Phần 3: Cấu trúc dữ liệu và giải thuật

Chương 12: Các giải thuật sắp xếp

Nội dung chính

1. Đặt vấn đề
2. Các giải thuật sắp xếp cơ bản
   - Sắp xếp chọn (selection sort)
   - Sắp xếp nổi bọt (bubble sort)
   - Sắp xếp chèn (insertion sort)
3. Các giải thuật sắp xếp nâng cao
   - Sắp xếp nhanh (quick sort)
   - Sắp xếp vun đống (heap sort)
   - Sắp xếp trộn (merge sort)

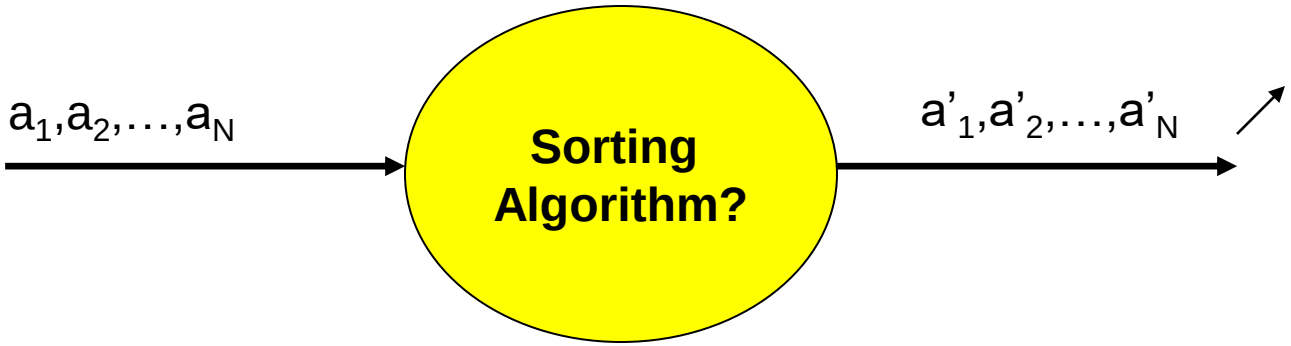
Nội dung chính

4. Các giải thuật sắp xếp tuyến tính
   - Sắp xếp đếm (counting sort)
   - Sắp xếp cơ số (radix sort)

Đặt vấn đề

- Yêu cầu:
  - **Bài toán tổng quát**: Cho trước một dãy A gồm N phần tử: $A = (a_1, a_2, \ldots, a_N)$. Ta cần tìm giải thuật sắp xếp các phần tử của dãy trên trên một thứ tự nào đó theo một tiêu chuẩn nào đó.
  - **Bài toán đơn giản**: Không giảm tính tổng quát của các giải thuật sắp xếp, đồng thời để đơn giản hóa việc trình bầy, sau này ta sẽ minh họa các giải thuật thông qua việc **sắp xếp một dãy N số theo trật tự tăng dần**.

Đặt vấn đề



- Với mỗi giải thuật, sẽ đưa ra:
  - Ý tưởng giải thuật
  - Cài đặt cơ bản (gồm một hoặc một số hàm)

Các giải thuật sắp xếp cơ bản

- Giới thiệu chung:
  - Các GTSX cơ bản đều có chung ý tưởng là ở mỗi bước, chỉ tập trung vào việc đưa từng phần tử của dãy cần SX vào đúng vị trí của nó trong dãy kết quả, mà không cần quan tâm đến vị trí của các phần tử khác
  - Với các GTSX nâng cao, mỗi bước của giải thuật không chỉ đưa từng phần tử vào đúng vị trí của nó trong dãy kết quả, mà nó còn kết hợp bố trí hợp lý các phần tử còn lại, nhằm giảm thiểu số thao tác ở những bước sau.
  - Chính vì lý do ở trên, các GTSX nâng cao thường chạy nhanh hơn các GTSX cơ bản, nhưng cũng thường phức tạp hơn trong ý tưởng giải thuật và cài đặt.

Các giải thuật sắp xếp cơ bản

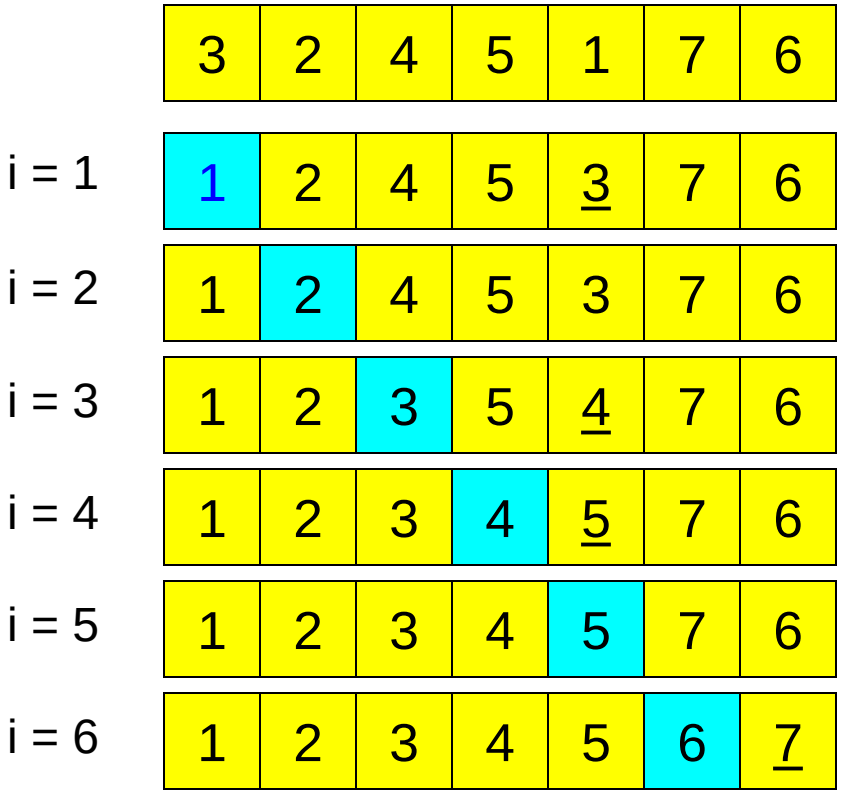
- Sắp xếp chọn
- Sắp xếp nổi bọt
- Sắp xếp chèn

Sắp xếp chọn

- Ý tưởng giải thuật
  - Đầu vào: dãy N số a_1, a_2,…, a_N
  - Đầu ra: dãy vào đã được sx theo chiều tăng dần
  - Giải thuật:
    - GT thực hiện trong đúng N-1 bước, đánh số các bước i =1, 2,…, N-1
    - Ở bước thứ i, tìm số nhỏ thứ i rồi đưa nó vào vị trí thứ i trong dãy

Minh họa hoạt động của GT

- G/s cho dãy ban đầu với N=7



Sắp xếp chọn

- Mô tả tựa lập trình

```
for (i=1; i<=N-1; i++) {
    //Tìm phần tử bé thứ i (bé nhất kể từ a_i đến a_N)
    m=i;
    for (k=i+1; k<=N; k++)
        if (a_k < a_m) m=k;
    //Đưa phần tử bé thứ i về vị trí thứ i
    if (m<>i) swap(a_m,a_i);
}
```

