INTRODUCTION TO DATA STRUCTURES AND ALGORITHM COURSE

DATA STRUCTURES AND ALGORITHMS

ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort
2. Đổi chỗ trực tiếp - Interchange Sort
3. Chèn trực tiếp - Insertion Sort
4. Chèn nhị phân - Binary Insertion Sort
5. **Quick Sort**
6. Heap Sort
7. Merge Sort

Quick Sort

- Được phát triển bởi Tony Hoare vào năm 1959
- Thuật toán Quicksort dựa trên chiến lược chia để trị (*divide-and-conquer*).

Quick sort

- Like mergesort, Quicksort is also based on the divide-and-conquer paradigm.
- But it uses this technique in a somewhat opposite manner, as all the hard work is done before the recursive calls.
- It works as follows:

1. First, it partitions an array into two parts,

2. Then, it sorts the parts independently,

3. Finally, it combines the sorted subsequences by a simple concatenation.

Quick sort

The quick-sort algorithm consists of the following three steps:

1. **Divide**: Partition the list.

    • To partition the list, we first choose element (call pivot) from the list for which we hope about half the elements will come before and half after. Call this element the pivot.

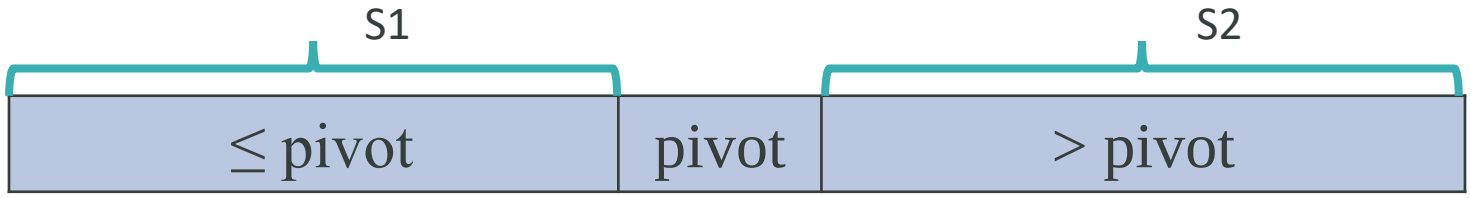
    • Then we partition the elements so that all those with values less than the pivot come in one sublist and all those with greater values come in another.

2. **Recursion**: Recursively sort the sublists separately.

3. **Conquer**: Put the sorted sublists together.

Quick Sort: Ý tưởng

- Giải thuật QuickSort sắp xếp dãy $\mathbf{a_0, a_1 \ldots, a_{n-1}}$ dựa trên việc phân hoạch dãy ban đầu thành 3 phần với pivot là giá trị của một phần tử **tùy ý** trong dãy ban đầu.



- Nếu các đoạn S1 và S3 chỉ có 1 phần tử: đã có thứ tự → khi đó dãy con ban đầu đã được sắp.

- Nếu các đoạn S1 và S3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn S1, S3 được sắp.

=> Để sắp xếp các Đoạn S1 và S3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày …

Quick Sort: Chọn Pivot

- Chọn phần tử ngẫu nhiên
- Chọn phần tử ở vị trí giữa
- Phần tử trung vị (Median) hay *Median-of-Three*
- Chọn phần tử đầu tiên, cuối cùng
  - The popular, uninformed choice is to use the first element as the pivot. This is acceptable if the input is random, but if the input is presorted (or input with a large presorted section) or in reverse order, then the pivot provides a poor partition, because either all the elements go into *part* 1 or they go into *part* 2.
- …

Quick Sort: Thuật toán

Thuật toán: QuickSort

- **Bước 1**: Nếu left ≥ right thì Kết thúc; // dãy đã có thứ tự
- **Bước 2**: Phân hoạch (Partition) dãy a_{left} .. a_{right} thành các đoạn a_{left} ..a_{i-1}, a_i, a_{i+1} .. a_{right}

  Đoạn 1: $a_{left} .. a_{i-1} \leq x$
  Đoạn 2: $a_i = x$
  Đoạn 3: $a_{i+1} .. a_{right} > x$

- **Bước 3**: Gọi đệ quy QuickSort sắp xếp đoạn 1: a_{left} .. a_{i-1}
- **Bước 4**: Gọi đệ quy QuickSort Sắp xếp đoạn 3: a_{i+1} .. a_{right}

Quick Sort: Giải thuật Partition

- **Bước 1**: Chọn tùy ý một phần tử a[k] trong dãy làm giá trị pivot

  (left ≤ k ≤ right): pivot **= a[k]; i = left; j = right;**

- **Bước 2**: Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ:
  - Bước 2a: Trong khi (a[i] < pivot) i++;
  - Bước 2b: Trong khi (a[j] > pivot) j--;
  - Bước 2c: Nếu i ≤ j:

    Doicho(a[i], a[j]);

    i++, j--

- **Bước 3**: Nếu i ≤ j: Lặp lại Bước 2.

