

# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

## SẮP XẾP

# Nội dung

---

- Giới thiệu bài toán sắp xếp
- Thuật toán sắp xếp chèn
- Thuật toán sắp xếp lựa chọn
- Thuật toán sắp xếp nổi bọt
- Thuật toán sắp xếp trộn
- Thuật toán sắp xếp nhanh
- Thuật toán sắp xếp vun đống

# Giới thiệu bài toán sắp xếp

---

- Sắp xếp là việc đưa các phần tử của một dãy theo đúng thứ tự (không giảm hoặc không tăng) dựa vào 1 giá trị khoá
- Thiết kế thuật toán sắp xếp hiệu quả là một việc đặc biệt quan trọng do việc sắp xếp xuất hiện trong rất nhiều tình huống tính toán
- Hai thao tác cơ bản trong một thuật toán sắp xếp
  - $\text{Compare}(a, b)$ : so sánh khoá của 2 phần tử  $a$  và  $b$
  - $\text{Swap}(a, b)$ : đổi chỗ 2 phần tử  $a$  và  $b$  cho nhau
- Không giảm tổng quát, giả sử cần sắp xếp dãy  $a_1, a_2, \dots, a_n$  theo thứ tự không giảm của giá trị

# Giới thiệu bài toán sắp xếp

---

- Phân loại thuật toán sắp xếp
  - Sắp xếp *tại chỗ*: sử dụng bộ nhớ trung gian là hằng số, không phụ thuộc độ dài dãy đầu vào
  - Sắp xếp *ổn định*: duy trì thứ tự tương đối giữa 2 phần tử có cùng giá trị khoá (vị trí tương đối giữa 2 phần tử có cùng khoá không đổi trước và sau khi sắp xếp)
  - Thuật toán sắp xếp dựa trên so sánh: sử dụng phép so sánh để quyết định thứ tự phần tử (counting sort không phải là thuật toán sắp xếp dựa trên so sánh)

# Thuật toán sắp xếp chèn (insertion sort)

- Thuật toán diễn ra qua các bước lặp  $k = 2, 3, \dots, n$
- Tại mỗi bước thứ  $k$ : chèn  $a_k$  vào đúng vị trí trong dãy đã được sắp  $a_1, a_2, \dots, a_{k-1}$  để thu được dãy được sắp đúng thứ tự
- Sau bước thứ  $k$  thì dãy  $a_1, a_3, \dots, a_k$  đã được sắp đúng thứ tự, dãy còn lại  $a_{k+1}, \dots, a_n$  giữ nguyên vị trí

```
void insertionSort(int A[], int N)
{
    // index từ 1 -> N
    for(int k = 2; k <= N; k++){
        int last = A[k];
        int j = k;
        while(j > 1 && A[j-1] >
            last){
            A[j] = A[j-1];
            j--;
        }
        A[j] = last;
    }
}
```

# Thuật toán sắp xếp chèn (insertion sort)

- Ví dụ: 5, 7, 3, 8, 1, 2, 9, 4, 6

5	7	3	8	1	2	9	4	6
---	---	---	---	---	---	---	---	---

3	5	7	8	1	2	9	4	6
---	---	---	---	---	---	---	---	---

3	5	7	8	1	2	9	4	6
---	---	---	---	---	---	---	---	---

1	3	5	7	8	2	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	5	7	8	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	5	7	8	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	4	5	7	8	9	6
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

# Thuật toán sắp xếp lựa chọn (selection sort)

- Chọn số nhỏ nhất xếp vào vị trí thứ 1
- Chọn số nhỏ thứ 2 xếp vào vị trí thứ 2
- Chọn số nhỏ thứ 3 xếp vào vị trí thứ 3
- ...

```
void selectionSort(int A[], int N) {  
    // index từ 1 -> N  
    for(int k = 1; k <= N; k++){  
        int min = k;  
        for(int j = k+1; j <= N; j++){  
            if(A[min] > A[j]) min = j;  
        }  
        int tmp = A[min];  
        A[min] = A[k];  
        A[k] = tmp;  
    }  
}
```

# Thuật toán sắp xếp lựa chọn (selection sort)

- Ví dụ: 5, 7, 3, 8, 1, 2, 9, 4, 6

5	7	3	8	1	2	9	4	6
---	---	---	---	---	---	---	---	---

1	7	3	8	5	2	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	8	5	7	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	8	5	7	9	4	6
---	---	---	---	---	---	---	---	---

1	2	3	4	5	7	9	8	6
---	---	---	---	---	---	---	---	---

1	2	3	4	5	7	9	8	6
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	9	8	7
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



# Thuật toán sắp xếp nổi bọt (bubble sort)

- Duyệt dãy từ trái qua phải (hoặc từ phải qua trái)
- Tại mỗi bước, so sánh 2 phần tử đứng cạnh nhau và tiến hành đổi chỗ 2 phần tử đó nếu phần tử trước lớn hơn phần tử sau
- Lặp lại quá trình trên khi nào trong dãy vẫn còn 2 phần tử đứng cạnh nhau mà phần tử trước lớn hơn phần tử sau

```
void bubbleSort(int A[], int N) {  
    // index từ 1 đến N  
    int swapped;  
    do{  
        swapped = 0;  
        for(int i = 1; i < N; i++){  
            if(A[i] > A[i+1]){  
                int tmp = A[i];  
                A[i] = A[i+1];  
                A[i+1] = tmp;  
                swapped = 1;  
            }  
        }  
    }while(swapped == 1);  
}
```

# Thuật toán sắp xếp nổi bọt (bubble sort)

---

- Ví dụ: 5, 7, 3, 8, 1, 2, 9, 4, 6

5	3	7	1	2	8	4	6	9
---	---	---	---	---	---	---	---	---

3	5	1	2	7	4	6	8	9
---	---	---	---	---	---	---	---	---

3	1	2	5	4	6	7	8	9
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

# Thuật toán sắp xếp trộn (merge sort)

- Dựa trên chia để trị
- Chia dãy  $a_1, \dots, a_n$  thành 2 dãy con có độ dài bằng nhau
- Sắp xếp 2 dãy con bằng thuật toán sắp xếp trộn
- Trộn 2 dãy con đã được sắp với nhau để thu được dãy ban đầu được sắp thứ tự

```
void mergeSort(int A[], int L, int R) {  
    if(L < R){  
        int M = (L+R)/2;  
        mergeSort(A,L,M);  
        mergeSort(A,M+1,R);  
        merge(A,L,M,R);  
    }  
}
```

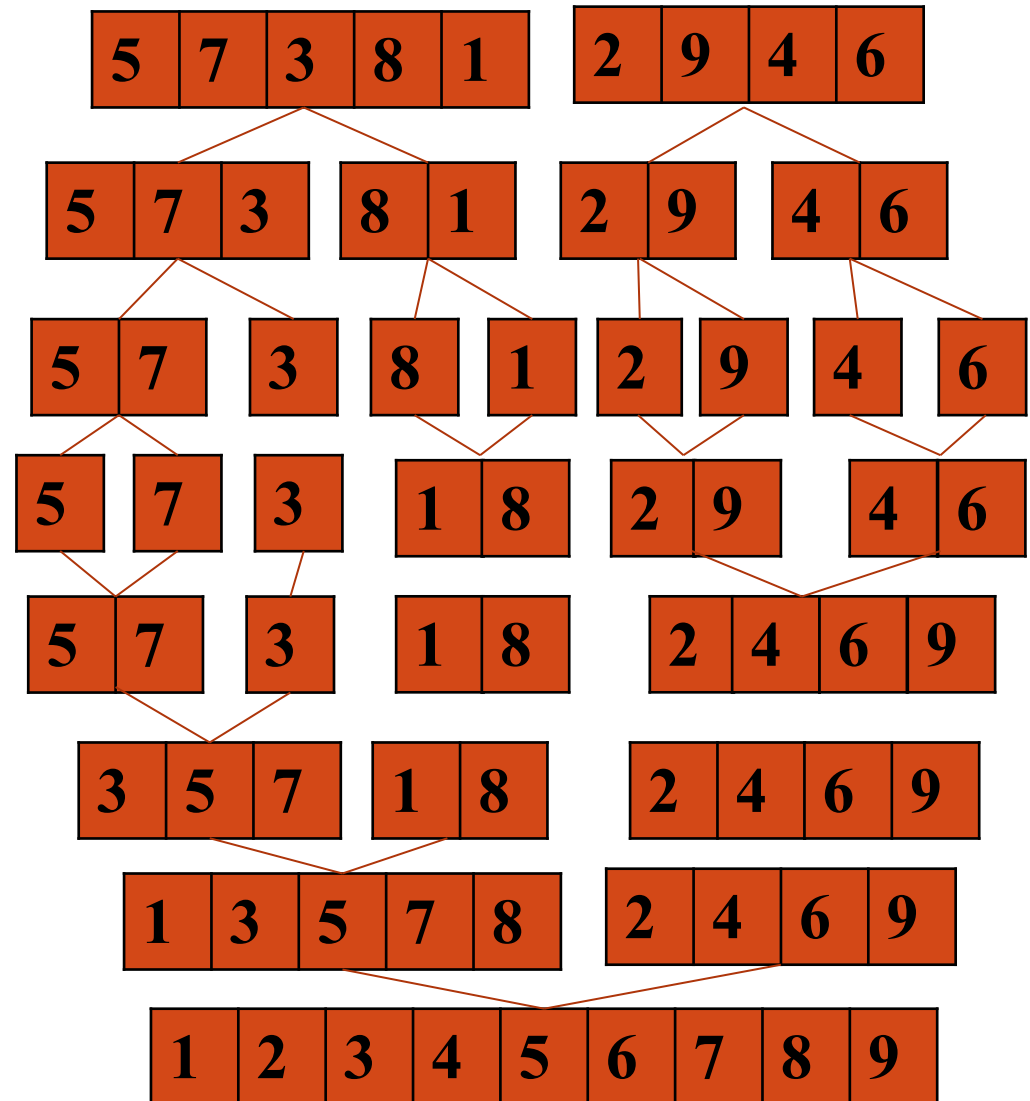
# Thuật toán sắp xếp trộn (merge sort)

- Sử dụng mảng trung gian để lưu trữ tạm thời trong quá trình trộn

```
void merge(int A[], int L, int M, int R) {  
    // tron 2 day da sap A[L..M] va A[M+1..R]  
    int i = L; int j = M+1;  
    for(int k = L; k <= R; k++){  
        if(i > M){ TA[k] = A[j]; j++;}  
        else if(j > R){TA[k] = A[i]; i++;}  
        else{  
            if(A[i] < A[j]){  
                TA[k] = A[i]; i++;  
            }  
            else {  
                TA[k] = A[j]; j++;  
            }  
        }  
    }  
    for(int k = L; k <= R; k++) A[k] = TA[k];  
}
```

# Thuật toán sắp xếp trộn (merge sort)

- Ví dụ: 5, 7, 3, 8, 1, 2, 9, 4, 6



# Thuật toán sắp xếp nhanh (quick sort)

---

- Chọn một phần tử bất kỳ dùng làm phần tử trụ (pivot)
- Sắp xếp lại dãy sao cho
  - Các phần tử đứng trước phần tử trụ sẽ không lớn hơn phần tử trụ
  - Các phần tử đứng sau phần tử trụ không nhỏ hơn phần tử trụ
- Khi đó phần tử trụ (có thể bị thay đổi vị trí) đã đứng đúng vị trí trong dãy khi được sắp thứ tự
- Tiến hành sắp xếp dãy con đứng trước và sau phần tử trụ bằng sắp xếp nhanh

# Thuật toán sắp xếp nhanh (quick sort)

---

```
void quickSort(int A[], int L, int R) {  
    if(L < R){  
        int index = (L + R)/2;  
        index = partition(A, L, R, index);  
        if(L < index)  
            quickSort(A, L, index-1);  
        if(index < R)  
            quickSort(A, index+1, R);  
    }  
}
```

# Thuật toán sắp xếp nhanh (quick sort)

---

```
int partition(int A[], int L, int R, int indexPivot) {
    int pivot = A[indexPivot];
    swap(A[indexPivot], A[R]);
    int storeIndex = L;
    for(int i = L; i <= R-1; i++){
        if(A[i] < pivot){
            swap(A[storeIndex], A[i]);
            storeIndex++;
        }
    }
    swap(A[storeIndex], A[R]);
    return storeIndex;
}
```



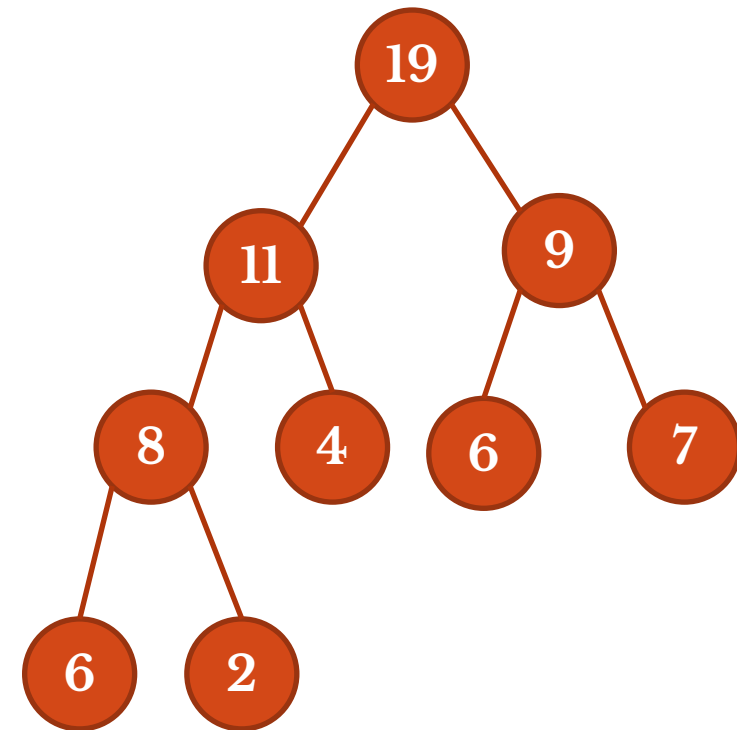
# Thuật toán sắp xếp nhanh (quick sort)

- Ví dụ: 5, 7, 3, 8, 1, 2, 9, 4, 6



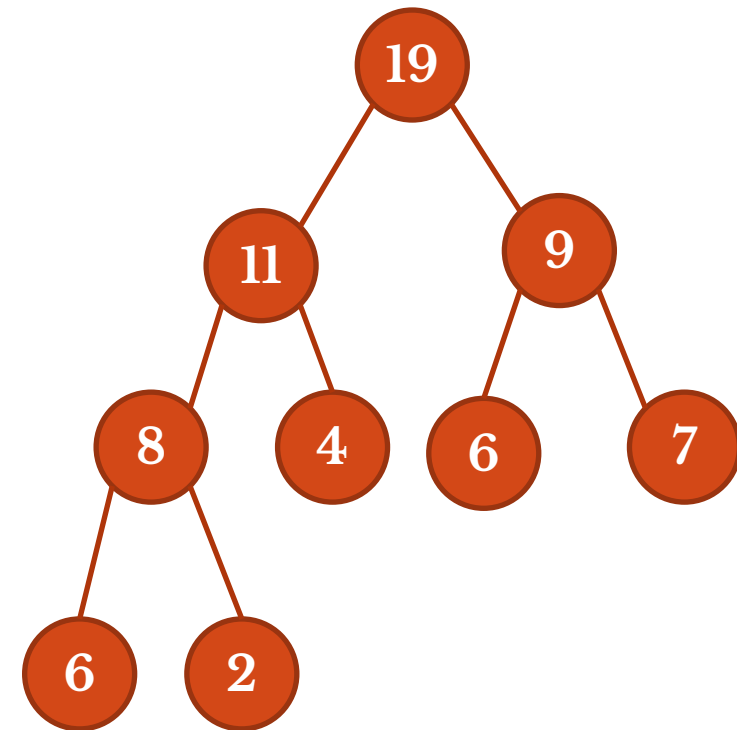
# Thuật toán sắp xếp vun đống (heap sort)

- Cấu trúc đống (max-heap)
  - Cây nhị phân đầy đủ (complete tree)
  - Khoá của mỗi nút lớn hơn hoặc bằng khoá của 2 nút con (tính chất của max-heap)
- Ánh xạ từ dãy  $A[1 \dots N]$  sang cây nhị phân đầy đủ
  - Gốc là  $A[1]$
  - $A[2i]$  và  $A[2i+1]$  là con trái và con phải của  $A[i]$
  - Chiều cao của cây là  $\log N + 1$



# Thuật toán sắp xếp vun đống (Heap Sort)

- Vun lại đống (heapify)
  - Tình trạng:
    - Tính chất max-heap ở  $A[i]$  bị phá vỡ
    - Tính chất max-heap ở các cây con của  $A[i]$  đã được thoả mãn
  - Vun lại đống để khôi phục lại tính chất max-heap trên cây gốc  $A[i]$



# Thuật toán sắp xếp vun đống (Heap Sort)

- Vun lại đống (heapify)
- Chọn nút con lớn nhất
- Đổi chỗ nút con và  $A[i]$  cho nhau nếu nút con này lớn hơn  $A[i]$  và vun lại đống bắt đầu từ nút con này

```
void heapify(int A[], int i, int N)
{
    int L = 2*i;
    int R = 2*i+1;
    int max = i;
    if(L <= N && A[L] > A[i])
        max = L;
    if(R <= N && A[R] > A[max])
        max = R;
    if(max != i){
        swap(A[i], A[max]);
        heapify(A,max,N);
    }
}
```

# Thuật toán sắp xếp vun đống (Heap Sort)

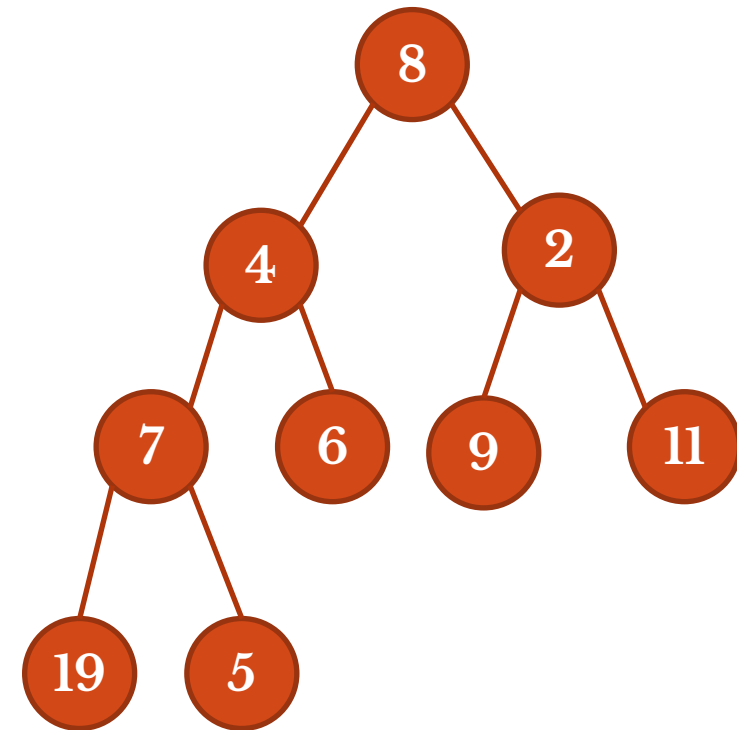
- Sắp xếp vun đống
  - Xây dựng max-heap (thủ tục buildHeap)
  - Đổi chỗ  $A[1]$  và  $A[N]$  cho nhau
  - Vun lại đống bắt đầu từ  $A[1]$  cho  $A[1..N-1]$
  - Đổi chỗ  $A[1]$  và  $A[N-1]$  cho nhau
  - Vun lại đống bắt đầu từ  $A[1]$  cho  $A[1..N-2]$
  - ...

```
void buildHeap(int A[], int N) {  
    for(int i = N/2; i >= 1; i--)  
        heapify(A,i,N);  
}  
  
void heapSort(int A[], int N) {  
    // index từ 1 -> N  
    buildHeap(A,N);  
    for(int i = N; i > 1; i--) {  
        swap(A[1], A[i]);  
        heapify(A, 1, i-1);  
    }  
}
```

# Thuật toán sắp xếp vun đống (Heap Sort)

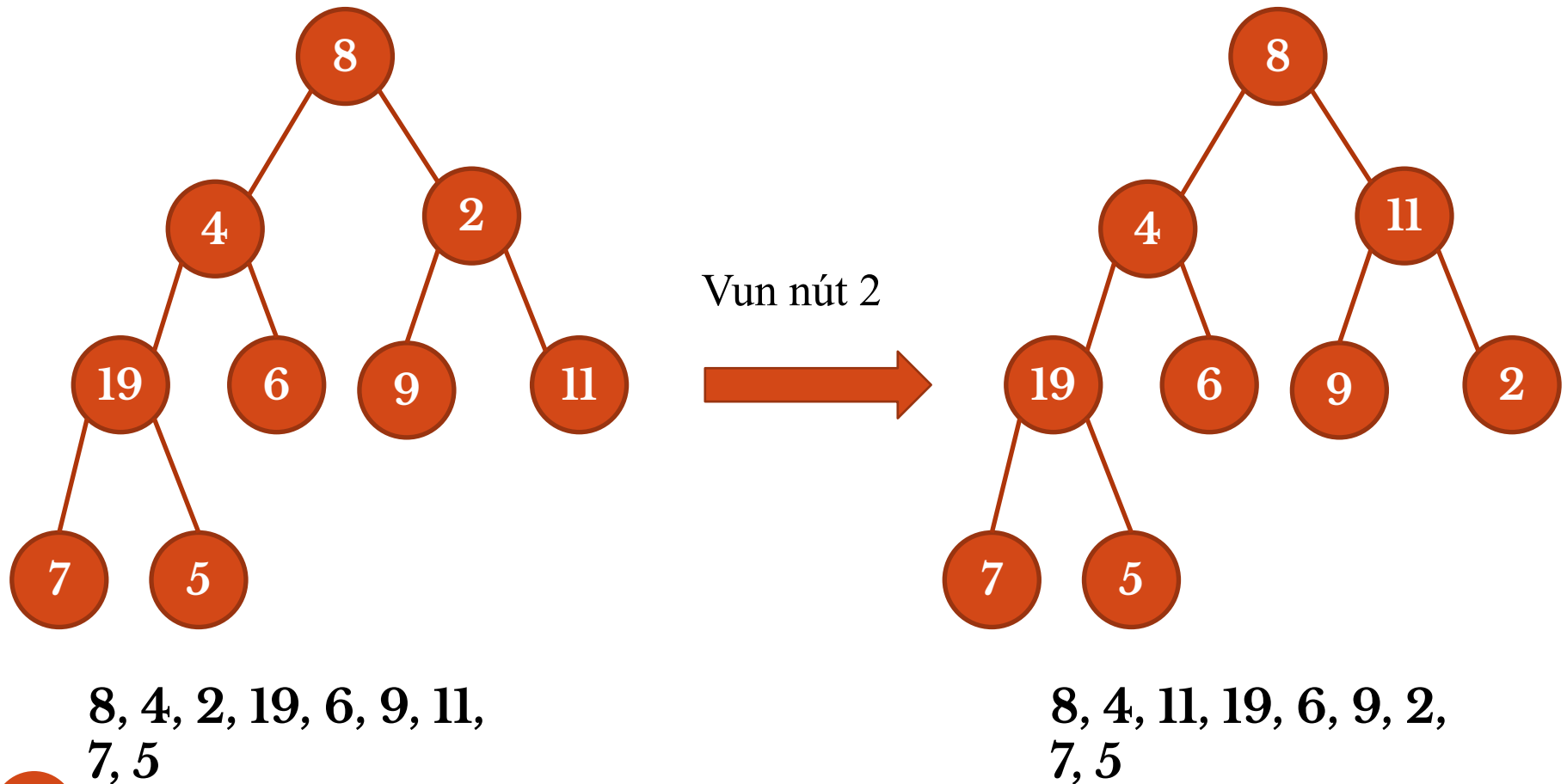
- Ví dụ: sắp xếp dãy sau theo thứ tự không giảm của khoá  
8, 4, 2, 7, 6, 9, 11, 19, 5

Cây nhị phân đầy đủ

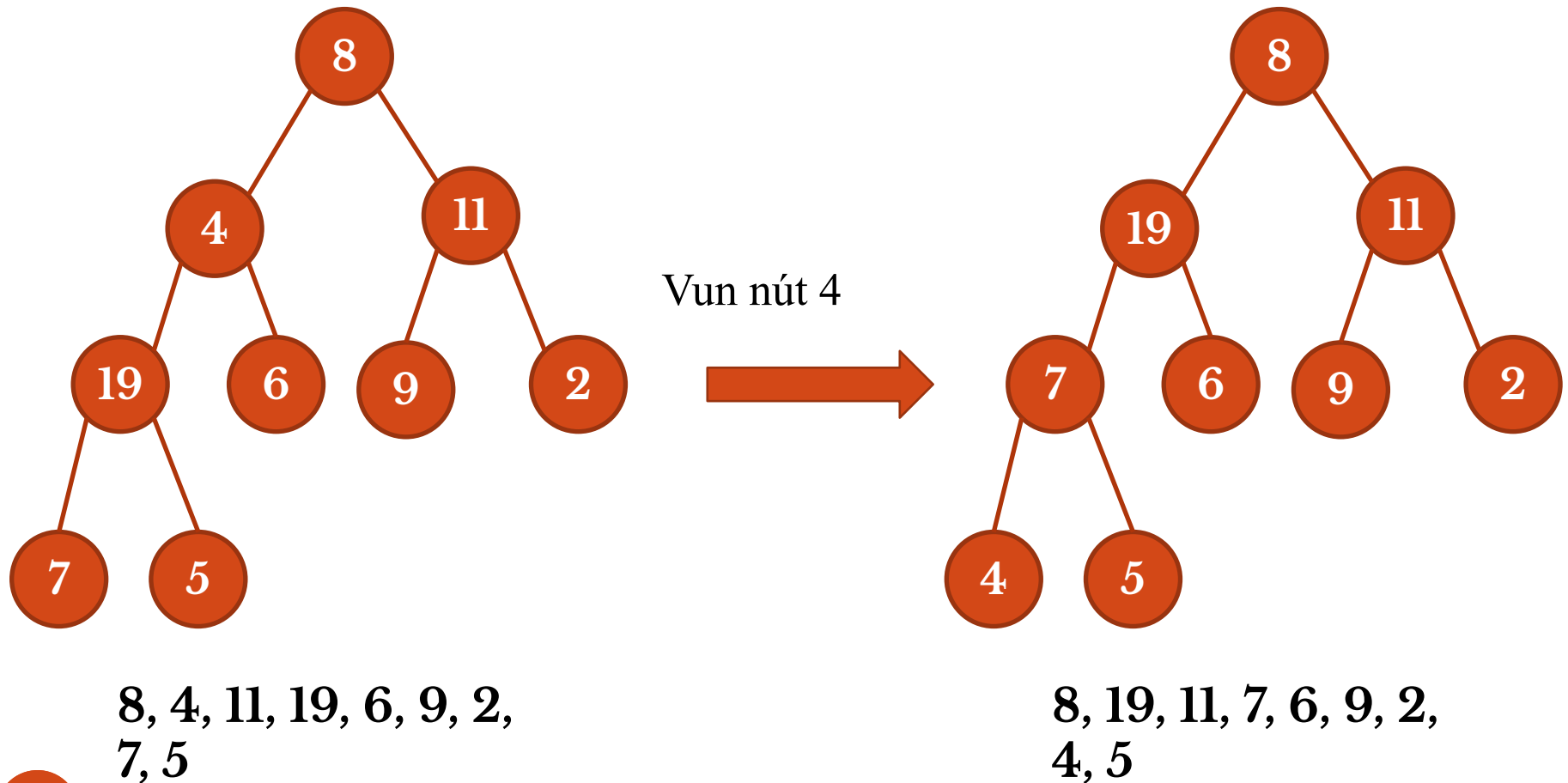


8, 4, 2, 7, 6, 9, 11,  
19, 5

# Thuật toán sắp xếp vun đống (Heap Sort)

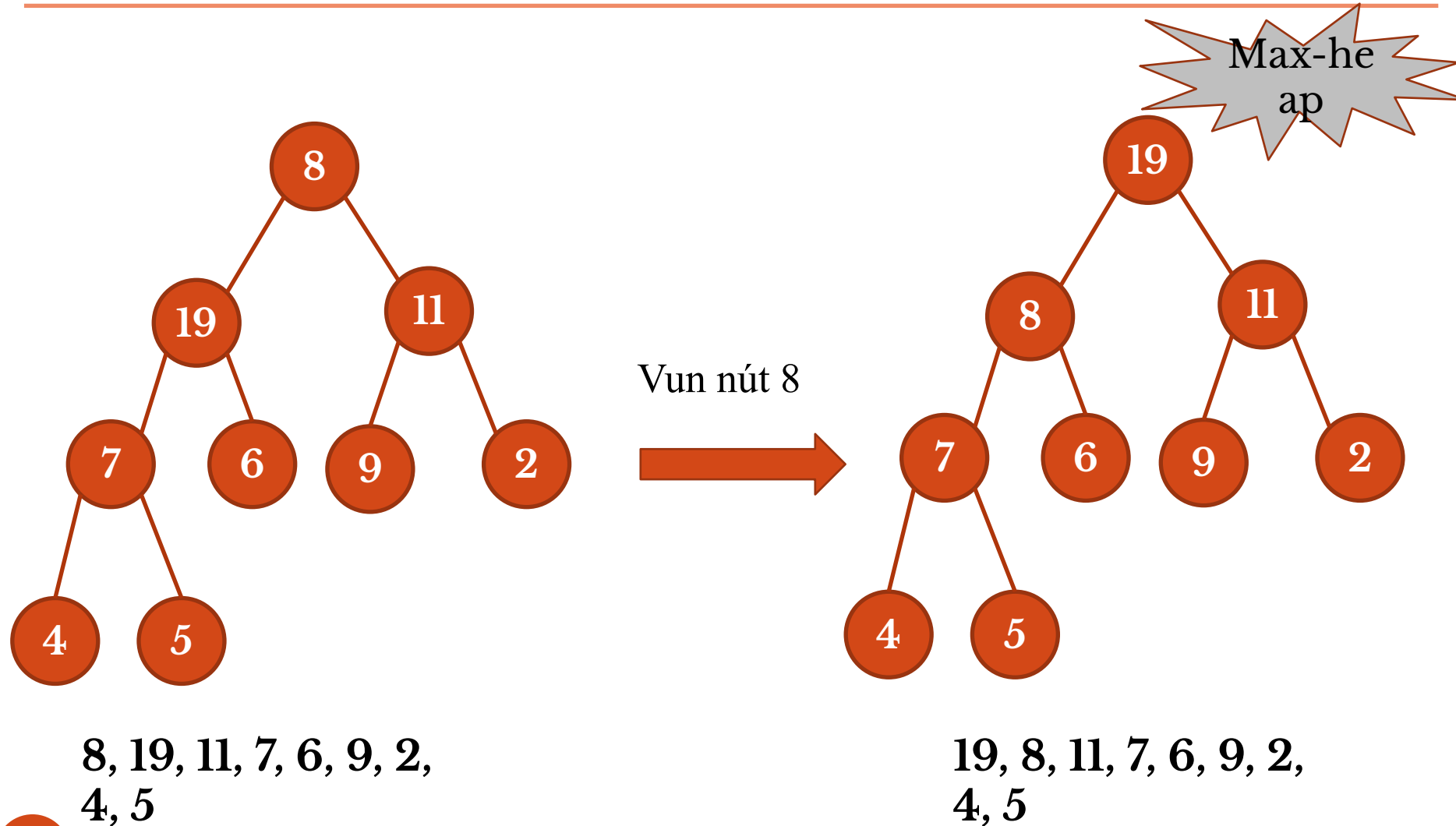


# Thuật toán sắp xếp vun đống (Heap Sort)

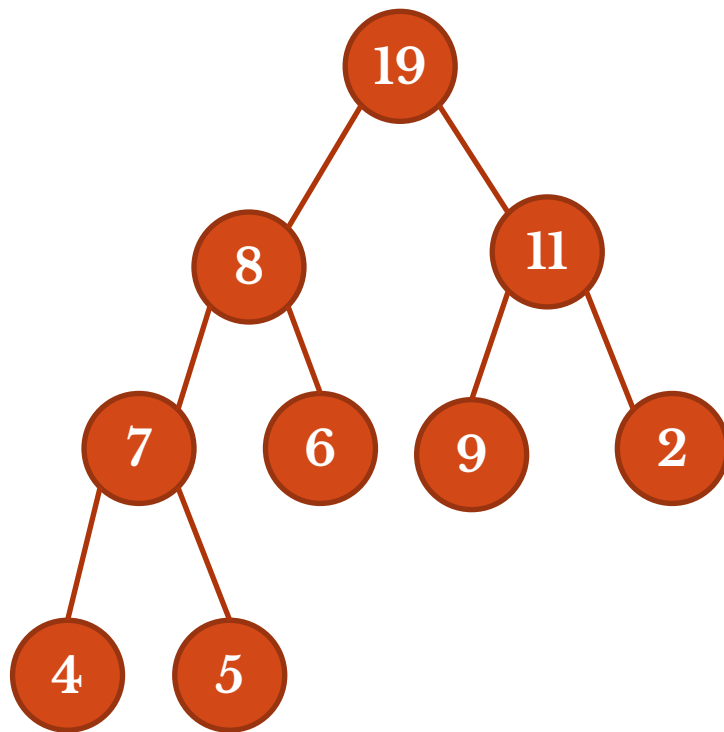




# Thuật toán sắp xếp vun đống (Heap Sort)

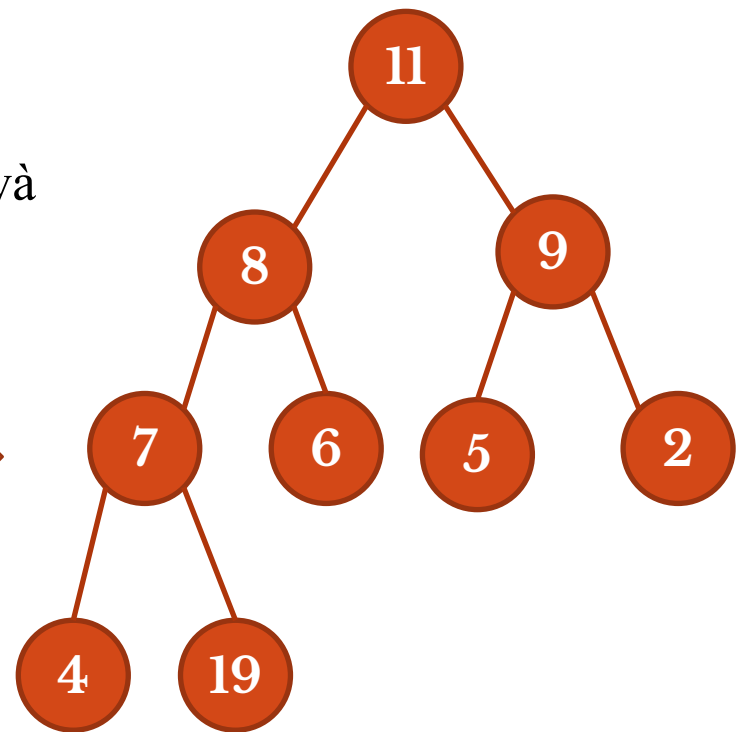


# Thuật toán sắp xếp vun đống (Heap Sort)



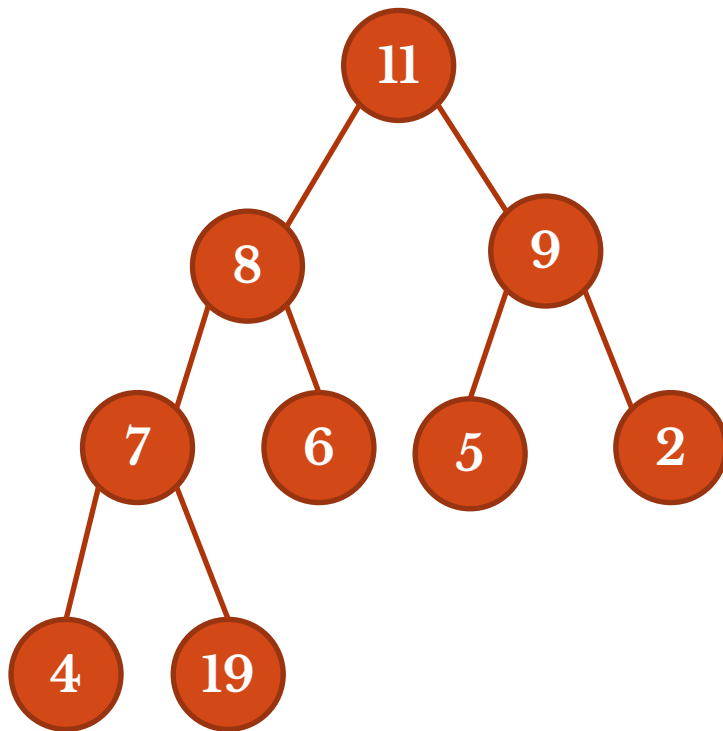
19, 8, 11, 7, 6, 9, 2,  
4, 5

Đổi chỗ 19 và  
5 cho nhau,  
vun lại heap



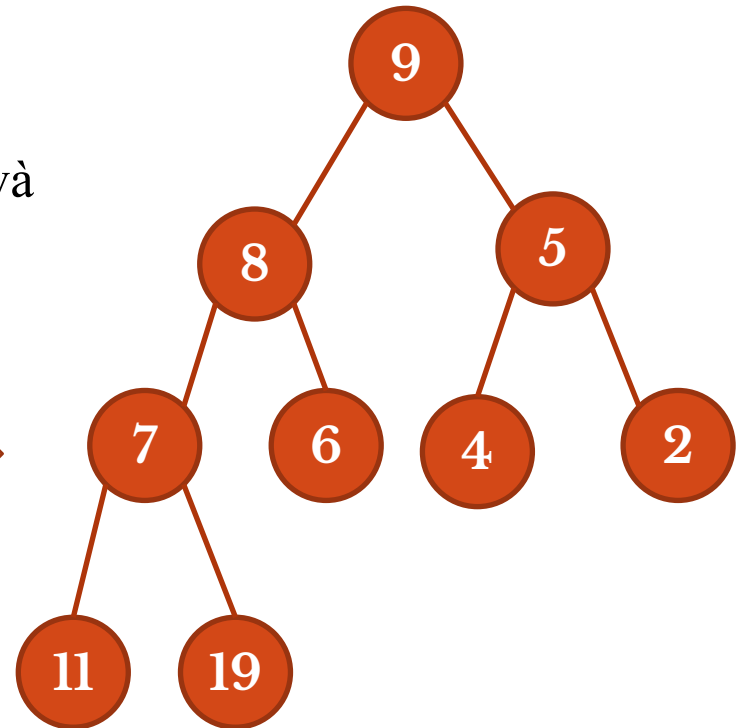
11, 8, 9, 7, 6, 5, 2, 4,  
19

# Thuật toán sắp xếp vun đống (Heap Sort)



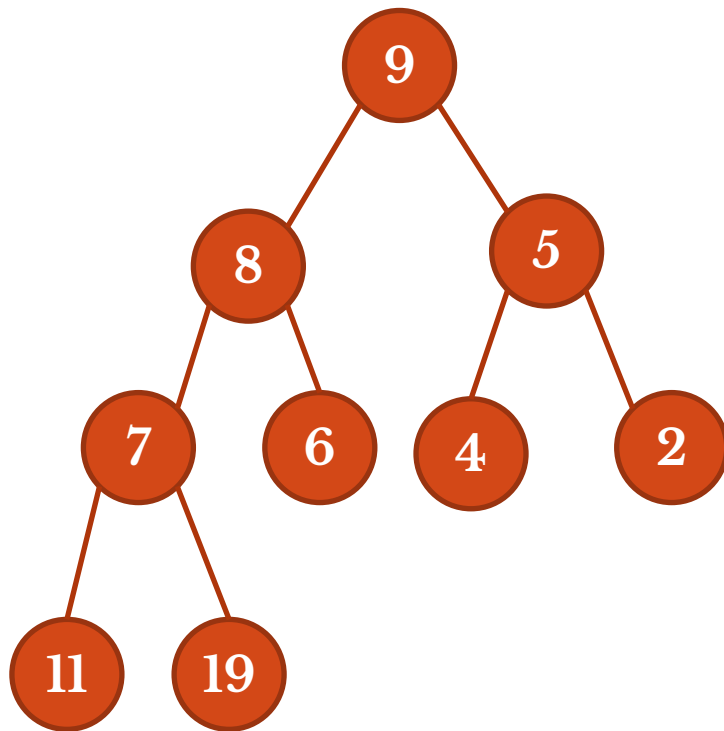
11, 8, 9, 7, 6, 5, 2, 4,  
19

Đổi chỗ 11 và  
4 cho nhau,  
vun lại heap



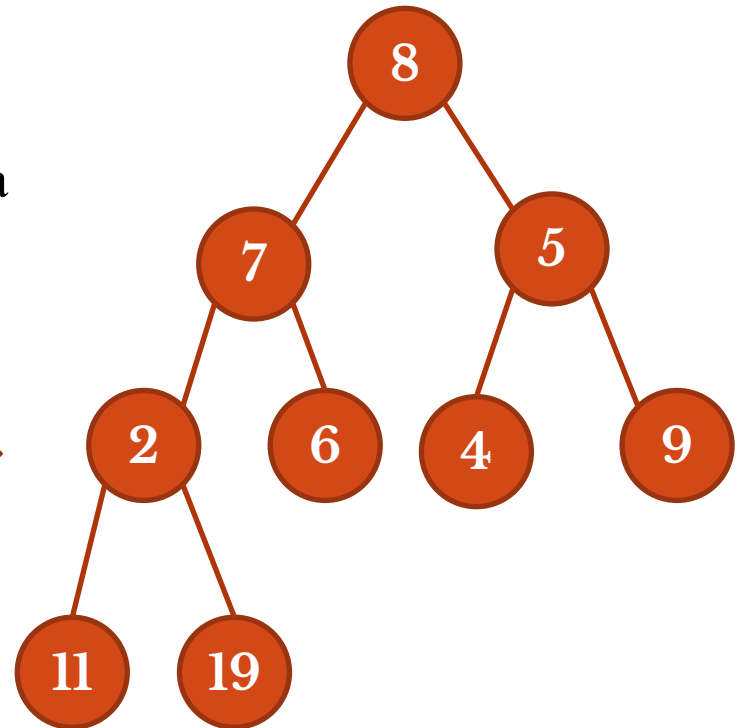
9, 8, 5, 7, 6, 4, 2, 11,  
19

# Thuật toán sắp xếp vun đống (Heap Sort)



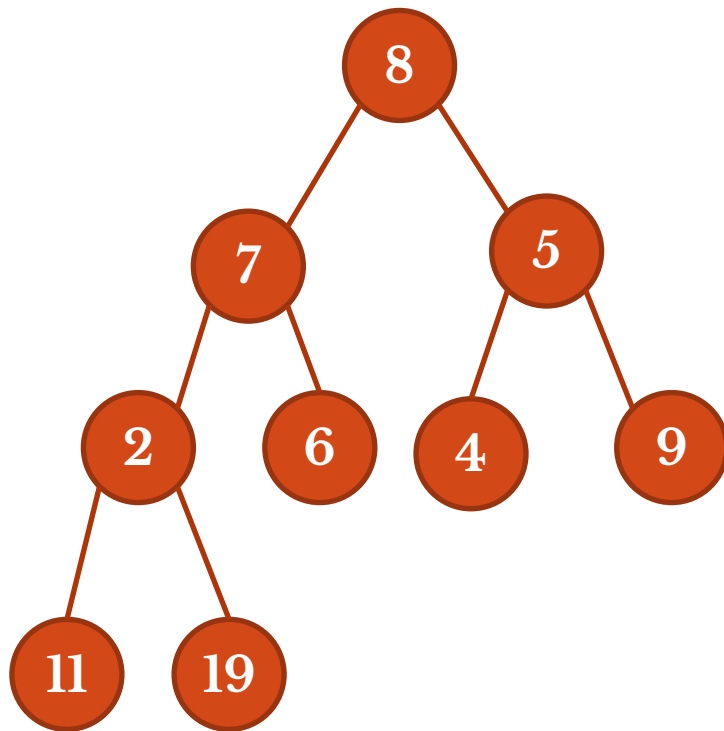
9, 8, 5, 7, 6, 4, 2, 11,  
19

Đổi chỗ 9 và  
2 cho nhau,  
vun lại heap



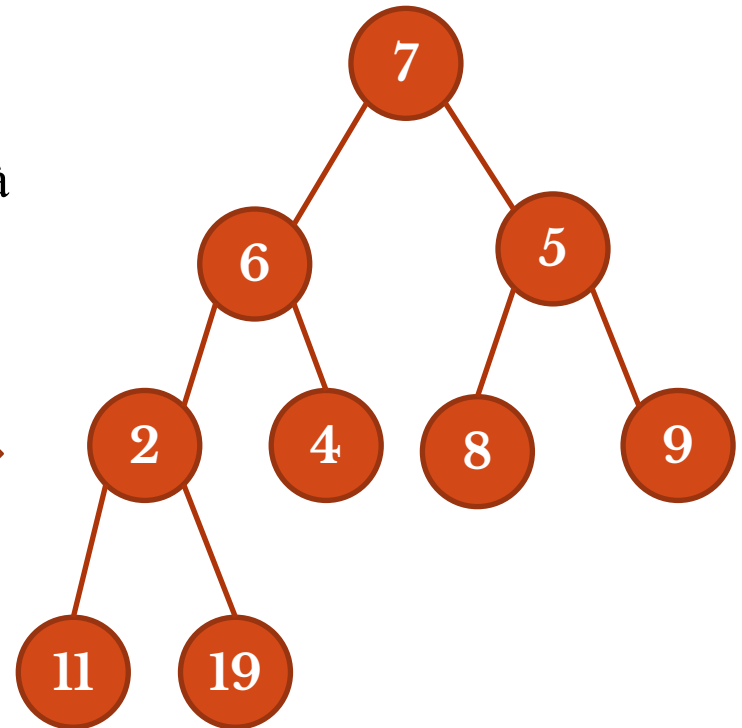
8, 7, 5, 2, 6, 4, 9, 11,  
19

# Thuật toán sắp xếp vun đống (Heap Sort)



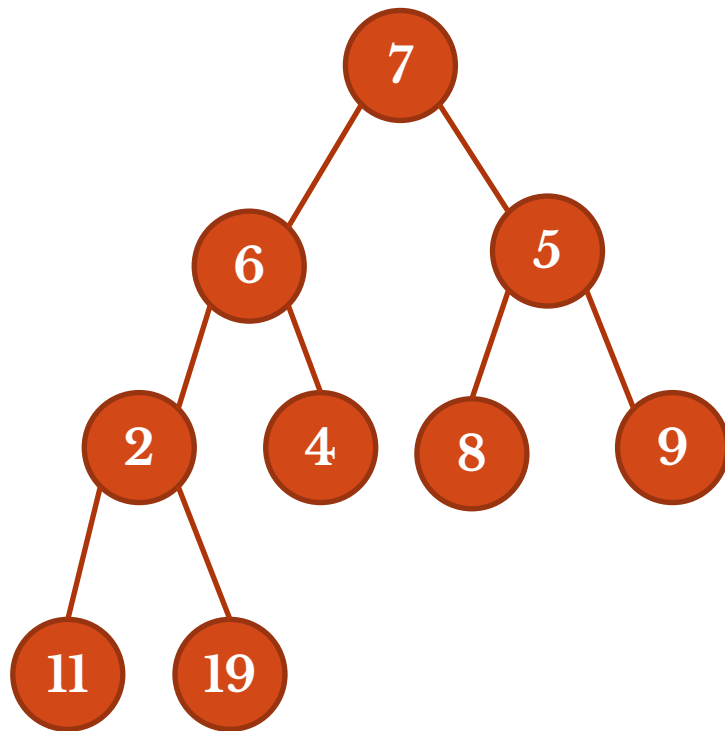
8, 7, 5, 2, 6, 4, 9, 11,  
19

Đổi chỗ 8 và  
4 cho nhau,  
vun lại heap



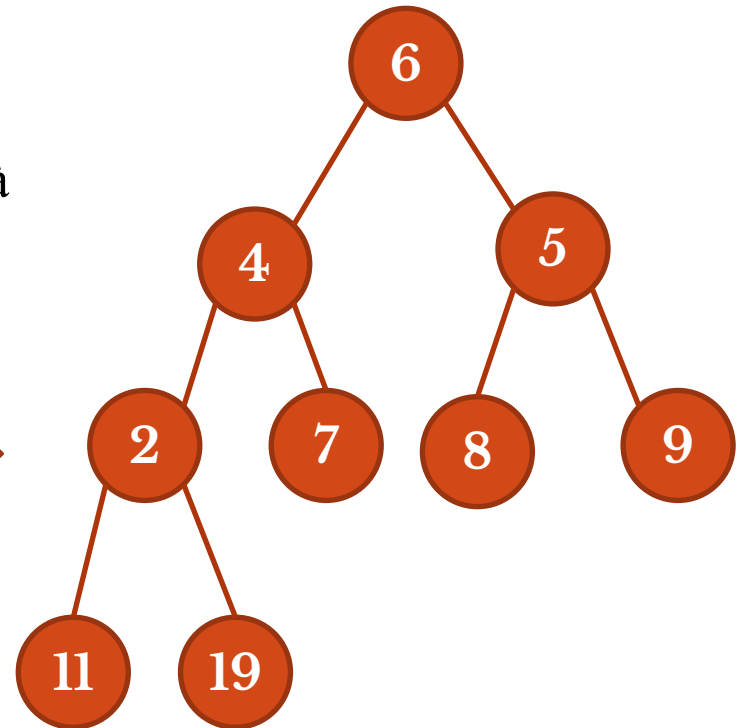
7, 6, 5, 2, 4, 8, 9, 11,  
19

# Thuật toán sắp xếp vun đống (Heap Sort)



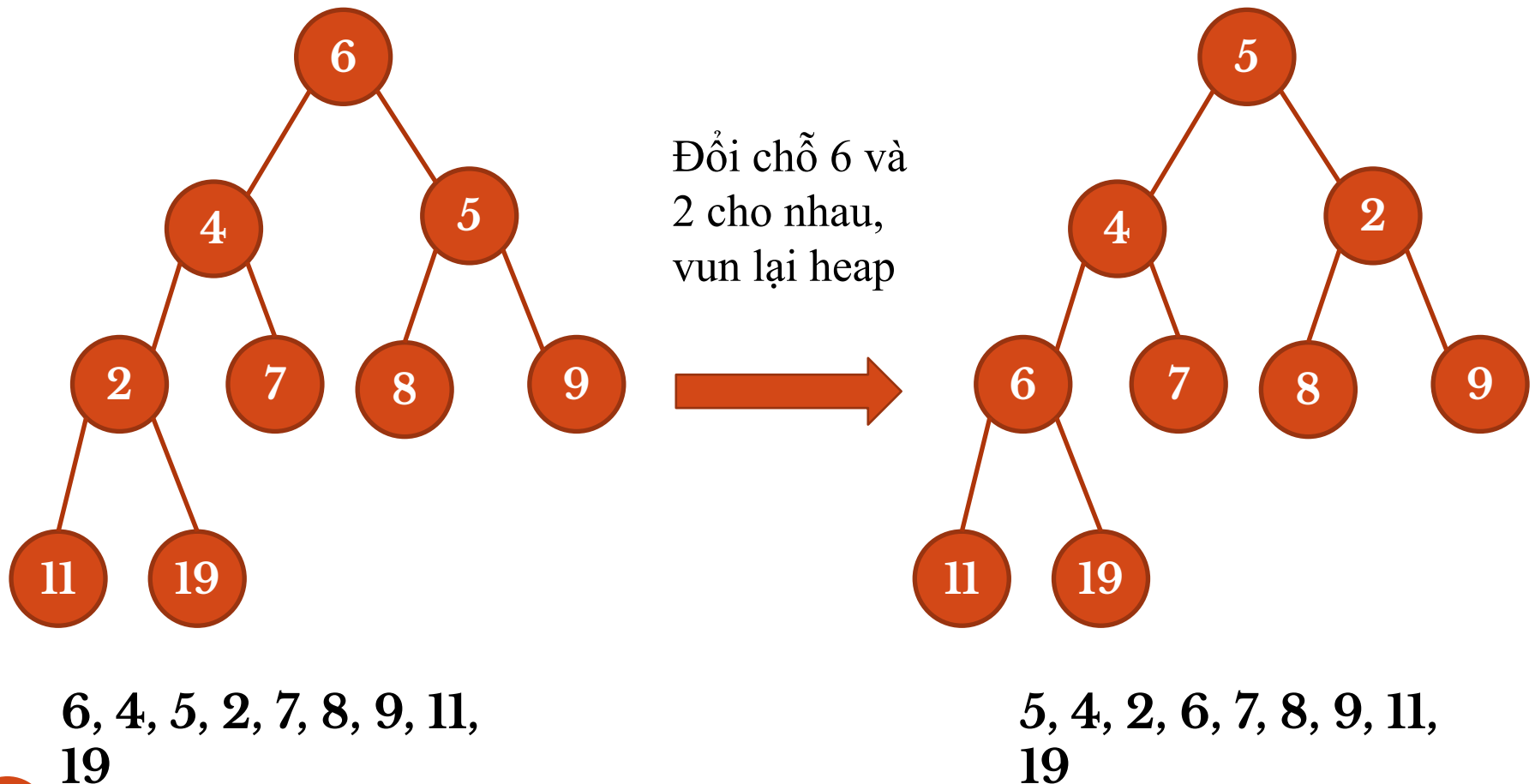
7, 6, 5, 2, 4, 8, 9, 11,  
19

Đổi chỗ 7 và  
4 cho nhau,  
vun lại heap

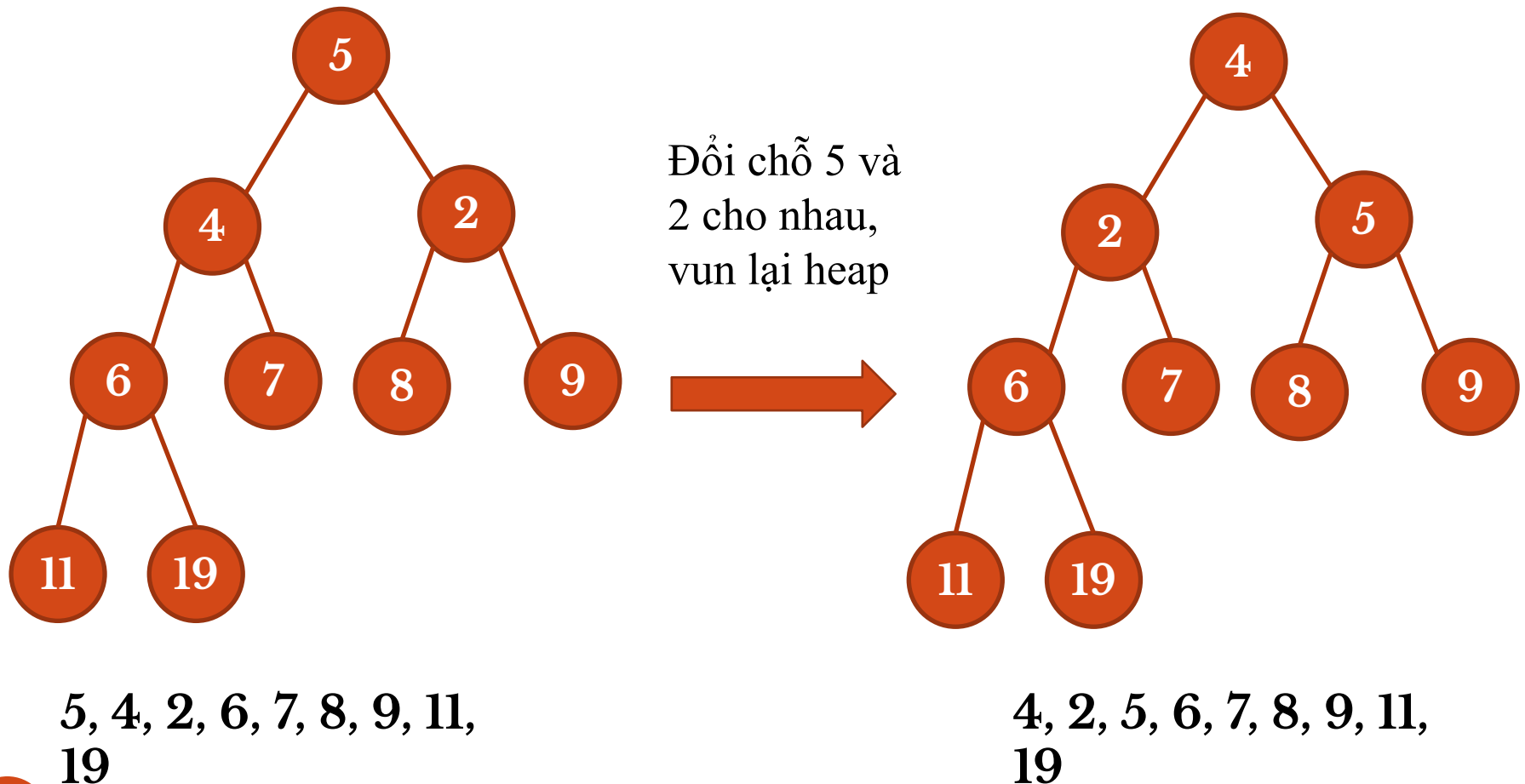


6, 4, 5, 2, 7, 8, 9, 11,  
19

# Thuật toán sắp xếp vun đống (Heap Sort)



# Thuật toán sắp xếp vun đống (Heap Sort)





# Thuật toán sắp xếp vun đống (Heap Sort)

