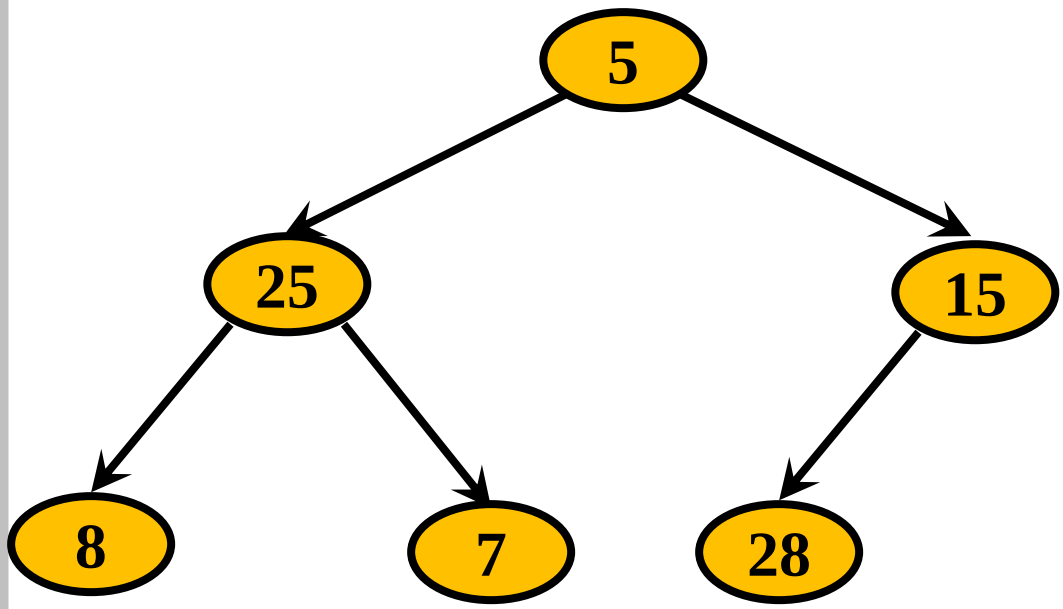
Chương 1
THUẬT TOÁN HEAP SORT

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

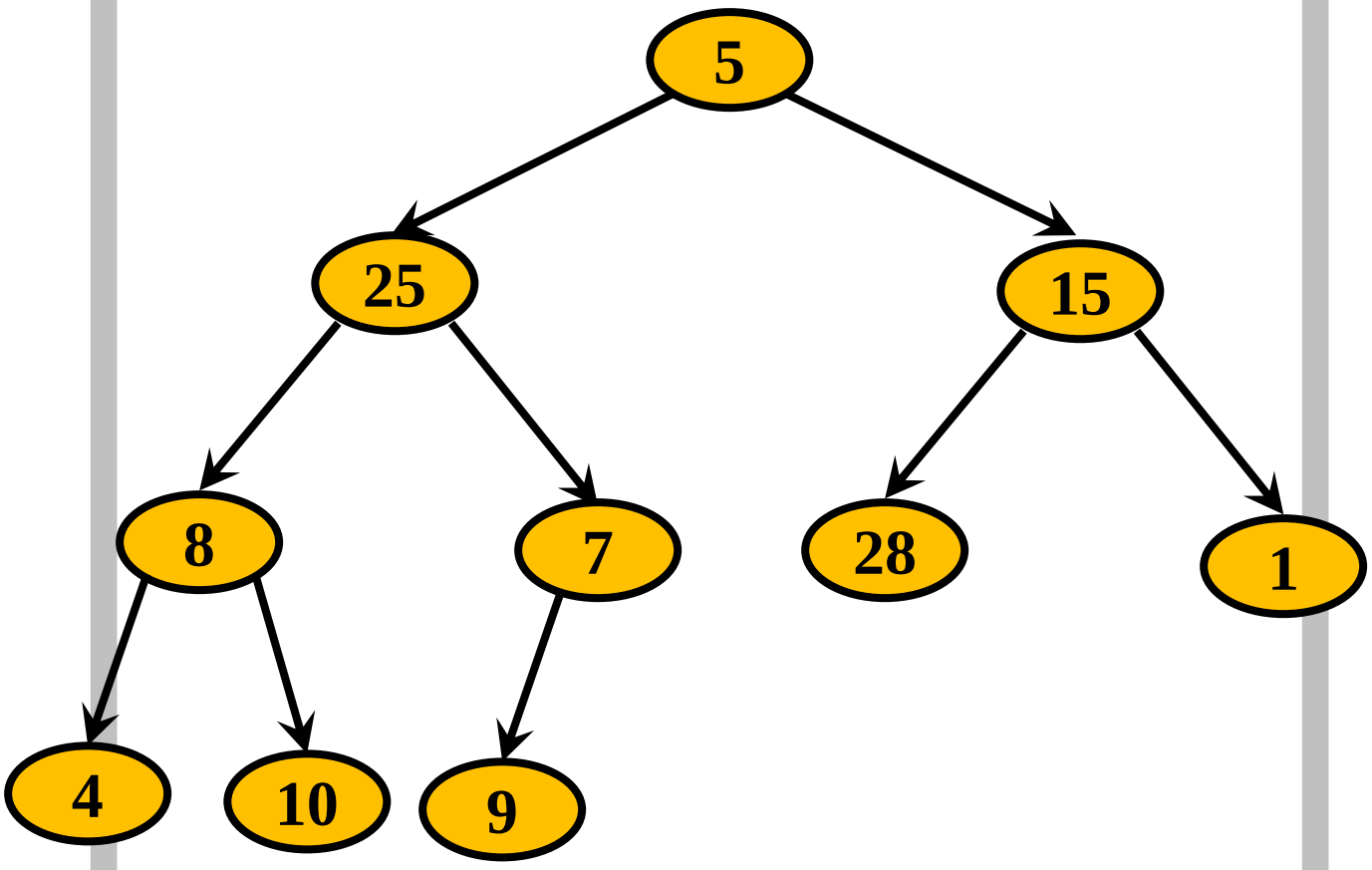
1. ĐỊNH NGHĨA HEAP

- **Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ**
- Ví dụ 1:



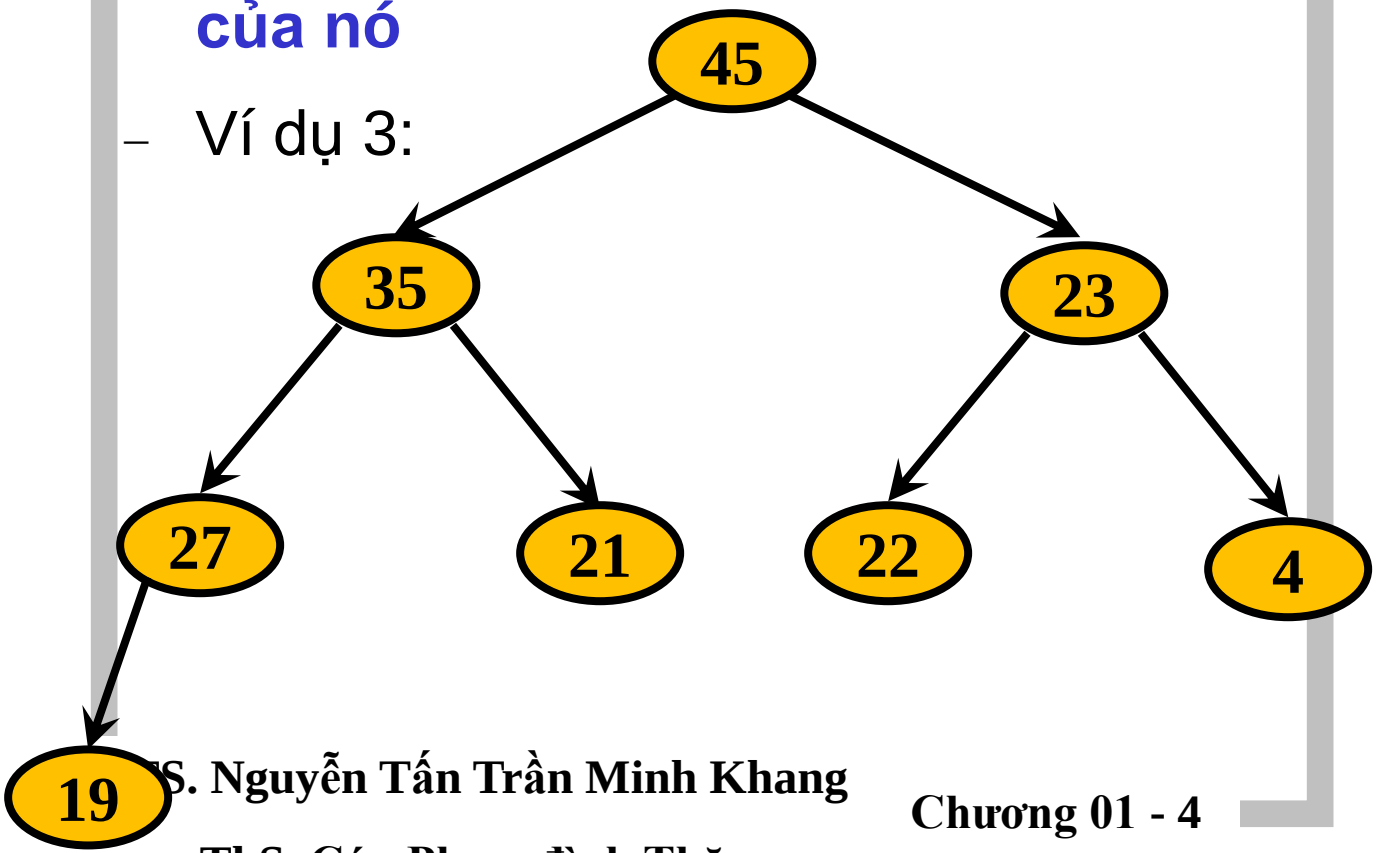
1. ĐỊNH NGHĨA HEAP

- **Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ**
- Ví dụ 2:



1. ĐỊNH NGHĨA HEAP

- **Định nghĩa Heap sơ khởi: Heap là một cây nhị phân đầy đủ**
- Mỗi nút trong Heap chứa một giá trị có thể so sánh với giá trị của nút khác.
- **Đặc điểm của Heap là giá trị của mỗi nút >= giá trị của các nút con của nó**
- Ví dụ 3:

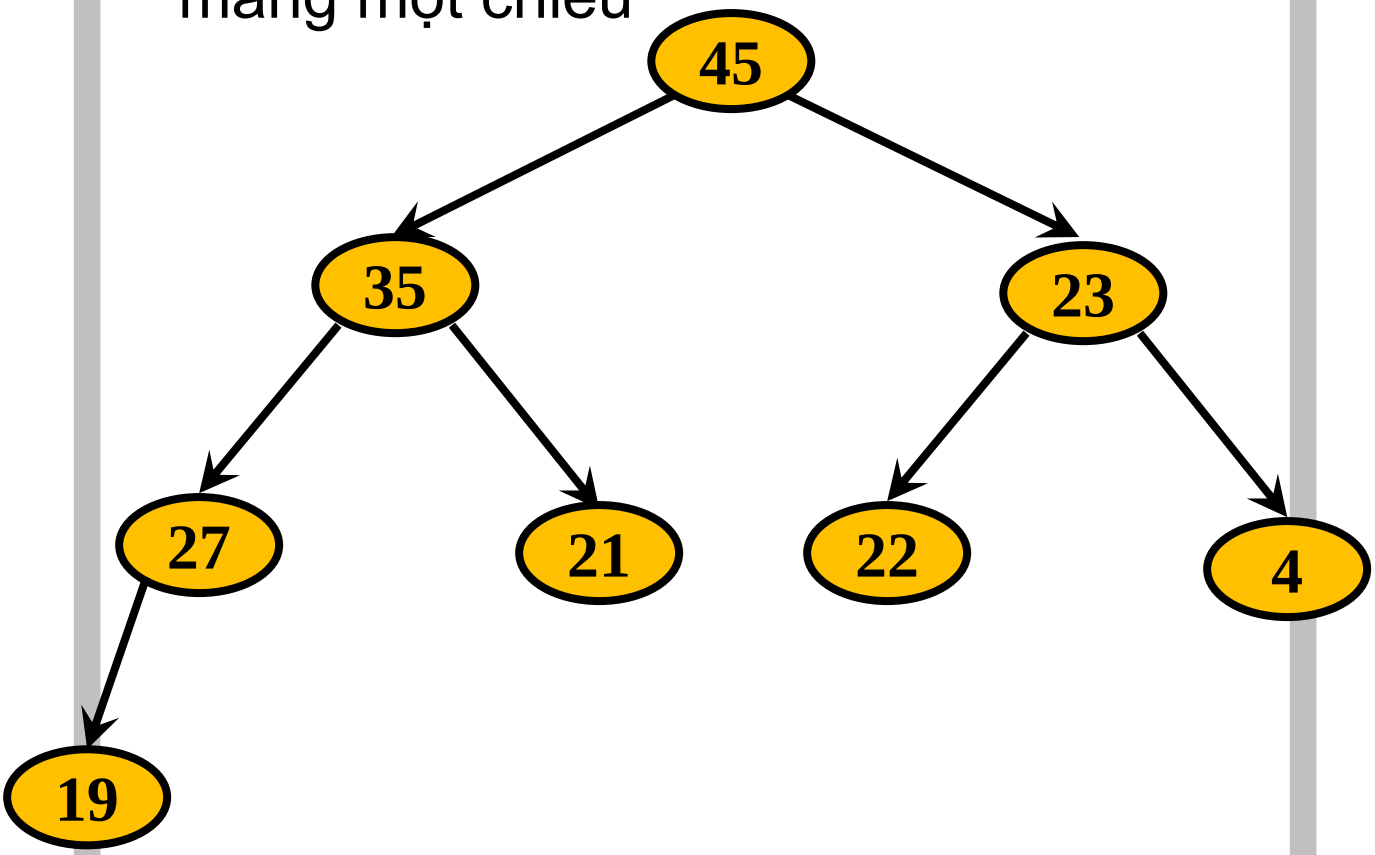


1. ĐỊNH NGHĨA HEAP

- **Heap là một cây nhị phân thỏa các tính chất sau:**
  - **Heap là một cây đầy đủ;**
  - **Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con**
- Hệ quả:
  - Nút lớn nhất là … ?

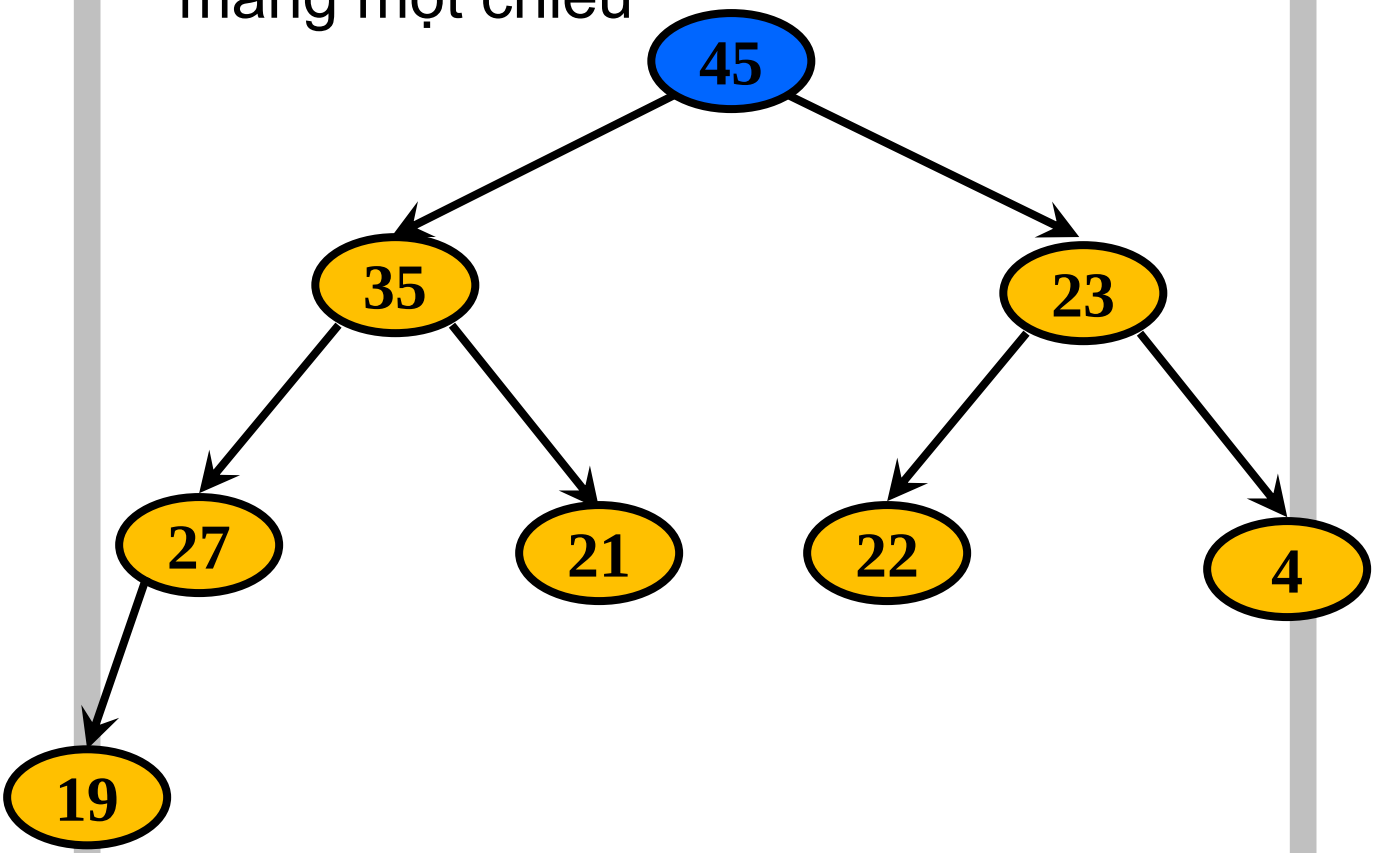
2. BIỂU DIỄN HEAP BẰNG MẢNG

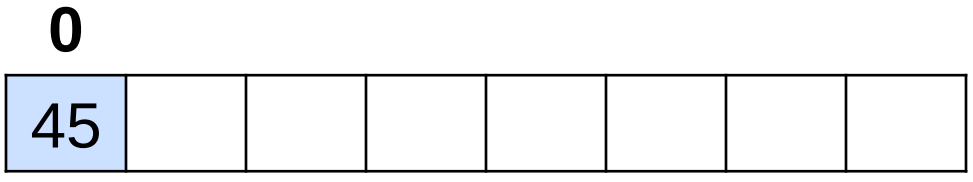
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều



2. BIỂU DIỄN HEAP BẰNG MẢNG

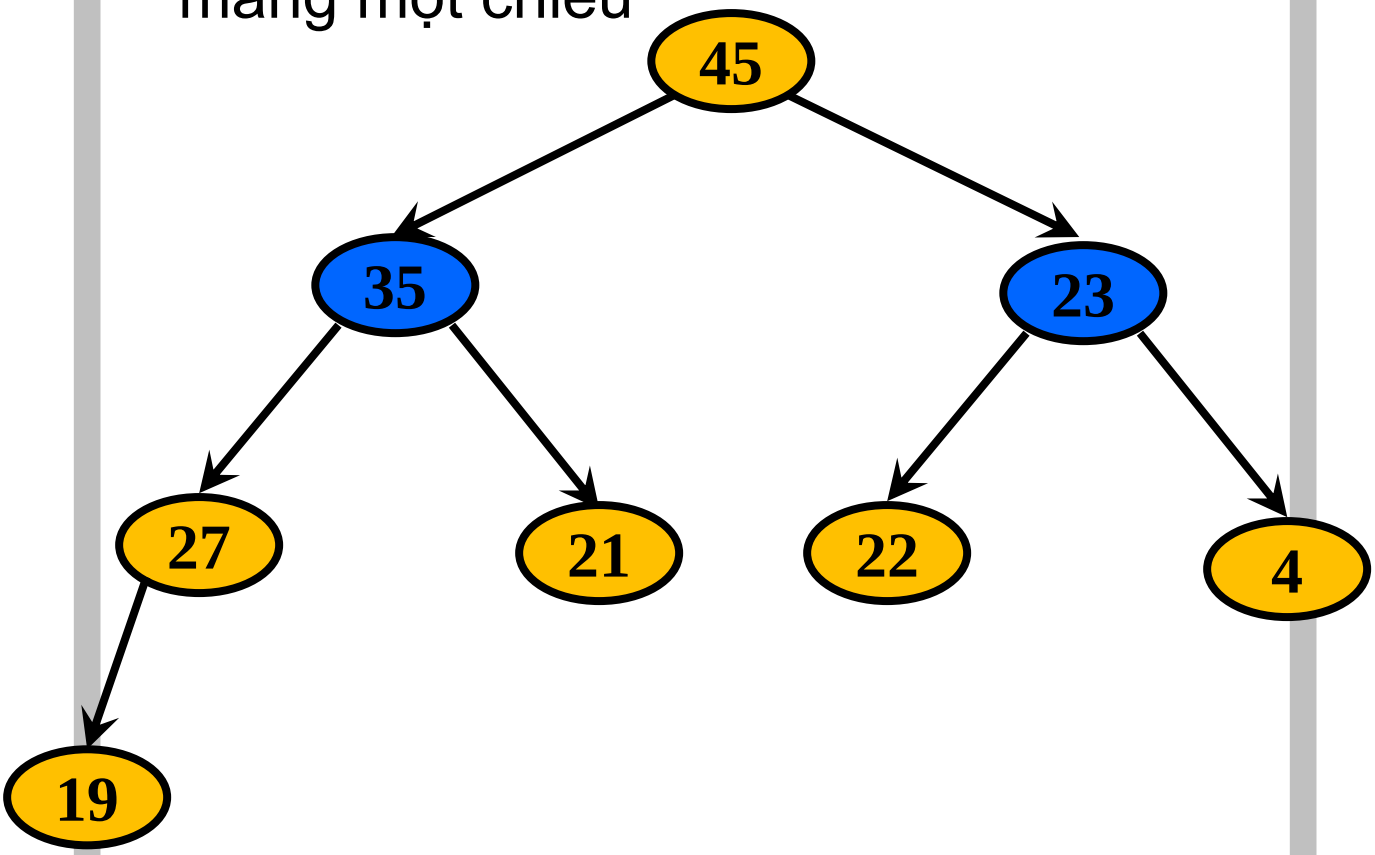
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều





2. BIỂU DIỄN HEAP BẰNG MẢNG

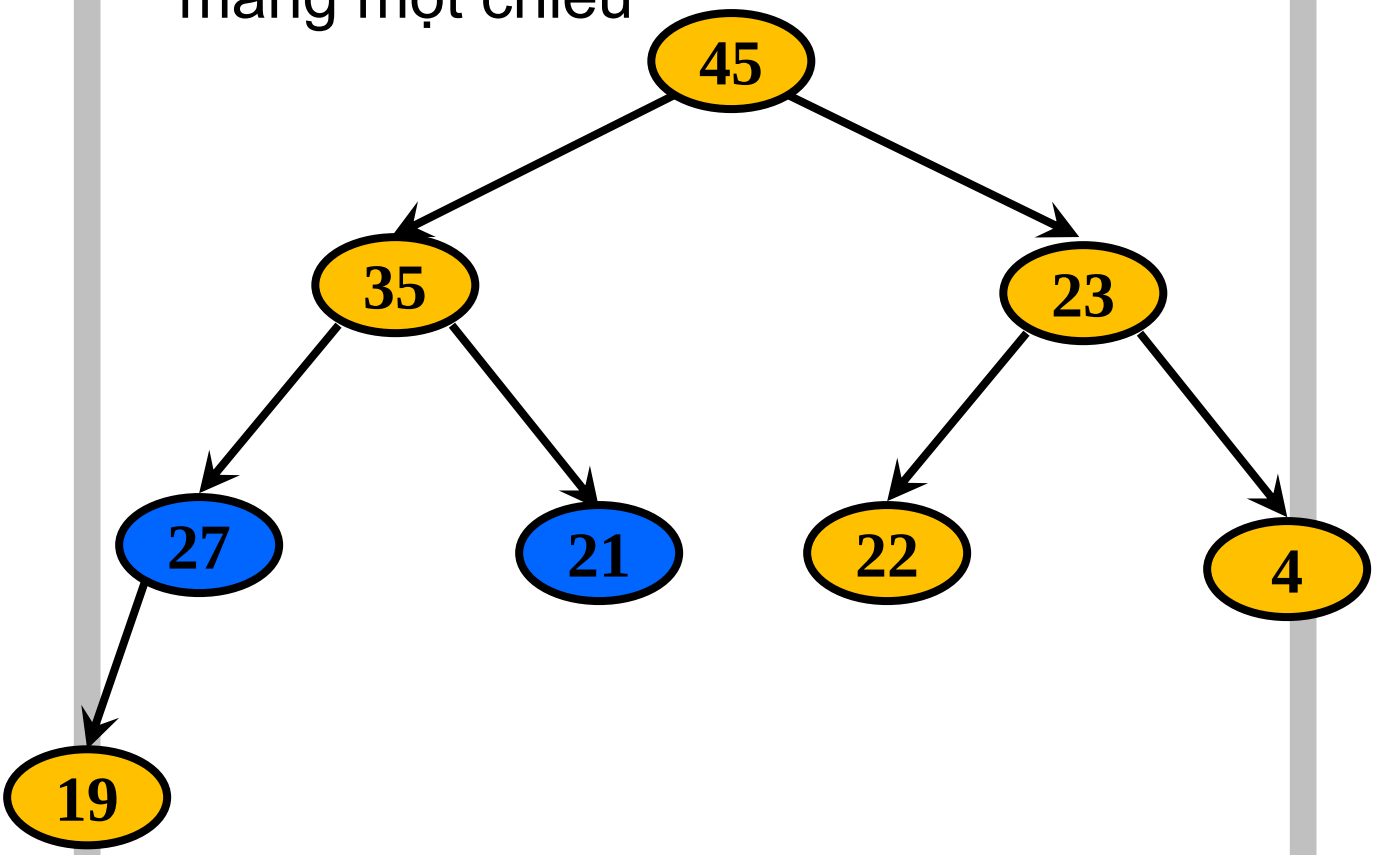
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều



0	1	2					
45	35	23					

2. BIỂU DIỄN HEAP BẰNG MẢNG

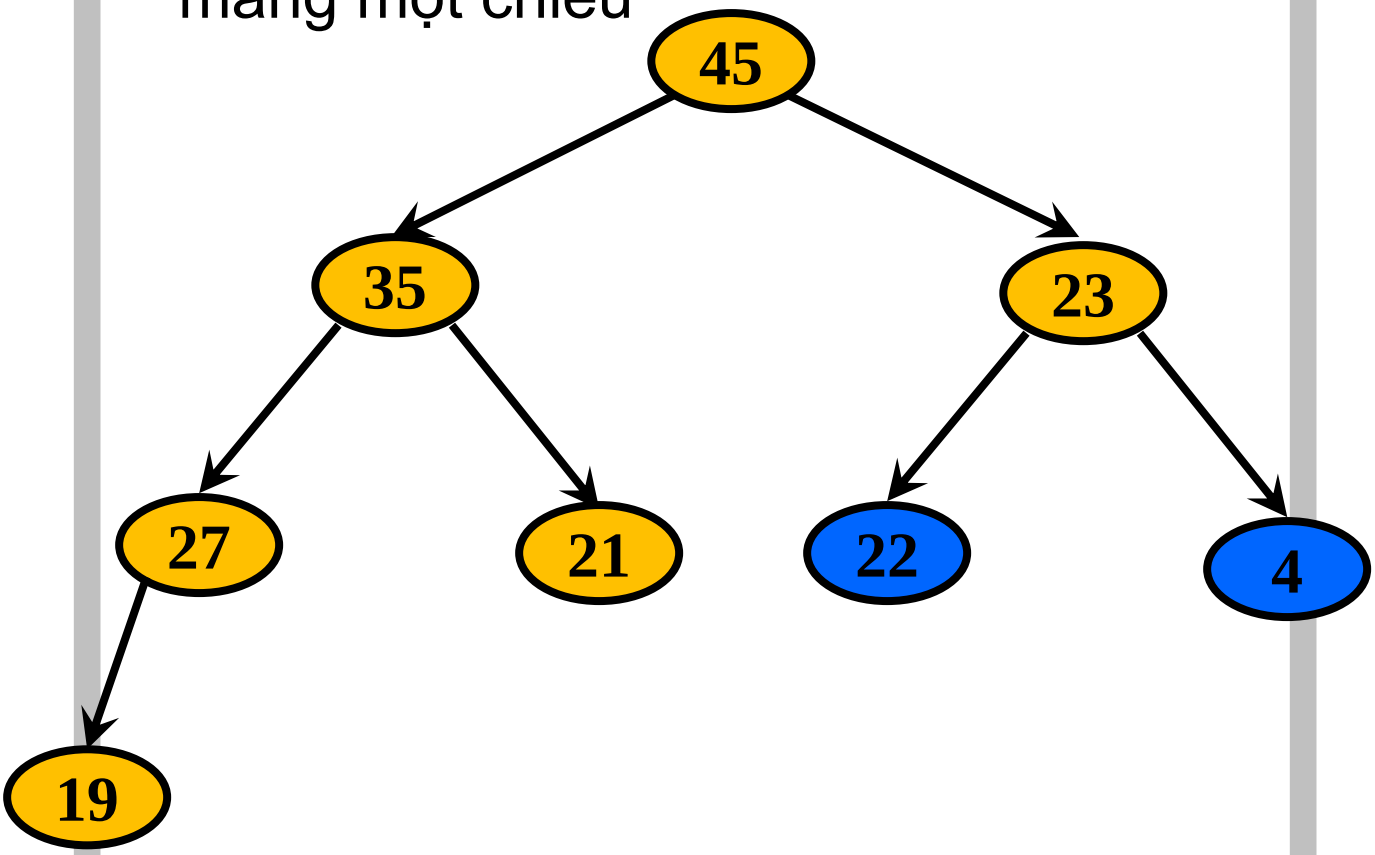
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều



0	1	2	3	4			
45	35	23	27	21			

2. BIỂU DIỄN HEAP BẰNG MẢNG

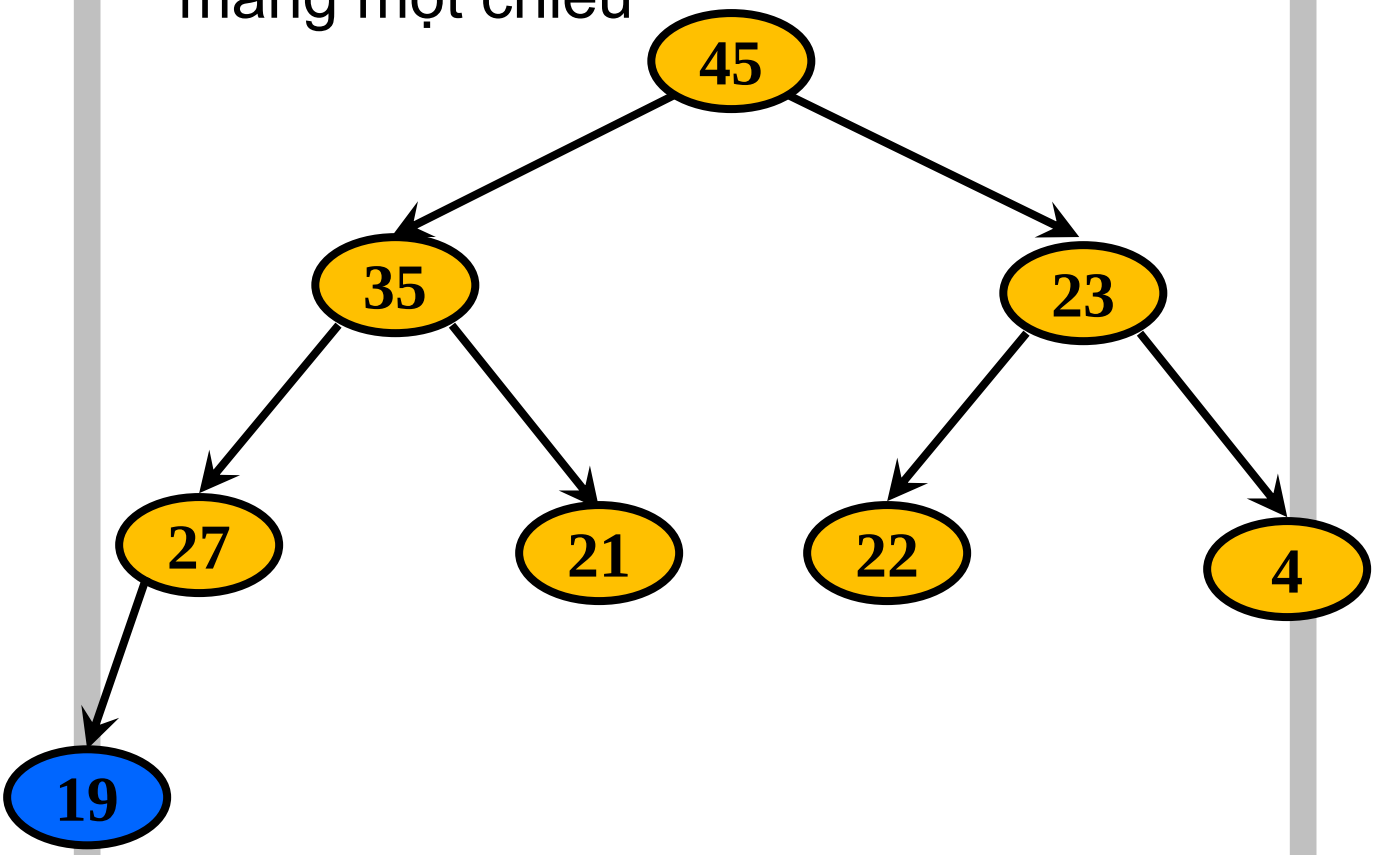
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều



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

2. BIỂU DIỄN HEAP BẰNG MẢNG

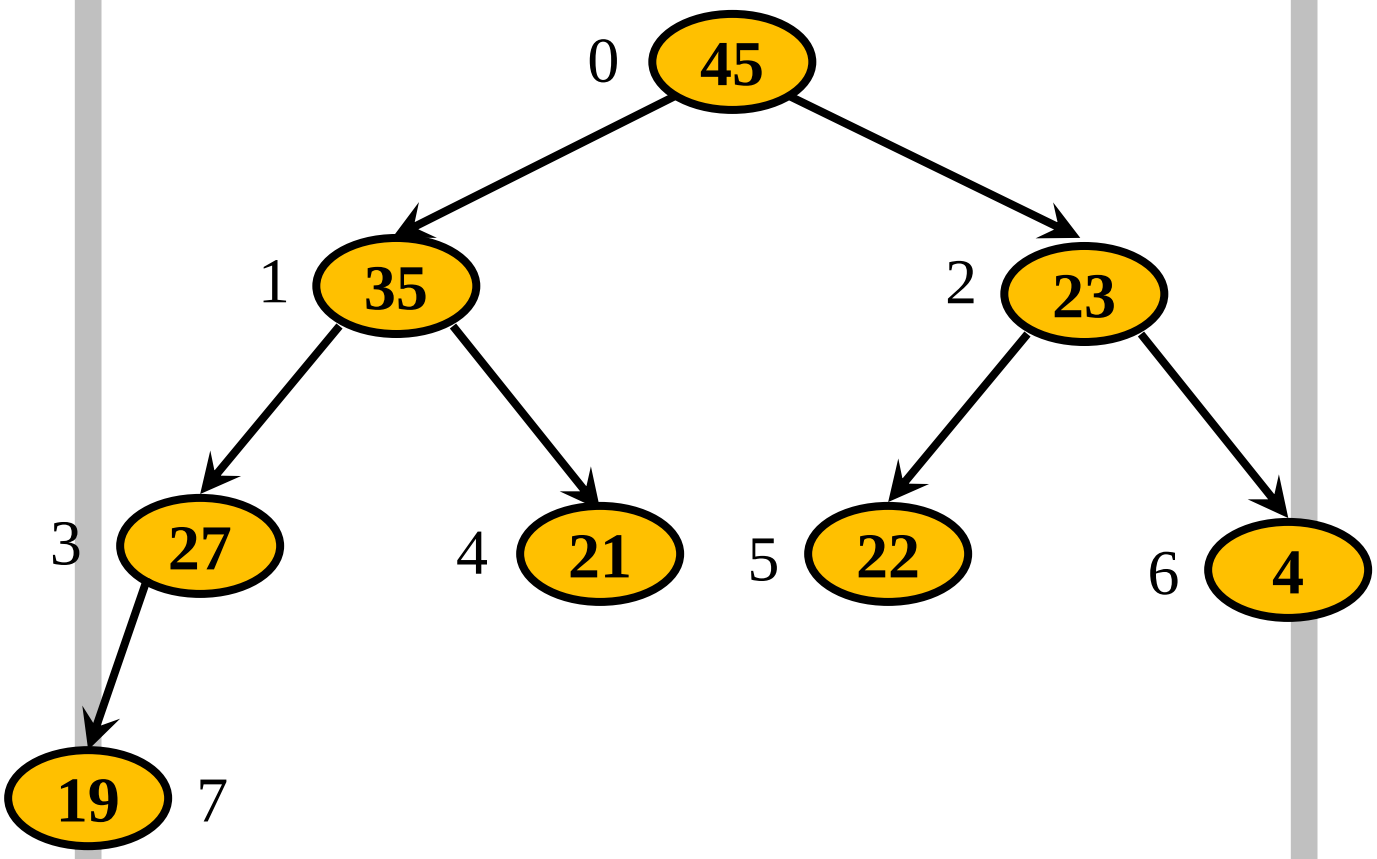
- Ta sẽ lưu giá trị của các nút trong một mảng một chiều



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

2. 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.



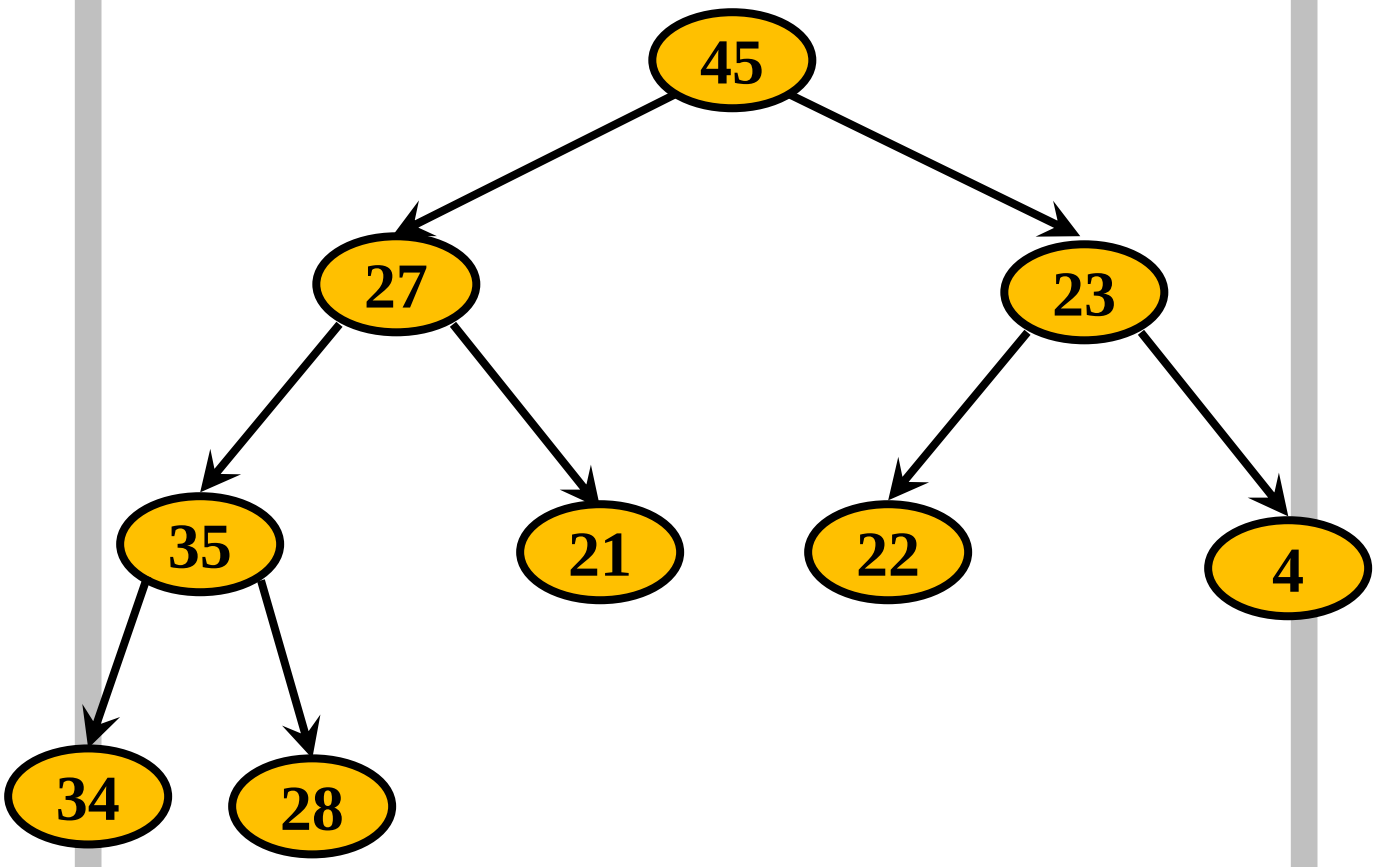
0	1	2	3	4	5	6	7
45	35	23	27	21	22	4	19

- 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. THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ Heapify

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

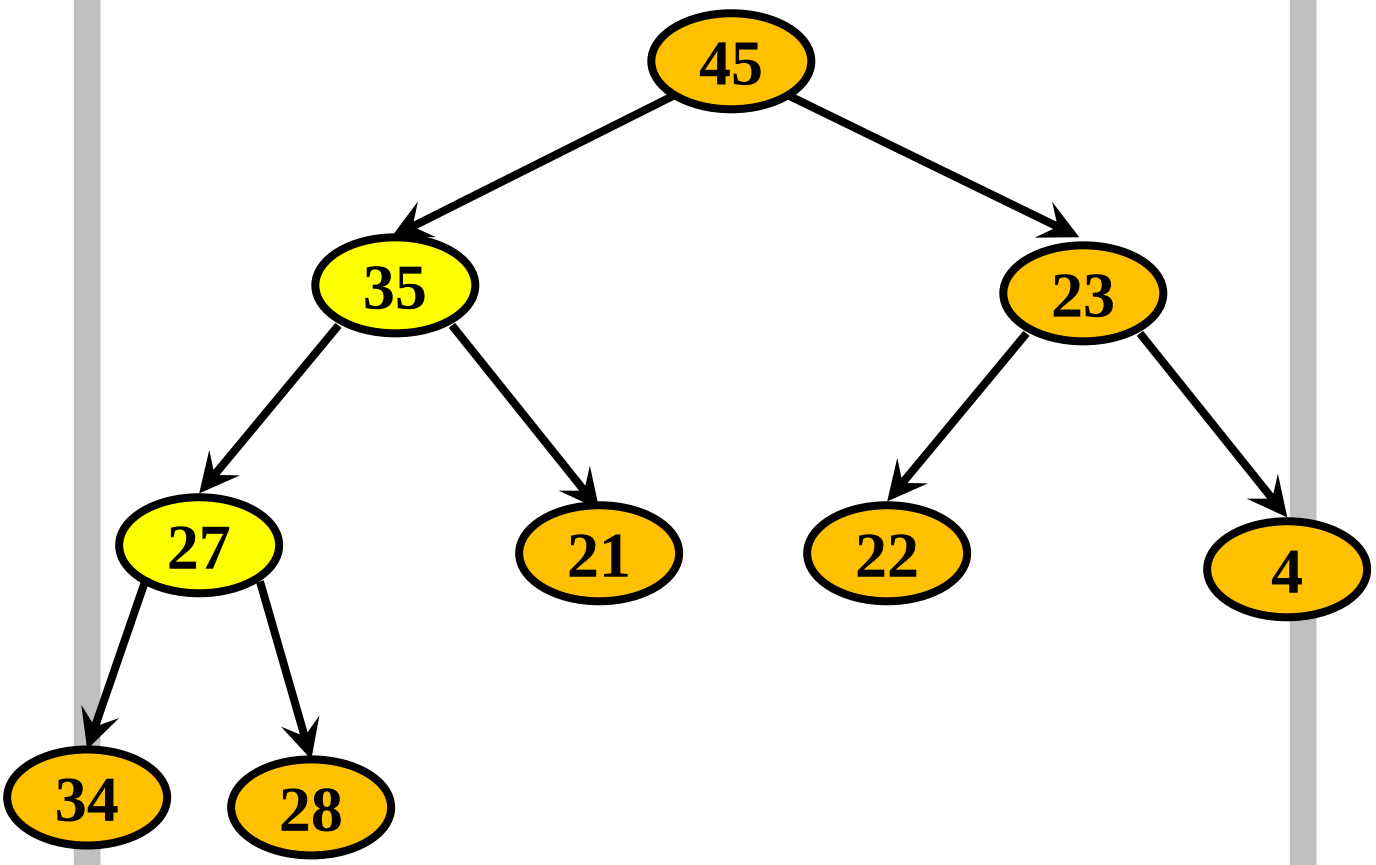
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



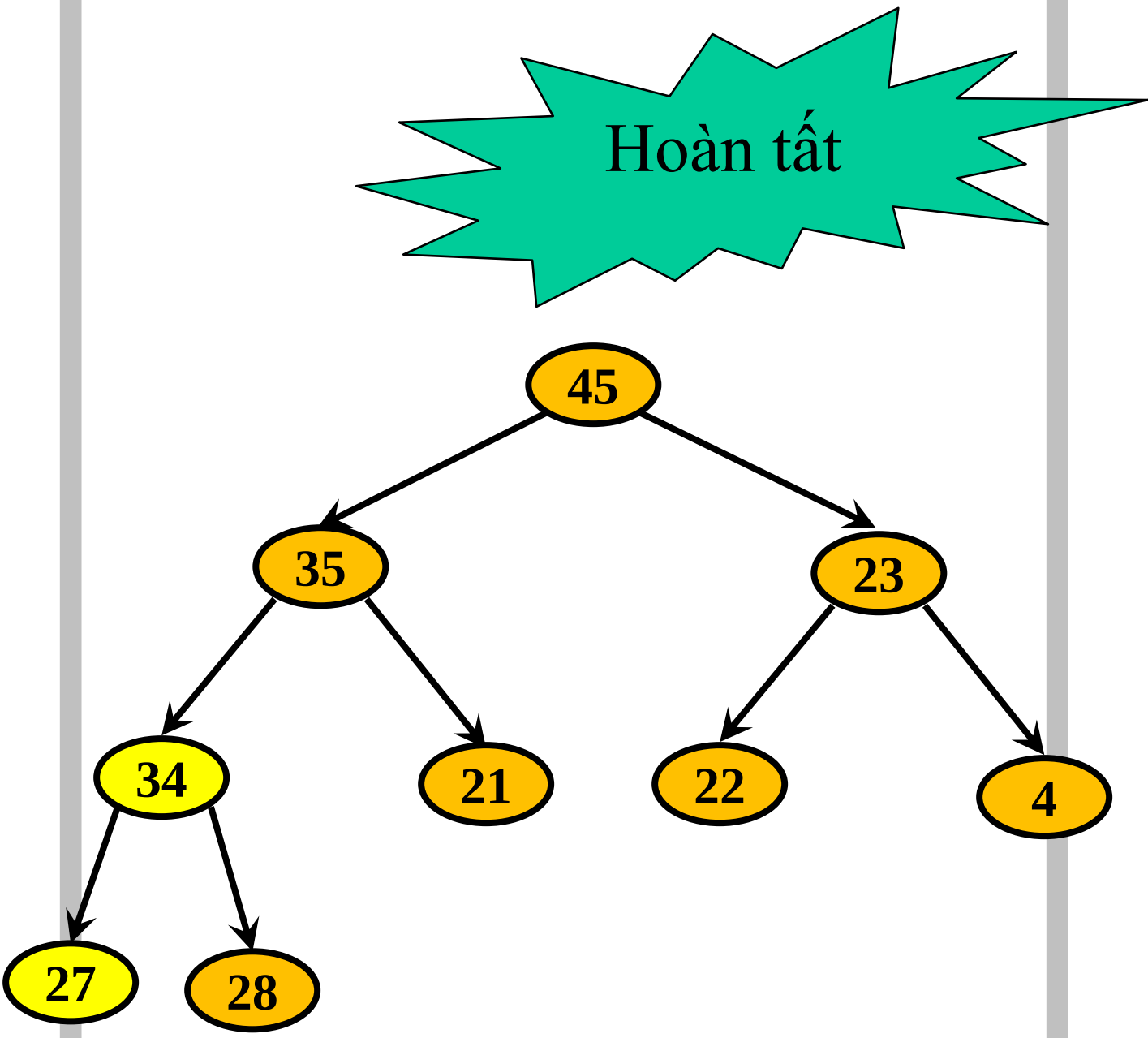
3. THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ Heapify

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

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



3. THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ Heapify



TS. Nguyễn Tấn Trần Minh Khang
ThS. Cáp Phạm đình Thăng

3. THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ Heapify

```
10. void Heapify(int a[],int n,
                 int vt)
11. {
12.     while(vt<=n/2-1)
13.     {
14.         int child1 = 2*vt+1;
15.         int child2 = 2*vt+2;
16.         int lc = child1;
17.         if(child2<n &&
18.            a[lc]<a[child2])
19.            lc = child2;
20.         if(a[vt]<a[lc])
21.            HoanVi(a[vt],a[lc]);
22.         vt = lc;
23.     }
24. }
```

3. THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ Heapify

```
10. void Heapify(int a[],int n,
                 int vt)
11. {
12.     while(vt<n/2)
13.     {
14.         int child1 = 2*vt+1;
15.         int child2 = 2*vt+2;
16.         int lc = child1;
17.         if(child2<n &&
               a[lc]<a[child2])
18.             lc = child2;
19.         if(a[vt]<a[lc])
20.             HoanVi(a[vt],a[lc]);
21.         vt = lc;
22.     }
23. }
```

3. THAO TÁC ĐIỀU CHỈNH MỘT 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. }
```

4. XÂY DỰNG HEAP

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

4. 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. }
```

4. XÂY DỰNG HEAP

```
10. void Heapify(int a[],int n,
                 int vt)
11. {
12.     while(vt<=n/2-1)
13.     {
14.         int child1 = 2*vt+1;
15.         int child2 = 2*vt+2;
16.         int lc = child1;
17.         if(child2<n &&
               a[lc]<a[child2])
18.             lc = child2;
19.         if(a[vt]<a[lc])
20.             HoanVi(a[vt],a[lc]);
21.         vt = lc;
22.     }
23. }
```

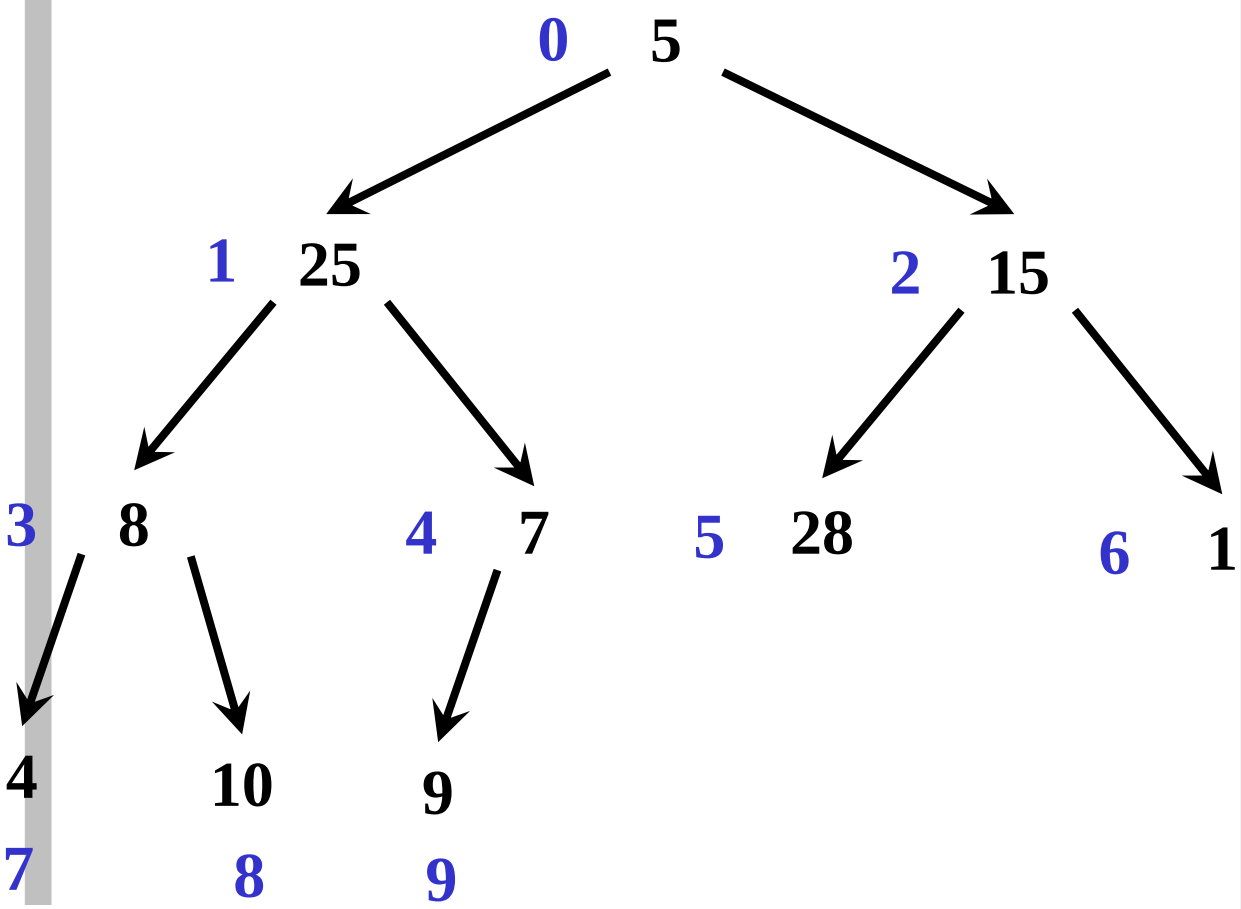
4. XÂY DỰNG HEAP

- Bài toán: Hãy xây dựng mảng sau thành một Heap.

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

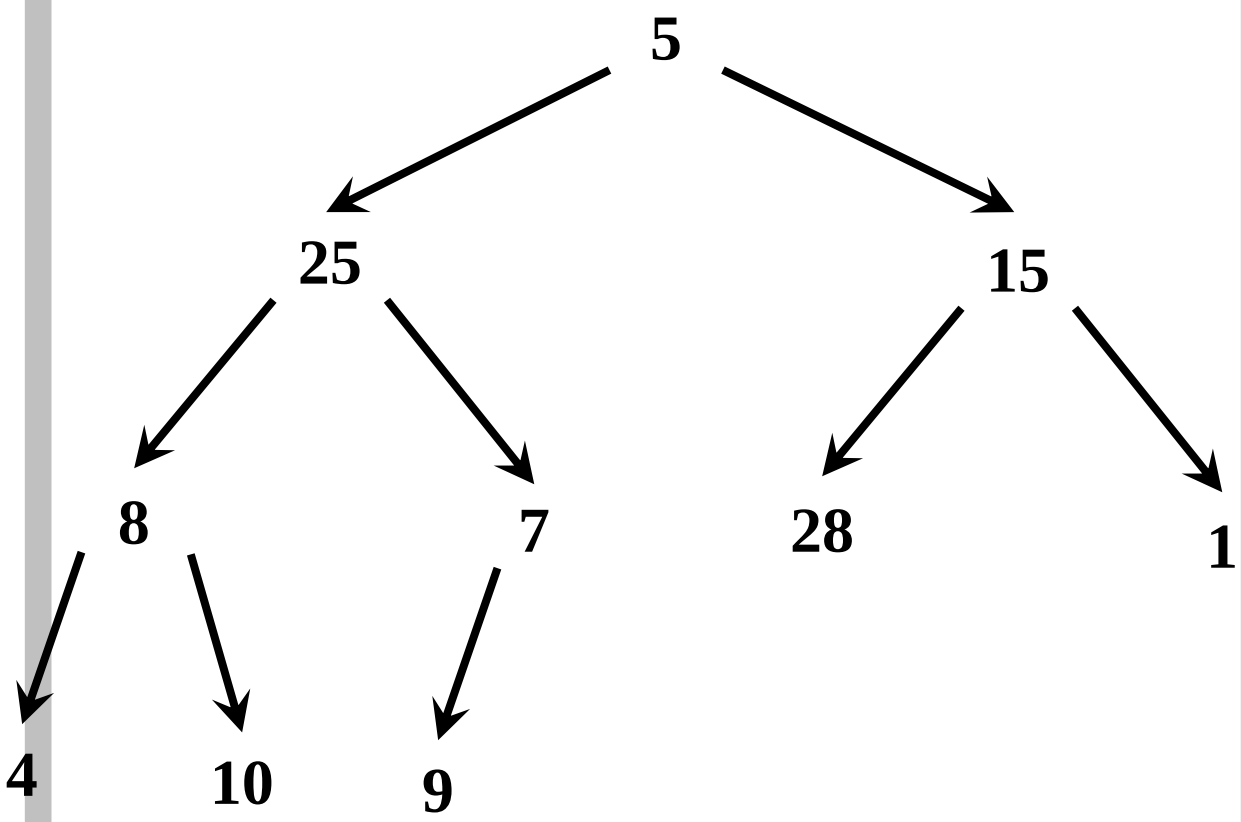
4. XÂY DỰNG HEAP

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



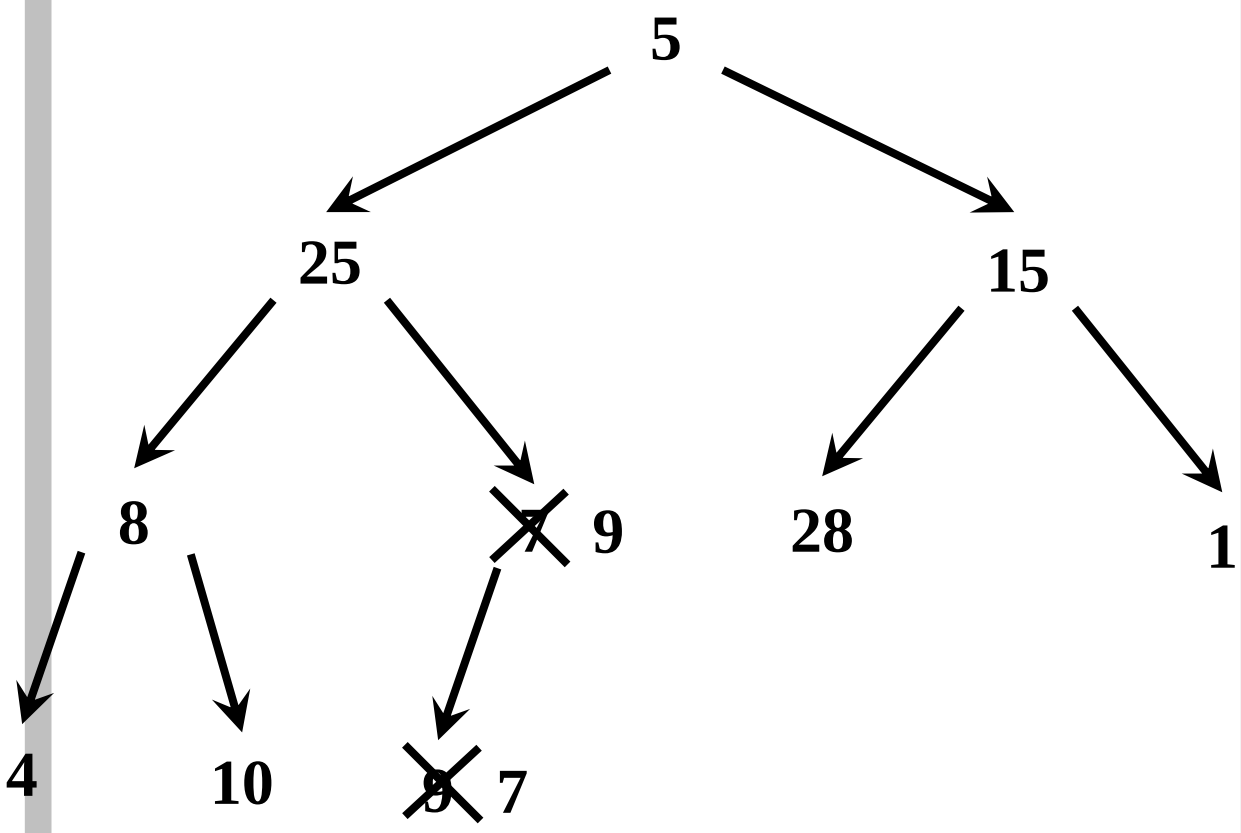
4. XÂY DỰNG HEAP

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



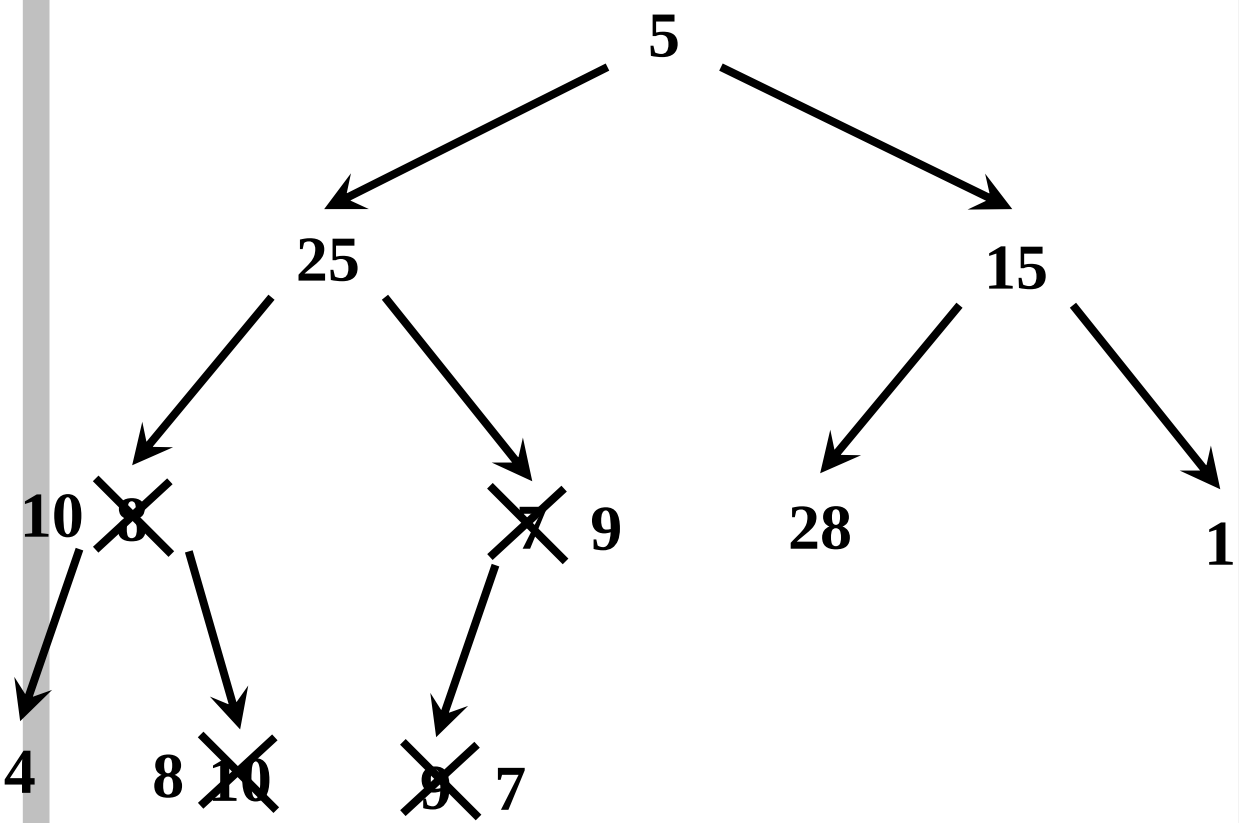
4. XÂY DỰNG HEAP

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



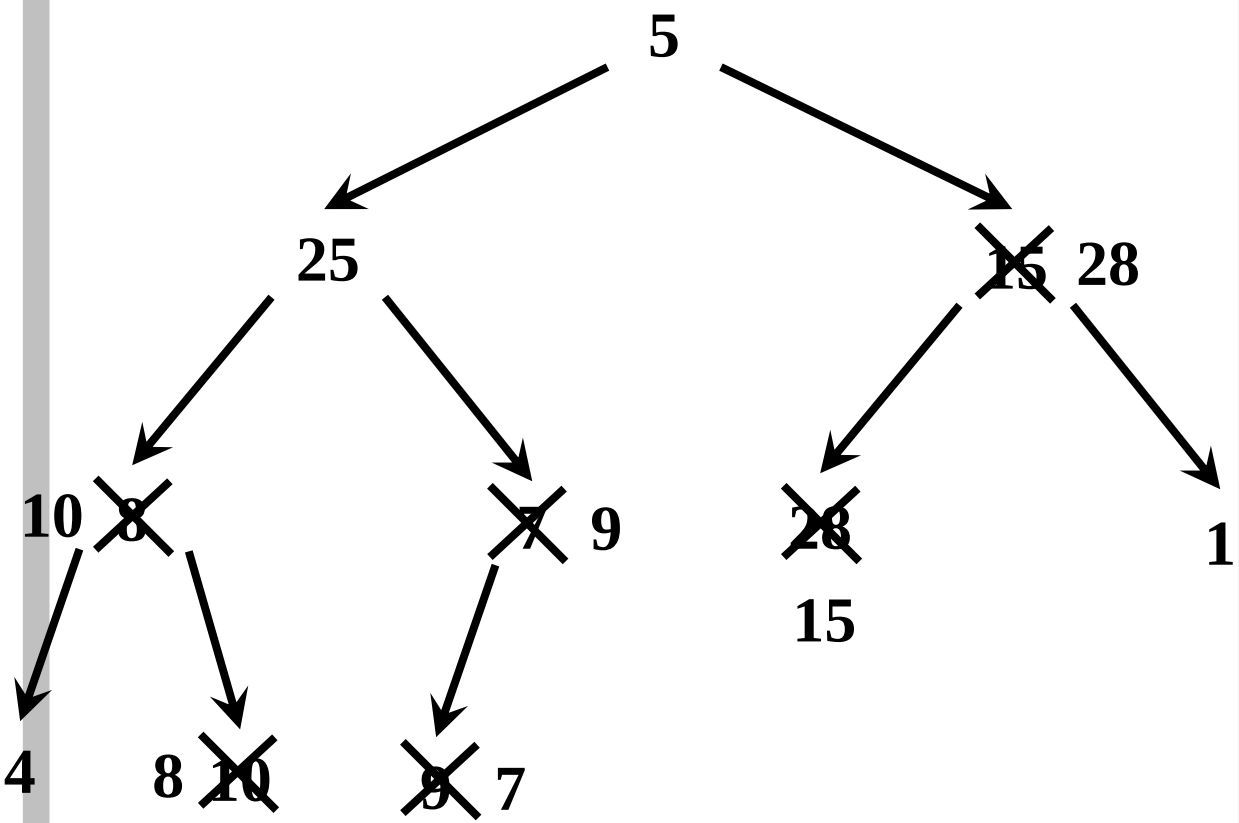
4. XÂY DỰNG HEAP

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

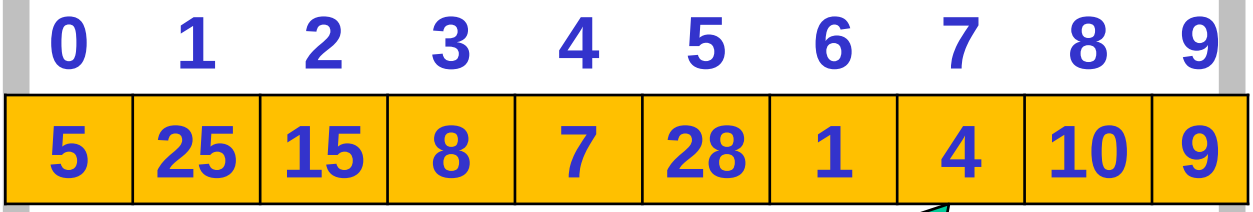


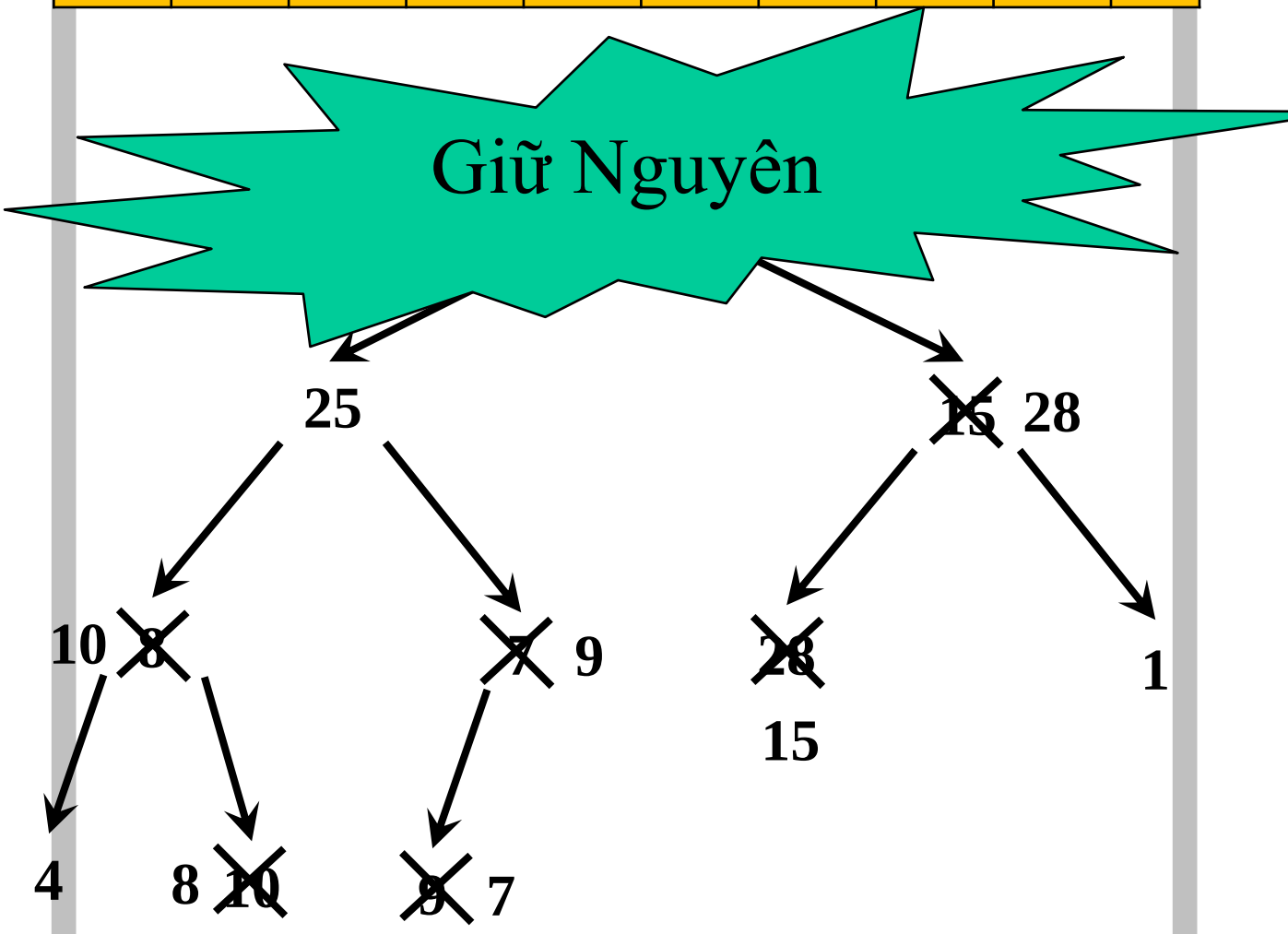
4. XÂY DỰNG HEAP

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



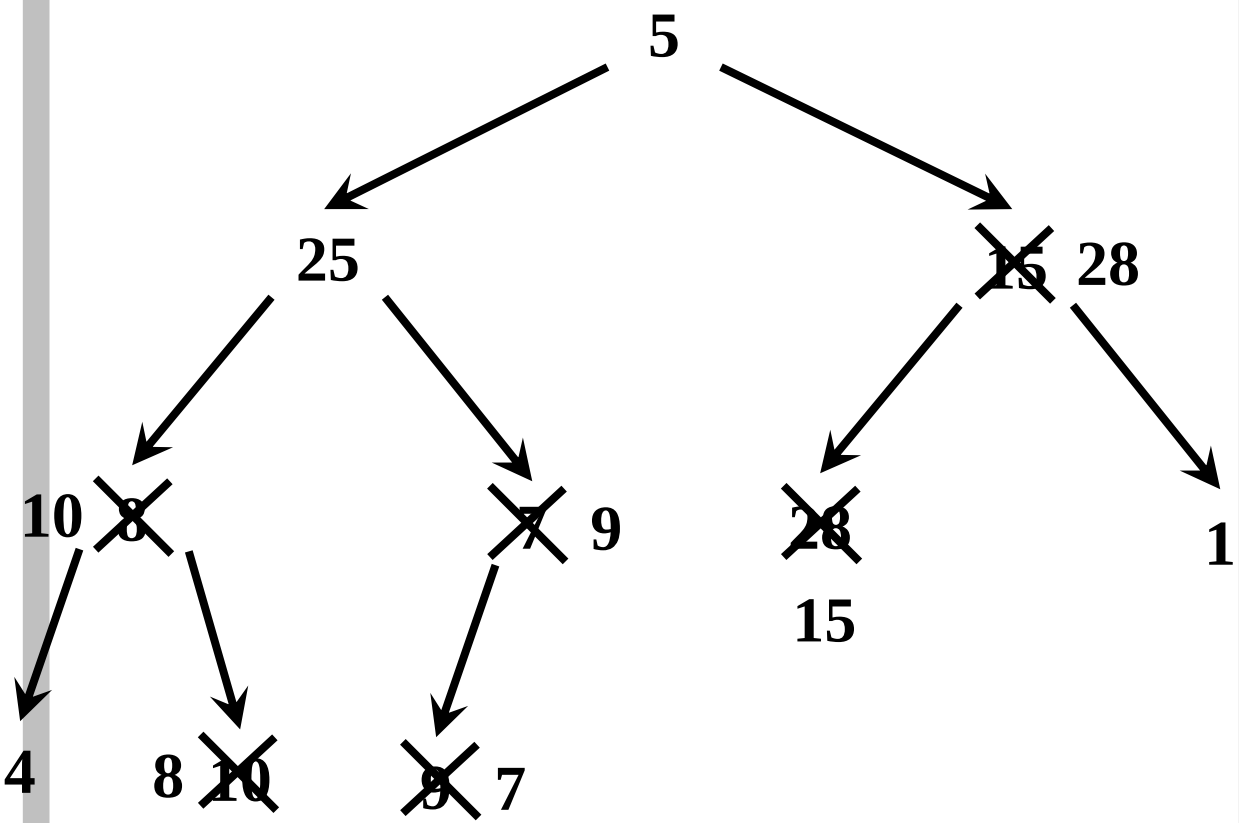
4. XÂY DỰNG HEAP





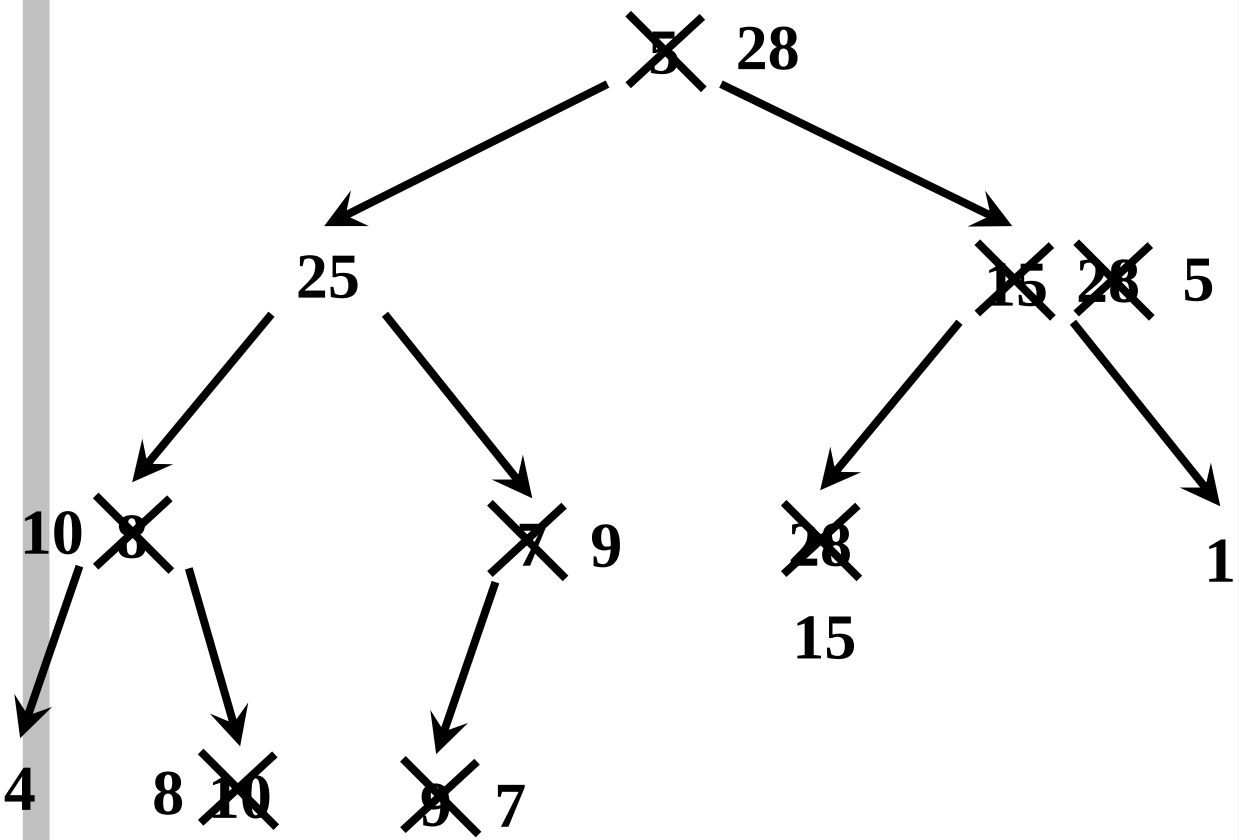
4. XÂY DỰNG HEAP

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



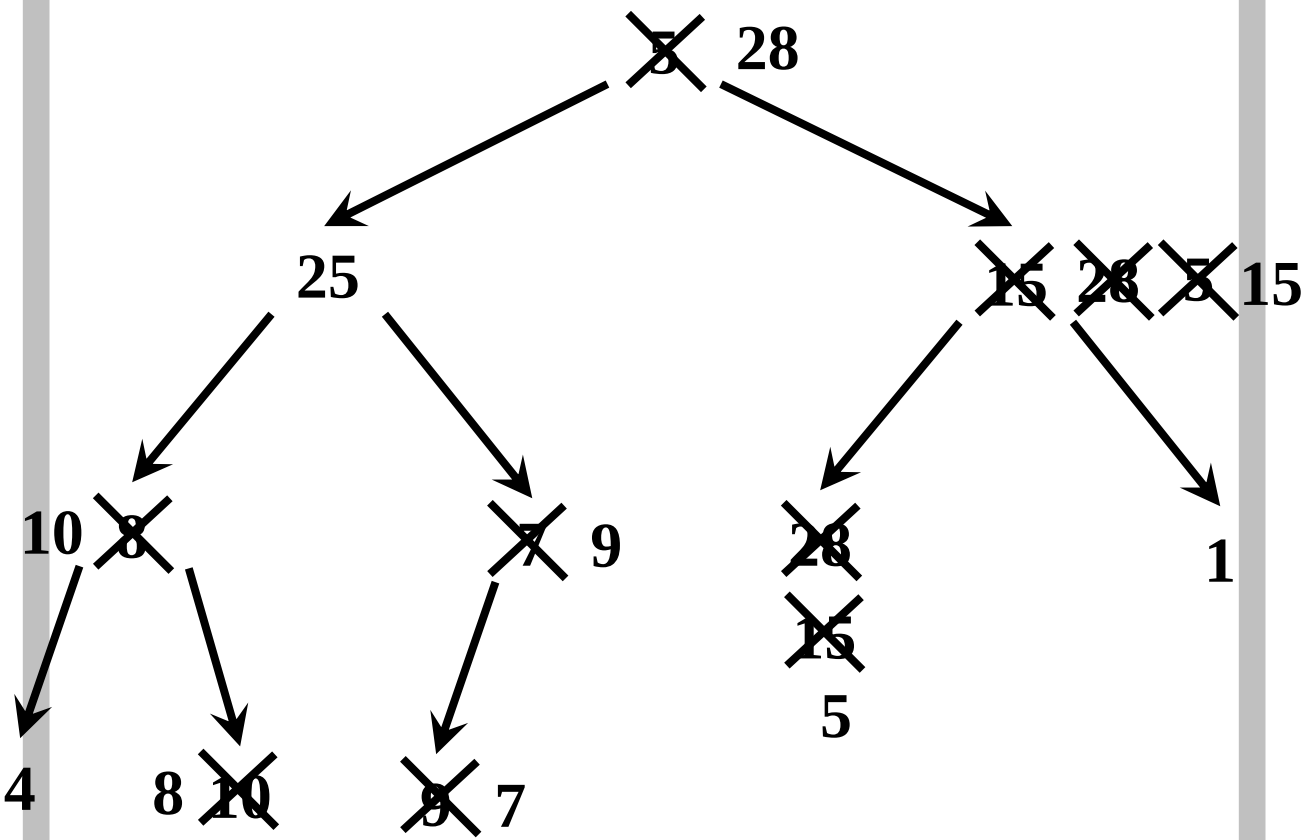
4. XÂY DỰNG HEAP

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



4. XÂY DỰNG HEAP

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



4. XÂY DỰNG HEAP

- Bài toán: Hãy xây dựng mảng sau thành một Heap.

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

- Kết quả hiệu chỉnh mảng trên thành Heap là:

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

4. XÂY DỰNG HEAP

- Bài tập: Hãy xây dựng mảng sau thành một Heap.

0	1	2	3	4	5	6	7	8	9
-7	75	28	81	47	89	92	15	57	78

- Kết quả hiệu chỉnh mảng trên thành Heap là:

0	1	2	3	4	5	6	7	8	9
92	81	89	75	78	-7	28	15	57	47

5. 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.

5. 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. }
```

5. 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. }
```

5. THUẬT TOÁN HEAP SORT

```
1. void Heapify(int a[],int n,
                int vt)
2. {
3.     while(vt<=n/2-1)
4.     {
5.         int child1 = 2*vt+1;
6.         int child2 = 2*vt+2;
7.         int lc = child1;
8.         if(child2<n &&
               a[lc]<a[child2])
9.             lc = child2;
10.        if(a[vt]<a[lc])
11.            HoanVi(a[vt],a[lc]);
12.        vt = lc;
13.    }
14. }
```

6. ÁP DỤNG THUẬT TOÁN

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

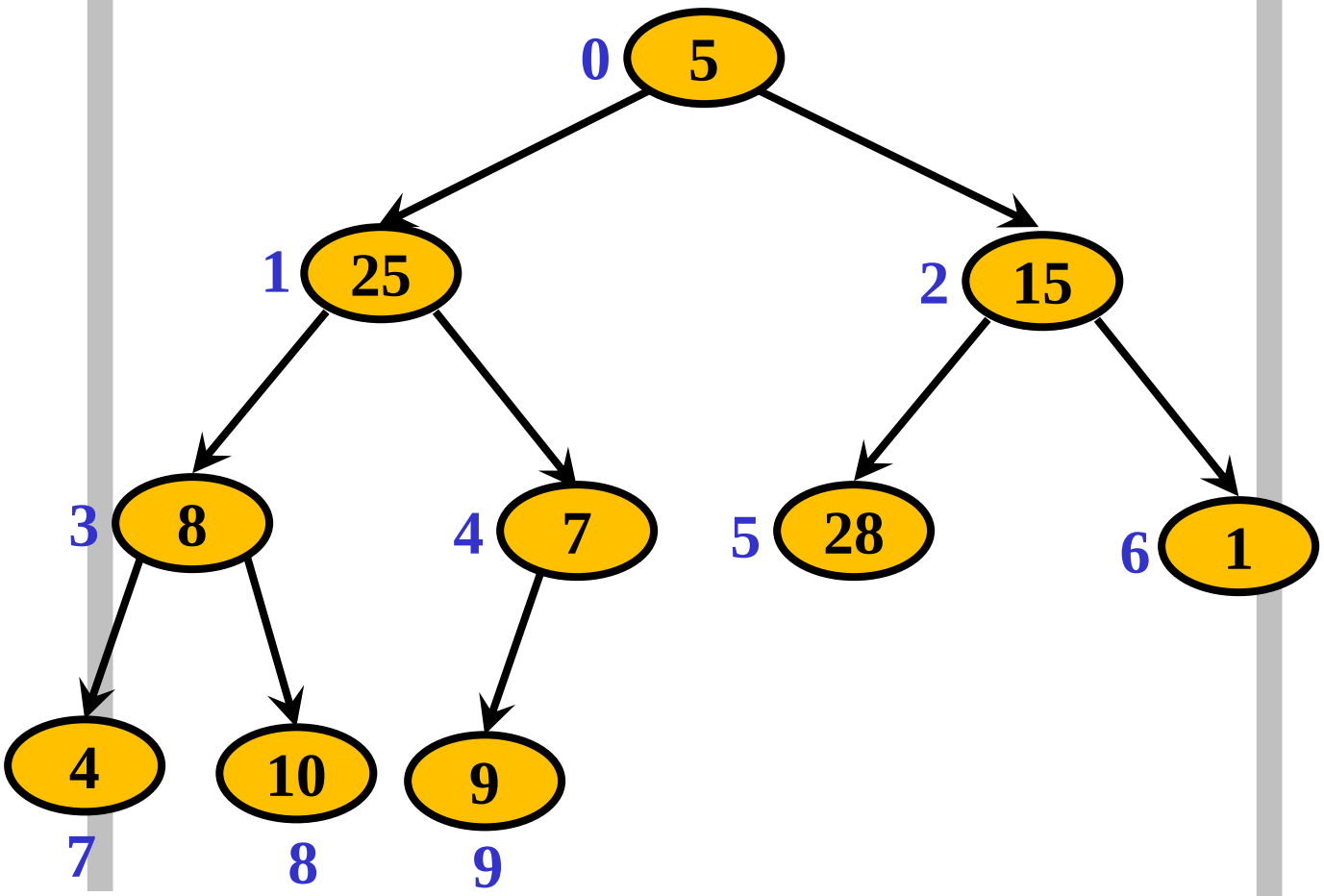
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

- **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.

ÁP DỤNG THUẬT TOÁN

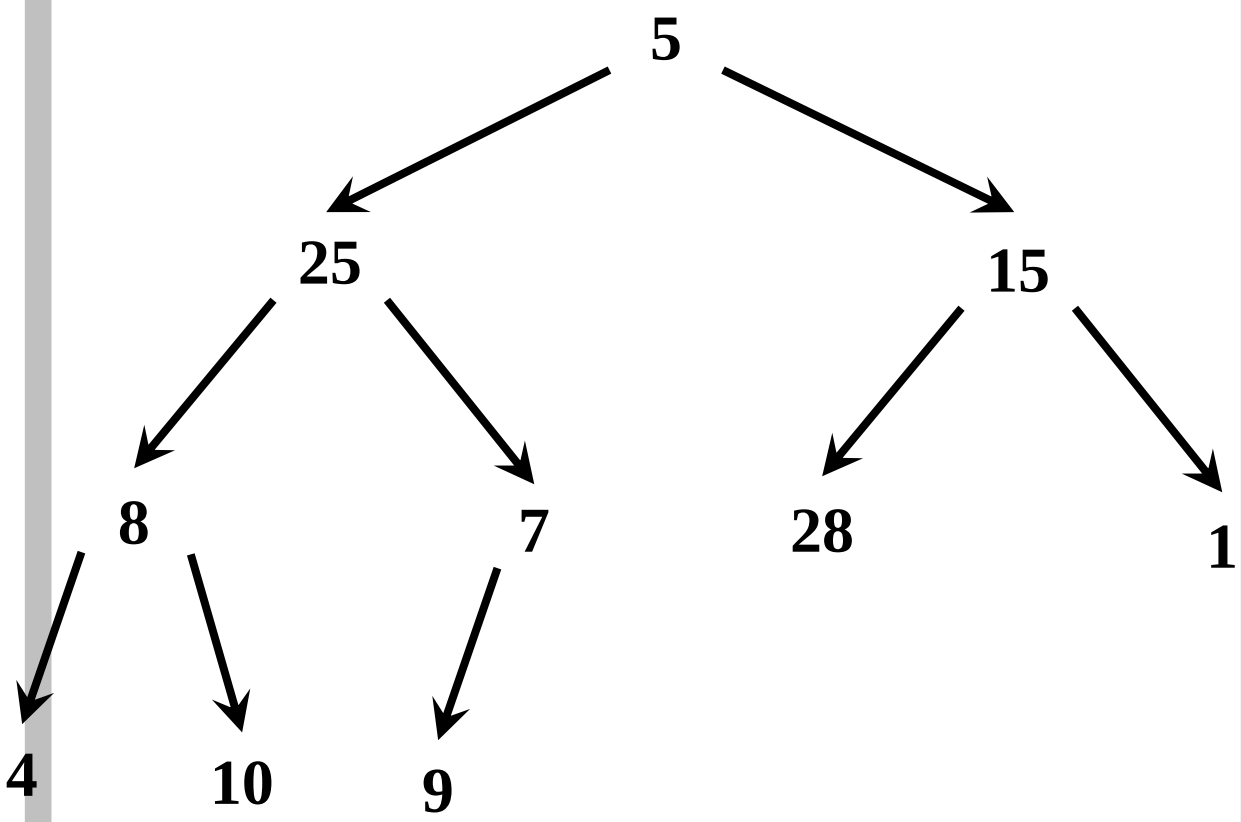
- Bước 1: Xây dựng Heap

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



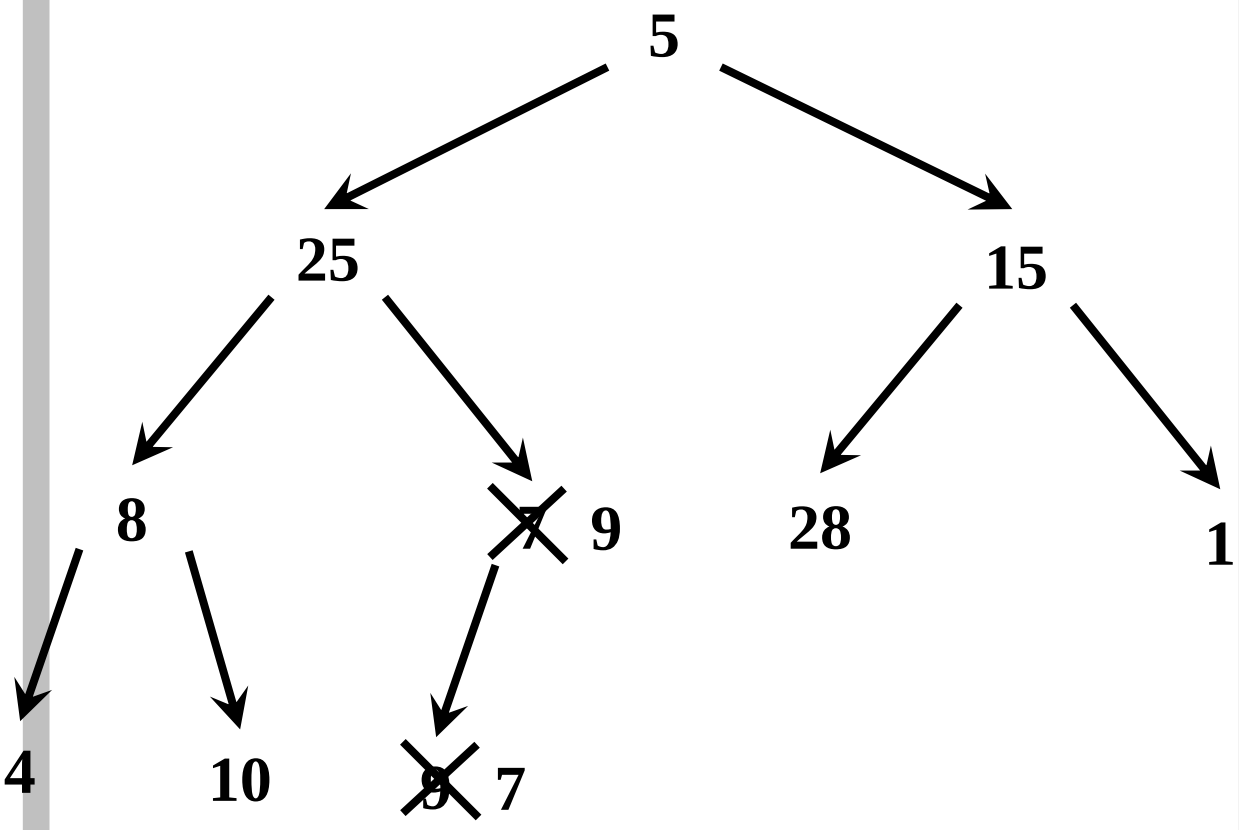
ÁP DỤNG THUẬT TOÁN

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



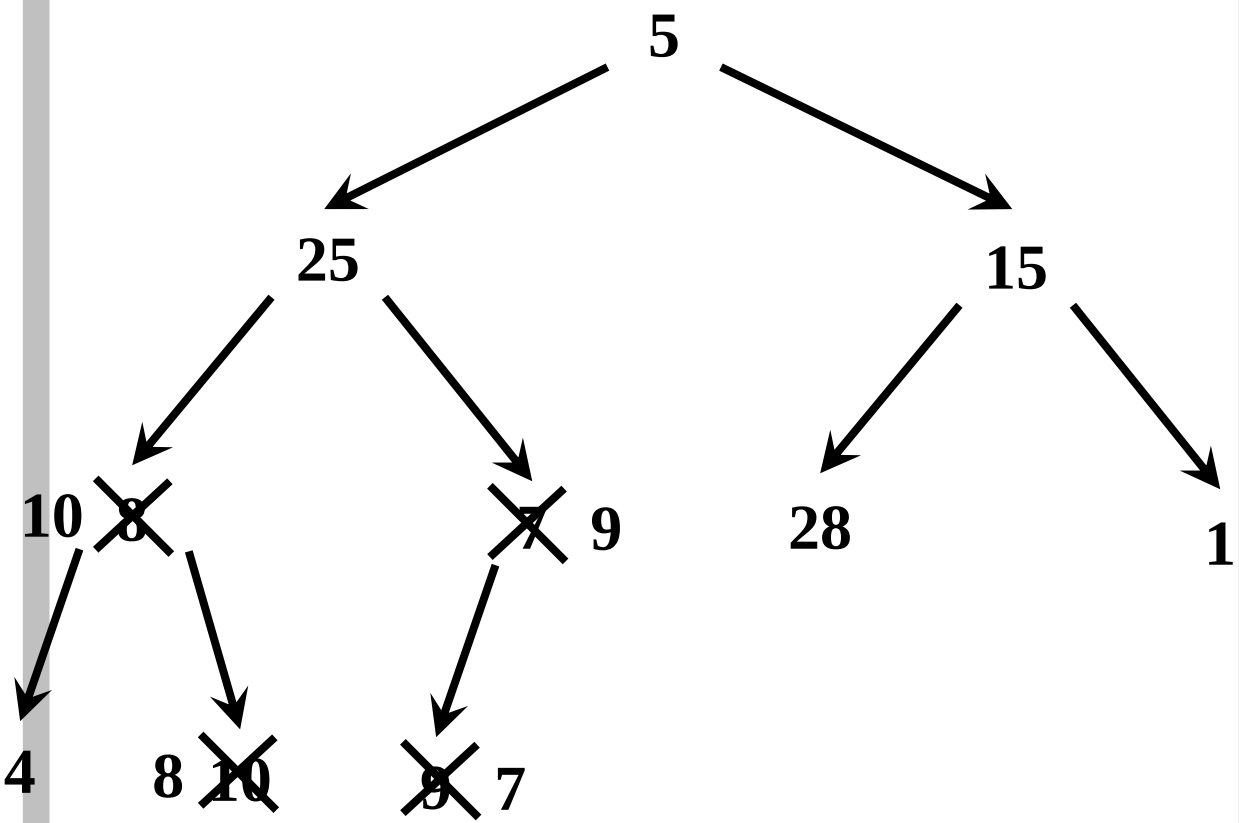
ÁP DỤNG THUẬT TOÁN

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



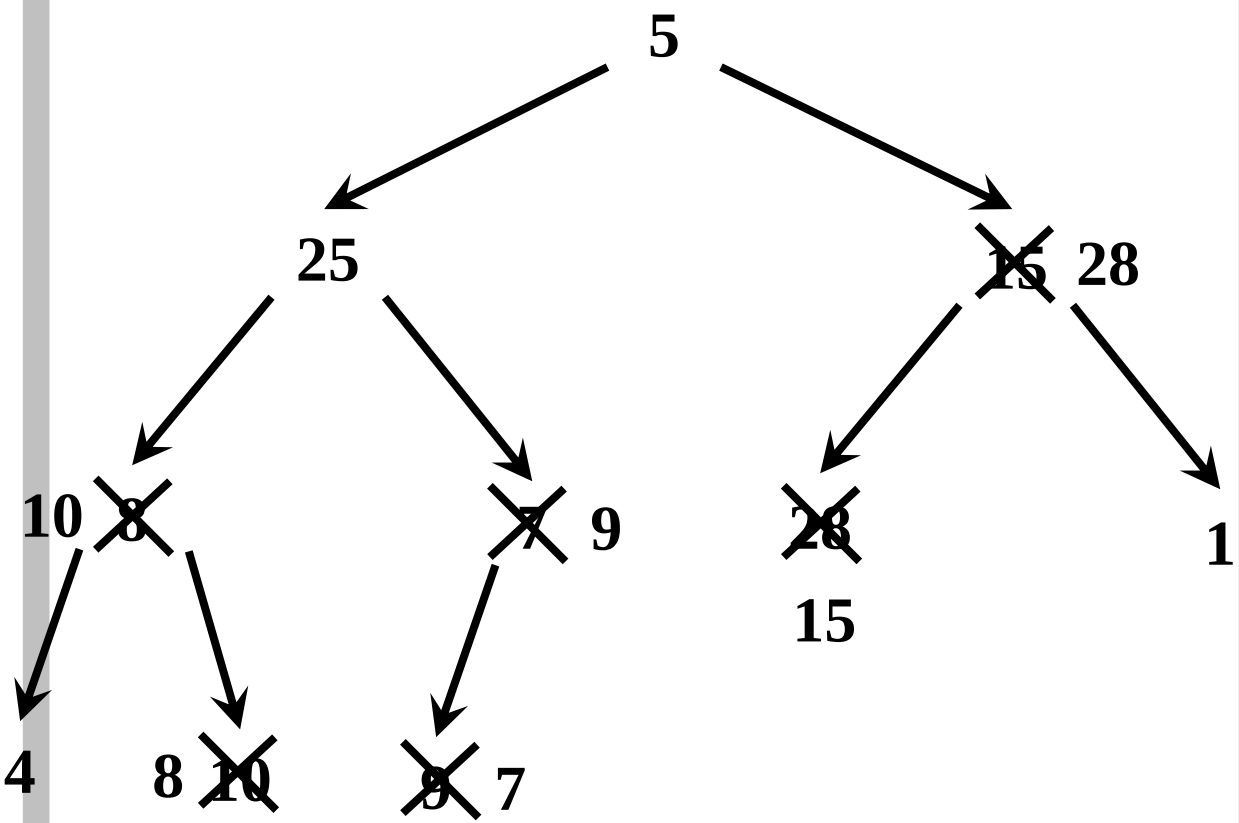
ÁP DỤNG THUẬT TOÁN

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



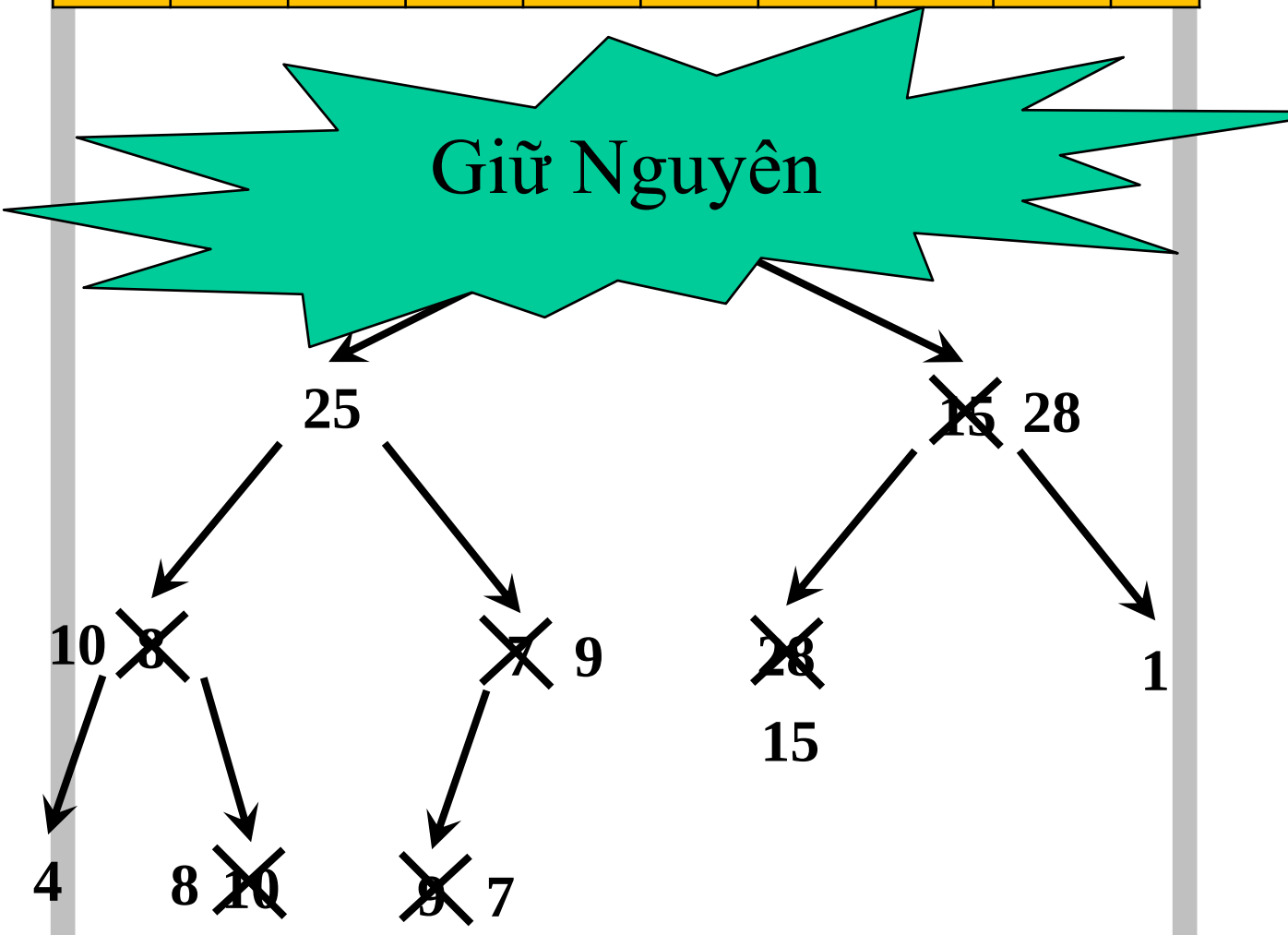
ÁP DỤNG THUẬT TOÁN

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



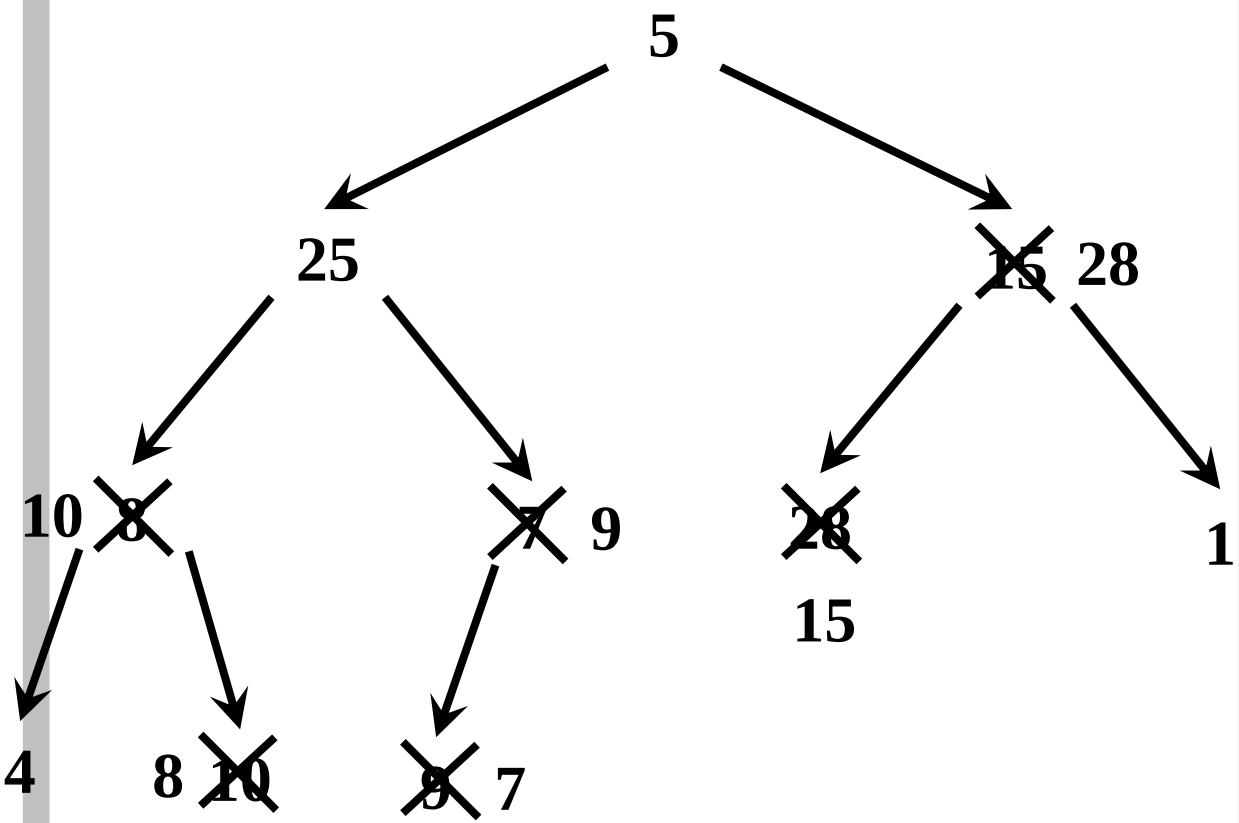
ÁP DỤNG THUẬT TOÁN

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



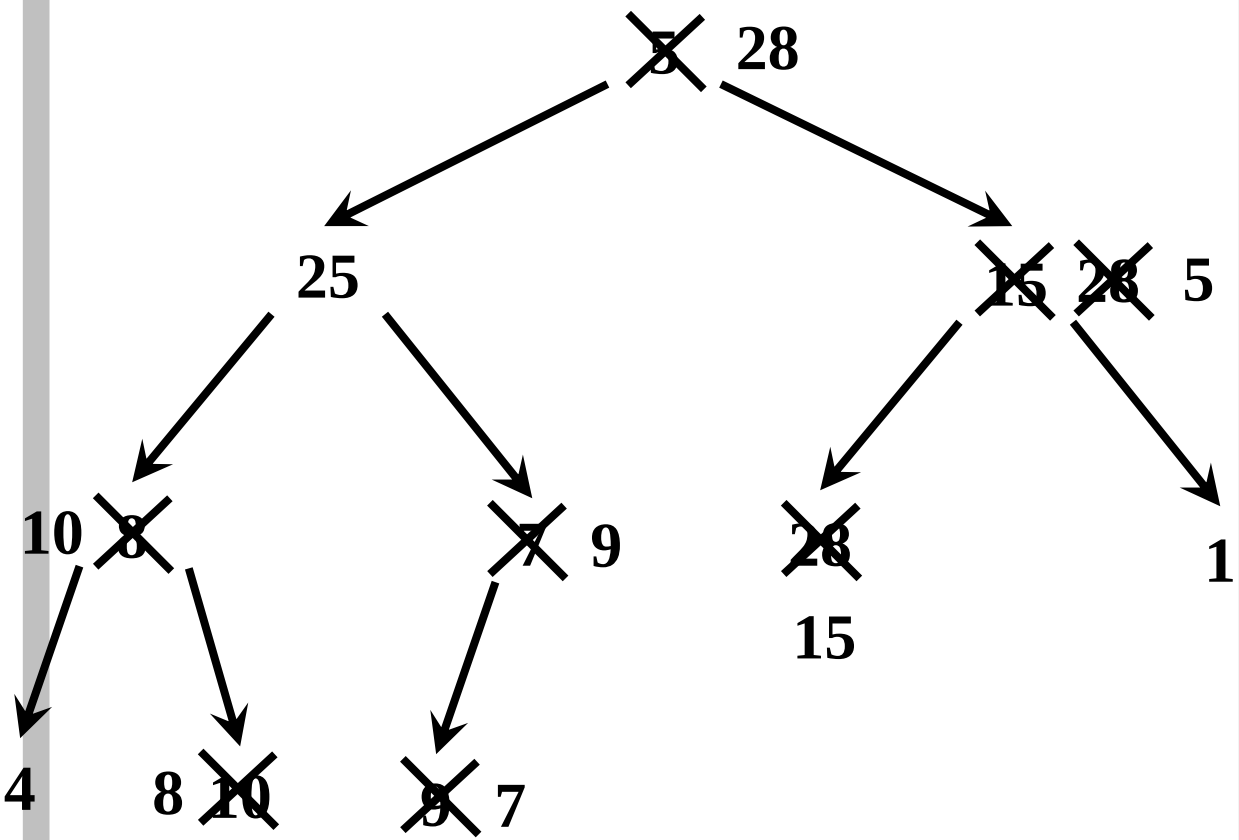
ÁP DỤNG THUẬT TOÁN

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



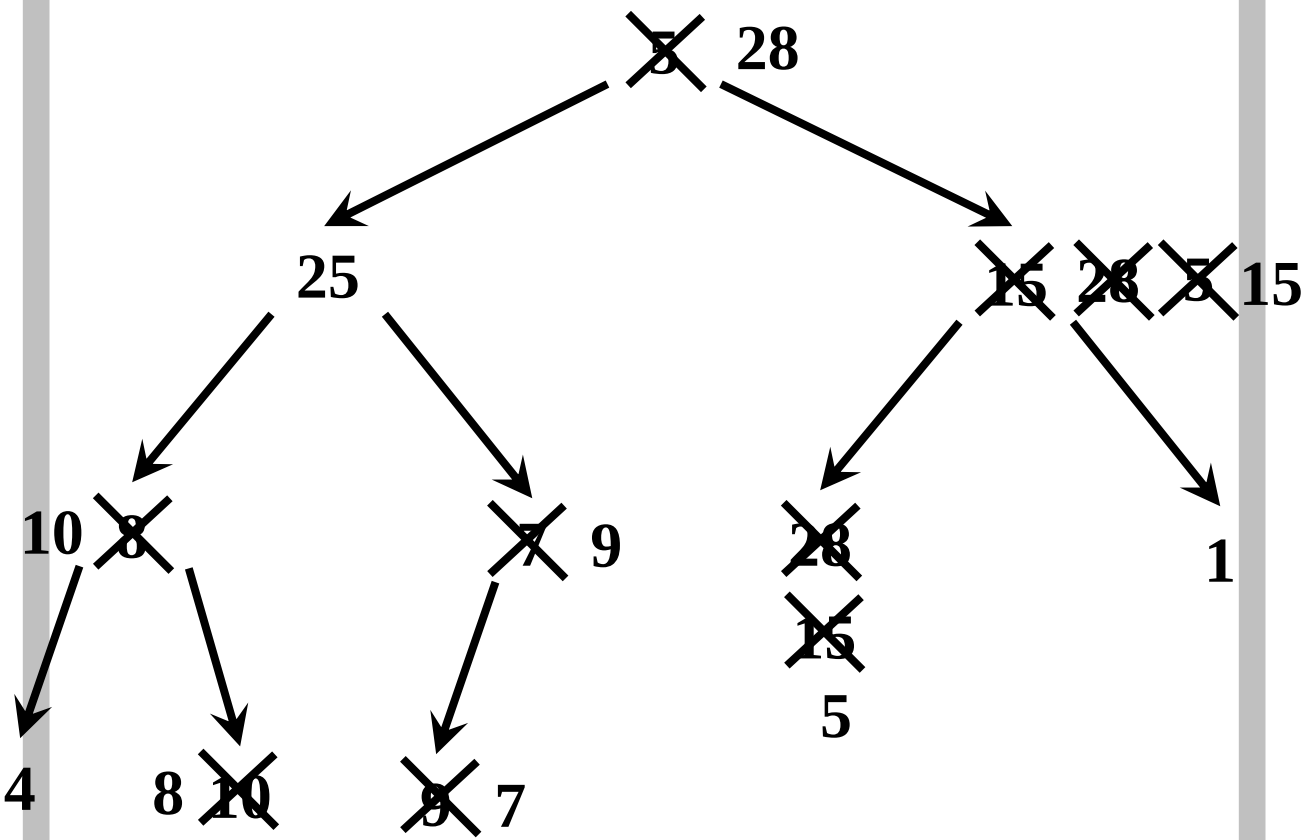
ÁP DỤNG THUẬT TOÁN

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



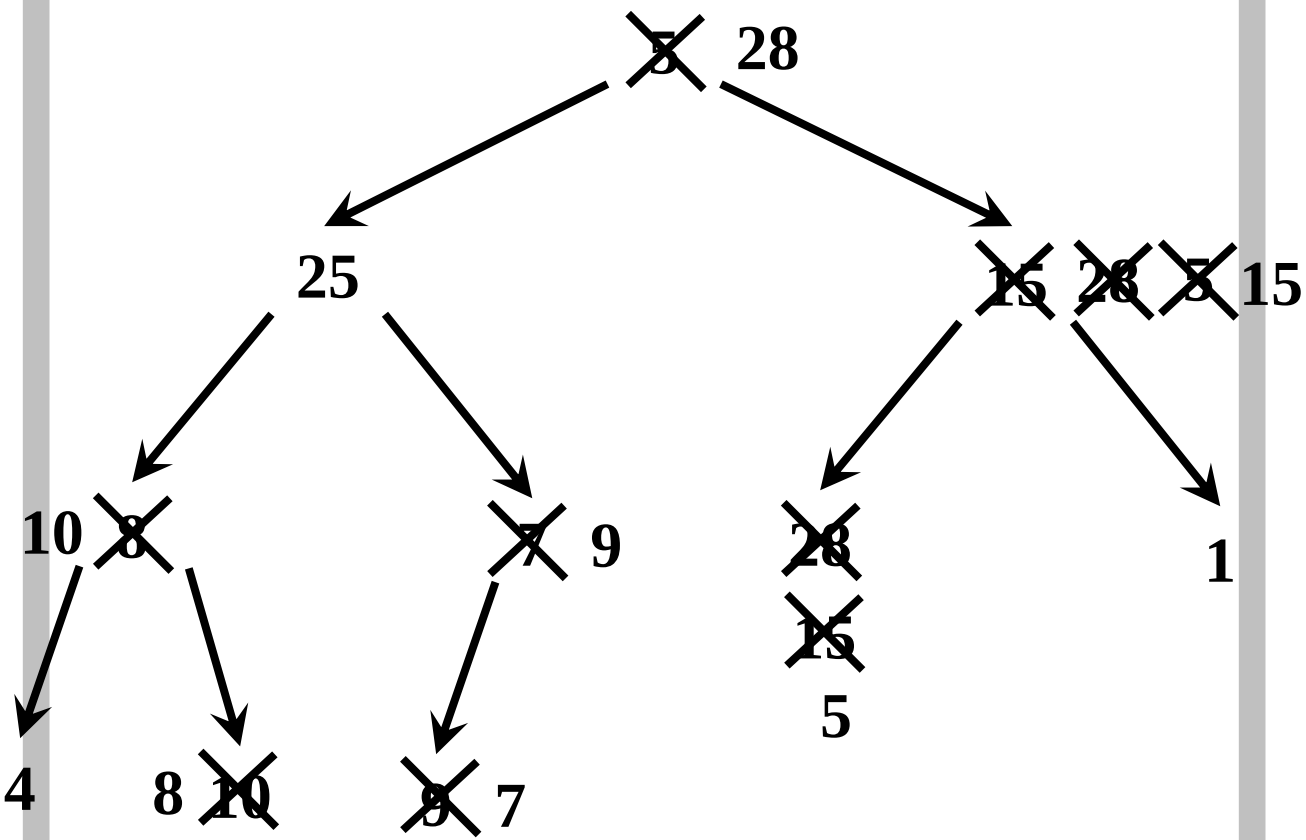
ÁP DỤNG THUẬT TOÁN

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



ÁP DỤNG THUẬT TOÁN

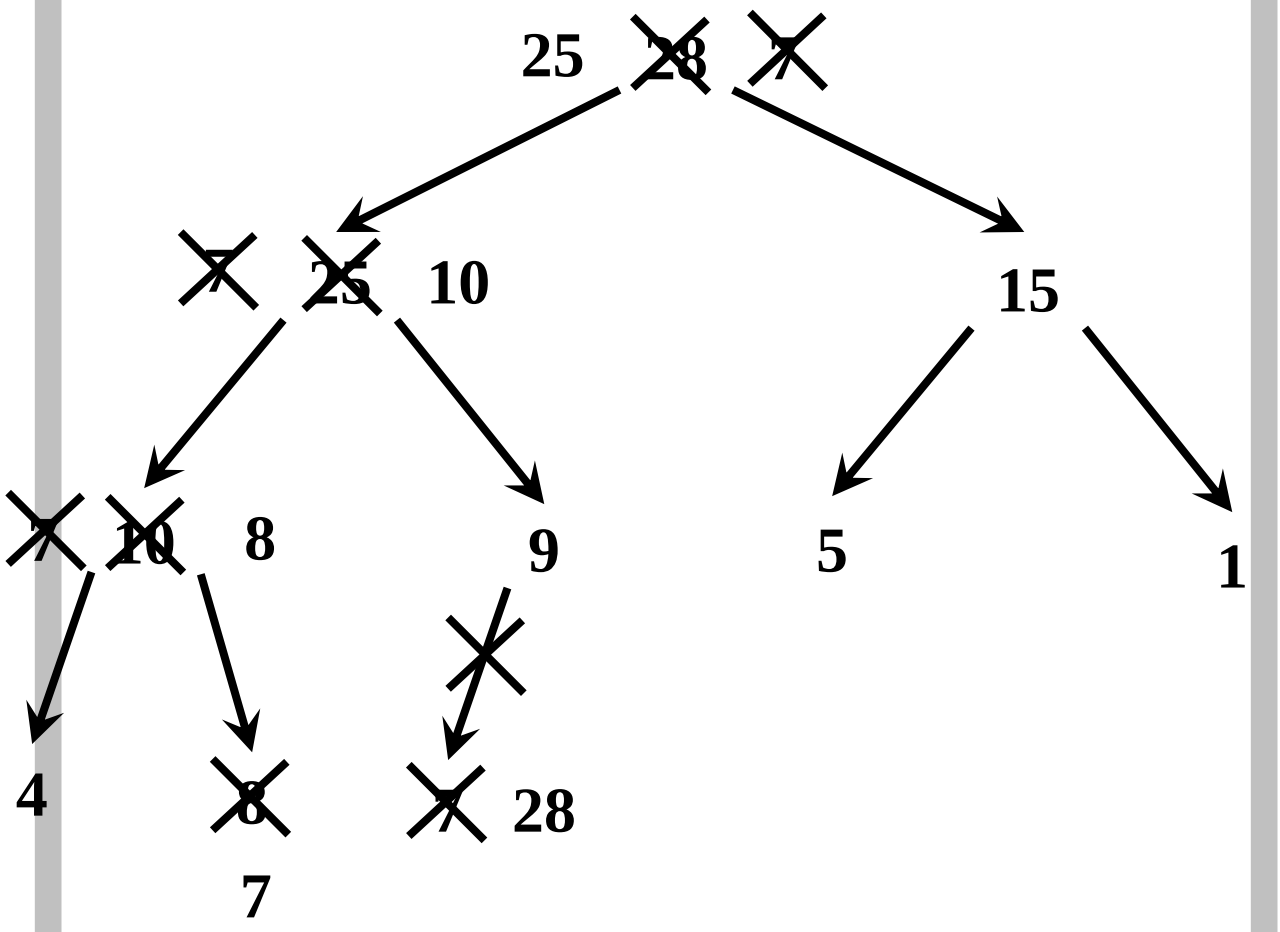
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



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

– Bước 2: Sắp xếp – **Lần lặp 1**

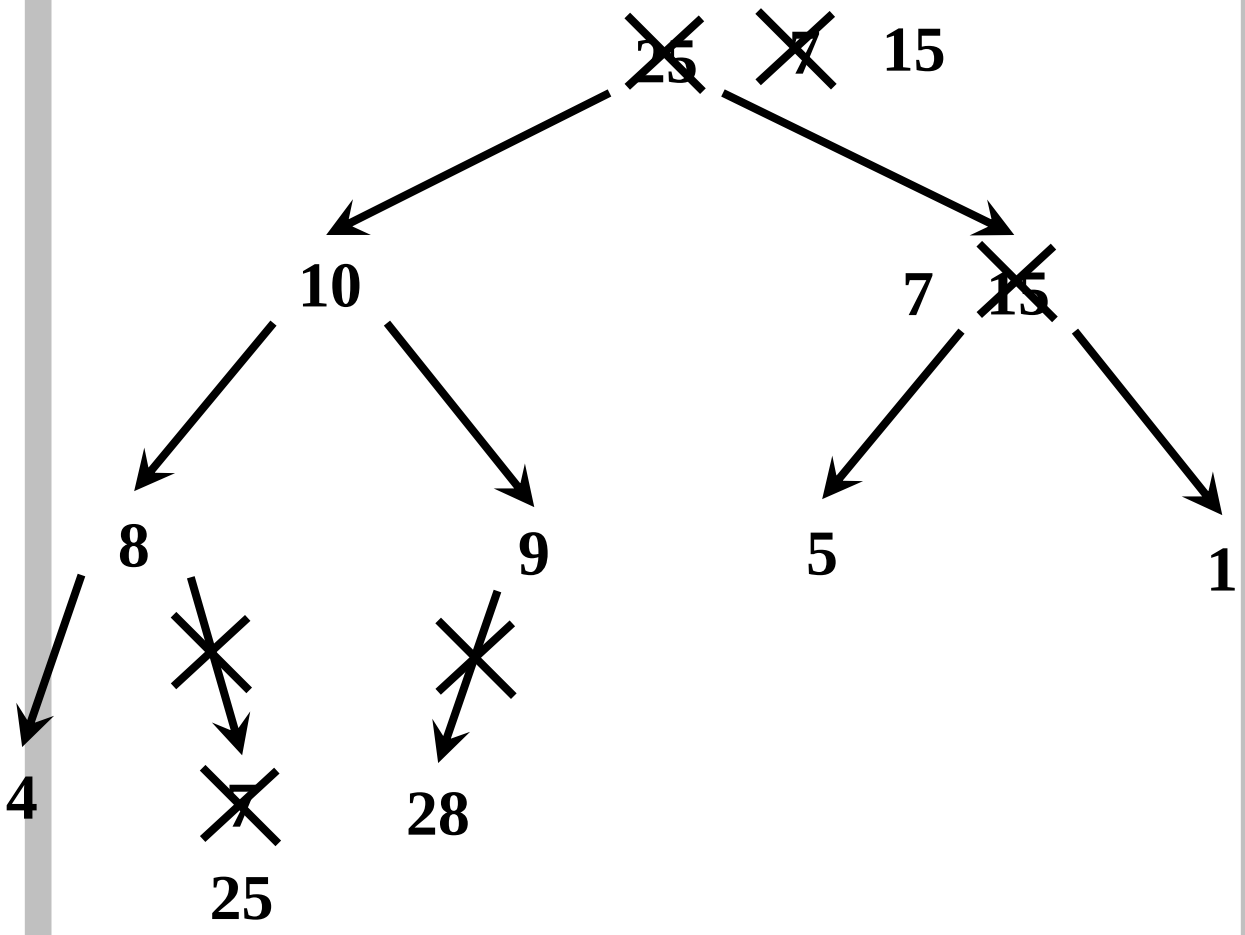
0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7



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

– Bước 2: Sắp xếp – **Lần lặp 2**

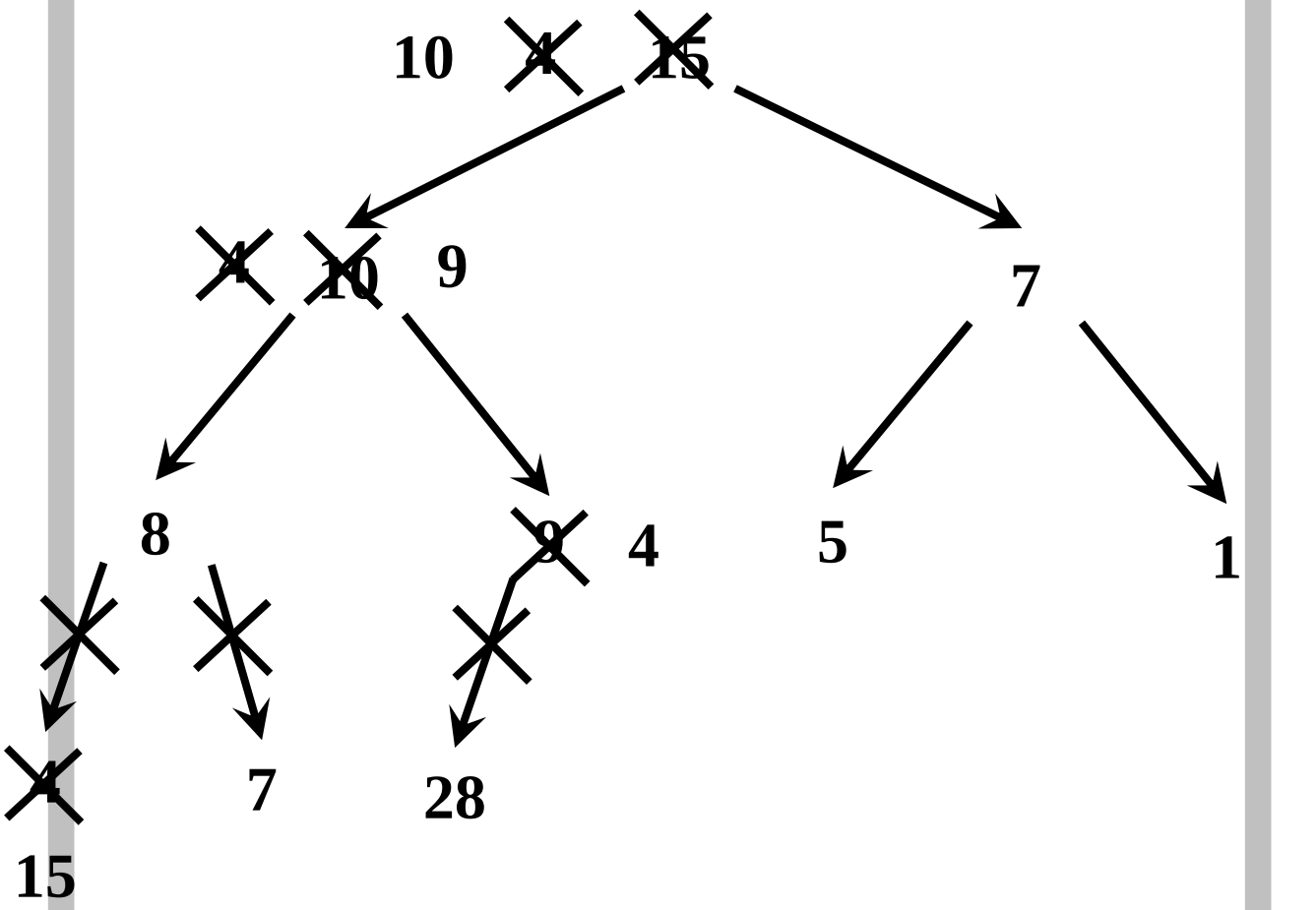
0	1	2	3	4	5	6	7	8	9
25	10	15	8	9	5	1	4	7	28



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

- Bước 2: Sắp xếp – **Lần lặp 3**

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



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

7. BÀI TẬP

- **Sắp mảng một chiều số nguyên giảm dần bằng thuật toán HeapSort.**

```
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. }
```

5. 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. }
```

5. THUẬT TOÁN HEAP SORT

```
1. void Heapify(int a[],int n,
                int vt)
2. {
3.     while(vt<=n/2-1)
4.     {
5.         int child1 = 2*vt+1;
6.         int child2 = 2*vt+2;
7.         int lc = child1;
8.         if(child2<n &&
               a[lc]<a[child2])
9.             lc = child2;
10.        if(a[vt]>a[lc])
11.            HoanVi(a[vt],a[lc]);
12.        vt = lc;
13.    }
14. }
```