  Ngược lại: Dừng

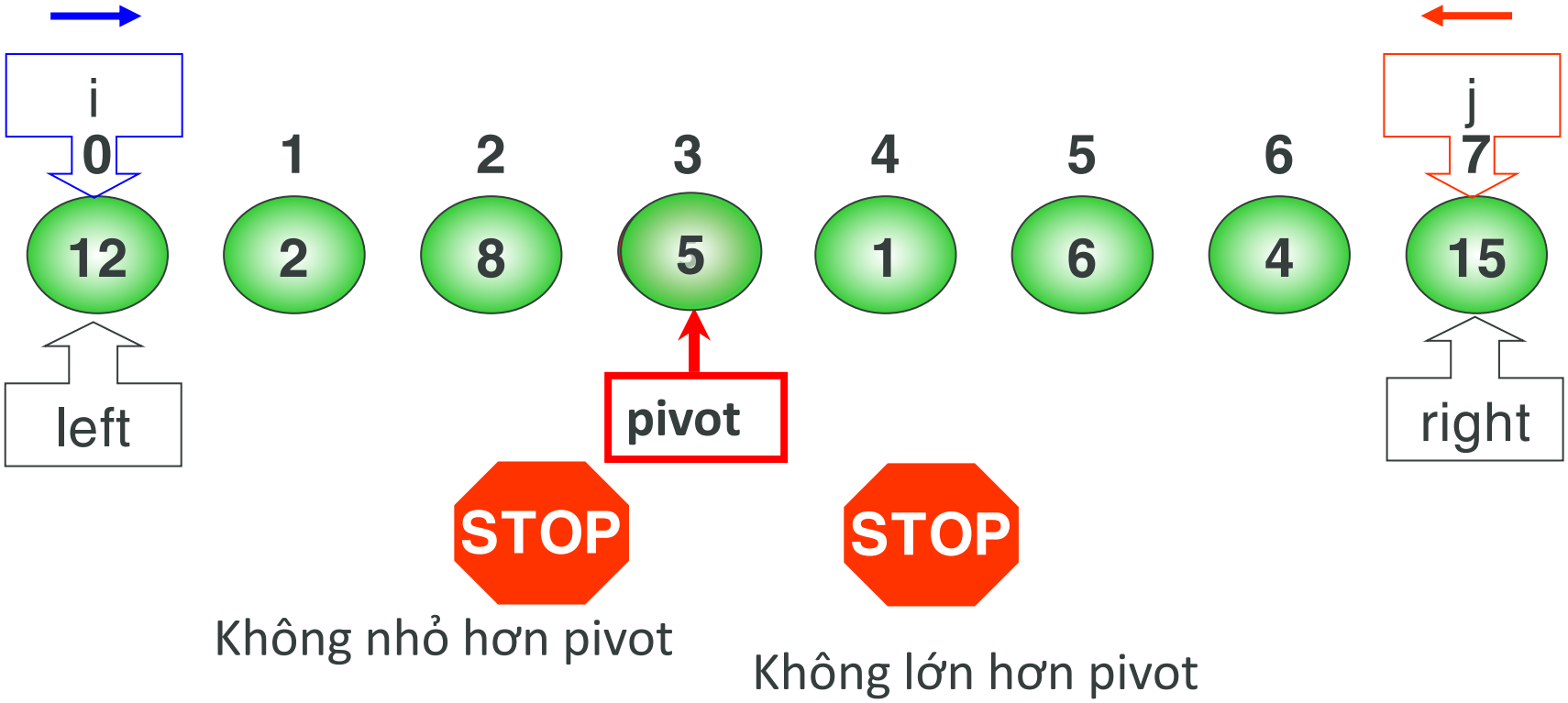
Quick Sort: Code C/C++

```
void QuickSort(int a[], int left, int right) {
    int i, j, pivot;
    pivot = a[(left + right) / 2];
    i = left; j = right;

    while (i <= j) {
        while (a[i] < pivot) i++;
        while (a[j] > pivot) j--;
        if (i <= j) {
            Swap(a[i], a[j]);
            i++; j--;
        }
    }

    if (left < j)     QuickSort(a, left, j);
    if (i < right)    QuickSort(a, i, right);
}


void QuickSort(int a[], int n) {

    QuickSort(a, 0, n - 1);
}
```

