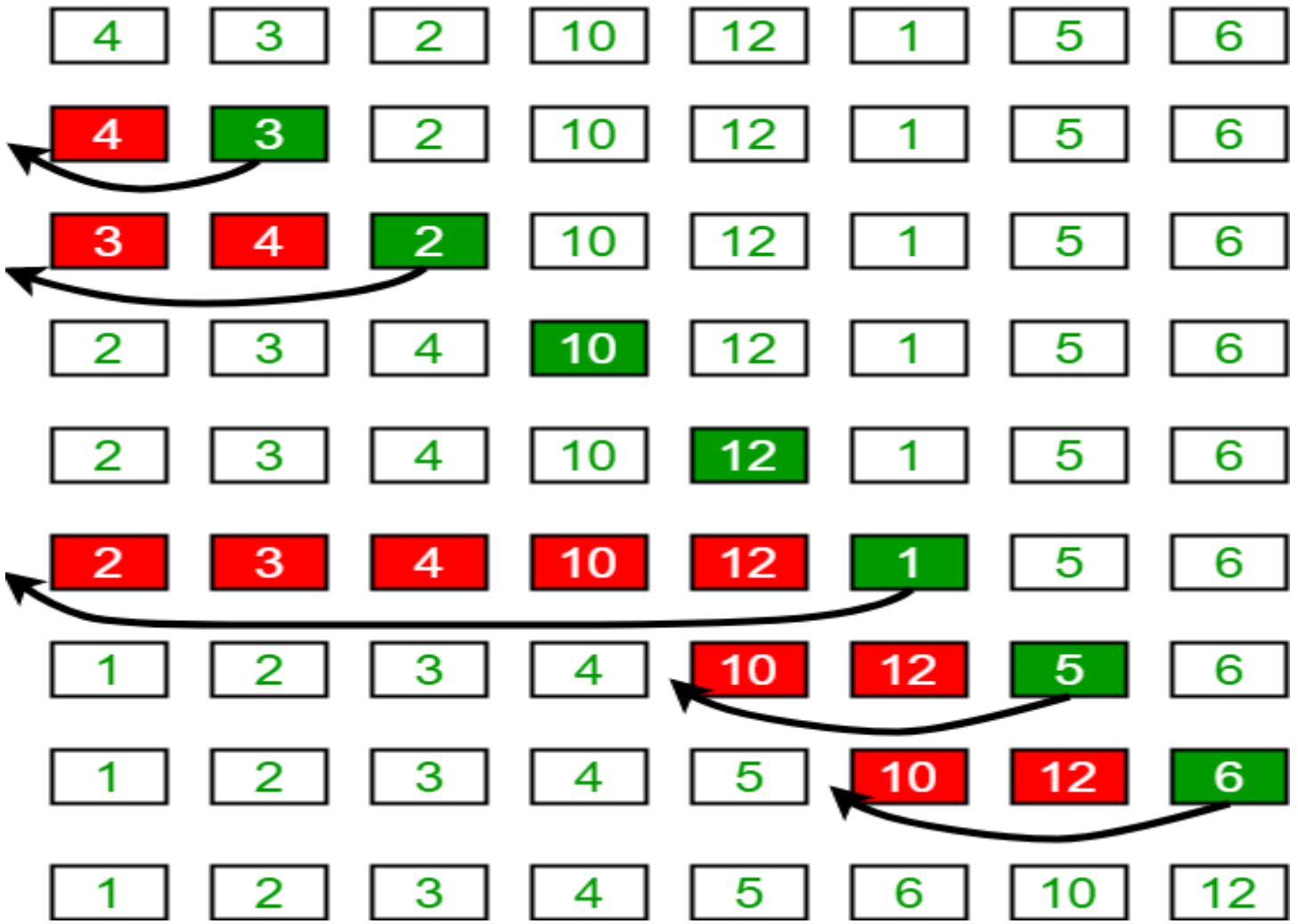
Ví dụ

Ví dụ 2: Sắp xếp danh sách {23, 1, 10, 5, 2}



©CMC Univ. 2022

Ví dụ



©CMC Univ. 2022

Ví dụ

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

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chèn

```
//Viết bằng C++
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
    }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp chèn

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

©CMC Univ. 2022

Đánh giá

Số phép so sánh lần lặp đầu tiên là 1, lần lặp thứ hai là 2, … lần lặp thứ n-1 là n-1.

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

- Độ phức tạp về thời gian trong trường hợp tồi nhất của thuật toán sắp xếp chèn là $O(n^2)$, khi danh sách các phần tử đã cho được sắp xếp theo thứ tự ngược lại. Trong trường hợp đó, mỗi phần tử sẽ phải được so sánh với từng phần tử khác.
- Độ phức tạp trong trường hợp tốt nhất của thuật toán sắp xếp chèn là $O(n)$, khi danh sách đầu vào đã được sắp xếp, trong đó mỗi phần tử từ danh sách con chưa được sắp xếp chỉ được so sánh với phần tử ngoài cùng bên phải của danh sách con được sắp xếp trong mỗi lần lặp
- Thuật toán sắp xếp chèn rất tốt để sử dụng khi danh sách đã cho có số lượng phần tử nhỏ và phù hợp nhất khi dữ liệu đầu vào được đưa vào từng phần tử một.

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP TRỘN

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp trộn (merge sort)

Thuật toán sắp xếp trộn là thuật toán sắp xếp được phát triển dựa trên chia để trị, bao gồm các thao tác sau:

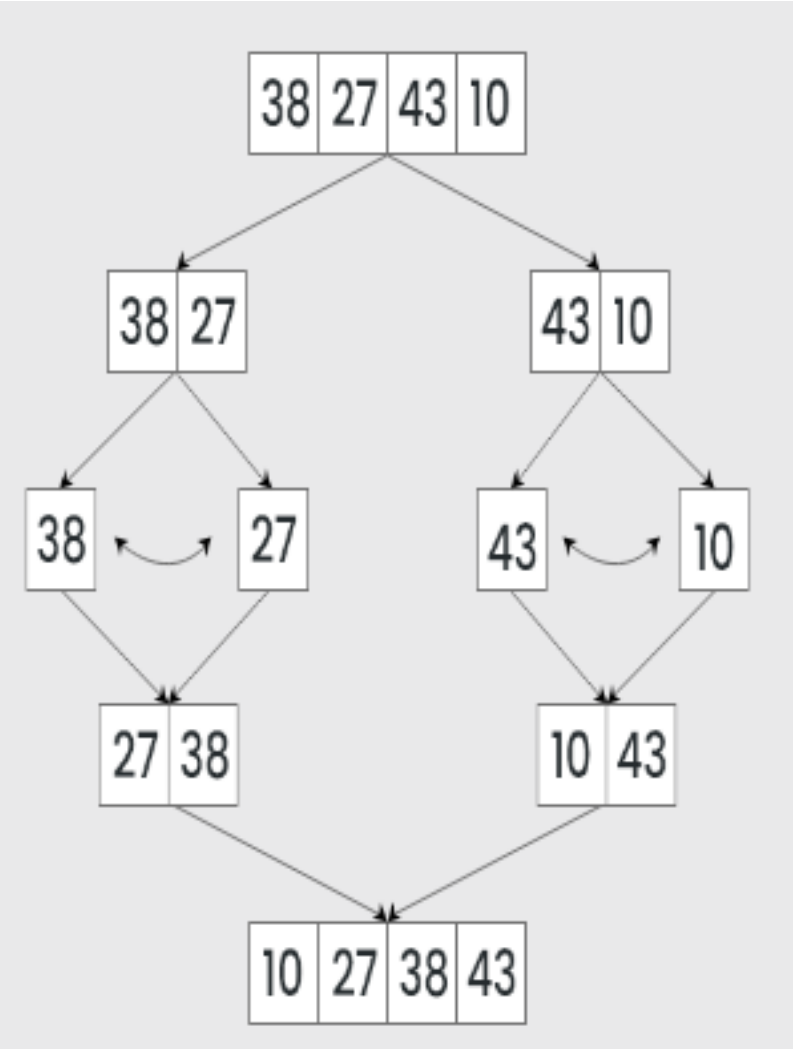
- ***Chia*** (Divide): Chia dãy gồm n phần tử cần sắp xếp thành hai dãy, mỗi dãy có n/2 phần tử
- ***Trị*** (Conquer):
  - Sắp xếp mỗi dãy con một cách đệ quy sử dụng sắp xếp trộn
  - Khi dãy chỉ còn một phần tử thì trả lại phần tử này
- ***Kết hợp*** (combine): Trộn (merge) hai dãy con được sắp xếp để thu được một dãy được sắp xếp gồm tất cả các phần tử của cả hai dãy con.

