HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Sắp xếp: bài toán sắp xếp, thuật toán sắp xếp cơ bản: sắp xếp lựa chọn, sắp xếp chèn, sắp xếp nổi bọt

ONE LOVE. ONE FUTURE.

NỘI DUNG

1. Bài toán sắp xếp
2. Sắp xếp chèn
3. Sắp xếp chọn
4. Sắp xếp nổi bọt

MỤC TIÊU

Sau bài học này, người học có thể:

1. Nắm vững thuật toán sắp xếp chèn, sắp xếp chọn, sắp xếp nổi bọt.

2. Cài đặt các thuật toán theo đúng định dạng

NỘI DUNG TIẾP THEO

1. Bài toán sắp xếp

2. Sắp xếp chèn
3. Sắp xếp chọn
4. Sắp xếp nổi bọt

1. BÀI TOÁN SẮP XẾP

- Sắp xếp (Sorting)
  - Quá trình tổ chức lại dữ liệu theo thứ tự giảm dần hoặc tăng dần
- Dữ liệu cần sắp xếp có thể là
  - Số nguyên
  - Xâu ký tự
  - Đối tượng (Objects)
- Khoá sắp xếp (Sort key)
  - Bộ phận của bản ghi xác định thứ tự sắp xếp của bản ghi trong danh sách.
  - Bản ghi sắp xếp theo thứ tự của các khoá.

1. BÀI TOÁN SẮP XẾP

➢ Chú ý

- Nếu sắp xếp trực tiếp trên bản ghi

  → Đòi hỏi di chuyển vị trí bản ghi → tốn kém.

- Xây dựng bảng khoá gồm các bản ghi chỉ có 2 trường:
  - "khoá" chứa giá trị khoá,
  - "con trỏ" để ghi địa chỉ của bản ghi tương ứng.
- Sắp xếp trên bảng khoá không làm thay đổi bảng chính
- Nhưng trình tự các bản ghi trong bảng khoá cho phép xác định trình tự các bản ghi trong bảng chính.

- Dạng tổng quát

<u>Input</u>: Dãy n số $A = (a_1, a_2, ..., a_n)$

<u>Output</u>: Một hoán vị (sắp xếp lại) $(a'_1,..., a'_n)$ của dãy số đã cho thoả mãn

$$a'_1 \leq ... \leq a'_n$$

1. BÀI TOÁN SẮP XẾP

- Các loại thuật toán sắp xếp
  - Sắp xếp trong (internal sort)
    - Đòi hỏi họ dữ liệu được đưa toàn bộ vào bộ nhớ trong của máy tính
  - Sắp xếp ngoài (external sort)
    - Họ dữ liệu không thể cùng lúc đưa toàn bộ vào bộ nhớ trong, nhưng có thể đọc vào từng bộ phận từ bộ nhớ ngoài

1. BÀI TOÁN SẮP XẾP

- Các loại thuật toán sắp xếp
  - Sắp xếp trong (internal sort)
    - Đòi hỏi họ dữ liệu được đưa toàn bộ vào bộ nhớ trong của máy tính
  - Sắp xếp ngoài (external sort)
    - Họ dữ liệu không thể cùng lúc đưa toàn bộ vào bộ nhớ trong, nhưng có thể đọc vào từng bộ phận từ bộ nhớ ngoài

➢ Các đặc trưng

- Tại chỗ (in place)
  - Nếu không gian nhớ phụ mà thuật toán đòi hỏi là *O*(1), 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.

1. BÀI TOÁN SẮP XẾP

- Hai phép toán cơ bản thường phải sử dụng
  - **Đổi chỗ** (Swap): Thời gian thực hiện là $O(1)$

```
void swap( datatype &a, datatype &b){
        datatype temp = a;  //datatype-kiểu dữ liệu của phần tử
        a = b;
        b = temp;
}
```

  - **So sánh:** Compare(a, b) trả lại
    - true nếu a đi trước b trong thứ tự cần sắp xếp
    - false nếu trái lại.

1. BÀI TOÁN SẮP XẾP

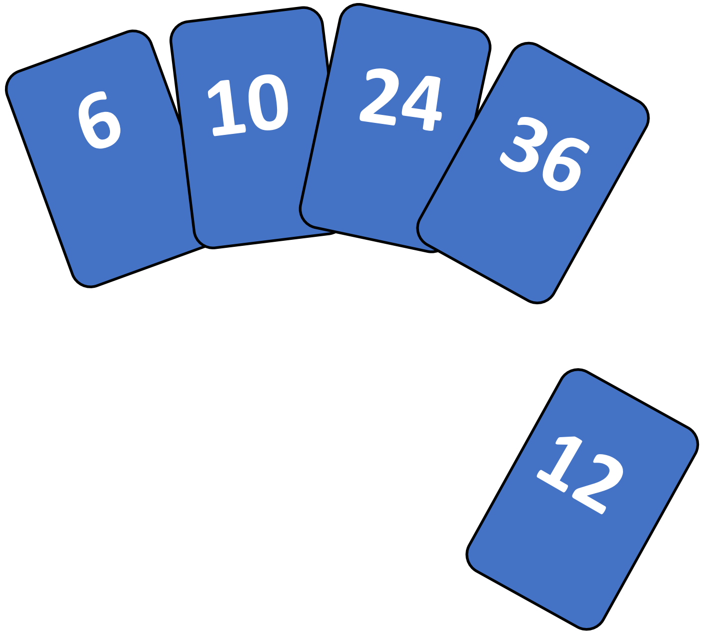
- Ứng dụng của sắp xếp
  - Quản trị cơ sở dữ liệu
  - Khoa học và kỹ thuật
  - Các thuật toán lập lịch,
    - Ví dụ thiết kế chương trình dịch, truyền thông,...
  - Máy tìm kiếm web
  - và nhiều ứng dụng khác...

NỘI DUNG TIẾP THEO

1. Bài toán sắp xếp
2. **Sắp xếp chèn**
3. Sắp xếp chọn
4. Sắp xếp nổi bọt

2. SẮP XẾP CHÈN

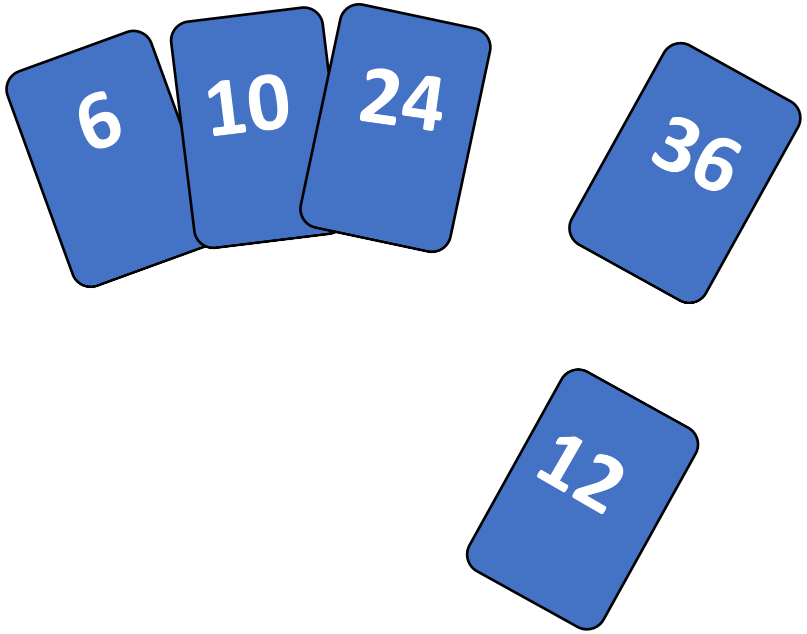
Phỏng theo cách làm của người chơi bài khi cần "chèn" thêm một con bài vào bộ bài đã được sắp xếp trên tay.



Để chèn 12, ta cần tạo chỗ cho nó bởi việc dịch chuyển đầu tiên là 36 và sau đó là 24.

2. SẮP XẾP CHÈN

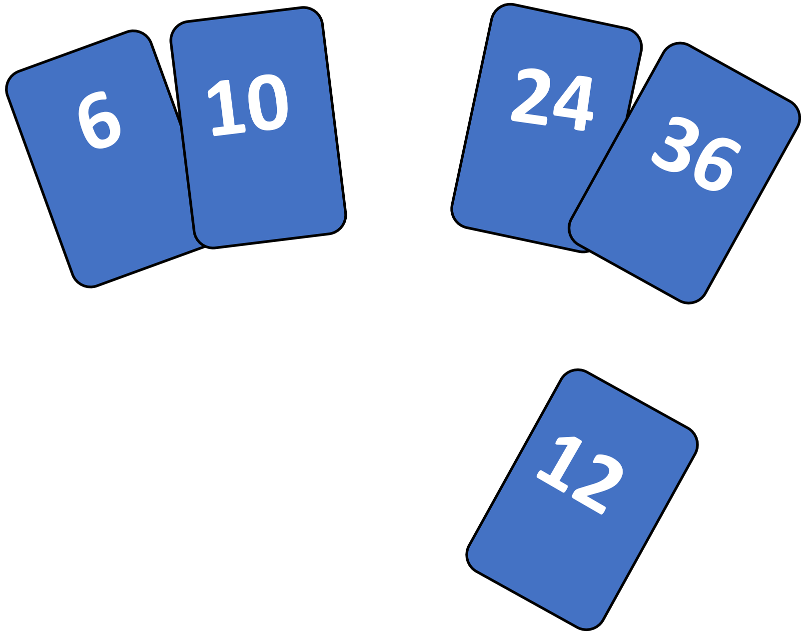
Phỏng theo cách làm của người chơi bài khi cần "chèn" thêm một con bài vào bộ bài đã được sắp xếp trên tay.



Để chèn 12, ta cần tạo chỗ cho nó bởi việc dịch chuyển đầu tiên là 36 và sau đó là 24.

2. SẮP XẾP CHÈN

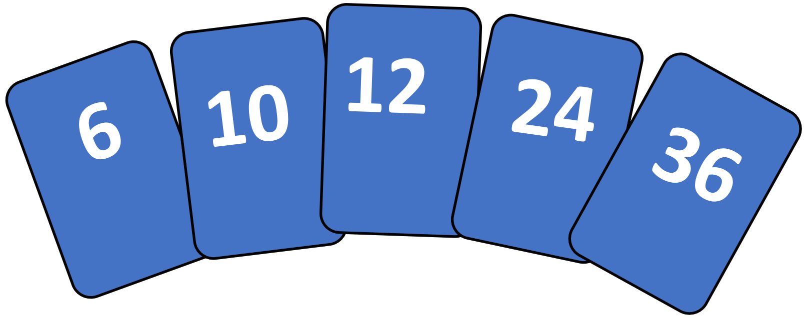
Phỏng theo cách làm của người chơi bài khi cần "chèn" thêm một con bài vào bộ bài đã được sắp xếp trên tay.



Để chèn 12, ta cần tạo chỗ cho nó bởi việc dịch chuyển đầu tiên là 36 và sau đó là 24.

2. SẮP XẾP CHÈN

Phỏng theo cách làm của người chơi bài khi cần "chèn" thêm một con bài vào bộ bài đã được sắp xếp trên tay.



Để chèn 12, ta cần tạo chỗ cho nó bởi việc dịch chuyển đầu tiên là 36 và sau đó là 24.

