THUẬT TOÁN HEAP SORT

1. Hồ Thái Ngọc
2. ThS. Võ Duy Nguyên
3. TS. Nguyễn Tấn Trần Minh Khang

NỘI DUNG

Nội dung

1. Định nghĩa Heap
2. Biểu diễn Heap bằng mảng
3. Thao tác điều chỉnh một phần tử Heapify
4. Xây dựng Heap
5. Thuật toán Heap sort
6. Chạy từng bước thuật toán

ĐỊNH NGHĨA HEAP

Định nghĩa heap

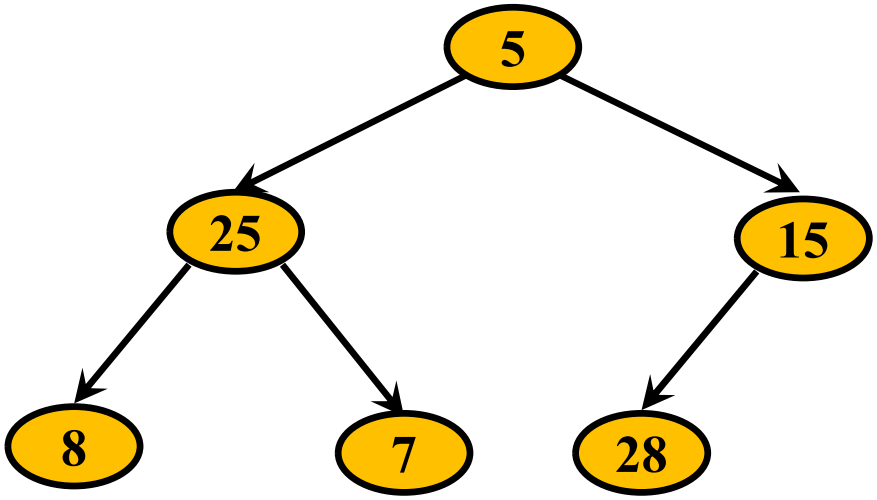
- Heap là một cây nhị phân thỏa các tính chất sau:
  - + Heap là một cây nhị phân gần đầy đủ (gần hoàn chỉnh);
  - + Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con.

Định nghĩa heap

- Heap là một cây nhị phân thỏa các tính chất sau:
  + Heap là một cây nhị phân gần đầy đủ;

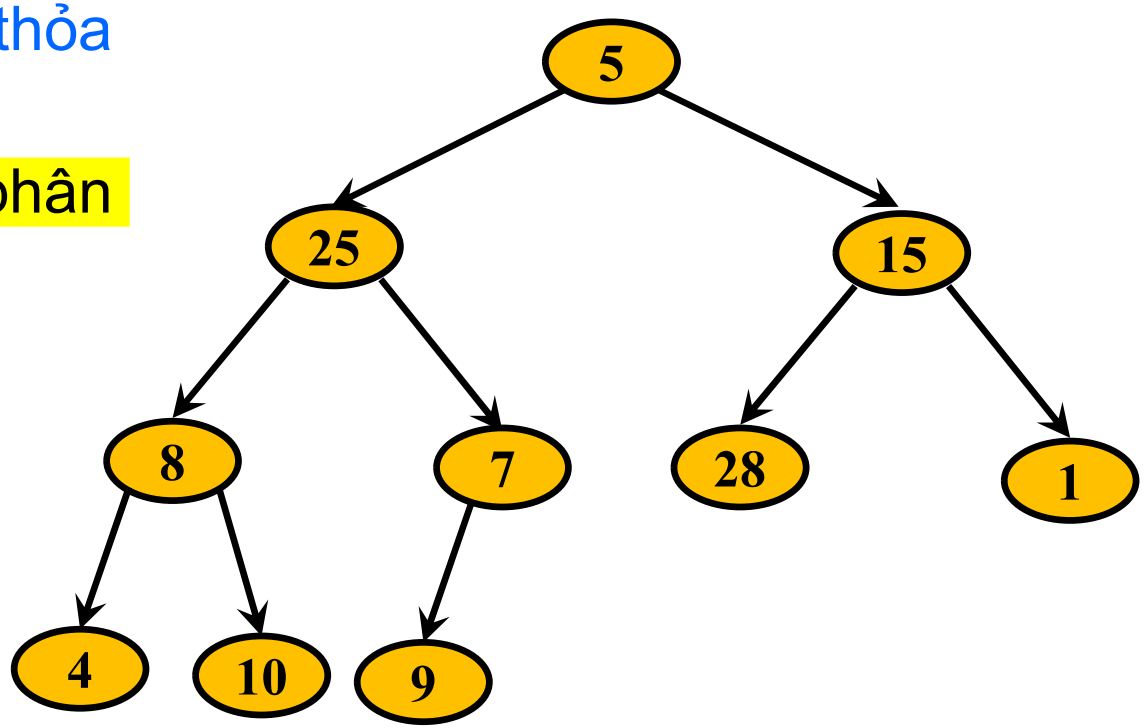
Định nghĩa heap

- Heap là một cây nhị phân thỏa các tính chất sau:
  - + Heap là một cây nhị phân gần đầy đủ;



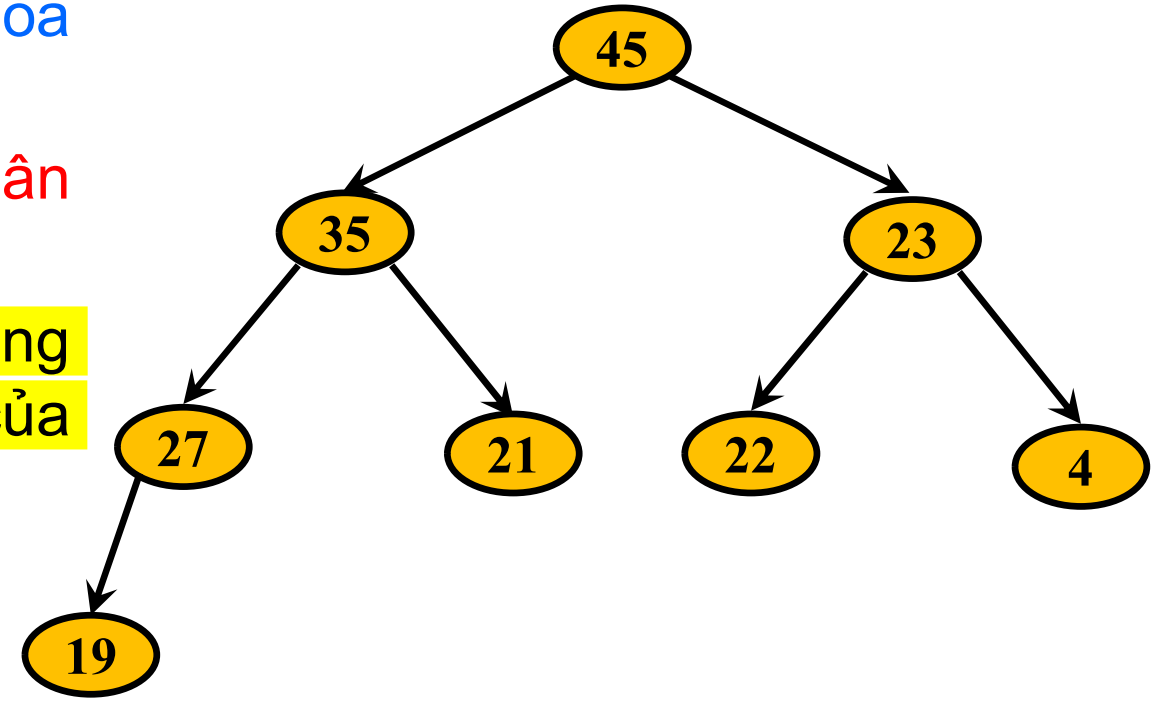
Định nghĩa heap

- Heap là một cây nhị phân thỏa các tính chất sau:
  - + Heap là một cây nhị phân gần đầy đủ;



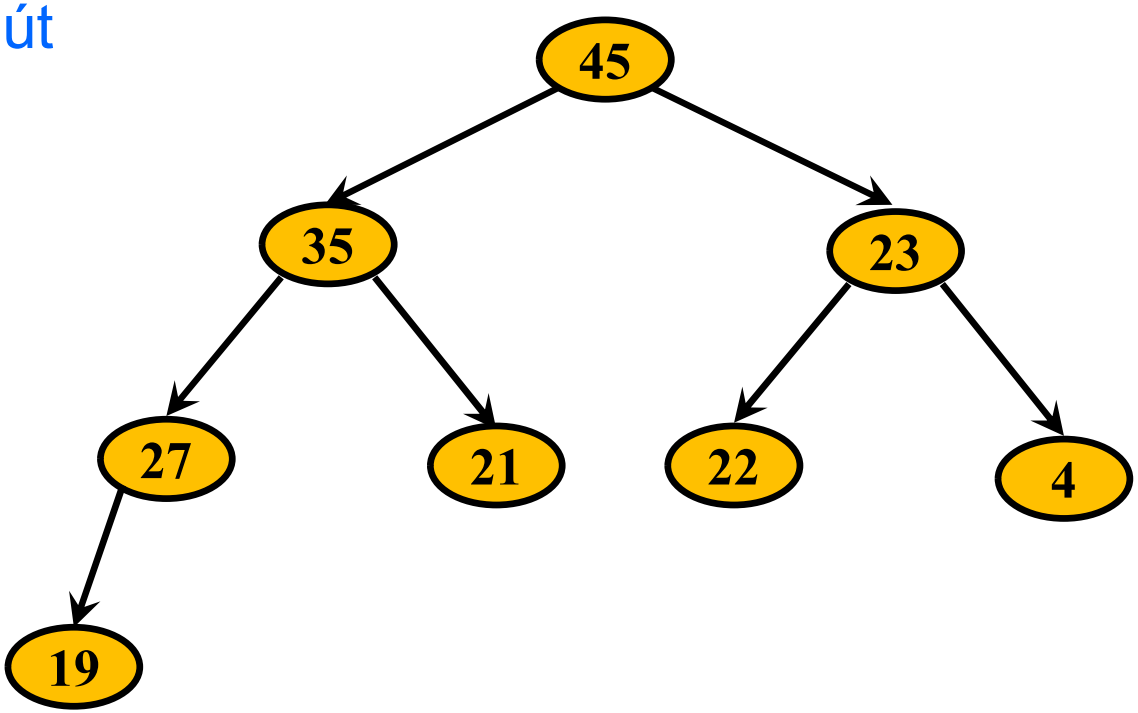
Định nghĩa heap

- Heap là một cây nhị phân thỏa các tính chất sau:
  - + Heap là một cây nhị phân gần đầy đủ;
  - + Giá trị của mỗi nút không bao giờ bé hơn giá trị của các nút con.



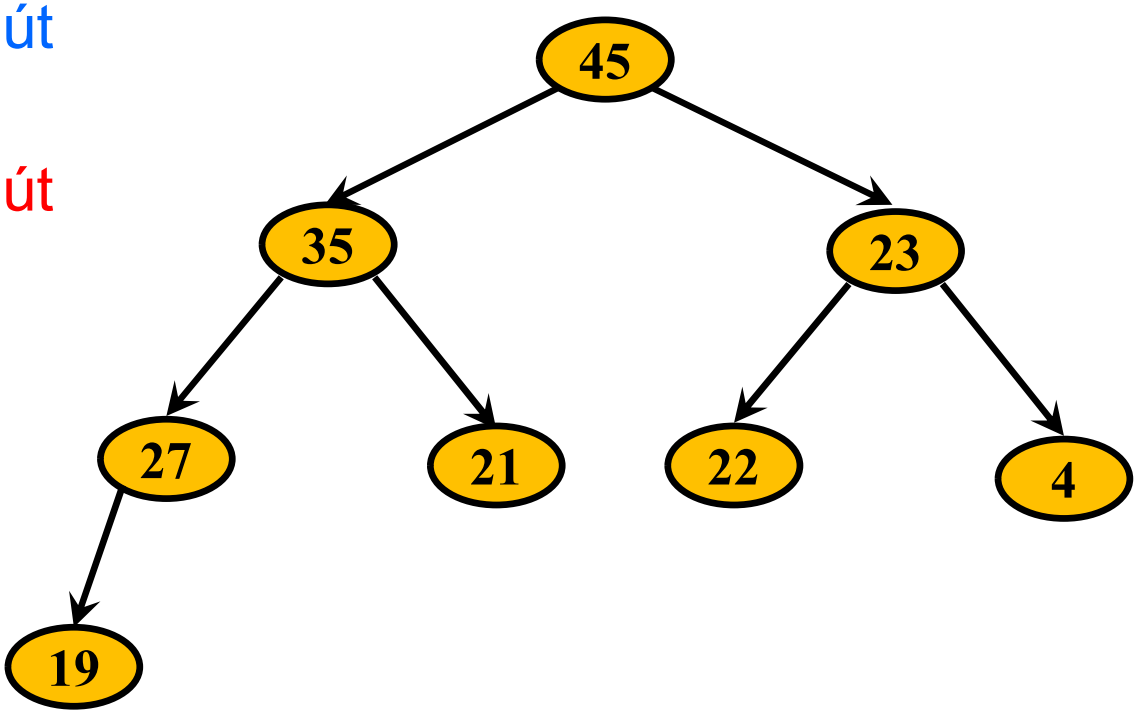
Định nghĩa heap

- Câu hỏi: Nút lớn nhất là nút nào?



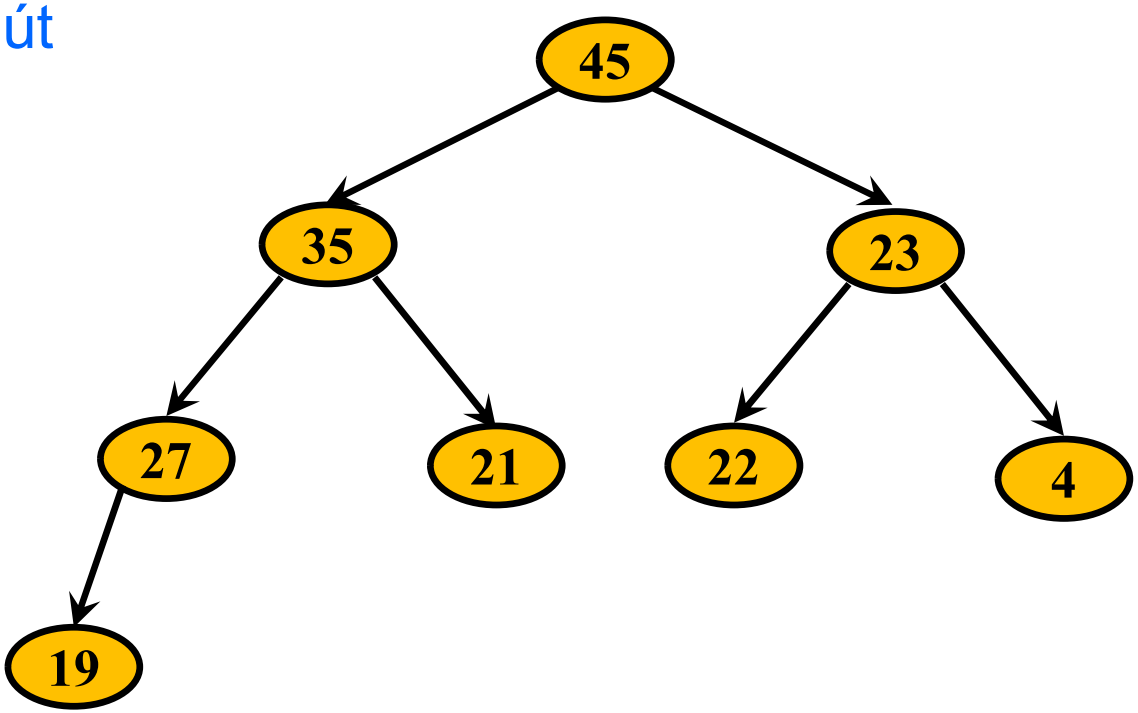
Định nghĩa heap

- Câu hỏi: Nút lớn nhất là nút nào?
- Trả lời: Nút lớn nhất là nút gốc.



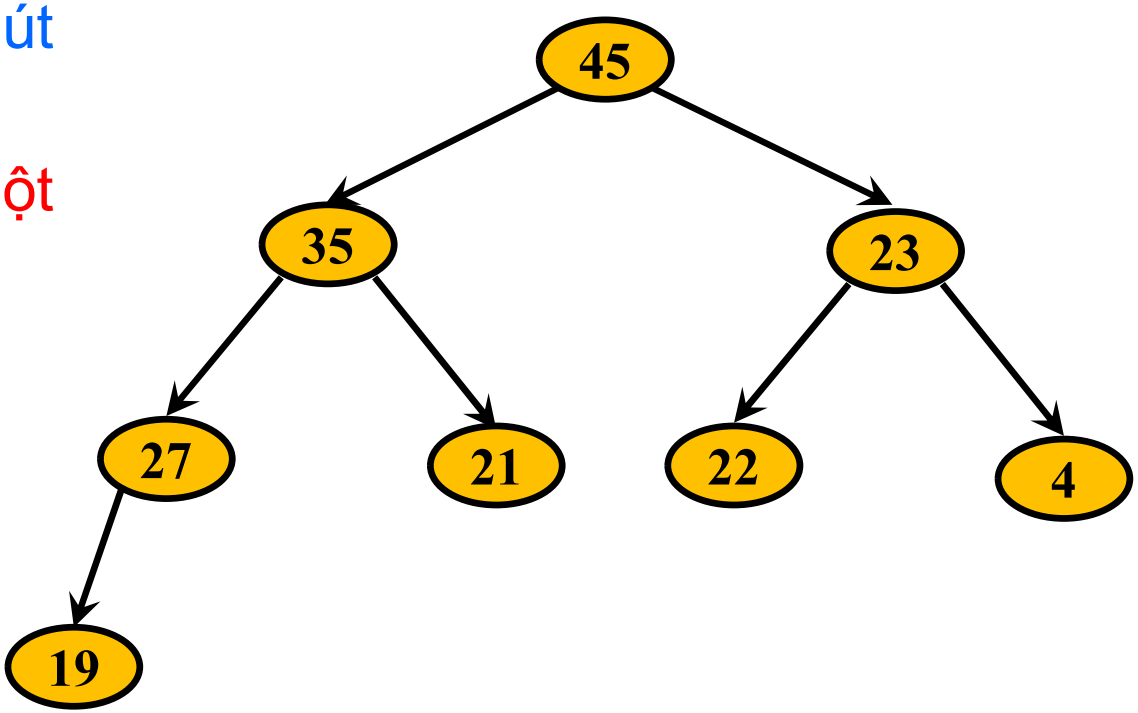
Định nghĩa heap

- Câu hỏi: Nút nhỏ nhất là nút nào?



Định nghĩa heap

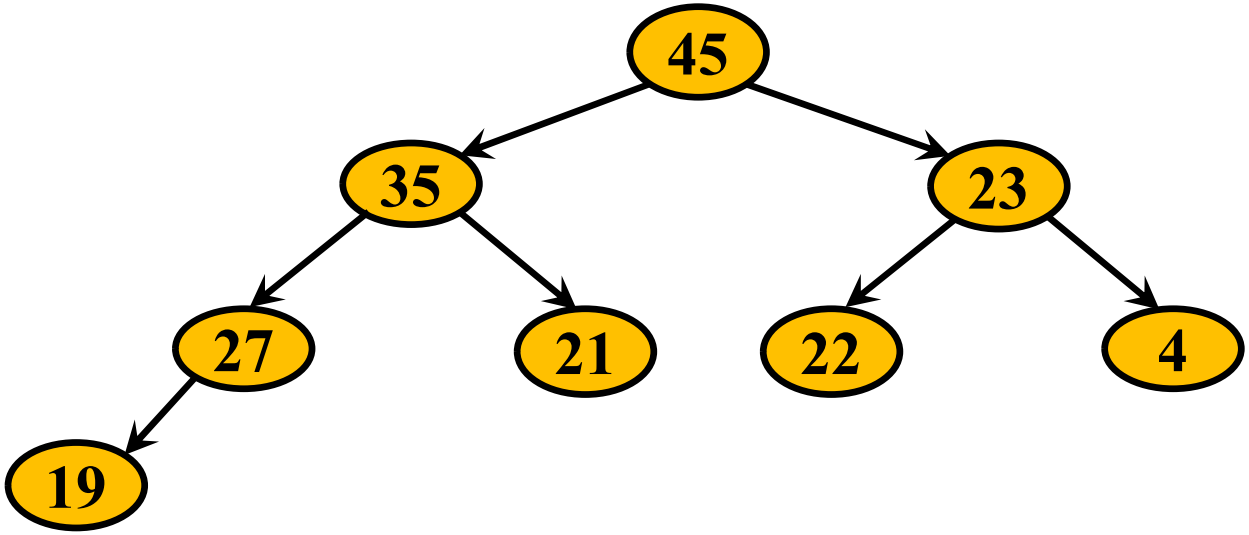
- Câu hỏi: Nút nhỏ nhất là nút nào?
- Trả lời: Nút nhỏ nhất là một trong các nút lá.

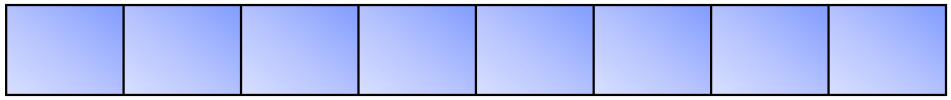


BIỂU DIỄN HEAP BẰNG MẢNG

Biểu diễn heap bằng mảng

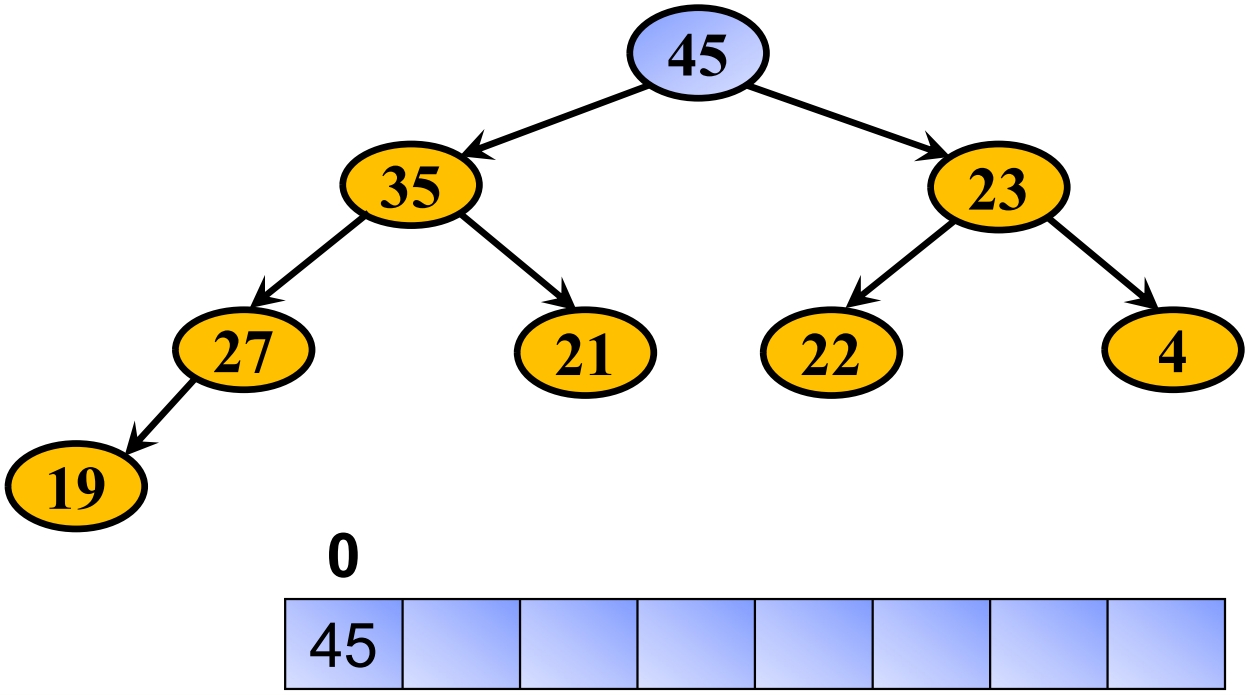
– Ta sẽ lưu giá trị của các nút trong một mảng một chiều





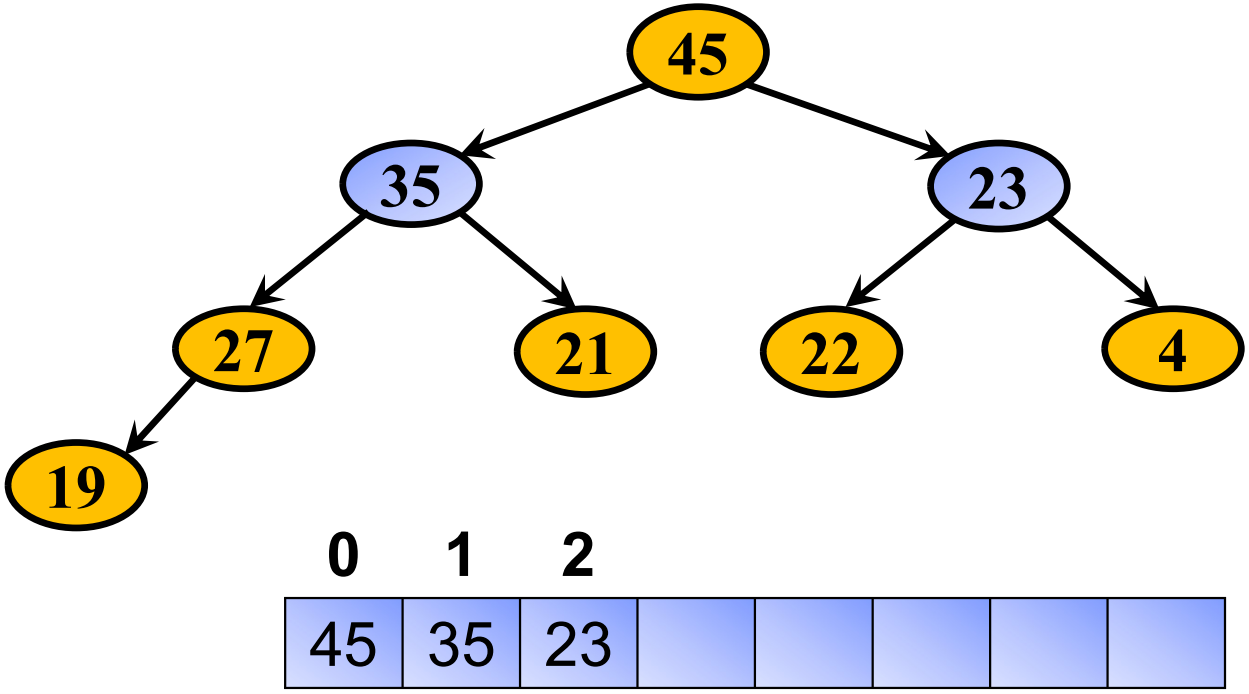
Biểu diễn heap bằng mảng

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



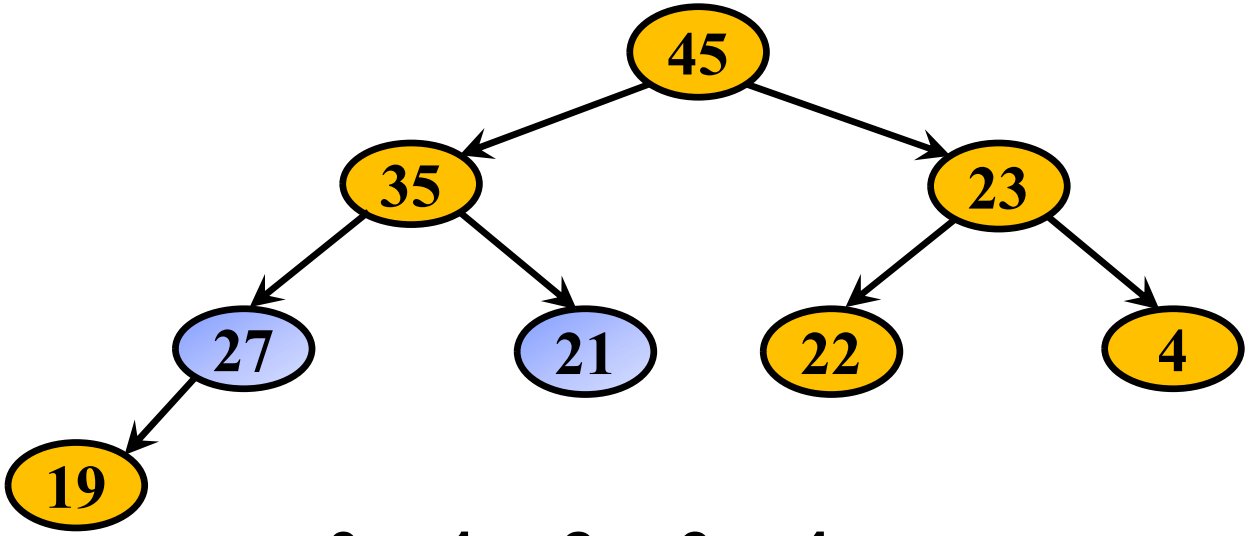
Biểu diễn heap bằng mảng

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



Biểu diễn heap bằng mảng

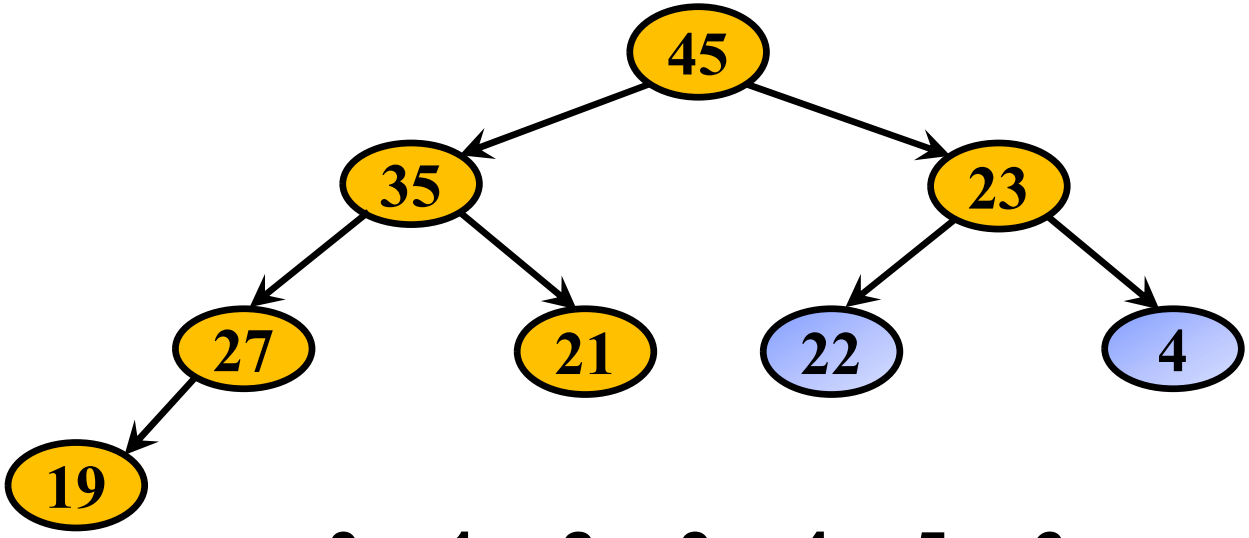
– Ta sẽ lưu giá trị của các nút trong một mảng một chiều



0	1	2	3	4			
45	35	23	27	21			

Biểu diễn heap bằng mảng

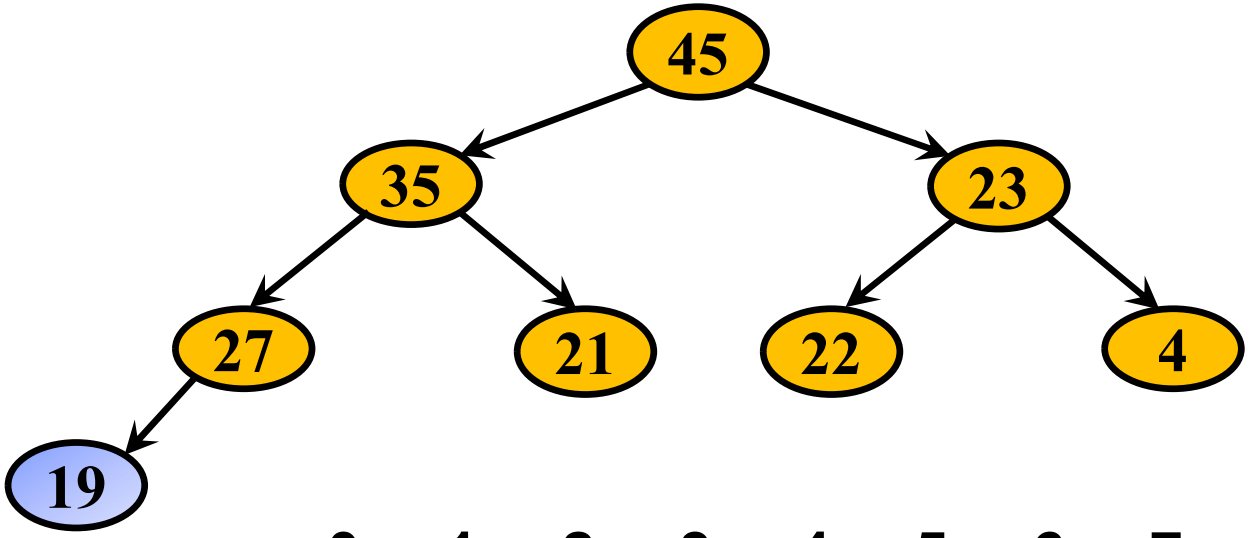
– Ta sẽ lưu giá trị của các nút trong một mảng một chiều



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

Biểu diễn heap bằng mảng

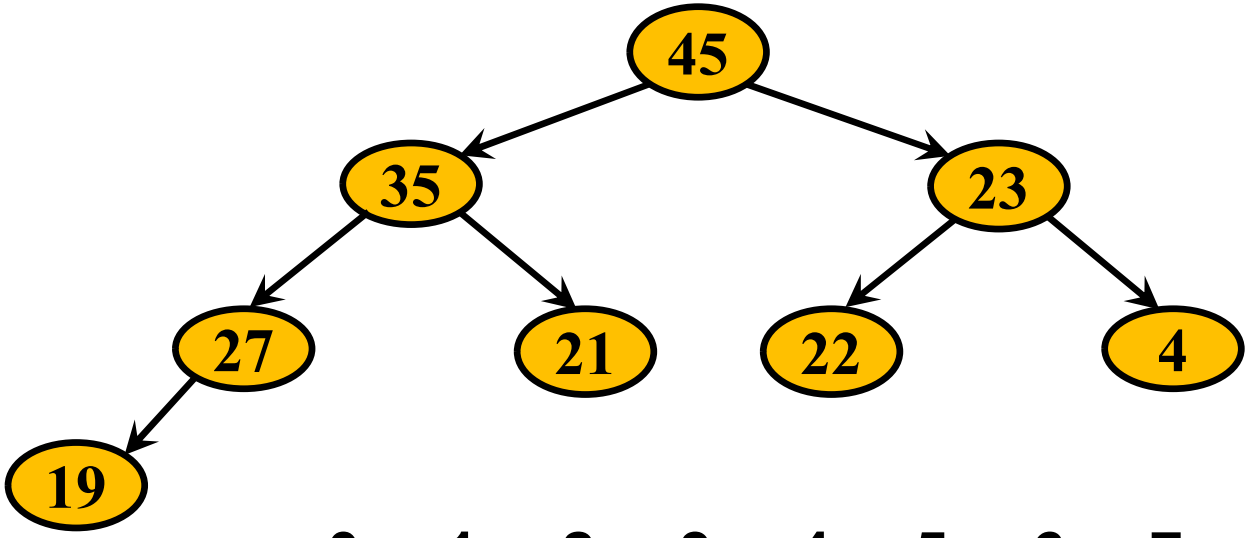
– Ta sẽ lưu giá trị của các nút trong một mảng một chiều



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

Biểu diễn heap bằng mảng

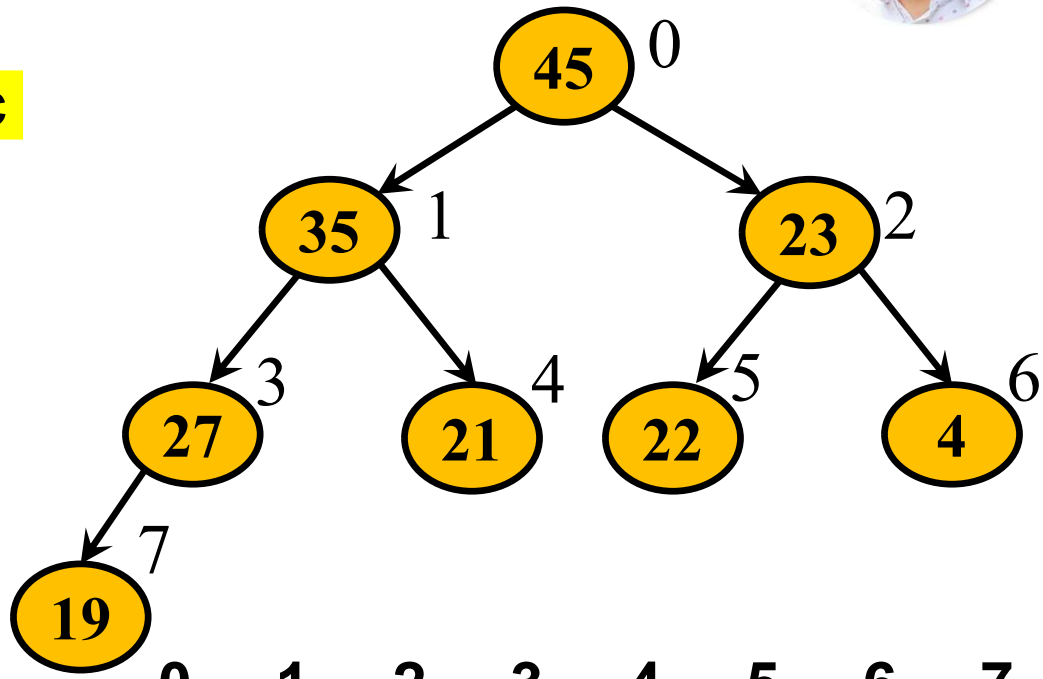
– Ta sẽ lưu giá trị của các nút trong một mảng một chiều



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

Biểu diễn heap bằng mảng

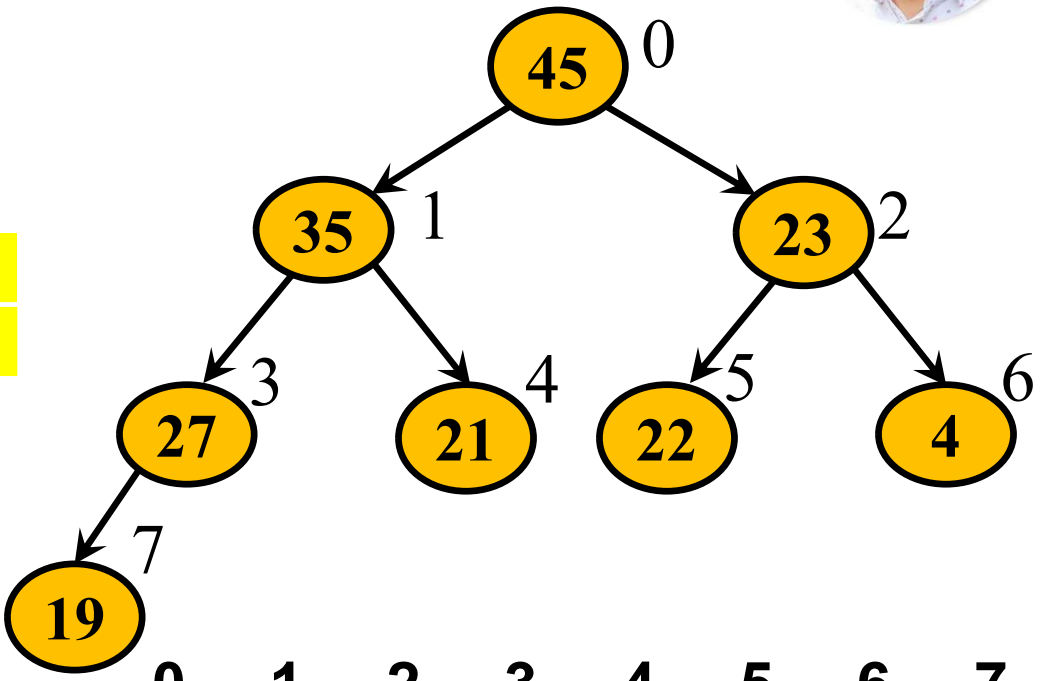
– Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải.



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

Biểu diễn heap bằng mảng

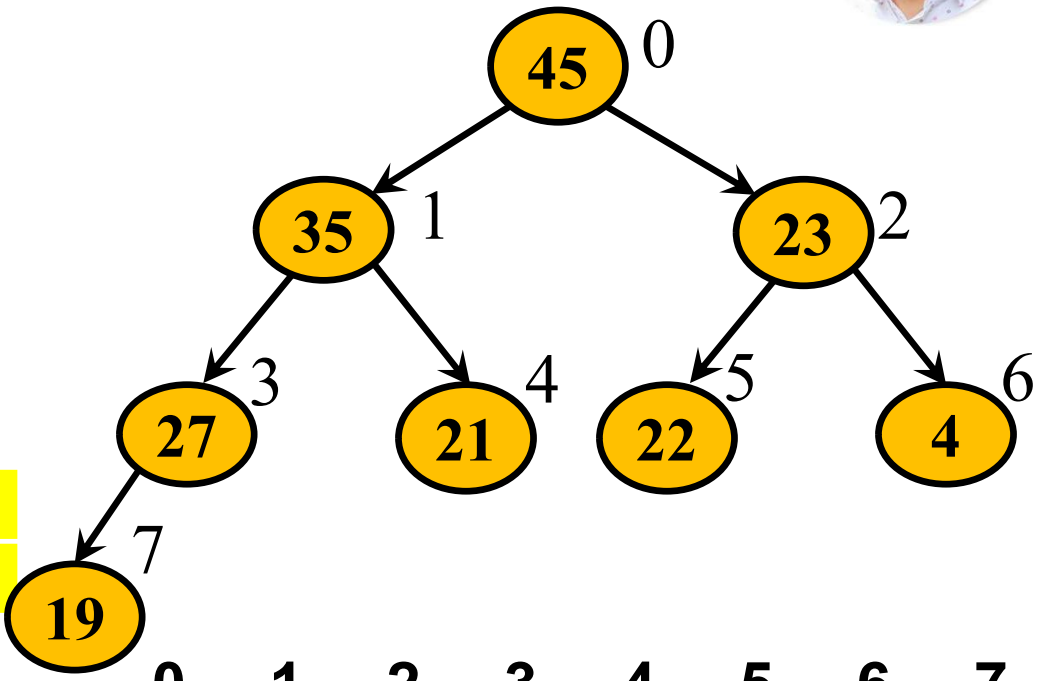
- Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải.
- Liên kết giữa các nút được hiểu ngầm, không trực tiếp dùng con trỏ.



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

Biểu diễn heap bằng mảng

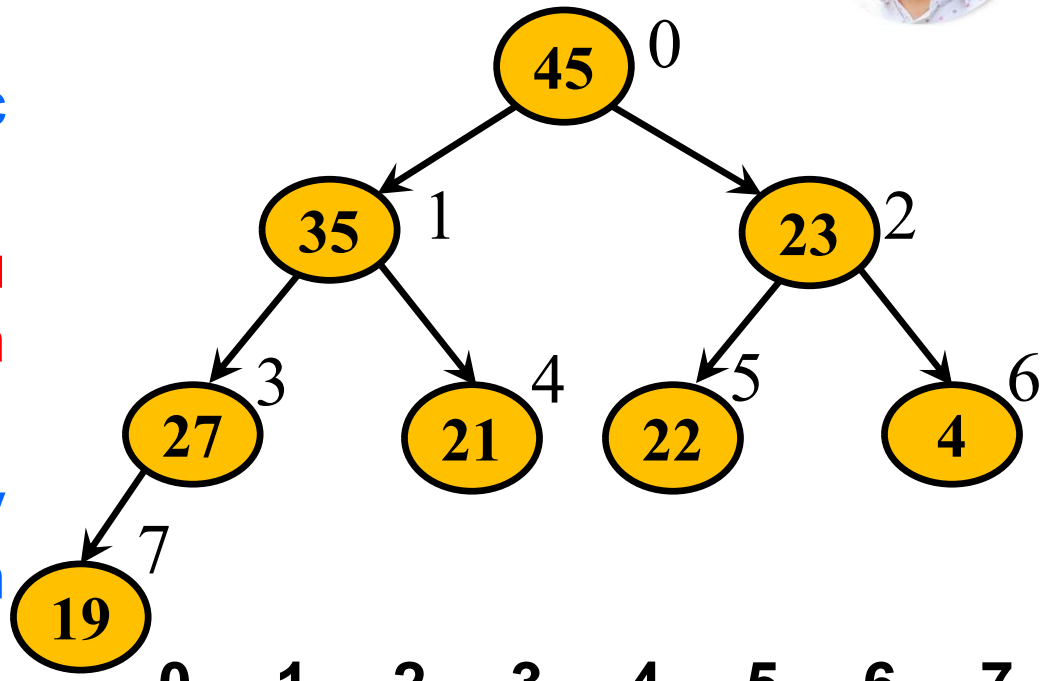
- Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải.
- Liên kết giữa các nút được hiểu ngầm, không trực tiếp dùng con trỏ.
- Mảng một chiều được xem là cây chỉ do cách ta xử lý dữ liệu trên đó.



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

Biểu diễn heap bằng mảng

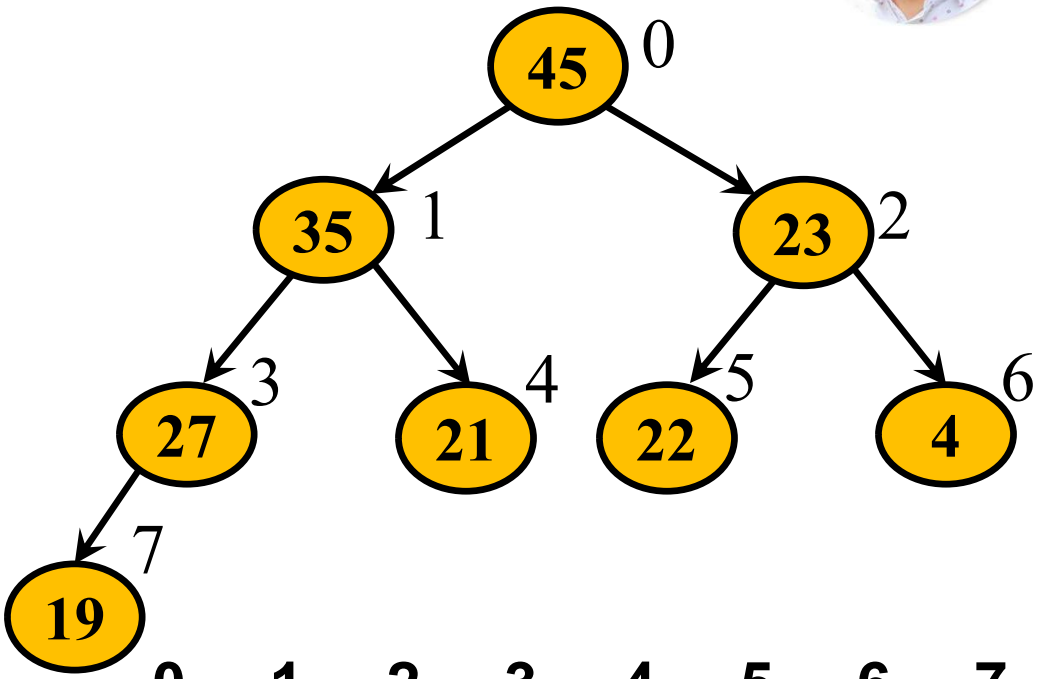
- Thứ tự lưu trữ trên mảng được thực hiện từ trái sang phải.
- Liên kết giữa các nút được hiểu ngầm, không trực tiếp dùng con trỏ.
- Mảng một chiều được xem là cây chỉ do cách ta xử lý dữ liệu trên đó.



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

Biểu diễn heap bằng mảng

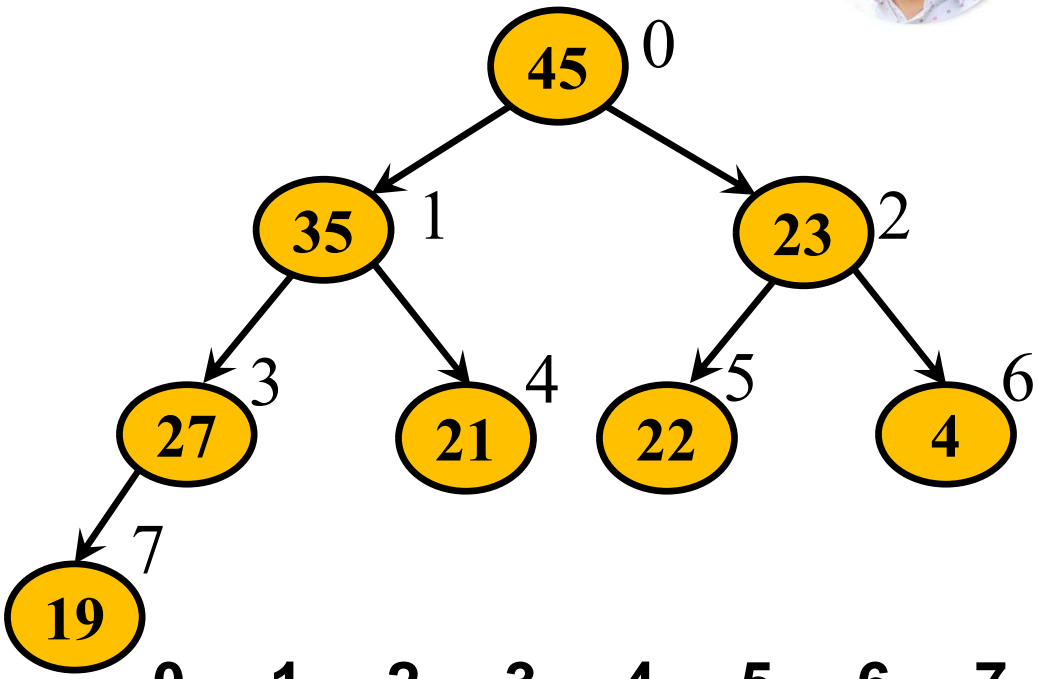
– Nút gốc có chỉ số [0].



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

Biểu diễn heap bằng mảng

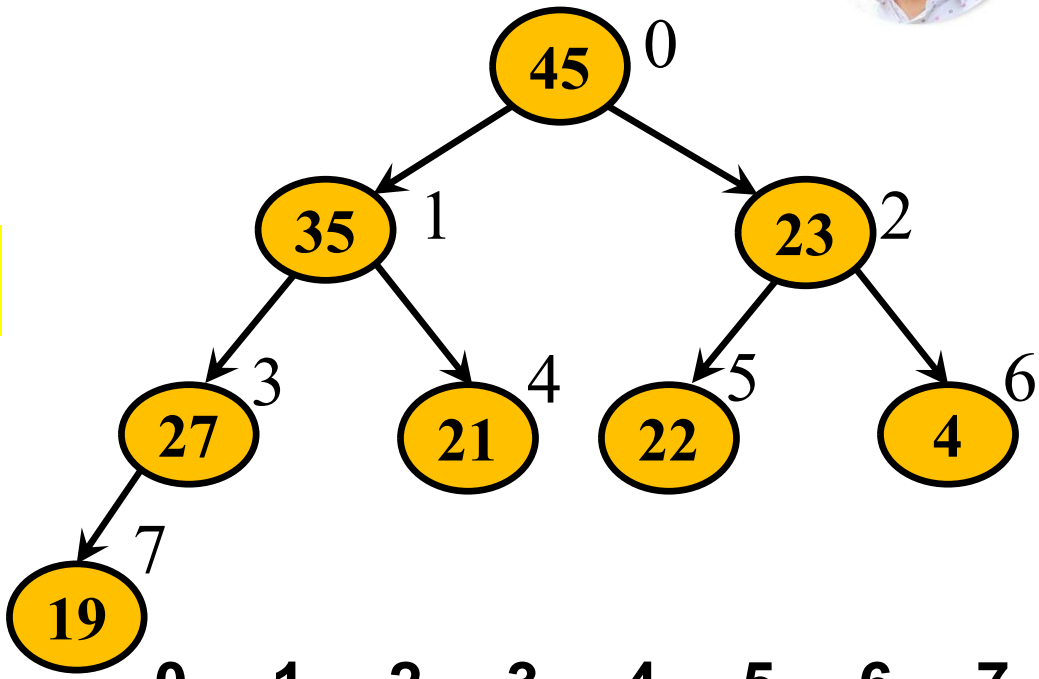
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.



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

Biểu diễn heap bằng mảng

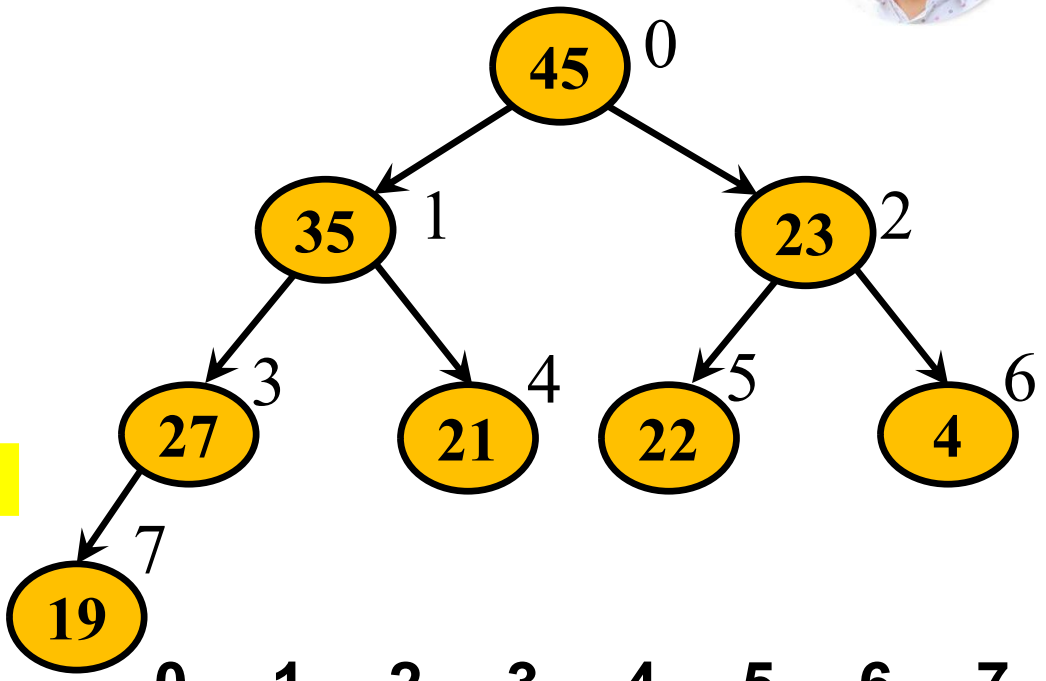
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$



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

Biểu diễn heap bằng mảng

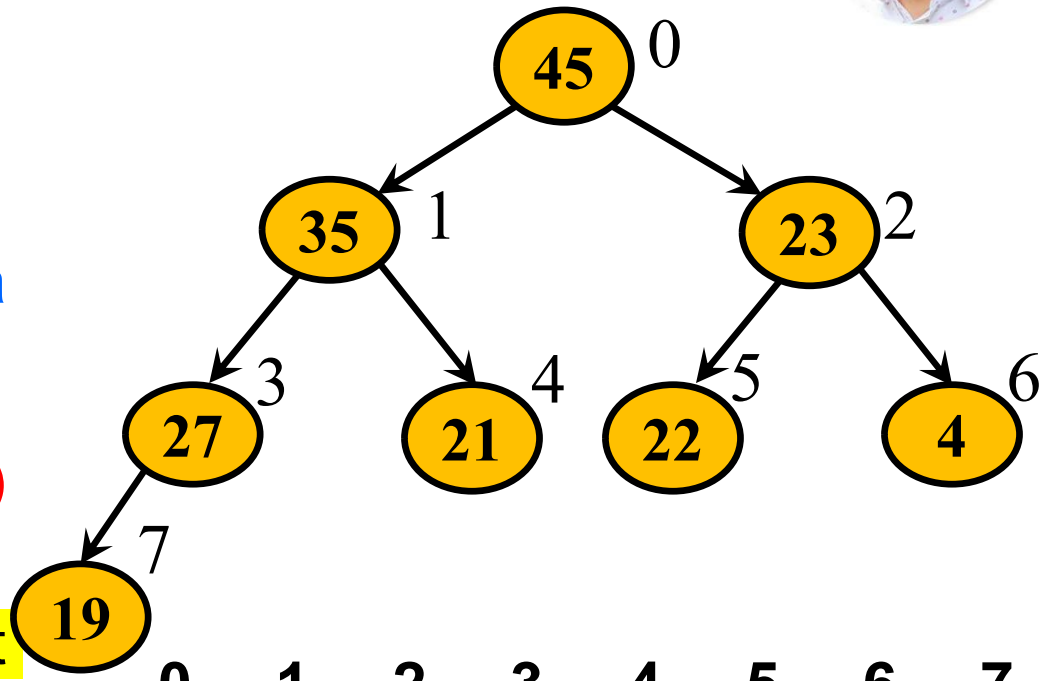
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$.
- Các nút con của nút $[i]$ (nếu có) có chỉ số $[2i+1]$ và $[2i+2]$.



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

Biểu diễn heap bằng mảng

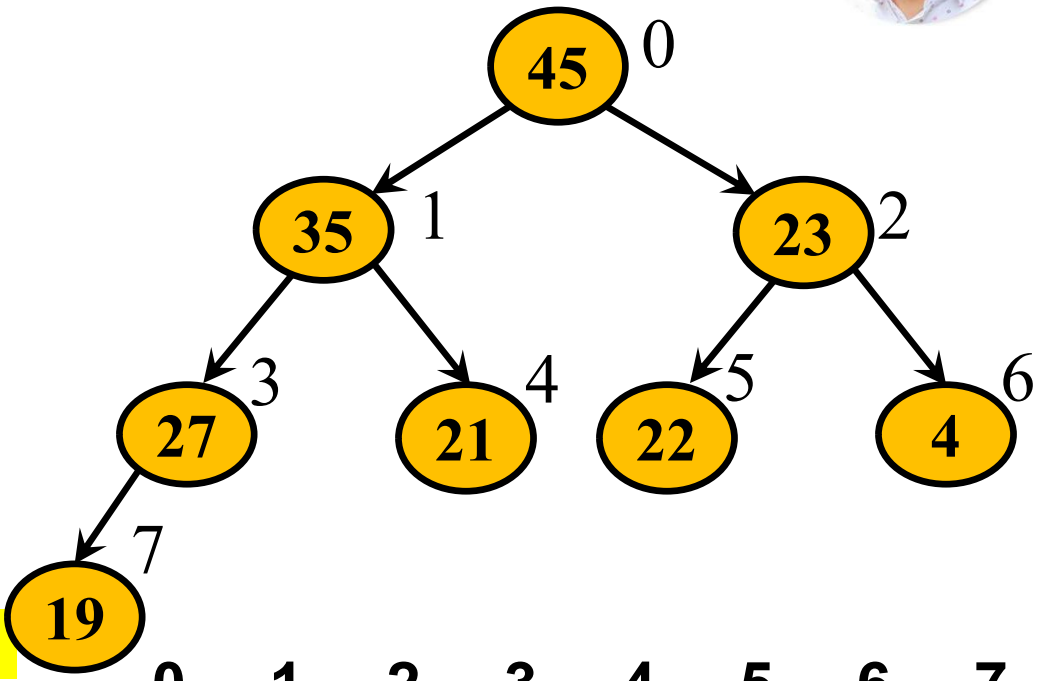
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$
- Các nút con của nút $[i]$ (nếu có) có chỉ số $[2i+1]$ và $[2i+2]$.
- Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{(n-1)-1}{2}\right]$.



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

Biểu diễn heap bằng mảng

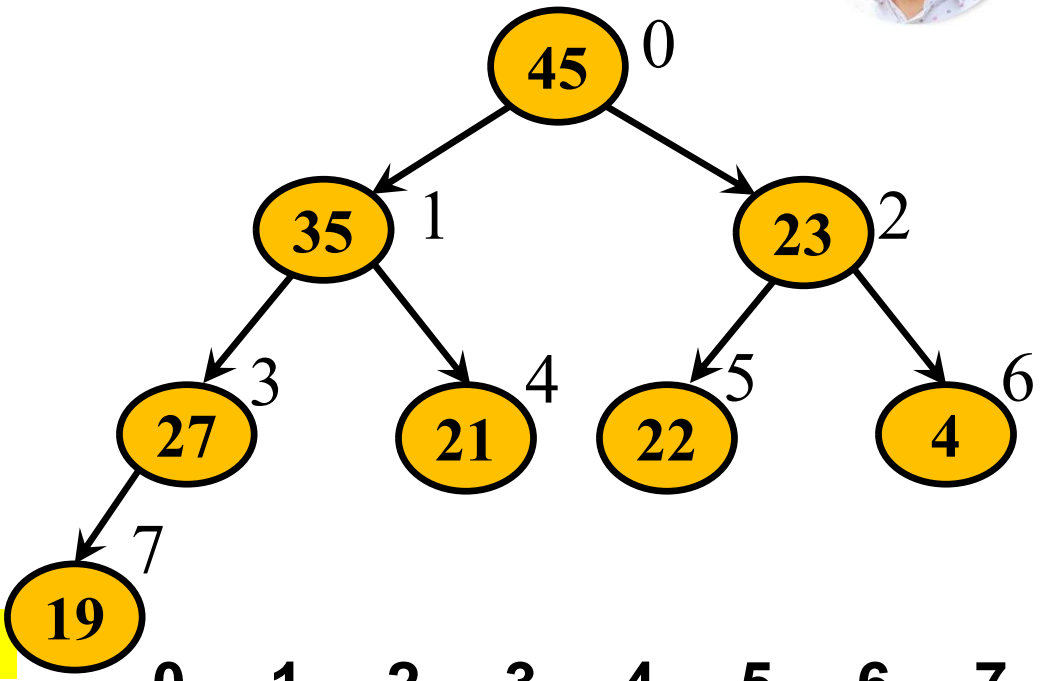
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$
- Các nút con của nút $[i]$ (nếu có) có chỉ số $[2i+1]$ và $[2i+2]$.
- Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{n-2}{2}\right]$.



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

Biểu diễn heap bằng mảng

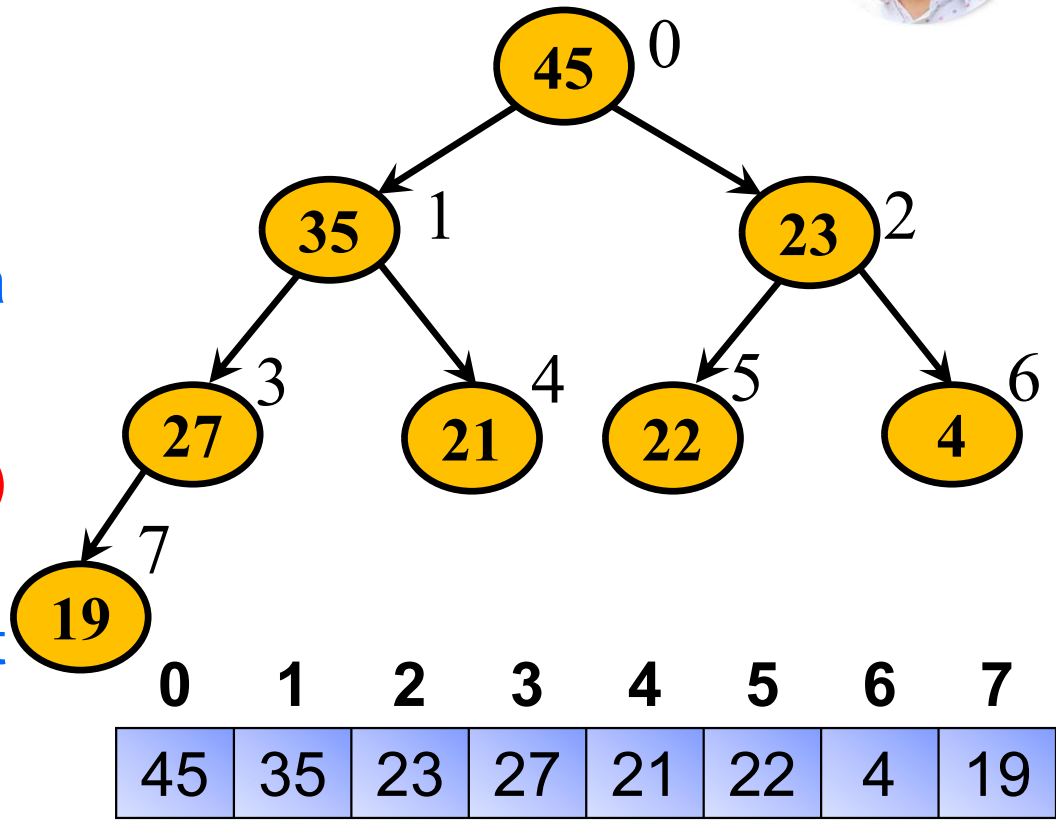
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$
- Các nút con của nút $[i]$ (nếu có) có chỉ số $[2i+1]$ và $[2i+2]$.
- Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{n}{2}-1\right]$.



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

Biểu diễn heap bằng mảng

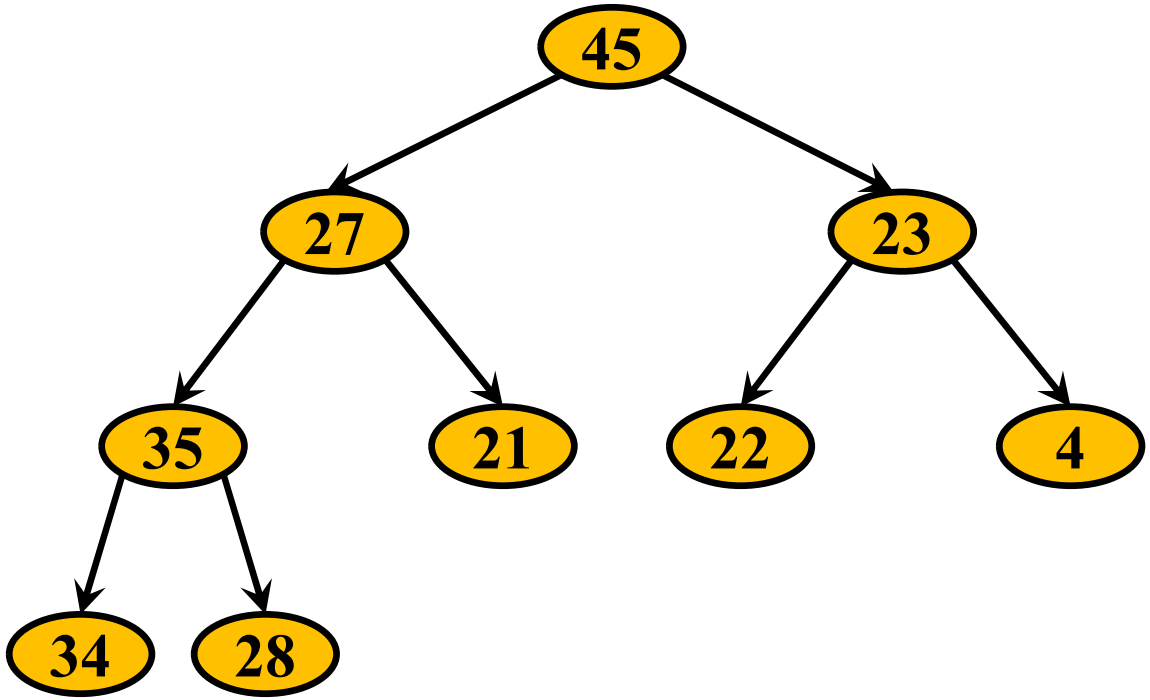
- Nút gốc có chỉ số $[0]$.
- Nút cuối cùng có chỉ số $[n-1]$.
- Nút cha của nút $[i]$ có chỉ số là $\left[\frac{(i-1)}{2}\right]$
- Các nút con của nút $[i]$ (nếu có) có chỉ số $[2i+1]$ và $[2i+2]$.
- Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{n}{2}-1\right]$.



Thao tác điều chỉnh một phần tử Heapify

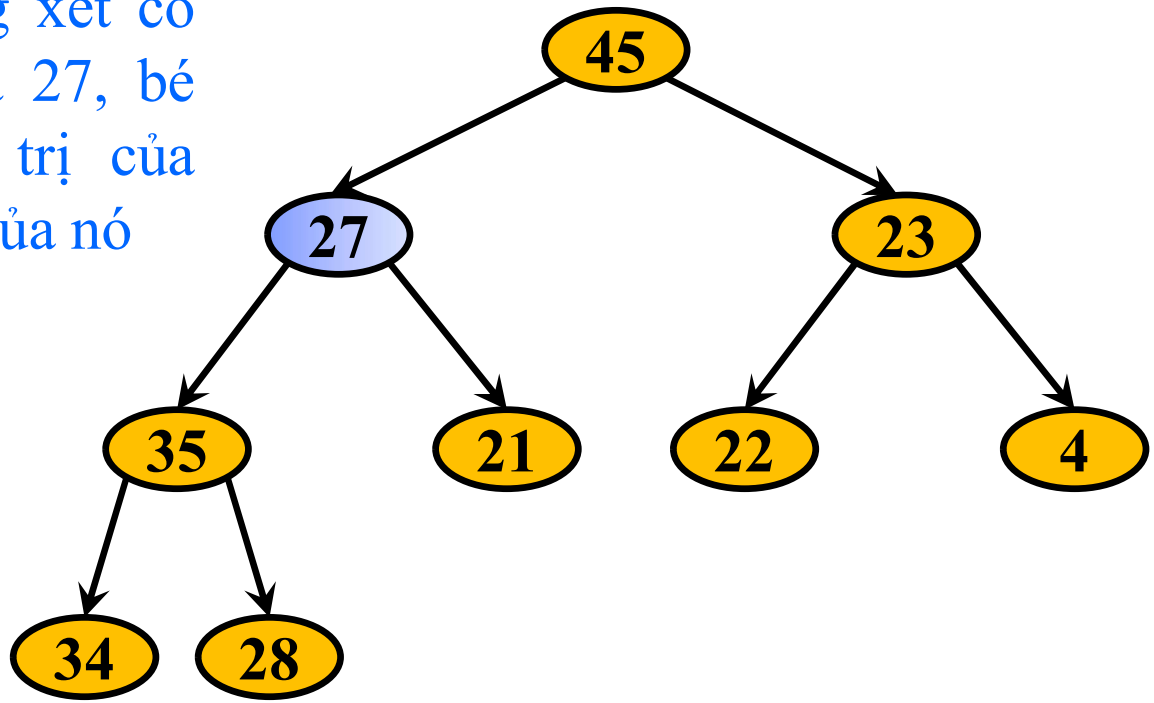
THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ HEAPIFY

Thao tác điều chỉnh một phần tử



Thao tác điều chỉnh một phần tử

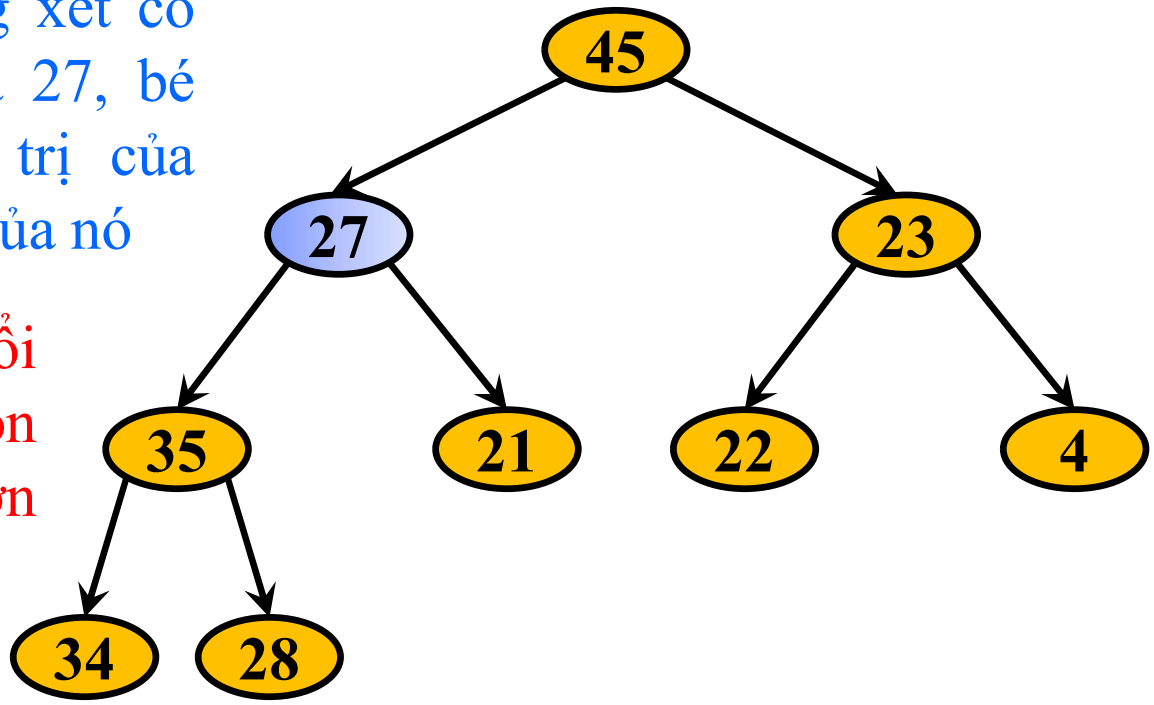
Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó



Thao tác điều chỉnh một phần tử

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

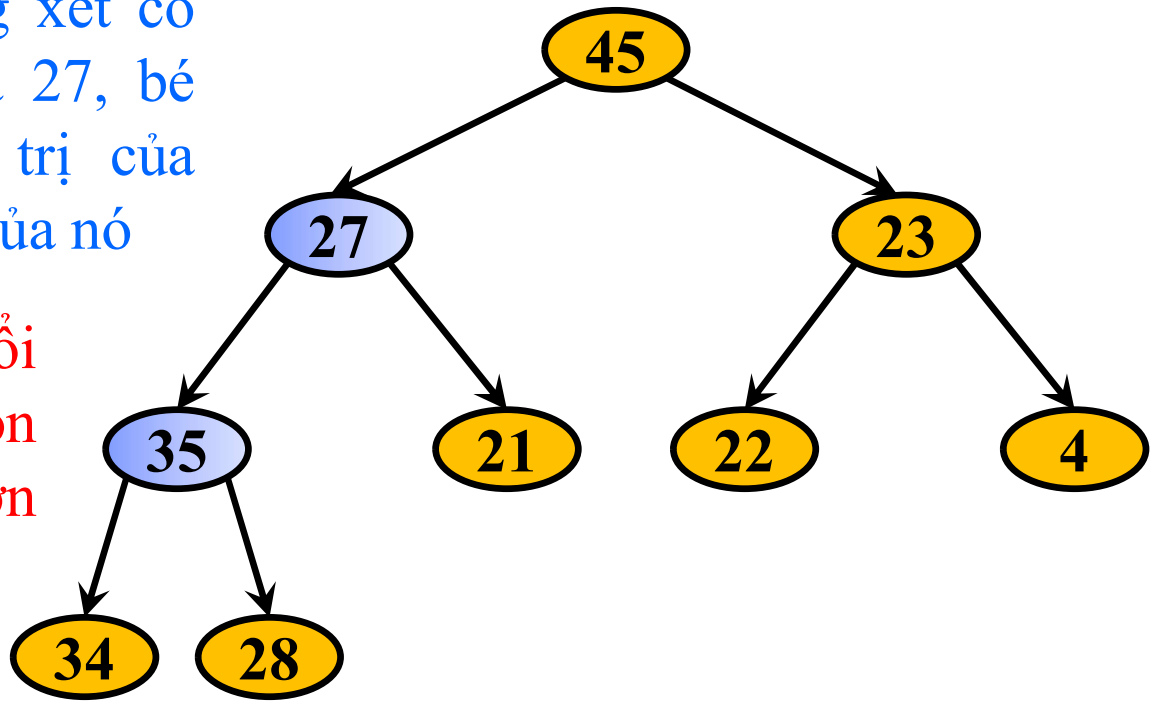
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



Thao tác điều chỉnh một phần tử

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

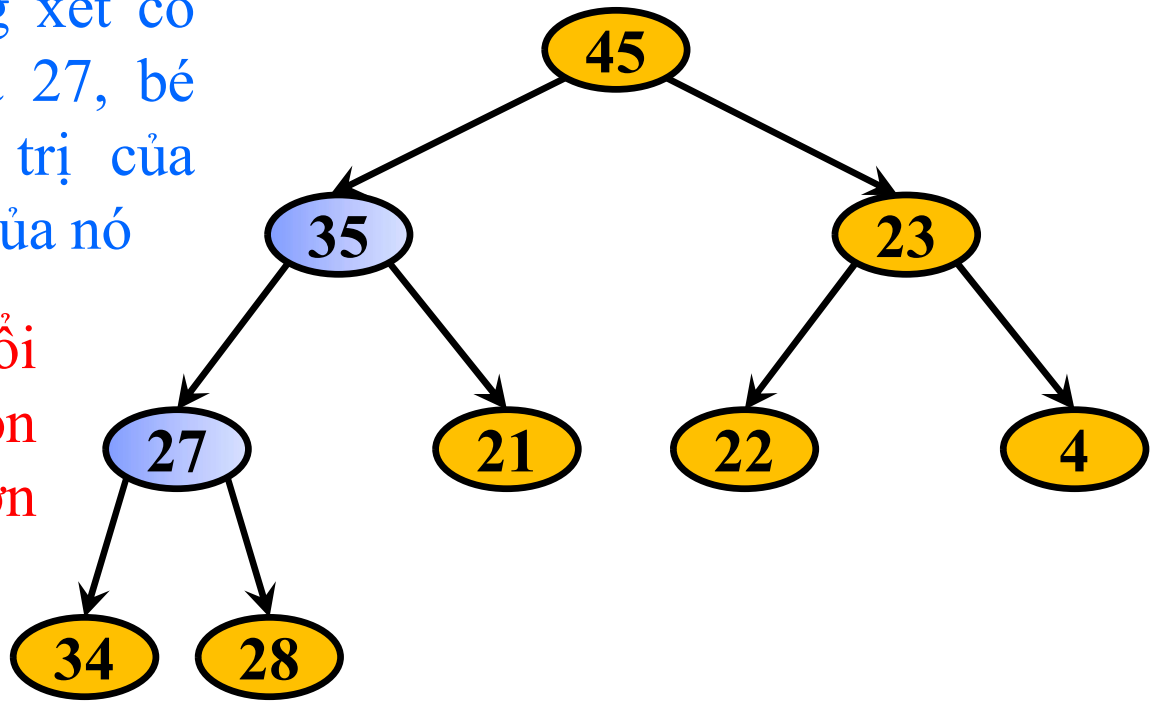
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



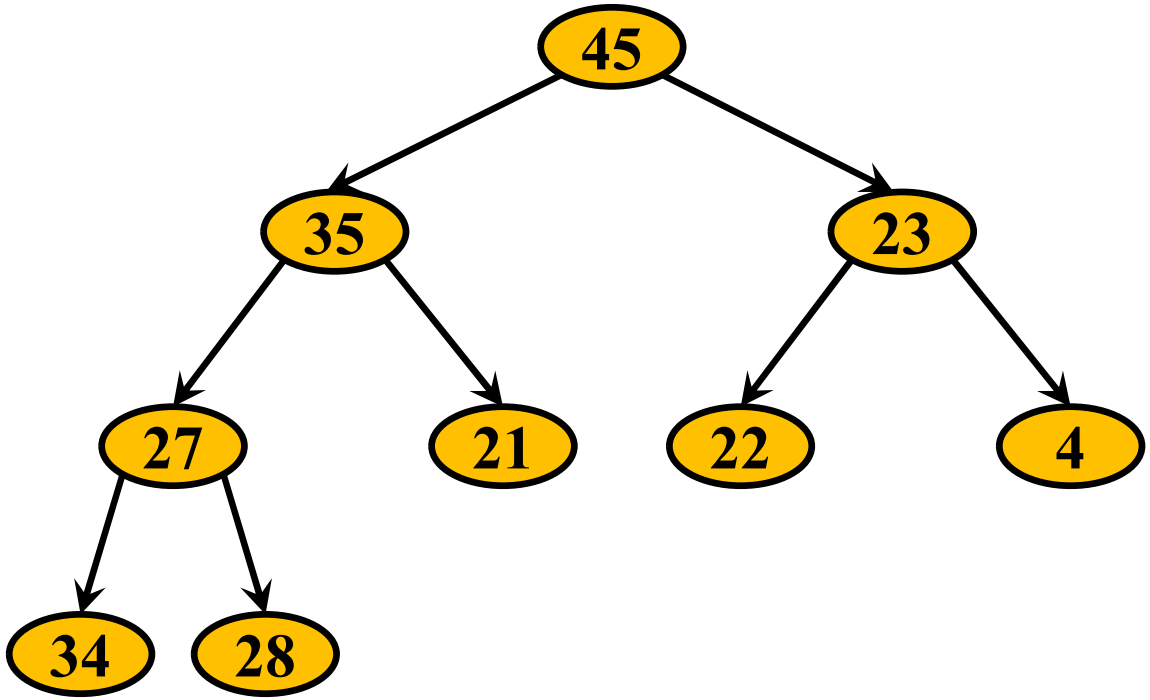
Thao tác điều chỉnh một phần tử

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

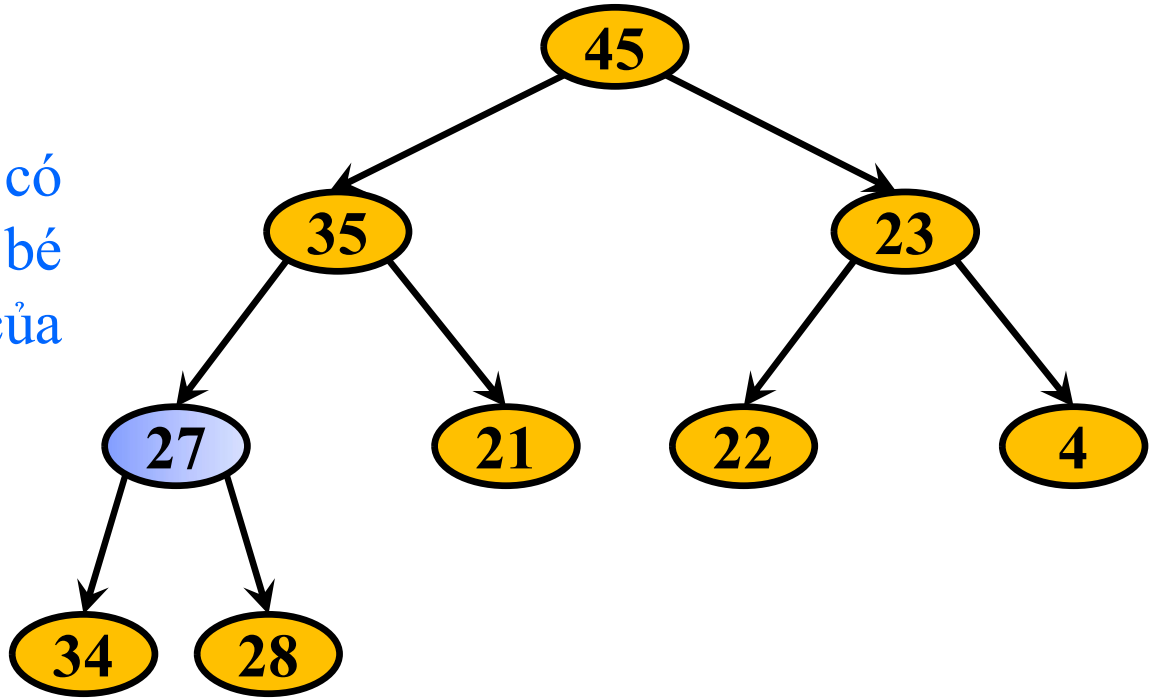


Thao tác điều chỉnh một phần tử



Thao tác điều chỉnh một phần tử

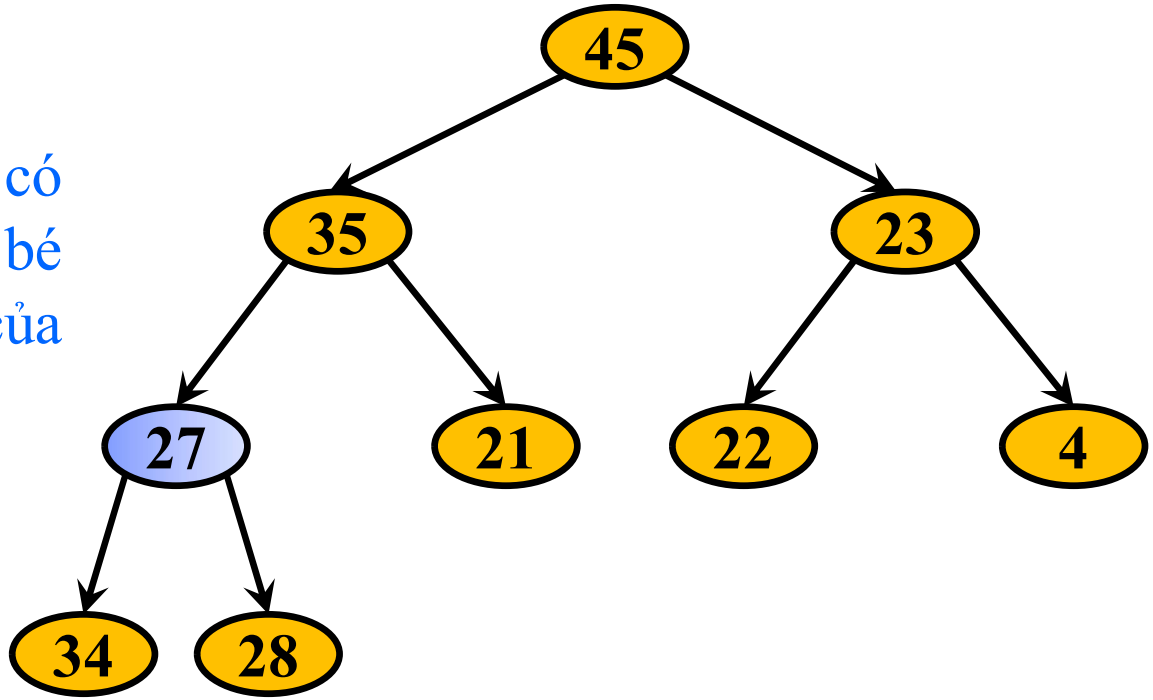
Nút đang xét có giá trị là 27, bé hơn giá trị của nút con của nó



Thao tác điều chỉnh một phần tử

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

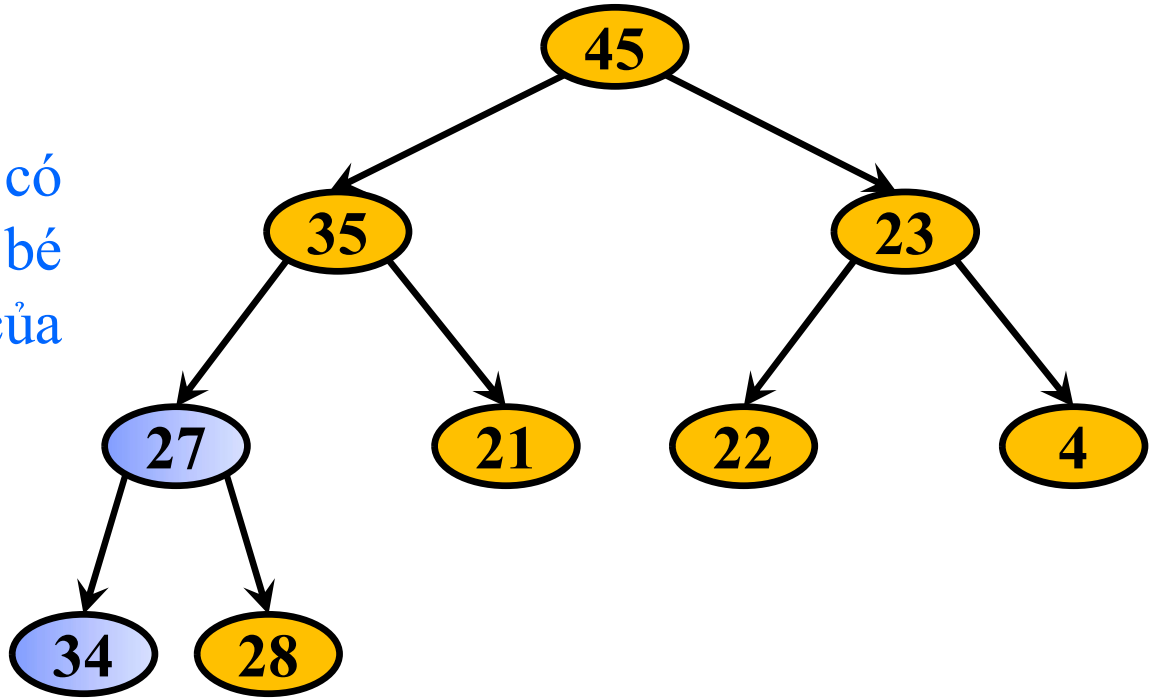
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



Thao tác điều chỉnh một phần tử

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

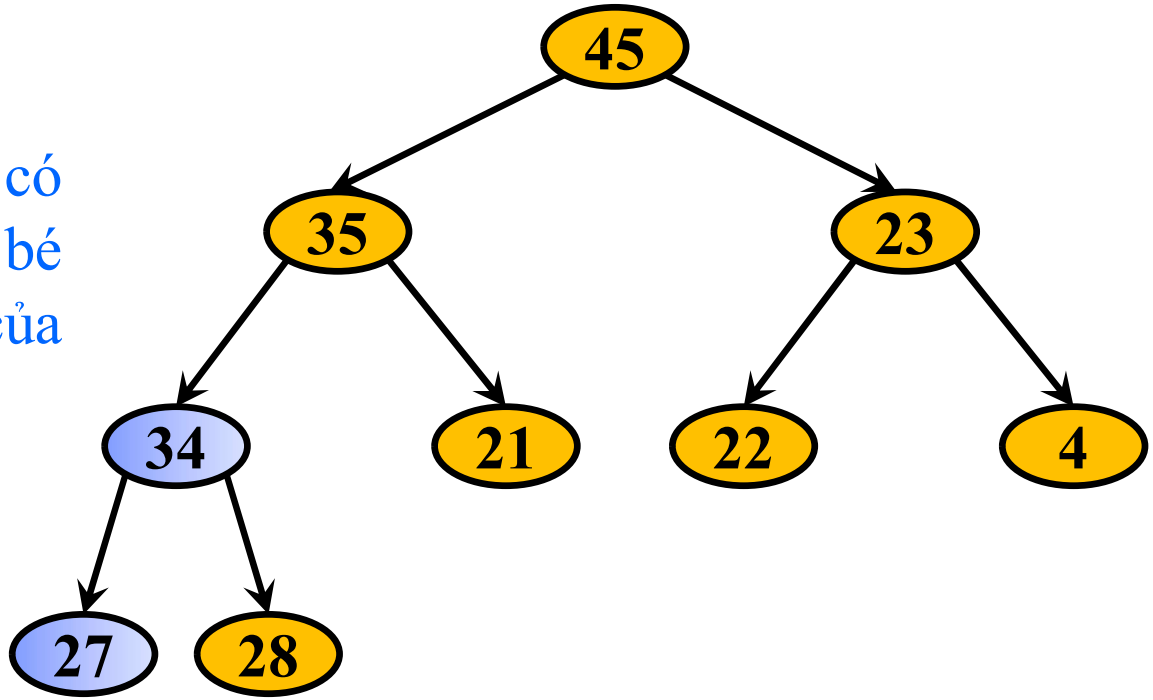
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



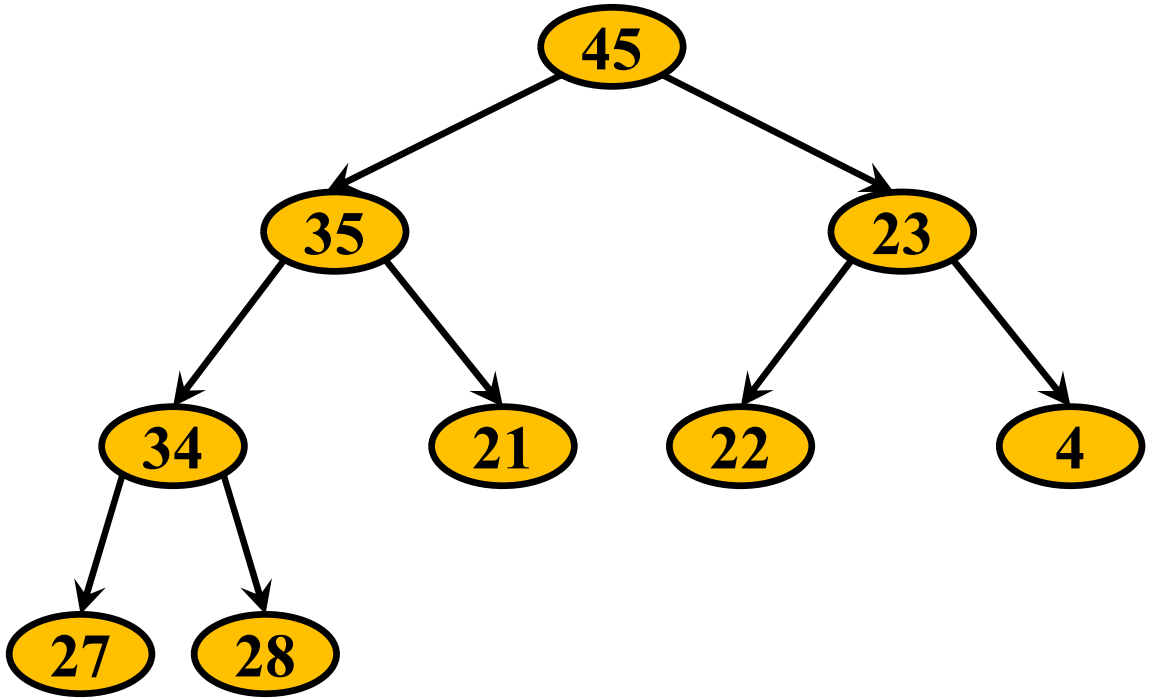
Thao tác điều chỉnh một phần tử

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

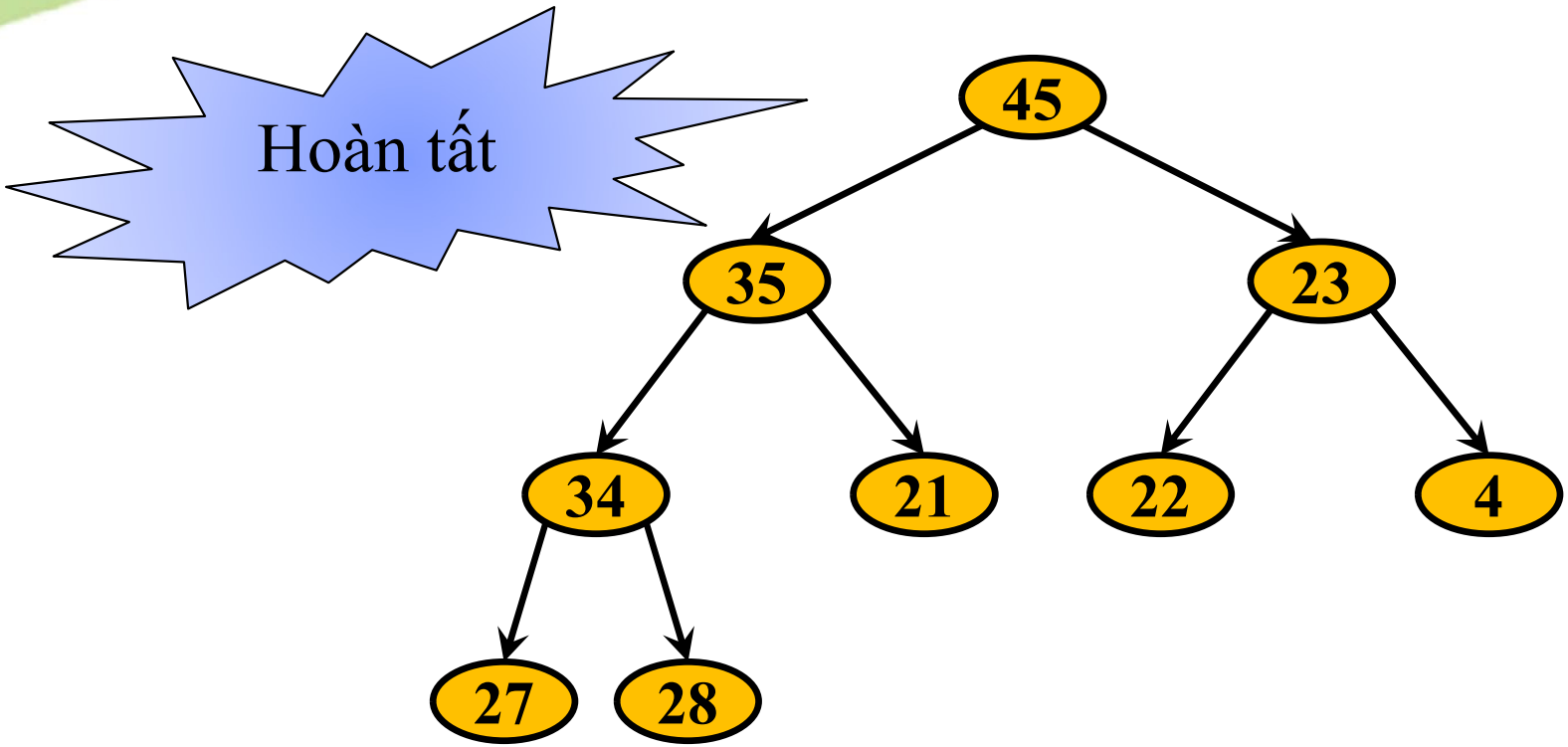
Tiến hành đổi chỗ với nút con có giá trị lớn nhất



Thao tác điều chỉnh một phần tử



Thao tác điều chỉnh một phần tử



HÀM CÀI ĐẶT THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ

Heapify

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

Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{n}{2}-1\right]$.

Các nút con của nút [vt] (nếu có) có chỉ số $[2vt+1]$ và $[2vt+2]$.

Heapify

```
11.void Heapify(int a[],int n,int vt)
12.{
13.|    while(vt<=n/2-1)
14.|    {
15.|    |    …
16.|    |    if(a[vt]<a[lc])
17.|    |        HoanVi(a[vt],a[lc]);
18.|    |    vt = lc;
19.|    }
20.}
```

CẢI TIẾN HÀM CÀI ĐẶT THAO TÁC ĐIỀU CHỈNH MỘT PHẦN TỬ

Cải tiến Heapify

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

Nút cuối cùng có con trong một heap có n phần tử là: $\left[\frac{n}{2}-1\right]$.

Các nút con của nút [vt] (nếu có) có chỉ số $[2vt+1]$ và $[2vt+2]$.

XÂY DỰNG HEAP

Xây dựng heap

- Tất cả các phần tử trên mảng (trên heap) có chỉ số $\left[\frac{n}{2}\right]$ 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 $\left[\frac{n}{2}-1\right]$ tới 0 ta sẽ tạo ra một Heap có n phần tử.

Xây dựng heap

— Hàm cài đặt

```
11.void BuildHeap(int a[],int n)
12.{
13.    for(int i=n/2-1;i>=0;i--)
14.        Heapify(a,n,i);
15.}
```

Xây dựng heap

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

CHẠY TỪNG BƯỚC XÂY DỰNG HEAP

Chạy từng bước xây dựng heap

– Bài toán: Hãy xây dựng mảng sau thành một Max Heap (yêu cầu sắp mảng tăng).

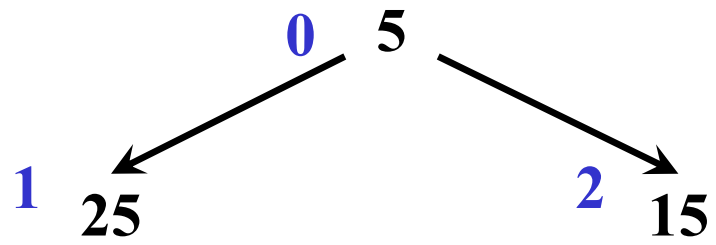
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

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

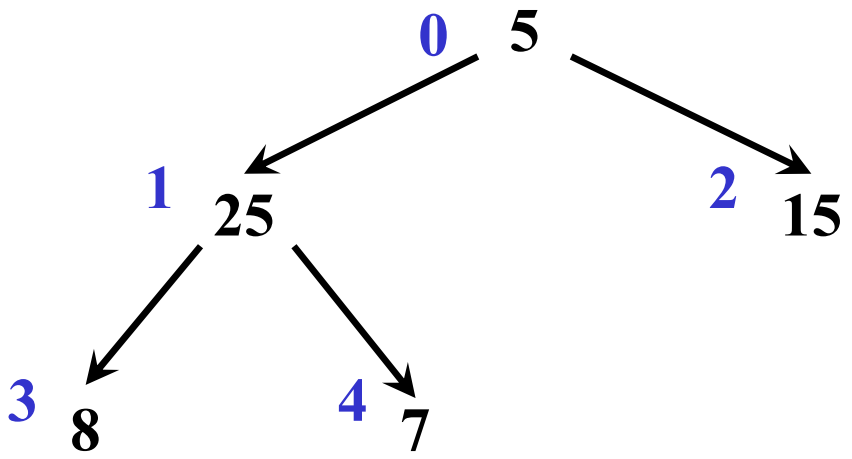
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

0 5

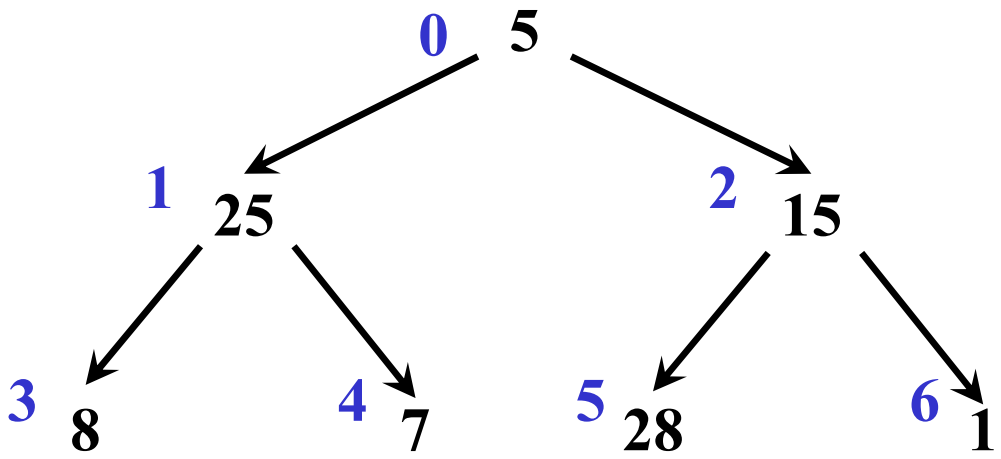
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



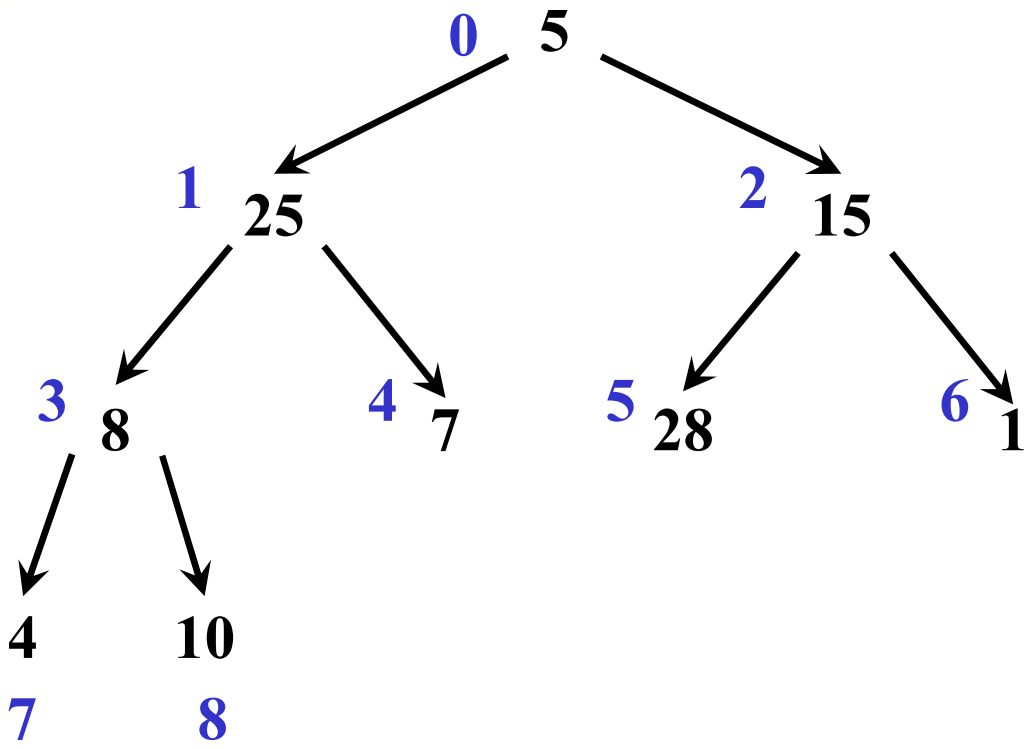
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



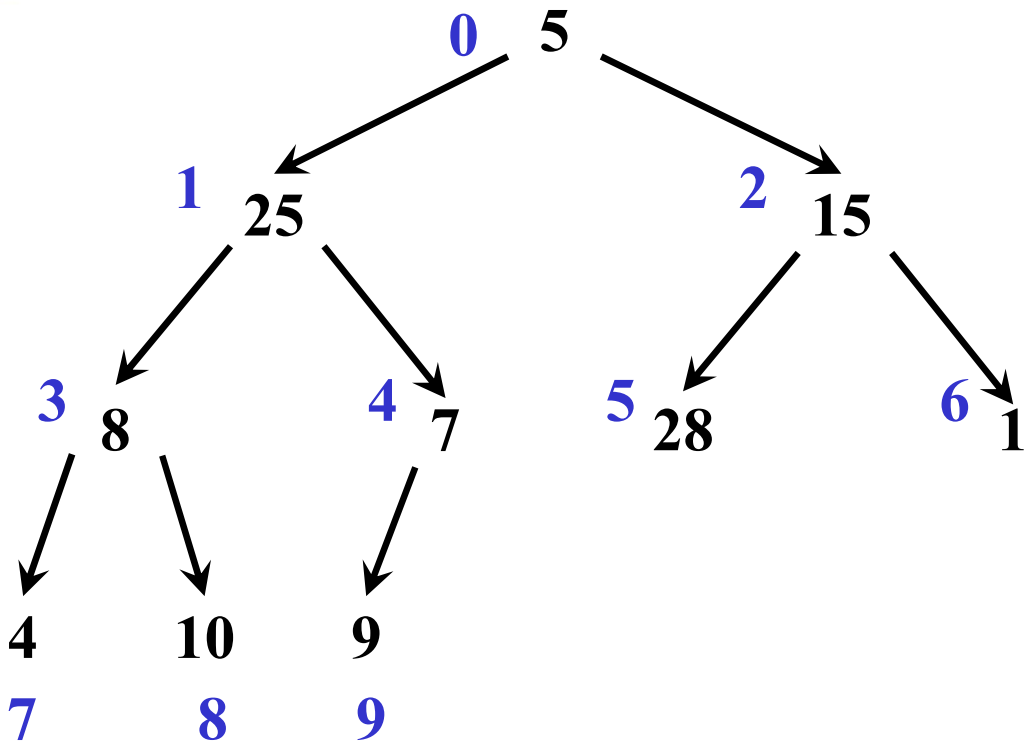
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



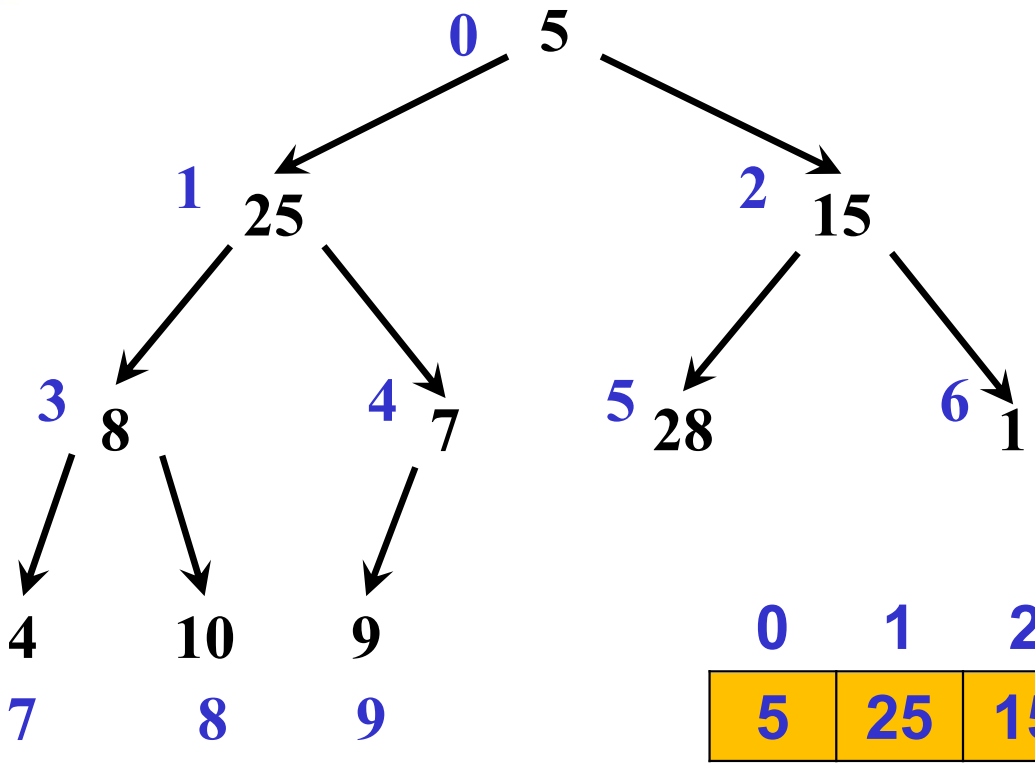
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



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

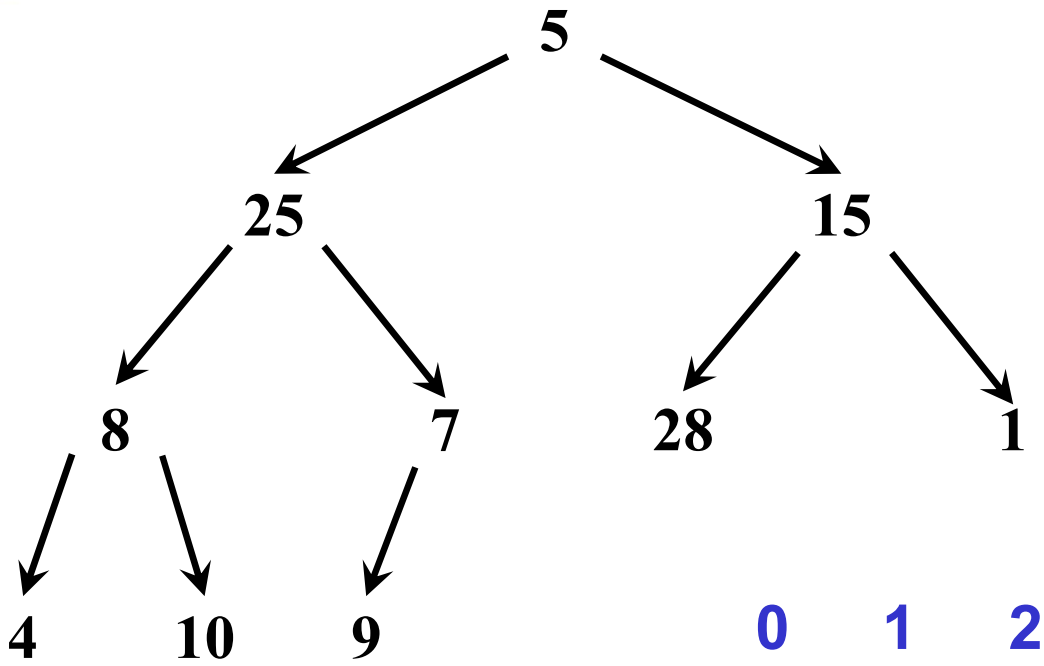


0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



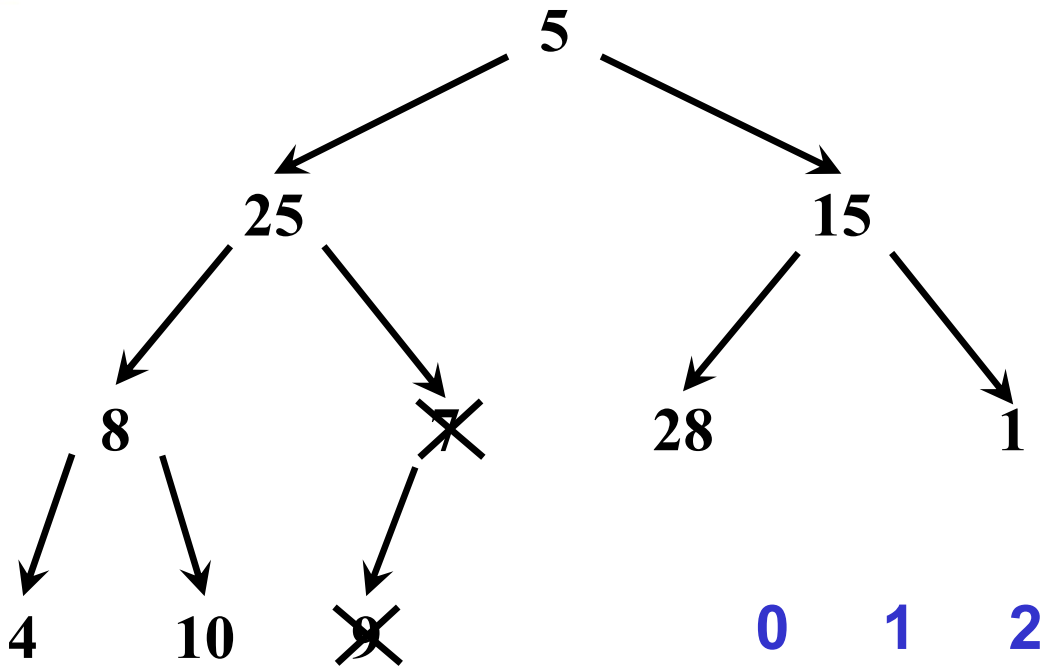
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

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



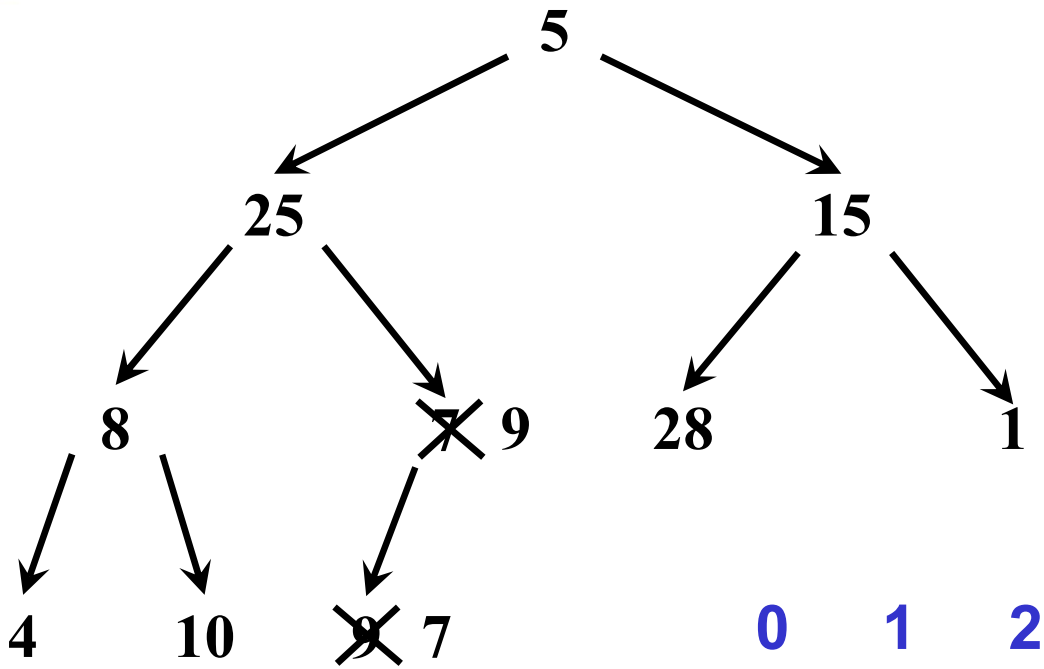
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

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



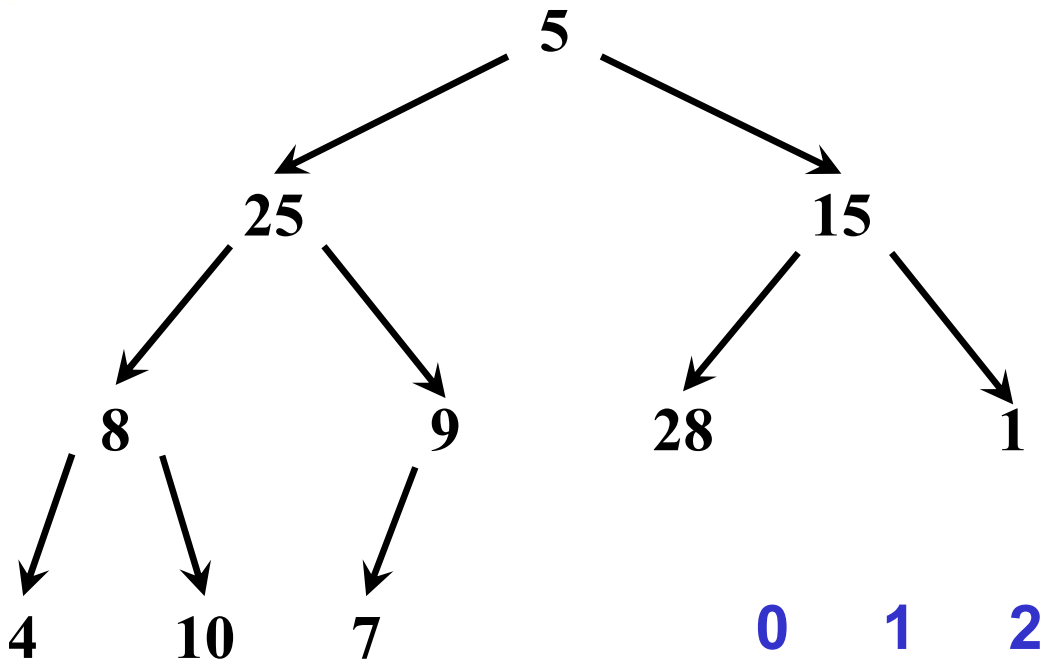
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9

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



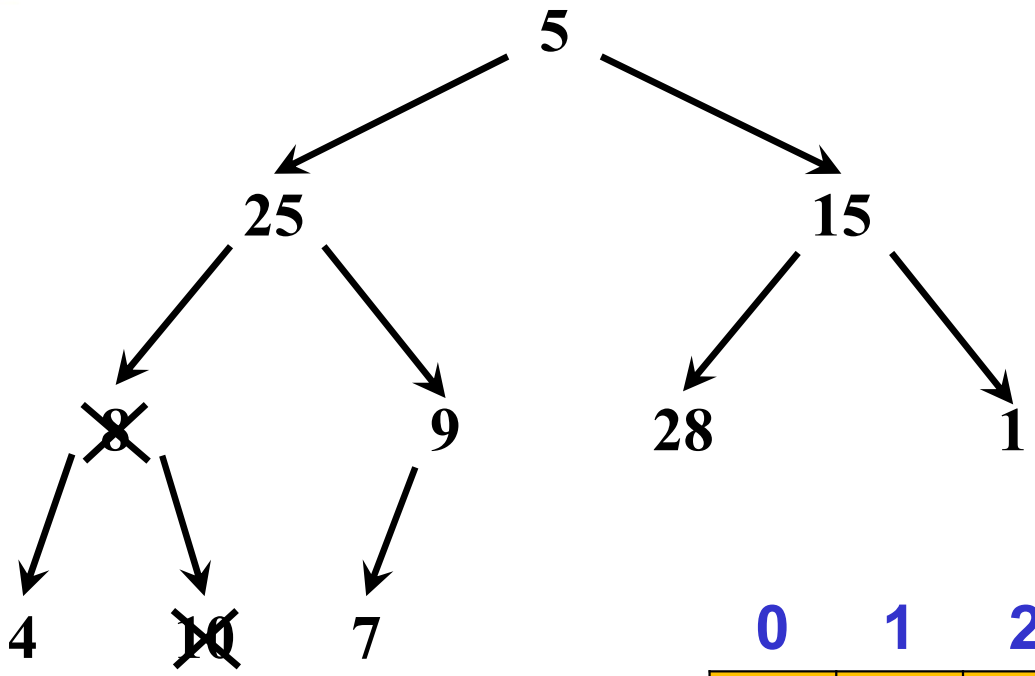
0	1	2	3	4	5	6	7	8	9
5	25	15	8	9	28	1	4	10	7

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



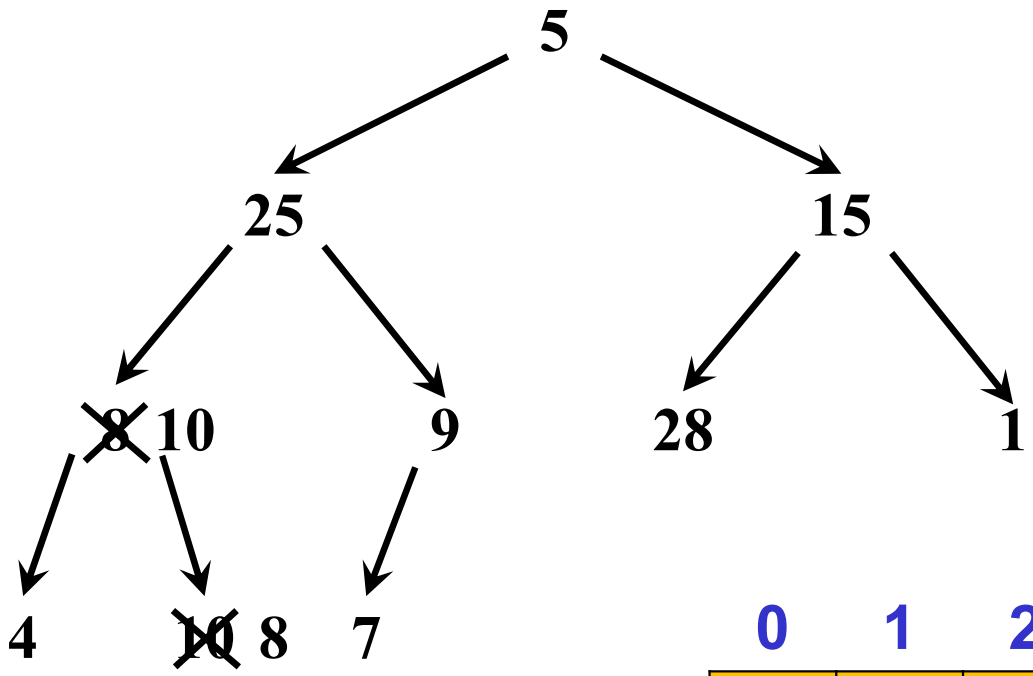
0	1	2	3	4	5	6	7	8	9
5	25	15	8	9	28	1	4	10	7

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



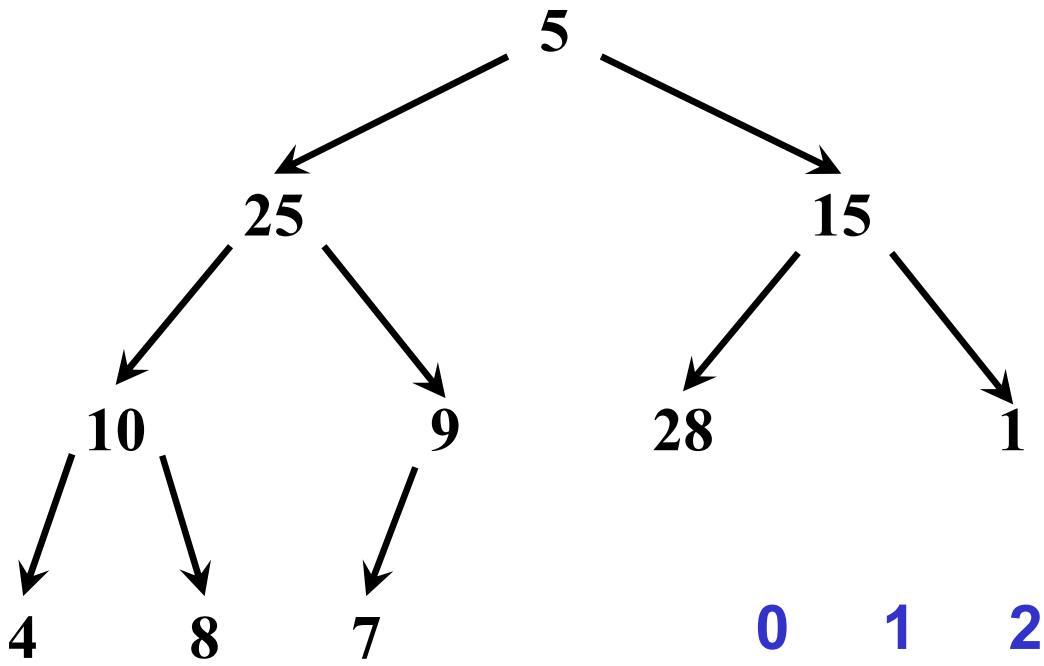
0	1	2	3	4	5	6	7	8	9
5	25	15	8	9	28	1	4	10	7

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



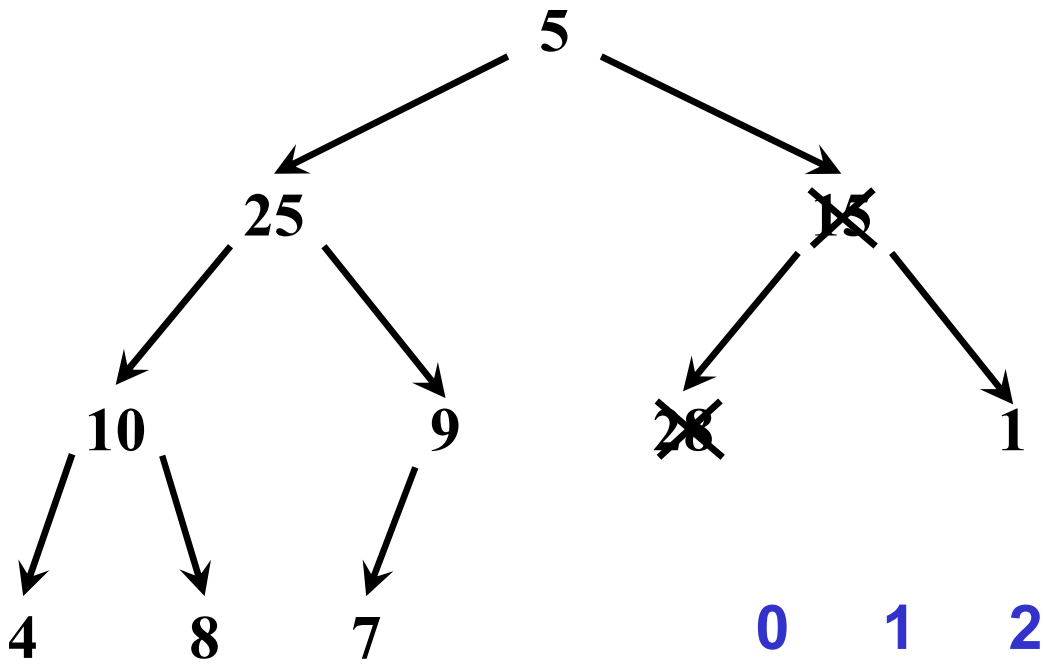
0	1	2	3	4	5	6	7	8	9
5	25	15	10	9	28	1	4	8	7

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



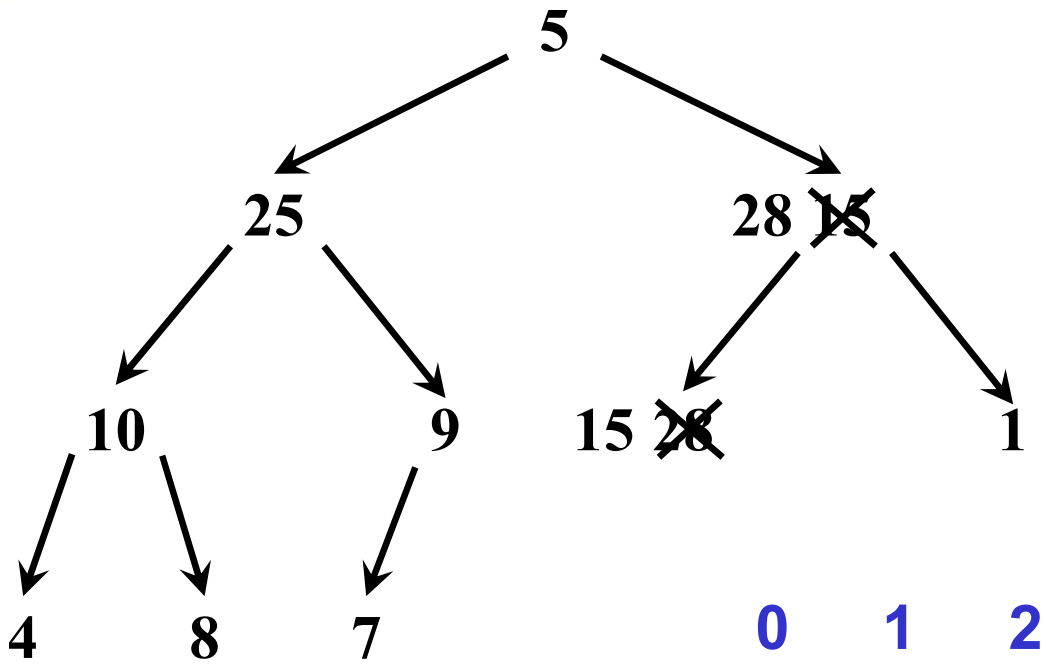
0	1	2	3	4	5	6	7	8	9
5	25	15	10	9	28	1	4	8	7

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



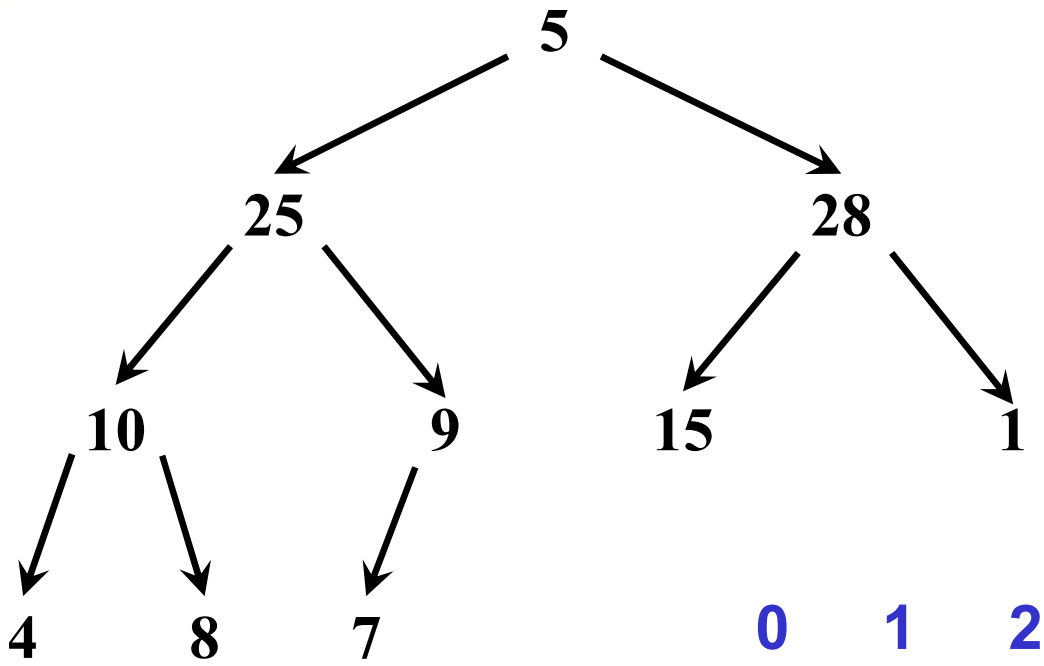
0	1	2	3	4	5	6	7	8	9
5	25	15	10	9	28	1	4	8	7

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



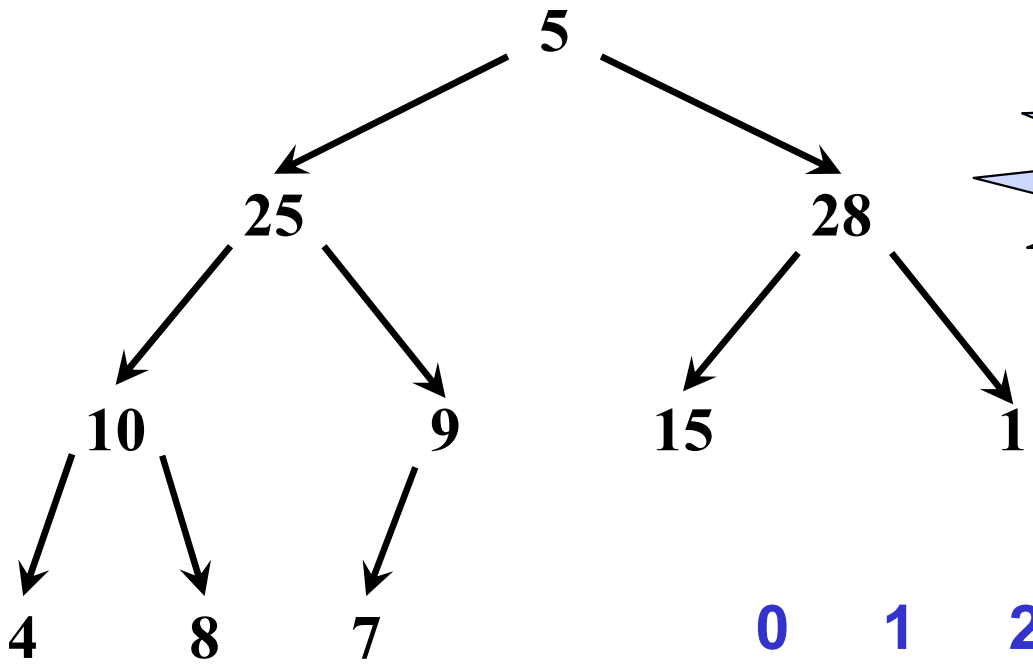
0	1	2	3	4	5	6	7	8	9
5	25	28	10	9	15	1	4	8	7

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
5	25	28	10	9	15	1	4	8	7

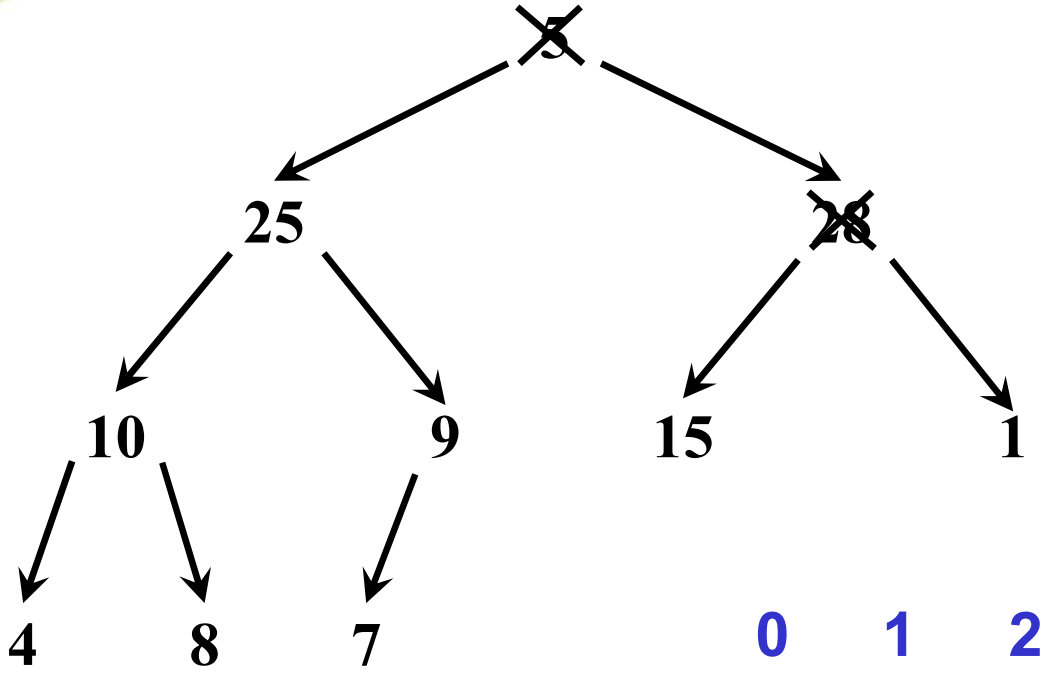
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



Giữ nguyên

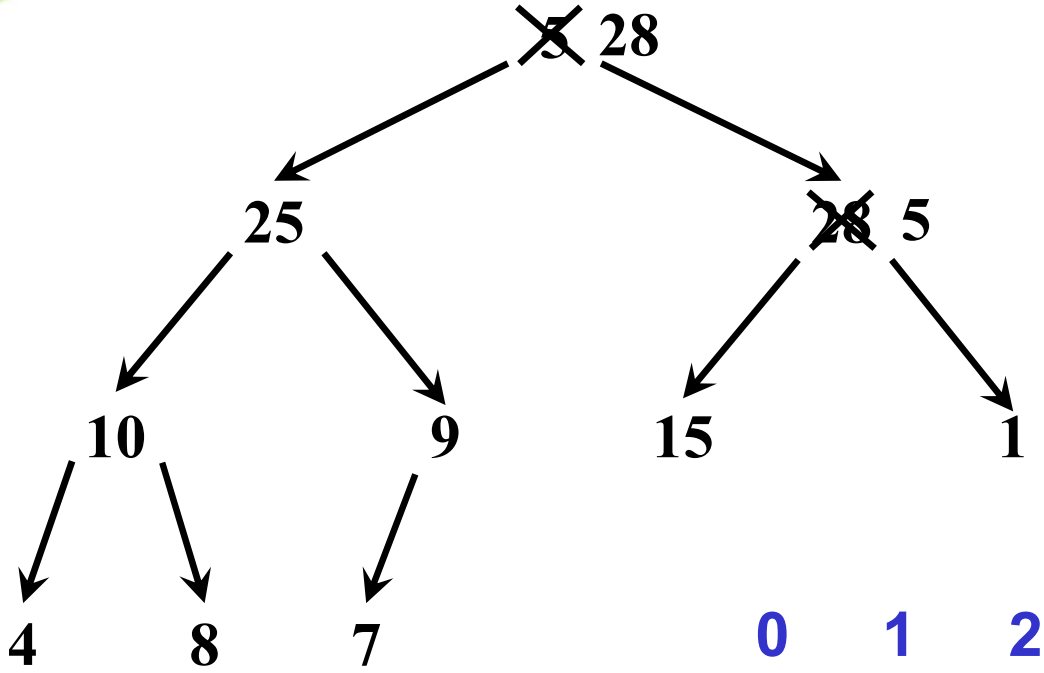
0	1	2	3	4	5	6	7	8	9
5	25	28	10	9	15	1	4	8	7

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



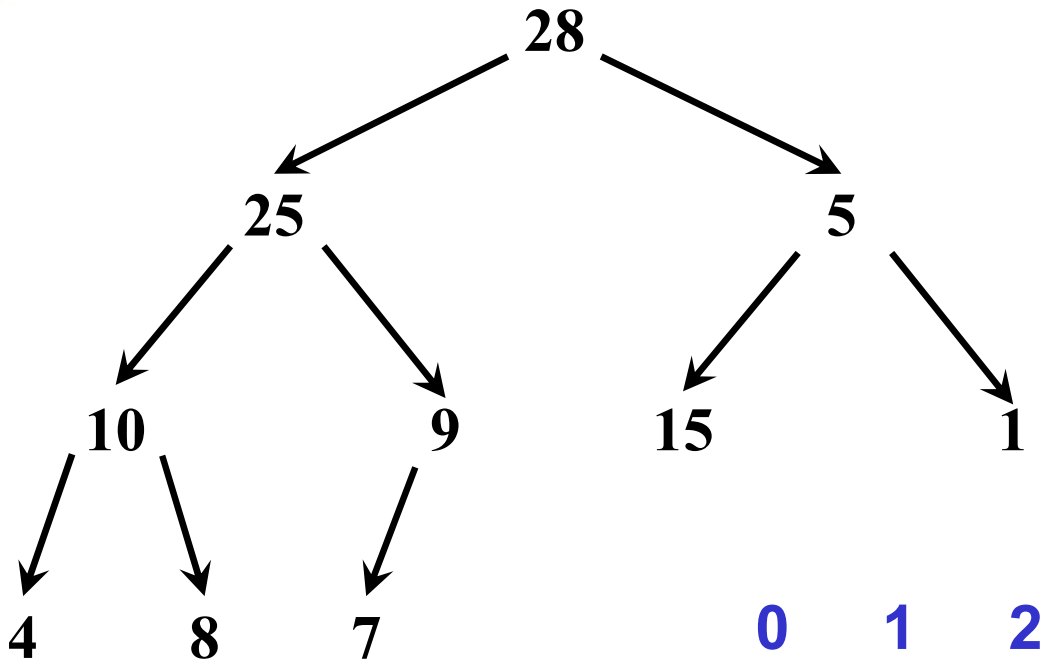
0	1	2	3	4	5	6	7	8	9
5	25	28	10	9	15	1	4	8	7

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



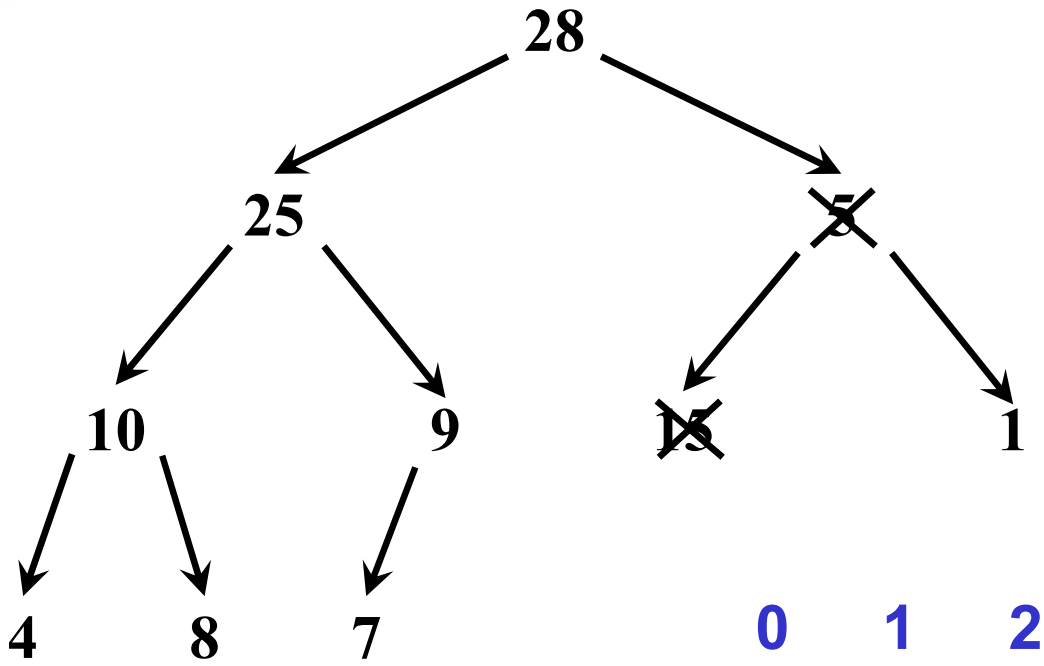
0	1	2	3	4	5	6	7	8	9
28	25	5	10	9	15	1	4	8	7

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



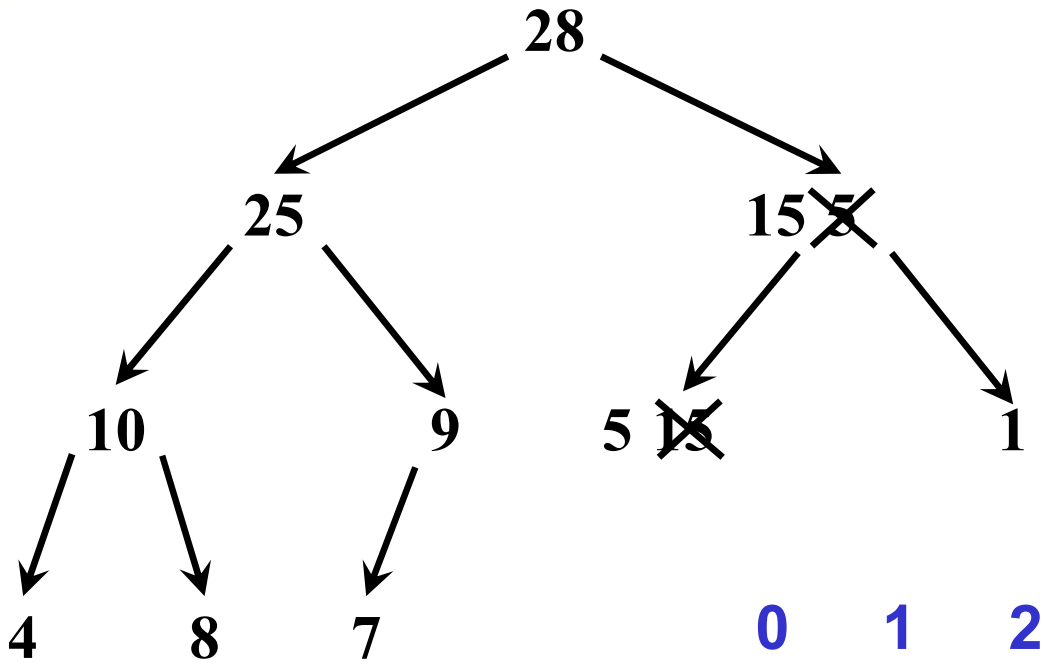
0	1	2	3	4	5	6	7	8	9
28	25	5	10	9	15	1	4	8	7

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



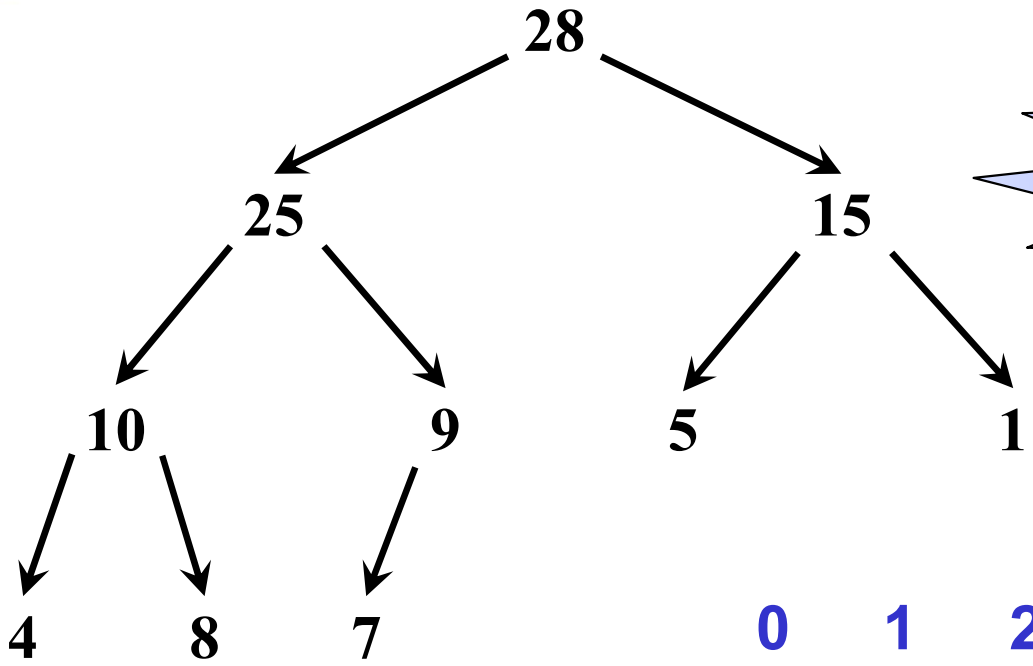
0	1	2	3	4	5	6	7	8	9
28	25	5	10	9	15	1	4	8	7

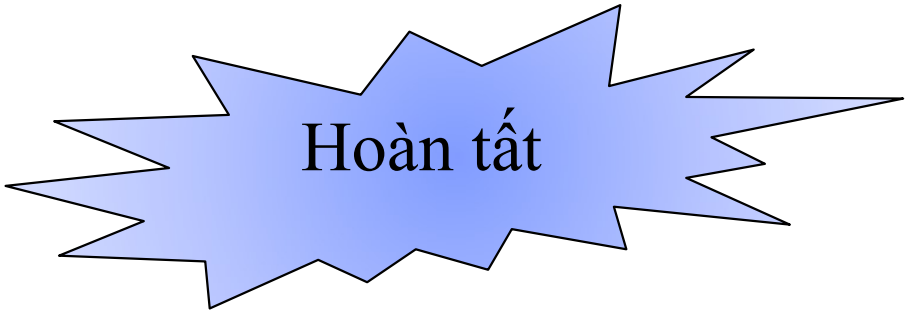
0	1	2	3	4	5	6	7	8	9
5	25	15	8	7	28	1	4	10	9



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

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

Chạy từng bước xây dựng heap

- Bài toán: Hãy xây dựng mảng sau thành một Max Heap (yêu cầu sắp mảng tăng).

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

- Kết quả

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

TƯ TƯỞNG THUẬT TOÁN

Tư tưởng thuật toán

THUẬT TOÁN HEAP SORT

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

```
11.void HeapSort(int a[],int n)
12.{
13.|    BuildHeap(a,n);
14.|    int length = n;
15.|    while(length>1)
16.|    {
17.|    |    HoanVi(a[0],a[length-1]);
18.|    |    length--;
19.|    |    Heapify(a,length,0);
20.|    }
21.}
```

Bước 1 – Xây dựng Heap

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

```
11. void HeapSort(int a[],int n)
12. {
13.     BuildHeap(a,n);
14.     int length = n;
15.     while(length>1)
16.     {
17.         HoanVi(a[0],a[length-1]);
18.         length--;
19.         Heapify(a,length,0);
20.     }
21. }
```

Thuật toán heap sort

— Hàm cài đặt

```
11.void BuildHeap(int a[],int n)
12.{
13.    for(int i=n/2-1;i>=0;i--)
14.        Heapify(a,n,i);
15.}
```

Thuật toán heap sort

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

CHẠY TỪNG BƯỚC THUẬT TOÁN

Chạy từng bước 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

Chạy từng bước thuật toán

— Bước 01: Xây dựng heap

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

— Kết quả

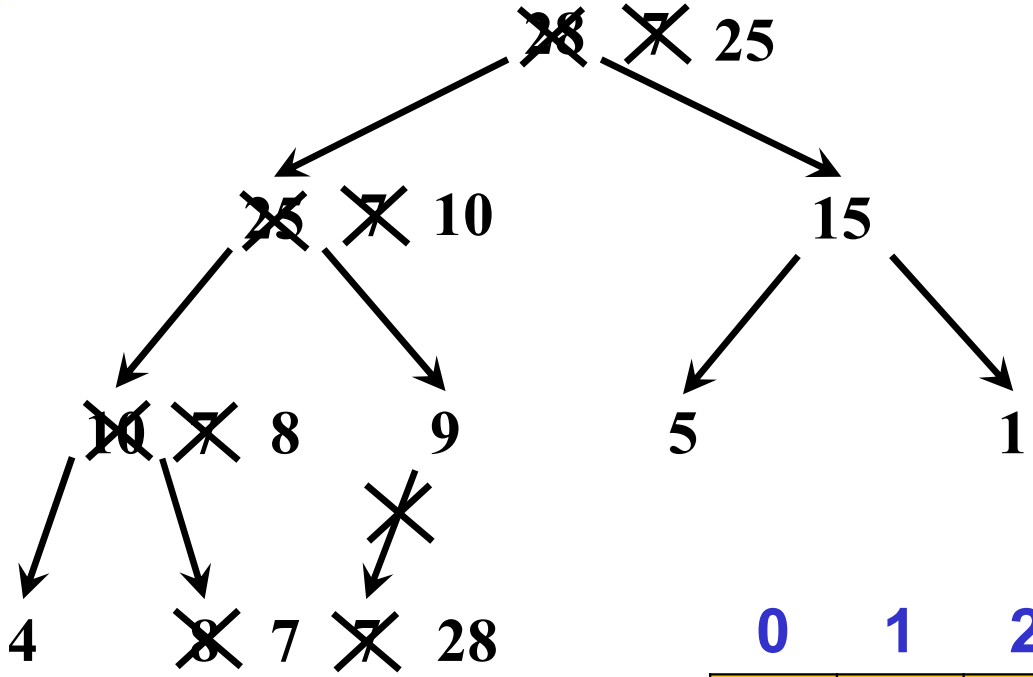
0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 01

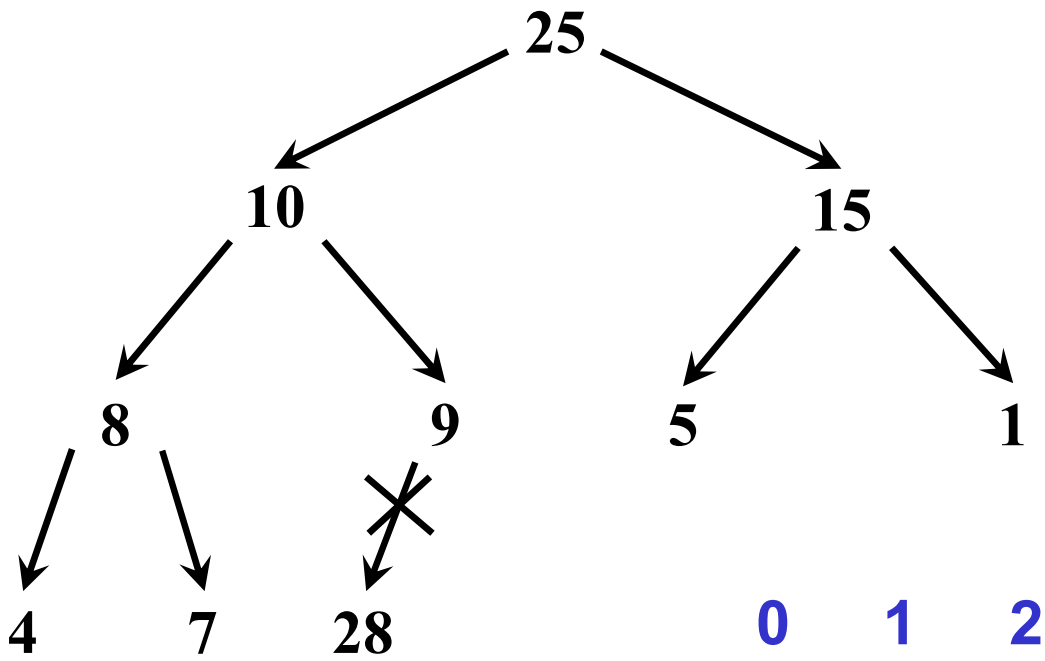
0	1	2	3	4	5	6	7	8	9
28	25	15	10	9	5	1	4	8	7

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



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
28	25	15	10	9	5	1	4	8	7



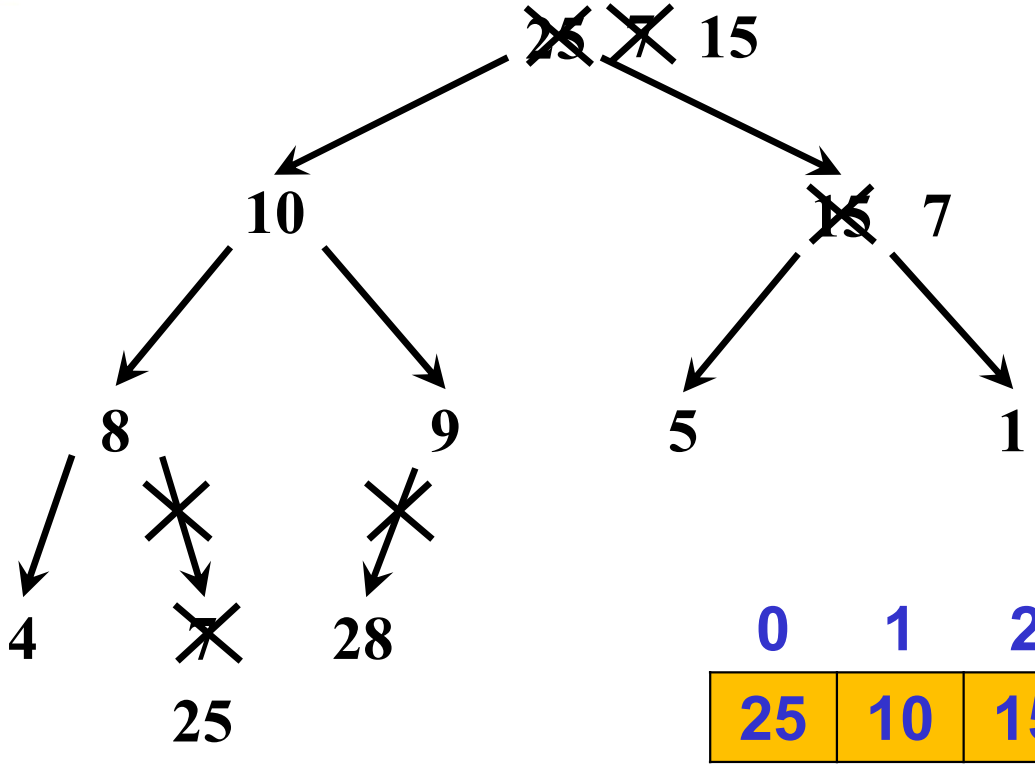
0	1	2	3	4	5	6	7	8	9
25	10	15	8	9	5	1	4	7	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 02

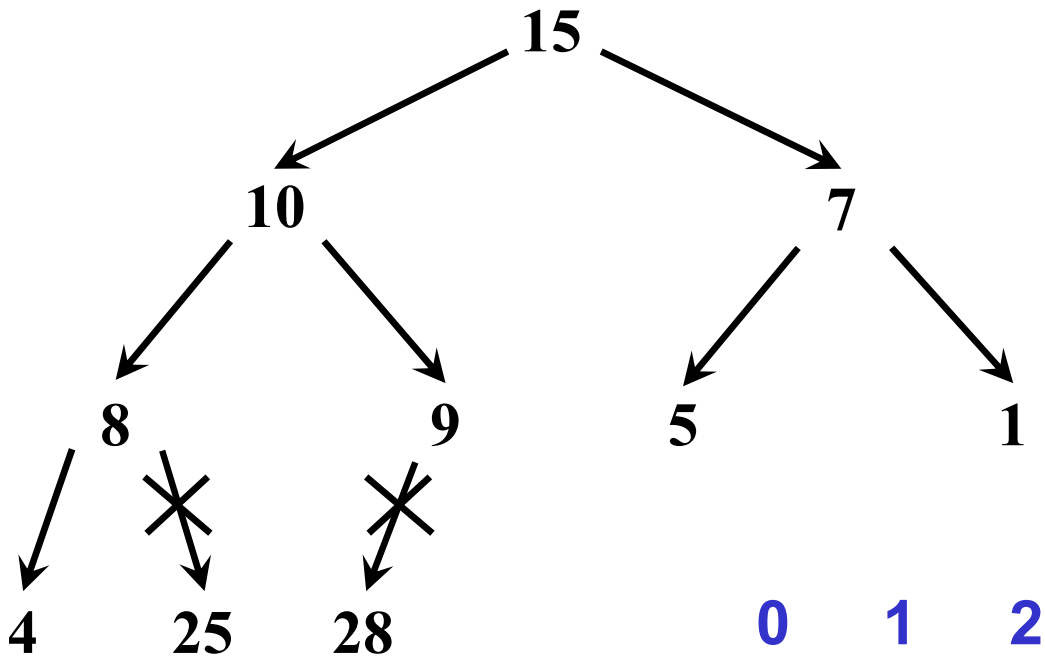
0	1	2	3	4	5	6	7	8	9
25	10	15	8	9	5	1	4	7	28

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



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
25	10	15	8	9	5	1	4	7	28



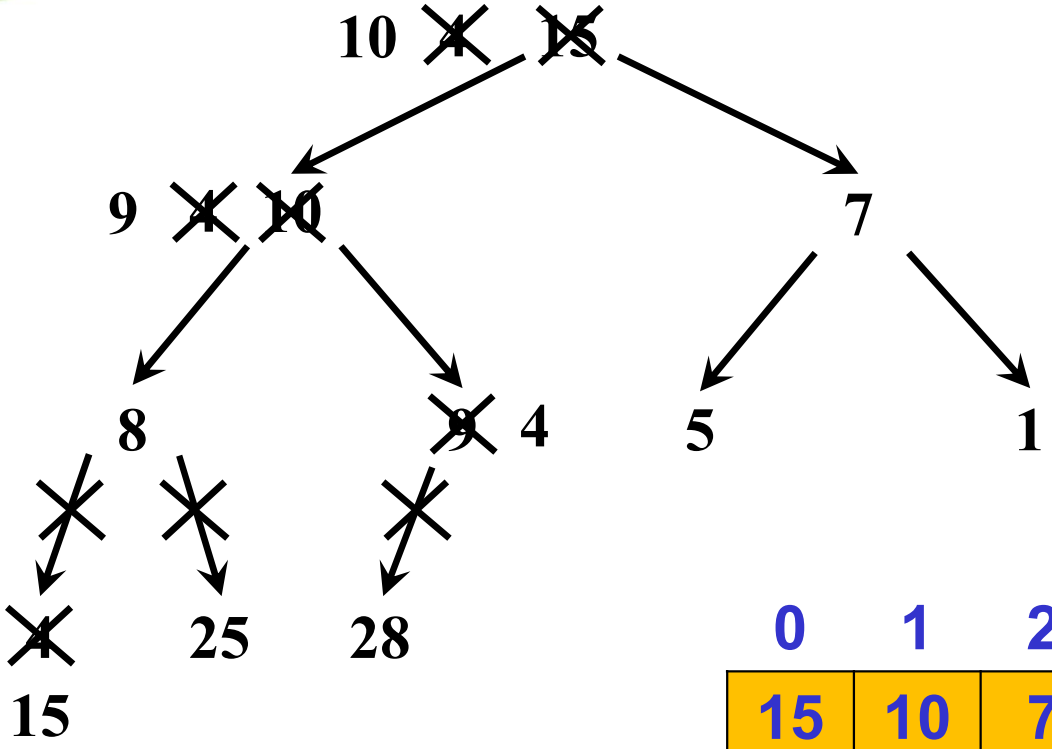
0	1	2	3	4	5	6	7	8	9
15	10	7	8	9	5	1	4	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 03

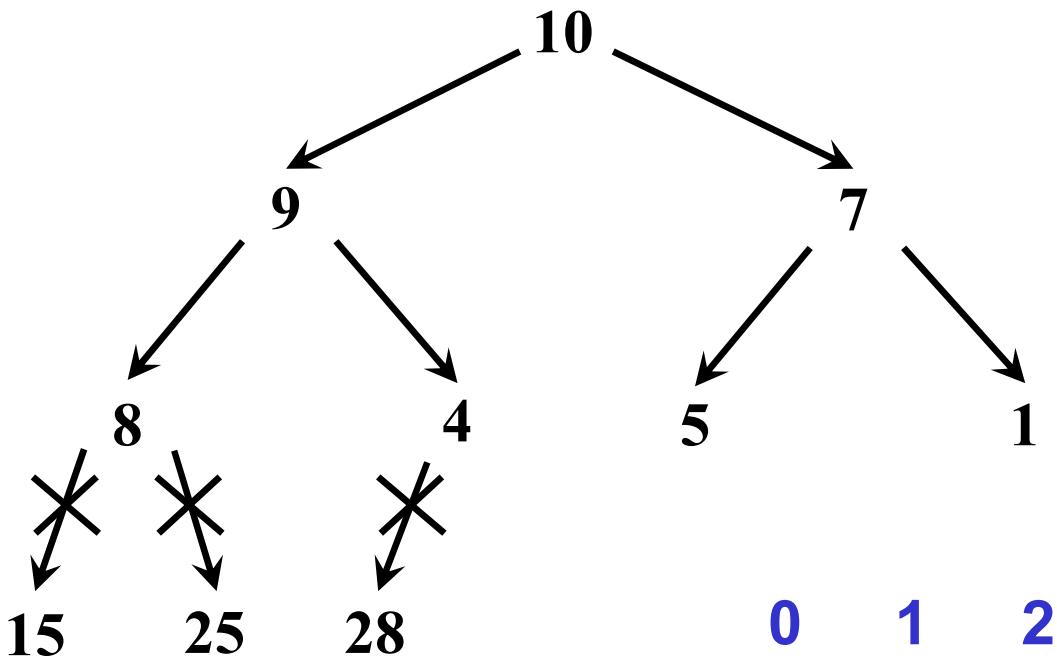
0	1	2	3	4	5	6	7	8	9
15	10	7	8	9	5	1	4	25	28

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



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
15	10	7	8	9	5	1	4	25	28



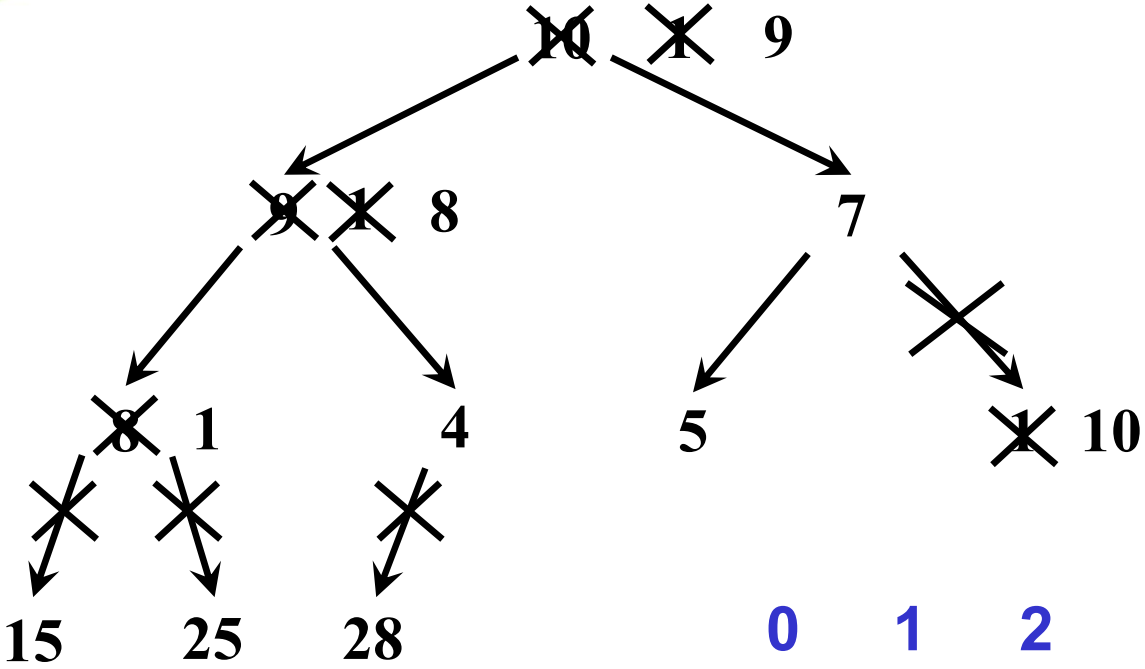
0	1	2	3	4	5	6	7	8	9
10	9	7	8	4	5	1	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 04

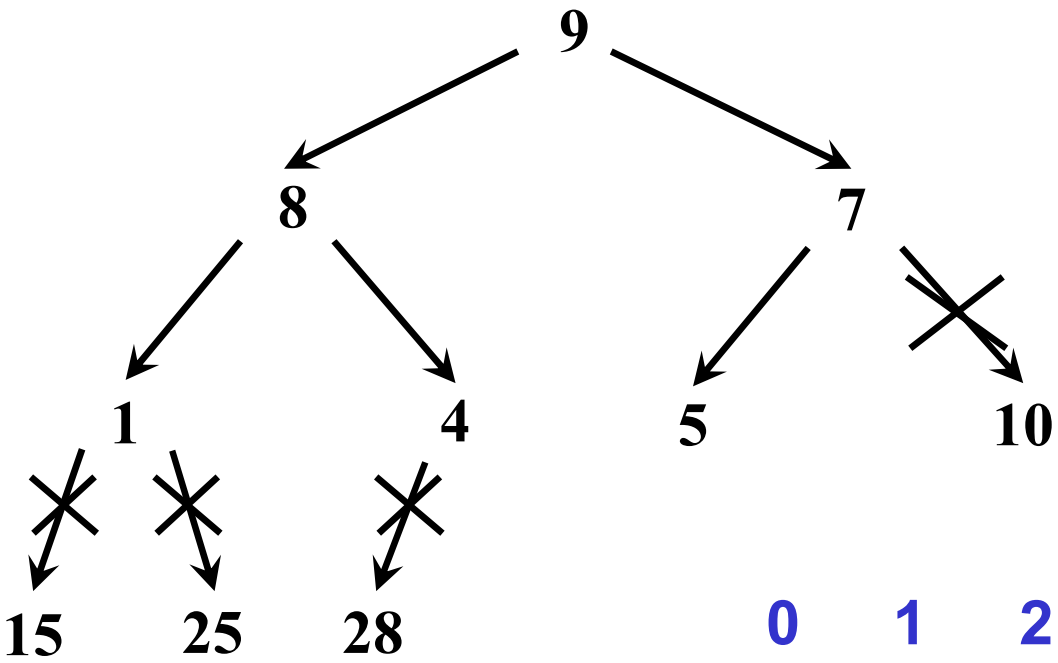
0	1	2	3	4	5	6	7	8	9
10	9	7	8	4	5	1	15	25	28

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



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

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



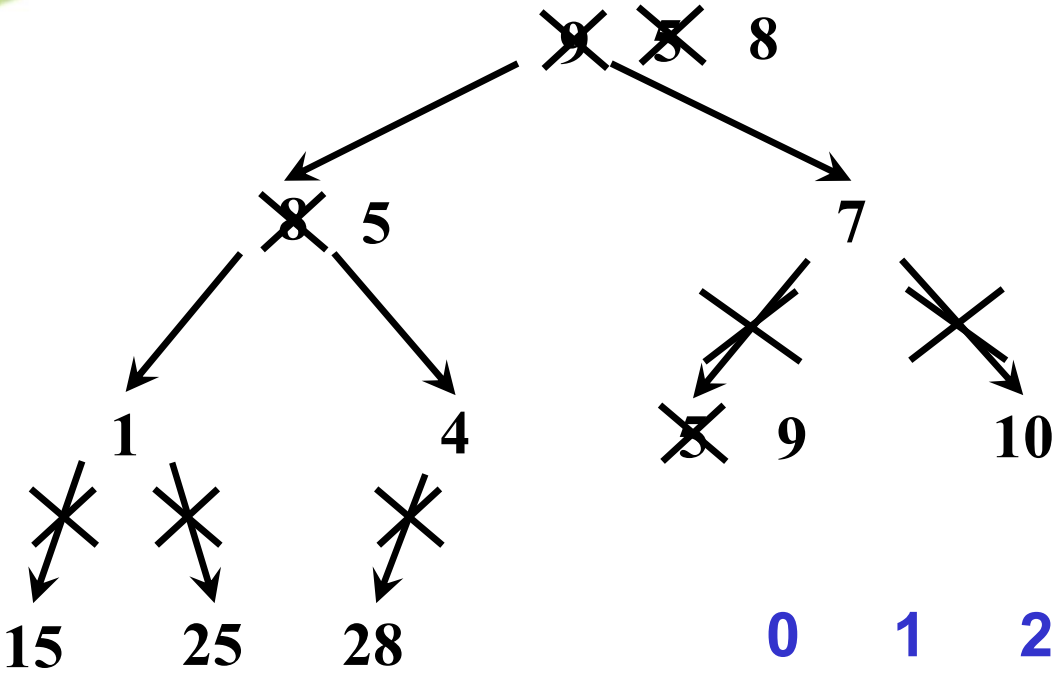
0	1	2	3	4	5	6	7	8	9
9	8	7	1	4	5	10	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 05

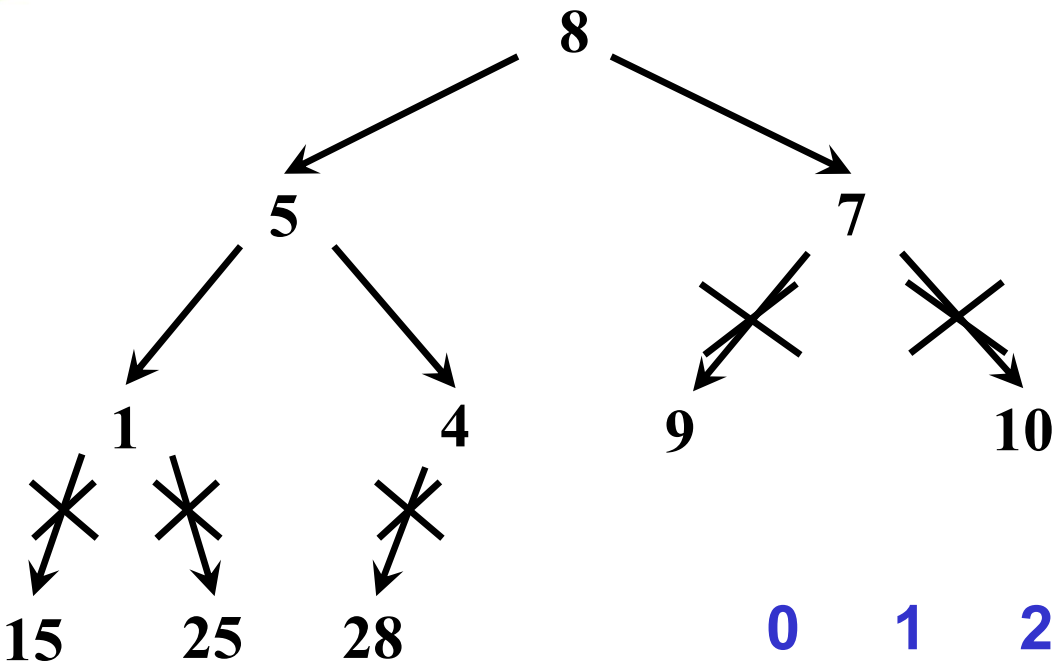
0	1	2	3	4	5	6	7	8	9
9	8	7	1	4	5	10	15	25	28

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



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

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



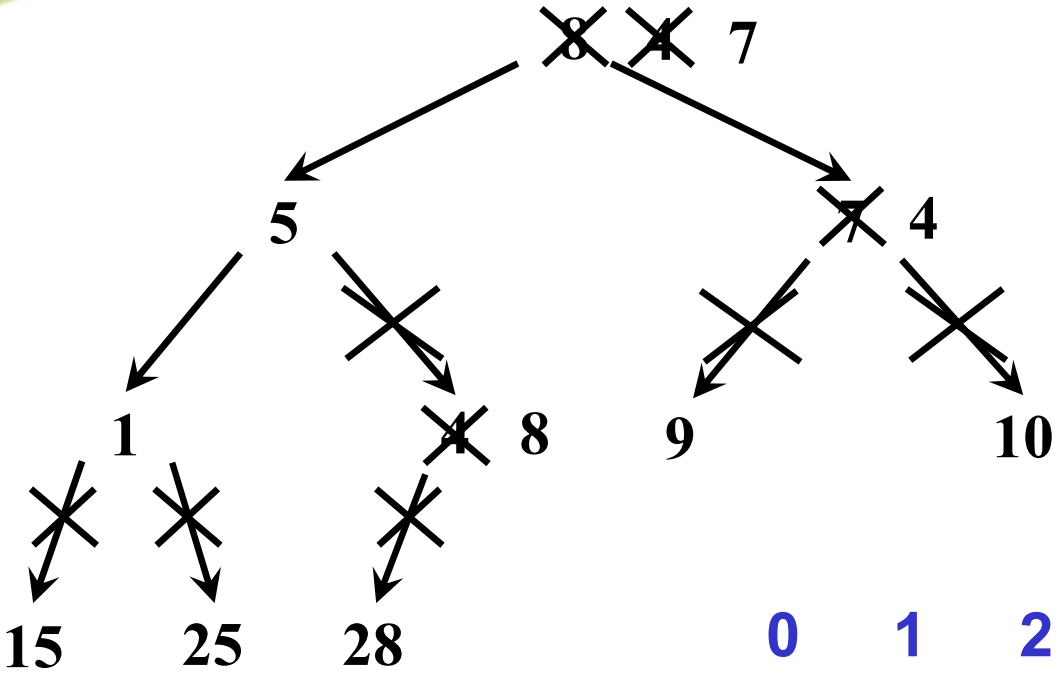
0	1	2	3	4	5	6	7	8	9
8	5	7	1	4	9	10	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 06

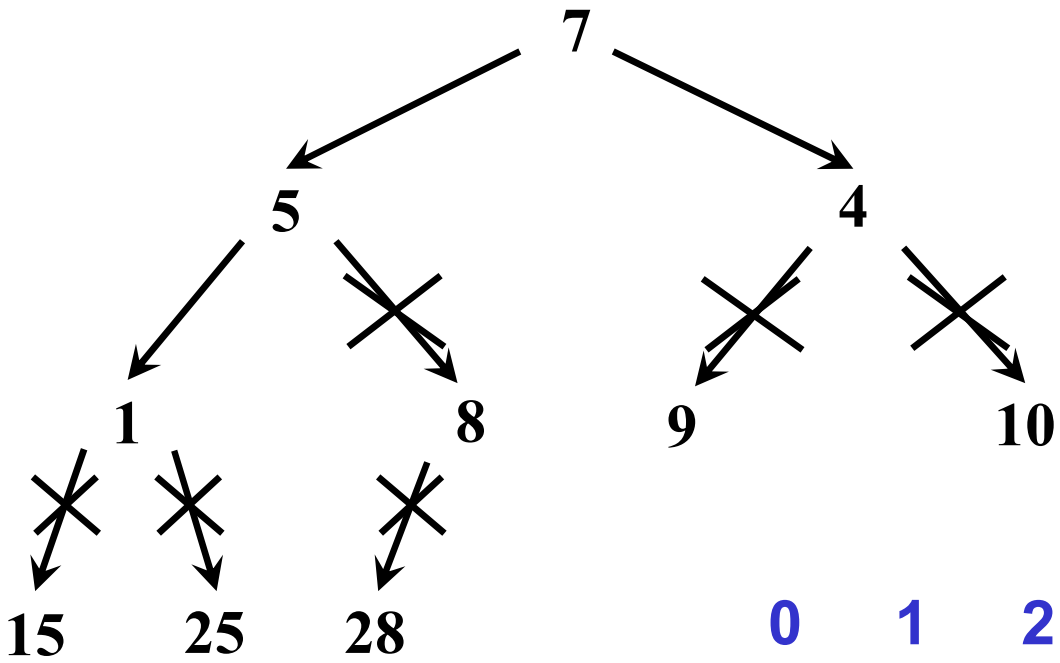
0	1	2	3	4	5	6	7	8	9
8	5	7	1	4	9	10	15	25	28

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



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

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



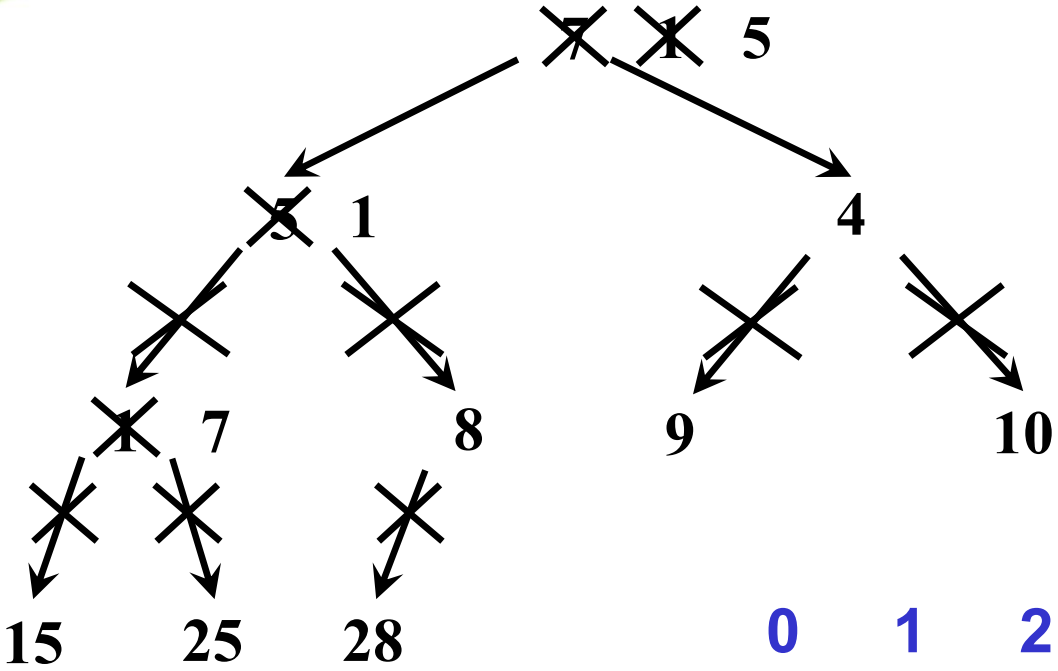
0	1	2	3	4	5	6	7	8	9
7	5	4	1	8	9	10	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 07

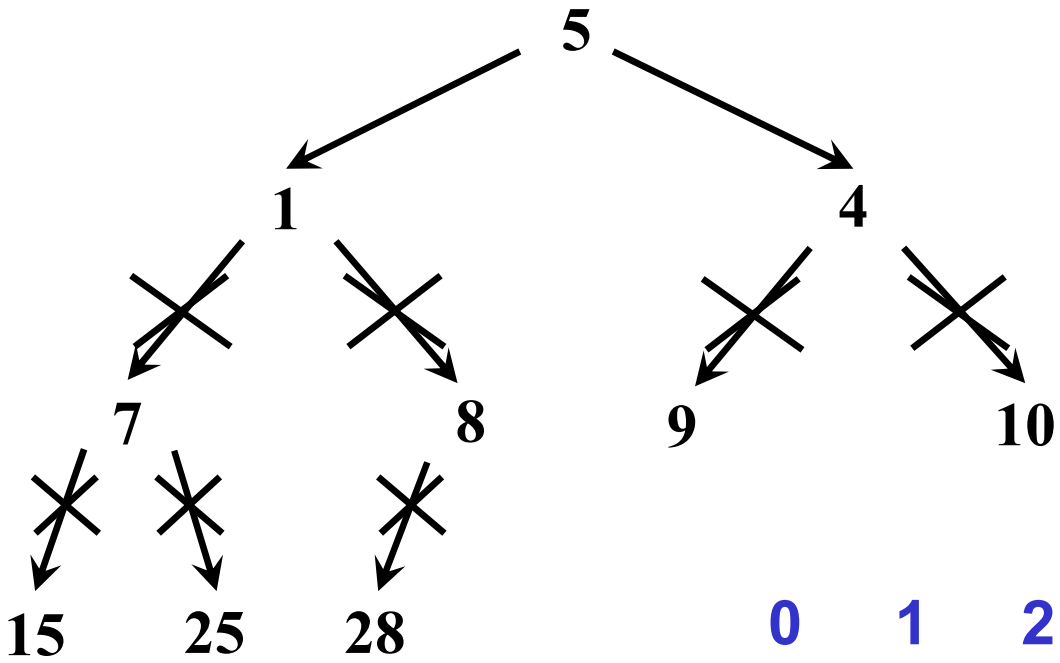
0	1	2	3	4	5	6	7	8	9
7	5	4	1	8	9	10	15	25	28

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



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

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



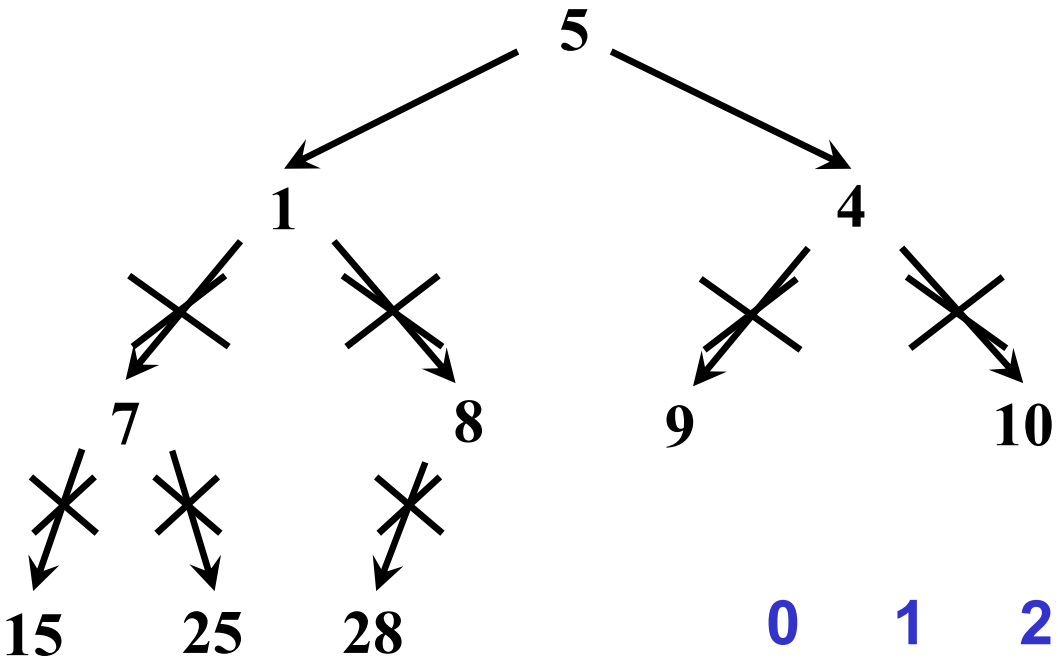
0	1	2	3	4	5	6	7	8	9
5	1	4	7	8	9	10	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 08

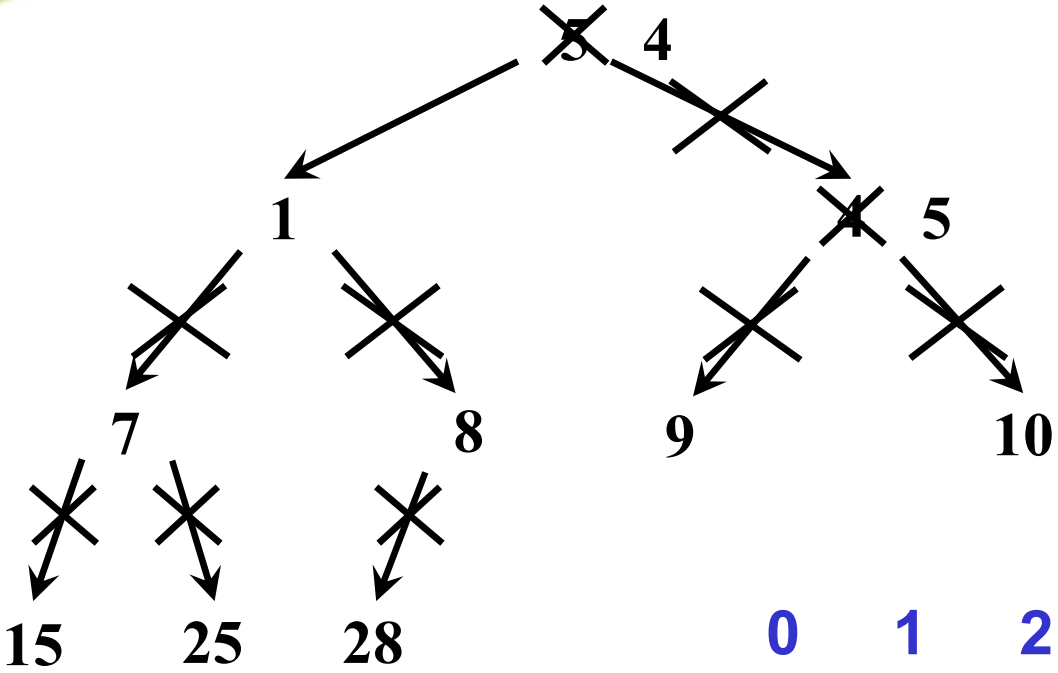
0	1	2	3	4	5	6	7	8	9
5	1	4	7	8	9	10	15	25	28

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



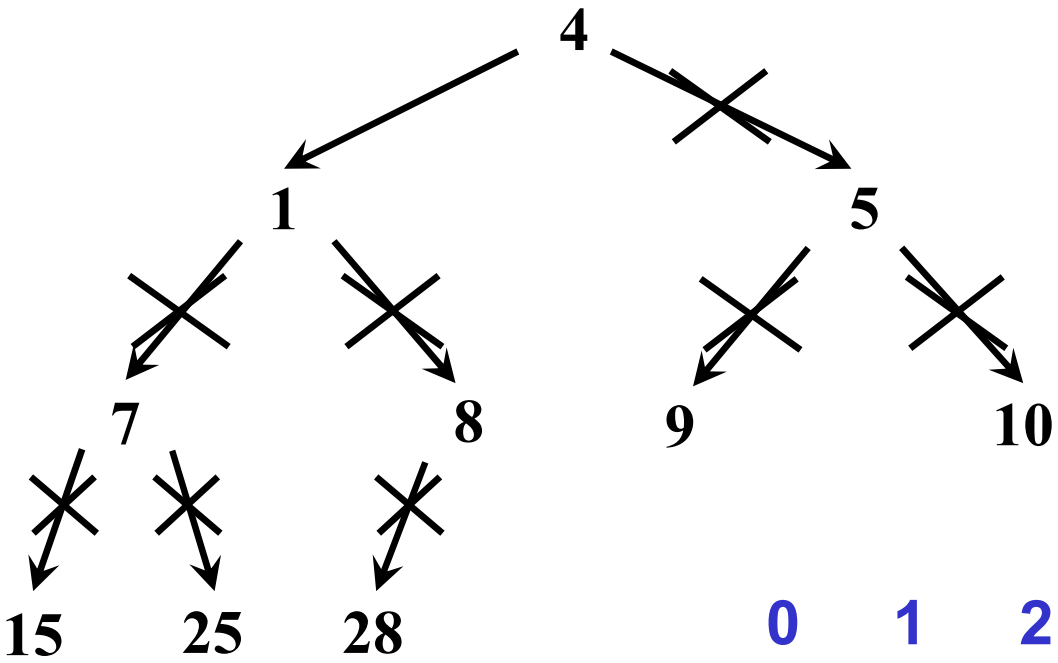
0	1	2	3	4	5	6	7	8	9
5	1	4	7	8	9	10	15	25	28

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



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

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



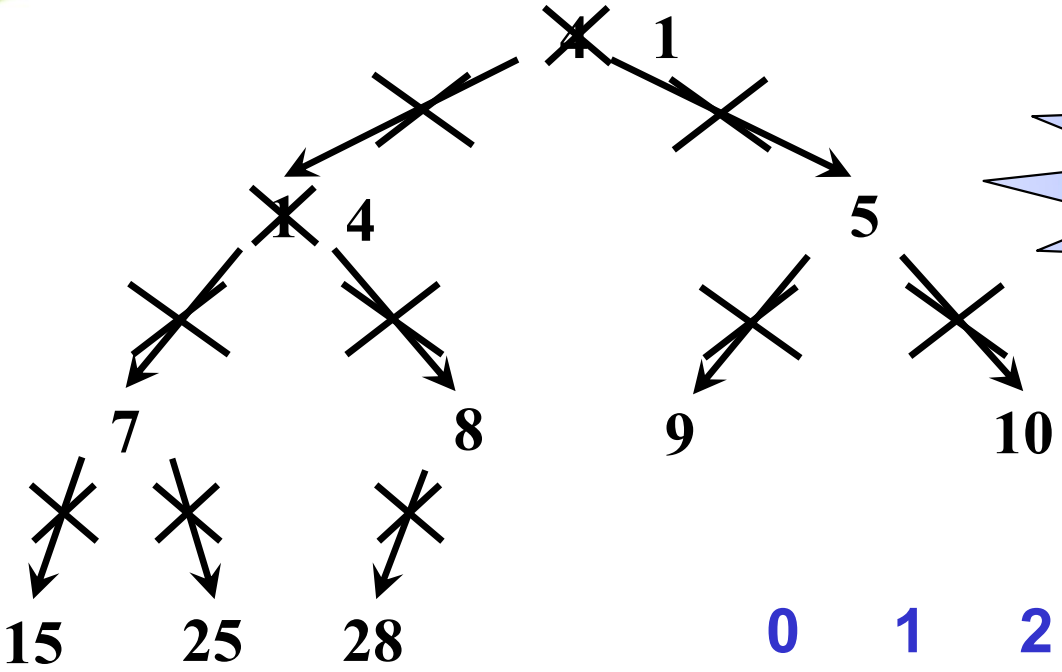
0	1	2	3	4	5	6	7	8	9
4	1	5	7	8	9	10	15	25	28

Chạy từng bước thuật toán

– Bước 02: Sắp xếp – Lần lặp 09

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

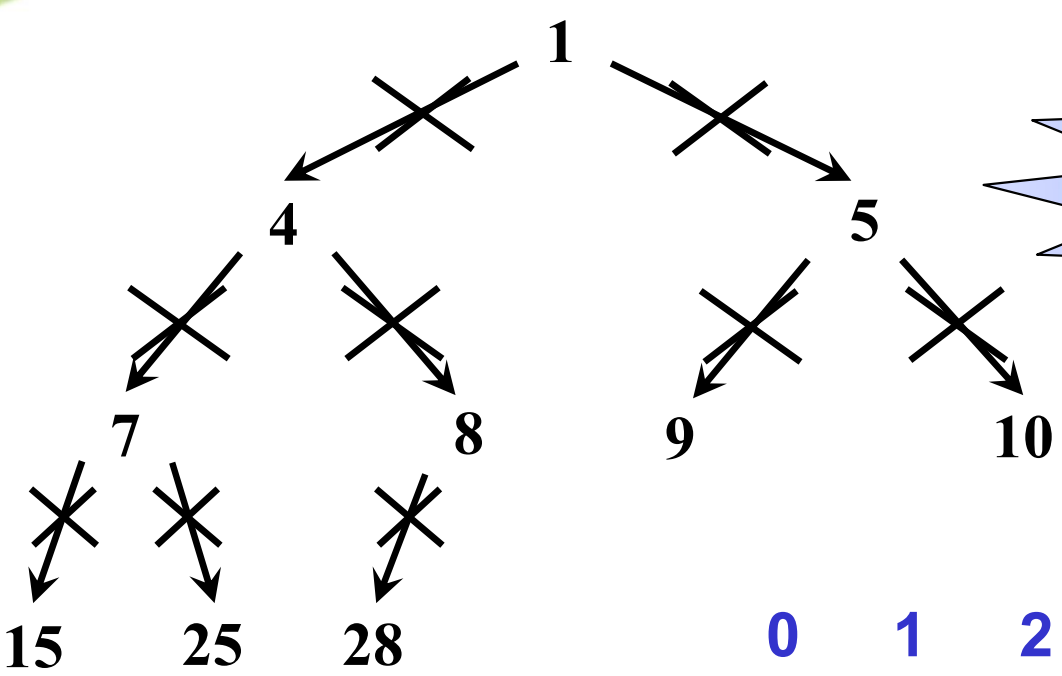
0	1	2	3	4	5	6	7	8	9
4	1	5	7	8	9	10	15	25	28



Hoàn tất

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

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



Hoàn tất

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

Thuật toán Heap sort

ĐẶC ĐIỂM – ĐIỂM MẠNH – ĐIỂM YẾU

Đặc điểm – điểm mạnh – điểm yếu

— Đặc điểm thuật toán Heap sort:

+ Độ phức tạp về thời gian (time complexity): $O(n\log(n))$.
+ Độ phức tạp về bộ nhớ (space complexity): $O(1)$.
+ Trường hợp xấu nhất (worst case): $O(n\log(n))$.
+ Trường hợp trung bình (average case): $O(n\log(n))$.
+ Trường hợp tốt nhất (best case): $O(n\log(n))$.
+ Thuộc họ giải thuật chia để trị (Divide and Conquer).
+ Không ổn định.

Đặc điểm – điểm mạnh – điểm yếu

- Điểm mạnh:
  - + Hiệu quả trên dữ liệu lớn.
  - + Hiệu suất thuật toán là tối ưu.
  - + Thuật toán thực hiện nhanh.
  - + Thuật toán có thể chạy song song.

Đặc điểm – điểm mạnh – điểm yếu

– Điểm yếu:

Cảm ơn quí vị đã lắng nghe

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN TP.HCM
TOÀN DIỆN – SÁNG TẠO – PHỤNG SỰ