Sắp xếp chọn – Cài đặt hàm

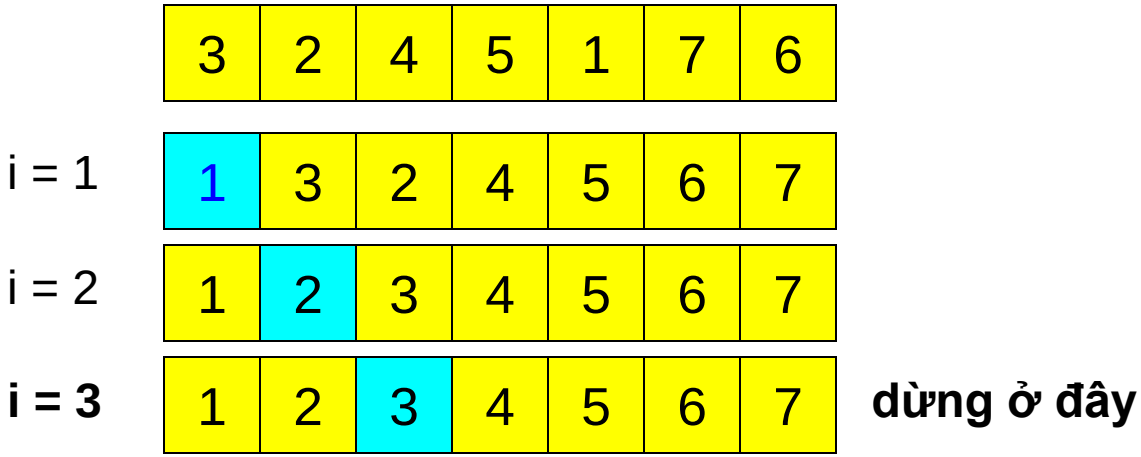
```
void selectionSort(int A[], int N) {
    int m;
    for (int i=0; i < N-1; i++){
        m=i;
        for (int k=i+1; k < N; k++)
            if (A[k] < A[m]) m=k;
        if (m != i) swap(A[i],A[m]);
    }
}
```

Sắp xếp nổi bọt

- Ý tưởng giải thuật: đây là một giải thuật cải tiến so với GTSX chọn, bằng cách thay đổi cách tìm và đưa phần tử nhỏ nhất về đầu dãy ở mỗi bước
  - Đầu vào: dãy N số a_1, a_2,…, a_N
  - Đầu ra: dãy vào đã được sx theo chiều tăng dần
  - Giải thuật:
    - GT thực hiện trong **tối đa** là N-1 bước, đánh số các bước i = 1, 2, 3,…
    - Ở bước thứ i, so sánh lần lượt N-i cặp số (a_k và a_{k+1}), với k = N-1, N-2,…, i; sao cho nếu $a_k > a_{k+1}$ thì hoán đổi a_k và a_{k+1}
    - Nếu ở một bước i nào đó mà ta không phải hoán đổi cặp nào thì chứng tỏ dãy đã sắp xếp, nên dừng GT ở đây.

Minh họa hoạt động của GT

- G/s cho dãy N = 7 số



Mô tả tựa lập trình

```
i = 1;
sorted = False;
while (!sorted && i<N) {
    sorted = True;
    for (k=N-1; k>=i; k--)
        if (a_k > a_k+1) {
            swap(a_k, a_k+1);
            sorted = False;
        }
    i++;
}
```

Sắp xếp nổi bọt – Cài đặt hàm

```
void bubbleSort(int A[], int N) {
    int i = 0;
    bool sorted = false;
    while (!sorted && i<N-1) {
        sorted = true;
        for (k=N-2; k>=i; k--)
            if (A[k] > A[k+1]) {
                swap(A[k], A[k+1]);
                sorted = false;
            }
        i++;
    }
}
```

Sắp xếp chèn

- Ý tưởng giải thuật: tiến hành xây dựng dãy được sắp xếp có số phần tử tăng dần, ban đầu là 1 phần tử, sau đó là 2 phần tử, … và cuối cùng đủ N phần tử được sắp xếp. Ở mỗi bước, ta cần chèn một phần tử trong dãy chưa được sắp xếp vào một dãy các phần tử đã được sắp xếp, sao cho sau khi chèn thì dãy mới cũng được sx.
  - Đầu vào: dãy N số a_1, a_2,…, a_N
  - Đầu ra: dãy vào đã được sx theo chiều tăng dần
  - Giải thuật:
    - GT thực hiện trong N-1 bước, đánh số các bước i = 1, 2,…, N-1
    - Ở bước thứ i, ta được dãy (a_1, a_2,…, a_i) đã được sắp xếp, ta cần chèn phần tử a_{i+1} vào dãy trên sao cho sau khi chèn ta được dãy mới cũng được sx.