2. SẮP XẾP CHÈN

Thuật toán:

- Tại bước *k* = 1, 2, ..., *n*, đưa phần tử thứ *k* trong mảng đã cho vào đúng vị trí trong dãy gồm *k* phần tử đầu tiên.
- Kết quả là sau bước *k*, *k* phần tử đầu tiên là được sắp thứ tự.

Giải thích:

- Ở đầu lần lặp i của vòng "for" ngoài, dữ liệu từ a[0] đến a[*i*-1] là được sắp xếp.

- Vòng lặp "while" tìm vị trí cho phần tử tiếp theo (last =a[i]) trong dãy gồm i phần tử đầu tiên.

```
void insertionSort(int a[], int array_size) {
  int i, j, last;
  for (i=1; i < array_size; i++)  {
       last = a[i];
       j = i;
       while ((j > 0) && (a[j-1] > last))   {
              a[j] = a[j-1];
              j = j - 1;  }// end while
       a[j] = last;
  } // end for
} // end of isort
```

➢ Ví dụ

1	2	3	8	7	9	10	12	23	18	15	16	17	14
1	2	3	7	8	9	10	12	23	18	15	16	17	14
1	2	3	7	8	9	10	12	18	23	15	16	17	14
1	2	3	7	8	9	10	12	18	15	23	16	17	14
1	2	3	7	8	9	10	12	15	18	23	16	17	14
1	2	3	7	8	9	10	12	15	18	16	23	17	14
1	2	3	7	8	9	10	12	15	16	18	23	17	14

2. SẮP XẾP CHÈN

➢ Ví dụ

13 phép đổi chỗ:

20 phép so sánh:

1	2	3	7	8	9	10	12	15	16	18	17	23	14
1	2	3	7	8	9	10	12	15	16	17	18	23	14
1	2	3	7	8	9	10	12	15	16	17	18	14	23
1	2	3	7	8	9	10	12	15	16	17	14	18	23
1	2	3	7	8	9	10	12	15	16	14	17	18	23
1	2	3	7	8	9	10	12	15	14	16	17	18	23
1	2	3	7	8	9	10	12	14	15	16	17	18	23

2. SẮP XẾP CHÈN

- Các đặc tính của sắp xếp chèn
- Tại chỗ và ổn định (In place and Stable)
- Thời gian tính
  - ***Best Case:*** 0 hoán đổi, *n*-1 so sánh (*khi dãy đầu vào là đã được sắp*)
  - ***Worst Case:*** $n^2/2$ hoán đổi và so sánh (*khi dãy đầu vào có thứ tự ngược lại với thứ tự cần sắp xếp*)
  - ***Average Case:*** $n^2/4$ hoán đổi và so sánh
  - Trong tình huống tốt nhất là tốt nhất
- Thuật toán sắp xếp tốt đối với dãy đã *gần được sắp xếp*
  - Mỗi phần tử đã đứng ở vị trí rất gần vị trí trong thứ tự cần sắp xếp

NỘI DUNG TIẾP THEO

1. Bài toán sắp xếp
2. Sắp xếp chèn
3. **Sắp xếp chọn**
4. Sắp xếp nổi bọt

3. SẮP XẾP CHỌN

- Thuật toán
  - Tìm phần tử nhỏ nhất đưa vào vị trí 1
  - Tìm phần tử nhỏ tiếp theo đưa vào vị trí 2
  - Tìm phần tử nhỏ tiếp theo đưa vào vị trí 3
  - …

```
void swap(int &a,int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

```
void selectionSort(int a[], int n){
  int i, j, min, temp;
  for (i = 0; i < n-1; i++) {
    min = i;
    for (j = i+1; j < n; j++){
       if (a[j] < a[min]) min = j;
    }
    swap(a[i], a[min]);
  }
}
```

3. SẮP XẾP CHỌN

➢ Nhận xét

- **Best case:** 0 đổi chỗ (n-1 như trong đoạn mã), $n^2/2$ so sánh.
- **Worst case:** n - 1 đổi chỗ và $n^2/2$ so sánh.
- **Average case:** $O(n)$ đổi chỗ và $n^2/2$ so sánh.
- Ưu điểm nổi bật: số phép đổi chỗ ít.
  - Có ý nghĩa nếu như thao tác đổi chỗ tốn kém.

- Ví dụ

	i=0	1	2	3	4	5	6
42	13	13	13	13	13	13	13
20	20	14	14	14	14	14	14
17	17	17	15	15	15	15	15
13	42	42	42	17	17	17	17
28	28	28	28	28	20	20	20
14	14	20	20	20	28	23	23
23	23	23	23	23	23	28	28
15	15	15	17	42	42	42	42

NỘI DUNG TIẾP THEO

1. Bài toán sắp xếp
2. Sắp xếp chèn
3. Sắp xếp chọn
4. **Sắp xếp nổi bọt**

4. SẮP XẾP NỔI BỌT

➢ Ví dụ

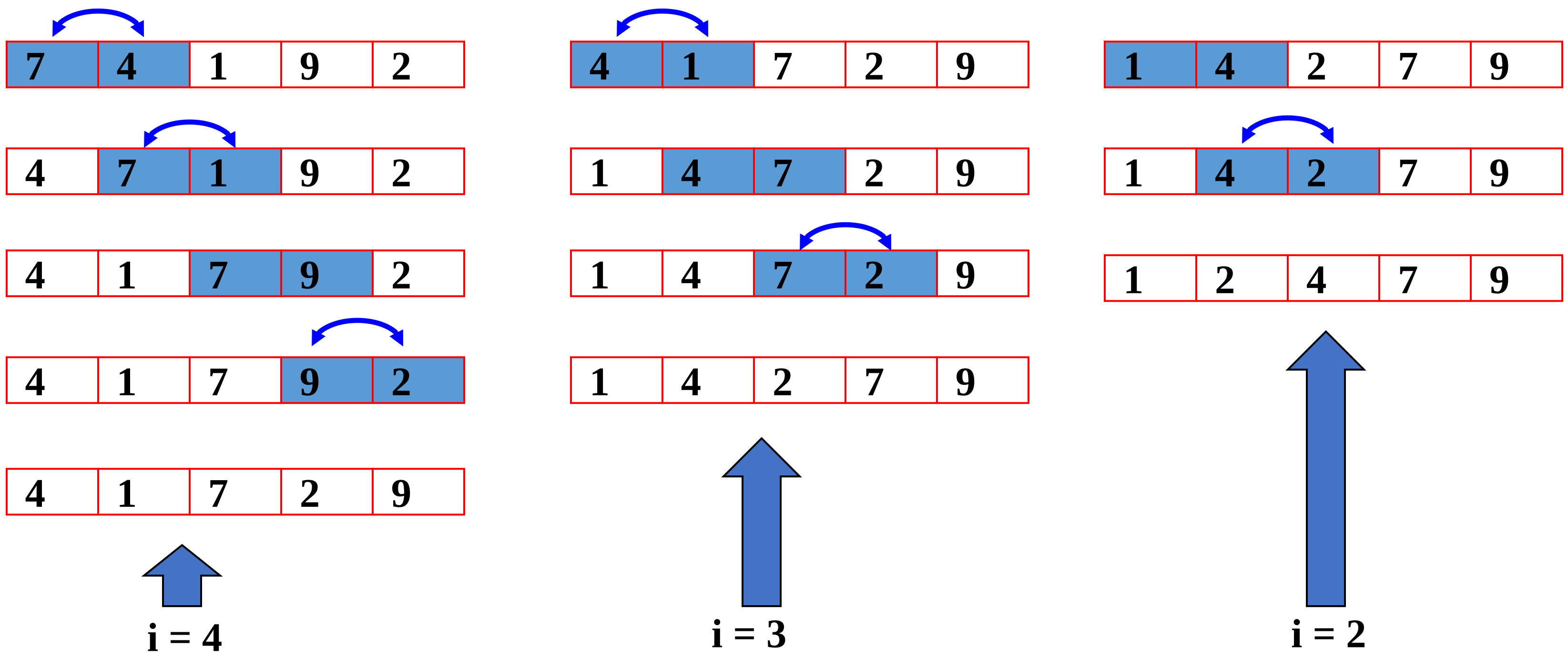
- Bắt đầu từ đầu dãy, thuật toán tiến hành so sánh mỗi phần tử với phần tử đi sau nó và thực hiện đổi chỗ, nếu chúng không theo đúng thứ tự.
- Quá trình này sẽ được lặp lại cho đến khi gặp lần duyệt từ đầu dãy đến cuối dãy mà không phải thực hiện đổi chỗ (tức là tất cả các phần tử đã đứng đúng vị trí).
- Cách làm này đã đẩy phần tử lớn nhất xuống cuối dãy, trong khi đó những phần tử có giá trị nhỏ hơn được dịch chuyển về đầu dãy.

4. SẮP XẾP NỔI BỌT

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

```
void swap(int &a,int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

- Best case: 0 đổi chỗ, $n^2/2$ so sánh.
- Worst case: $n^2/2$ đổi chỗ và so sánh.
- Average case: $n^2/4$ đổi chỗ và $n^2/2$ so sánh.



Chú ý:

- Các phần tử được đánh chỉ số bắt đầu từ 0.
- n=5

TỔNG KẾT BA THUẬT TOÁN SẮP XẾP CƠ BẢN

	Insertion	Bubble	Selection
Số phép so sánh:			
Best Case	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Average Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Worst Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Số phép đổi chỗ:			
Best Case	0	0	$\Theta(n)$
Average Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Worst Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$

Chương 7 – Sắp xếp

Bài 2. Sắp xếp trộn (merge sort)

NỘI DUNG TIẾP THEO

1. **Sắp xếp trộn (merge sort)**
   1.1. Sơ đồ thuật toán
   1.2. Thời gian tính
   1.3. Ví dụ
2. Cài đặt thuật toán

1. SẮP XẾP TRỘN (MERGE SORT)

- 1.1. Sơ đồ thuật toán
  - **Chia (Divide)**
    - Chia dãy gồm *n* phần tử cần sắp xếp ra thành 2 dãy, mỗi dãy có *n*/2 phần tử
  - **Trị (Conquer)**
    - Sắp xếp mỗi dãy con một cách đệ qui sử dụng ***sắp xếp trộn***
    - Khi dãy chỉ còn một phần tử thì trả lại phần tử này
  - **Tổ hợp (Combine)**
    - Trộn (Merge) hai dãy con được sắp xếp để thu được dãy được sắp xếp gồm tất cả các phần tử của cả hai dãy con

- 1.1. Sơ đồ thuật toán

MERGE-SORT(A, p, r)

p			q				r
1	2	3	4	5	6	7	8
5	2	4	7	1	3	2	6

```
if p < r                                ▷ Kiểm tra điều kiện neo
  then q ← ⌊(p + r)/2⌋                  ▷ Chia (Divide)
       MERGE-SORT(A, p, q)              ▷Trị (Conquer)
       MERGE-SORT(A, q + 1, r)          ▷ Trị (Conquer)
       MERGE(A, p, q, r)                ▷ Tổ hợp (Combine)
endif
```