©CMC Univ. 2022

Thuật toán sắp xếp trộn

Xét dãy gồm n phần tử.

- Nếu n = 1: dãy đã được sắp xếp
- 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.



©CMC Univ. 2022

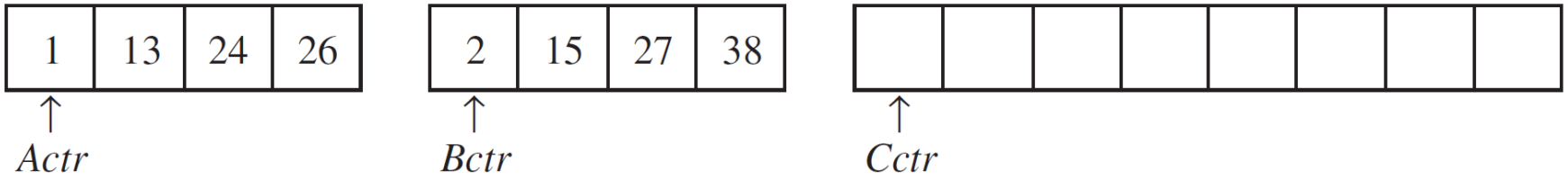
CMC UNIVERSITY

Thao tác trộn (1)

Input: Hai dãy A và B đã sắp xếp.

Output: 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.
- Tại 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.

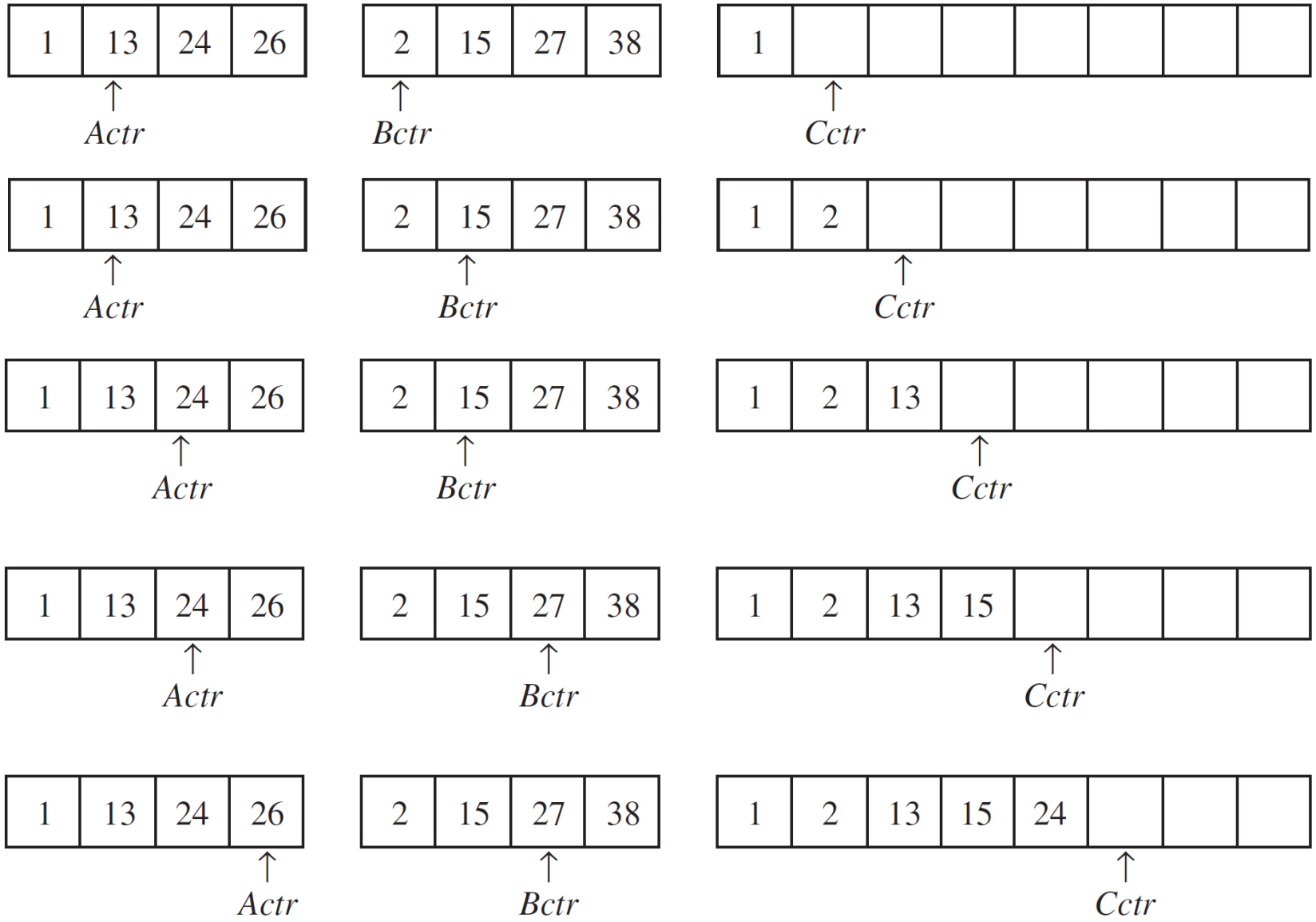


(Sẽ sao chép 1, tăng *Actr* và *Cctr*)

©CMC Univ. 2022

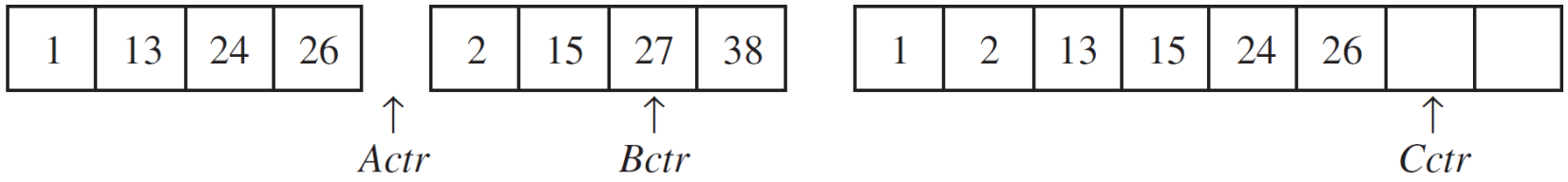
CMC UNIVERSITY

Thao tác trộn (2)

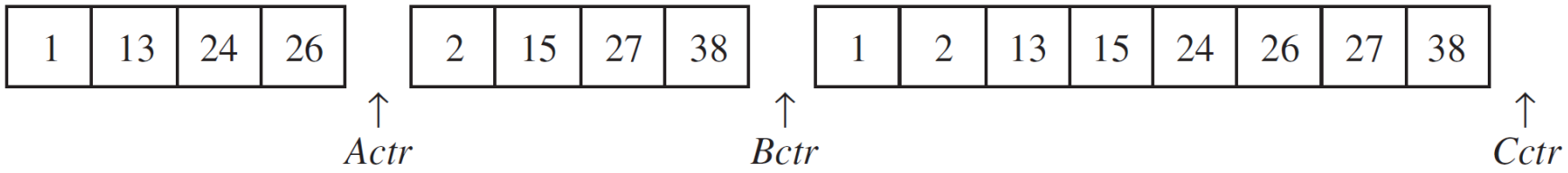


©CMC Univ. 2022

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 số để sao chép một phần tử từ A hoặc B sang C.

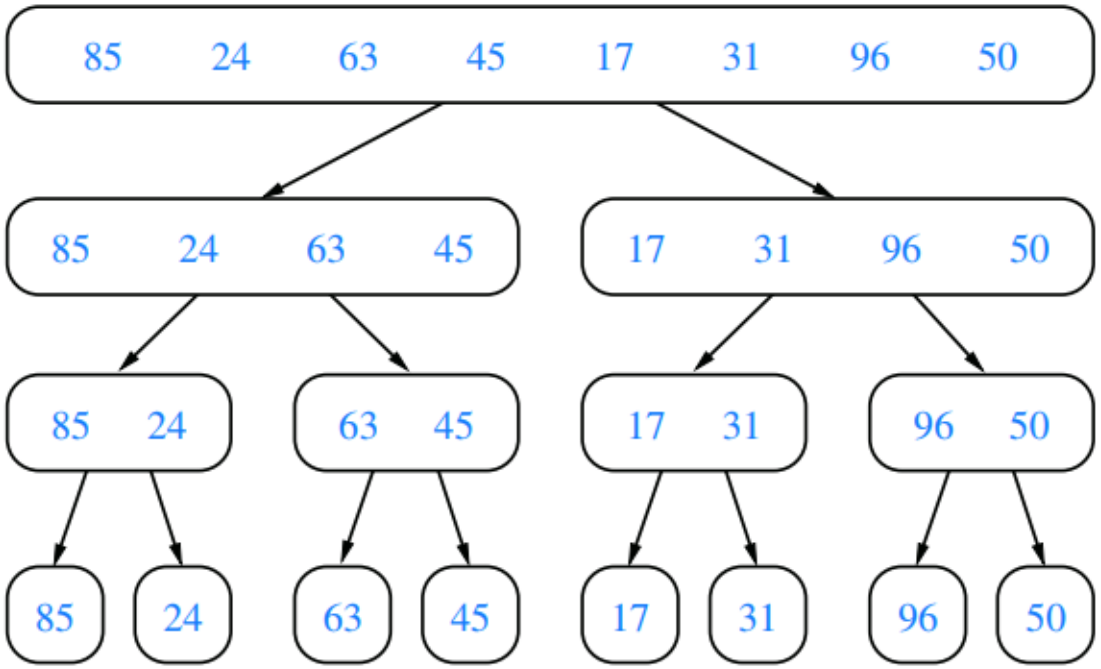
⇒ Thời gian chạy của thao tác trộn là O(n).

©CMC Univ. 2022

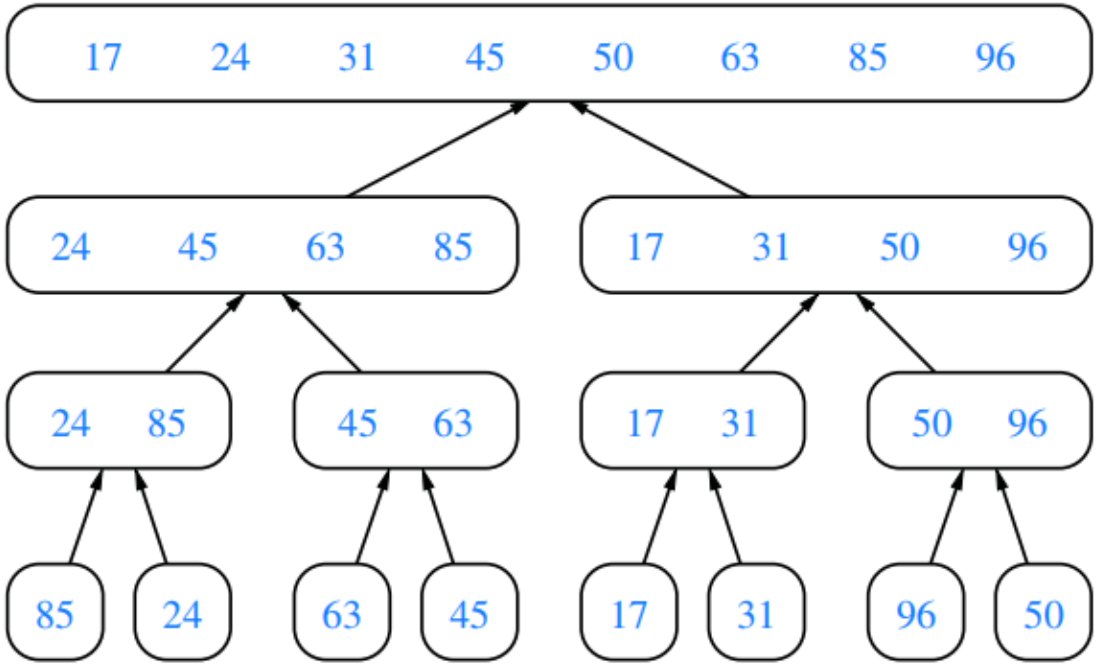
CMC UNIVERSITY

Ví dụ

Minh họa thuật toán Merge Sort cho dãy: {85, 24, 63, 45, 17, 31, 96, 50}



a)



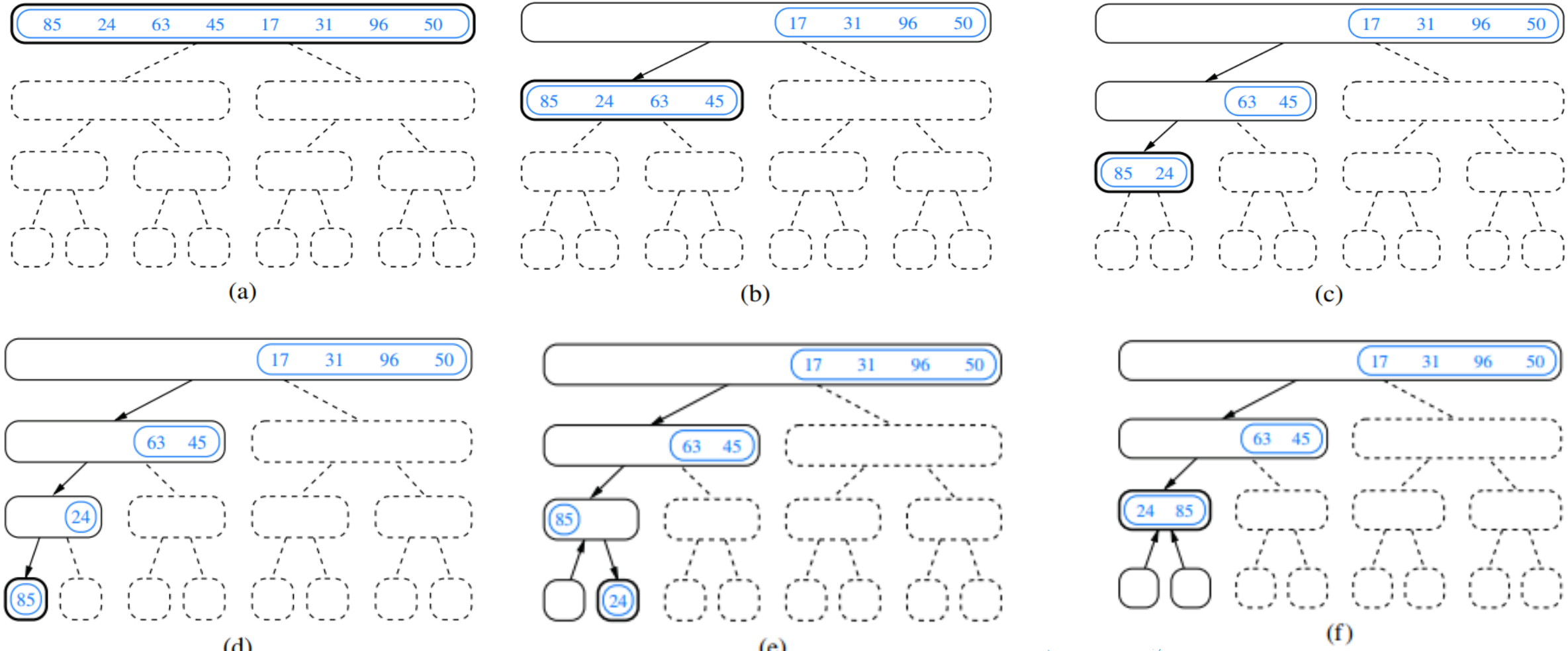
b)

©CMC Univ. 2022

CMC UNIVERSITY

Ví dụ

Minh họa thuật toán Merge Sort cho dãy: {85, 24, 63, 45, 17, 31, 96, 50}

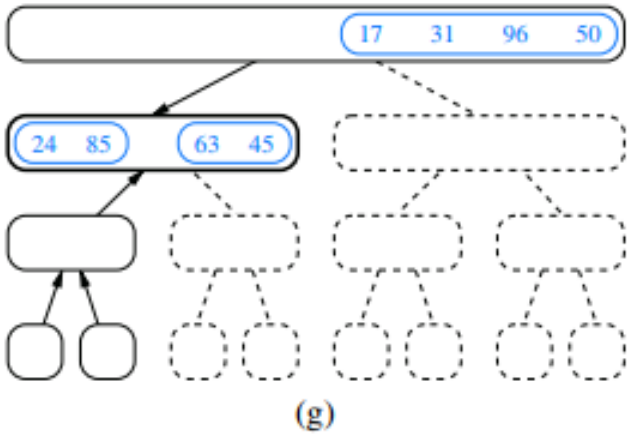


©CMC Univ. 2022

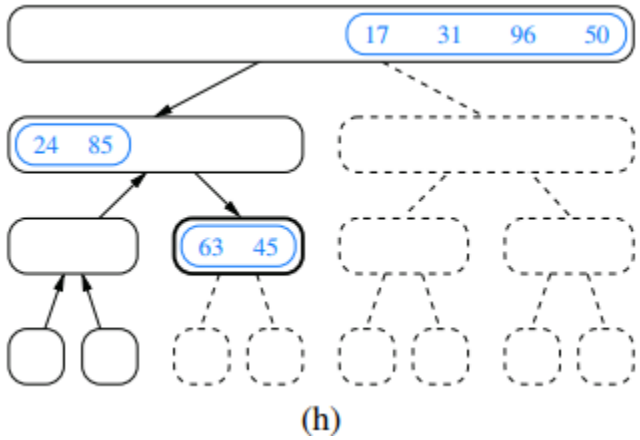
CMC UNIVERSITY

Ví dụ

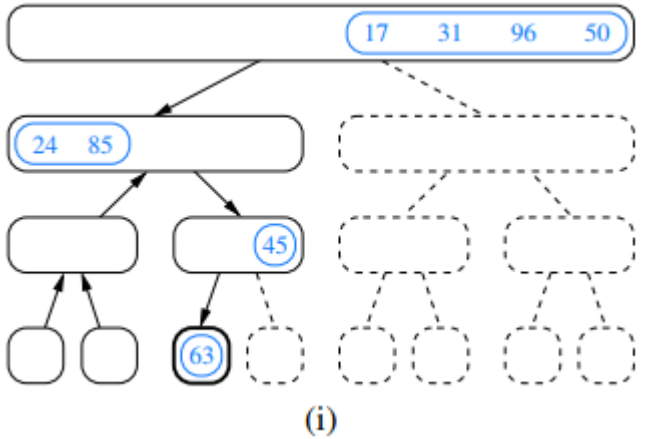
Minh họa thuật toán Merge Sort cho dãy: {85, 24, 63, 45, 17, 31, 96, 50}



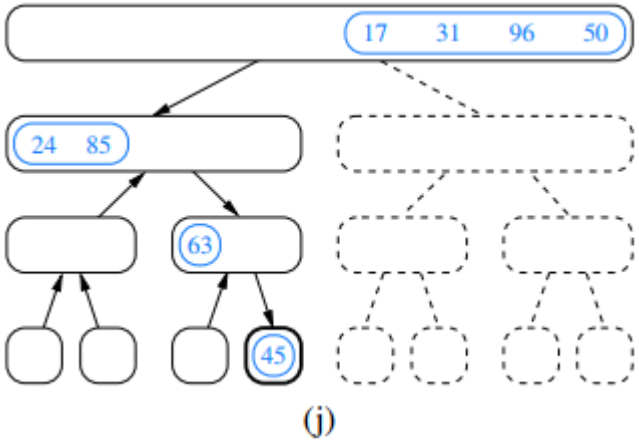
(g)



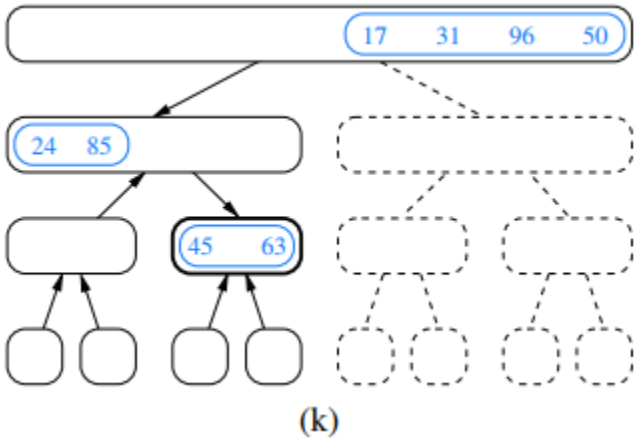
(h)



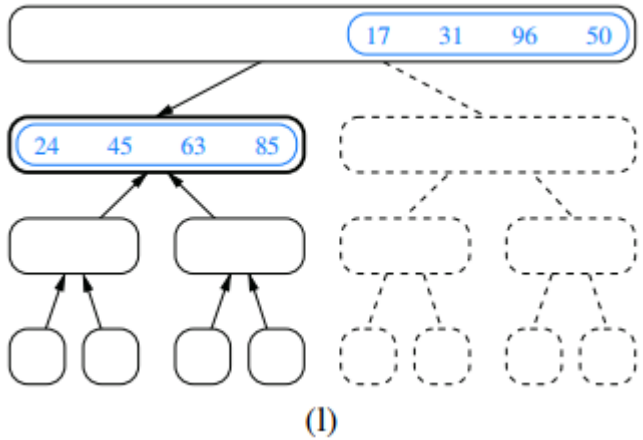
(i)



(j)



(k)



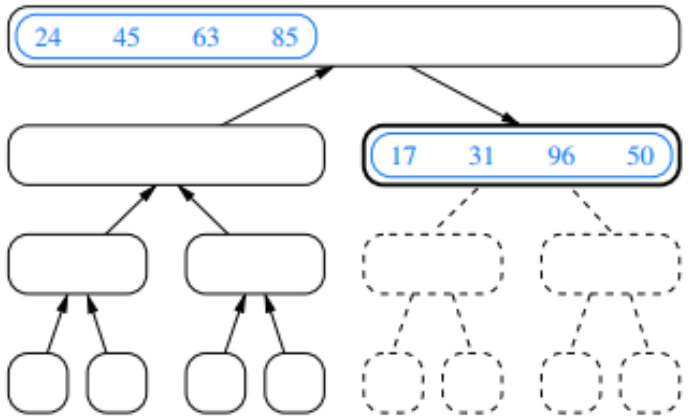
(l)

©CMC Univ. 2022

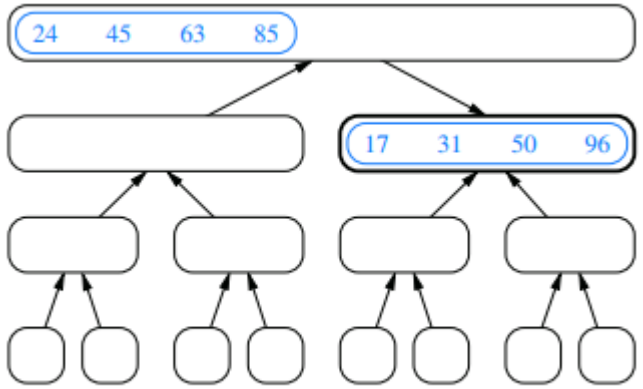
CMC UNIVERSITY

Ví dụ

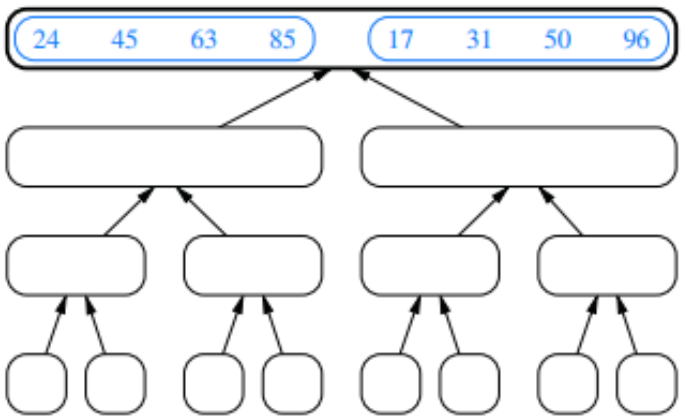
Minh họa thuật toán Merge Sort cho dãy: {85, 24, 63, 45, 17, 31, 96, 50}



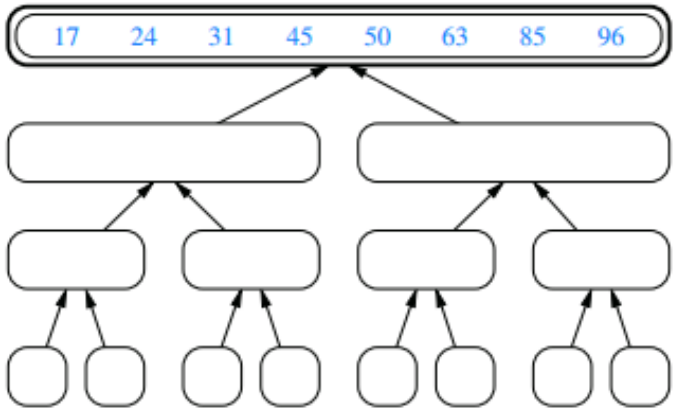
(m)



(n)



(o)



(p)

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp trộn

