2.Minh họa Thuật Toán Heap Sort

Bài toán

- Hãy sắp xếp mảng tăng dần bằng thuật toán Heap Sort.

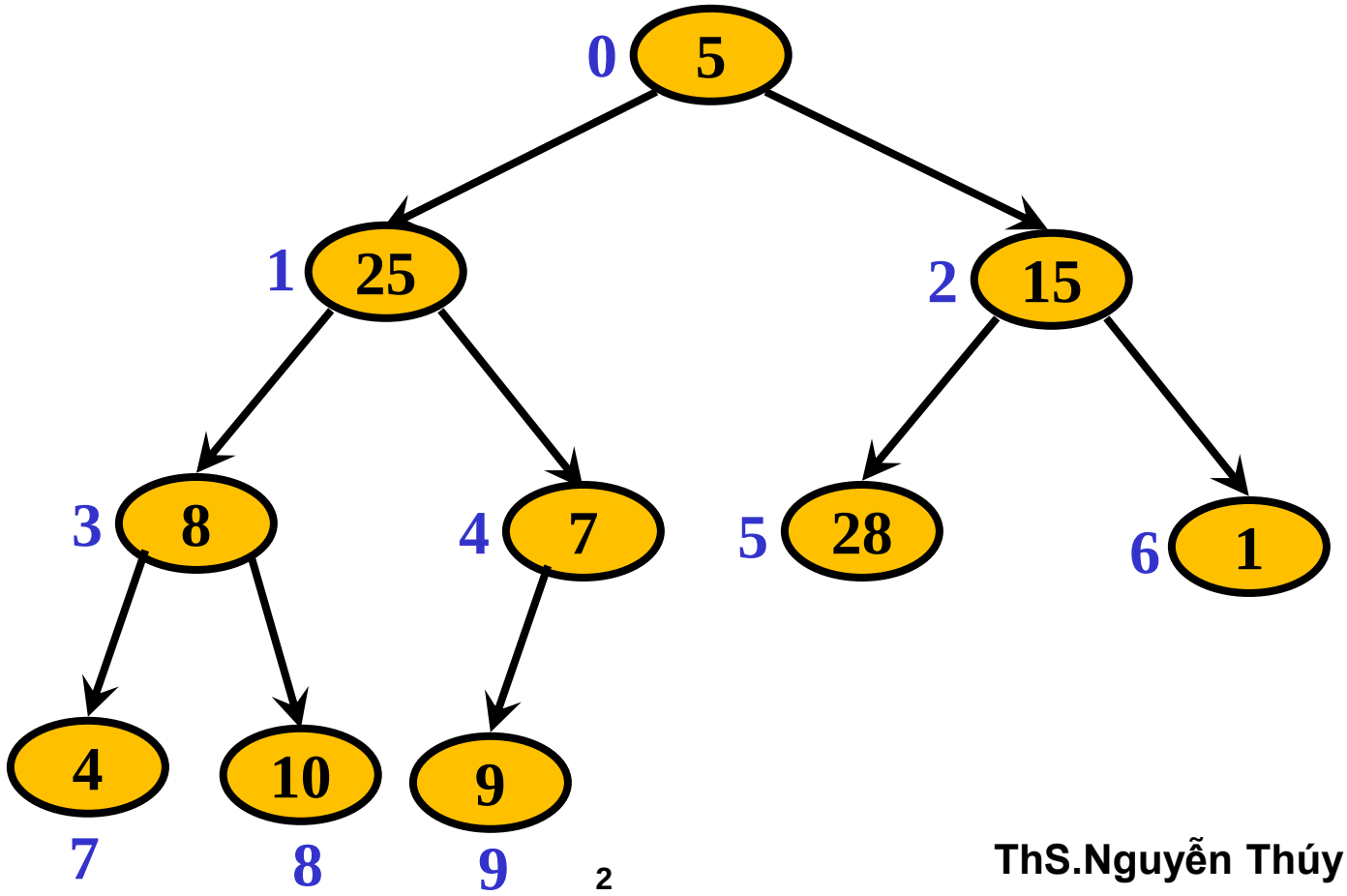
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

- **Heap:** Là một dãy các phần tử $a_l , a_2 , \ldots , a_r$ thoả các quan hệ với mọi $i \in [l, r]$:
  - $a_i \geq a_{2i+1}$
  - $a_i \geq a_{2i+2}$ // (a_i , a_{2i+1}), (a_i , a_{2i+2}) là các cặp phần tử liên đới