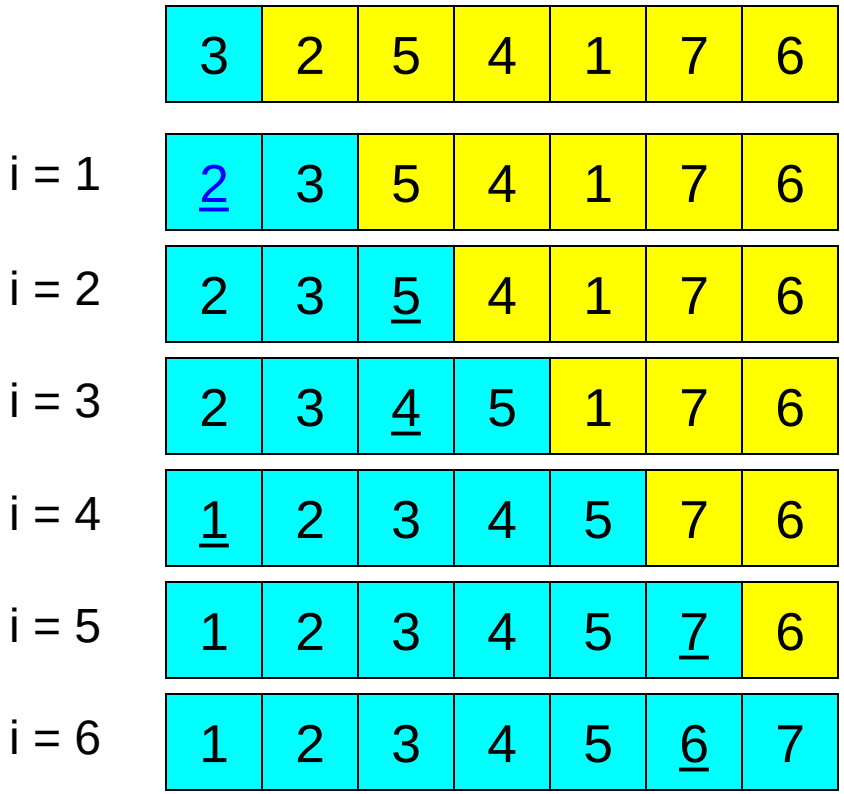
Giải thuật chèn

- **Đầu vào:**
  - Dãy (a_1, a_2,…, a_i) đã được sắp xếp, và phần tử cần chèn b
- **Đầu ra:**
  - Dãy (a_1, a_2,…, a_i, a_{i+1}) cũng được sx.
- **Giải thuật:**
  - Tìm vị trí k cần chèn trong dãy ban đầu thỏa mãn các điều kiện sau:
    - $1 \le k \le i+1$
    - $b < a_1$ hoặc $a_i \le b$ hoặc $a_{k-1} \le b < a_k$
  - Dịch các phần tử (a_k, a_{k+1},…, a_i) sang phải một vị trí để nhường chỗ cho phần tử b
  - Chèn b vào vị trí k
- **Lưu ý:** để kết hợp việc tìm vị trí k và dịch các phần tử sang phải, ta sẽ tiến hành tìm từ cuối dãy trở về đầu. Ban đầu so sánh b với a_{i+1}, sau đó với a_i,…, rồi cứ lùi dần cho đến khi điều kiện vị trí ở trên thỏa mãn. Đồng thời ở mỗi bước trên ta lại dịch được một phần tử vừa so sánh với b sang bên phải 1 vị trí.

Mô tả tựa lập trình cho GTSX chèn

```
for (i=1; i<N; i++){
   k = i;
   b = a_{i+1};
   while (k>=1 AND b<a_k){
        a_{k+1} = a_k;
        k--;
   }
   a_{k+1} = b;
}
```

Minh họa hoạt động của GT



Sắp xếp chèn – Cài đặt hàm

```
void insertionSort(int A[], int N) {
  int i,k,b;
  for (i=0; i<N-1; i++){
    k = i;
    b = A[i+1];
    while (k>=0 && b<A[k]){
        A[k+1] = A[k];
        k--;
    }
    A[k+1] = b;
  }
}
```

Tổng kết về các giải thuật sắp xếp cơ bản

- Ý tưởng giải thuật khá đơn giản
- Độ phức tạp tính toán: $0(n^2)$

Các giải thuật sắp xếp nâng cao

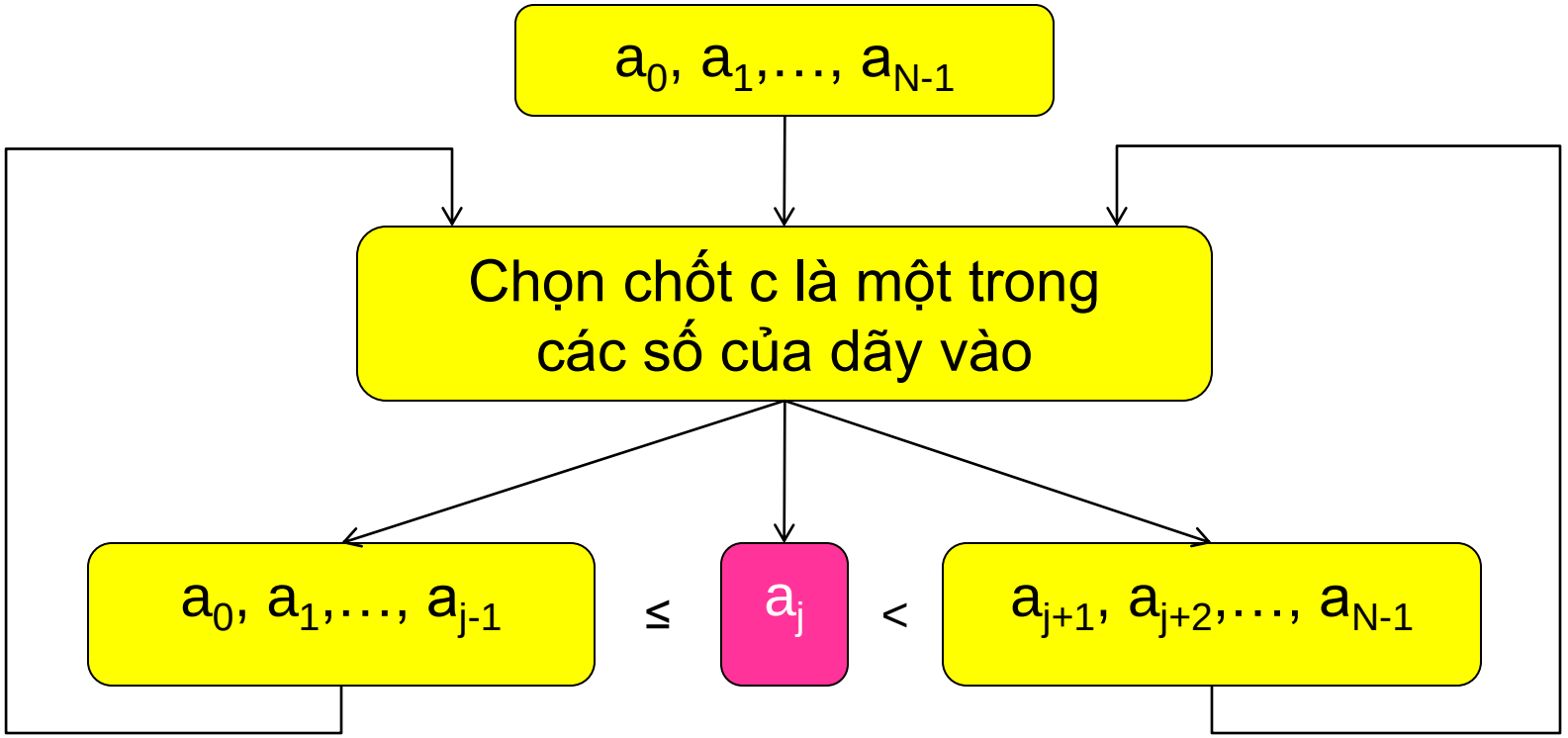
- Sắp xếp nhanh (Quick Sort)
- Sắp xếp trộn (Merge Sort)
- Sắp xếp vun đống (Heap Sort)

Sắp xếp nhanh (QuickSort)

- Ý tưởng giải thuật:
  - **Bước 1**: Chọn ra một phần tử bất kỳ trong dãy gọi nó là chốt ***c*** (pivot). Thông thường là chọn phần tử đầu dãy.
  - **Bước 2**: Đưa chốt vào đúng vị trí của nó trong dãy sẽ được sắp xếp, g/s gọi đó là vị trí j. Đồng thời bố trí các phần tử còn lại của danh sách sao cho tất cả các phần tử đứng trước chốt đều nhỏ hơn hoặc bằng ***c***, và tất cả các phần tử đứng sau chốt đều lớn hơn ***c***. Bước 1 và 2 còn được gọi là quá trình **Phân đoạn** (partition), nên giải thuật này còn được gọi là **Sắp xếp Phân đoạn**
  - **Bước 3**: Lặp lại giải thuật trên cho dãy đứng trước chốt và dãy đứng sau chốt cho đến khi toàn bộ dãy được sx.