```
private static void mergeSort(int arr[], int l, int r)
{
        if (l < r) {

                // Find the middle point
                int m = (l + r)/2;

                // Sort first and second halves
                mergeSort(arr, l, m);
                mergeSort(arr, m + 1, r);

                // Merge the sorted halves
                merge(arr, l, m, r);
        }
}
```

©CMC Univ. 2022

Cài đặt thuật toán sắp xếp trộn

```
private static void merge(int arr[], int l, int m, int r)
{
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;
        // Create temp arrays
        int L[] = new int[n1];
        int R[] = new int[n2];
        // Copy data to temp arrays
        for (int i = 0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[m + 1 + j];
        // Initial indices of first and second subarrays
        int i = 0, j = 0;
```

©CMC Univ. 2022

Cài đặt thuật toán sắp xếp trộn

```
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i]; i++;
            }
            else {
                arr[k] = R[j]; j++;
            }
            k++;
        }
        while (i < n1) {
            arr[k] = L[i]; i++; k++;
        }
        while (j < n2) {
            arr[k] = R[j]; j++; k++;
        }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp trộn

```
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Cho n = ");
        int n = input.nextInt();

        int[]arr = new int[n];
        System.out.println("Nhap day so:");
        inputArray(arr, n);
        System.out.println("In lai day so:");
        showArray(arr);
        System.out.println("Sap xep theo Merge Sort.");
        mergeSort(arr,0,arr.length-1);
        System.out.println("Day sap tang dan:");
        showArray(arr);
}
```

```
Cho n = 8
Nhap day so:
a[0] = 85
a[1] = 24
a[2] = 63
a[3] = 45
a[4] = 17
a[5] = 31
a[6] = 96
a[7] = 50
In lai day so:
85 24 63 45 17 31 96 50
Sap xep theo Merge Sort.
Day sap tang dan:
17 24 31 45 50 63 85 96
```

©CMC Univ. 2022

CMC UNIVERSITY

Đánh giá

Đánh giá độ phức tạp thời gian dựa vào phép chuyển chỗ. Vì phương thức merge, số lượng phép chuyển chỗ thường vượt phép so sánh.