Quick Sort on Linked List

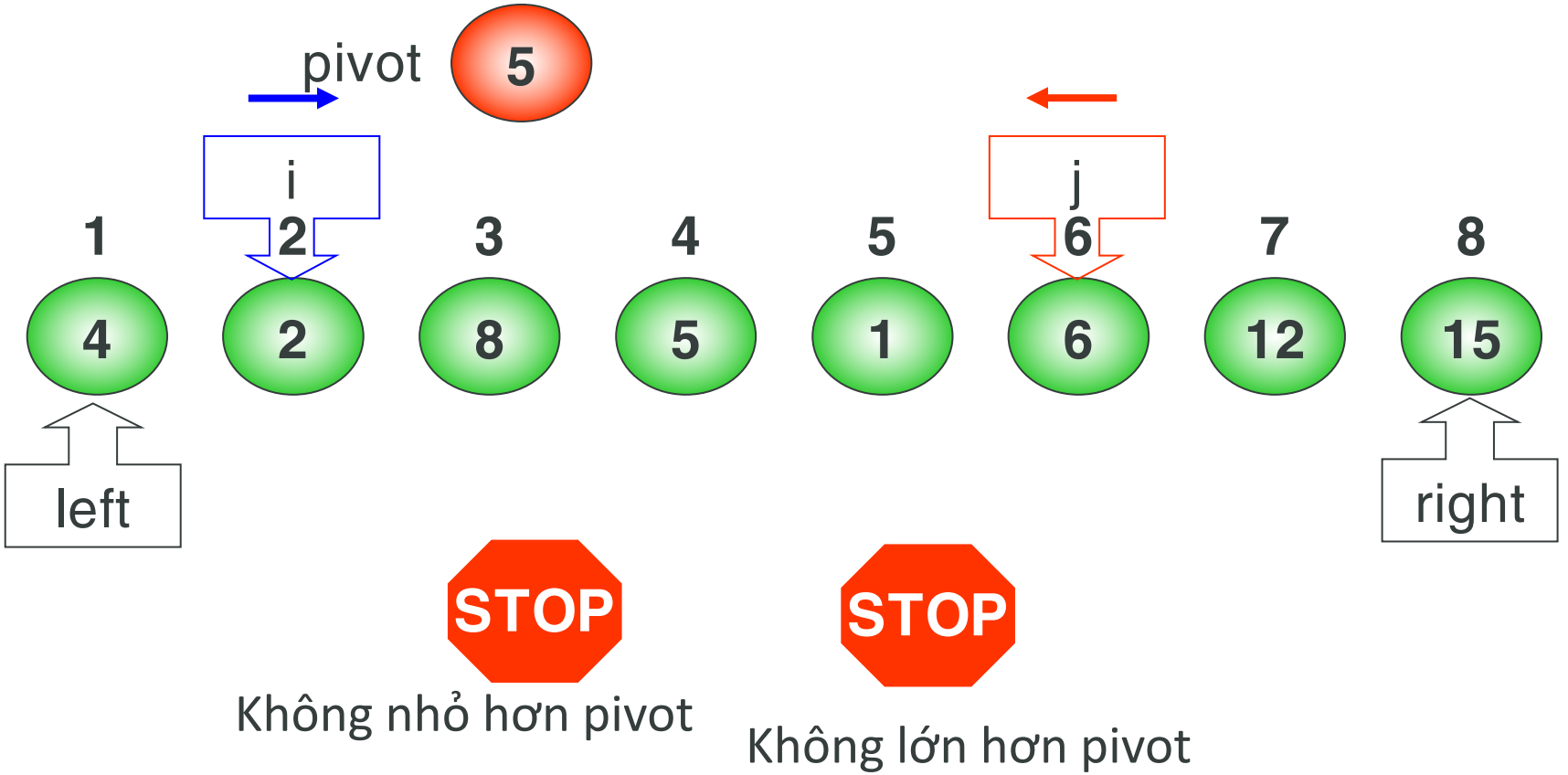
Quick Sort: Minh họa

Phân hoạch dãy

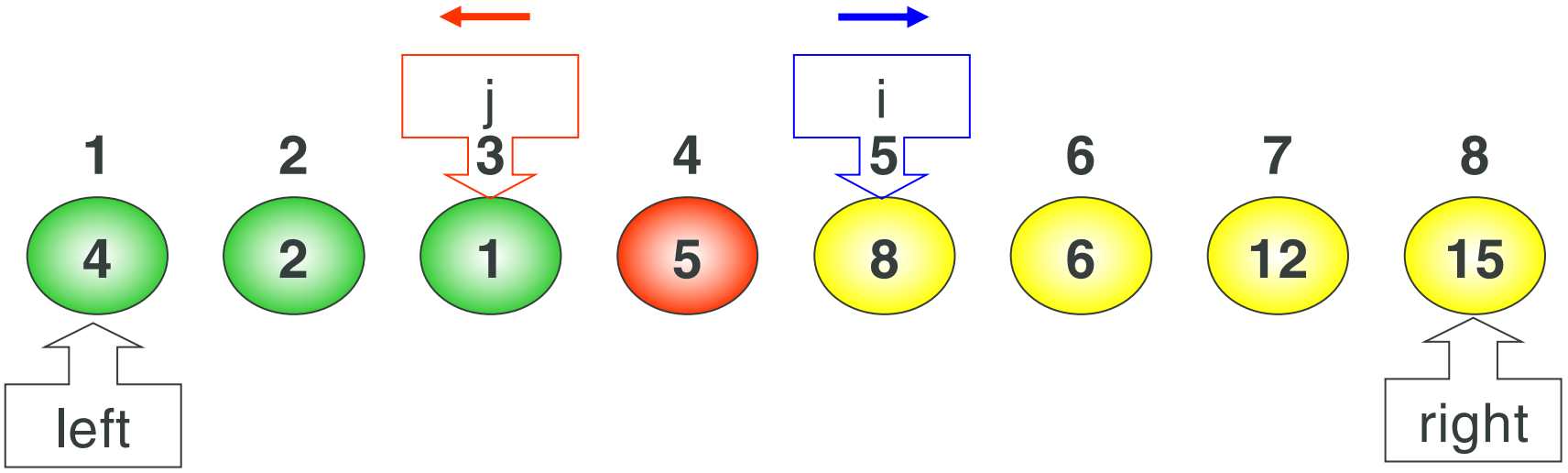


Quick Sort: Minh họa

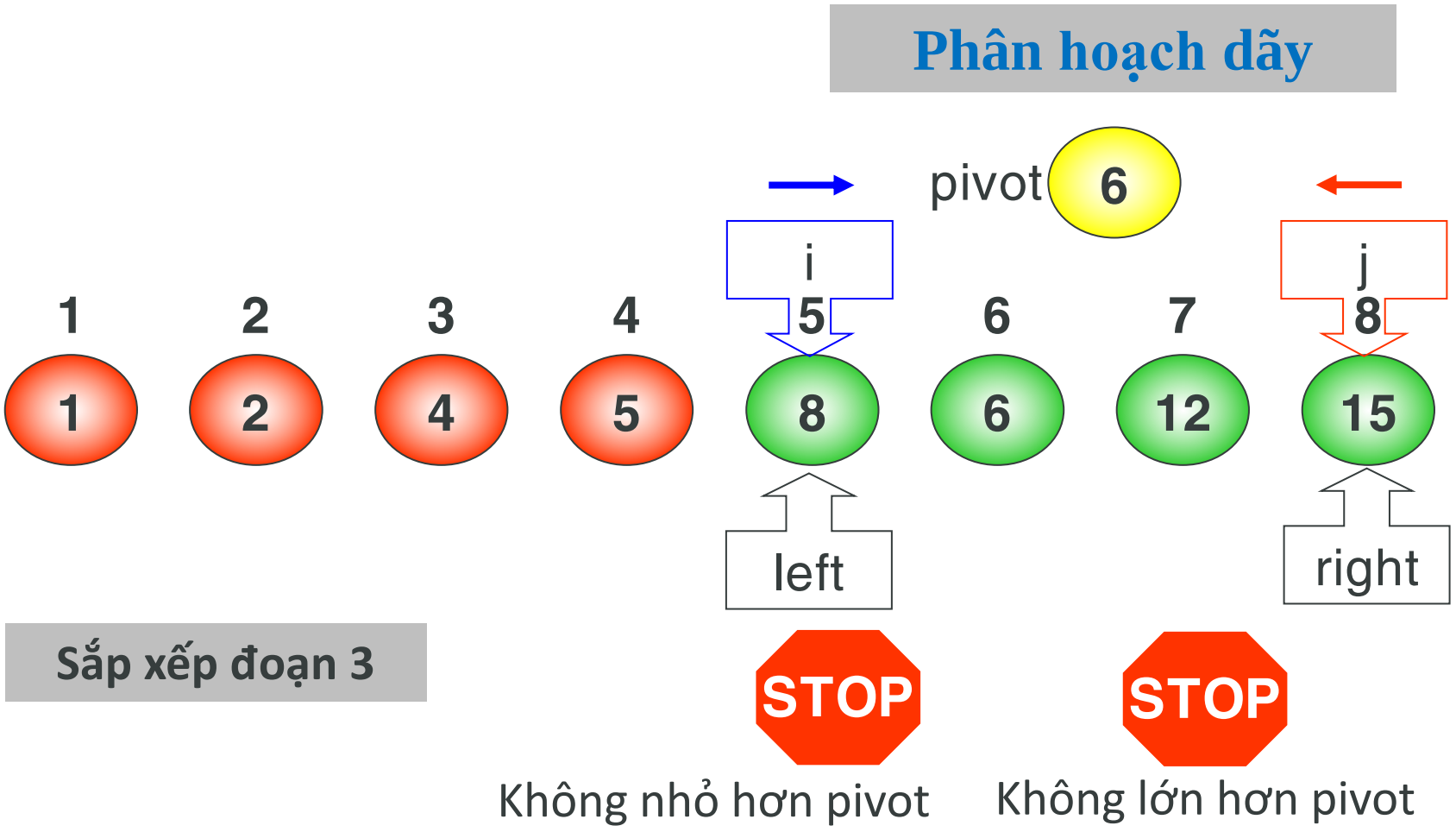
Phân hoạch dãy



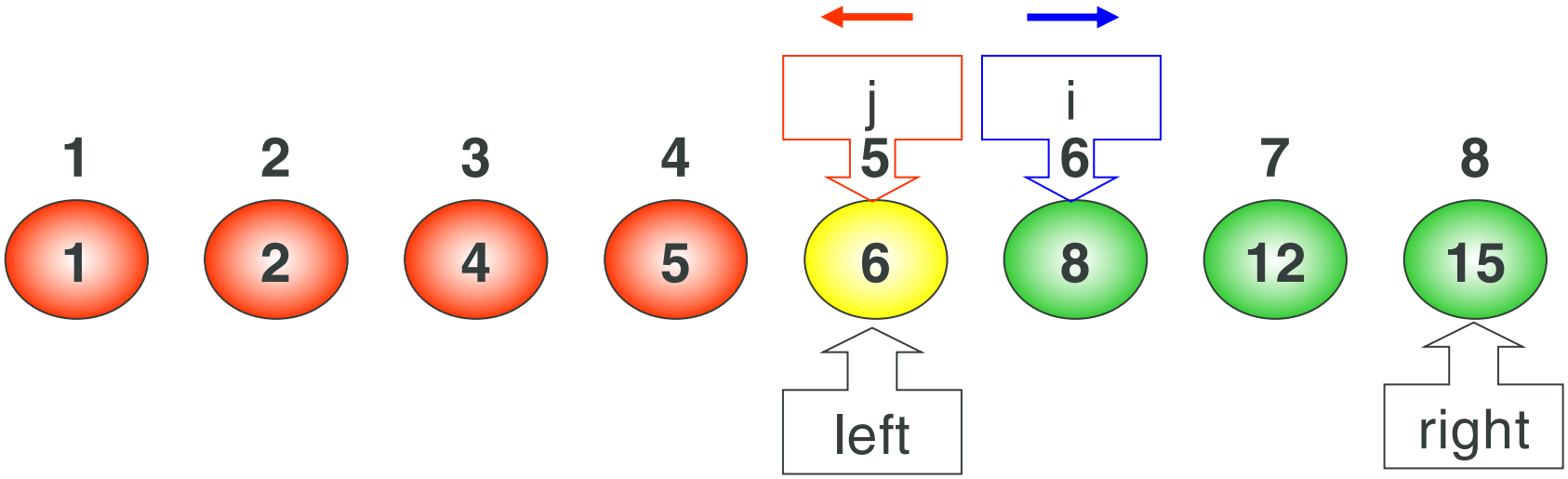
Quick Sort: Minh họa



Quick Sort: Minh họa



Quick Sort: Minh họa



Sắp xếp đoạn 3

Quick Sort: Độ phức tạp

- The worst case depends on strategy for choosing pivot: leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.
    1. Array is already sorted in same order.
    2. Array is already sorted in reverse order.
    3. All elements are same (special case of case 1 and 2)

Trường hợp	Độ phức tạp
Tốt nhất	$nlog_2n$
Trung bình	$nlog_2n$
Xấu nhất	n^2

- Quicksort là một trong những thuật toán hiệu quả nhất sử dụng phép so sánh.