Sắp xếp nhanh

- Mô tả ý tưởng giải thuật



Giải thuật phân đoạn

- **Đầu vào:** Dãy (a_f, a_{f+1},…, a_h)
- **Đầu ra:** Dãy (a_f, a_{f+1},…, a_{j-1}, a_j, a_{j+1}, …, a_h) sao cho:

$$(a_f, a_{f+1},\ldots, a_{j-1}) \leq a_j < (a_{j+1}, \ldots, a_h)$$

- **Giải thuật:**
  - Chọn chốt c = a_f;
  - Dùng 2 biến chạy i = f+1; và j=h
  - Đưa các phần tử khác chốt vào đúng vị trí:

```
Chừng nào (i ≤ j) {
    Chừng nào (a_i ≤ c) i++;
    Chừng nào (a_j > c) j--;
    Nếu (i<j) Hoán Đổi (a_i,a_j);
}
```

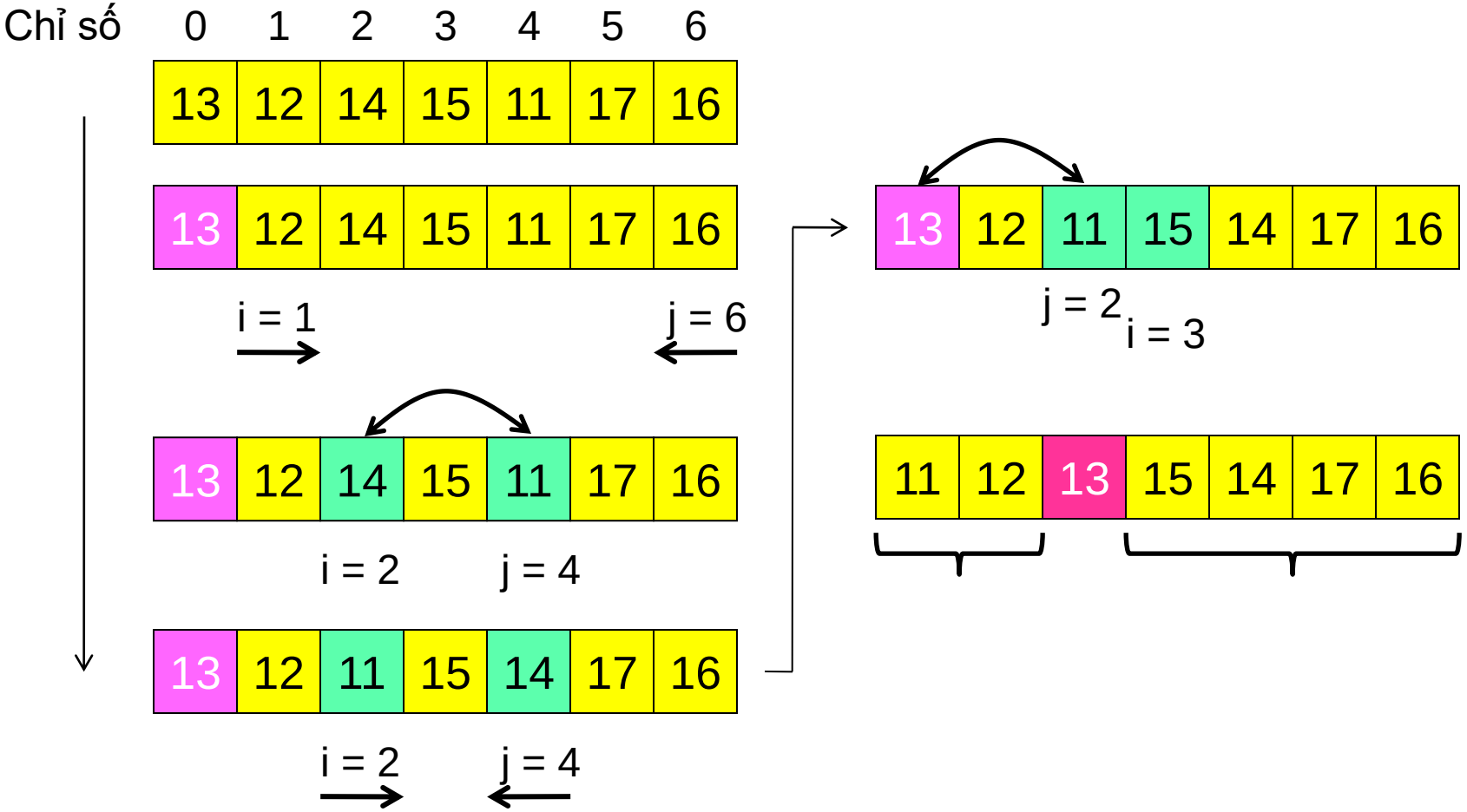
  - Đưa chốt vào đúng vị trí: Hoán đổi (a_f, a_j);
  - **Lưu ý**: giải thuật này dừng lại khi dãy chỉ có 1 hoặc 0 có phần tử nào

Sắp xếp nhanh – Cài đặt

- Thủ tục Phân đoạn

```
void Partition(int A[], int first, int last){
        if (first>=last) return;
        int c=A[first];
        int i=first+1,j=last;
        while (i<=j){
                while (A[i]<=c && i<=j) i++;
                while (A[j]>c && i<=j) j--;
                if (i<j) swap(A[i],A[j]);
        }
        swap(A[first],A[j]);
        Partition(A, first,j-1);
        Partition(A, j+1,last);
}
```

Ví dụ


Chỉ số 0 1 2 3 4 5 6
13 12 14 15 11 17 16
13 12 14 15 11 17 16
i = 1
j = 6
13 12 14 15 11 17 16
i = 2
j = 4
13 12 11 15 14 17 16
i = 2
j = 4
13 12 11 15 14 17 16
j = 2
i = 3
11 12 13 15 14 17 16

Sắp xếp nhanh – Cài đặt

- Hàm QuickSort

```
void QuickSort(int A[], int N){
      Partition(A,0,N-1);

}
```

Sắp xếp trộn (Merge Sort)

- Ý tưởng giải thuật: sử dụng giải thuật đệ quy như sau:
  - Chia dãy ban đầu thành hai dãy con có kích thước khác nhau không quá 1
  - Sắp xếp hai dãy con trên
  - Trộn hai dãy con đã sắp xếp để được dãy ban đầu cũng được sắp xếp
  - **Điểm dừng**: khi kích thước của dãy **không** > 1

Giải thuật trộn

- Đầu vào:
  - 2 dãy A = (a_0, a_1,…, a_{m-1}) và B = (b_0, b_1,…, b_{n-1}) đã được sắp xếp tăng dần
- Đầu ra:
  - Dãy C = (c_0, c_1,…, c_{m+n-1}) là kết quả trộn của A và B và cũng được sắp xếp
- Giải thuật:
  - Khởi tạo: dùng 2 biến chạy i = j = 0; C = rỗng
  - Chừng nào (i<m và j<n) // 2 dãy A và B còn chưa được chạy hết
    - Nếu a_i<=b_j thì { C = C + a_i ; i++; }
    - Trái lại, thì { C = C+b_j ; j++; }
  - Nếu i ≥ m (dãy A đã được chạy hết) thì C = C + (b_j, b_{j+1},…, b_{n-1})
  - Trái lại thì C = C + (a_i, a_{i+1},…, a_{m-1})