- Ở mỗi lượt sắp xếp thì toàn bộ các khóa được chuyển sang mảng mới, như vậy chi phí cho một lượt là O(n).
- Số lượt gọi merge là $\lceil log_2 n \rceil$, vì
  - Ở lượt 1, kích thước của mảng con là 2^0
  - Ở lượt i, kích thước của mảng con là 2^{i-1}
  - Sau lượt gọi cuối cùng thì mảng con có kích thước n

Độ phức tạp: O(nlogn)

©CMC Univ. 2022

CMC UNIVERSITY

Đánh giá

- Ưu điểm:
  - Tính ổn định, có nghĩa là nó duy trì thứ tự tương đối của các phần tử bằng nhau trong mảng đầu vào
  - Hiệu suất được được đảm bảo cả trong trường hợp tồi nhất, O(nlog n)
  - Cách thực hiện đơn giản: cách tiếp cận chia để trị đơn giản
  - Phương pháp này thường được sử dụng khi sắp xếp ngoài, đối với các tệp
- Nhược điểm:
  - Độ phức tạp không gian: Thuật toán sắp xếp trộn đòi hỏi bộ nhớ bổ sung để lưu các mảng con trong quá trình sắp xếp, và yêu cầu đến 2n phần tử nhớ, gấp đôi so với phương pháp thông thường
  - Không phải là thuật toán sắp xếp in-place (tại chỗ)

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP NHANH

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp nhanh (quick sort)

Sắp xếp nhanh (Quick Sort) hay sắp xếp phân đoạn (Partition) là thuật toán sắp xếp dựa trên kỹ thuật chia để trị.

Cụ thể ý tưởng: chọn một điểm làm chốt (gọi là pivot), sắp xếp mọi phần tử bên trái chốt đều nhỏ hơn chốt và mọi phần tử bên phải đều lớn hơn chốt, sau khi xong ta được 2 dãy con bên trái và bên phải, áp dụng tương tự cách sắp xếp này cho 2 dãy con vừa tìm được cho đến khi dãy con chỉ còn 1 phần tử.

Áp dụng thuật toán cho mảng như sau:

1. Chọn chốt: chọn một phần tử làm chốt
2. Phân hoạch: Sắp xếp phần tử bên trái nhỏ hơn chốt, sắp xếp phần tử bên phải lớn hơn chốt. Cuối cùng, đặt pivot vào vị trí đúng của nó trong dãy đã sắp xếp
3. Đệ qui: Áp dụng quicksort sắp xếp hai mảng con bên trái và bên phải chốt

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp nhanh

- Phần tử được chọn làm chốt rất quan trọng, nó quyết định thời gian thực thi của thuật toán.
- Phần tử được chọn làm chốt tối ưu nhất là phần tử trung vị, phần tử này làm cho số phần tử nhỏ hơn trong dãy bằng hoặc xấp xỉ số phần tử lớn hơn trong dãy. Tuy nhiên, việc tìm phần tử chốt này rất tốn kém, phải có thuật toán tìm riêng, từ đó làm giảm hiệu suất của thuật toán tìm kiếm nhanh.
- Do đó, để đơn giản, người ta có thể chọn:
  - ✓ sử dụng phần tử đầu tiên làm chốt.
  - ✓ sử dụng phần tử cuối cùng làm chốt.
  - ✓ sử dụng phần tử ngẫu nhiên làm chốt.
  - ✓ sử dụng phần tử ở giữa làm chốt.

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp nhanh

Thuật toán sắp xếp dãy A[low, high] và chọn phần tử cuối làm chốt:

1. Chọn pivot: Lấy phần tử cuối cùng của dãy làm pivot.
2. Phân hoạch (Partition): Duyệt qua các phần tử của dãy và hoán đổi các phần tử nhỏ hơn pivot về phía trước. Cụ thể:
   - Khởi tạo một chỉ số i đại diện cho vị trí cuối cùng của phần tử nhỏ hơn pivot, bắt đầu từ low - 1.
   - Duyệt qua các phần tử từ low đến high - 1. Nếu phần tử hiện tại nhỏ hơn pivot, tăng i lên 1 và hoán đổi phần tử tại i với phần tử hiện tại.
   - Sau khi duyệt xong, hoán đổi pivot (phần tử cuối) với phần tử tại i + 1 để đưa pivot về đúng vị trí của nó..
3. Đệ quy (Recursion): Gọi đệ quy hàm Quick Sort cho dãy con bên trái và bên phải của pivot. Quá trình này tiếp tục cho đến khi các dãy con có kích thước bằng 1 hoặc 0, khi đó chúng đã được sắp xếp

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp nhanh

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.
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 { quickSort(A_1), {v}, quickSort(A_2) }

Ví dụ 1:

Nếu quickSort(A_1) = {1, 3}, v = 4, quickSort(A_2) = {6, 8, 9}

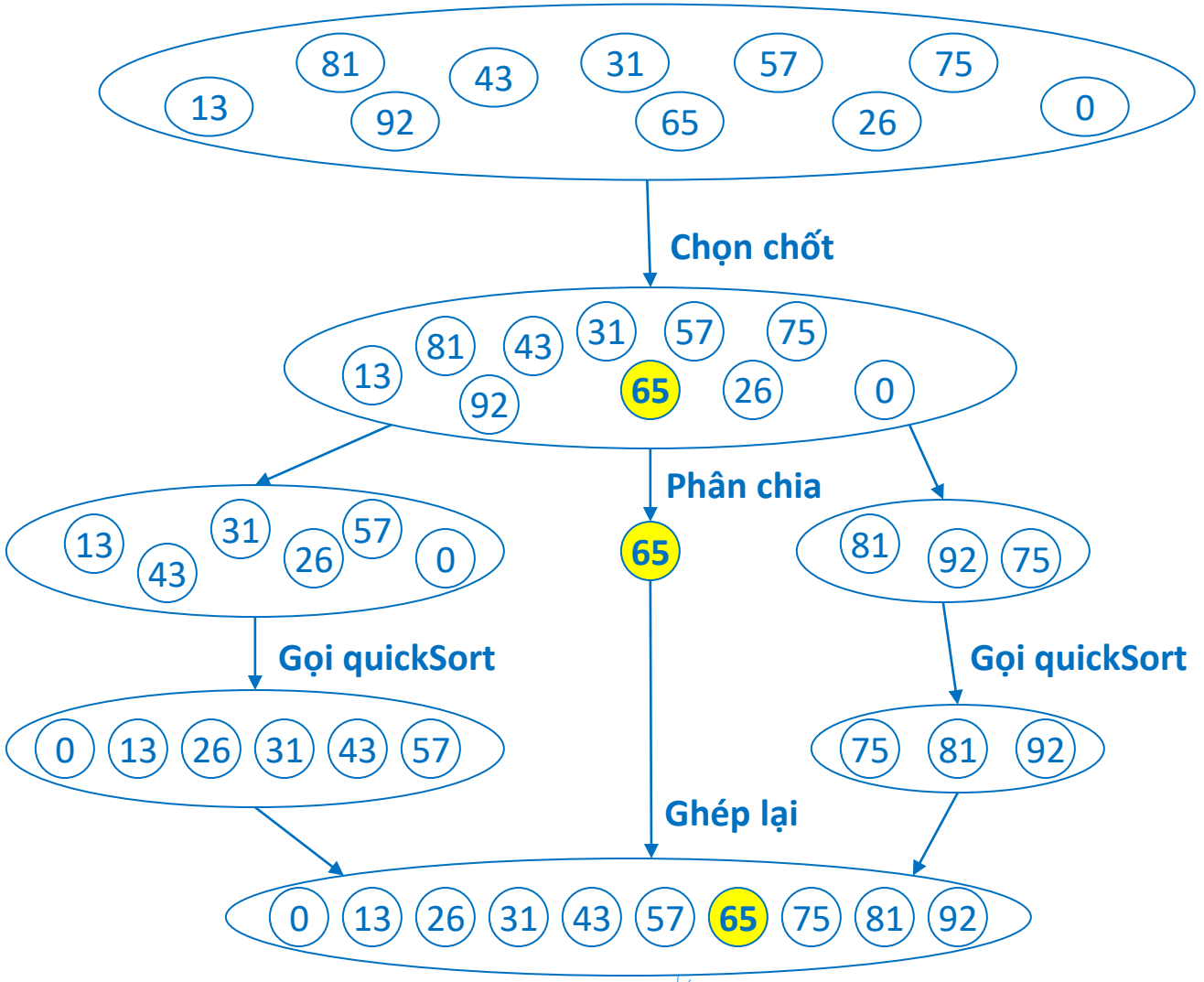
thì trả về {1, 3, 4, 6, 8, 9}.