ThS.Nguyễn Thúy Loan

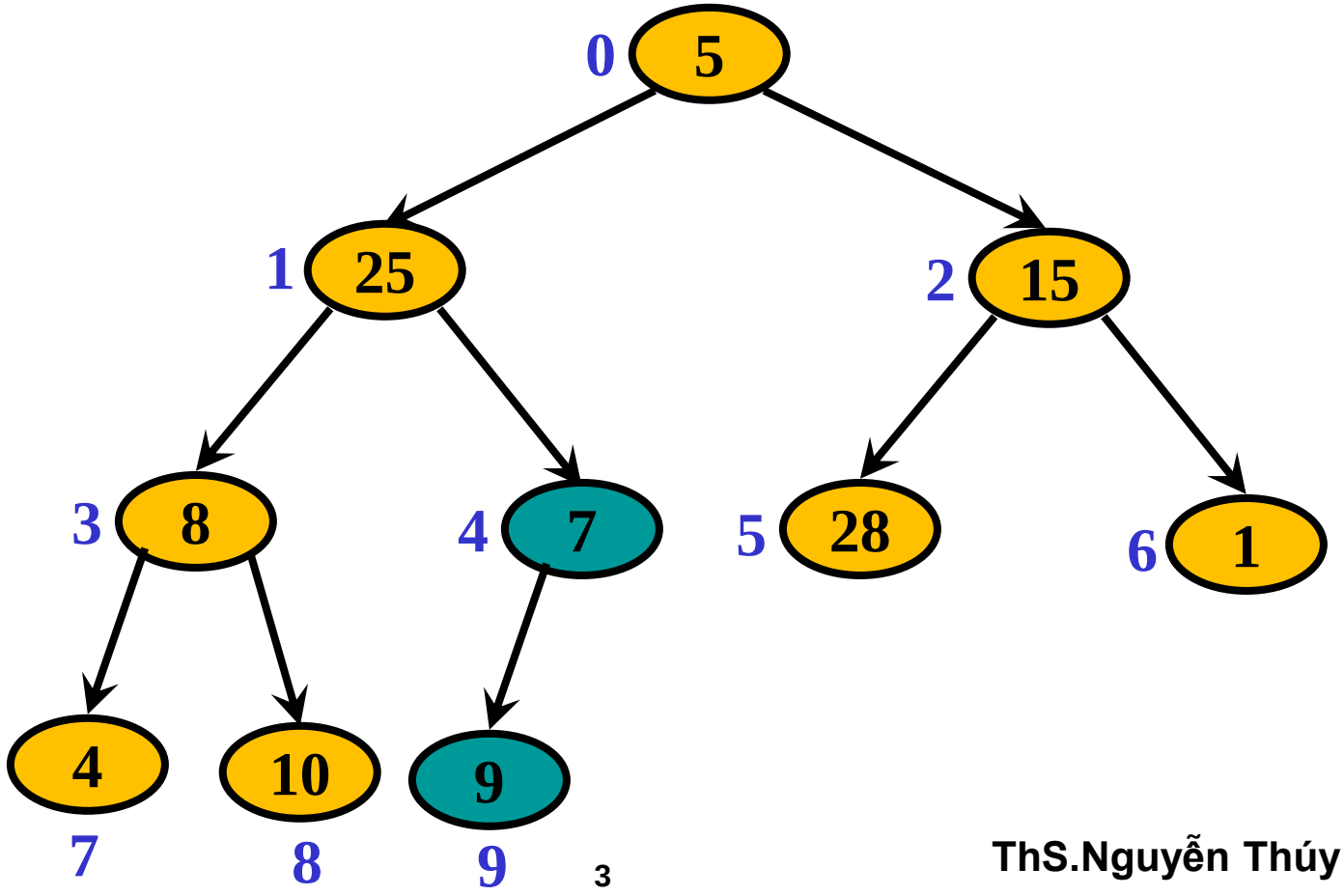
Bước 1 Xây dựng Heap

0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



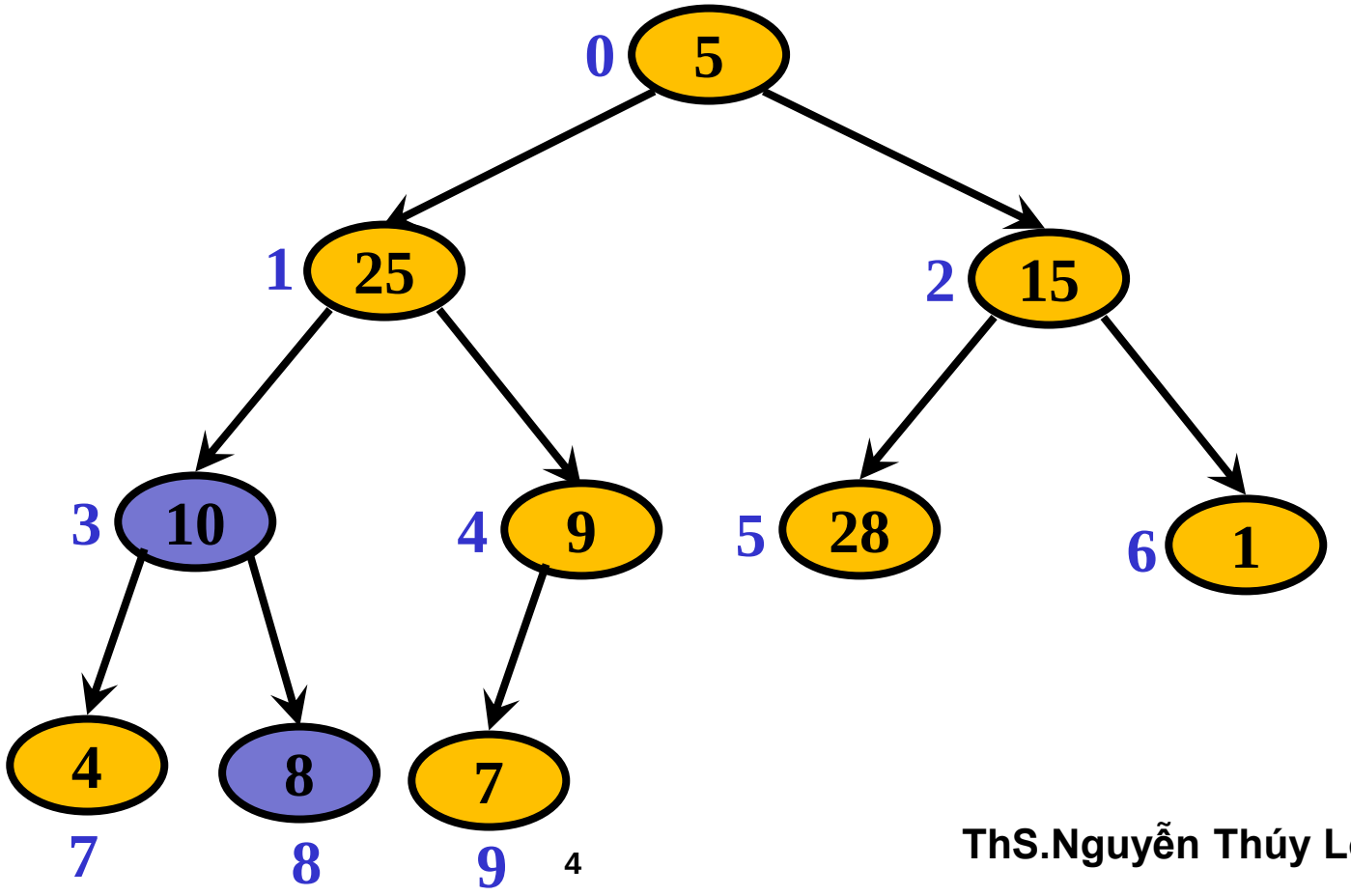
Xây dựng Heap –Điều chỉnh Heap

0	1	2	3	4	5	6	7	8	9
5	25	15	8	9	28	1	4	10	7



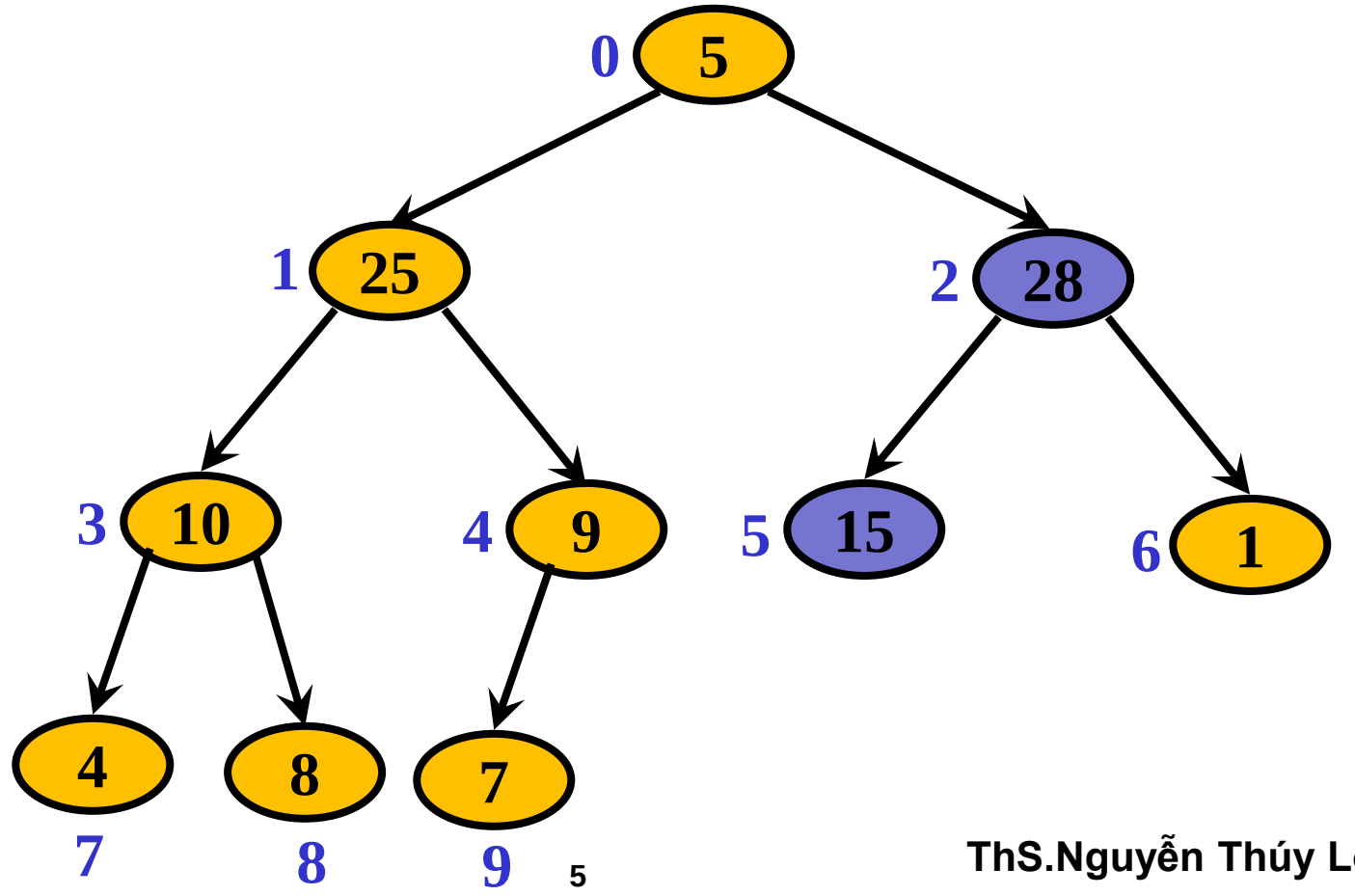
Xây dựng Heap –Điều chỉnh Heap

0	1	2	3	4	5	6	7	8	9
5	25	15	10	9	28	1	4	8	7



Xây dựng Heap –Điều chỉnh Heap

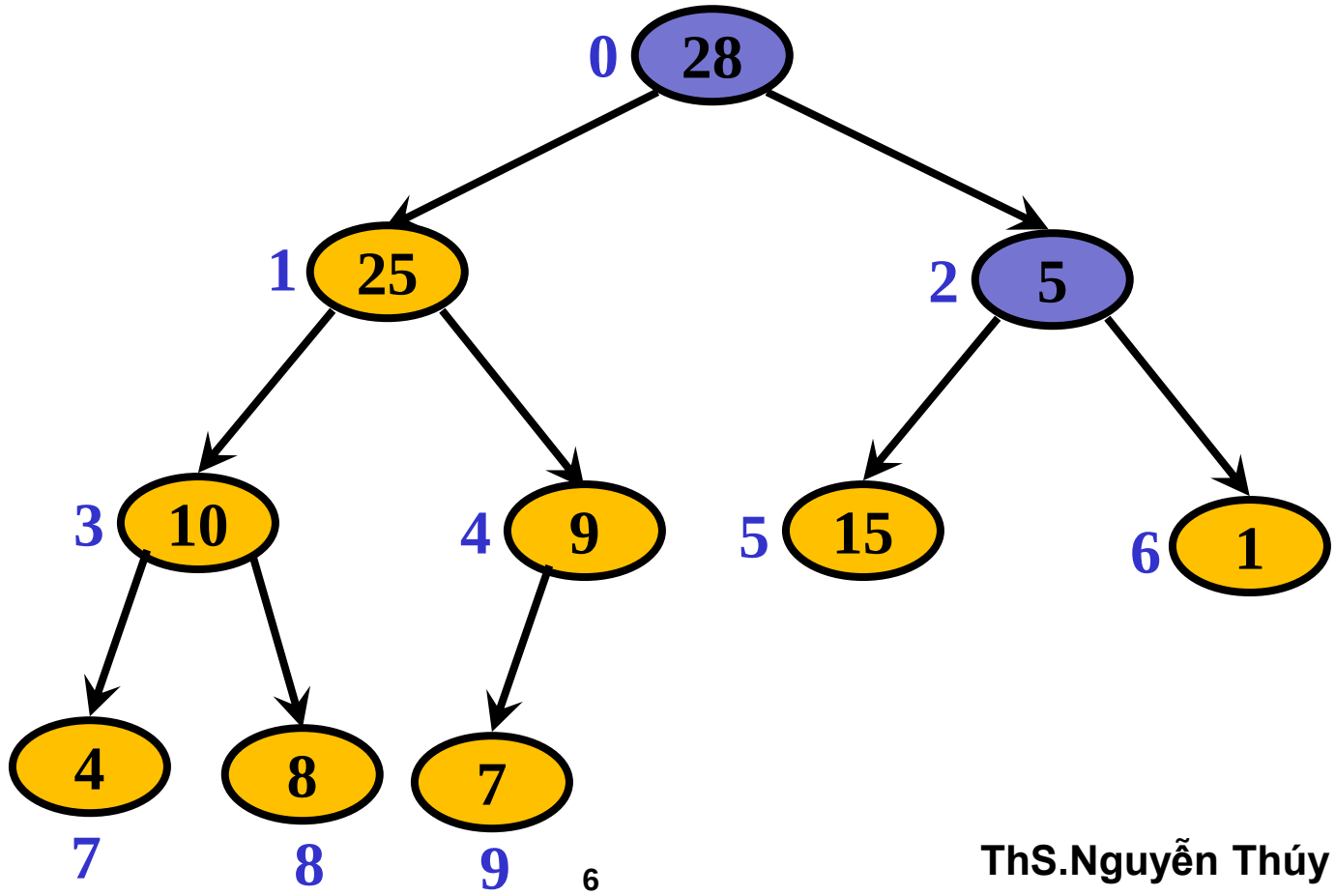
0	1	2	3	4	5	6	7	8	9
5	25	28	10	9	15	1	4	8	7