8. Quick Sort

10. Merge Sort

III. CÁC GIẢI THUẬT SẮP XẾP

❖PHƯƠNG PHÁP TRỘN

Từ khóa: Merge Sort

Phân tích: Giả sử dãy A_1 và A_2 có **k** phần tử đã có thứ tự $\mathfrak{R}$, khi đó, có thể tạo dãy **A** có thứ tự $\mathfrak{R}$ gồm các phần tử của A_1 và A_2 với:

- Chi phí thời gian là **O(k)**
- Trộn từng theo thứ tự từ đầu danh sách.
- Phần tử trên hai danh sách được trộn theo thứ tự $\mathfrak{R}$

III. CÁC GIẢI THUẬT SẮP XẾP

❖PHƯƠNG PHÁP TRỘN

Ý tưởng: Áp dụng chiến lược chia để trị:

- Danh sách có 1 phần tử luôn có thứ thự.
- Để sắp xếp danh sách **A**:
  - Chia **A** thành hai danh sách $\mathbf{A_1}$ và $\mathbf{A_2}$
  - Sắp xếp $\mathbf{A_1}$ và $\mathbf{A_2}$ theo thứ tự $\mathfrak{R}$
  - Trộn $\mathbf{A_1}$ và $\mathbf{A_2}$ theo thứ tự $\mathfrak{R}$

III. CÁC GIẢI THUẬT SẮP XẾP

❖PHƯƠNG PHÁP TRỘN

Thuật toán:

mergeSort(A)

Đầu vào: $A=\{a_0, a_1, .., a_{n-1}\}$ chưa có thứ tự $\mathfrak{R}$

Đầu ra: $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\mathfrak{R}$

Nội dung

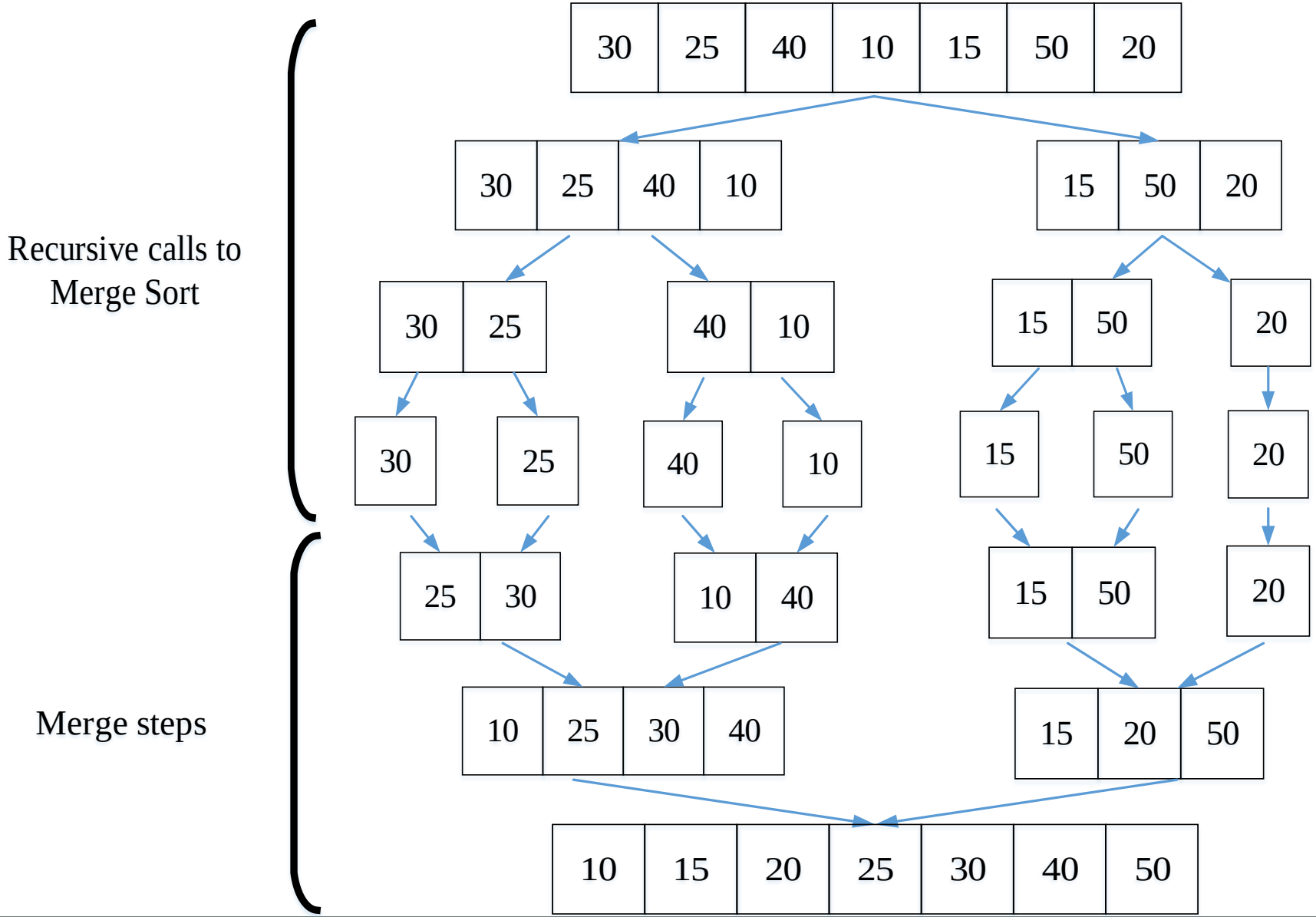
- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort
2. Đổi chỗ trực tiếp - Interchange Sort
3. Chèn trực tiếp - Insertion Sort
4. Chèn nhị phân - Binary Insertion Sort
5. Quick Sort
6. Heap Sort
7. **Merge Sort**

Merge Sort

- Thuật toán Mergesort áp dụng thuật toán chia để trị (divide and conquer) được phát minh bởi John von Neumann in 1945.
- Về mặt khái niệm, Thuật toán Mergesort hoạt động như sau:
  - Chia danh sách gồm n phần tử ban đầu thành n danh sách con (sublist), mỗi danh sách chứa 1 phần tử (danh sách 1 phần tử là danh sách có thứ tự).
  - Lặp lại việc trộn (merge) để tạo ra các danh sách con mới có thứ tự cho tới khi chỉ còn 1 danh sách con. Đây sẽ là danh sách được sắp xếp.

