CHƯƠNG 3. CÁC THUẬT TOÁN TÌM KIẾM & SẮP XẾP

SẮP XẾP VUN ĐỐNG (HEAPSORT)

ThS. Nguyễn Chí Hiếu

2021

NỘI DUNG

NỘI DUNG

Các khái niệm cơ bản

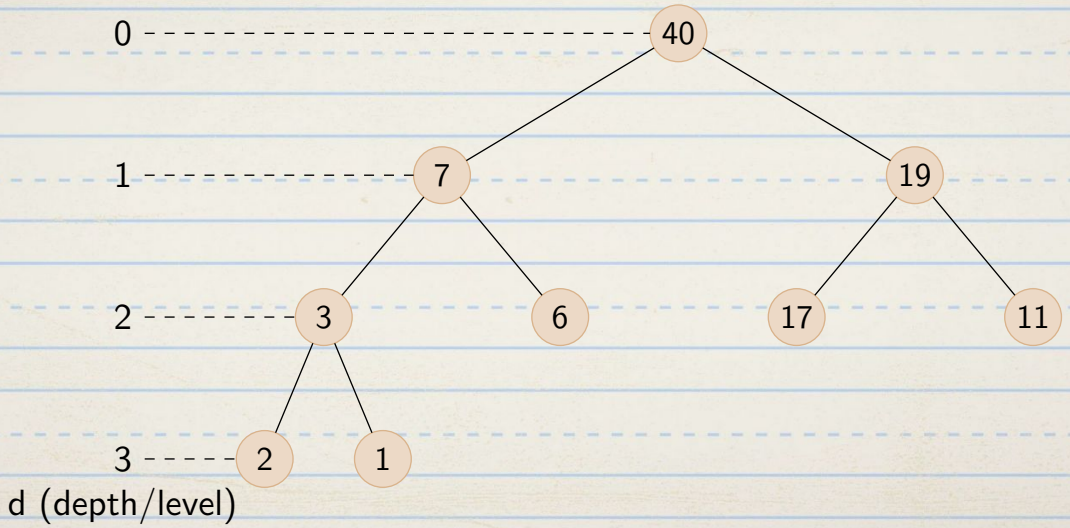
Định nghĩa

Heap là một cây nhị phân đầy đủ thỏa các điều kiện sau:

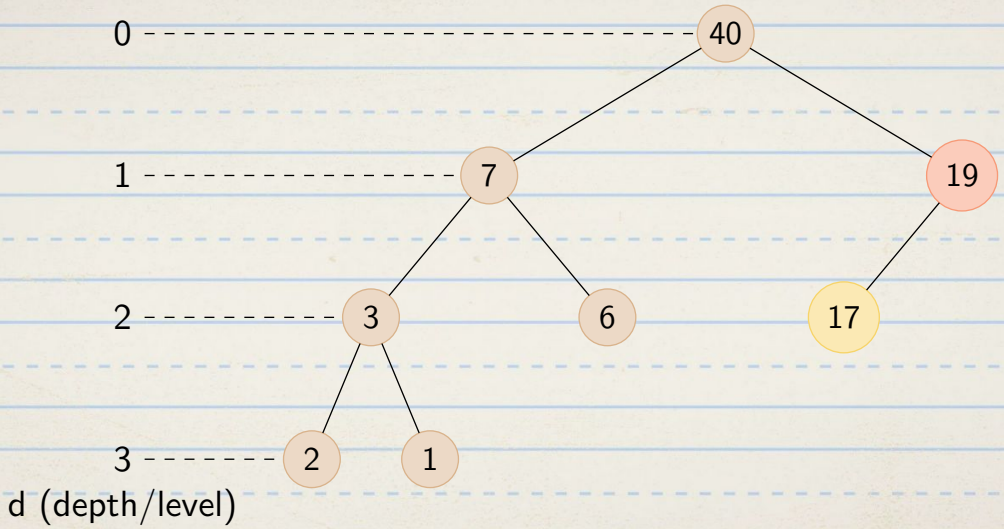
- Các nút từ mức 0 đến $d-1$ đều có đủ số lượng nút *(với d là mức của cây)*.
- Các nút ở mức d được thêm vào từ trái sang phải.
- Giá trị của một nút luôn lớn hơn hay bằng giá trị các nút con của nó (max-heap).
  Xét phần tử a_i
  - $a_i \geq a_{2i+1}$ và $a_i \geq a_{2i+2}$
  - Cặp phần tử liên đới là a_{2i+1} và a_{2i+2}.
  - Trường hợp đặt $j = 2 \cdot i + 1$ thì cặp phần tử liên đới là a_j và a_{j+1}.

Max-Heap

Ví dụ 1

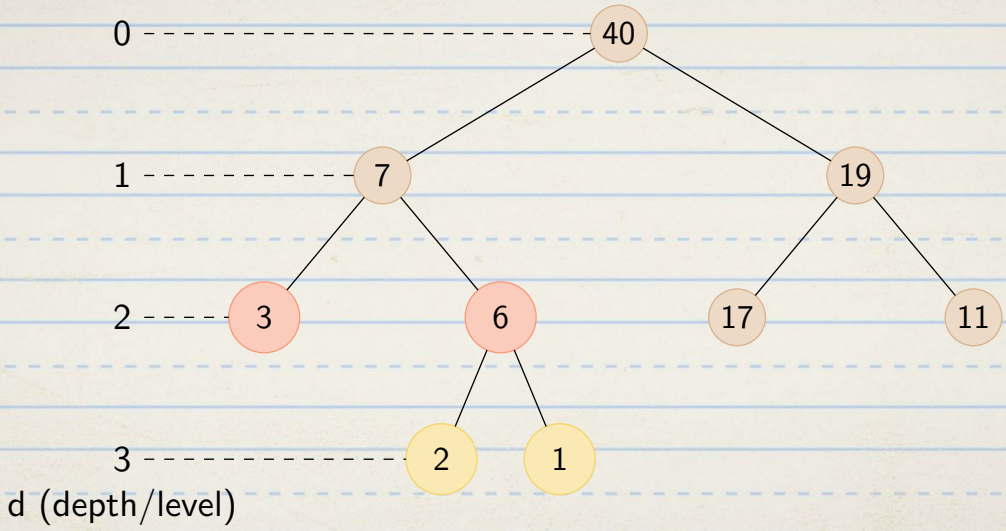


Max-Heap



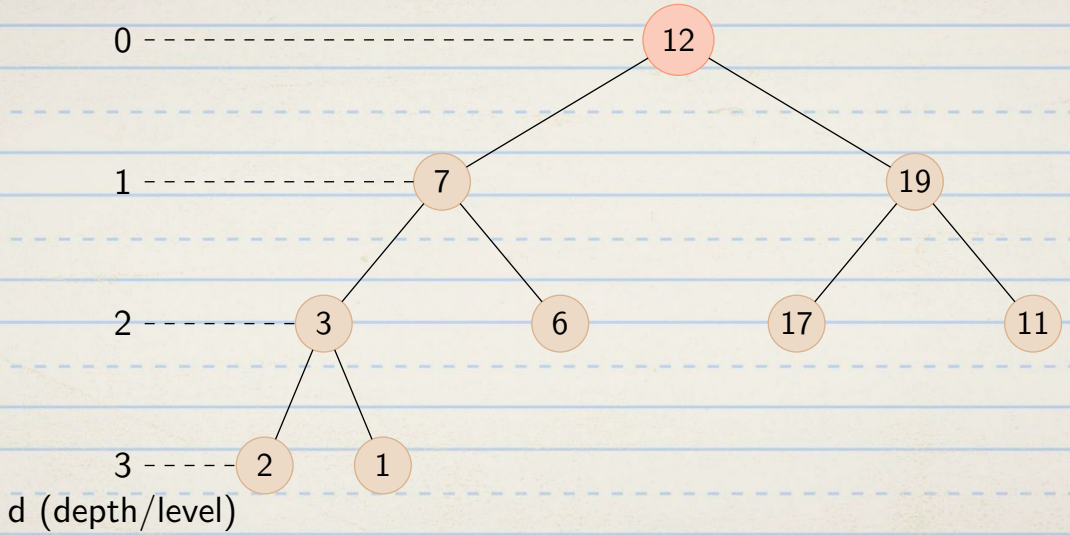
Tại mức $d-1$, số lượng nút không đảm bảo.

Max-Heap



Tại mức d, các nút phải điền từ trái sang phải.

Max-Heap



Nút cha $\geq$ nút con.

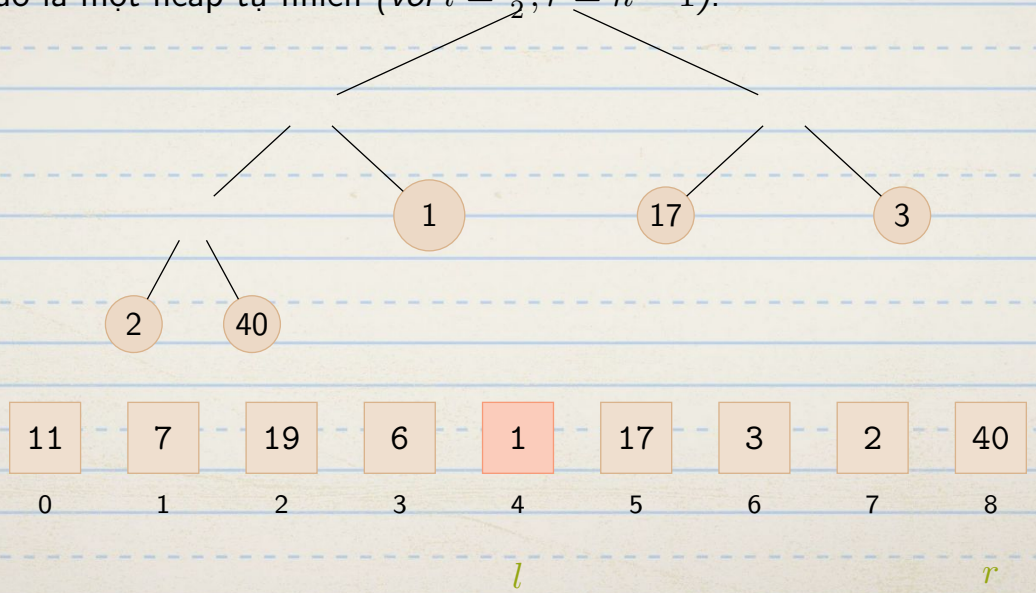
NỘI DUNG

Một số tính chất của Heap

- Nếu $a_0, a_1, ..., a_r$ là một heap thì sau khi bỏ một số phần tử ở hai đầu, dãy còn lại vẫn là một heap.
- Nếu $a_0, a_1, ..., a_r$ là một heap thì phần tử a_0 có giá trị lớn nhất/nhỏ nhất *(trường hợp max-heap/min-heap)*.

Một số tính chất của Heap

- Cho dãy a gồm n phần tử. Nếu dãy $a_l, a_{l+1}, ..., a_r$ thỏa điều kiện $2l + 1 > r$ thì dãy đó là một heap tự nhiên *(với $l = \frac{n}{2}, r = n - 1$)*.



11	7	19	6	1	17	3	2	40
0	1	2	3	4	5	6	7	8

l r

NỘI DUNG

Giới thiệu HeapSort

Ý tưởng

- Dựa vào cấu trúc heap *(đống)*, một max-heap *(đống cực đại)* thì phần tử ở đỉnh luôn có giá trị lớn nhất.
- HeapSort *(sắp xếp vun đống)* thực hiện qua hai giai đoạn:
  - Giai đoạn 1. Tạo (vun đống) và hiệu chỉnh dãy ban đầu thành heap.
  - Giai đoạn 2. Sắp xếp dãy dựa trên heap: thực hiện một vòng lặp
    - Lấy phần tử ở đỉnh của heap.
    - Hoán vị với phần tử cuối cùng của dãy số.
    - Hiệu chỉnh lại heap.

Tạo và hiệu chỉnh heap

- Theo tính chất 3, mọi dãy $a_l, a_{l+1}, ..., a_r$ thỏa điều kiện $2l + 1 > r$ là một heap tự nhiên (với $l = \frac{n}{2}, r = n - 1$).
- Do đó, giai đoạn tạo heap chỉ cần lần lượt ghép thêm các phần tử $a_{l-1}, a_{l-2}, \cdots a_0$ vào heap tự nhiên này.
- Mỗi lần ghép thêm một phần tử thực hiện thao tác hiệu chỉnh *(Heapify/Sift Down)* lại heap.

Thuật toán 1: `MakeHeap(a[], n)`

- Đầu vào: mảng a gồm n phần tử.
- Đầu ra: mảng a là một max-heap.

```
1 l ← n / 2
2 l ← l - 1
3 while l ≥ 0
4 __Heapify(a, l, n - 1)
5 __l ← l - 1
```

Tạo và hiệu chỉnh heap

Thuật toán 2: Heapify(a[], l, r)

- Đầu vào: mảng a và mảng con cần hiệu chỉnh $a_l, a_{l+1} \cdots a_r$.
- Đầu ra: mảng con đã hiệu chỉnh thành max-heap.

```
1  i ← l
2  j ← 2 * i + 1
3  x ← a[i]
4  while j <= r
5      if j < r and a[j] < a[j + 1]
6          j ← j + 1
7      if a[j] < x
8          return
9      else
10         a[i] ← a[j]
11         a[j] ← x
12         i ← j
13         j ← 2 * i + 1
```


Tạo và hiệu chỉnh heap

Giải thích

- Dòng 2: hai phần tử liên đới của a_i là a_j và a_{j+1} .
- Dòng 4 $\rightarrow$ 13: nếu a_i có phần tử liên đới thì thực hiện:
  - Dòng 5, 6: tìm phần tử liên đới có giá trị lớn nhất.
  - Dòng 7, 8: nếu thỏa quan hệ liên đới. Kết thúc thao tác hiệu chỉnh.
  - Dòng 10, 11: ngược lại, hoán vị phần tử a_i với phần tử liên đới có giá trị lớn nhất.
  - Dòng 12, 13: xét hiệu chỉnh lan truyền sau khi thực hiện hoán vị.

Sắp xếp heap

Thuật toán 3: `HeapSort(a[], n)`

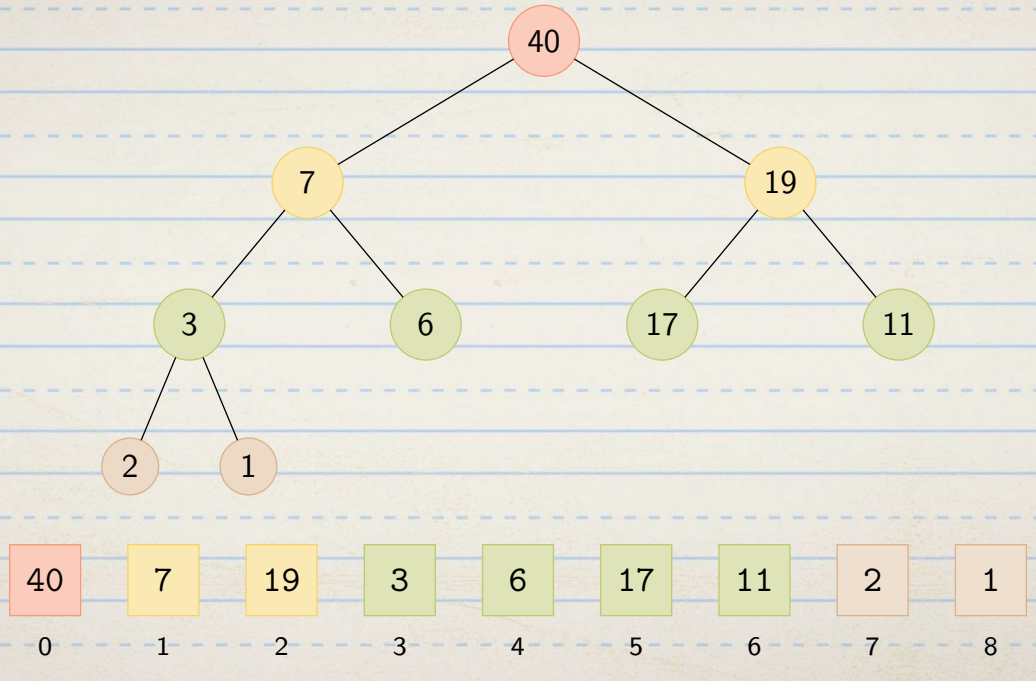
- Đầu vào: mảng a gồm n phần tử.
- Đầu ra: mảng a có thứ tự tăng dần.

```
1  MakeHeap(a, n)
2  r ← n - 1
3  while r > 0
4      Swap(a[0], a[r])
5      r ← r - 1
6      Heapify(a, 0, r)
```

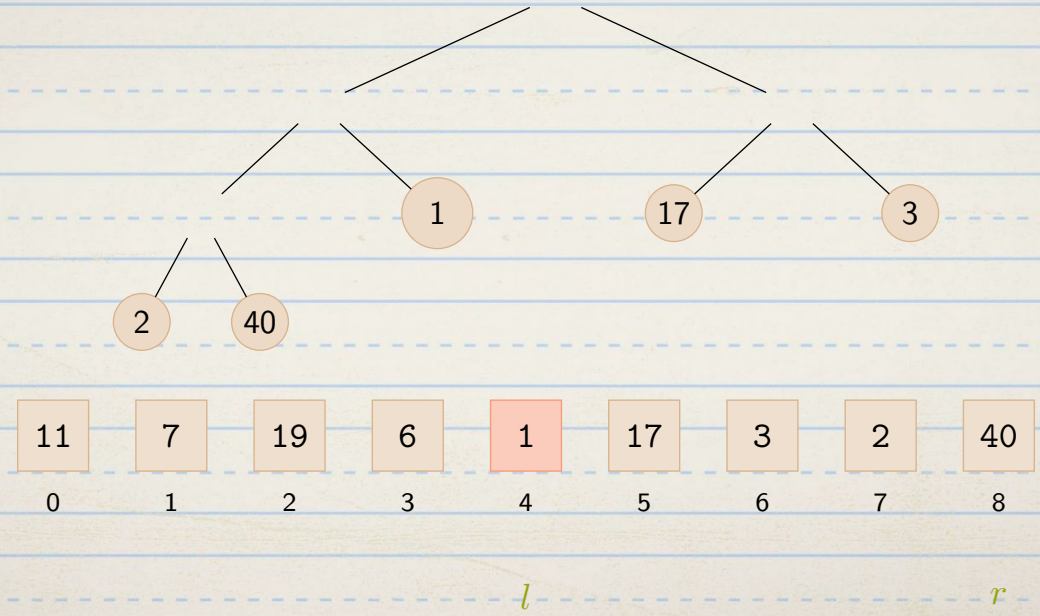
Giải thích

- Dòng 1: gọi hàm tạo max-heap từ mảng ban đầu.
- Dòng 3 $\rightarrow$ 6: mỗi lần lặp, hoán vị phần tử đầu và cuối mảng. Sau đó, gọi hàm hiệu chỉnh mảng tử $a_0, a_1, ..., a_{r-1}$.

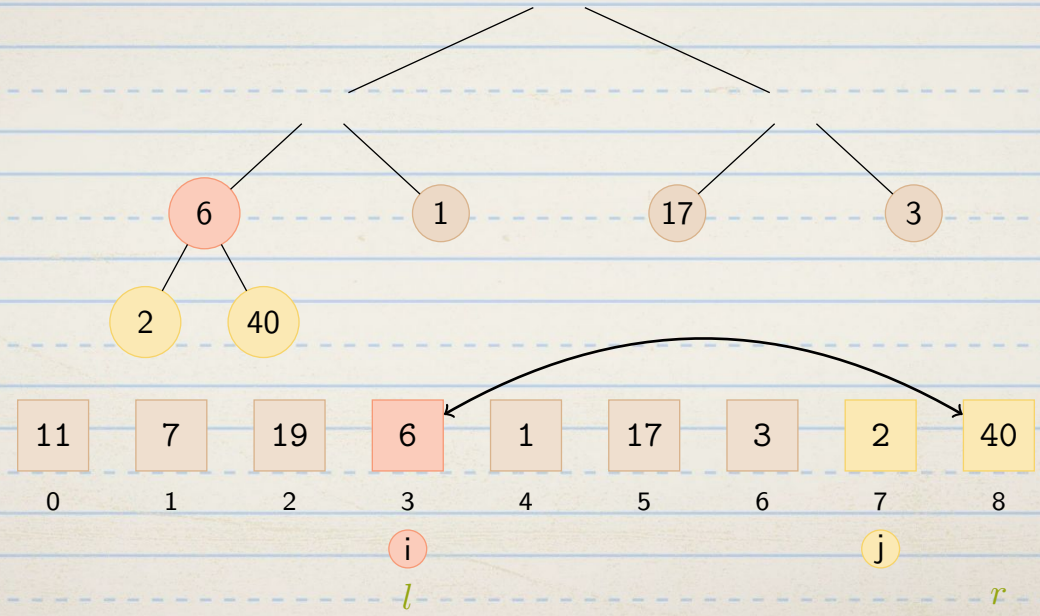
Biểu diễn heap bằng cây nhị phân đầy đủ



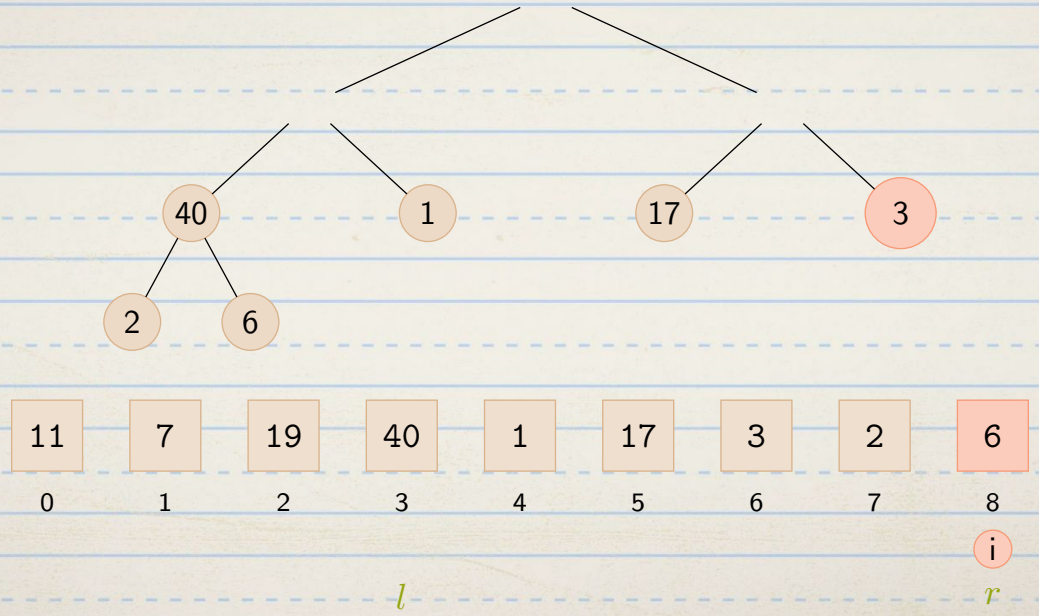
Tạo và hiệu chỉnh dãy thành max-heap



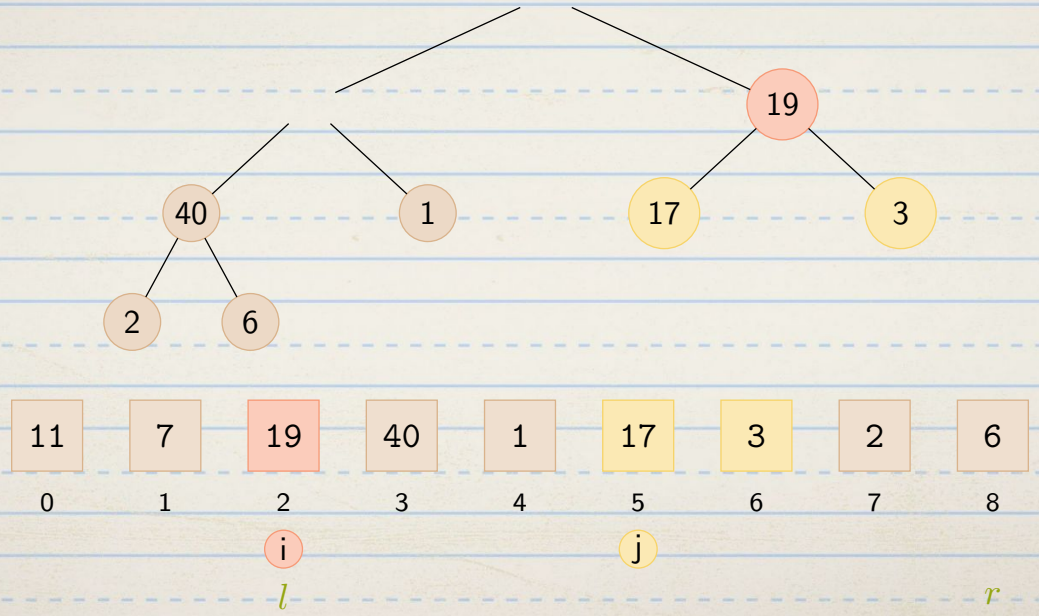
Tạo và hiệu chỉnh dãy thành max-heap



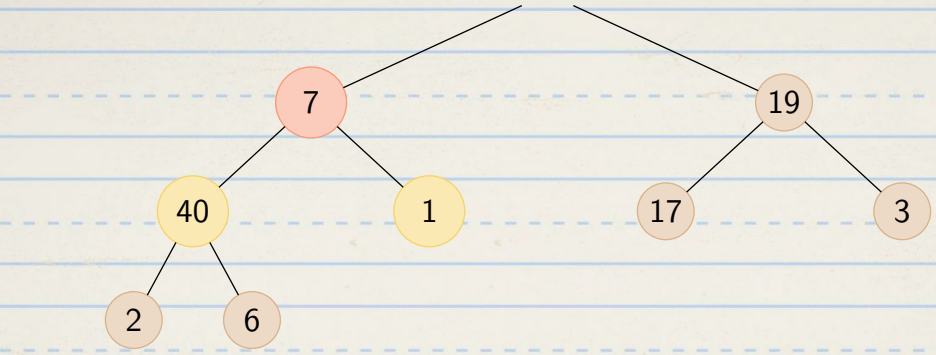
Tạo và hiệu chỉnh dãy thành max-heap

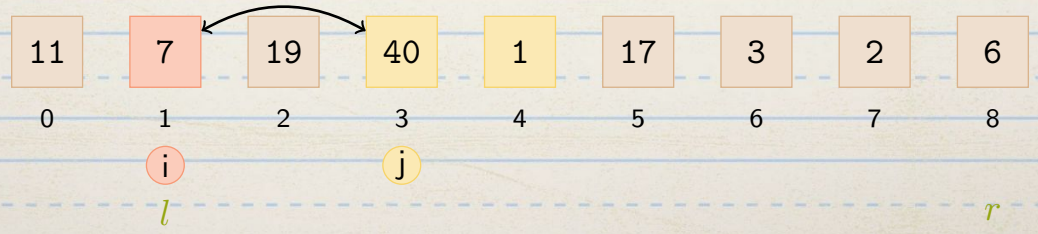


Tạo và hiệu chỉnh dãy thành max-heap

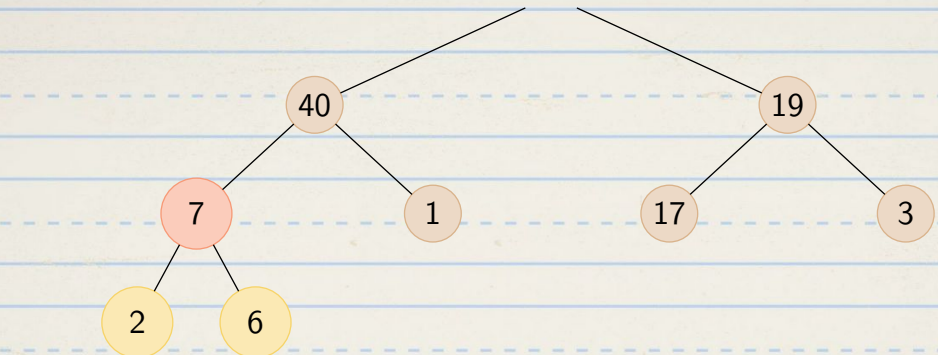


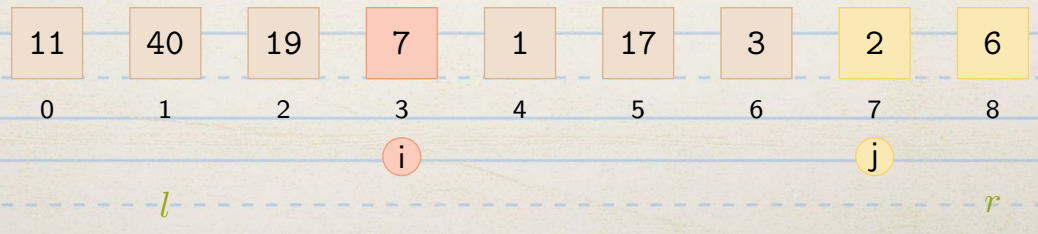
Tạo và hiệu chỉnh dãy thành max-heap



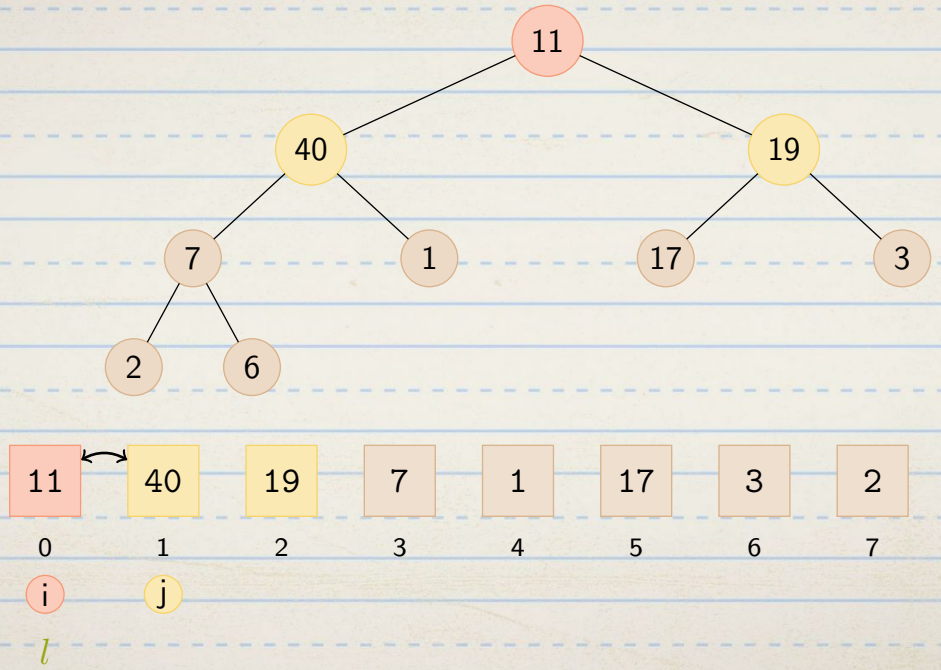


Tạo và hiệu chỉnh dãy thành max-heap

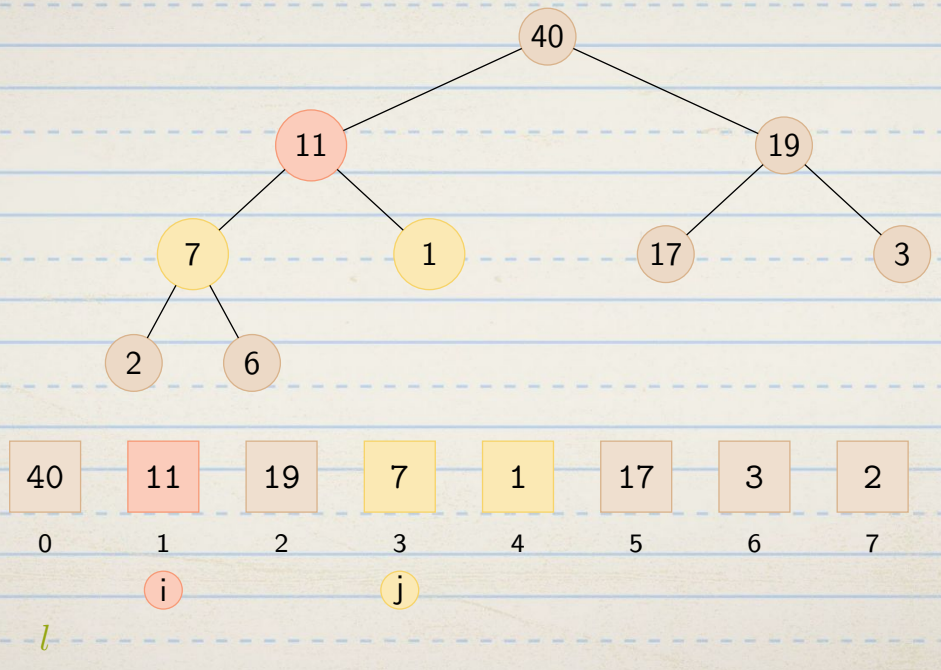




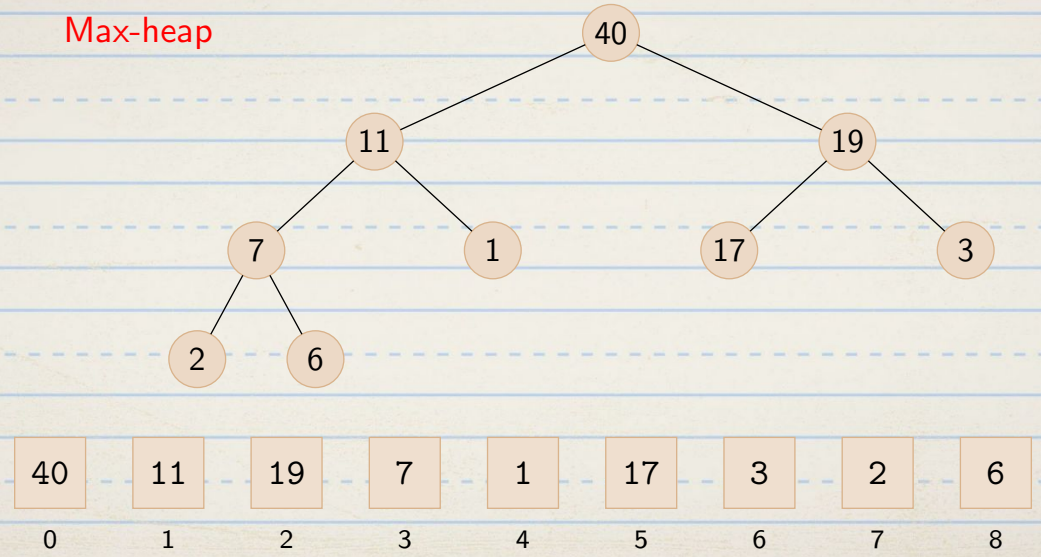
Tạo và hiệu chỉnh dãy thành max-heap



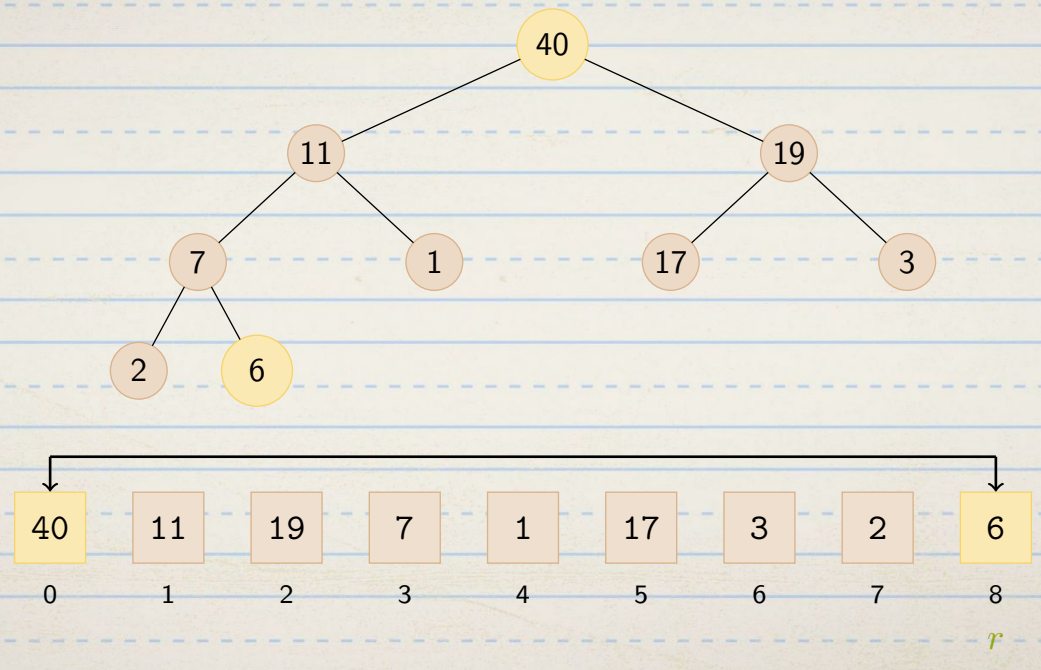
Tạo và hiệu chỉnh dãy thành max-heap



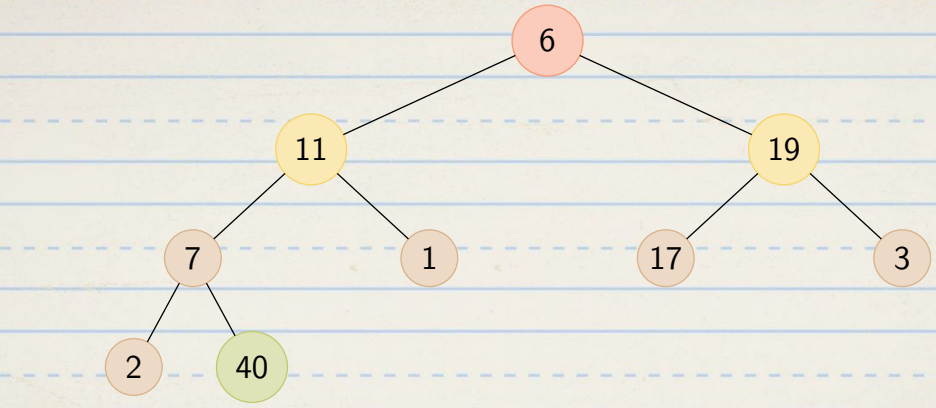
Tạo và hiệu chỉnh dãy thành max-heap

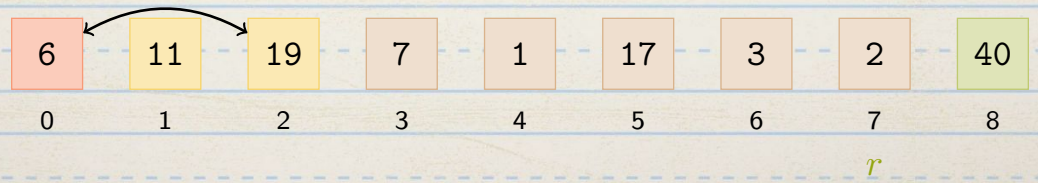


Sắp xếp dãy số dựa trên max-heap

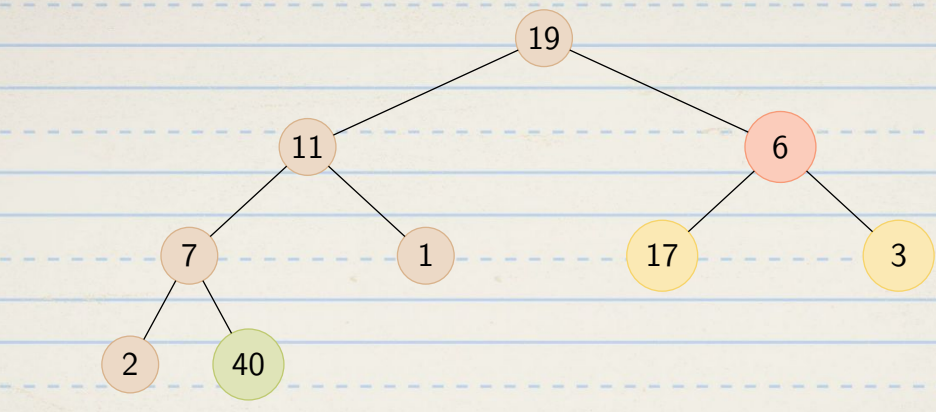


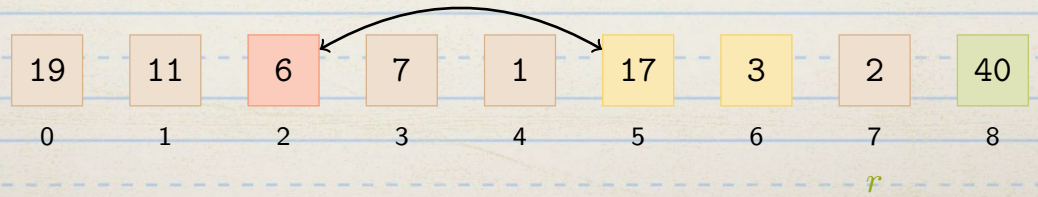
Sắp xếp dãy số dựa trên max-heap



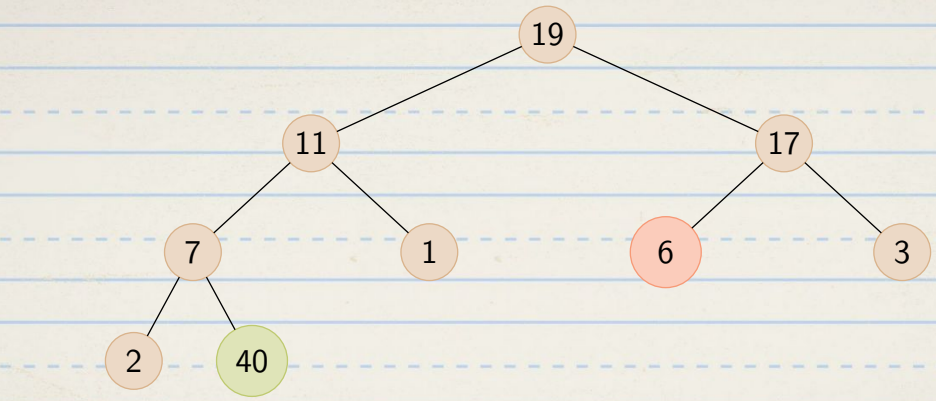


Sắp xếp dãy số dựa trên max-heap





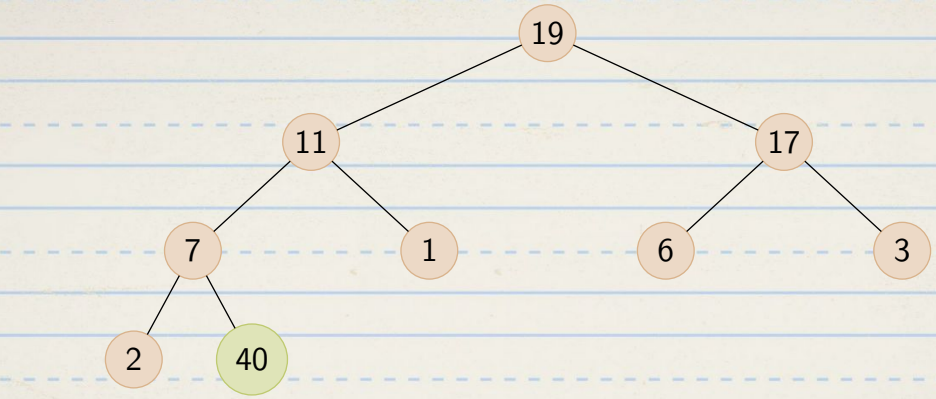
Sắp xếp dãy số dựa trên max-heap



19	11	17	7	1	6	3	2	40
0	1	2	3	4	5	6	7	8

r

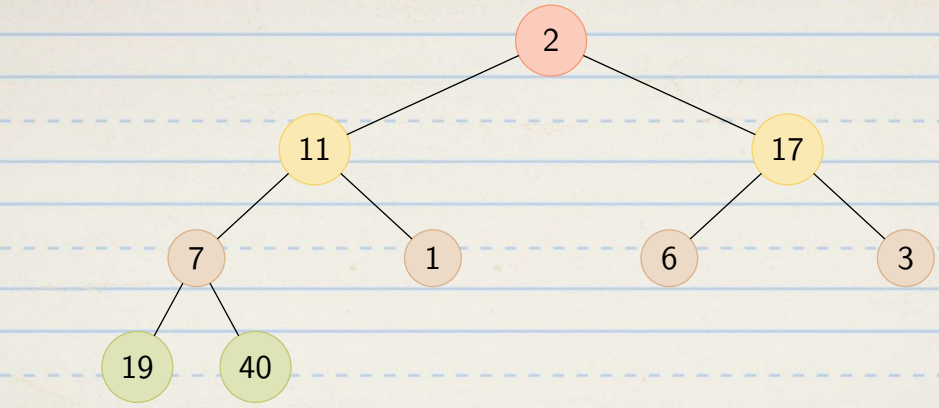
Sắp xếp dãy số dựa trên max-heap

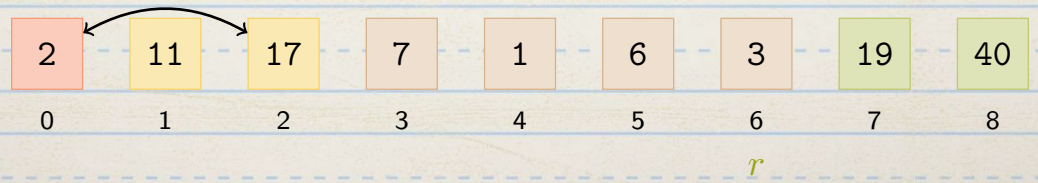


19	11	17	7	1	6	3	2	40
0	1	2	3	4	5	6	7	8

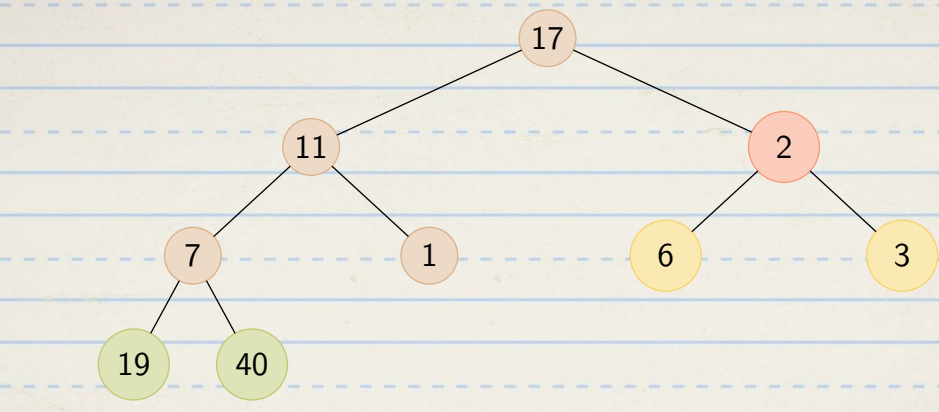
r

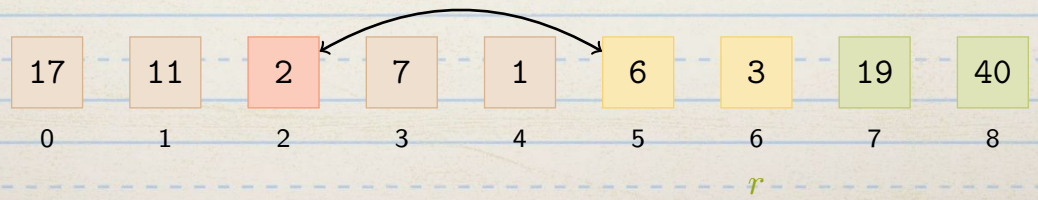
Sắp xếp dãy số dựa trên max-heap



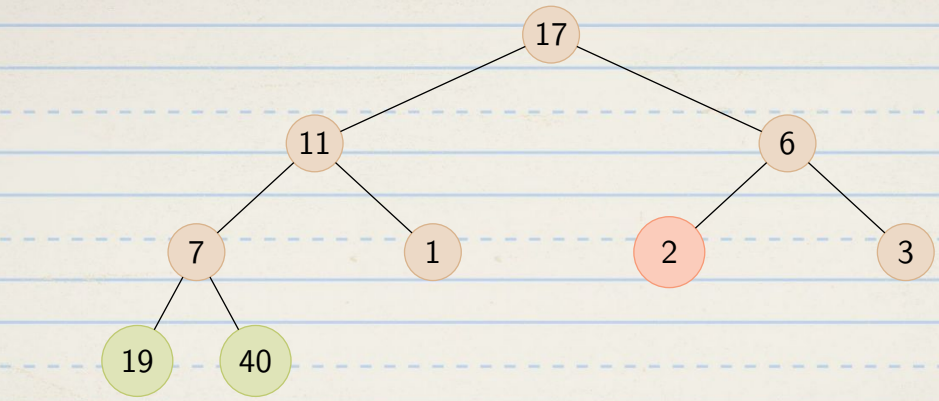


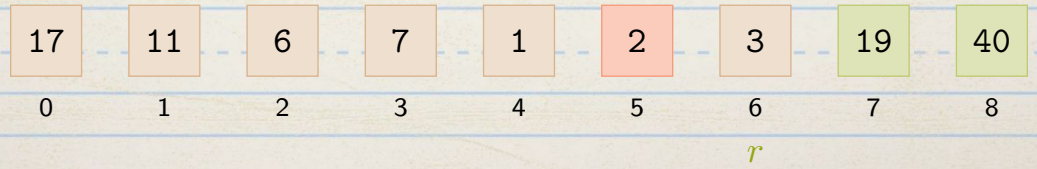
Sắp xếp dãy số dựa trên max-heap



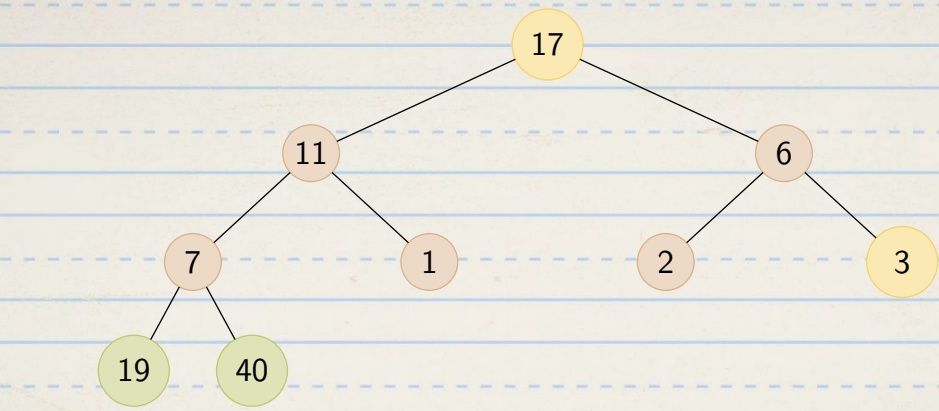


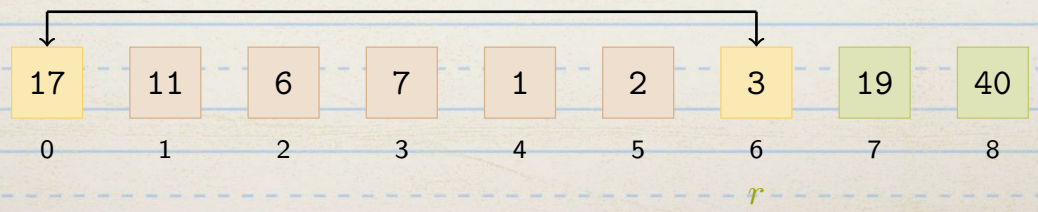
Sắp xếp dãy số dựa trên max-heap



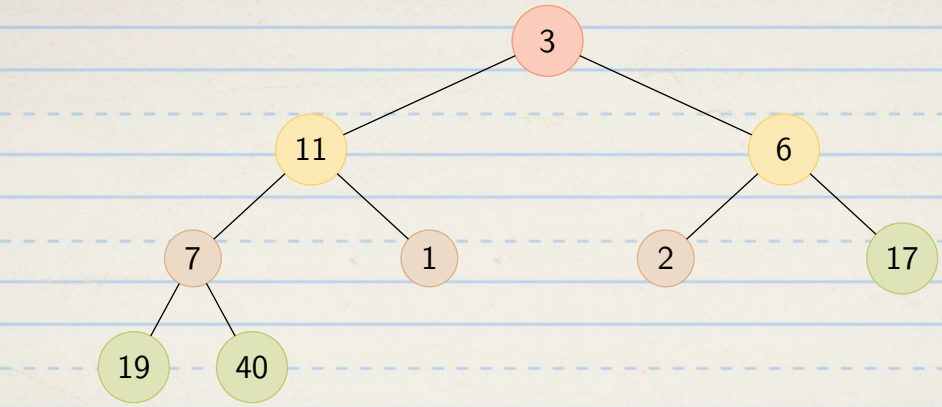


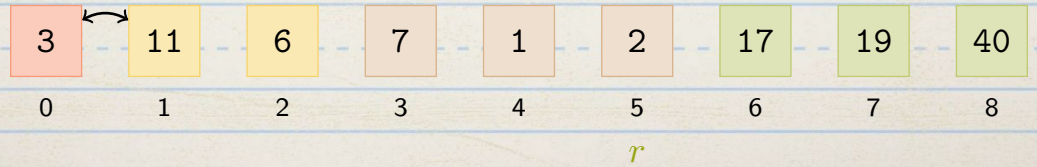
Sắp xếp dãy số dựa trên max-heap



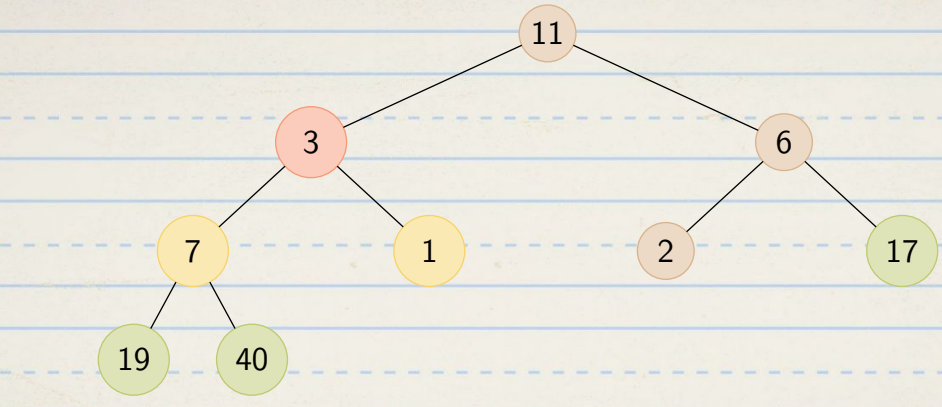


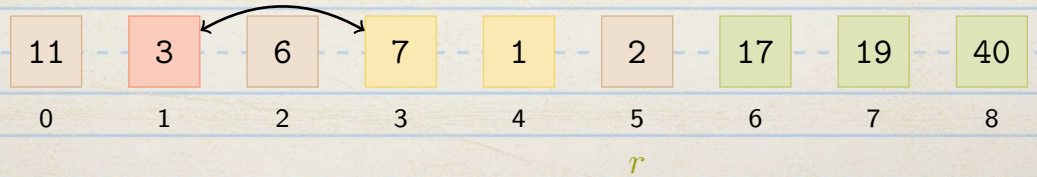
Sắp xếp dãy số dựa trên max-heap



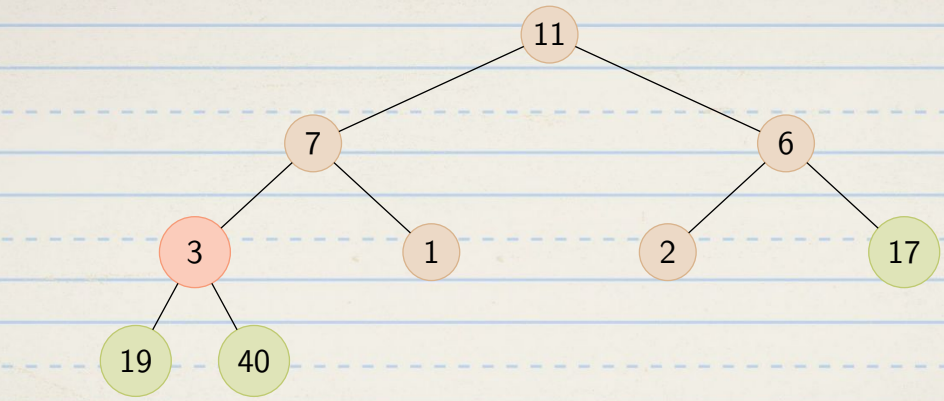


Sắp xếp dãy số dựa trên max-heap





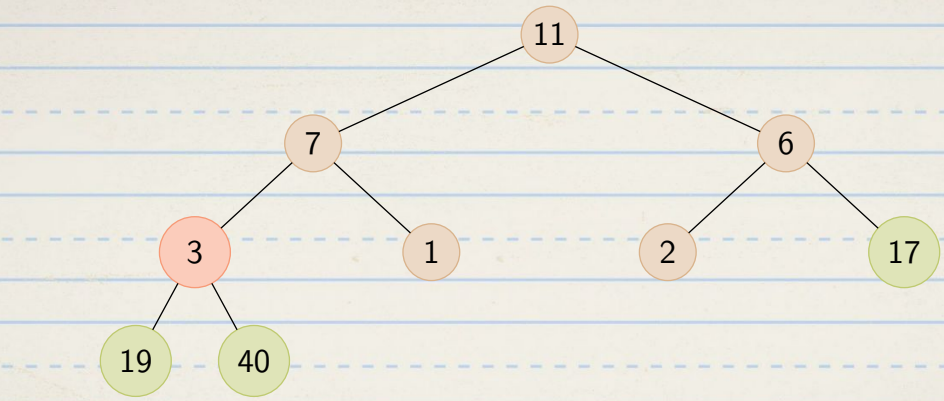
Sắp xếp dãy số dựa trên max-heap



11	7	6	3	1	2	17	19	40
0	1	2	3	4	5	6	7	8

r

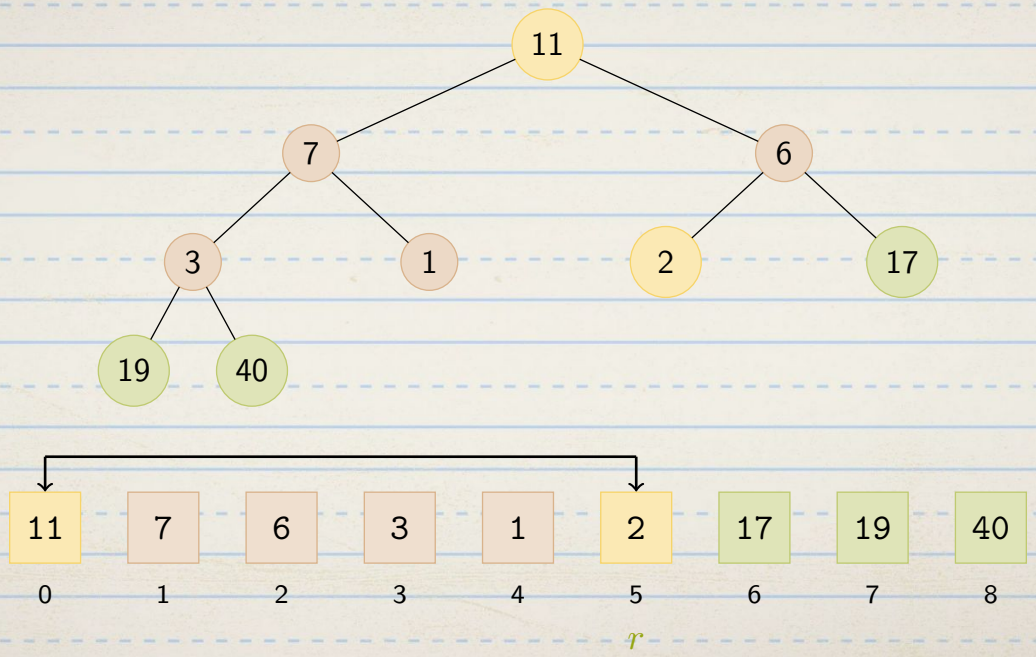
Sắp xếp dãy số dựa trên max-heap



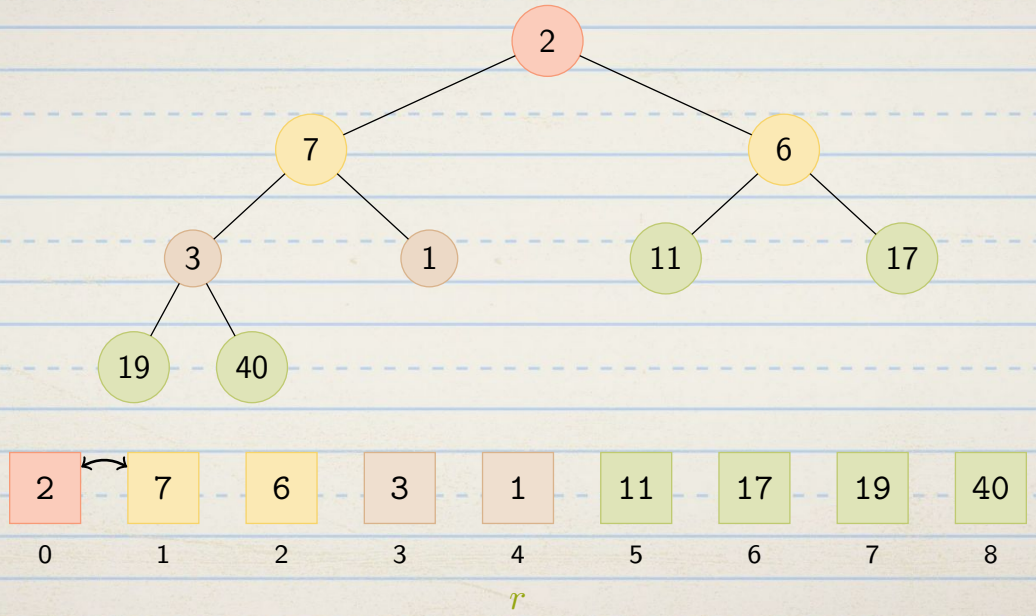
11	7	6	3	1	2	17	19	40
0	1	2	3	4	5	6	7	8

r

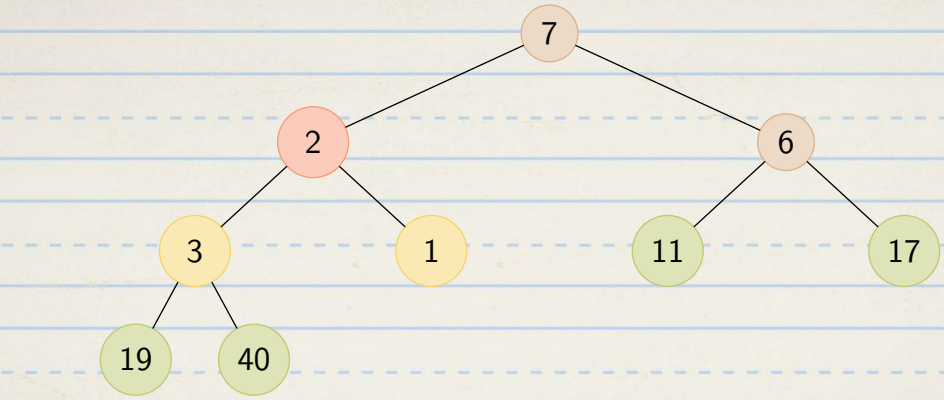
Sắp xếp dãy số dựa trên max-heap

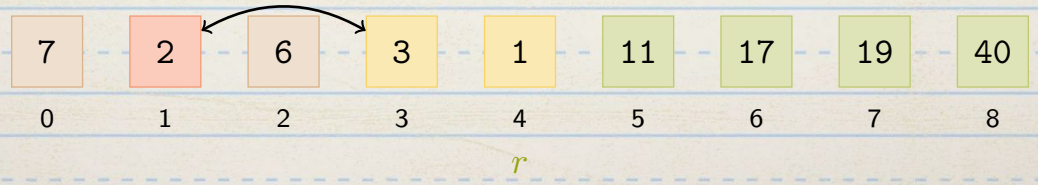


Sắp xếp dãy số dựa trên max-heap

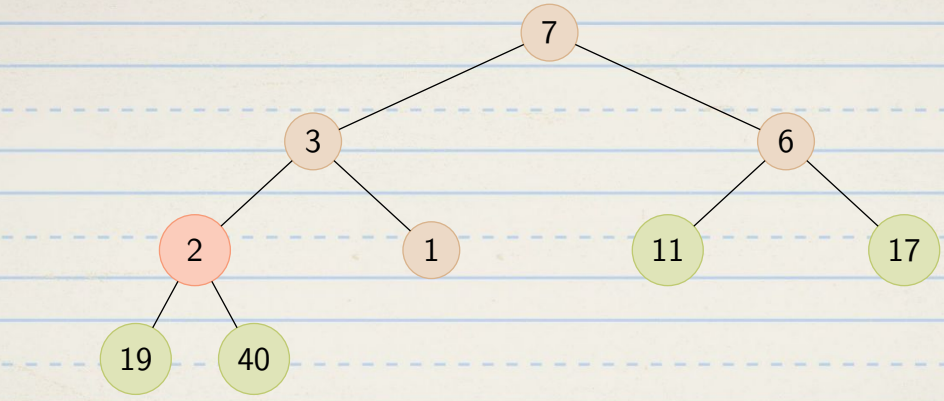


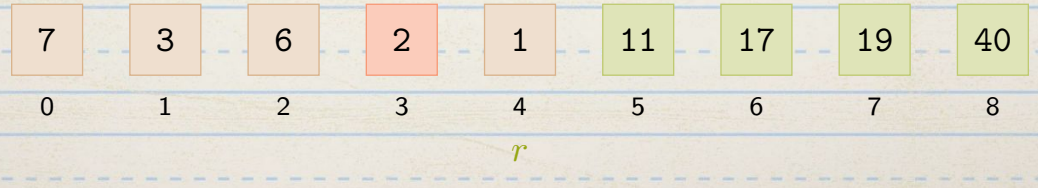
Sắp xếp dãy số dựa trên max-heap



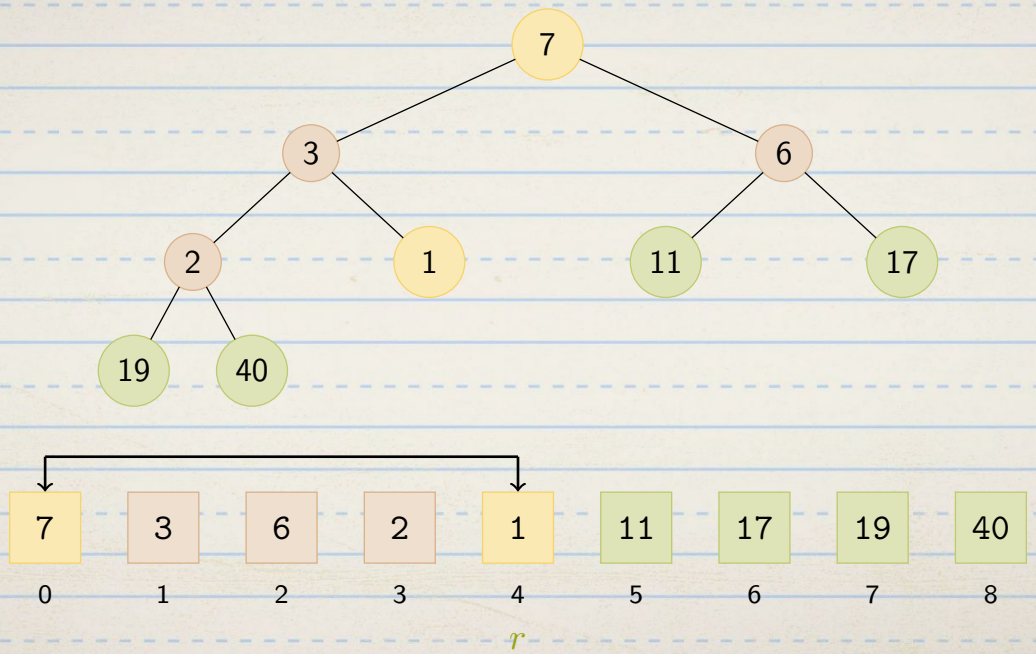


Sắp xếp dãy số dựa trên max-heap

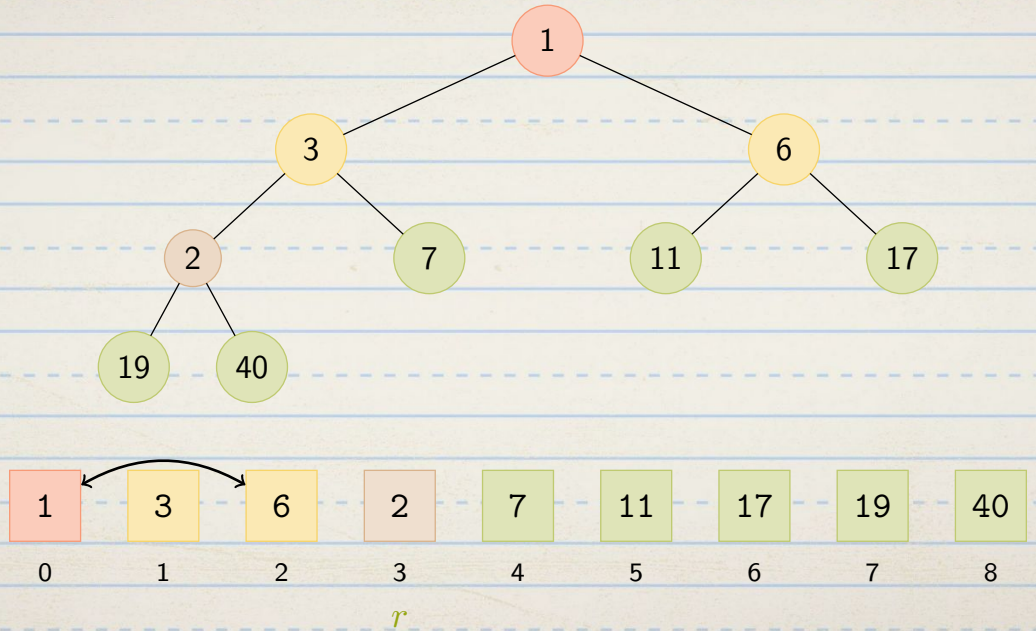




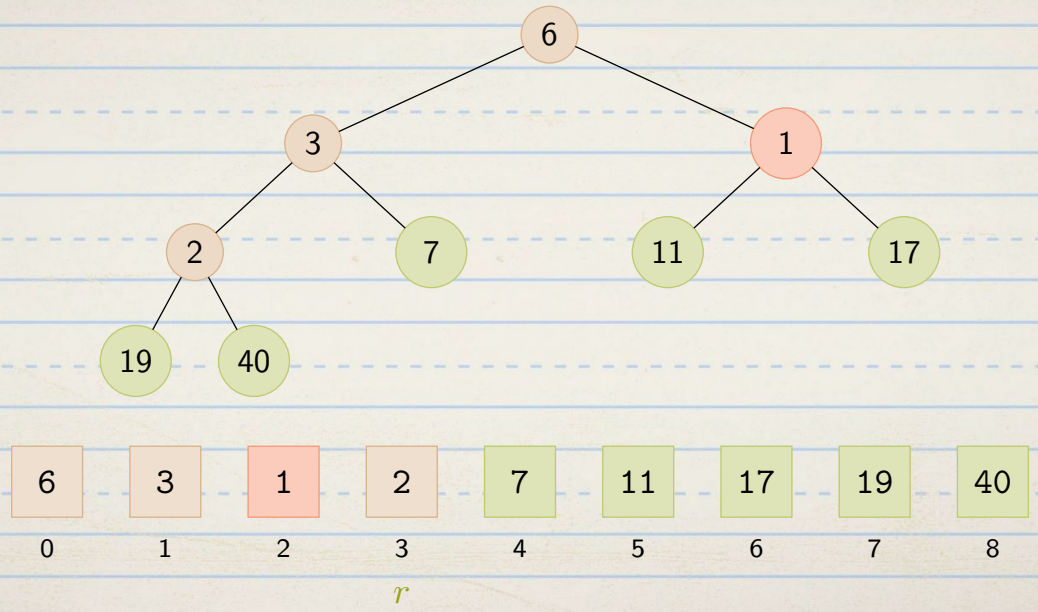
Sắp xếp dãy số dựa trên max-heap



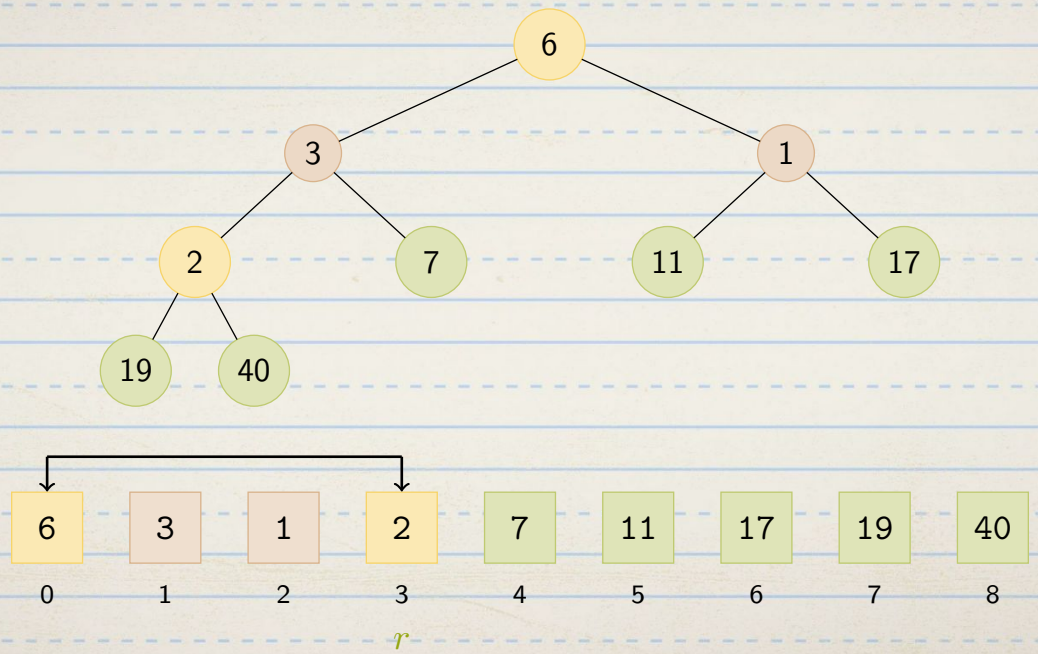
Sắp xếp dãy số dựa trên max-heap



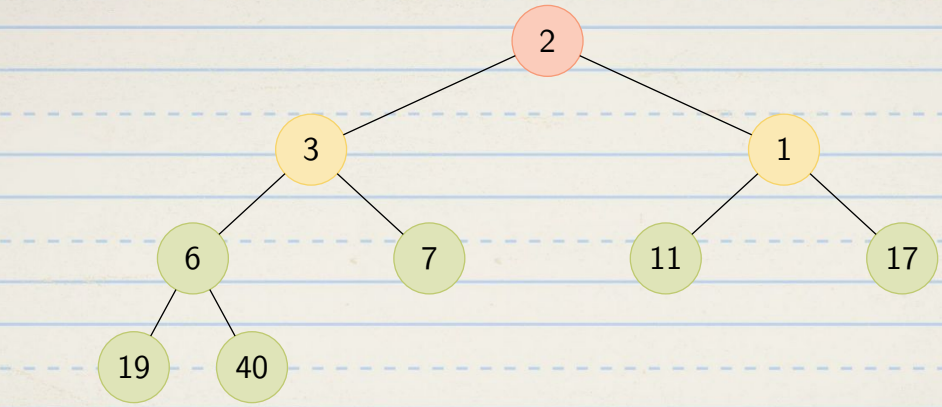
Sắp xếp dãy số dựa trên max-heap

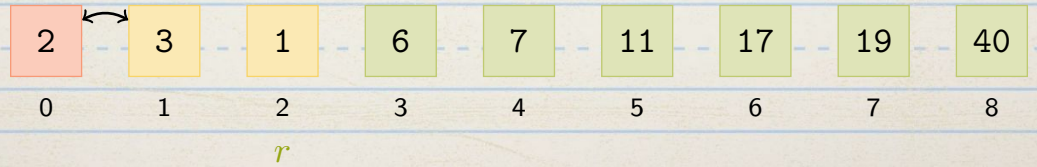


Sắp xếp dãy số dựa trên max-heap

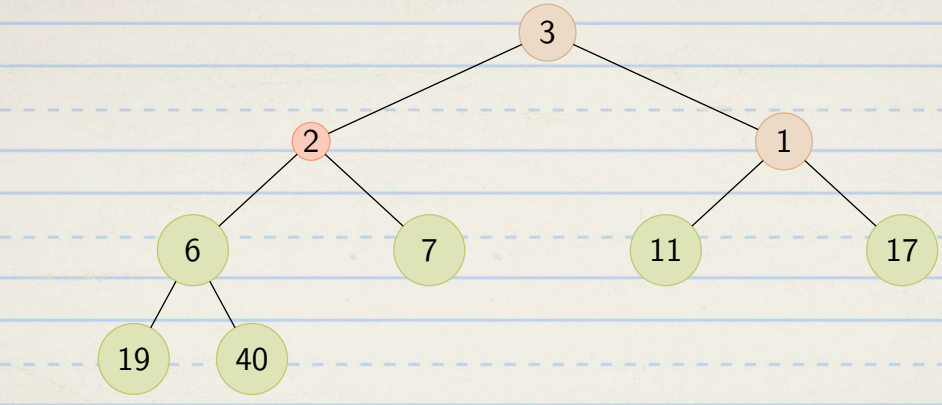


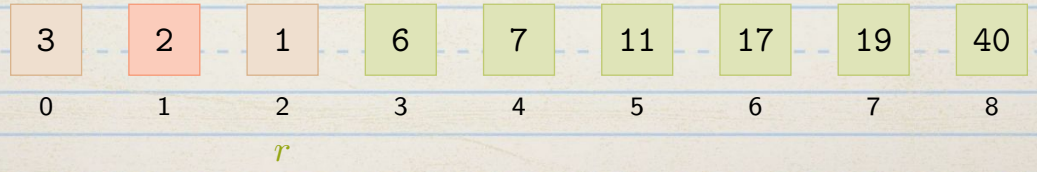
Sắp xếp dãy số dựa trên max-heap



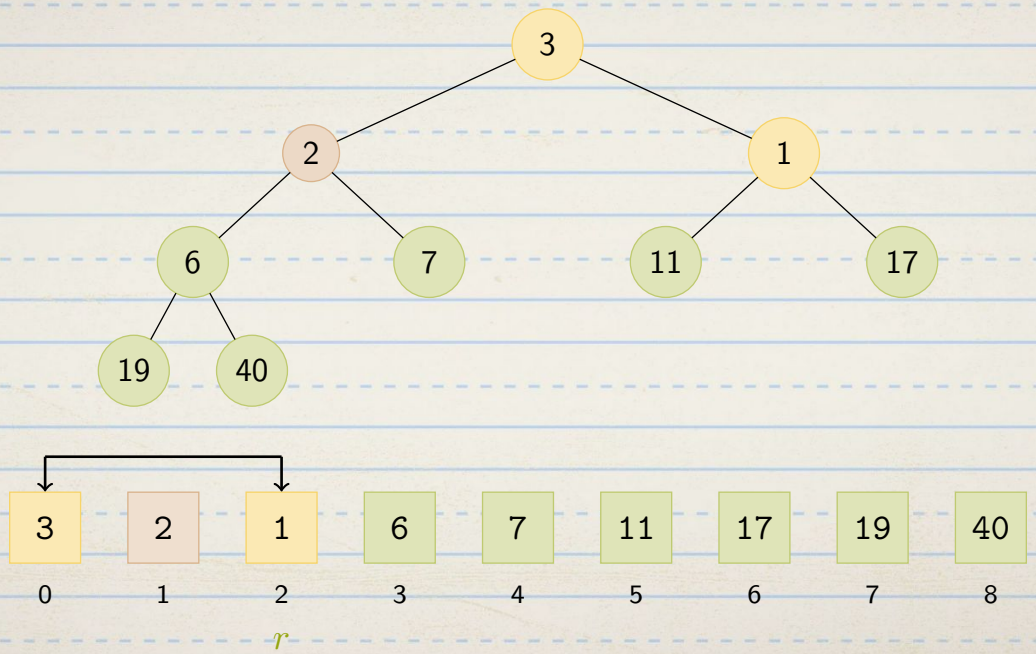


Sắp xếp dãy số dựa trên max-heap

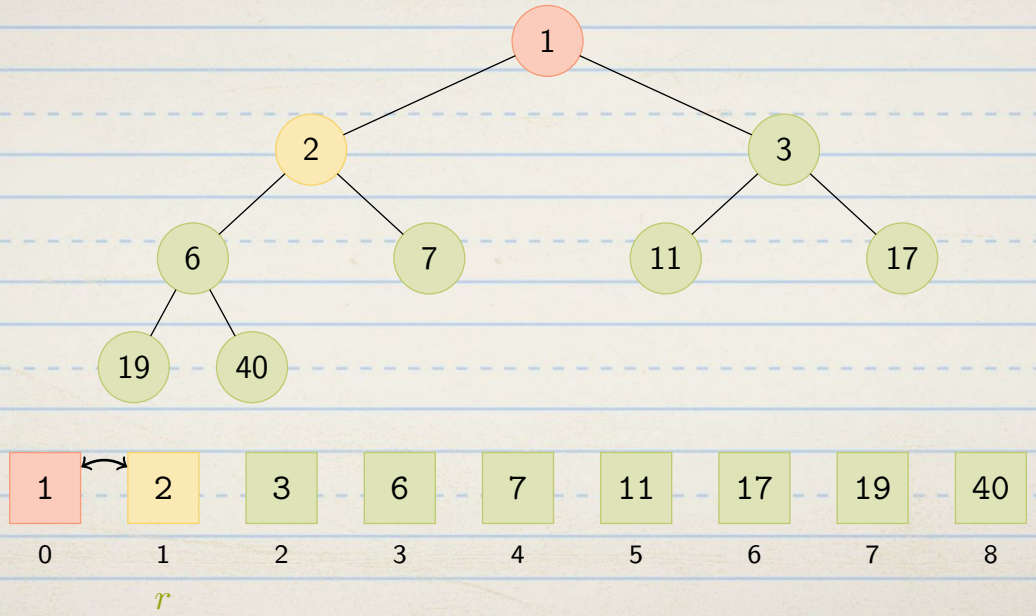




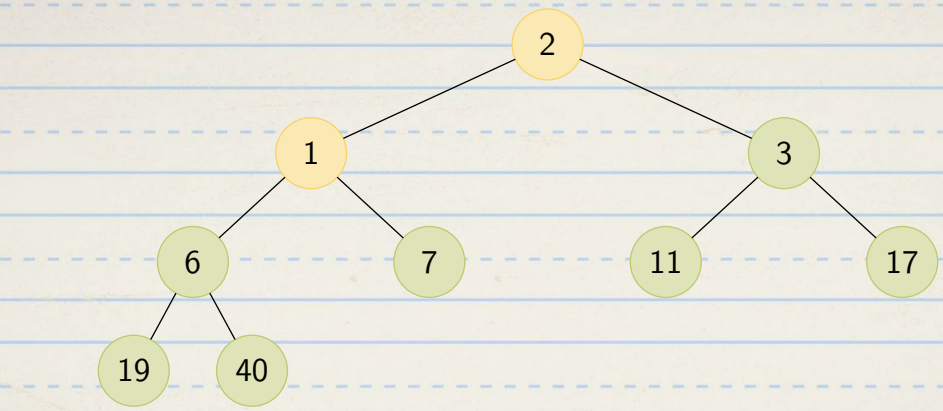
Sắp xếp dãy số dựa trên max-heap

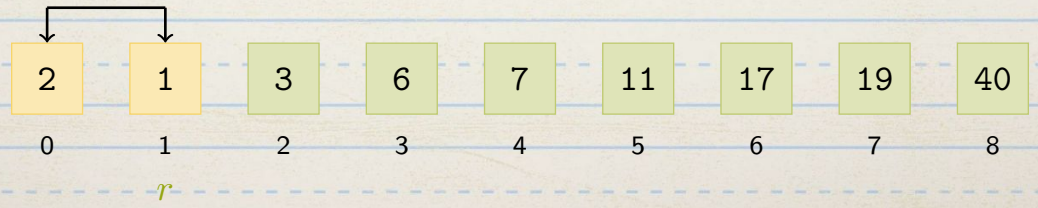


Sắp xếp dãy số dựa trên max-heap

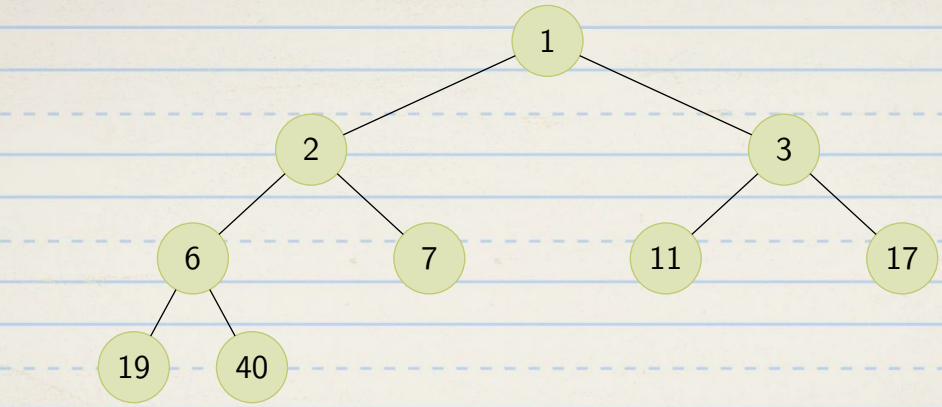


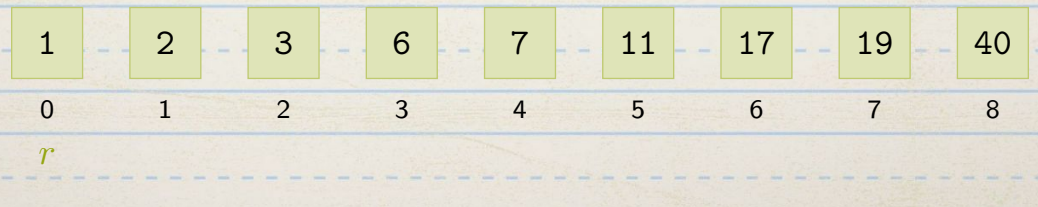
Sắp xếp dãy số dựa trên max-heap





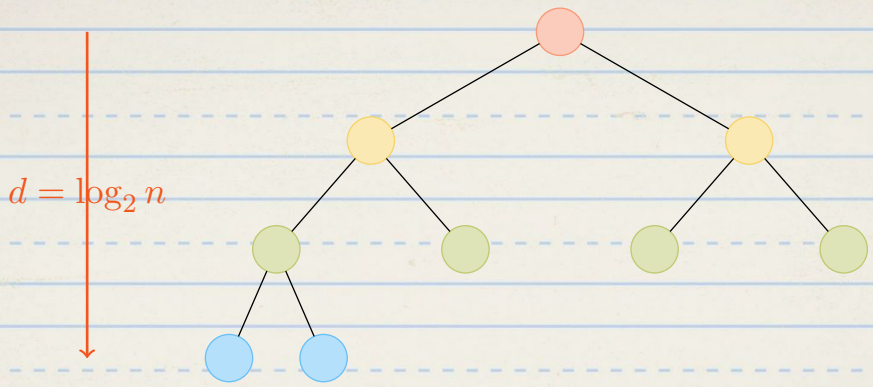
Sắp xếp dãy số dựa trên max-heap





NỘI DUNG

Đánh giá thuật toán



- Trường hợp xấu nhất, hàm hiệu chỉnh `Heapify` sẽ thực hiện từ nút gốc đến các nút lá. Cây có mức là d thì độ phức tạp thời gian của hàm hiệu chỉnh là

$$T(n) = O(d) = O(\log_2 n).$$

Đánh giá thuật toán

- Hiệu chỉnh heap là thao tác chính của thuật toán HeapSort.
  - Bước 1. Tạo heap
    $$\frac{n}{2} - 1.$$
  - Bước 2. Sắp xếp heap
    $$n - 1.$$
- Do đó, độ phức tạp thời gian của thuật toán HeapSort

$$T(n) = \left(\frac{3n}{2} - 2\right) \log_2 n = O(n \log n).$$

Bài tập

1. Khi biểu diễn Heap bằng mảng, xét một phần tử x tại vị trí i, hãy cho biết phần tử bên phải của phần tử bên phải của phần tử x nằm ở vị trí nào?
2. Giả sử Max-Heap là một dãy số gồm nhiều phần tử khác nhau, hãy cho biết phần tử nhỏ nhất nằm ở vị trí nào trong trường hợp biểu diễn bằng mảng, cấu trúc cây nhị phân?
3. Giả sử Max-Heap là một dãy số gồm nhiều phần tử khác nhau, thời gian tìm phần tử nhỏ nhất là $O(logn)$ phải không?
4. Áp dụng thuật toán HeapSort sắp xếp dãy b và c theo thứ tự tăng dần.
   - $a = 1, 2, 3, 4, 5.$
   - $b = 5, 4, 3, 2, 1.$

   Hãy nhận xét sau khi thực hiện xong thuật toán.

Tài liệu tham khảo


Donald E. Knuth.
The Art of Computer Programming, Volume 3.
Addison-Wesley, 1998.


Dương Anh Đức, Trần Hạnh Nhi.
Nhập môn Cấu trúc dữ liệu và Thuật toán.
Đại học Khoa học tự nhiên TP Hồ Chí Minh, 2003.


Niklaus Wirth.
Algorithms + Data Structures = Programs.
Prentice-Hall, 1976.


Robert Sedgewick.
Algorithms in C.
Addison-Wesley, 1990.