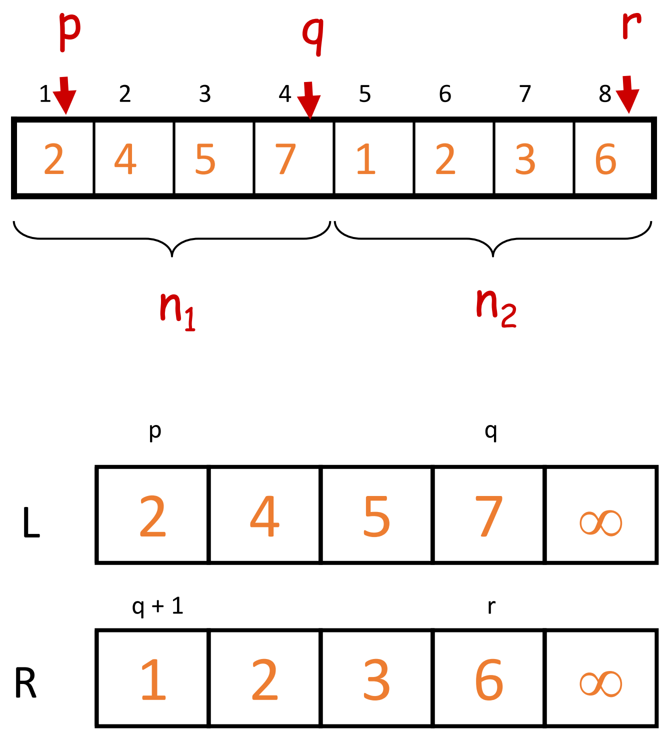
- Lệnh gọi thực hiện thuật toán: MERGE-SORT(A, 1, n)

1. SẮP XẾP TRỘN (MERGE SORT)

- 1.1. Sơ đồ thuật toán

MERGE(A, p, q, r)

1. Tính n_1 và n_2
2. Sao n_1 phần tử đầu tiên vào L[1 . . n_1] và n_2 phần tử tiếp theo vào R[1 . . n_2]
3. L[n_1 + 1] ← ∞;   R[n_2 + 1] ← ∞
4. i ← 1;   j ← 1
5. **for** k ← p **to** r **do**
6.   **if** L[i] ≤ R[j]
7.   **then** A[k] ← L[i]
8.   i ←i + 1
9.   **else** A[k] ← R[j]
10.   j ← j + 1



1.2. Thời gian tính

- Thời gian tính của bước trộn
  - Khởi tạo (tạo 2 mảng con tạm thời L và R):
    - $\Theta(n_1 + n_2) = \Theta(n)$
  - Đưa các phần tử vào mảng kết quả (vòng lặp **for** cuối cùng):
    - n lần lặp, mỗi lần đòi hỏi thời gian hằng số $\Rightarrow \Theta(n)$
  - Tổng cộng thời gian của trộn là:
    - $\Theta(n)$

1.2. Thời gian tính

- Thời gian tính của sắp xếp trộn
  - **Chia:**
    - tính q như là giá trị trung bình của p và r: $D(n) = \Theta(1)$
  - **Trị:**
    - giải đệ qui 2 bài toán con, mỗi bài toán kích thước n/2 $\Rightarrow$ 2T (n/2)
  - **Tổ hợp:**
    - TRỘN (MERGE) trên các mảng con cỡ n phần tử đòi hỏi thời gian $\Theta(n)$ $\Rightarrow$ $C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{nếu } n = 1 \\ 2T(n/2) + \Theta(n) & \text{nếu } n > 1 \end{cases}$$

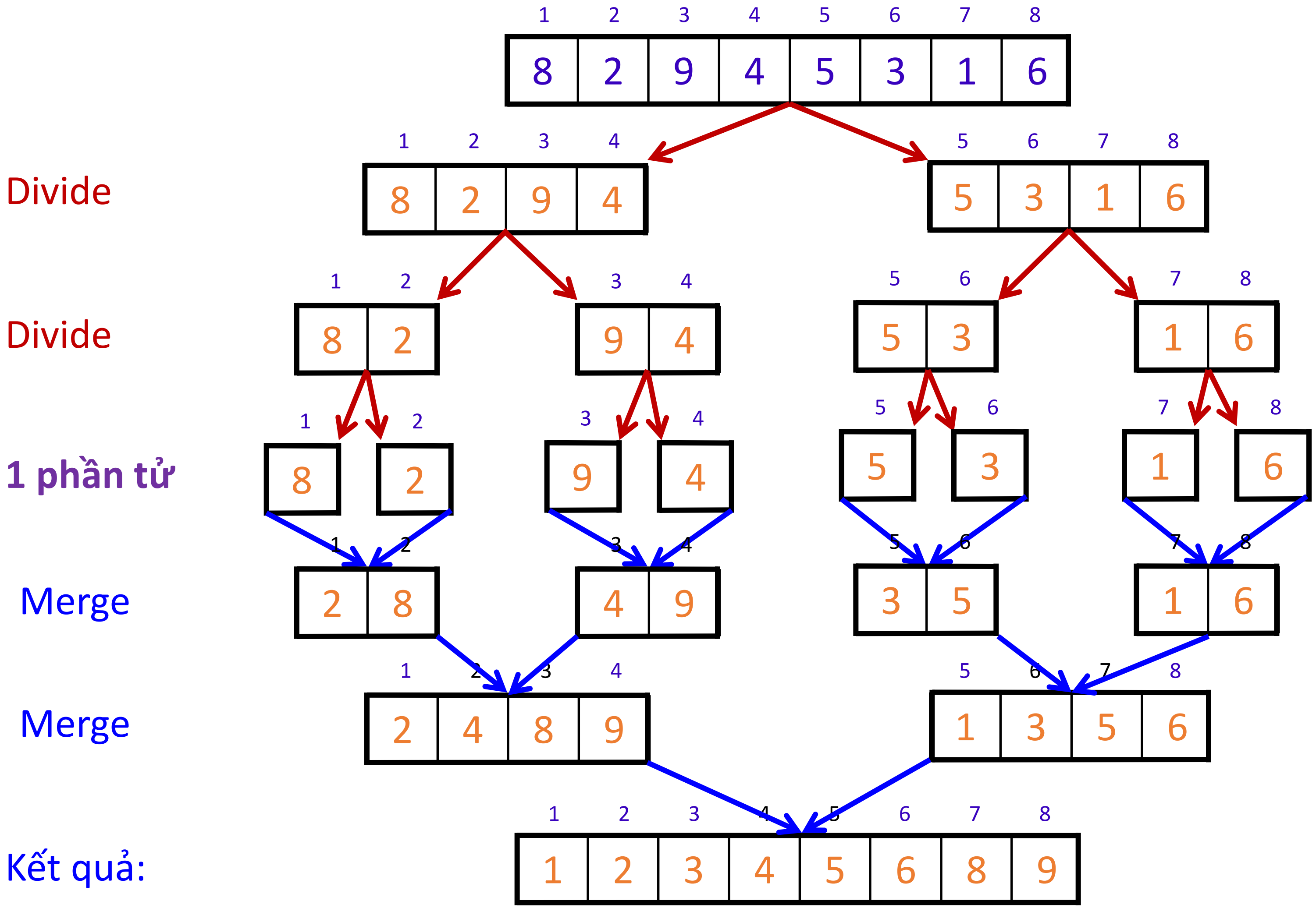
Suy ra theo định lý thợ: *T*(*n*) = Θ(*n* log *n*)

HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

1.3. Ví dụ


8 2 9 4 5 3 1 6
Divide
8 2 9 4
5 3 1 6
Divide
8 2
9 4
5 3
1 6
1 phần tử
8
2
9
4
5
3
1
6
Merge
2 8
4 9
3 5
1 6
Merge
2 4 8 9
1 3 5 6
Kết quả:
1 2 3 4 5 6 8 9

NỘI DUNG TIẾP THEO

1. Sắp xếp trộn (merge sort)
2. **Cài đặt thuật toán**

2. CÀI ĐẶT THUẬT TOÁN

- Cài đặt merge: Trộn A[first..mid] và A[mid+1.. last]

```
void merge(DataType A[], int first, int mid, int last){   DataType tempA[MAX_SIZE];
  // mảng phụ
  int first1 = first; int last1 = mid;
  int first2 = mid + 1; int last2 = last; int index = first1;
  for (; (first1 <= last1) && (first2 <= last2); ++index){
     if (A[first1] < A[first2])     {
       tempA[index] = A[first1]; ++first1;}
     else
     {  tempA[index] = A[first2]; ++first2;}   }
  for (; first1 <= last1; ++first1, ++index)
    tempA[index] = A[first1]; // sao nốt dãy con 1
  for (; first2 <= last2; ++first2, ++index)
    tempA[index] = A[first2]; // sao nốt dãy con 2
  for (index = first; index <= last; ++index)
    A[index] = tempA[index]; // sao trả mảng kết quả
} // end merge
```

Chú ý: DataType: kiểu dữ liệu phần tử mảng.

2. CÀI ĐẶT THUẬT TOÁN

- Cài đặt mergesort

```
void mergesort(DataType A[], int first, int last)
{
   if (first < last)
   {  // chia thành hai dãy con
      int mid = (first + last)/2;    // chỉ số điểm giữa
      // sắp xếp dãy con trái A[first..mid]
      mergesort(A, first, mid);
      // sắp xếp dãy con phải A[mid+1..last]
      mergesort(A, mid+1, last);
     // Trộn hai dãy con
     merge(A, first, mid, last);
  }  // end if
}  // end mergesort
```

Lệnh gọi thực hiện mergesort(A,0,n-1)

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Chương 7 – Sắp xếp

Bài 4. Sắp xếp nhanh (quick sort)

ONE LOVE. ONE FUTURE.