Giải thuật trộn – Cài đặt

- Thủ tục trộn

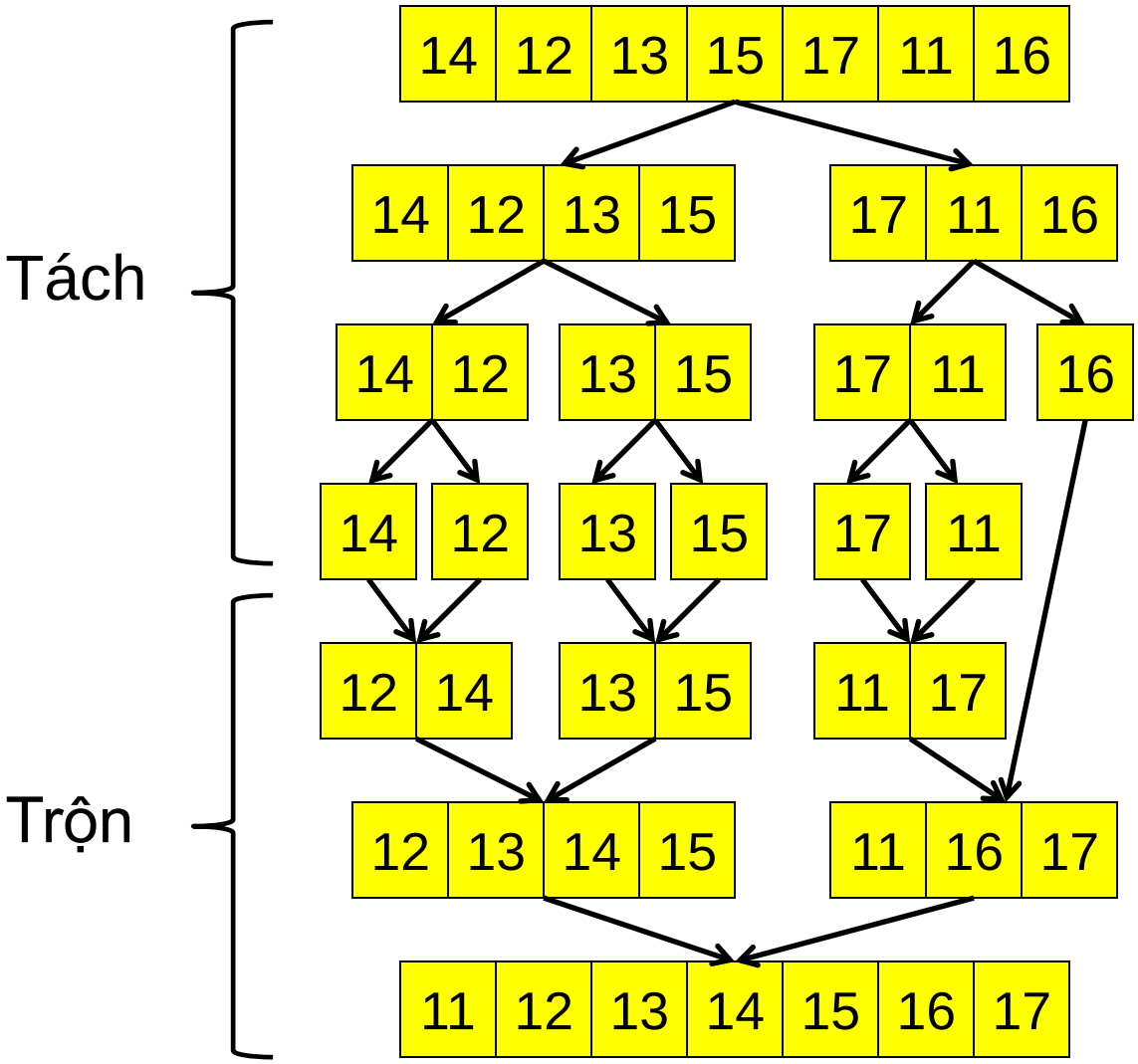
```
//Tron 2 day con ma da duoc sap xep trong A
//L1=A[m],A[m+1],...,A[n]; //L2=A[n+1],A[n+2],...,A[p]
void MergeArrays(int A[],int m, int n, int p){
        int i=m,j=n+1;
        while (i<j && j<=p){
                if (A[i]<=A[j]) i++;
                else {//chen Aj vao vi tri i
                        int x=A[j];
                        for (int k=j-1;k>=i;k--)
                                A[k+1]=A[k];
                        A[i]=x;
                        i++; j++;
                }
        }
}
```

Giải thuật trộn – Cài đặt

- Các thủ tục sắp xếp trộn

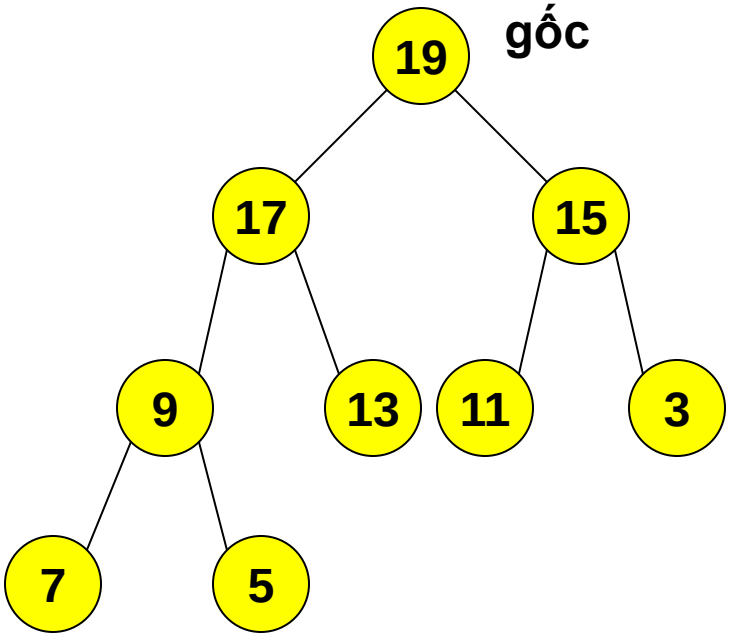
```
void Merge(int A[], int first, int last){
        if (first>=last) return;
        int m=(first+last)/2;
        Merge(A,first,m);
        Merge(A,m+1,last);
        MergeArrays(A,first,m,last);
}

void MergeSort(int A[], int N){
        if (N<2) return;
        Merge(A,0,N-1);
}
```

Ví dụ


Tách
Trộn
14 12 13 15 17 11 16
14 12 13 15
17 11 16
14 12
13 15
17 11
16
14
12
13
15
17
11
12 14
13 15
11 17
12 13 14 15
11 16 17
11 12 13 14 15 16 17

Sắp xếp vun đống (Heap Sort)

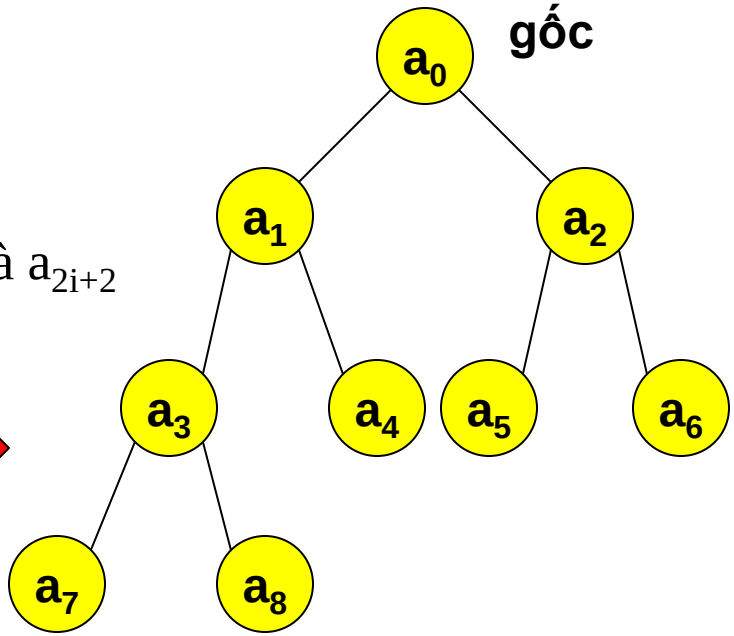
- Một số khái niệm cơ bản
  - **Đống** (Heap): Là một cây nhị phân hoàn chỉnh mà mọi cây con của nó (bao gồm cả chính cây đó) có gốc là nút có giá trị lớn nhất
  - **Vun đống**: là quá trình đưa một cây hoàn chỉnh mà chưa phải là đống chuyển thành một đống



Sắp xếp vun đống (Heap Sort)

- Quan hệ giữa cây nhị phân hoàn chỉnh và danh sách (a_0, a_1,…, a_{n-1}):
  - Coi mỗi phần tử của danh sách là một nút của cây
  - Mọi nút trong a_i :
    - Nếu có 1 con thì là nút a_{2i+1}
    - Nếu có 2 nút con thì chúng là a_{2i+1} và a_{2i+2}

$a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8$ ⟺



Sắp xếp vun đống – Ý tưởng

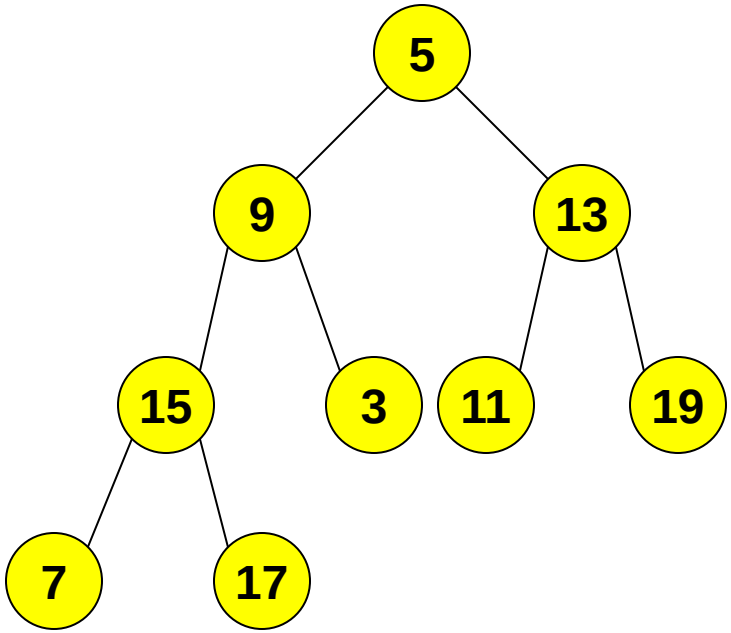
- Gồm các bước:
  - Vun đống
  - Thực hiện lặp lại trong N-1 lần (i=1..N-1)
    - Hoán đổi (a_0, a_{N-i})
    - Vun lại đống bắt đầu từ nút gốc a_0.

Giải thuật vun đống (MakeHeap)- ý tưởng

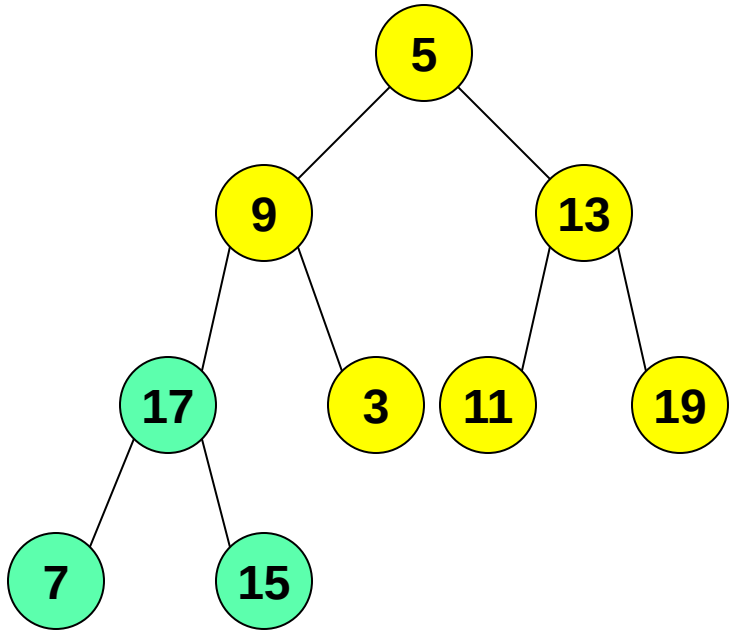
- Lần lượt vun đống cho các cây con mà có nút gốc là các nút trong theo thứ tự từ ngoài vào trong

Ví dụ minh họa

Dãy ban đầu:
5, 9, 13, 15, 3, 11, 19, 7, 17

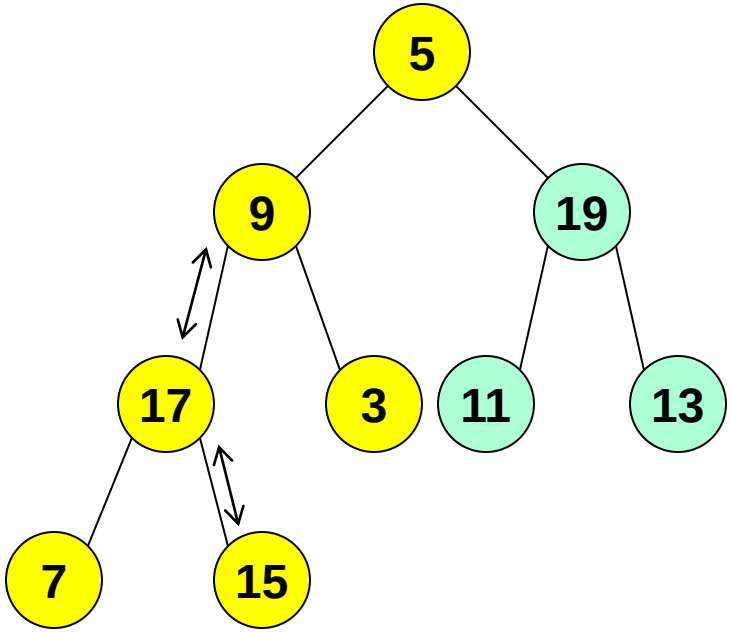


Cây ban đầu

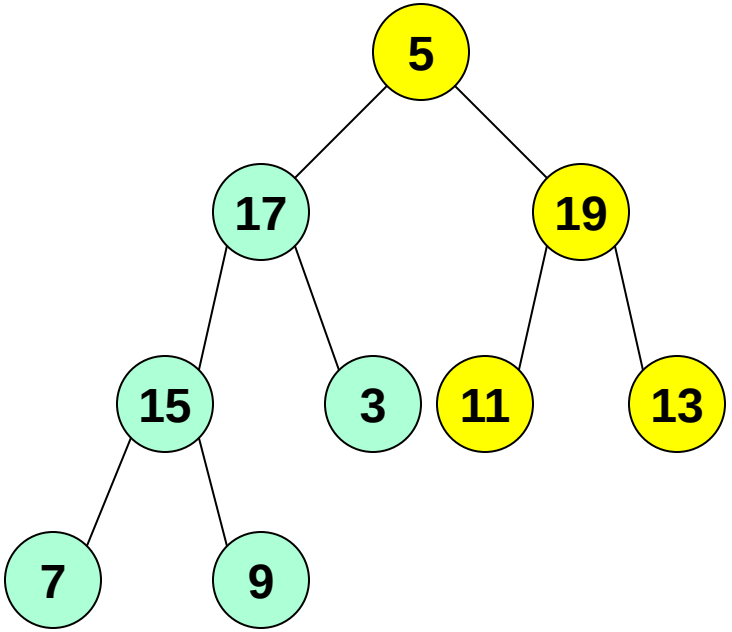


1.

Ví dụ minh họa (tiếp)

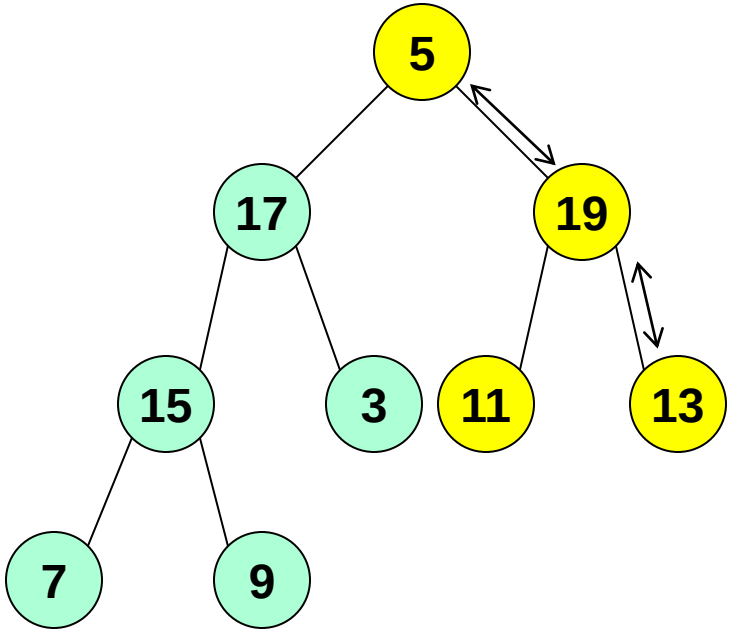


2.

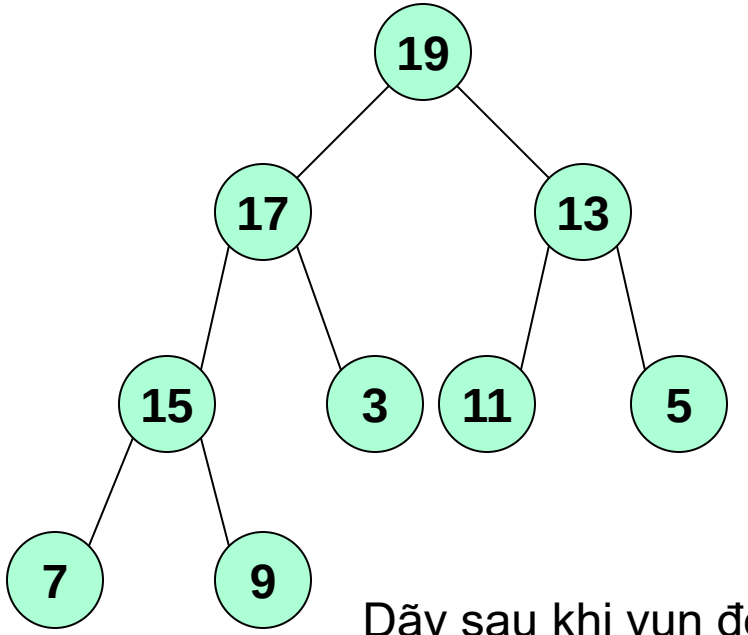


3.

Ví dụ minh họa (tiếp)



4.



Dãy sau khi vun đống:
19, 17, 13, 15, 3, 11, 5, 7, 9

5.

Giải thuật vun đống cho một cây con (MakeSubHeap)

- Giả sử cần vun đống cho một cây con có nút gốc là a_r, với điều kiện các cây con của cây con này cũng đã là các đống. Thực hiện theo các bước sau:
  - Nếu a_r là nút lá hoặc cây con tại a_r đã là đống thì dừng
  - Trái lại, tìm nút a_c là con của a_r mà có giá trị lớn nhất ($a_c > a_r$). Sau đó thực hiện:
    - Hoán đổi (a_r, a_c)
    - Vun đống cho cây con có nút gốc là a_c.

Cài đặt thủ tục MakeSubHeap

```
void MakeSubHeap(int A[], int r, int N){
   if (2*r+1 >= N) return;//Nếu r là nút lá thì kết thúc
   //Tim con ma co gia tri lon nhat
   int maxChid=A[2*r+1];//giả sử nút con trái là lớn nhất
   int c=2*r+1;
   if (2*r+2<N)//nếu tồn tại nút con phải
        if (A[2*r+2]>A[2*r+1]){
               maxChid=A[2*r+2];
               c=2*r+2;
        }
   if (A[r]<A[c]) {
        swap(A[r],A[c]);
        MakeSubHeap(A,c,N);
   }
}
```