ThS.Nguyễn Thúy Loan

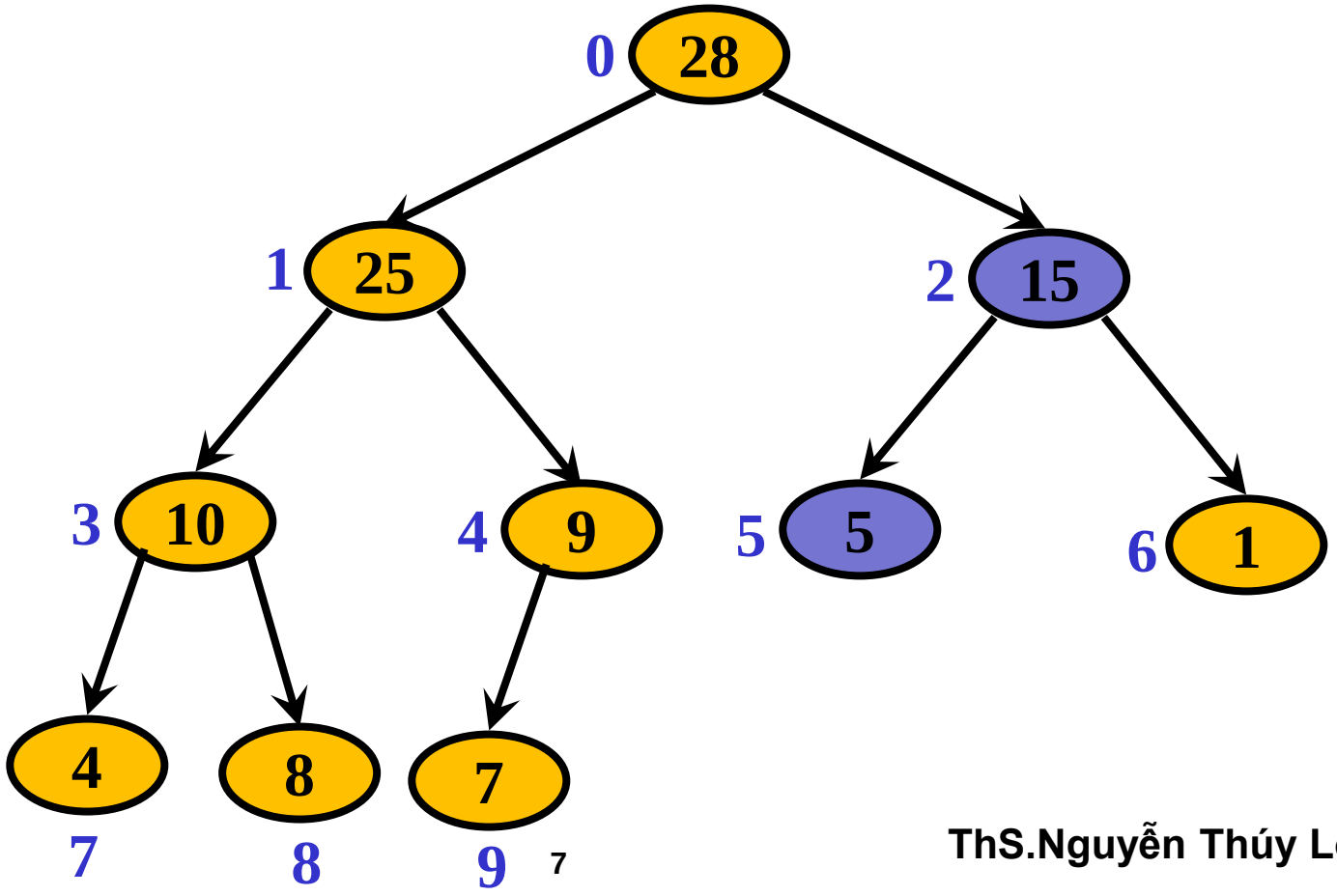
Xây dựng Heap –Điều chỉnh Heap

0	1	2	3	4	5	6	7	8	9
28	25	5	10	9	15	1	4	8	7



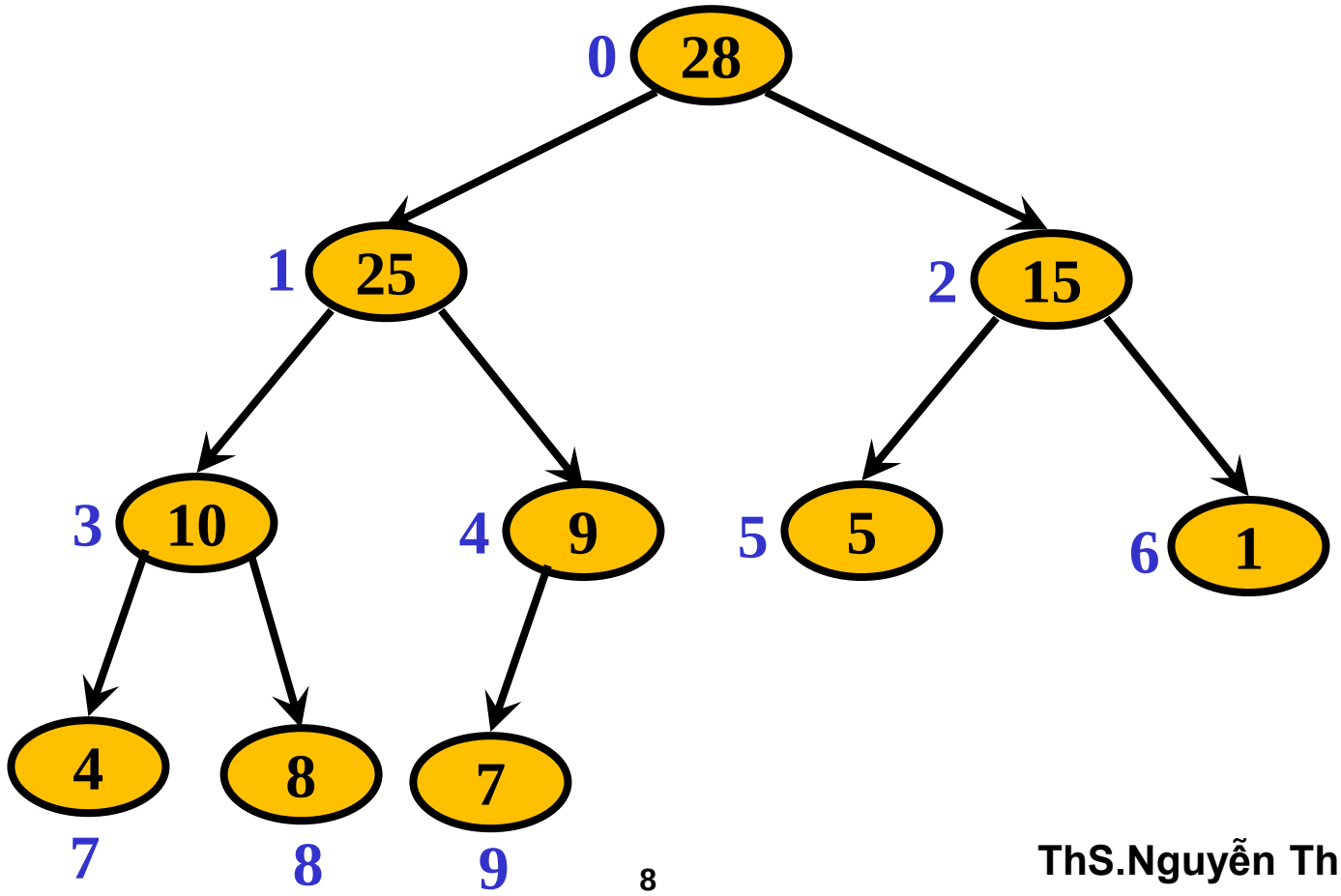
Xây dựng Heap –Điều chỉnh Heap

0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7