©CMC Univ. 2022

CMC UNIVERSITY

Ví dụ

Ví dụ 2: Sắp xếp dãy gồm các phần tử: 13, 81, 92, 43, 31, 65, 57, 26, 75, 0

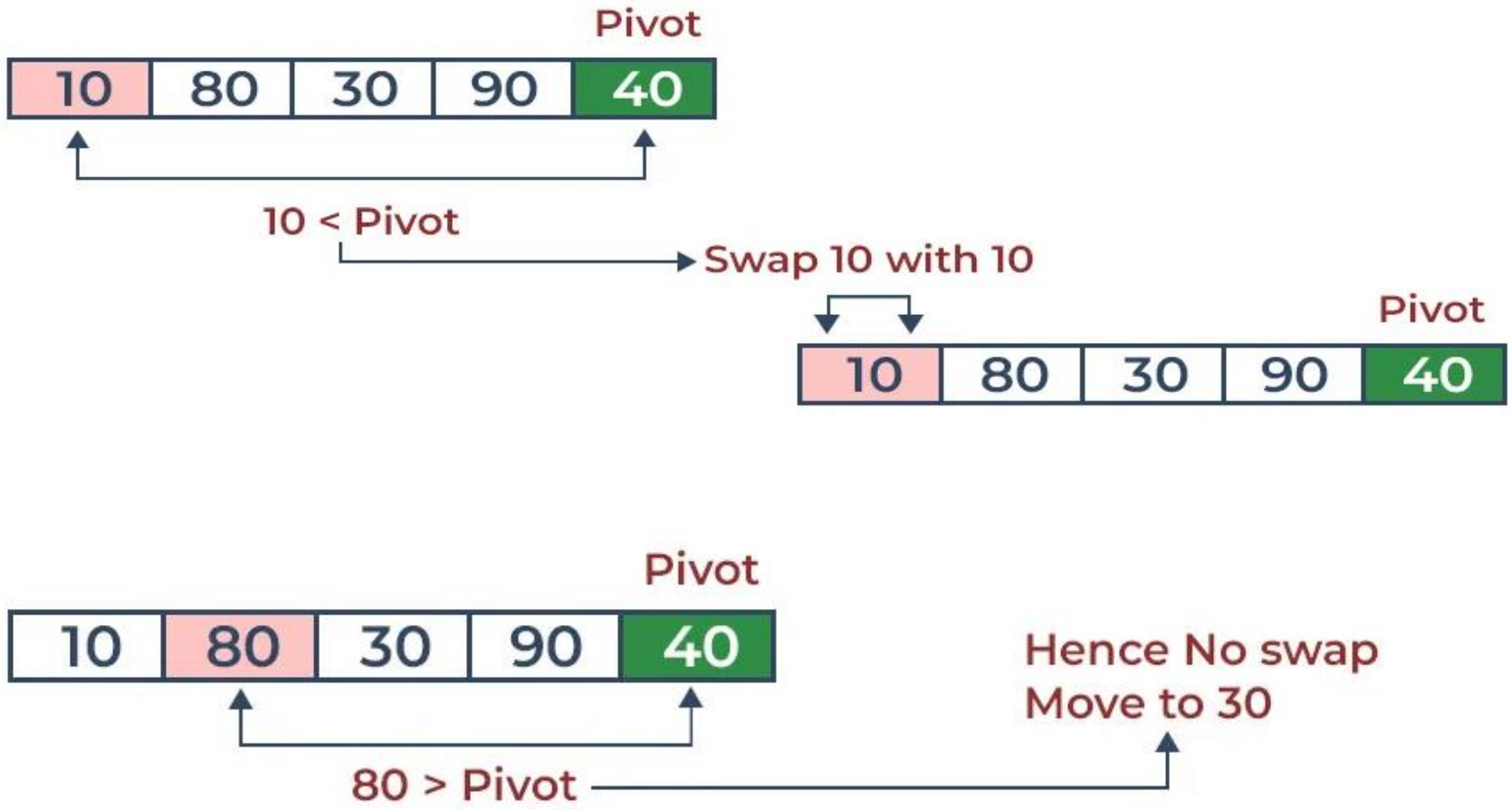


©CMC Univ. 2022

Ví dụ

Ví dụ 3: Sắp xếp dãy gồm 5 phần tử: {10, 80, 30, 90, 40}

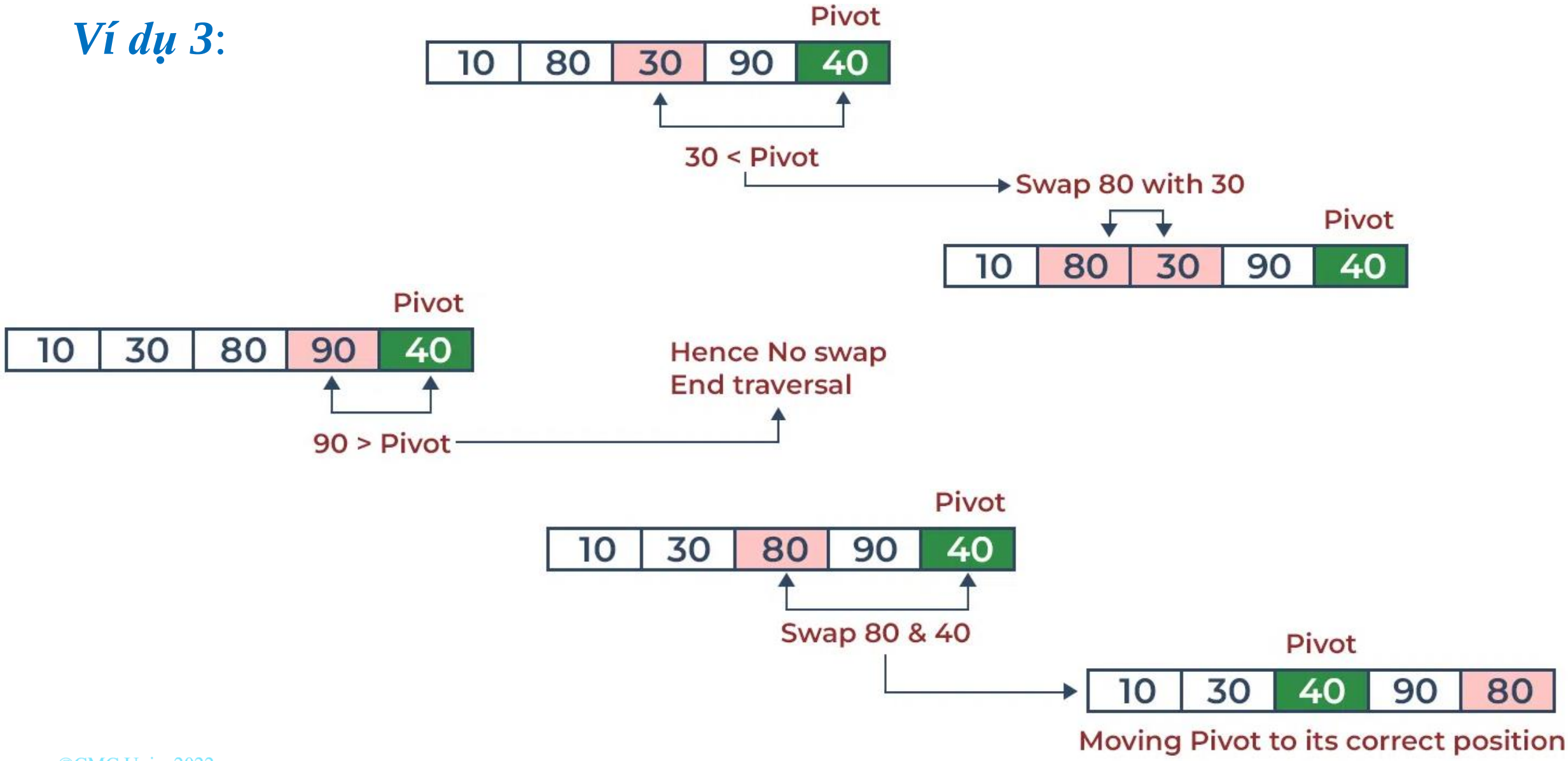
Giả sử ban đầu chọn chốt là phần tử cuối cùng 40



©CMC Univ. 2022

Ví dụ

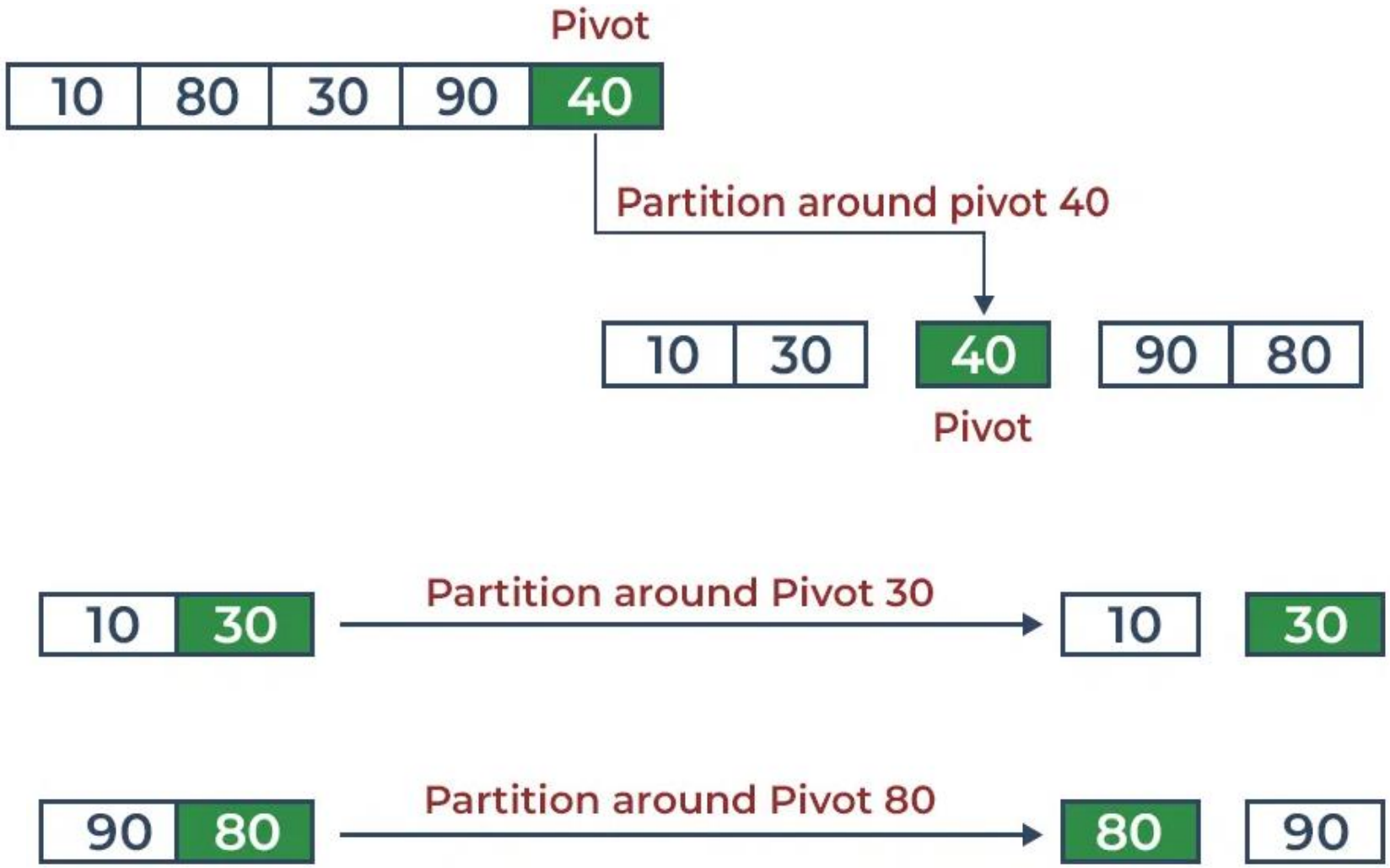
Ví dụ 3:



©CMC Univ. 2022

Ví dụ

***Ví dụ 3*:**



©CMC Univ. 2022

Ví dụ

Ví dụ 4: Sắp xếp dãy số {45, 23, 87, 12, 72, 4, 54, 32, 52}

j = 0 1 2 3

45	23	87	12	72	4	54	32	52

45 < 52
23 < 52
87 > 52
12 < 52

i = -1 0 1 2 j = 3 4 5

45	23	12	87	72	4	54	32	52

72 > 52
4 < 52

i = 2 3 j = 5 6 7

45	23	12	4	72	87	54	32	52

54 > 52
32 < 52

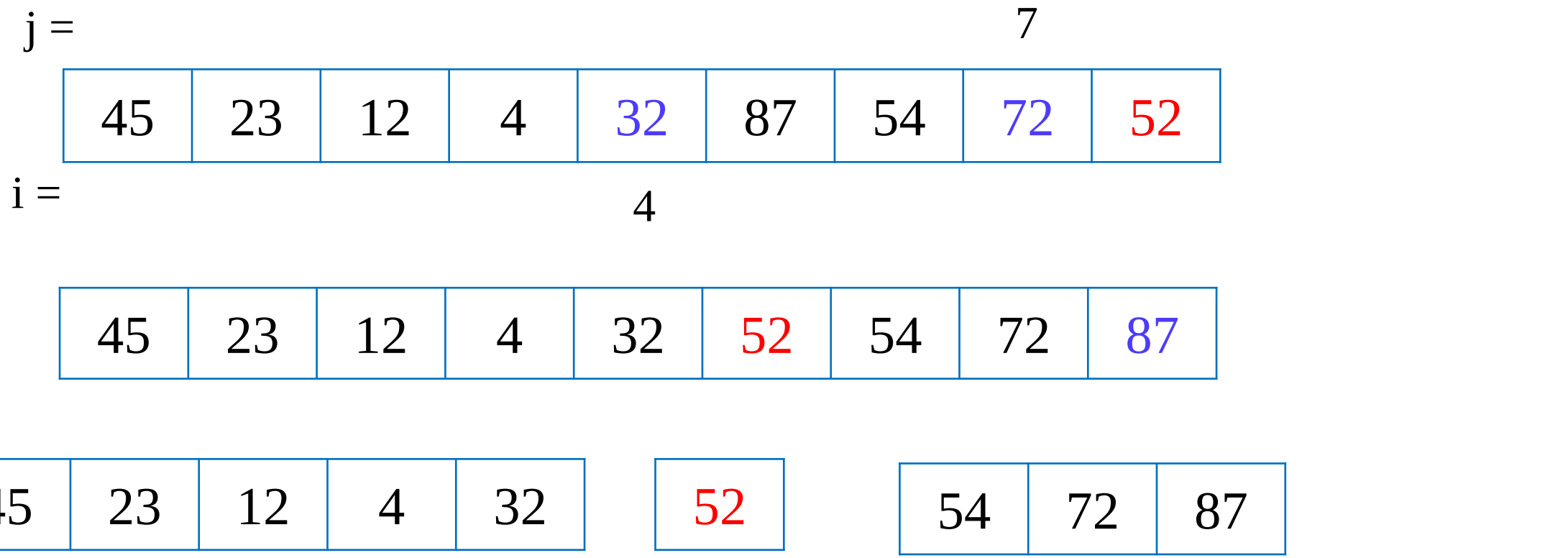
i = 3 4

45	23	12	4	32	87	54	72	52

©CMC Univ. 2022

Ví dụ

***Ví dụ 4*:**



©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nhanh

```
private static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(arr, low, high);
            quickSort(arr, low, pivotIndex - 1);
            quickSort(arr, pivotIndex + 1, high);
        }
}
private static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];   //chọn chốt là phần tử cuối
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j); }
        }
        swap(arr, i + 1, high);
        return i + 1;
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp nhanh (tiếp)