Top-down Merge Sort: Minh họa



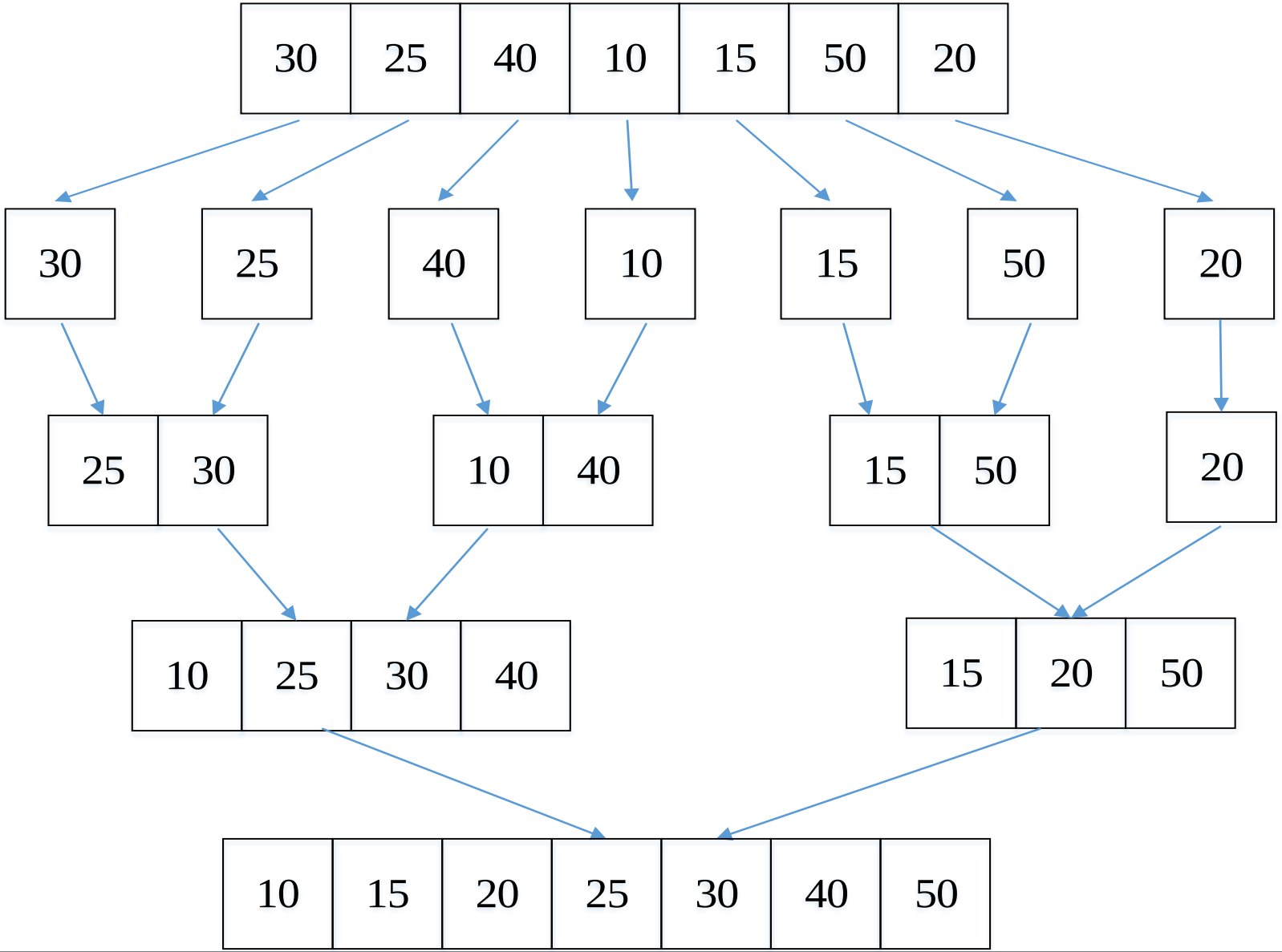
Top-down Merge Sort: Code C/C++

```
void TopDownMergeSort(int a[], int l, int r) {
     if (l < r) {
          int mid = (l + r) / 2;
          TopDownMergeSort(a, l, mid);
          TopDownMergeSort(a, mid + 1, r);
          TopDownMerge(a, l, r);
     }
}

void MergeSort(int a[], int n) {
     TopDownMergeSort(a, 0, n - 1);
}
```

Top-down Merge Sort: Code C/C++ (tt)

```
void TopDownMerge(int* a, int l, int r) {
    int *b, nb = r - l + 1; // Tạo mảng b lưu giá trị mảng a[l->r]
    b = new int[nb];
    Copy(b, a, l, r);

    int mid = (nb-1) / 2;
    // i0 là phần từ đầu tiên của dãy con thứ nhất trong b[0->mid]
    // i1 là phần từ đầu tiên của dãy con thứ nhất trong b[mid+1..nb-1]
    int i0 = 0, i1 = mid + 1;

    for (int j = l; j <= r; j++) {
      if (i0 <= mid && (i1 >= nb || b[i0] < b[i1]))
        a[j] = b[i0++];
      else
        a[j] = b[i1++];
    }
}
void Copy(int b[], int a[], int l, int r) {
    int nb = r - l + 1;
    for (int i = 0; i < nb; i++)
            b[i] = a[i+l];
}
```

Bottom-up Merge Sort: Minh họa



Bottom-up Merge Sort: Code C/C++ (tt)