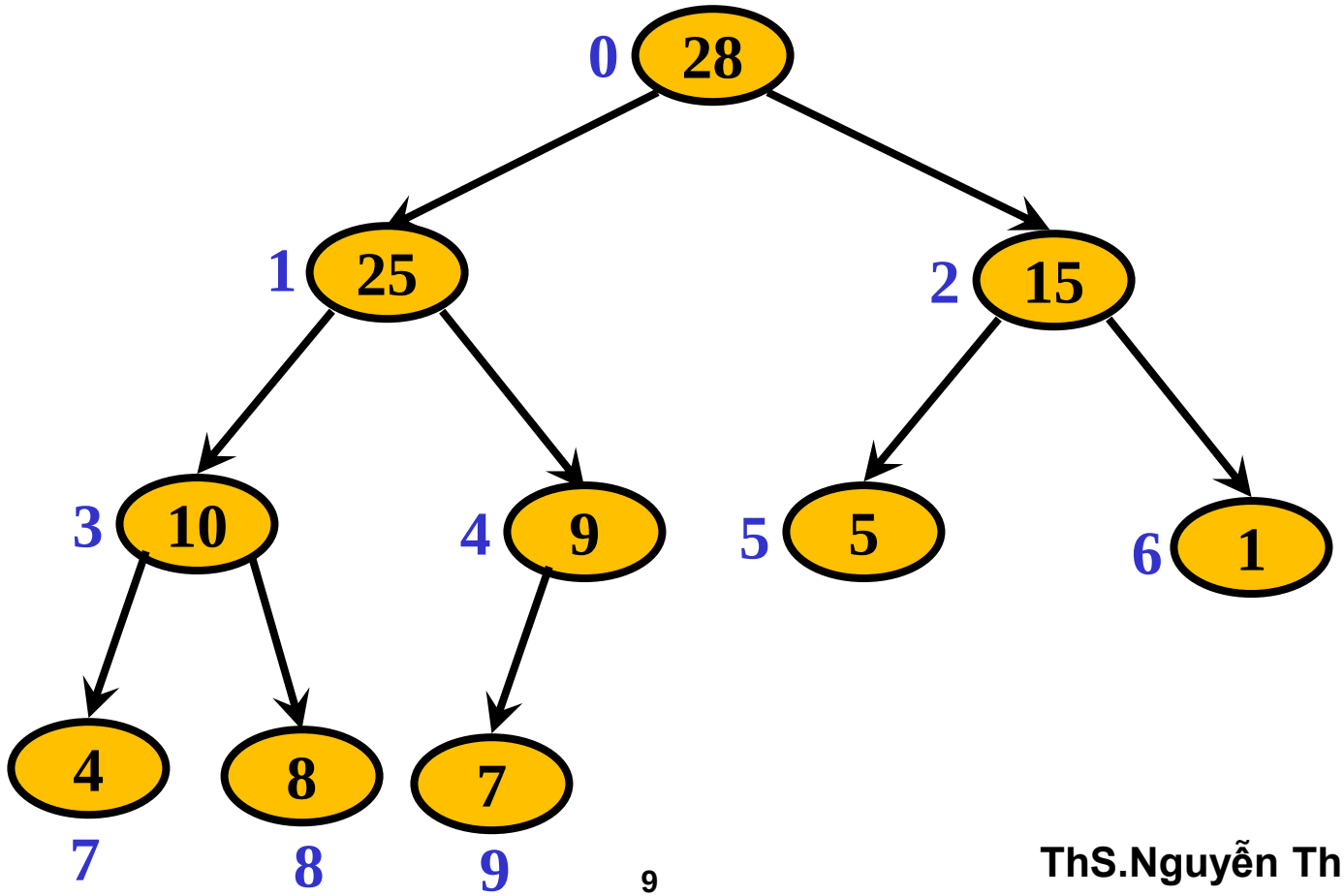
Kết quả Heap

0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7



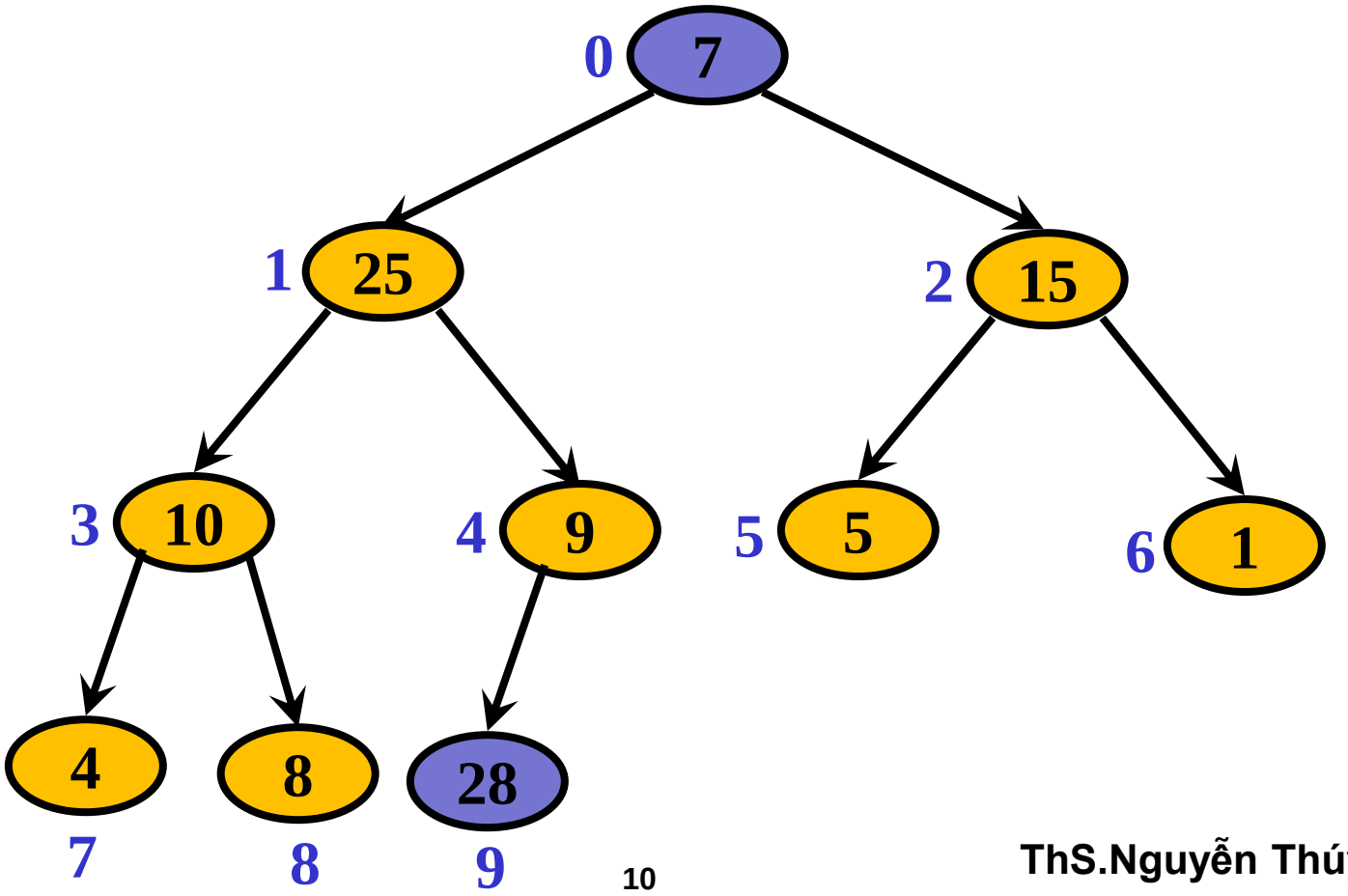
Bước 2: Sắp xếp – Hoán vị lần 1

0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7