```
public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
}
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Cho n = ");
        int n = input.nextInt();
        int[]arr = new int[n];
        System.out.println("Nhap day so:"); inputArray(arr, n);
        System.out.println("In lai day so:"); showArray(arr);
        System.out.println("Sap xep theo Quick Sort.");
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Day sap tang dan:");
        showArray(arr);
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Đánh giá

- Độ phức tạp trường hợp tồi nhất (chốt luôn là phần tử nhỏ nhất của dãy con đang xét): $O(n^2)$
- Độ phức tạp 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(nlogn)
- Độ phức tạp trường hợp trung bình: O(nlogn)
- Thuật toán quick sort hiệu quả trên tập dữ liệu lớn, nhưng không phải là lựa chọn tốt cho tập dữ liệu nhỏ.

©CMC Univ. 2022

Đánh giá

- Bảng so sánh độ phức tạp của các thuật toán sắp xếp :

Algorithm	Worst-case	Average-case	Best-case
Bubble sort	$O(n^2)$	$O(n^2)$	O(n)
Insertion sort	$O(n^2)$	$O(n^2)$	O(n)
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick sort	$O(n^2)$	O(nlogn)	O(nlogn)
Merge sort	O(nlogn)	O(nlogn)	O(n)

©CMC Univ. 2022

CMC UNIVERSITY

3.7. Sắp xếp vun đống (Heap Sort)

- Giới thiệu
- Cấu trúc dữ liệu đống (Heap)
- Thuật toán Heap sort
- Ví dụ minh họa
- Cài đặt thuật toán
- Độ phức tạp của thuật toán

©CMC Univ. 2022

CMC UNIVERSITY

SẮP XẾP VUN ĐỐNG

©CMC Univ. 2022

Giới thiệu

- Thuật toán sắp xếp vụ đống (heap sort) là một kỹ thuật sắp xếp dựa trên tổ chức dữ liệu Binary Heap.
- Heap sort giúp sắp xếp các phần tử trong danh sách sao cho phần tử lớn nhất được xếp vào cuối danh sách, và quá trình này sẽ lặp lại cho các phần tử còn lại trong danh sách.
- Heap sort thường được người dùng lựa chọn nhờ có tốc độ chạy nhanh và không quá phức tạp.

©CMC Univ. 2022

CMC UNIVERSITY

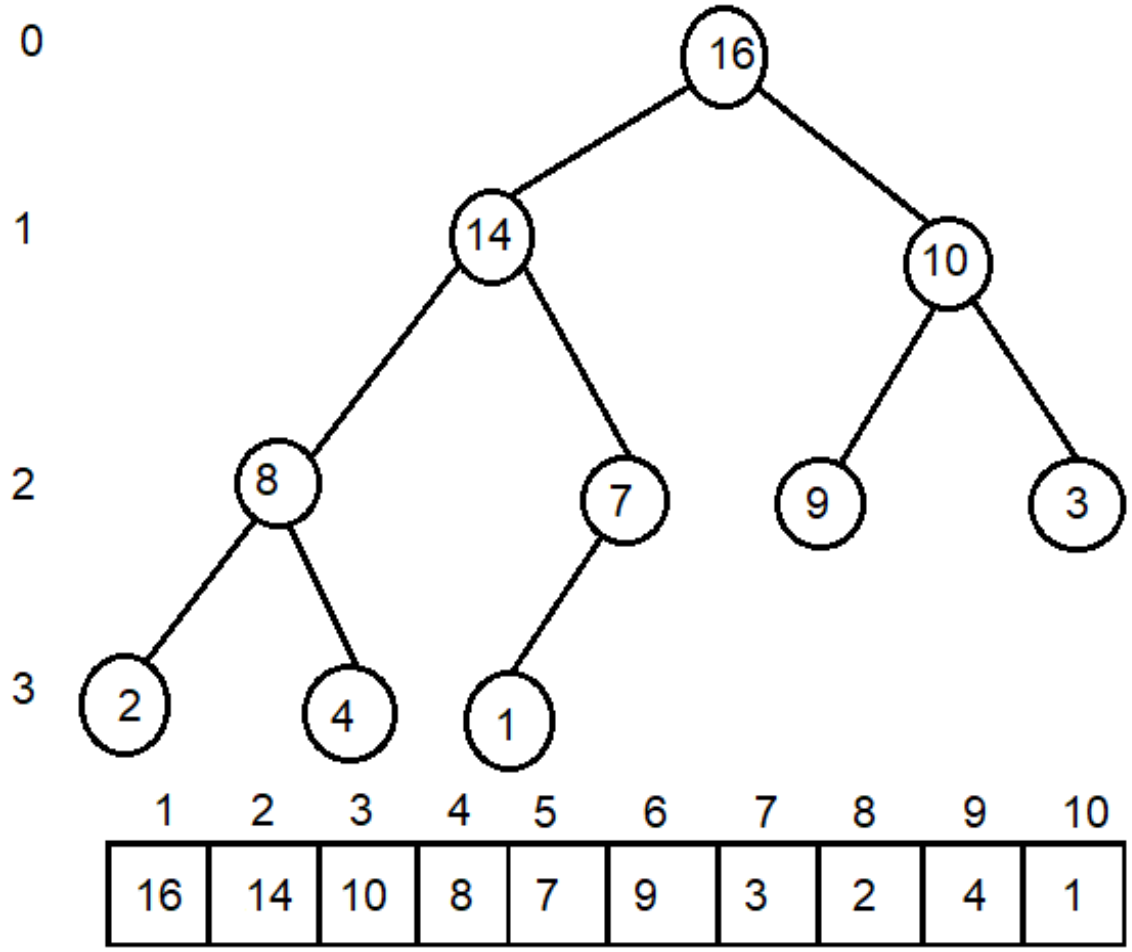
Cấu trúc dữ liệu đống (Heap)

- Đống (Heap) là cây nhị phân gần hoàn chỉnh có 2 tính chất sau:
  - ✓ Tính cấu trúc: tất cả các mức đều là đầy, ngoại trừ mức cuối cùng, mức cuối được điền từ trái qua phải
  - ✓ Tính có thứ tự: với mỗi nút x thì parent(x) ≥ x
- Từ tính chất đống suy ra: "Gốc chứa phần tử lớn nhất của đống".
- Như vậy, Heap là cây nhị phân được điền theo thứ tự

©CMC Univ. 2022

Cấu trúc dữ liệu đống (Heap)

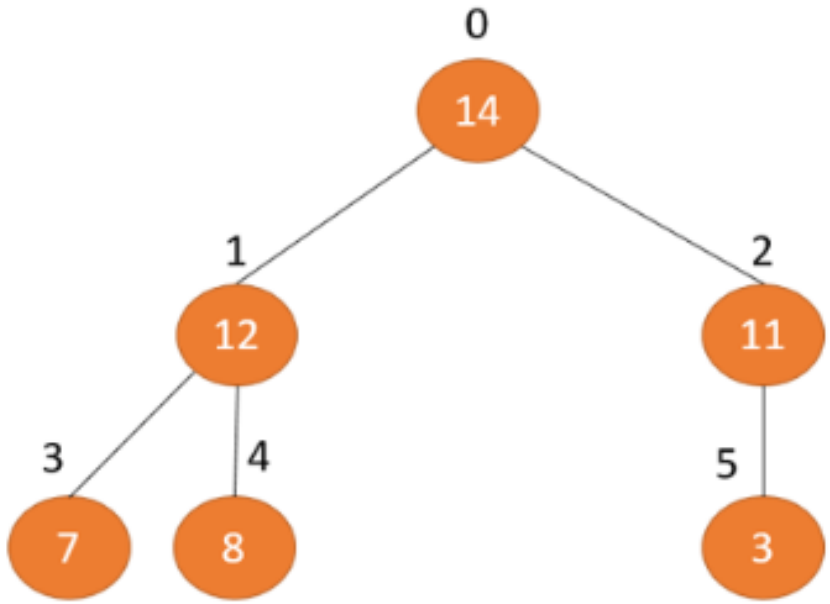
- Biểu diễn Heap bởi mảng
  - Gốc của cây là A[1]
  - Con trái của A[i] là A[2*i]
  - Con phải của A[i] là A[2*i+1]
  - Cha của A[i] là A[$\lfloor i/2 \rfloor$]



©CMC Univ. 2022

Cấu trúc dữ liệu đống (Heap)

- Max Heap: Phần tử trong nút cha luôn có giá trị lớn hơn so với tất cả các phần tử trong nút con. Và giá trị nút gốc là lớn nhất so với tất cả các nút khác.



Max Heap

©CMC Univ. 2022

CMC UNIVERSITY

Cấu trúc dữ liệu đống (Heap)

Các phép toán đối với Heap:

1. Bổ sung và loại bỏ nút
   - Nút mới được bổ sung vào mức đáy (từ trái qua phải)
   - Các nút được loại bỏ khỏi mức đáy (từ phải qua trái)
2. Các phép toán đối với Heap:
   - Khôi phục tính chất max-heap (vun lại đống): Max-Heapify
   - Tạo max-heap từ một mảng không được sắp xếp: Build-Max-Heap

©CMC Univ. 2022

Cấu trúc dữ liệu đống (Heap)

Thuật toán khôi phục tính chất Heap:

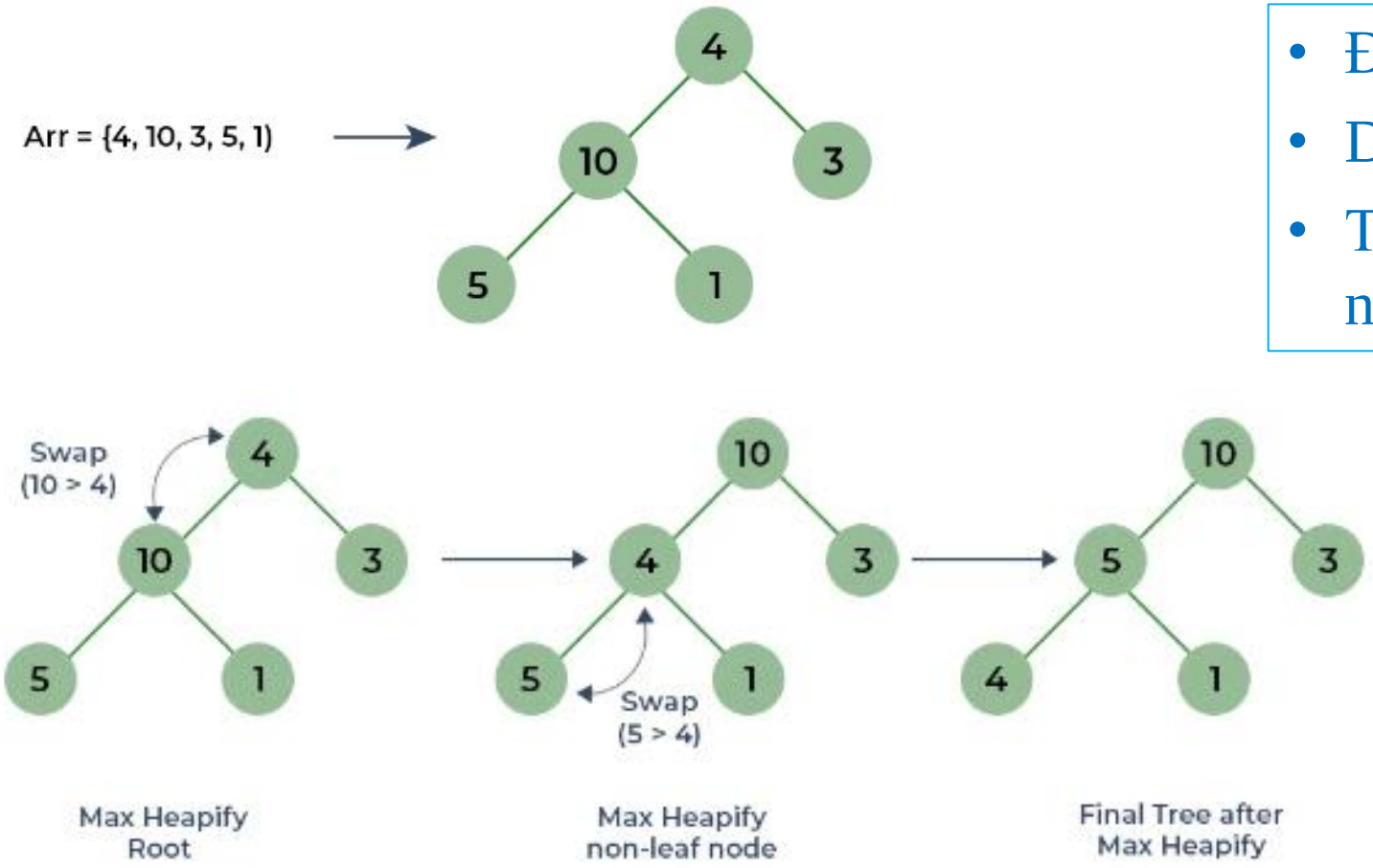
Giả sử có nút i với giá trị bé hơn con của nó. Giả thiết cây con trái và cây con phải của i đều là max-heaps. Để loại bỏ sự vi phạm này ta tiến hành như sau:

- Đổi chỗ với con lớn hơn
- Di chuyển xuống theo cây
- Tiếp tục quá trình cho tới khi nút không còn bé hơn con

©CMC Univ. 2022

Cấu trúc dữ liệu đống (Heap)

Thuật toán khôi phục tính chất Heap



- Đổi chỗ với con lớn hơn
- Di chuyển xuống theo cây
- Tiếp tục quá trình cho tới khi nút không còn bé hơn con

©CMC Univ. 2022

CMC UNIVERSITY

Thuật toán sắp xếp vun đống (Heap sort)

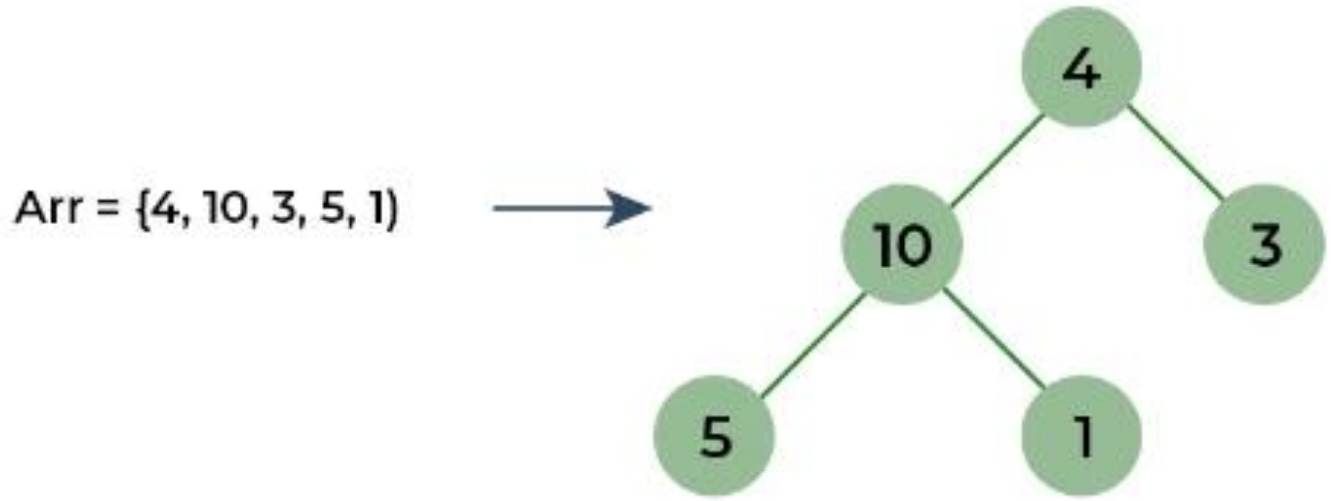
- Thuật toán Heap Sort hoạt động như sau:
  - Tạo Max-Heap từ mảng đã cho
  - Đổi chỗ gốc (phần tử lớn nhất) với phần tử cuối cùng trong mảng
  - Loại bỏ nút cuối cùng bằng cách giảm kích thước của Heap đi 1
  - Thực hiện Max-Heapify đối với gốc mới
  - Lặp lại quá trình trên cho đến khi Heap chỉ còn một nút (tức tất cả các phần tử của mảng được sắp xếp đúng).
- Thời gian chạy tổng thể: O(nlogn)

©CMC Univ. 2022

Minh họa thuật toán sắp xếp Heap sort

Cho mảng arr[] = {4, 10, 3, 5, 1}

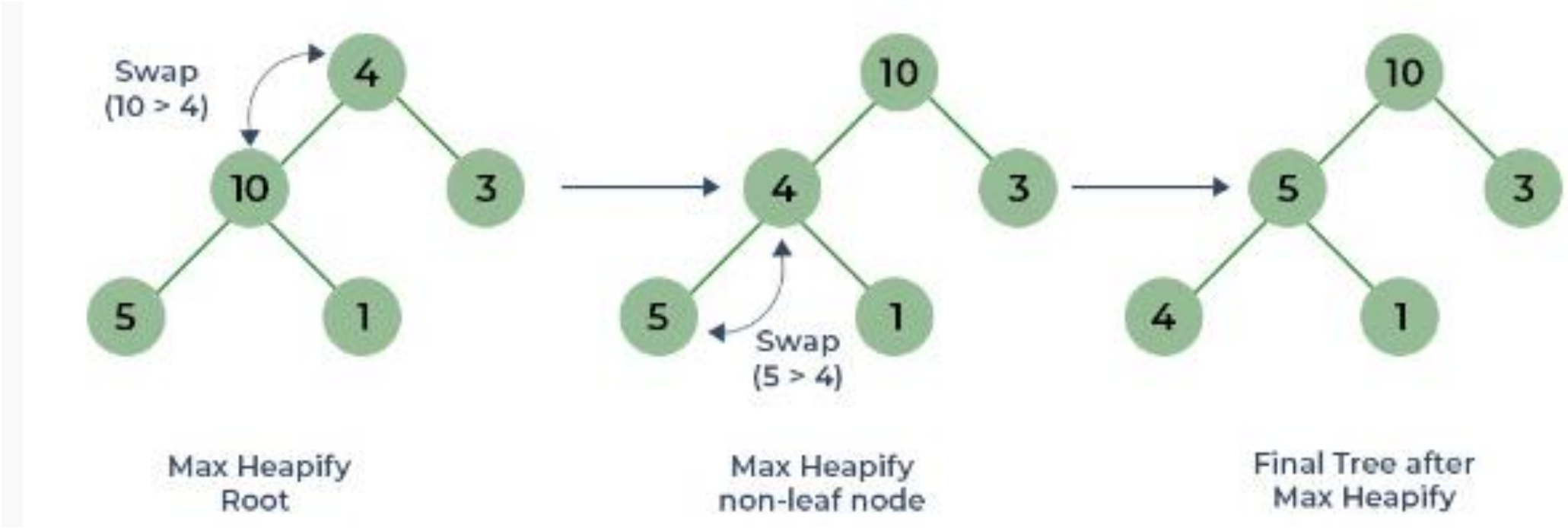
Bước 1: Xây dựng cây nhị phân đầy đủ từ mảng