NỘI DUNG TIẾP THEO

1. Sơ đồ tổng quát

2. Phép phân đoạn

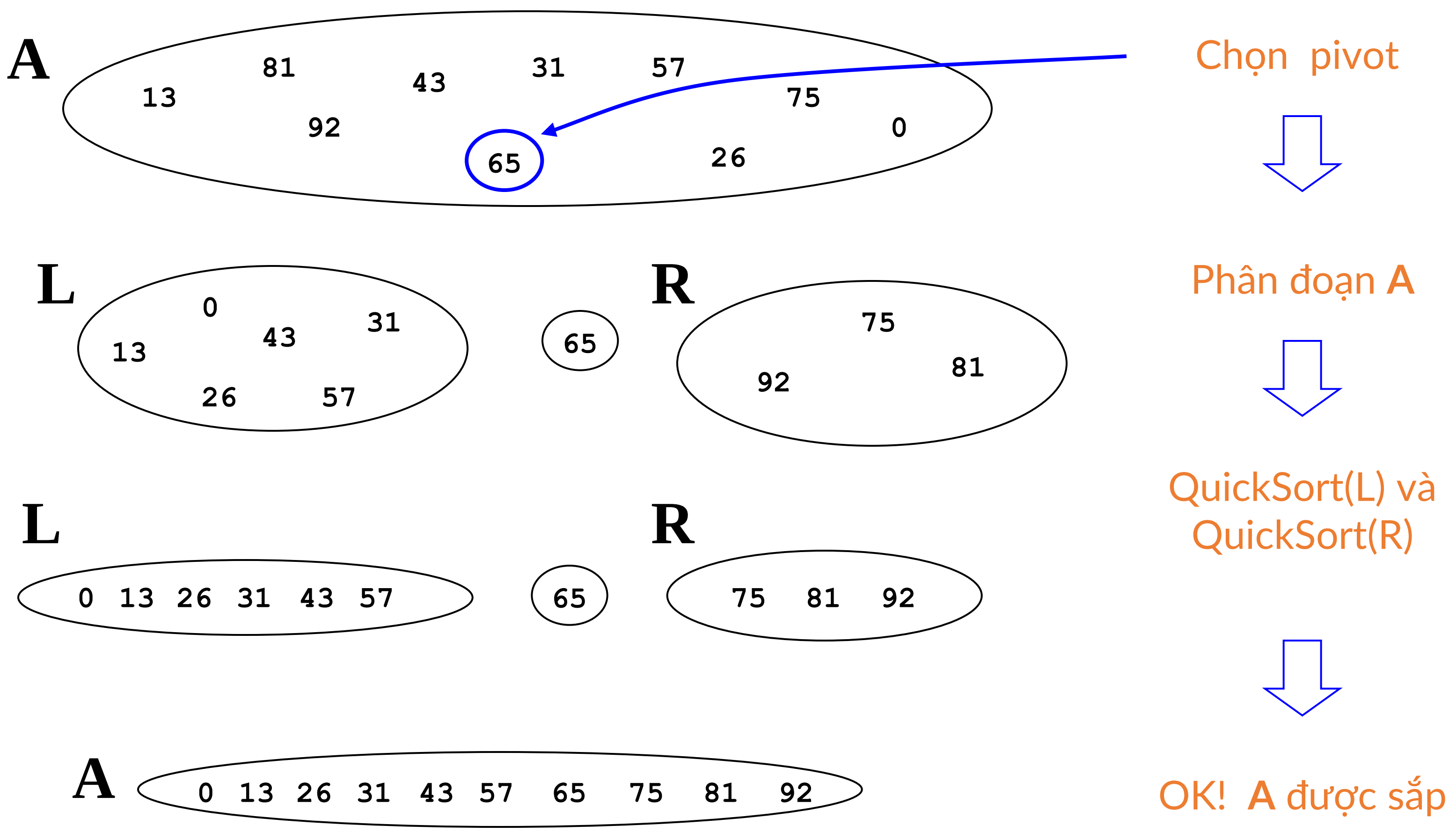
3. Độ phức tạp của sắp xếp nhanh

1. SƠ ĐỒ TỔNG QUÁT

- Sơ đồ Quick Sort

1. **Neo đệ qui**: Nếu dãy chỉ còn không quá 1 phần tử thì nó là dãy được sắp và trả lại ngay dãy này mà không phải làm gì cả.
2. **Chia**:
   - Chọn 1 phần tử trong dãy và gọi nó là phần tử chốt ***p*** (pivot).
   - Chia dãy đã cho ra thành 2 dãy con:
     - Dãy con trái (*L*) gồm những phần tử ≤ phần tử chốt,
     - Dãy con phải (*R*) gồm các phần tử > phần tử chốt.
     - Thao tác này được gọi là "Phân đoạn" (Partition).
3. **Trị**: Lặp lại một cách đệ qui thuật toán đối với 2 dãy con *L* và *R*.
4. **Tổng hợp** (Combine): Dãy được sắp xếp là *L p R*.

1. SƠ ĐỒ TỔNG QUÁT



1. SƠ ĐỒ TỔNG QUÁT

➢ Sơ đồ tổng quát của Quick Sort

Quick-Sort(*A*, *Left*, *Right*)

1. **if** (*Left* < *Right*) {
2. *Pivot* = Partition(*A*, *Left*, *Right*);
3. Quick-Sort(*A*, *Left*, *Pivot* –1);
4. Quick-Sort(*A*, *Pivot* +1, *Right*); }

- Hàm Partition(*A*, *Left*, *Right*) thực hiện chia *A*[*Left*..*Right*] thành 2 đoạn *A*[*Left*..*Pivot* –1] và *A*[*Pivot*+1..*Right*] sao cho:
  - Các phần tử trong *A*[*Left*..*Pivot* –1] ≤ *A*[*Pivot*]
  - Các phần tử trong *A*[*Pivot*+1..*Right*] ≥ *A*[*Pivot*]
- Lệnh gọi thực hiện thuật toán Quick-Sort(*A*, 1, *n*)

- Khi dãy con chỉ còn một số lượng không lớn phần tử (VD: 9 phần tử)

→ sử dụng các thuật toán đơn giản để sắp xếp dãy này, chứ không nên tiếp tục chia nhỏ.

- Thuật toán trong tình huống này:

Quick-Sort(*A*, *Left*, *Right*)

if (*Right* - *Left* < n_0)

    Insertion_sort(*A*, *Left*, *Right*);

else {

    Pivot = Partition(*A*, *Left*, *Right*);

    Quick-Sort(*A*, *Left*, *Pivot* –1);

    Quick-Sort(*A*, *Pivot* +1, *Right*); }

NỘI DUNG TIẾP THEO

1. Sơ đồ tổng quát

2. Phép phân đoạn

2.1. Chọn phần tử chốt

2.2. Phần tử chốt là phần tử đứng đầu

2.3. Phần tử chốt là phần tử đứng giữa

2.4. Phần tử chốt là phần tử đứng cuối

3. Độ phức tạp của sắp xếp nhanh

2. PHÉP PHÂN ĐOẠN

- Trong QS thao tác chia bao gồm 2 công việc:
  - Chọn phần tử chốt ***p***.
  - Phân đoạn: Chia dãy đã cho ra thành 2 dãy con
- Thao tác phân đoạn có thể cài đặt (tại chỗ) với thời gian $\Theta(n)$.
- Hiệu quả của thuật toán phụ thuộc rất nhiều vào việc phần tử nào được chọn làm phần tử chốt:
  - Thời gian tính trong tình huống tồi nhất của QS là $O(n^2)$.
    - Xảy ra khi danh sách là đã được sắp xếp và phần tử chốt được chọn là phần tử trái nhất của dãy.
  - Nếu phần tử chốt được chọn ngẫu nhiên -> QS có độ phức tạp tính toán là $O(n \log n)$.

2. PHÉP PHÂN ĐOẠN

- 2.1. Chọn phần tử chốt

  - Có vai trò quyết định đối với hiệu quả của thuật toán.
  - Tốt nhất: *phần tử đứng giữa trong danh sách được sắp xếp* (***trung vị/median***)
    - Sau $\log_2 n$ lần phân đoạn sẽ đạt tới danh sách với kích thước bằng 1.
    - Tuy nhiên khó thực hiện.

2. PHÉP PHÂN ĐOẠN

- 2.1. Chọn phần tử chốt
  - Thường sử dụng các cách chọn phần tử:
    - Trái nhất (đứng đầu).
    - Phải nhất (đứng cuối).
    - Đứng giữa danh sách.
    - Trung vị trong 3 phần tử đứng đầu, đứng giữa và đứng cuối.
    - Ngẫu nhiên một phần tử.

2. PHÉP PHÂN ĐOẠN

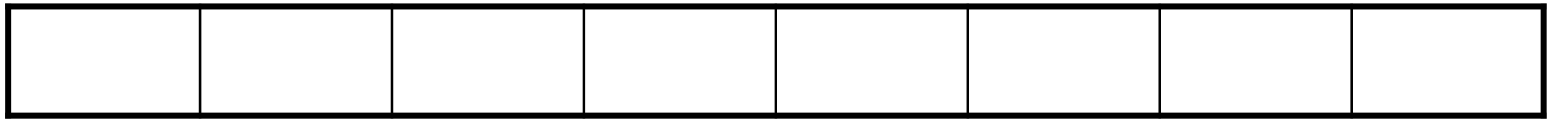
- 2.1. Chọn phần tử chốt
  - Xây dựng hàm **Partition(a, left, right)** làm việc sau:
  - **Input:** Mảng *a*[*left* .. *right*].
  - **Output:** Phân bố lại các phần tử của mảng đầu vào và trả lại chỉ số *jpivot* thoả mãn:
    - *a*[*jpivot*] chứa giá trị ban đầu của *a*[*left*],
    - *a*[*i*] ≤ *a*[*jpivot*], với mọi *left* ≤ *i* < *pivot*,
    - *a*[*j*] ≥ *a*[*jpivot*]), với mọi *pivot* < *j* ≤ *right*.

- 2.2. Phần tử chốt là phần tử đứng đầu

```
Partition(a, left, right)
    i = left; j = right + 1; pivot = a[left];
    while i < j do
        i = i + 1;
        while i ≤ right and a[i] < pivot do  i = i + 1;
        j = j - 1;
        while j ≥ left and a[j] > pivot do j = j - 1;
        swap(a[i] , a[j]);
    swap(a[i], a[j]); swap(a[j], a[left]);
    return j;
```

pivot được chọn là phần tử đứng đầu

j là chỉ số (jpivot) cần trả lại, do đó cần đổi chỗ a[left] và a[j]



i

Sau khi chọn *pivot*, dịch các con trỏ *i* và *j* từ đầu và cuối mảng và đổi chỗ cặp phần tử thoả mãn a[*i*] > *pivot* và a[*j*] < *pivot*

j

2.2. Phần tử chốt là phần tử đứng đầu

- Ví dụ 1

Vị trí:	0	1	2	3	4	5	6	7	8	9
Khoá (Key):	<u>**9**</u>	1	11	17	13	18	4	12	14	5
		>	>							<
lần 1×**while**:	<u>**9**</u>	1	5	17	13	18	4	12	14	11
				>			<	<	<	
lần 2×**while**:	<u>**9**</u>	1	5	4	13	18	17	12	14	11
				<	><	<				
lần 3×**while**:	<u>**9**</u>	1	5	13	4	18	17	12	14	11
2 lần đổi chỗ:	4	1	5	<u>**9**</u>	13	18	17	12	14	11

2.2. Phần tử chốt là phần tử đứng đầu

- Ví dụ 1

Chọn pivot:

7	2	8	3	5	9	6

Phân đoạn: Con trỏ

7	2	8	3	5	9	6

< ↑ ... > ↑

2 nhỏ hơn pivot

7	2	8	3	5	9	6

< ↑ ... > ↑

đổi chỗ 6, 8

7	2	6	3	5	9	8

< ↑ ... > ↑

3,5 nhỏ hơn 9 lớn hơn

7	2	6	3	5	9	8

Kết thúc phân đoạn

7	2	6	3	5	9	8

Đưa pivot vào vị trí

5	2	6	3	7	9	8

2. PHÉP PHÂN ĐOẠN

- 2.3. Phần tử chốt là phần tử đứng giữa

```
PartitionMid(a, left, right);
  i = left; j = right; pivot = a[(left + right)/2];
  repeat
      while a[i] < pivot do i = i + 1;
      while pivot < a[j] do j = j - 1;
      if  i <= j
          swap(a[i], a[j]);
          i = i + 1; j = j - 1;
  until i > j;
  return j;
```

pivot được chọn là phần tử đứng giữa

2.4. Phần tử chốt là phần tử đứng cuối

```
int PartitionR(int a[], int p, int r) {
  int x = a[r];
  int j = p - 1;
  for (int i = p; i < r; i++) {
     if (x >= a[i])
        {  j = j + 1;  swap(a[i], a[j]); }
  }
  a[r] = a[j + 1];  a[j + 1] = x;
  return (j + 1);
}
```

```
void quickSort(int a[], int p, int r) {
  if (p < r) {
     int q = PartitionR(a, p, r);
     quickSort(a, p, q - 1);
     quickSort(a, q + 1, r);
  }
}
```

2. PHÉP PHÂN ĐOẠN

- Cài đặt QUICK SORT

```
void swap(int &a,int &b)
{ int t = a; a = b; b = t; }
```

```
int Partition(int a[], int L, int R)
{
  int i, j, p;
  i = L;  j = R + 1;  p = a[L];
  while (i < j) {
      i = i + 1;
      while ((i <= R)&&(a[i]<p))  i++;
      j--;
      while ((j >= L)&& (a[j]>p)) j--;
      swap(a[i] , a[j]);
  }
  swap(a[i], a[j]);   swap(a[j], a[L]);
  return j;
}
```

```
void quick_sort(int a[], int left, int right)
{
  int p;
  if (left < right)
  {
     pivot = Partition(a, left, right);
     if (left < pivot)
           quick_sort(a, left, pivot-1);
     if (right > pivot)
           quick_sort(a, pivot+1, right);}
}
```

NỘI DUNG TIẾP THEO

1. Sơ đồ tổng quát
2. Phép phân đoạn
3. **Độ phức tạp của sắp xếp nhanh**
   3.1. Phân đoạn không cân bằng
   3.2. Phân đoạn hoàn hảo - Perfect partition
   3.3. Phân đoạn cân bằng

3. ĐỘ PHỨC TẠP CỦA SẮP XẾP NHANH

- Phụ thuộc vào việc phép phân chia là ***cân bằng (balanced)*** hay ***không (unbalanced)*** → phụ thuộc vào việc chọn phần tử chốt.