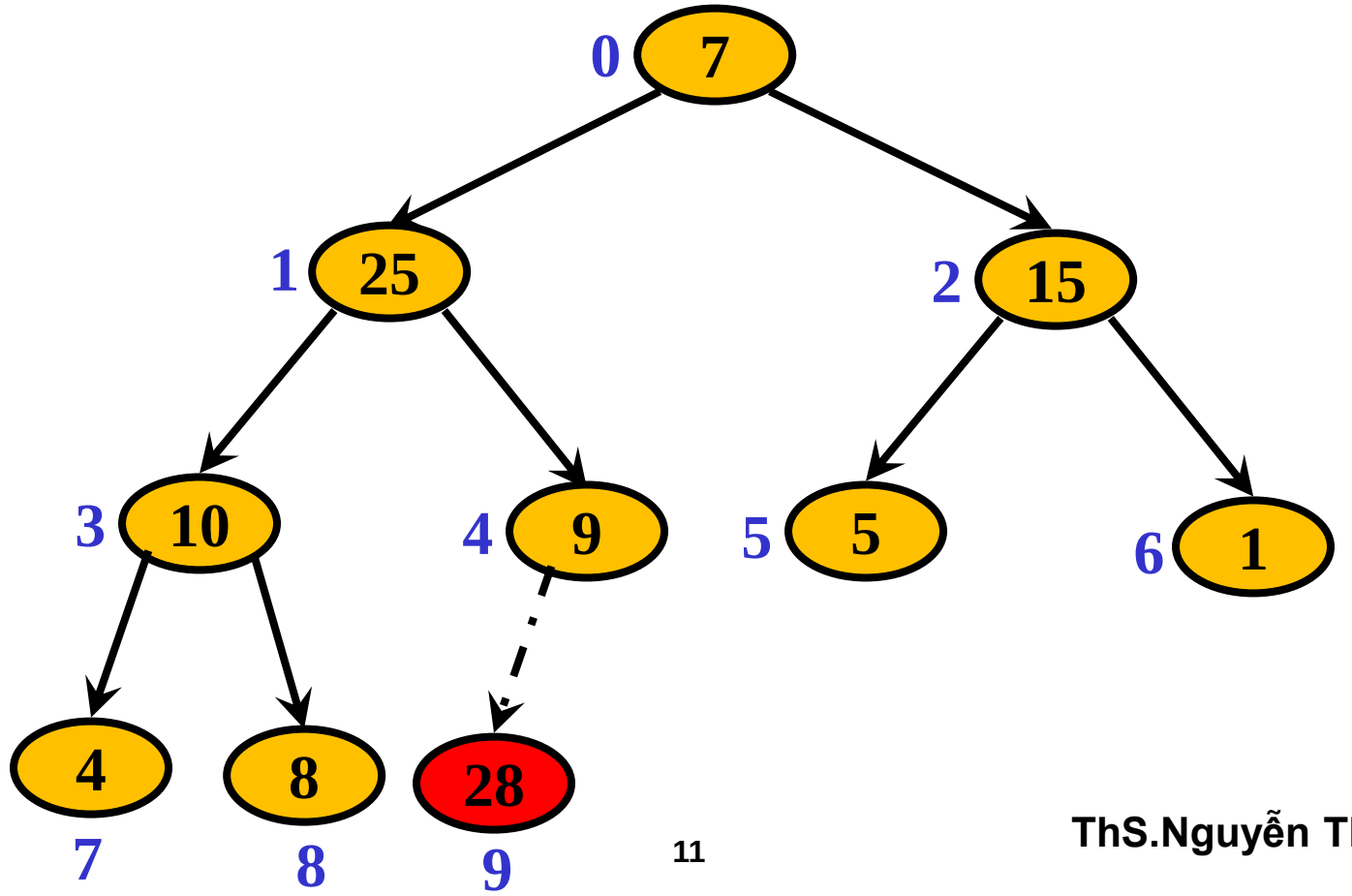
Bước 2: Sắp xếp – Hoán vị lần 1

0	1	2	3	4	5	6	7	8	9
7	25	15	10	9	5	1	4	8	28

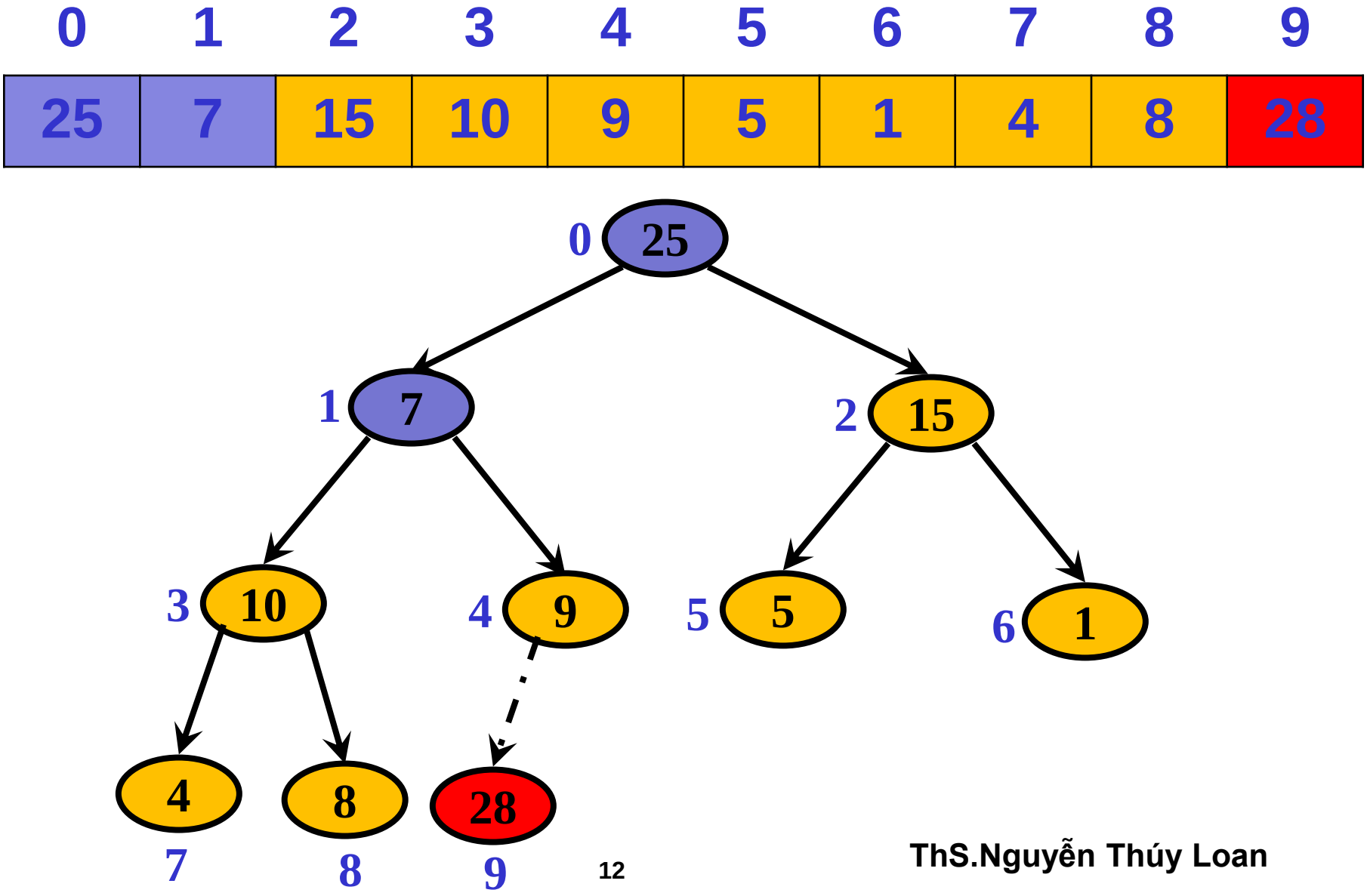


Bước 2: Sắp xếp – Hoán vị lần 1

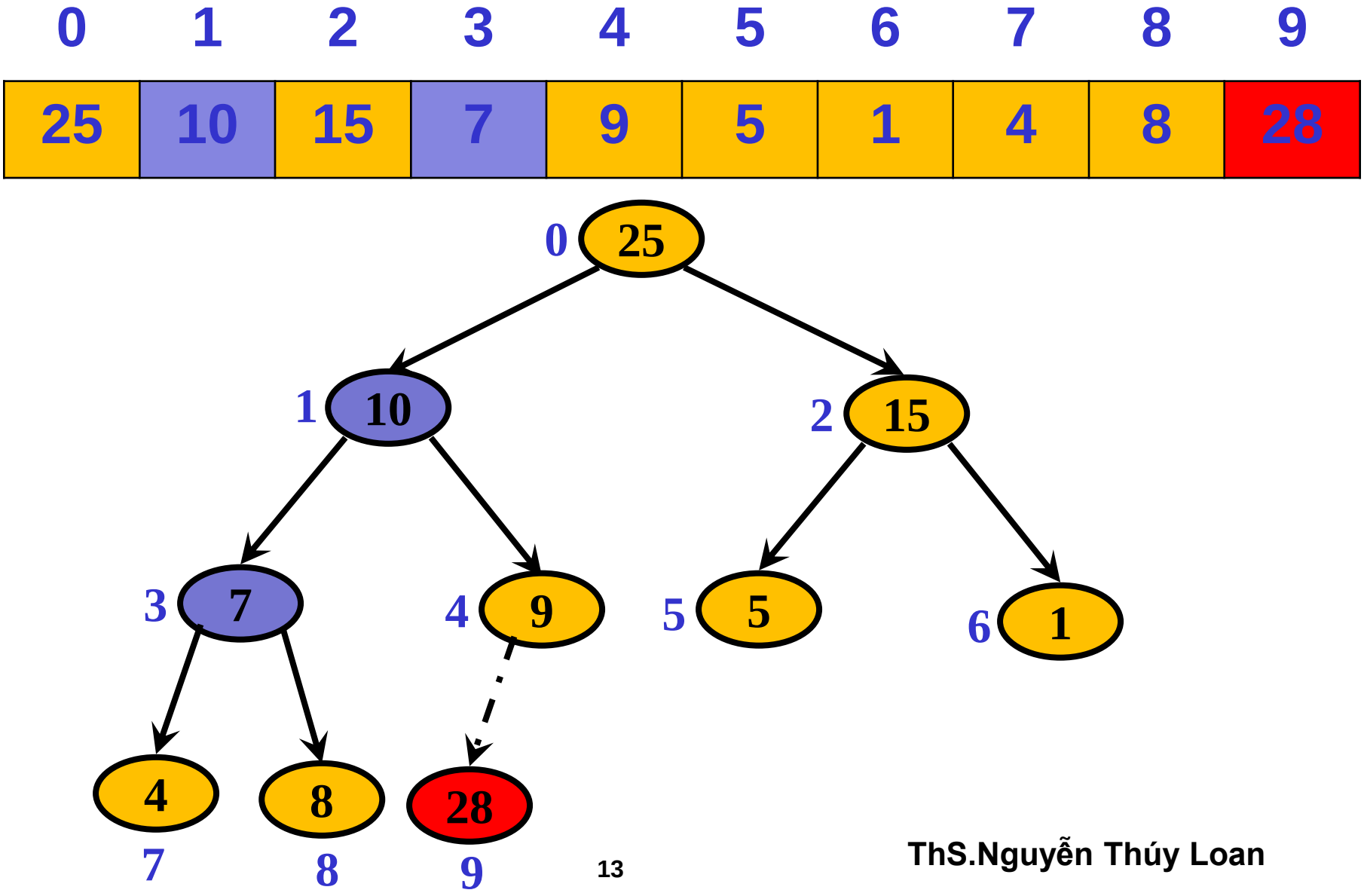
0	1	2	3	4	5	6	7	8	9
7	25	15	10	9	5	1	4	8	28



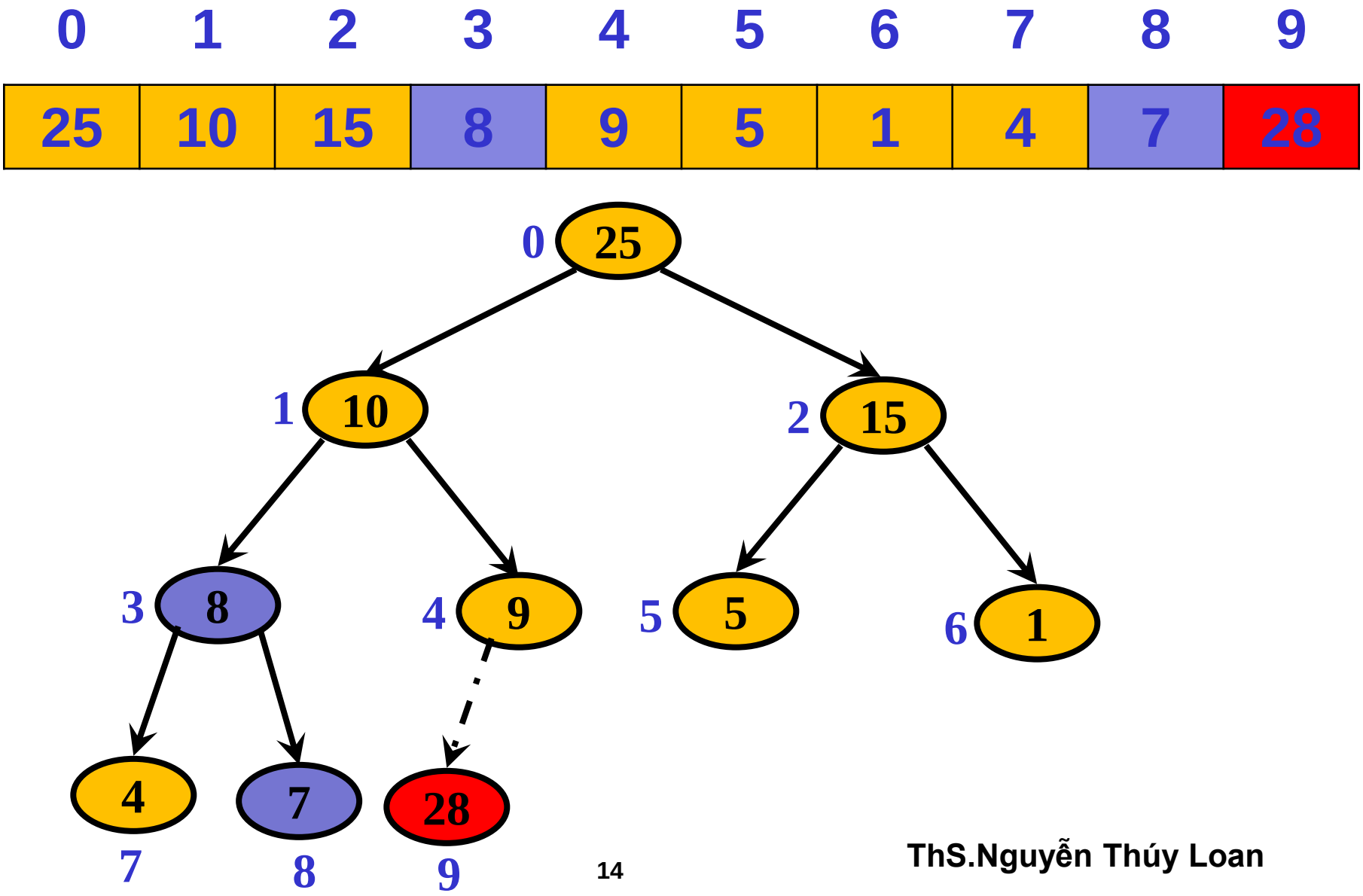
Hiệu chỉnh Heap



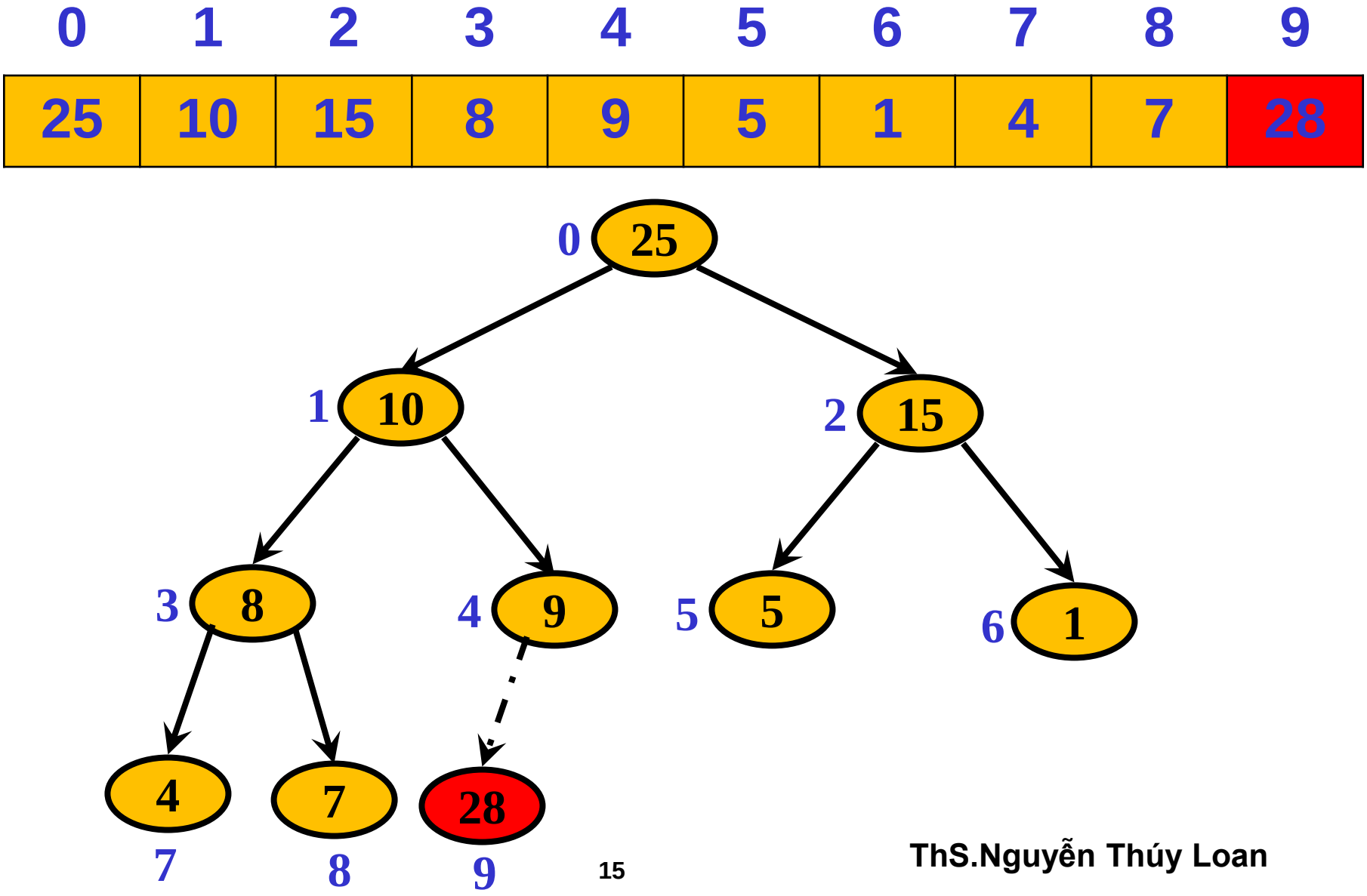
Hiệu chỉnh Heap



Hiệu chỉnh Heap

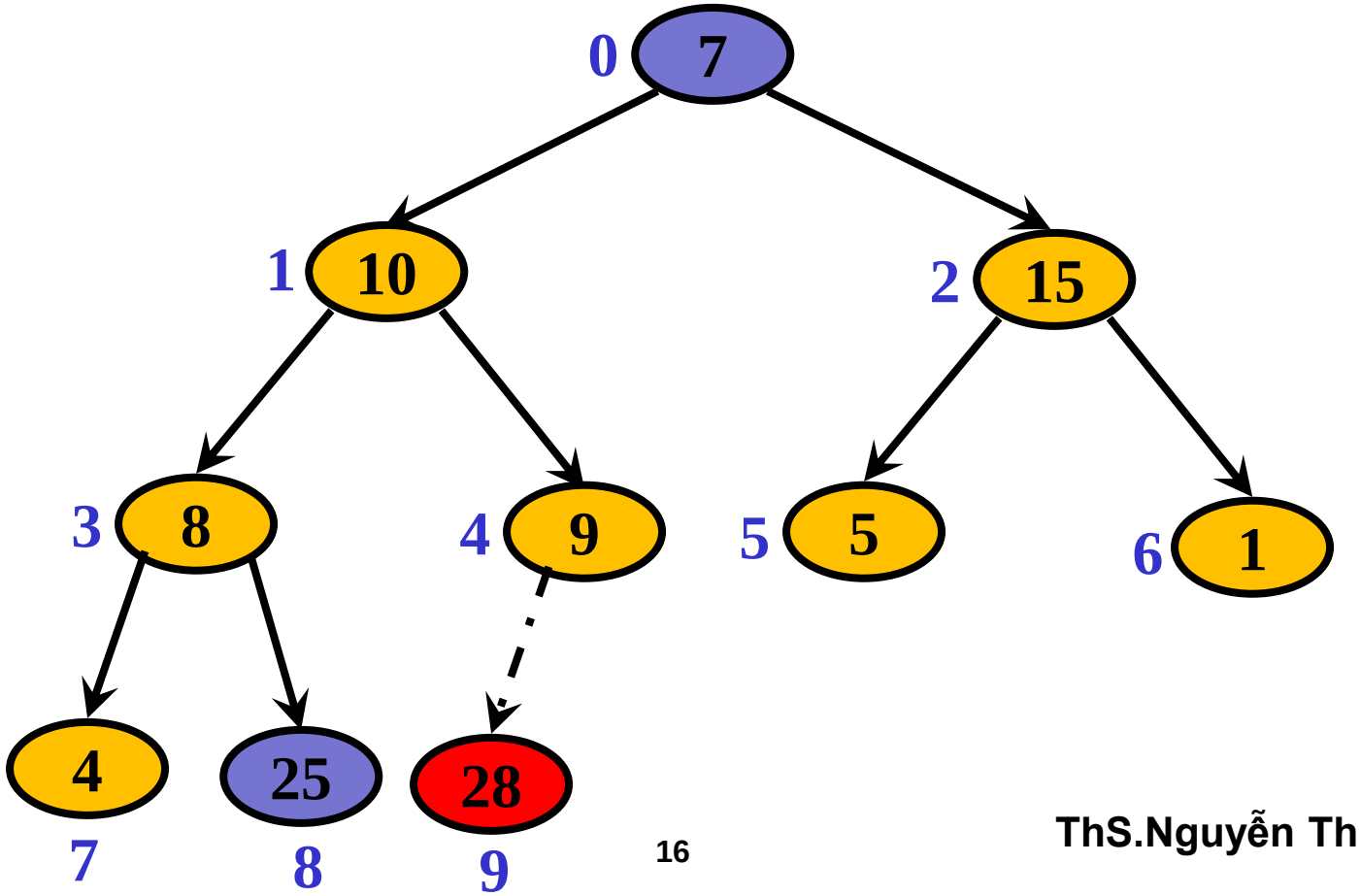


Hiệu chỉnh Heap



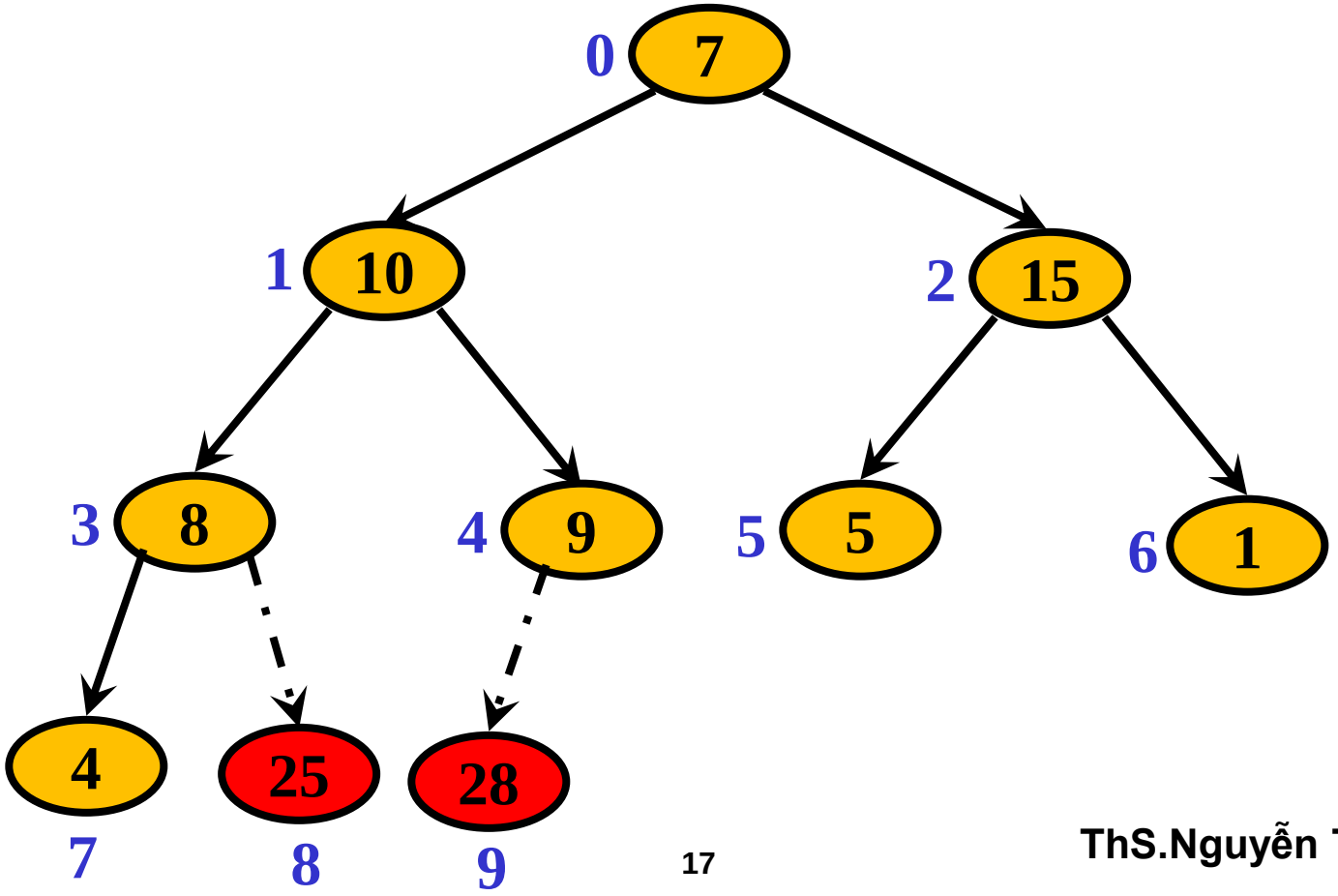
Hiệu chỉnh Heap

0	1	2	3	4	5	6	7	8	9
7	10	15	8	9	5	1	4	25	28



Hiệu chỉnh Heap

0	1	2	3	4	5	6	7	8	9
7	10	15	8	9	5	1	4	25	28



Nhận xét

- Ở cây trên, phần tử ở mức i chính là phần tử lớn trong cặp phần tử ở mức i +1, do đó phần tử ở nút gốc là phần tử lớn nhất.
- Nếu loại bỏ gốc ra khỏi cây, thì việc cập nhật cây chỉ xảy ra trên những nhánh liên quan đến phần tử mới loại bỏ, còn các nhánh khác thì bảo toàn.
- Bước kế tiếp có thể sử dụng lại kết quả so sánh của bước hiện tại.
- Vì thế độ phức tạp của thuật toán $O(n\log_2 n)$

3.Các Bước Thuật Toán

- Giai đoạn 1: Hiệu chỉnh dãy số ban đầu thành heap
- Giai đoạn 2: Sắp xếp dãy số dựa trên heap

Bước 1:Đưa phần tử lớn nhất về vị trí đứng ở cuối dãy bằng cách:

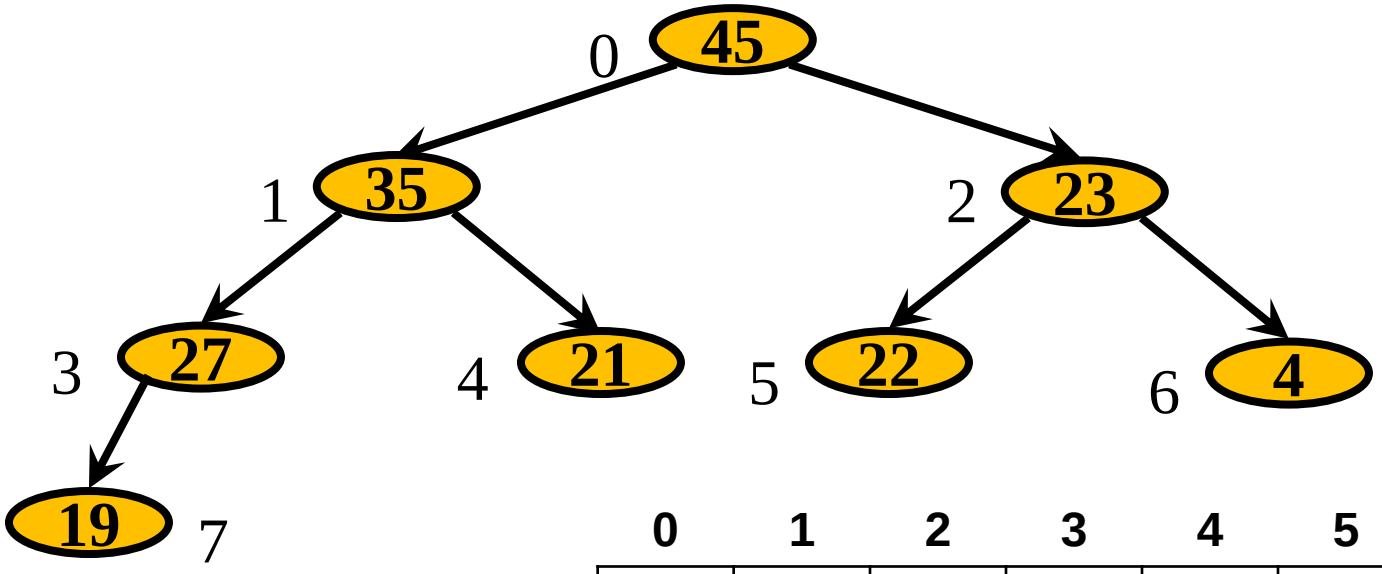
Hoánvị (a_1 , a_r);

Bước 2: Loại bỏ phần tử lớn nhất ra khỏi heap: r = r-1; Hiệu chỉnh phần còn lại của dãy từ a1 , a2 ... ar thành một heap.

Bước 3: Nếu r>1 (heap còn phần tử): Lặp lại Bước 2

Ngược lại : Dừng

3.1 BIỂU DIỄN HEAP BẰNG MẢNG



0	1	2	3	4	5	6	7
45	35	23	27	21	22	4	19

- Nút gốc ở chỉ số [0]
- Nút cha của nút [i] có chỉ số là [(i-1)/2]
- Các nút con của nút [i] (nếu có) có chỉ số [2i+1] và [2i+2]
- Nút cuối cùng có con trong một heap có n phần tử là: [n/2-1]

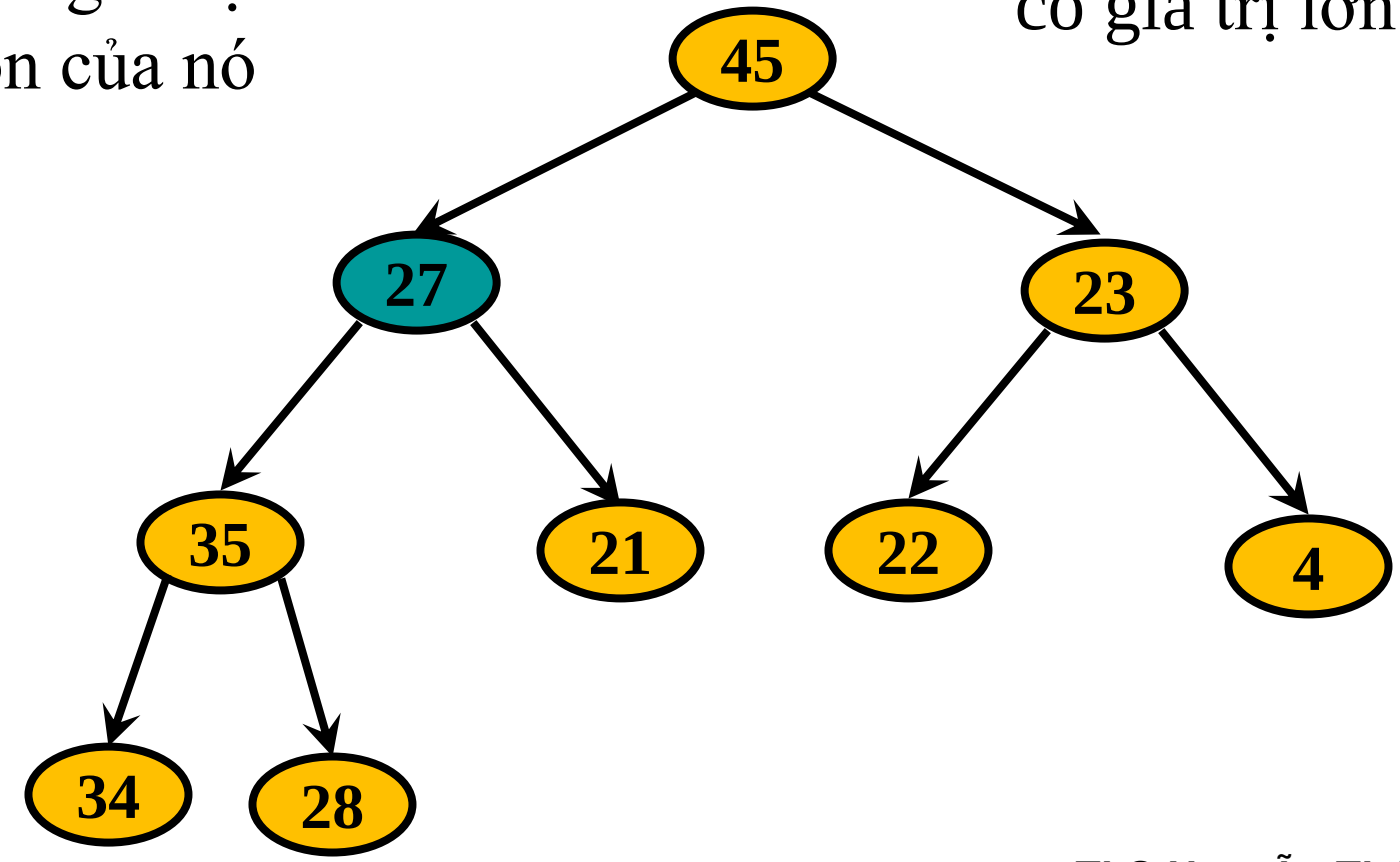
3.1 BIỂU DIỄN HEAP BẰNG MẢNG

- Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải.
- Liên kết giữa các nút được hiểu ngầm, không trực tiếp dùng con trỏ.
- Mảng một chiều được xem là cây chỉ do cách ta xử lý dữ liệu trên đó.
- Nếu ta biết được chỉ số của 1 phần tử trên mảng, ta sẽ dễ dàng xác định được chỉ số của nút cha và (các) nút con của nó.

Thao tác hiệu chỉnh 1 phần tử Heapify

1. Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó

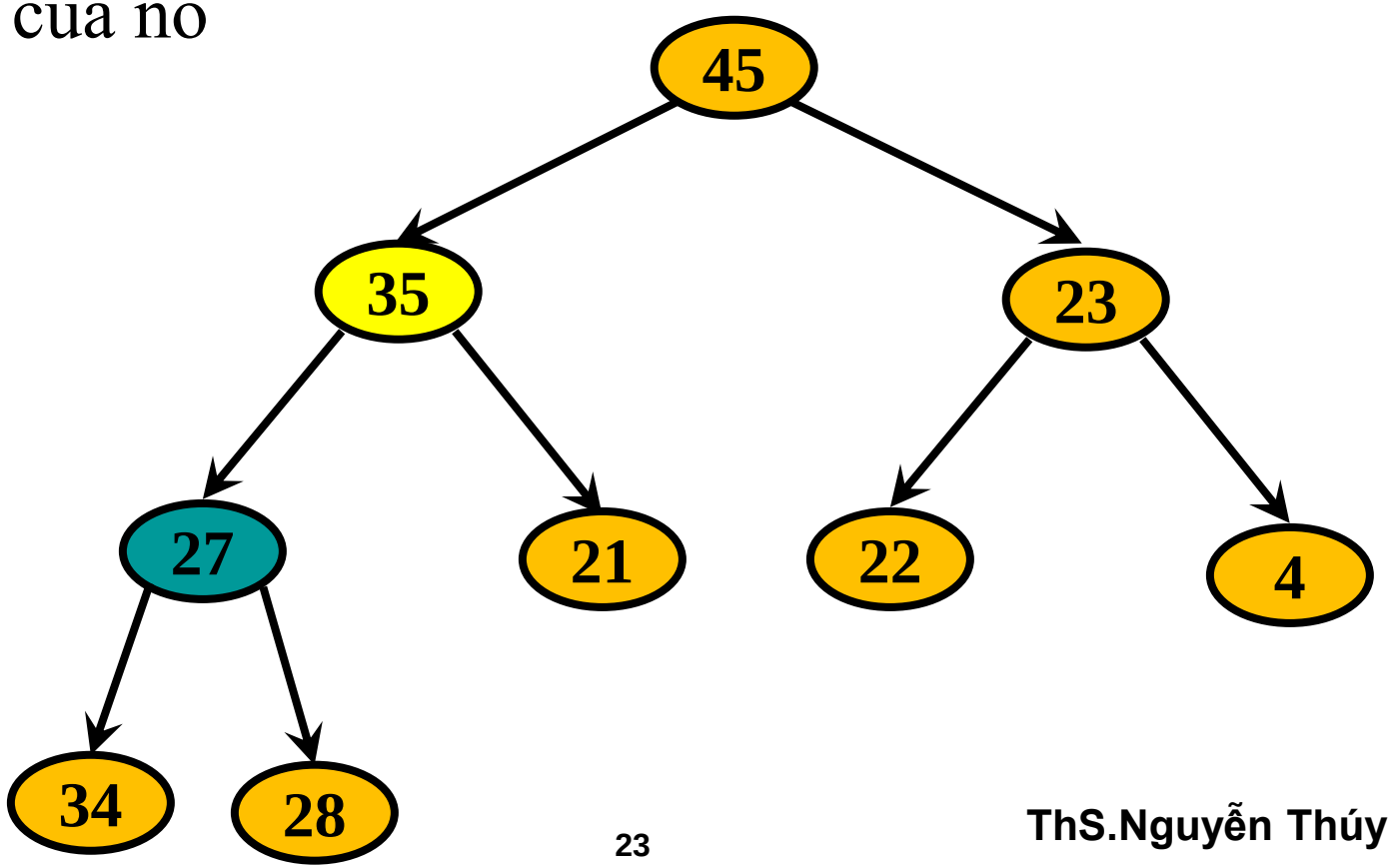
2. Tiến hành đổi chỗ với nút con có giá trị lớn nhất



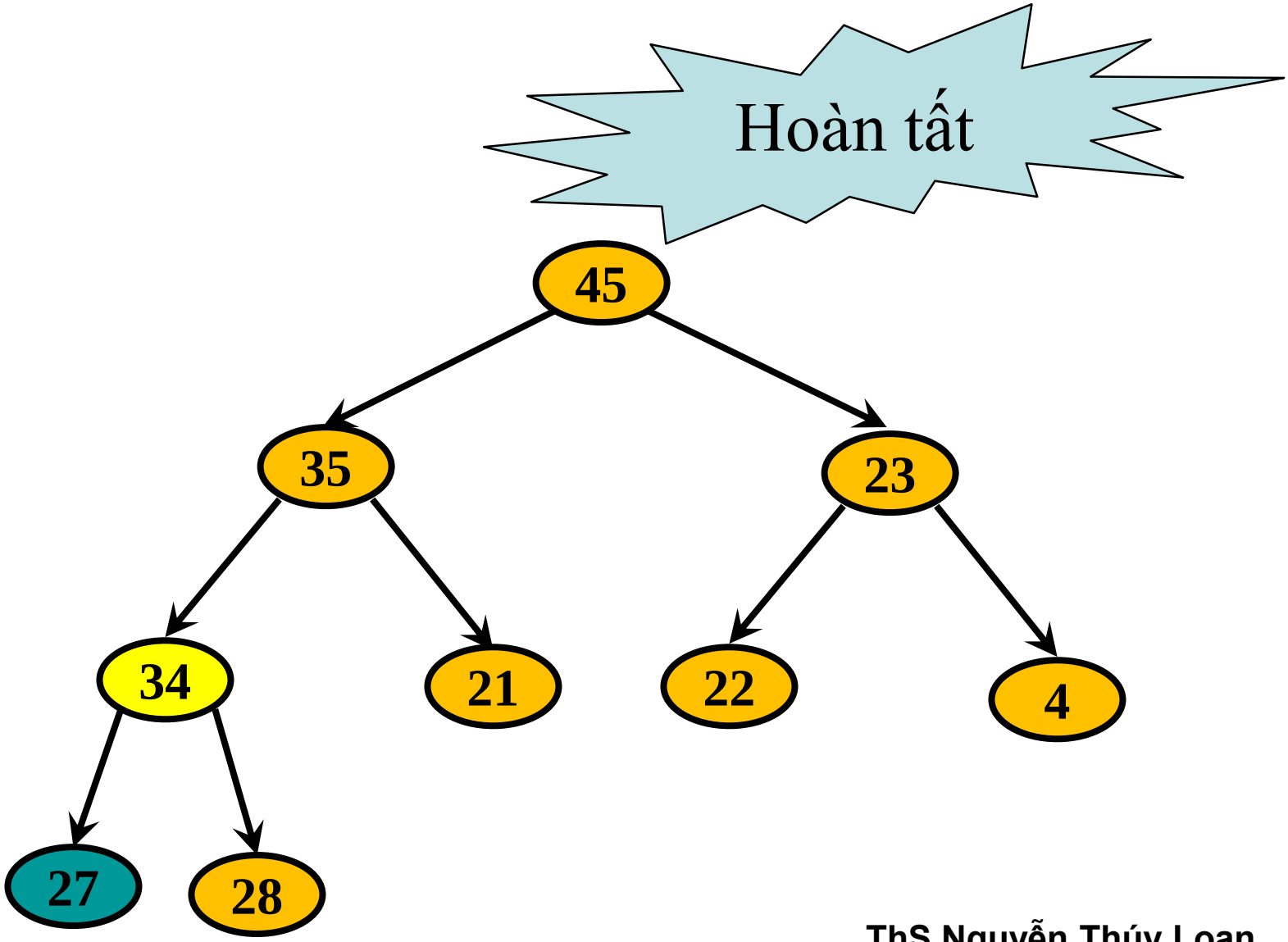
Thao tác hiệu chỉnh 1 phần tử Heapify

1. Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó

2. Tiến hành đổi chỗ với nút con có giá trị lớn nhất



Thao tác hiệu chỉnh 1 phần tử Heapify



Cài đặt thao tác hiệu chỉnh 1 phần tử Heapify

```
1.void Heapify(int a[],int n,int vt)
2.{
3.     while(vt<n/2)
4.     {
5.          int lc = 2*vt+1;
6.          if(lc+1<n && a[lc]<a[lc+1])
7.                    lc++;
8.          if(a[vt]<a[lc])
9.             HoanVi(a[vt],a[lc]);
10.         vt = lc;
11.     }
12.}
```

XÂY DỰNG HEAP

1. Tất cả các phần tử trên mảng có chỉ số từ [n/2] đến [n-1] đều là nút lá.
2. Mỗi nút lá được xem là Heap có duy nhất một phần tử.
3. Thực hiện thao tác Heapify trên các phần tử có chỉ số từ [n/2]-1 tới 0.

XÂY DỰNG HEAP

```
1. void BuildHeap(int a[],int n)
2. {
3.     for(int i=n/2-1;i>=0;i--)
4.         Heapify(a,n,i);
5. }
```

XÂY DỰNG HEAP

```
1.void Heapify(int a[],int n,int vt)
2.{
3.     while(vt<n/2-1)
4.     {
5.          int lc = 2*vt+1;
6.          if(lc+1<n && a[lc]<a[lc+1])
7.                    lc++;
8.          if(a[vt]<a[lc])
9.             HoanVi(a[vt],a[lc]);
10.         vt = lc;
11.     }
12.}
```

THUẬT TOÁN HEAP SORT

- **Bước 1 –**
  - **Xây dựng Heap**: Sử dụng thao tác Heapify để chuyển đổi một mảng bình thường thành Heap.
- **Bước 2 –**
  - **Sắp xếp**.
  - Hoán vị phần tử cuối cùng của Heap với phần tử đầu tiên của Heap.
  - Loại bỏ phần tử cuối cùng
  - Thực hiện thao tác Heapify để điều chỉnh phần tử đầu tiên.

THUẬT TOÁN HEAP SORT

```
10.void HeapSort(int a[],int n)
11.{
12.    BuildHeap(a,n);
13.    int length = n;
14.    while(length>1)
15.    {
16.        HoanVi(a[0],a[length-1]);
17.        length--;
18.        Heapify(a,length,0);
19.    }
20.}
```

THUẬT TOÁN HEAP SORT

```
1. void BuildHeap(int a[],int n)
2. {
3.     for(int i=n/2-1;i>=0;i--)
4.         Heapify(a,n,i);
5. }
```