©CMC Univ. 2022

Minh họa Thuật toán sắp xếp Heap sort

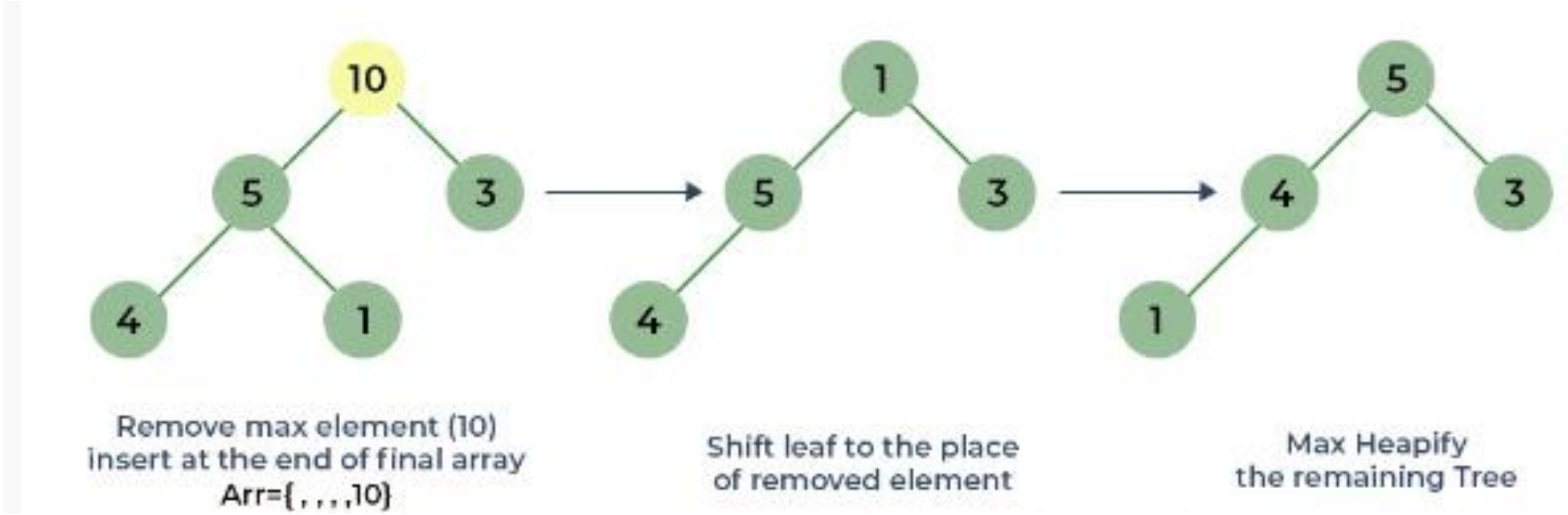
Bước 2: Tạo đống Max-Heap



©CMC Univ. 2022

Minh họa Thuật toán sắp xếp Heap sort

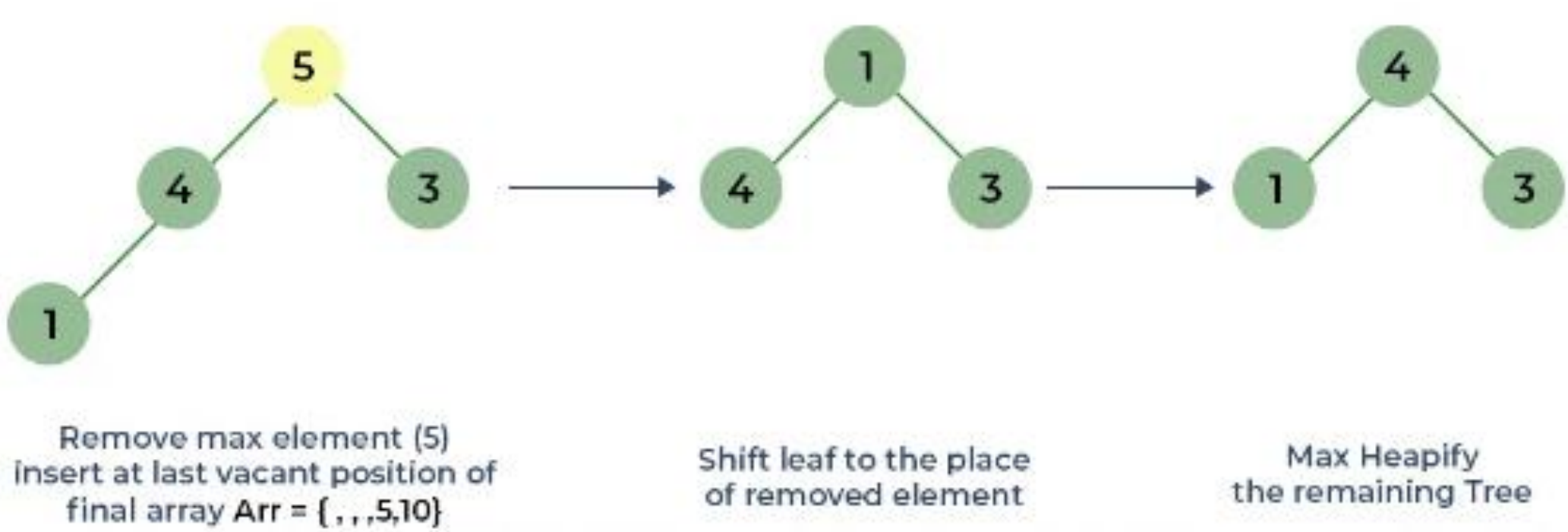
Bước 3: Loại bỏ phần tử lớn nhất khỏi gốc và tạo Max-Heap



©CMC Univ. 2022

Minh họa Thuật toán sắp xếp Heap sort

Bước 4: Loại bỏ phần tử lớn kế tiếp khỏi gốc và Max-Heap

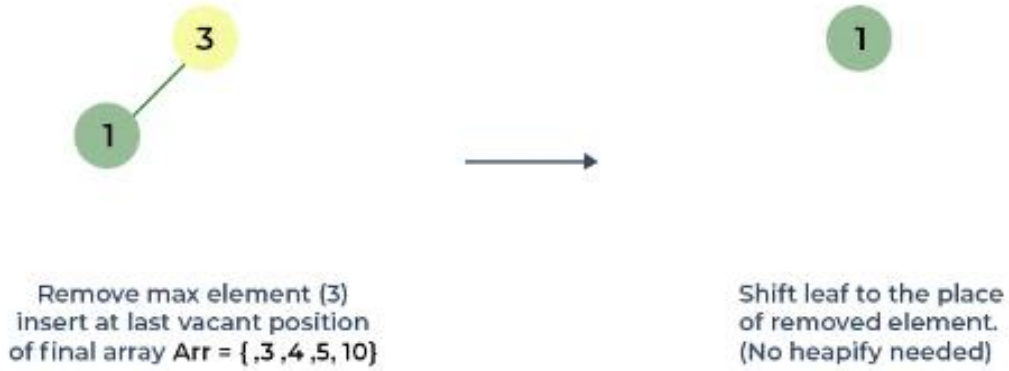


Bước 5: Tiếp tục thực hiện tương tự Bước 4

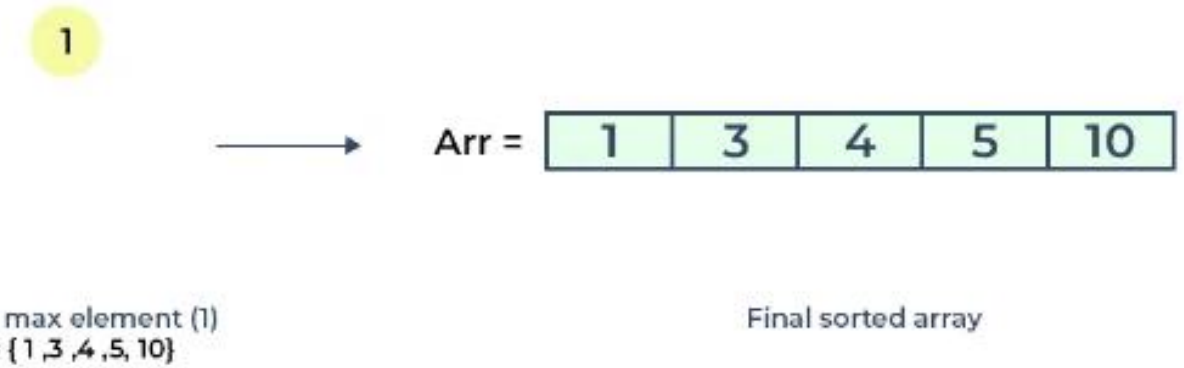
©CMC Univ. 2022

Minh họa thuật toán sắp xếp Heap sort

Bước 6: Loại bỏ phần tử lớn kế tiếp khỏi gốc và Max-Heap



Bước 7: Loại bỏ phần tử cuối cùng và nhận được mảng đã sắp xếp



©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp Heap Sort

```
private static void heapify (int arr[], int N, int i)
{
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if (left < N && arr[left] > arr[largest])
            largest = left;
        if (right < N && arr[right] > arr[largest])
            largest = right;
        if (largest != i) {
                int swap = arr[i];
                arr[i] = arr[largest];
                arr[largest] = swap;
                heapify(arr, N, largest);
            }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt thuật toán sắp xếp Heap Sort

```
public static void HeapSort(int arr[]){
        int N = arr.length;
        for (int i = N / 2 - 1; i >= 0; i--) // Build heap
            heapify(arr, N, i);
        for (int i = N - 1; i > 0; i--)
        { // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0); // call max heapify on the reduced heap
        }
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Cài đặt Thuật toán sắp xếp Heap sort

```
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Cho n = ");
        int n = input.nextInt();
        int[]arr = new int[n];
        System.out.println("Nhap day so:");
        inputArray(arr, n);
        System.out.println("In lai day so:");
        showArray(arr);
        System.out.println("Sap xep theo HeapSort.");
        HeapSort(arr);
        System.out.println("Day tang dan:");
        showArray(arr);
}
```

```
Cho n = 5
Nhap day so:
a[0] = 4
a[1] = 10
a[2] = 3
a[3] = 5
a[4] = 1
In lai day so:
4 10 3 5 1
Sap xep theo HeapSort.
Day tang dan:
1 3 4 5 10
```

©CMC Univ. 2022

Độ phức tạp của thuật toán Heap Sort

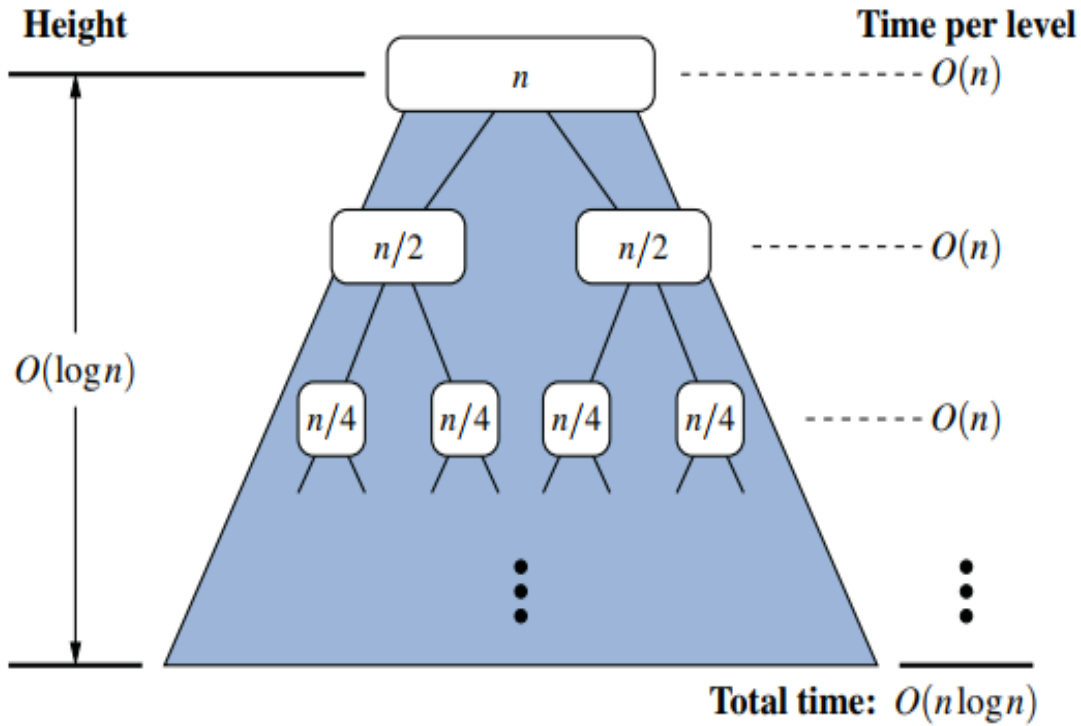
Thời gian tính toán:

- Max-Heapify: O(log n)
- Heap Sort: O(n log n)

Lưu ý:

Heap sort là một thuật toán sắp xếp trong

Có độ phức tạp O(nlog n) trong mọi trường hợp, nên nó hiệu quả cho tập dữ liệu sắp xếp lớn;



Độ phức tạp: O(n log n)

©CMC Univ. 2022

CMC UNIVERSITY

Sorting in java.util

Java cung cấp phương pháp sắp xếp cho mảng: Arrays.sort()

```
import java.util.Arrays;
import java.util.Scanner;
…
public static void main(String[] args) {
      Scanner input = new Scanner(System.in);
      System.out.print("Cho n = ");
      int n = input.nextInt();
      int[]arr = new int[n];
      System.out.println("Nhap day so:"); inputArray(arr, n);
      Arrays.sort(arr);  // gọi phương thức sort() của lớp Arrays
      System.out.println("Day tang dan:"); showArray(arr);
}
```

©CMC Univ. 2022

CMC UNIVERSITY

Đọc thêm

Sách: Data Structures and Algorithms in Java

- Sorting and Selection - 531
- Selection-Sort and Insertion-Sort - 386
- Bubble-sort (Exercise C-7.51)
- Quick-Sort - 544
- Merge-Sort - 532
- Heap-Sort - 388
- Linear-Time Sorting: Bucket-Sort and Radix-Sort - 558
- Comparing Sorting Algorithms - 561

©CMC Univ. 2022

CMC UNIVERSITY

Bài tập

1. Sắp xếp dãy { 3, 1, 4, 1, 5, 19, 2, 6, 15 } dùng thuật toán:

a. Sắp xếp chọn

b. Sắp xếp nổi bọt

c. Sắp xếp chèn

2. Dùng thuật toán sắp xếp trộn để sắp xếp dãy tăng dần:

{ 3, 1, 4, 1, 5, 9, 2, 6 }

3. Dùng thuật toán sắp xếp nhanh để sắp xếp dãy tăng dần:

{ 7, 6, 1, 5, 2, 8, 4, 3, 9, 10}

4. (Làm thêm) Dùng thuật toán sắp xếp vun đống để sắp xếp dãy:

{ 142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102 }

©CMC Univ. 2022


CMC UNIVERSITY
THANK YOU