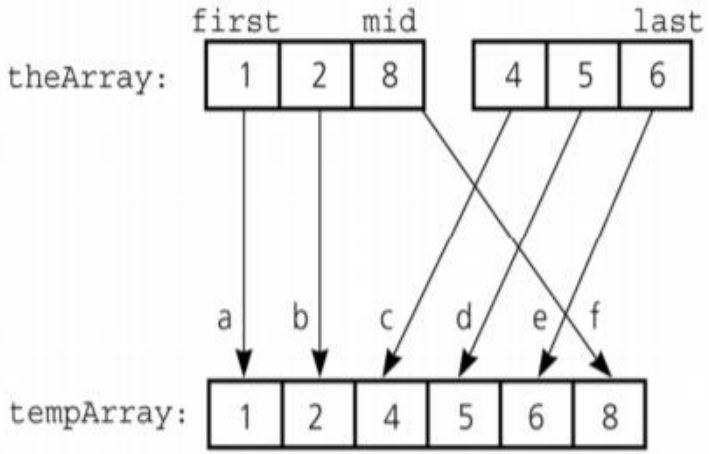
```
void Merge(int a[], int left, int middle, int right, int b[]) {
    int i = left, j = middle;
    for (int k = left; k <= right; k++)
        if (i < middle && (j > right || a[i] <= a[j]))
           b[k] = a[i++];
        else b[k] = a[j++];
}
void BottomUpMergeSort(int a[], int n) {
    int left, middle, right;
    int *b = new int[n];
    for (int width = 1; width < n; width = 2 * width) {
        for (int i = 0; i < n; i = i + 2 * width) {
            left = i; middle = min(i+width, n);
            right = min(i+2*width, n)-1;
            Merge(a, left, middle, right, b);
        }
        Copy(a, b, 0, n - 1);
    }
}
```

Minh họa thuật toán

width	Các bước	0	1	2	3	4	5	6
		30	25	40	10	15	50	20
w=1	i=0 Merge(a, 0, 1, 1, b)	**25**	**30**	40	10	15	50	20
	i=2 Merge(a, 2, 3, 3, b)	25	30	**10**	**40**	15	50	20
	i=4 Merge(a, 4, 5, 5, b)	25	30	10	40	**15**	**50**	20
	i=6 Merge(a, 6, 7, 6, b)	25	30	10	40	15	50	**20**
w=2	i=0 Merge(a, 0, 2, 3, b)	**10**	**25**	**30**	**40**	15	50	20
	i=4 Merge(a, 4, 6, 6, b)	10	25	30	40	**15**	**20**	**50**
w=4	i=0 Merge(a, 0, 4, 6, b)	10	15	20	25	30	40	50

Mergesort – Analysis of Merge

A worst-case instance of the merge step in *mergesort*

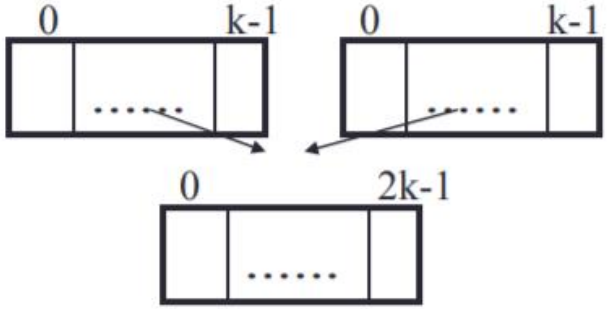


Merge the halves:

a. 1 < 4, so move 1 from `theArray[first..mid]` to `tempArray`
b. 2 < 4, so move 2 from `theArray[first..mid]` to `tempArray`
c. 8 > 4, so move 4 from `theArray[mid+1..last]` to `tempArray`
d. 8 > 5, so move 5 from `theArray[mid+1..last]` to `tempArray`
e. 8 > 6, so move 6 from `theArray[mid+1..last]` to `tempArray`
f. `theArray[mid+1..last]` is finished, so move 8 to `tempArray`

Mergesort – Analysis of Merge (cont.)

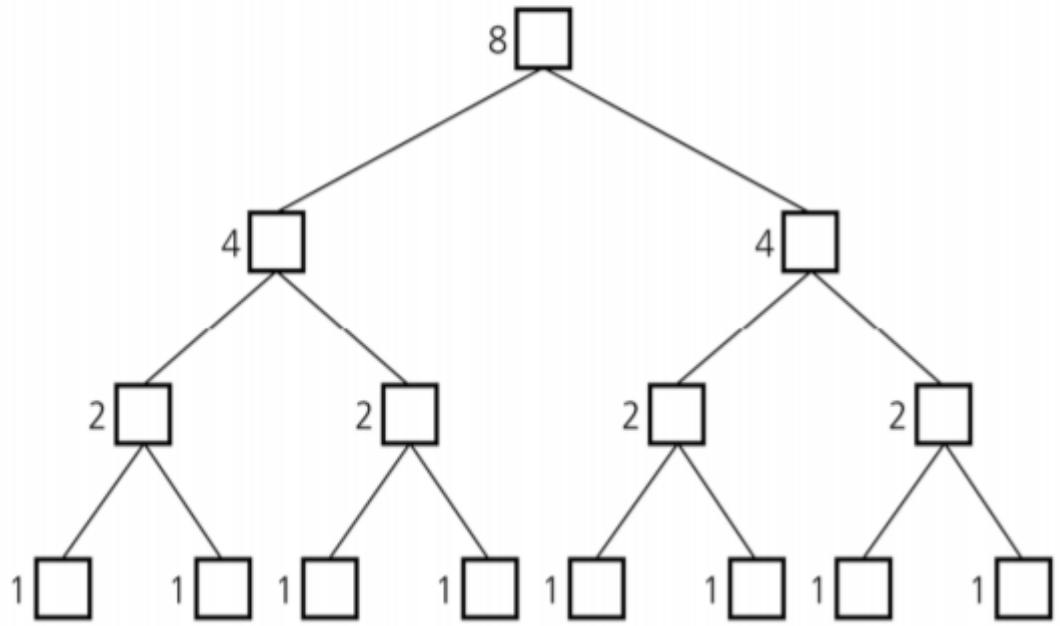
Merging two sorted arrays of size ***k***



- ***Best-case:***
  - All the elements in the first array are smaller (or larger) than all the elements in the second array.
  - The number of moves: 2k + 2k
  - The number of key comparisons: k
- ***Worst-case:***
  - The number of moves: 2k + 2k
  - The number of key comparisons: 2k-1

Mergesort - Analysis

Levels of recursive calls to *mergesort*, given an array of eight items



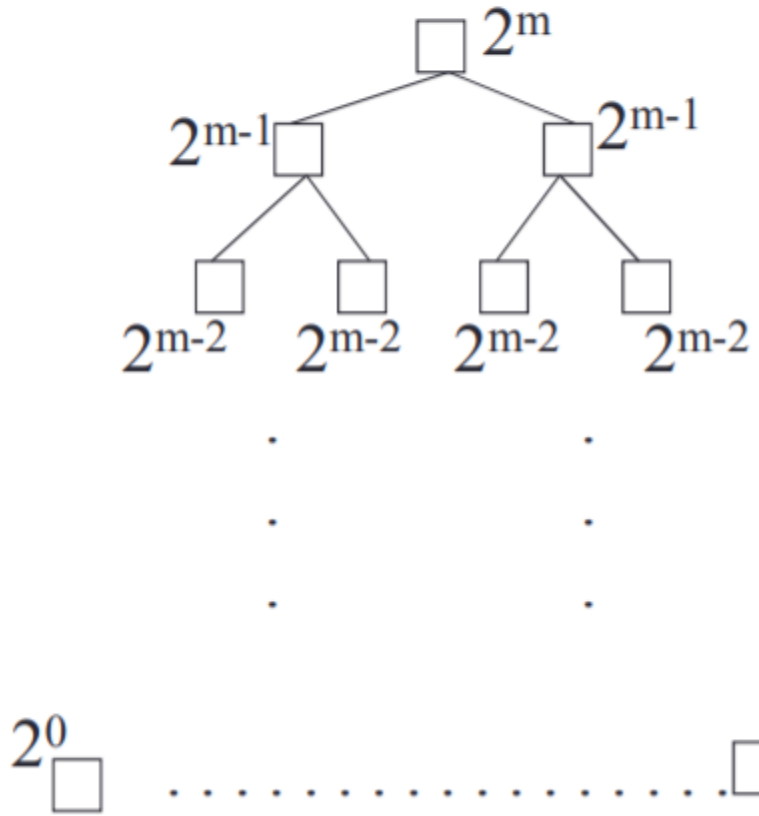
Level 0: `mergesort` 8 items

Level 1: 2 calls to `mergesort` with 4 items each

Level 2: 4 calls to `mergesort` with 2 items each

Level 3: 8 calls to `mergesort` with 1 item each

Mergesort - Analysis



level 0 : 1 merge (size 2^{m-1})

level 1 : 2 merges (size 2^{m-2})

level 2 : 4 merges (size 2^{m-3})

level m-1 : 2^{m-1} merges (size 2^0)

level m

Merge Sort: Phân tích độ phức tạp

Mergesort - Analysis

- ***Worst-case*** –

The number of key comparisons:

$= 2^0*(2*2^{m-1}-1) + 2^1*(2*2^{m-2}-1) + \ldots + 2^{m-1}*(2*2^0-1)$

$= (2^m - 1) + (2^m - 2) + \ldots + (2^m - 2^{m-1})$ (m terms)

$= m*2^m - \sum_{i=0}^{m-1} 2^i$

$= m*2^m - 2^m - 1$

Using $m = \log n$

$= n * \log_2 n - n - 1$

➔ $O(n * \log_2 n)$

Merge Sort: Đánh giá

- Trong mọi trường hợp độ phức tạp tính toán của phương pháp trộn là **O(nlogn).**

Trường hợp	Độ phức tạp
Tốt nhất	$nlog_2n$
Trung bình	$nlog_2n$
Xấu nhất	$nlog_2n$

- **Auxiliary Space:** O(n)

Merge Sort: Đánh giá

- Thích hợp cho các danh sách truy xuất tuần tự (file, danh sách đơn).
- Mergesort requires an extra array whose size equals to the size of the original array.
- If we use a linked list, we do not need an extra array
- Có thể thực hiện sắp xếp mà không cần nạp toàn bộ danh sách lên RAM (**External Sorting**).
- Trường hợp danh sách đã có những đoạn con có thứ tự → Trộn tự nhiên (**Natural Merge Sort**).

Comparison of Sorting Algorithms

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)
Radix Sort	Ω(nk)	θ(nk)	O(nk)

Câu hỏi và Bài tập

1. Trình bày ý tưởng của 3 thuật toán sắp xếp Quick sort, Merge sort, Radix sort?

2. Hãy trình bày những ưu khuyết điểm của 3 thuật toán sắp xếp ở câu 1? Theo bạn cách khắc phục những nhược điểm này là như thế nào?

3. Sử dụng hàm random trong C để tạo ra một dãy M có 1.000 số nguyên. Vận dụng 3 thuật toán sắp xếp ở câu 1 để sắp xếp các phần tử của mảng M theo thứ tự tăng dần về mặt giá trị. Với cùng một dữ liệu như nhau, cho biết thời gian thực hiện các thuật toán?

4. Thông tin về một sinh viên bao gồm: Mã số (là một số nguyên dương), Họ và đệm (là một chuỗi có tối đa 20 ký tự), Tên sinh viên (là một chuỗi có tối đa 10 ký tự), Ngày, tháng, năm sinh (là các số nguyên dương), Điểm trung bình (là các số thực có giá trị từ 0.00 ? 10.00).

Viết chương trình nhập vào danh sách sinh viên (ít nhất là 10 sinh viên, không nhập trùng mã giữa các sinh viên với nhau), sau đó vận dụng các thuật toán sắp xếp để sắp xếp danh sách sinh viên theo thứ tự tăng dần theo Mã sinh viên. In danh sách sinh viên sau khi sắp xếp ra màn hình.

Chúc các em học tốt!