1. **Phân đoạn không cân bằng:** một bài toán con có kích thước $n - 1$ còn bài toán kia có kích thước 0.
2. **Phân đoạn hoàn hảo (Perfect partition):** việc phân đoạn luôn được thực hiện dưới dạng phân đôi → mỗi bài toán con có kích thước cỡ $n/2$.
3. **Phân đoạn cân bằng:** việc phân đoạn được thực hiện ở đâu đó quanh điểm giữa → một bài toán con có kích thước $n - k$ còn bài toán kia có kích thước k.

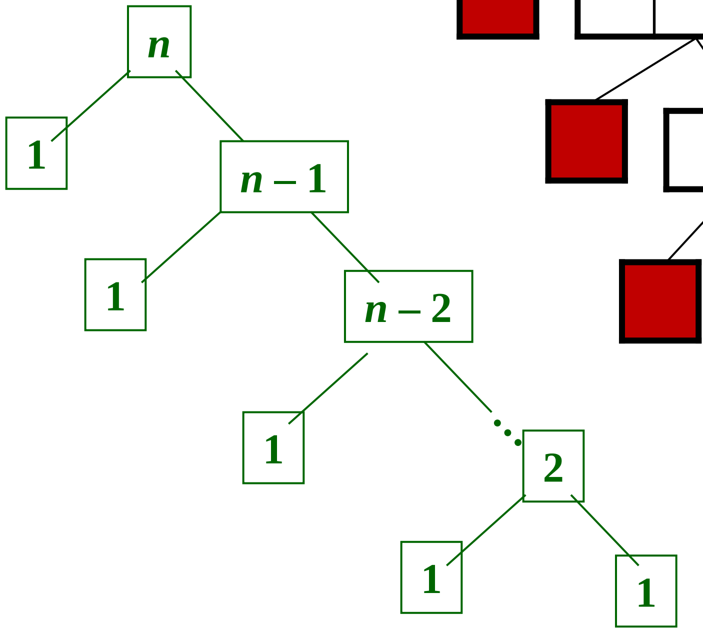
- 3.1. Phân đoạn không cân bằng

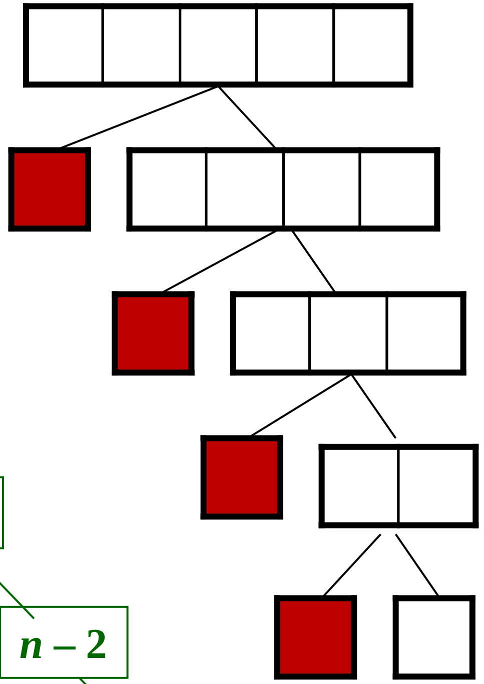
  Công thức đệ qui là:

  $T(n) = T(n - 1) + T(0) + \Theta(n)$

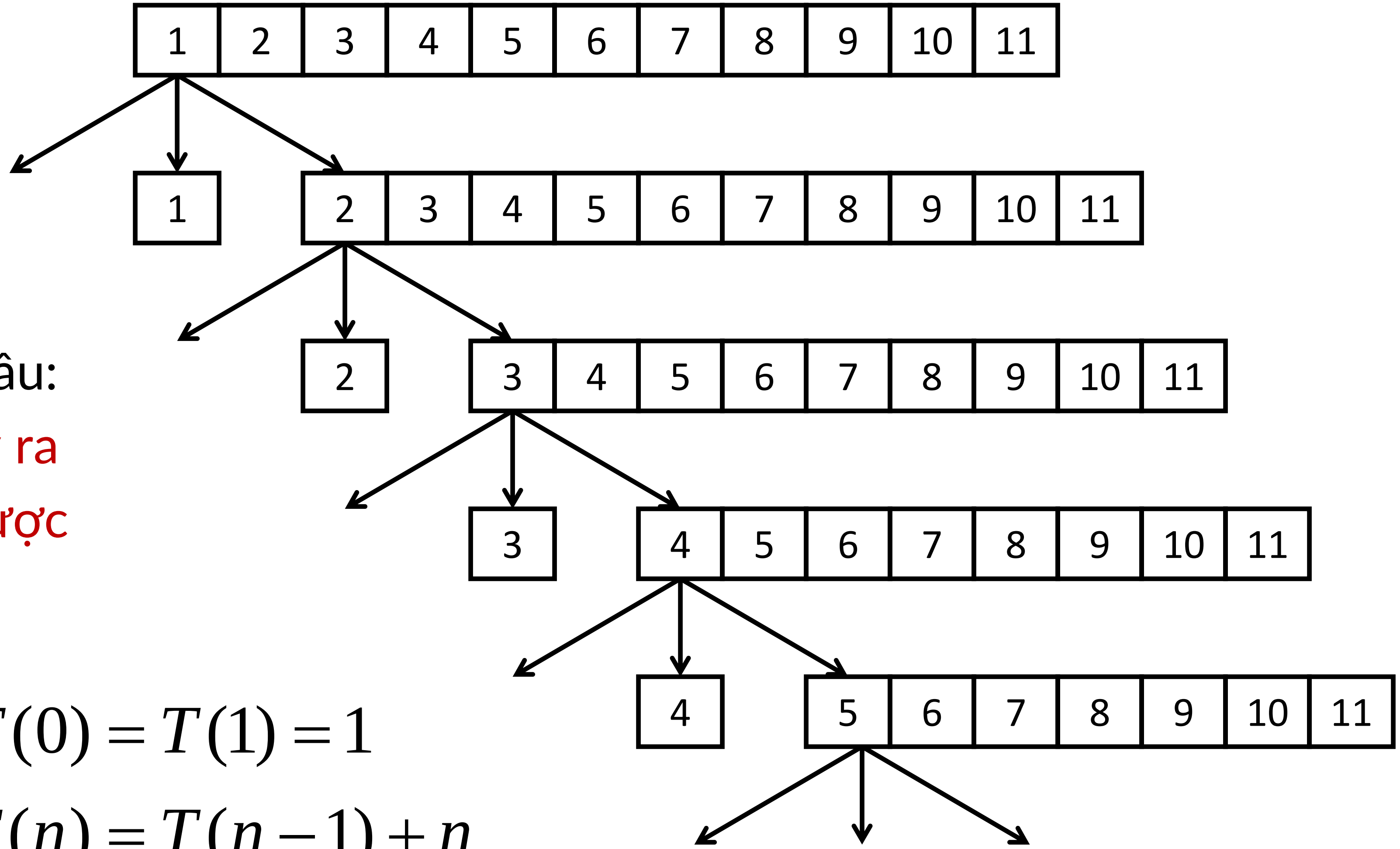
$$
\left.
\begin{aligned}
T(0) &= T(1) = 1 \\
T(n) &= \cancel{T(n-1)} + n \\
\cancel{T(n-1)} &= \cancel{T(n-2)} + (n-1) \\
\cancel{T(n-2)} &= \cancel{T(n-3)} + (n-2) \\
&\dots \\
\cancel{T(2)} &= T(1) + (2)
\end{aligned}
\right\}
$$

$$
\begin{aligned}
T(n) &= T(1) + \sum_{i=2}^{n} i \\
&= O(n^2)
\end{aligned}
$$





- 3.1. Phân đoạn không cân bằng



  - Khi phần tử chốt được chọn là phần tử đứng đầu: Tình huống tồi nhất xảy ra khi dãy đầu vào là đã được sắp xếp

$$T(0) = T(1) = 1$$

$$T(n) = T(n-1) + n$$

- 3.2. Phân đoạn hoàn hảo - Perfect partition

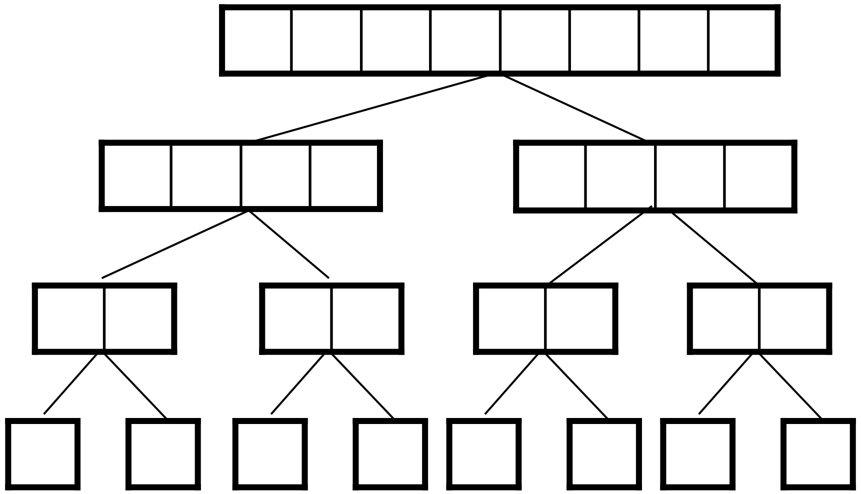
Công thức đệ qui:

$T(n) = T(n/2) + T(n/2) + \Theta(n)$

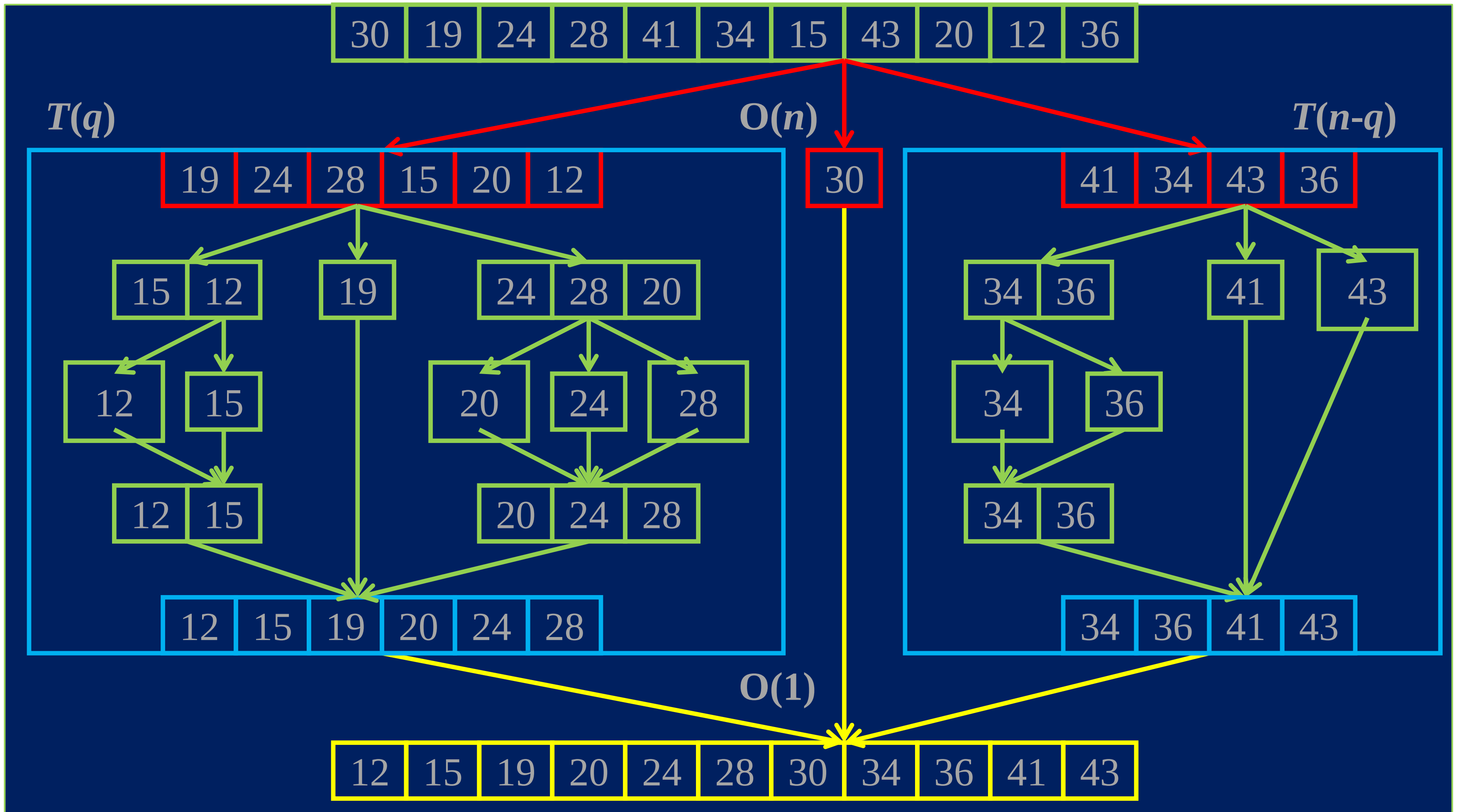
$$T(n) = 2T(n/2) + \Theta(n)$$

$$T(n) = \Theta(n \log n)$$

Theo định lý thợ !



3.3. Phân đoạn cân bằng



$$T(0) = T(1) = 1$$

$$T(n) = T(q) + T(n-q) + O(n) + O(1)$$

3.3. Phân đoạn cân bằng

- Giả sử rằng pivot được chọn ngẫu nhiên trong số các phần tử của dãy đầu vào
- Tất cả các tình huống sau đây là đồng khả năng:
  - Pivot là phần tử nhỏ nhất trong dãy
  - Pivot là phần tử nhỏ nhì trong dãy
  - Pivot là phần tử nhỏ thứ ba trong dãy

  …
  - Pivot là phần tử lớn nhất trong dãy
  - Pivot là phần tử đầu tiên

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Chương 7 – Sắp xếp

Bài 5. Sắp xếp vun đống (heap sort)

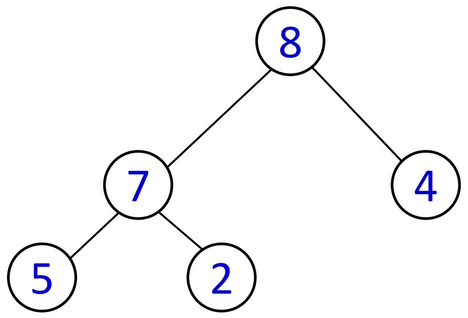
ONE LOVE. ONE FUTURE.

NỘI DUNG TIẾP THEO

1. **Cấu trúc dữ liệu đống**
2. Sắp xếp vun đống

- Định nghĩa: **Đống (heap)** là cây nhị phân *hoàn chỉnh* có 2 tính chất:
  - **Tính cấu trúc:** tất cả các mức đều là đầy, ngoại trừ mức cuối cùng, mức cuối được điền từ trái sang phải.
  - **Tính có thứ tự hay tính chất đống :** với mỗi nút *x*

    Parent(*x*) ≥ *x*.
- Cây được cài đặt bởi mảng *A*[*i*] có độ dài *length*[*A*]. Số lượng phần tử là *heapsize*[*A*]



Heap

Từ tính chất đống suy ra:

"Gốc chứa phần tử lớn nhất của đống!"

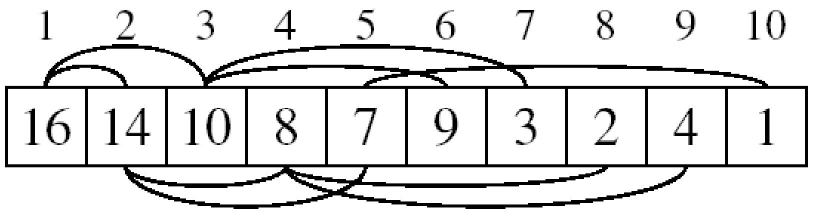
Như vậy có thể nói:

Đống là cây nhị phân được điền theo thứ tự

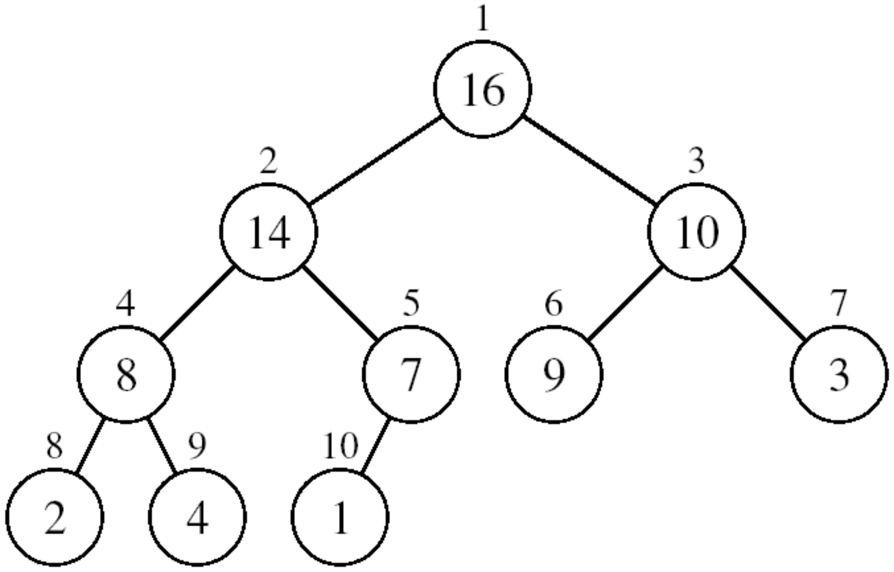
1. CẤU TRÚC DỮ LIỆU ĐỐNG

➢ Biểu diễn đống bởi mảng

- Đống có thể cất giữ trong mảng *A*.
  - 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[$\lfloor i/2 \rfloor$]
  - Heapsize[A] ≤ length[A]
- Các phần tử trong mảng con A[($\lfloor n/2 \rfloor$+1) .. n] là các lá

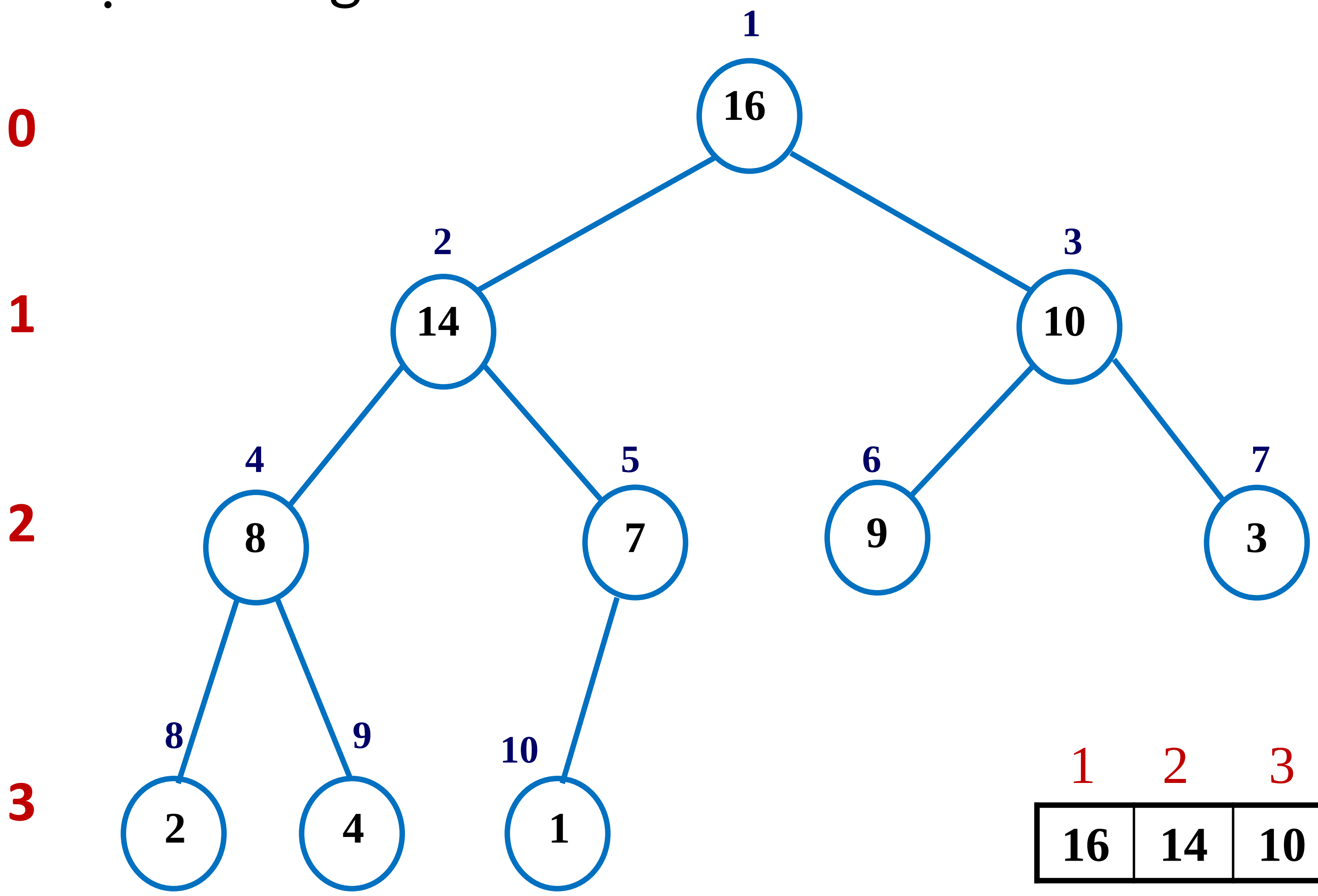


Mảng đầu vào



Đống tương ứng của mảng

- Ví dụ 1: Đống



parent(i) = ⌊i/2⌋

left-child(i) = 2i

right-child(i) = 2i +1

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

1. CẤU TRÚC DỮ LIỆU ĐỐNG

- Hai dạng đống
  - **Đống max - Max-heaps** (Phần tử lớn nhất ở gốc), có tính chất *max-heap*:
    - với mọi nút i, ngoại trừ gốc:

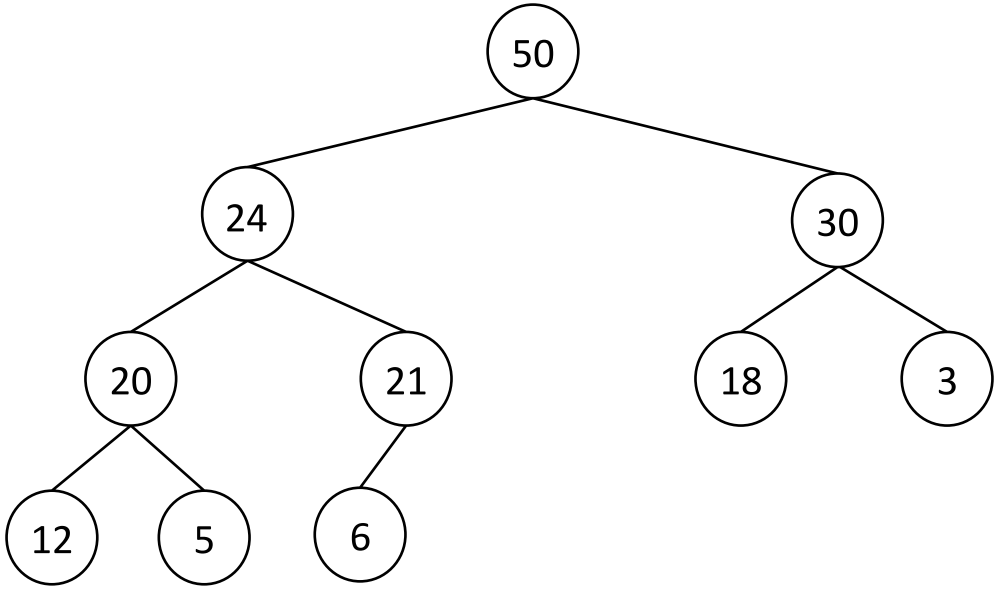
      A[parent(i)] ≥ A[i]
  - **Đống min - Min-heaps** (phần tử nhỏ nhất ở gốc), có tính chất *min-heap*:
    - với mọi nút i, ngoại trừ gốc:

      A[parent(i)] ≤ A[i]
  - Phần dưới đây ta sẽ chỉ xét đống max (max-heap). Đống min được xét hoàn toàn tương tự.

HUST

1. CẤU TRÚC DỮ LIỆU ĐỐNG

- Hai dạng đống
  - Nút mới được bổ sung vào mức đáy (từ trái sang phải)
  - Các nút được loại bỏ khỏi mức đáy (từ phải sang trái)

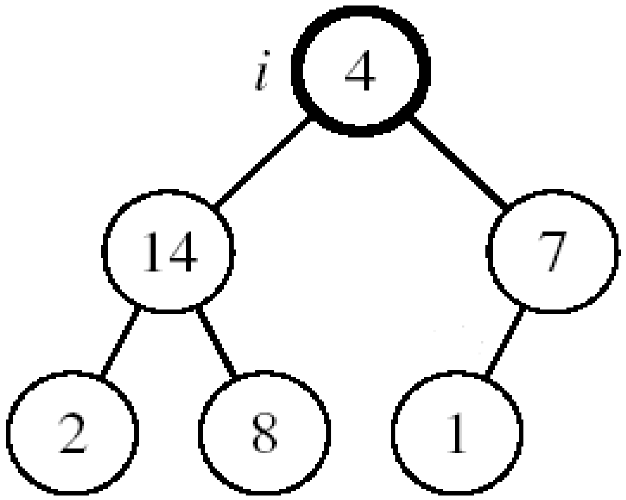


Ví dụ đống cực đại

1. CẤU TRÚC DỮ LIỆU ĐỐNG

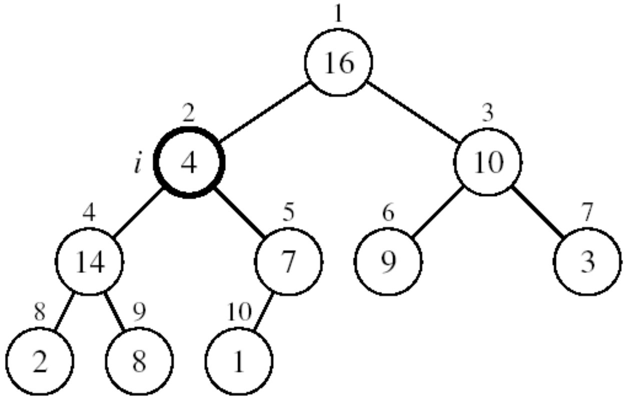
- Các phép toán đối với đống
  - Khôi phục tính chất max-heap (Vun lại đống)
    - Max-Heapify
  - Tạo max-heap từ một mảng không được sắp xếp
    - Build-Max-Heap

- Khôi phục tính chất đống
  - Giả sử có nút *i* với giá trị bé hơn con của nó
    - Giả thiết là: 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 đến khi nút không còn bé hơn con



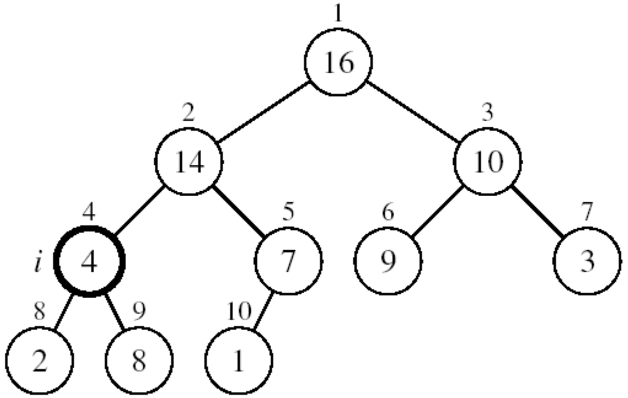
Nút i vi phạm tính chất đống

- Ví dụ 2



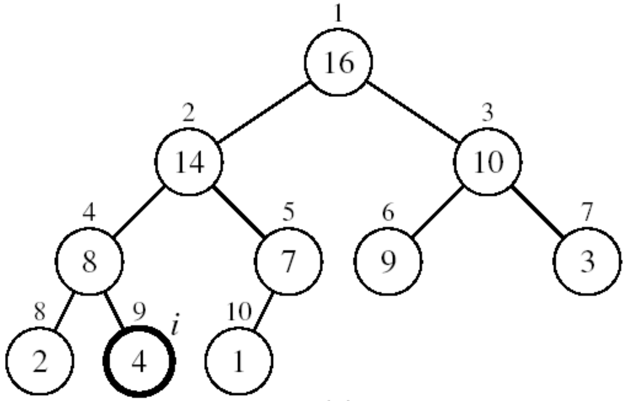
A[2] vi phạm tính chất đống

A[2] ↔ A[4]



A[4] vi phạm

A[4] ↔ A[9]

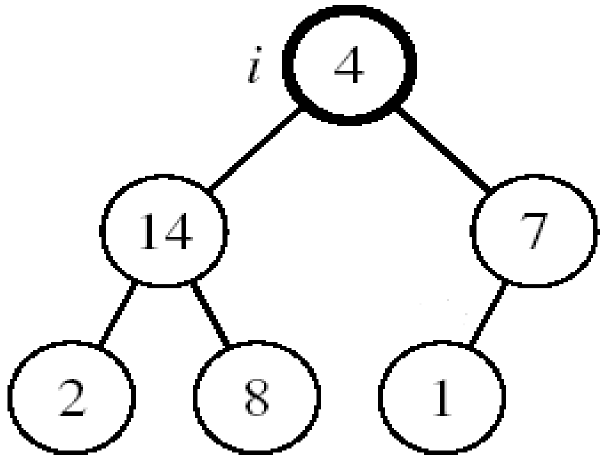


Tính chất đống được khôi phục

1. CẤU TRÚC DỮ LIỆU ĐỐNG

➢ Thuật toán khôi phục tính chất đống

- Giả thiết:
  - Cả hai cây con trái và phải của i đều là max-heaps
  - A[i] có thể bé hơn các con của nó




```
Max-Heapify(A, i, n)
// n = heapsize[A]
    l ← left-child(i)
    r ← right-child(i)
    if (l ≤ n) and (A[l] > A[i])
        then largest ← l
        else largest ← i
    if (r ≤ n) and (A[r] > A[largest])
        then largest ← r
    if largest != i
        then Exchange(A[i],A[largest])
    Max-Heapify(A,largest,n)
```

1. CẤU TRÚC DỮ LIỆU ĐỐNG

- Thời gian tính của MAX-HEAPIFY
  - Nhận thấy rằng:
    - Từ nút i phải di chuyển theo đường đi xuống phía dưới của cây. Độ dài của đường đi này không vượt quá độ dài đường đi từ gốc đến lá, nghĩa là không vượt quá *h*.
    - Ở mỗi mức phải thực hiện 2 phép so sánh.
    - Do đó tổng số phép so sánh không vượt quá 2*h*.
    - Vậy, thời gian tính là $O(h)$ hay $O(\log n)$.
  - Kết luận: Thời gian tính của MAX-HEAPIFY là $O(\log n)$
  - Nếu viết trong ngôn ngữ chiều cao của đống, thì thời gian này là $O(h)$

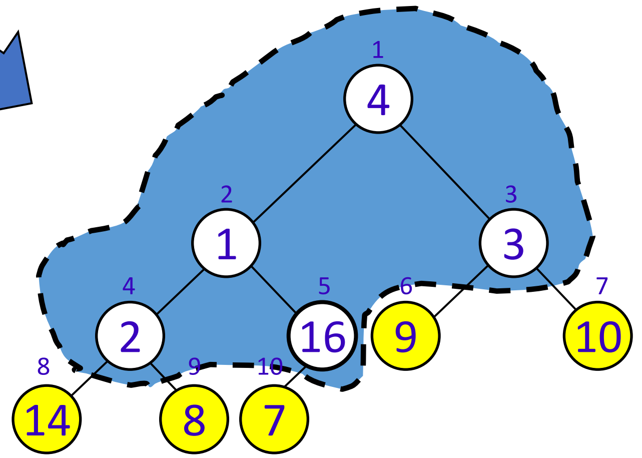
1. CẤU TRÚC DỮ LIỆU ĐỐNG

➢ Thời gian tính của MAX-HEAPIFY

- Biến đổi mảng A[1 … n] thành max-heap (n = length[A])
- Vì các phần tử của mảng con A[(⌊n/2⌋+1) .. n] là các lá
- Do đó để tạo đống ta chỉ cần áp dụng MAX-HEAPIFY đối với các phần tử từ 1 đến ⌊n/2⌋

Alg: Build-Max-Heap(A)

1. n = length[A]
2. **for** i ← ⌊n/2⌋ **downto** 1
3. **do** Max-Heappify(A, i, n)



A:

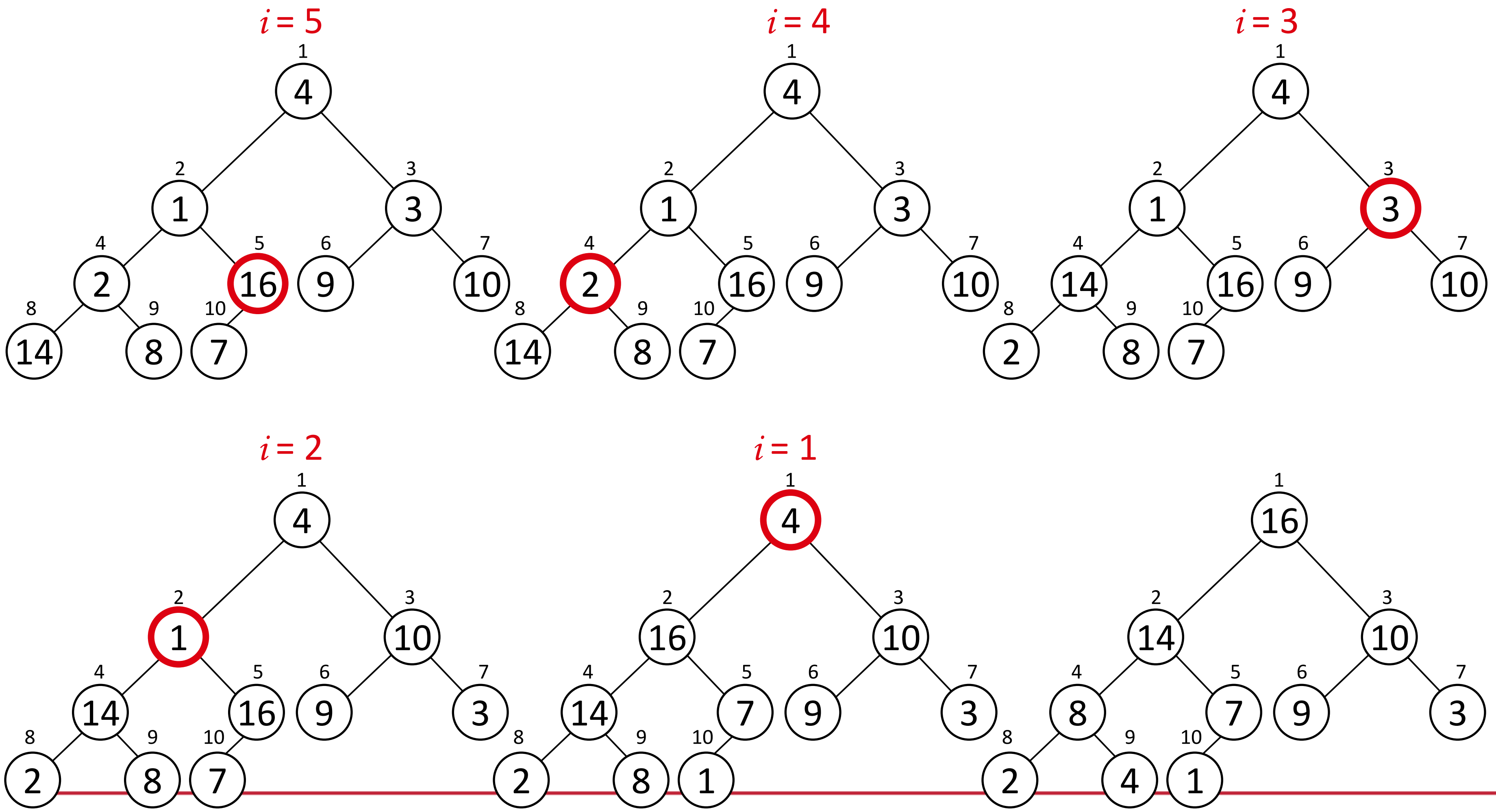
4	1	3	2	16	9	10	14	8	7

1. CẤU TRÚC DỮ LIỆU ĐỐNG

- Ví dụ 3

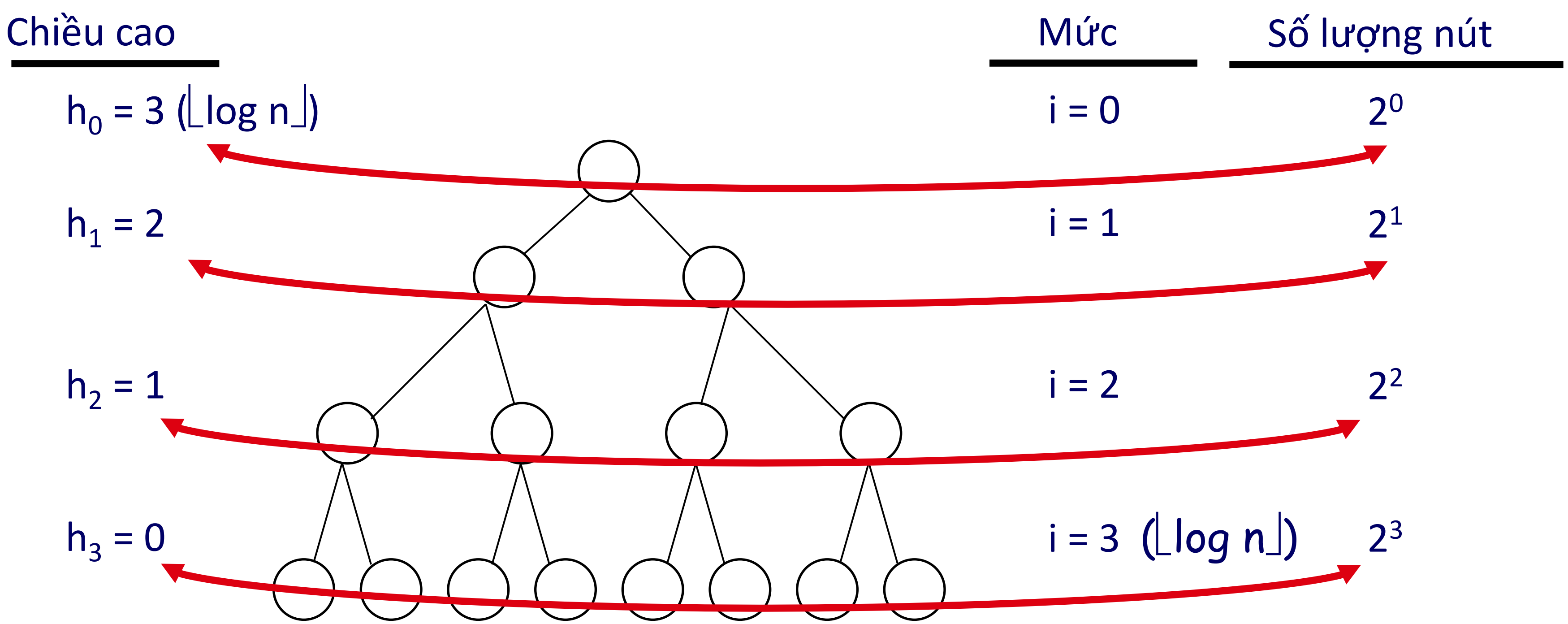
A:

4	1	3	2	16	9	10	14	8	7



- Thời gian tính của Buid-Max-Heap
  - Heapify đòi hỏi thời gian $O(h)$ $\Rightarrow$ chi phí của Heapify ở nút *i* là tỷ lệ với chiều cao của nút *i* trên cây

$$\Rightarrow T(n) = \sum_{i=0}^{h} n_i h_i = \sum_{i=0}^{h} 2^i (h-i) = O(n)$$



$h_i = h - i$ chiều cao của đống gốc tại mức i

$n_i = 2^i$ số lượng nút ở mức i

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu đống
2. **Sắp xếp vun đống**

2. SẮP XẾP VUN ĐỐNG

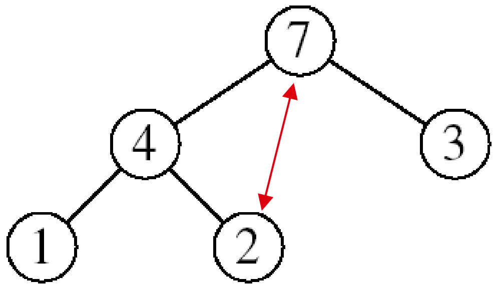
➢ Sơ đồ của thuật toán:

- Tạo đống **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 đống đi 1
- Thực hiện Max-Heapify đối với gốc mới
- Lặp lại quá trình cho đến khi đống chỉ còn 1 nút

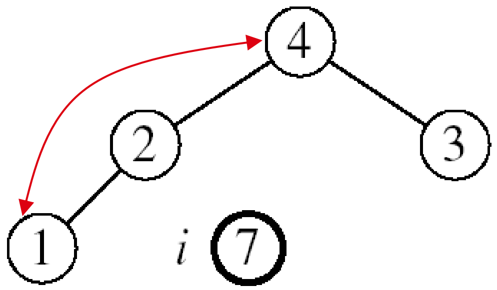
2. SẮP XẾP VUN ĐỐNG

- Ví dụ 4

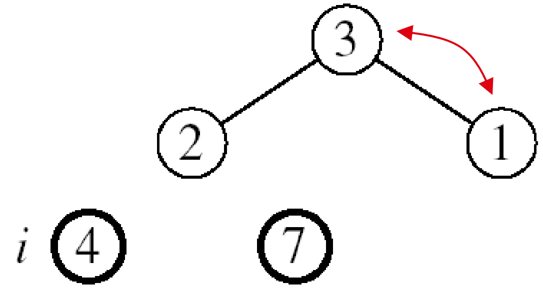
A=[7, 4, 3, 1, 2]



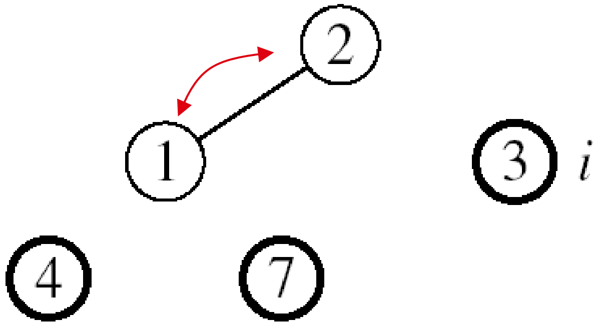
Max-Heapyfy(A, 1, 4)

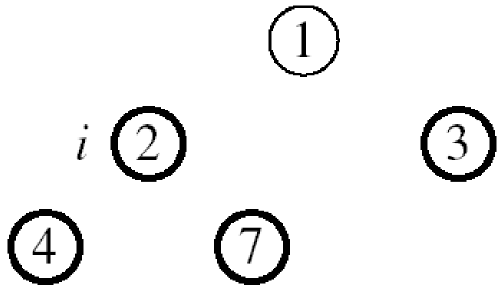


Max-Heapyfy(A, 1, 3)



Max-Heapyfy(A, 1, 2)





A	1	2	3	4	7

Max-Heapyfy(A, 1, 1)

2. SẮP XẾP VUN ĐỐNG

- *Algorithm:* HeapSort(A)

1. Build-Max-Heap(A) — $O(n)$
2. **for** i ← length[A] **downto** 2
3. **do** exchange A[1] ↔ A[i] — $O(\log n)$
4. Max-Heapify(A, 1, i - 1)

n-1 lần lặp

- Thời gian tính: $O(n \log n)$

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !