

Hàng đợi ưu tiên (Priority Queues)

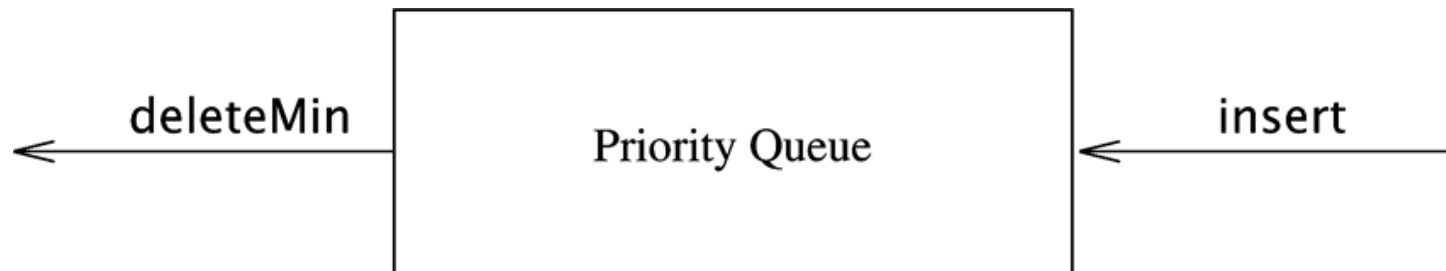
Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin

Trường Đại học Thủy Lợi

Hàng đợi ưu tiên

- Phần tử nhỏ nhất có độ ưu tiên cao nhất và sẽ được lấy ra đầu tiên.
- Chèn (insert)
 - Thời gian $O(\log N)$
- Xóa phần tử nhỏ nhất (deleteMin)
 - Thời gian $O(\log N)$



Cài đặt hàng đợi ưu tiên

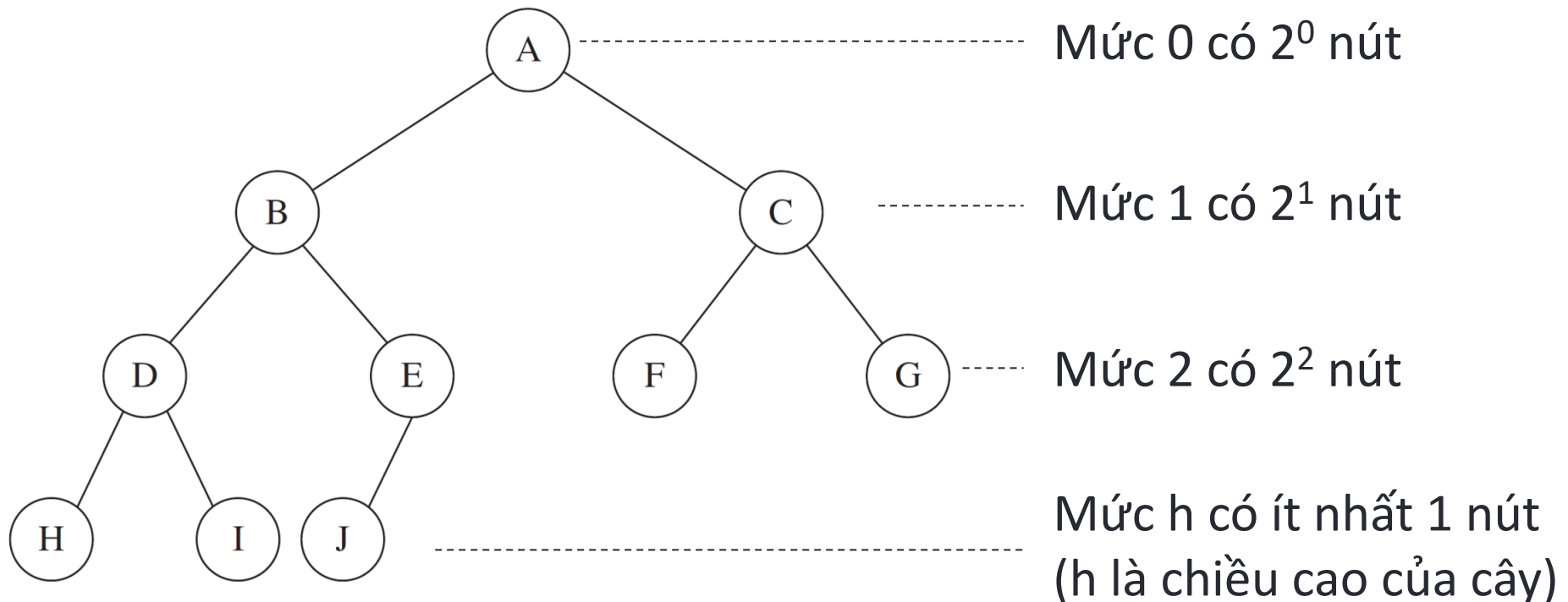
- Dùng danh sách liên kết:
 - insert (dùng pushFront) mất thời gian $O(1)$.
 - deleteMin (quét tìm rồi xóa min) mất thời gian $O(N)$.
- Dùng cây nhị phân tìm kiếm:
 - insert và deleteMin mất thời gian $O(\log N)$.
 - Tuy nhiên, cây nhị phân tìm kiếm quá phức tạp cho yêu cầu đơn giản hơn của hàng đợi ưu tiên.
- ***Đống nhị phân*** (binary heap):
 - Là cách cài đặt phổ biến của hàng đợi ưu tiên.
 - insert và deleteMin mất thời gian $O(\log N)$.

Đồng nhị phân

- Gọi tắt là đồng.
- Thỏa mãn:
 1. Là cây nhị phân đầy đủ.
 2. Có tính chất thứ tự đồng.

Cây nhị phân đầy đủ

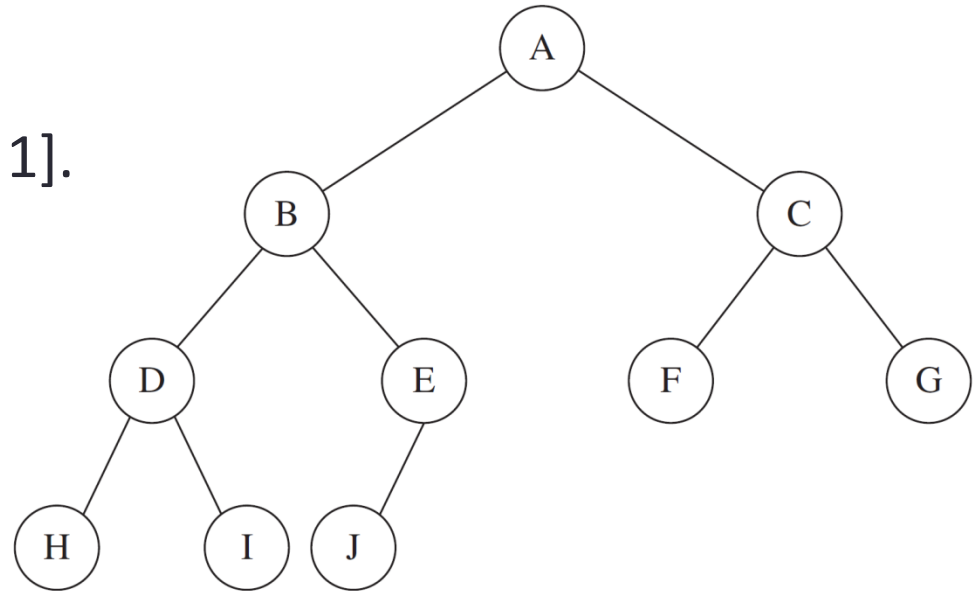
- Là cây nhị phân với tất cả các mức (trừ mức dưới cùng) đã được lấp đầy các nút từ trái sang phải.
- Số nút $n \geq 2^0 + 2^1 + \dots + 2^{h-1} + 1 = 2^h \Rightarrow h \leq \log n$



Cài đặt cây nhị phân đầy đủ

Lưu trữ các phần tử lần lượt từng mức vào vector v .

- Cha của $v[k]$ là $v[k/2]$.
- Con trái của $v[k]$ là $v[2k]$.
- Con phải của $v[k]$ là $v[2k + 1]$.

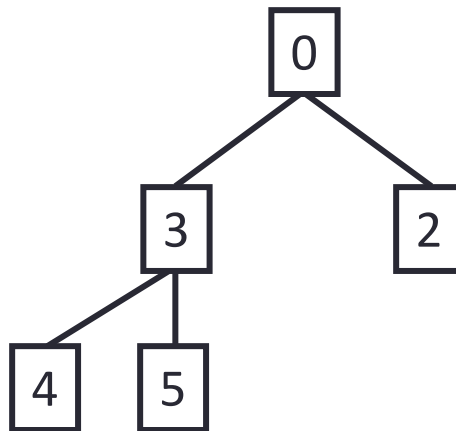


vector v

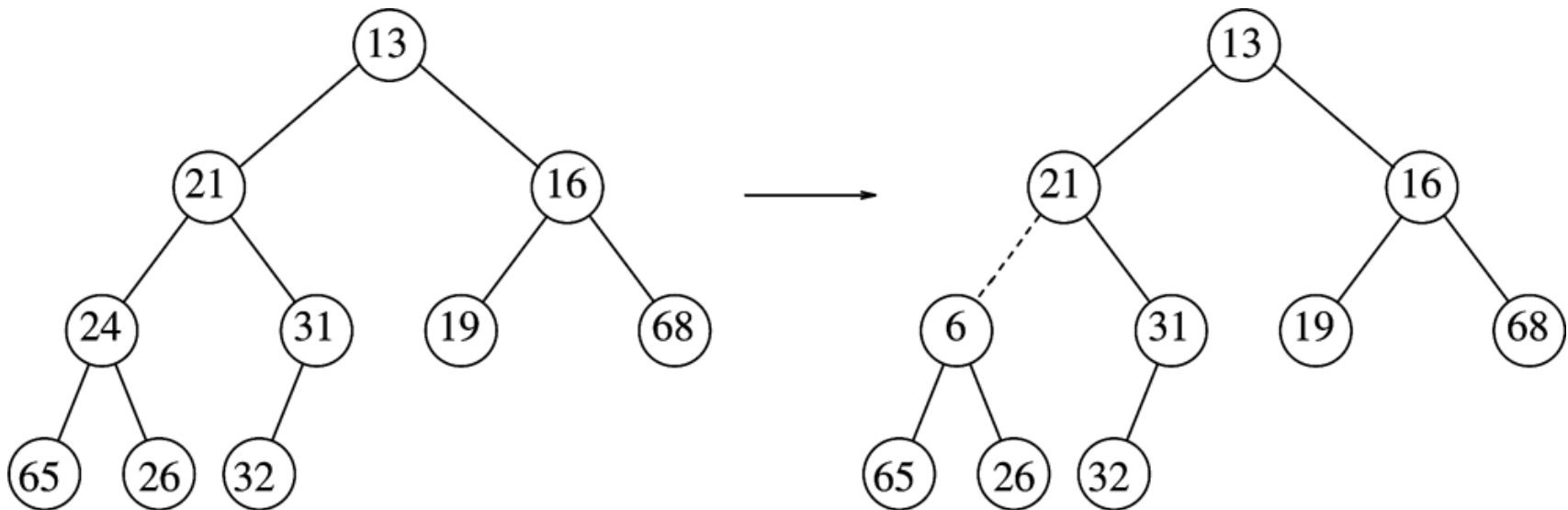
	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Tính chất thứ tự đồng

- Với mọi nút X trên đồng, giá trị của X nhỏ hơn giá trị của các nút con trái và phải của X.
- Suy ra:
 - Phần tử nhỏ nhất trên đồng nằm ở gốc.
 - Không có thứ tự nào giữa các nút con của cùng một nút.



Cây nào là đồng?



Cài đặt hàng đợi ưu tiên trong C++

```
// Kiểu phần tử  
typedef int T;
```

```
// Kiểu đồng nhị phân  
struct BinaryHeap {  
    int currentSize; // Số phần tử hiện có  
    vector<T> array; // Vector chứa các phần tử  
};
```

Ta dùng lớp vector trong thư viện chuẩn của C++

```
// Khởi tạo đồng rỗng với dung lượng đã cho  
void init(BinaryHeap & bh, int capacity = 100);
```

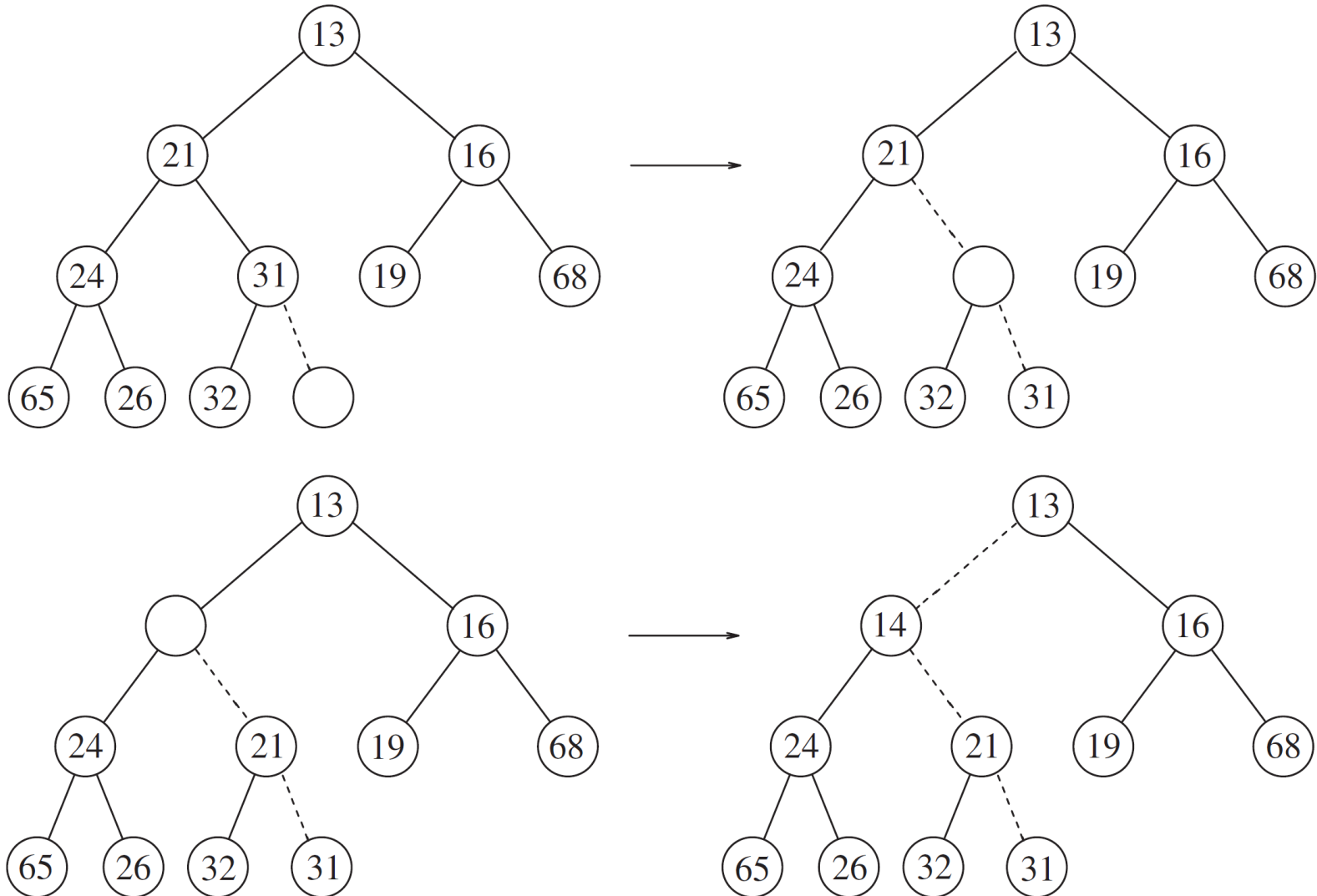
```
// Khởi tạo đồng từ danh sách giá trị đã cho  
void init(BinaryHeap & bh, vector<T> & elems);
```

```
T findMin(BinaryHeap & bh); // Tìm phần tử nhỏ nhất (ở gốc)
```

```
void insert(BinaryHeap & bh, T x); // Chèn x vào đồng
```

```
void deleteMin(BinaryHeap & bh); // Xóa phần tử nhỏ nhất (ở gốc)
```

Chèn vào đồng: insert (bh, 14)



Chèn vào đồng: insert (bh, x)

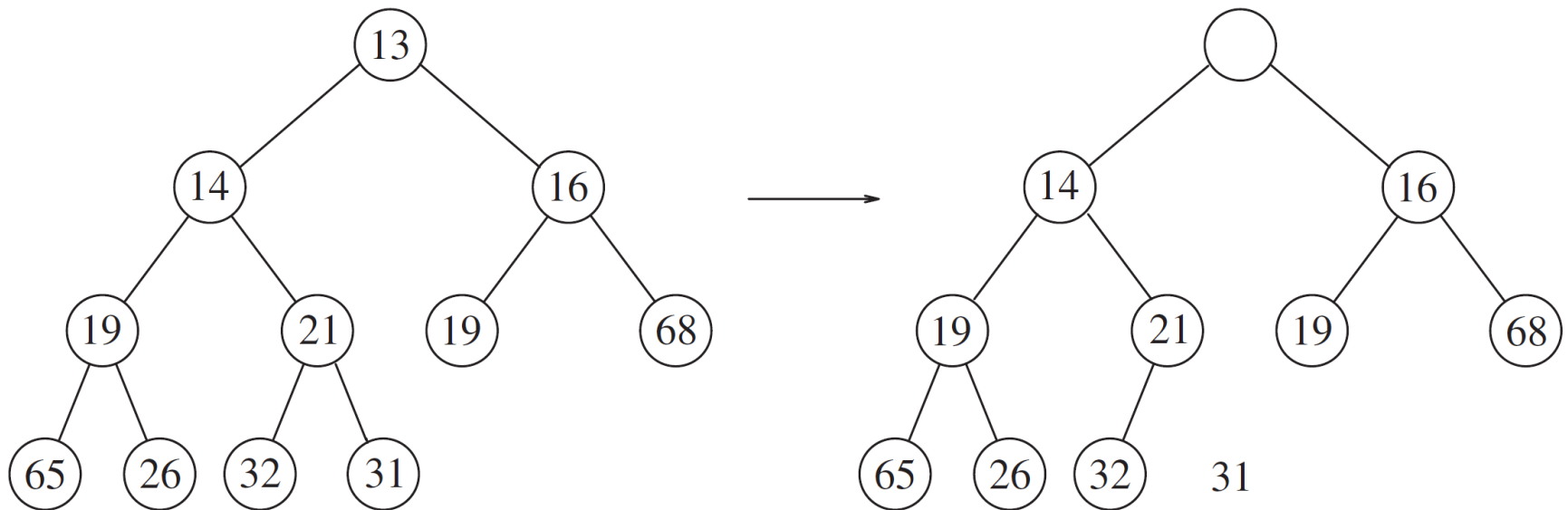
- Tạo lỗ trống (hole) ở vị trí có sẵn tiếp theo trong đồng; x là giá trị của lỗ trống nhưng chưa chính thức đặt vào lỗ trống.
- Lặp lại:
 - Nếu x nhỏ hơn giá trị ở nút cha của lỗ trống:
 - + Đổi chỗ lỗ trống và nút cha
 - Ngược lại:
 - + Dừng lặp
- Đặt x vào lỗ trống.

Quá trình lỗ trống dịch chuyển dần lên trên như vậy được gọi là thẩm thấu ngược (percolate up).

Chèn vào đống: Lập trình

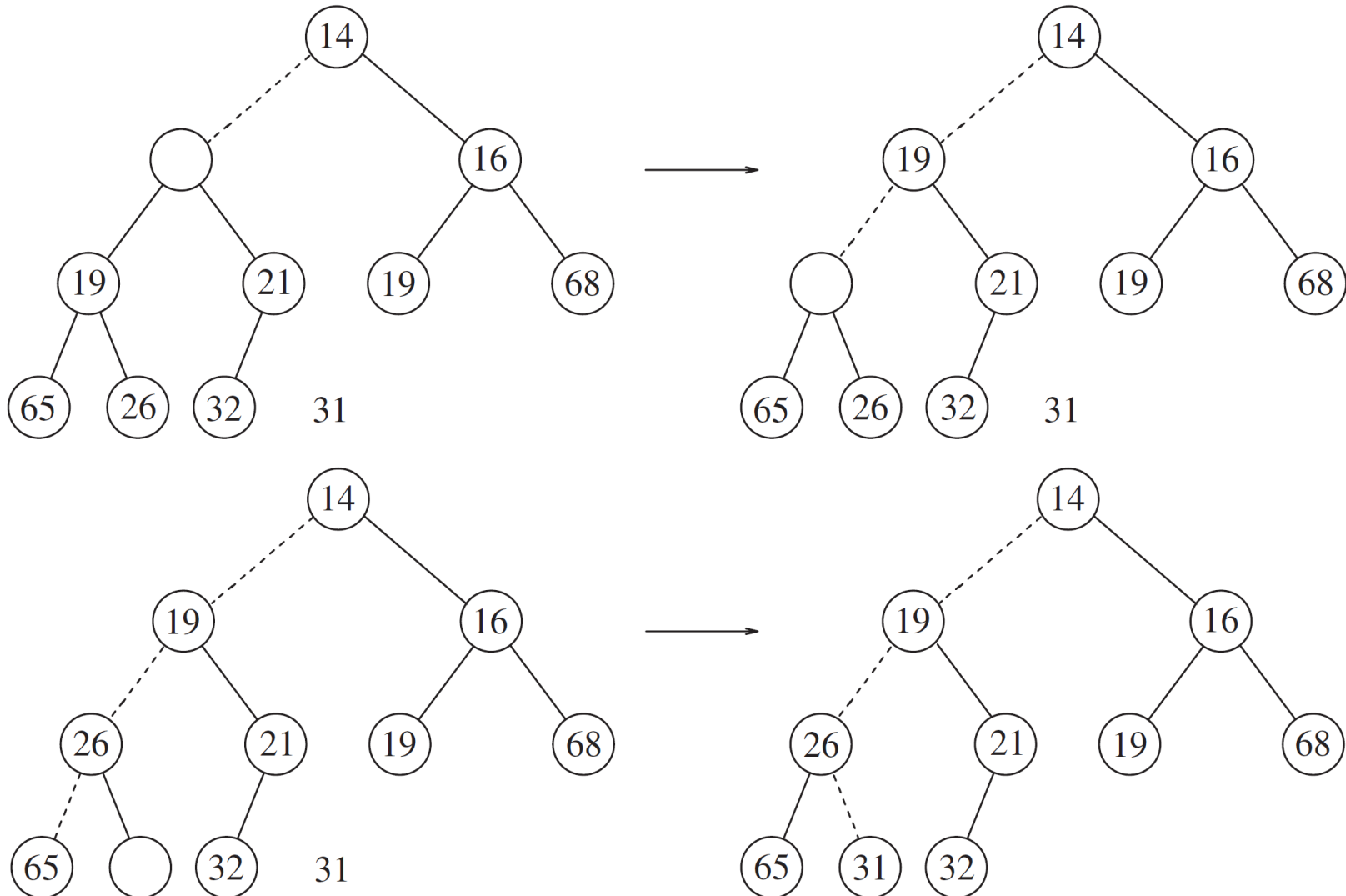
```
void insert(BinaryHeap & bh, T x) {  
    // Tăng kích thước 2 lần nếu vector đầy  
    if (bh.currentSize == bh.array.size() - 1)  
        bh.array.resize(bh.array.size() * 2);  
  
    // Thẩm thấu ngược  
    bh.currentSize++;  
    int hole = bh.currentSize;  
    while (hole > 1 && x < bh.array[hole / 2]) {  
        bh.array[hole] = bh.array[hole / 2];  
        hole = hole / 2;  
    }  
    bh.array[hole] = x;  
}
```

Xóa khỏi đồng: Ví dụ



Tạo lỗ trống ở gốc; Xóa phần tử cuối cùng ($x = 31$)

Xóa khỏi đồng: Ví dụ (tiếp)



Xóa khỏi đống: Thuật toán

- Tạo lỗ trống ở nút gốc.
- Gọi x là giá trị ở nút cuối cùng.
- Xóa nút cuối cùng.
- Lặp lại:
 - Xác định nút con nhỏ hơn của lỗ trống
 - Nếu x lớn hơn giá trị ở nút con nhỏ hơn:
 - + Đổi chỗ lỗ trống và nút con nhỏ hơn (thẩm thấu xuôi – percolate down)
 - Ngược lại:
 - + Dừng lặp
- Đặt x vào lỗ trống.

Xóa khỏi đống: Lập trình

```
void deleteMin(BinaryHeap & bh) {  
    bh.array[1] = bh.array[bh.currentSize];  
    bh.currentSize--;  
  
    // Thăm thấu xuôi (xem slide sau)  
    percolateDown(bh, 1);  
}
```


Thăm thấu xuôi

```
void percolateDown(BinaryHeap & bh, int hole) {
    T x = bh.array[hole];
    int child;
    while (hole * 2 <= bh.currentSize) {
        child = hole * 2;
        if (child < bh.currentSize
            && bh.array[child + 1] < bh.array[child])
            child++;
        if (x > bh.array[child]) {
            bh.array[hole] = bh.array[child];
            hole = child;
        }
        else
            break;
    }
    bh.array[hole] = x;
}
```

Thời gian chạy của chèn và xóa

- Số phép so sánh (số đoạn đứt nét) trong quá trình thẩm thấu ngược (chèn) hoặc xuôi (xóa) không quá chiều cao của cây.
- Ta đã biết chiều cao của cây $h \leq \log n$.
- Suy ra, thời gian chạy của các thao tác chèn và xóa là $O(\log n)$.

Xây dựng đồng từ danh sách giá trị đã cho

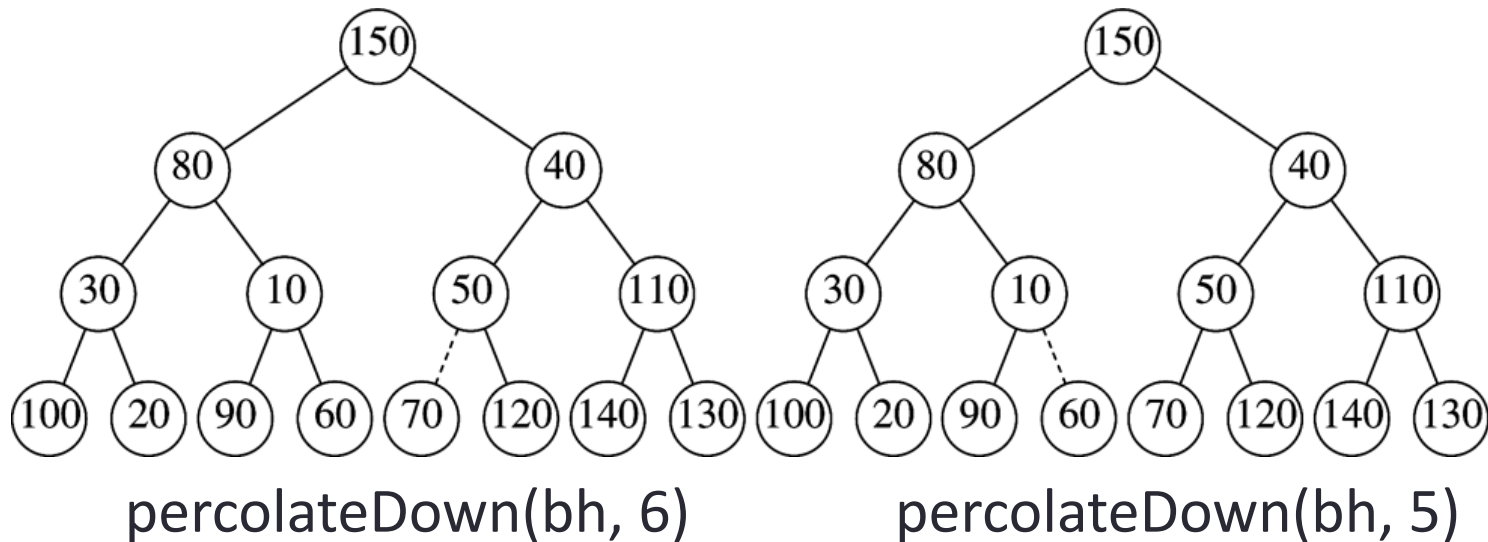
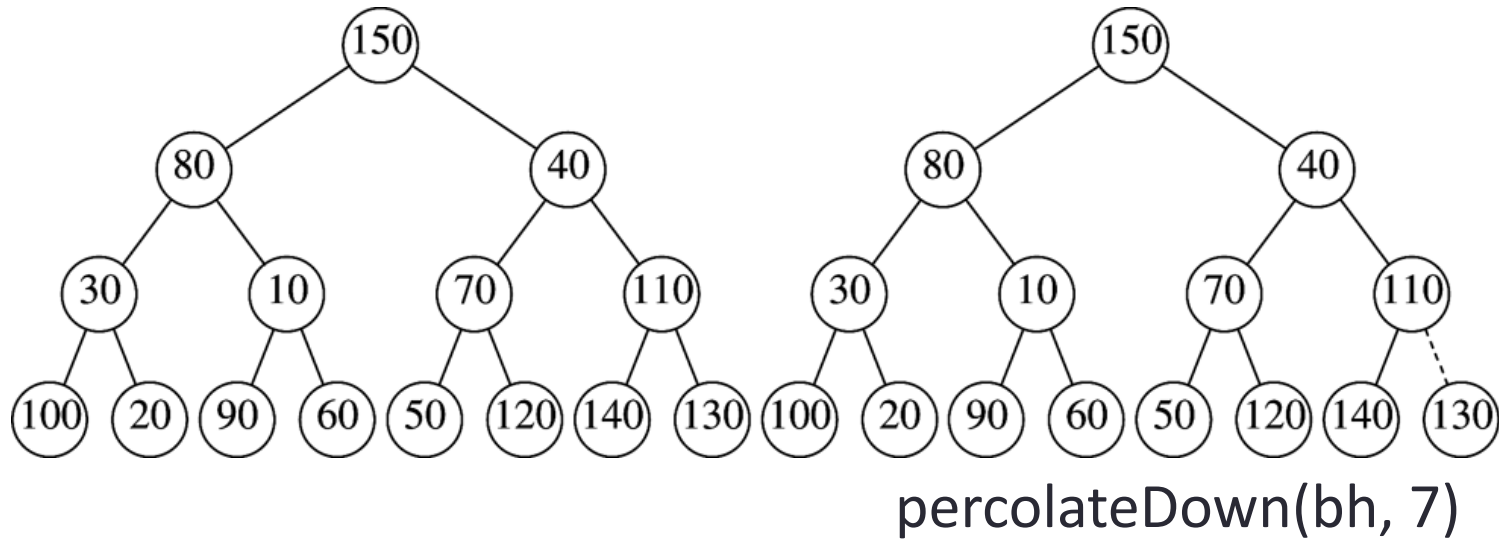
- Xây dựng đồng từ một tập phần tử đã có:

```
void init(BinaryHeap & bh, vector<T> & elems);
```
- Các bước:
 - Chèn lần lượt các phần tử vào cây mà không quan tâm tính chất thứ tự đồng.
 - Duyệt qua các nút từ dưới lên trên, từ phải sang trái:
 - + Tại mỗi nút, dùng phép thẩm thấu xuôi để nút đó thỏa mãn tính chất thứ tự đồng.
- Thời gian chạy là $O(n)$ (sẽ phân tích sau).

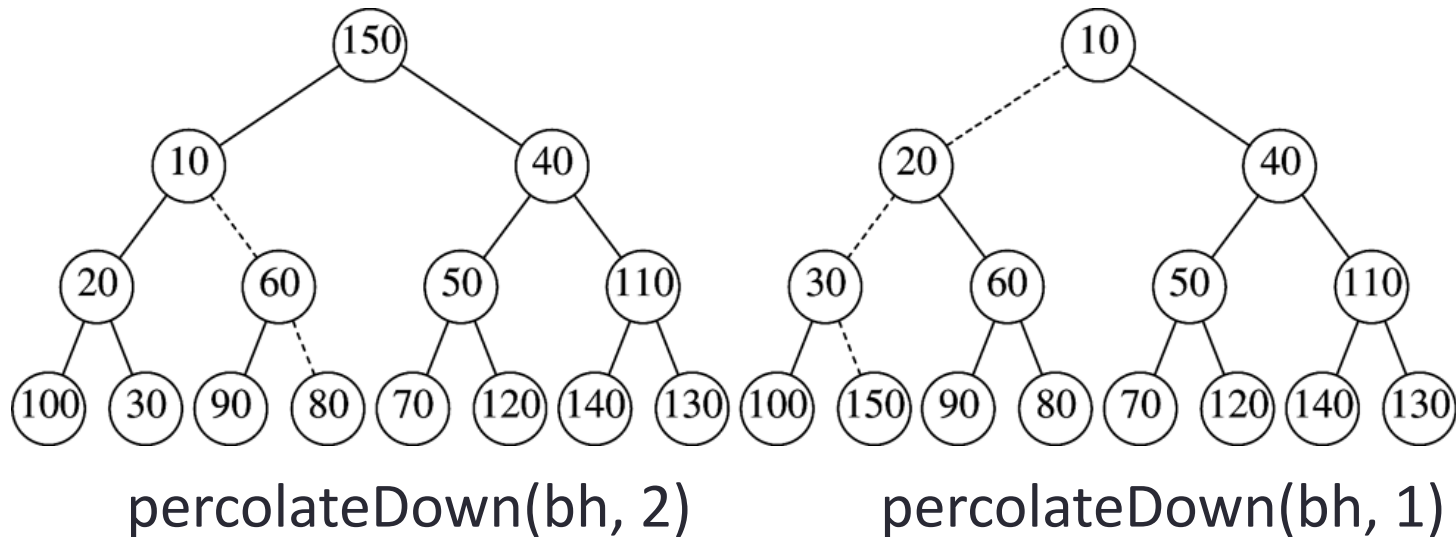
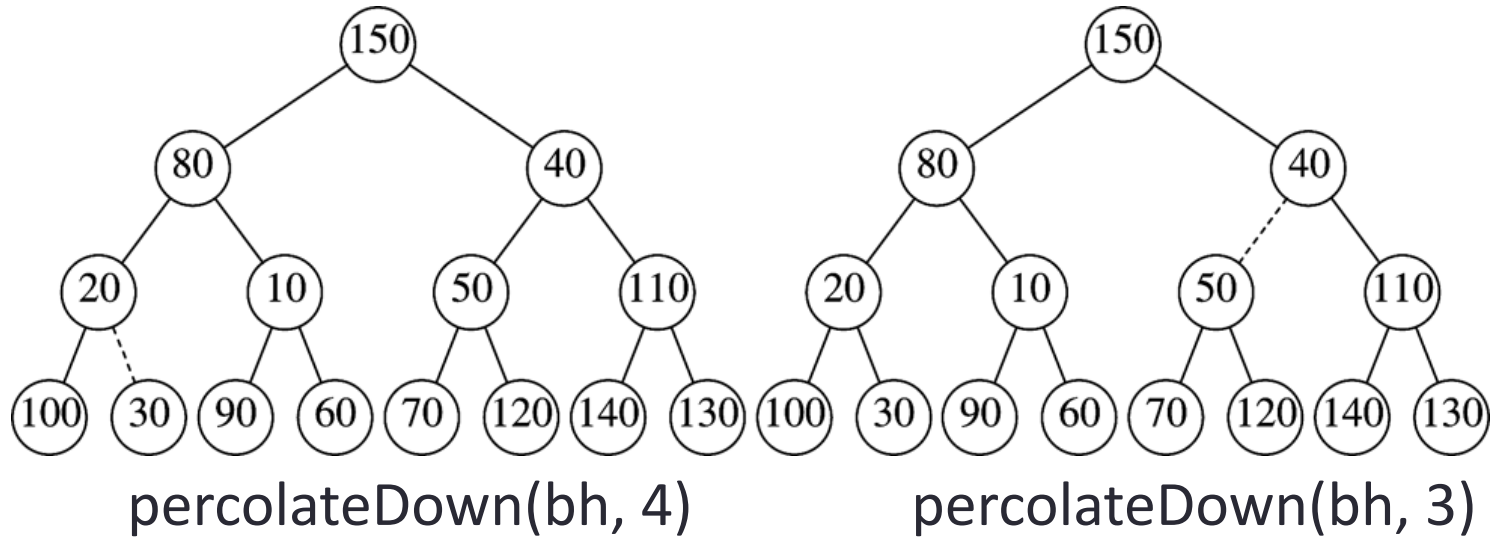
Xây dựng đống: Lập trình

```
void init(BinaryHeap & bh, vector<T> & elems) {  
    bh.array.resize(elems.size() + 10);  
    bh.currentSize = elems.size();  
    for (int i = 0; i < elems.size(); i++) // O(n)  
        bh.array[i + 1] = elems[i];  
  
    buildHead(); // O(n), sẽ phân tích sau  
}  
  
// Thiết lập tính chất thứ tự đống  
void buildHead(BinaryHeap & bh) {  
    for (int i = bh.currentSize / 2; i > 0; i--)  
        percolateDown(bh, i);  
}
```

Xây dựng đống: Ví dụ



Xây dựng đồng: Ví dụ (tiếp)



Thời gian chạy của buildHeap

- Thời gian chạy của buildHead tỉ lệ với tổng số đoạn đứt nét \leq tổng chiều cao S của tất cả các nút \leq tổng chiều cao S_2 của tất cả các nút trên cây nhị phân hoàn hảo tương ứng (mức dưới cùng cũng được lấp đầy).

- Cộng chiều cao của các nút theo từng mức trên cây:

$$S_2 = h + 2(h - 1) + 2^2(h - 2) + \dots + 2^{h-1}$$

- Nhân hai vế với 2:

$$2S_2 = 2h + 2^2(h - 1) + 2^3(h - 2) + \dots + 2^h$$

- Trừ hai đẳng thức trên theo từng vế:

$$\begin{aligned} S_2 &= -h + 2 + 2^2 + 2^3 + \dots + 2^{h-1} + 2^h = -h + 2 \times \frac{1 - 2^h}{1 - 2} \\ &= 2 \times 2^h - h - 2 < 2 \times 2^h \end{aligned}$$

- Ta đã biết tổng số nút $n \geq 2^h \Rightarrow S_2 < 2n \Rightarrow$ Thời gian chạy của buildHeap là $O(n)$.

Một ứng dụng của hàng đợi ưu tiên

- Bài toán: Tìm số nhỏ nhất thứ k trong n phần tử đã cho.
- Cách giải 1: thời gian $O(n^2)$
 - Sắp xếp n phần tử bằng một thuật toán đơn giản (VD: sắp xếp chọn), mất thời gian $O(n^2)$.
 - Trả về số thứ k .
- Cách giải 2: thời gian $O(n + k \log n)$
 - Tạo đống từ n phần tử đã cho $\rightarrow O(n)$
 - Thực hiện thao tác deleteMin $k - 1$ lần liên tiếp $\rightarrow O(k \log n)$
 - Thực hiện thao tác findMin $\rightarrow O(1)$

Bài tập

Bài tập 1:

a. Xét một đồng đang rỗng. Hãy chèn (insert) lần lượt vào đồng các giá trị sau đây:

{ 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, 2 }

b. Xây dựng đồng dùng hàm khởi tạo từ danh sách giá trị cho trong câu a.

Bài tập 2:

Thực hiện thao tác xóa (deleteMin) 3 lần liên tiếp trên các đồng thu được trong bài tập 1.