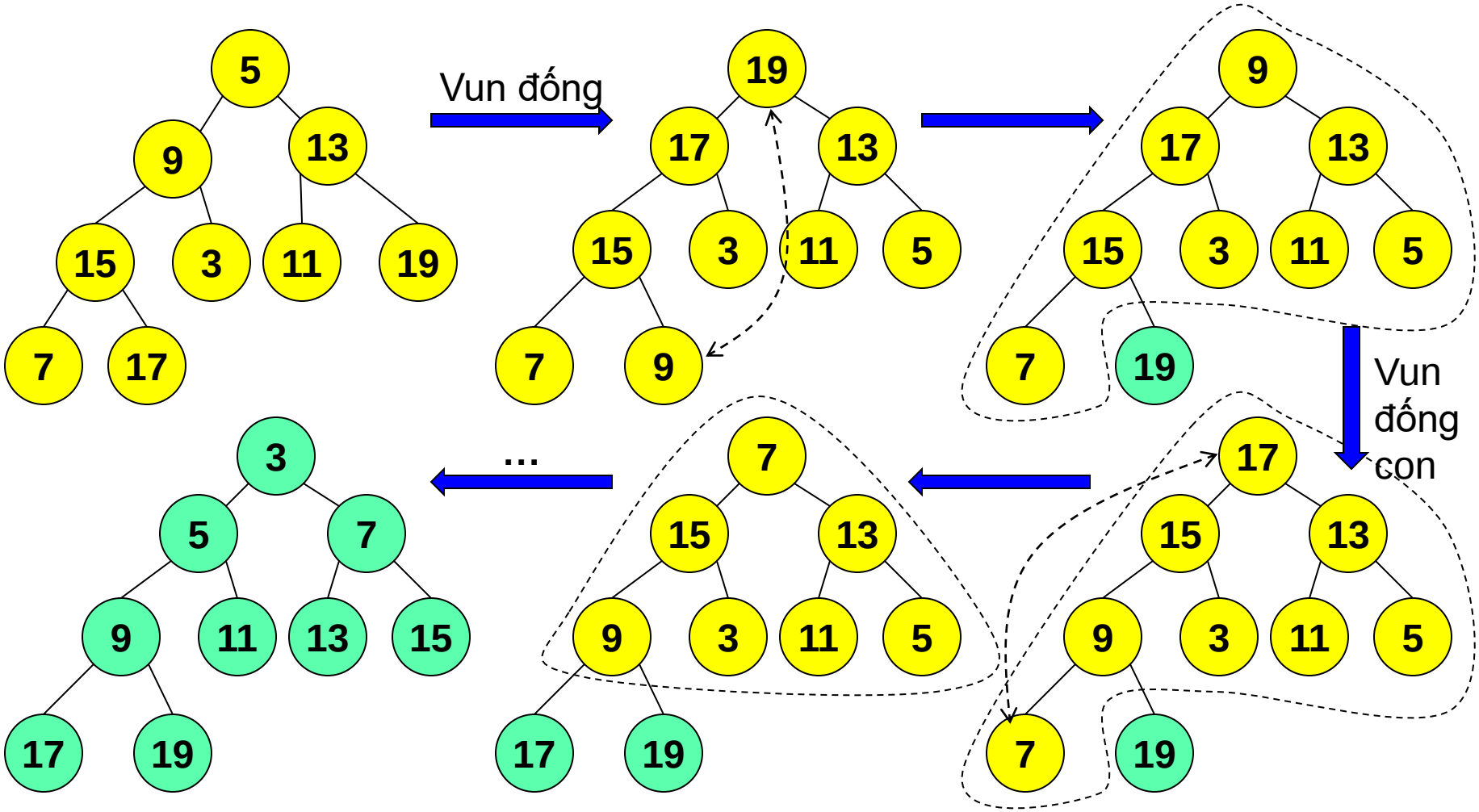
Cài đặt thủ tục vun đống (MakeHeap)

```
void MakeHeap(int A[], int N){
  if (N<2) return;
  int m=N/2-1; //Vị trí nút trong ngoài cùng
  for (int i=m;i>=0;i--)
     MakeSubHeap(A,i,N);
}
```

Cài đặt thủ tục sắp xếp vun đống HeapSort

```
void HeapSort(int A[], int N){
  if (N<2) return;
  MakeHeap(A,N);
  for (int i=1;i<N;i++){
     swap(A[0],A[N-i]);
     MakeSubHeap(A,0,N-i);
  }
}
```

Ví dụ


5
9
13
15
3
11
19
7
17
Vun đống
19
17
13
15
3
11
5
7
9
9
17
13
15
3
11
5
7
19
Vun đống con
17
15
13
9
3
11
5
7
19
7
15
13
9
3
11
5
17
19
...
3
5
7
9
11
13
15
17
19

Các giải thuật sắp xếp tuyến tính

- Sắp xếp đếm (Counting sort)
- Sắp xếp cơ số (Radix sort)

Sắp xếp đếm

- *Điều kiện dãy phần tử*: giải thuật này yêu cầu dãy phần tử phải là dãy số nguyên có giá trị nằm trong khoảng từ 0 đến k. Tức là dãy:

$$0 \leq (a_0, a_1, \ldots, a_{n-1}) \leq k$$

Sắp xếp đếm

- *Ý tưởng giải thuật:* Với mỗi phần tử a_i ($0 \le i \le n-1$) của dãy cần sắp xếp, ta cần đếm số phần tử mà nhỏ hơn hoặc bằng a_i, giả sử đó là c_i. Khi đó, vị trí của a_i trong dãy cần sắp xếp sẽ là c_i+1.
  - **Giải thuật đếm**
  - **Giải thuật đưa các phần tử vào vị trí thích hợp**

Giải thuật đếm

- *Đầu vào*: dãy A gồm n phần tử a_0, a_1,…, a_{n-1} nằm trong khoảng từ 0 đến k.
- *Đầu ra*: mảng C gồm k+1 phần tử, trong đó c_i là số phần mà bé hơn hoặc bằng i
- *Nội dung giải thuật*: gồm 2 bước chính:

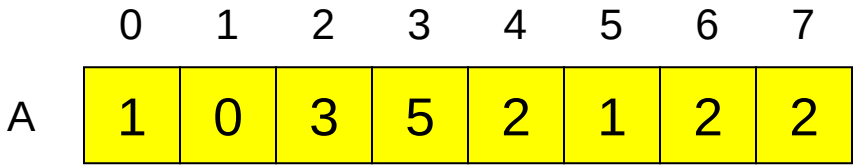
Bước 1: đếm số phần tử đúng bằng i (0 ≤ i ≤ k)

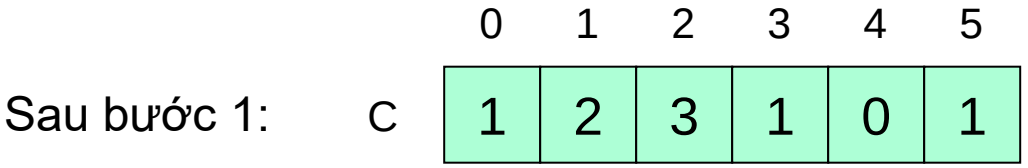
Bước 2: đếm số phần tử ≤ i

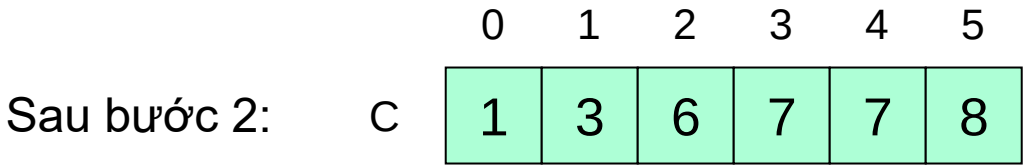
Nội dung giải thuật đếm

```
//Bước 1: đếm số phần tử đúng bằng i (0 ≤ i ≤ k)
for i=0 to k
   c[i] = 0;  //Khởi tạo mảng c
for i=0 to n-1
   c[a[i]]++;
//Bước 2: đếm số phần tử ≤ i
for i=1 to k
   c[i] = c[i] + c[i-1];
```

Ví dụ giải thuật đếm, với n=8, k=5







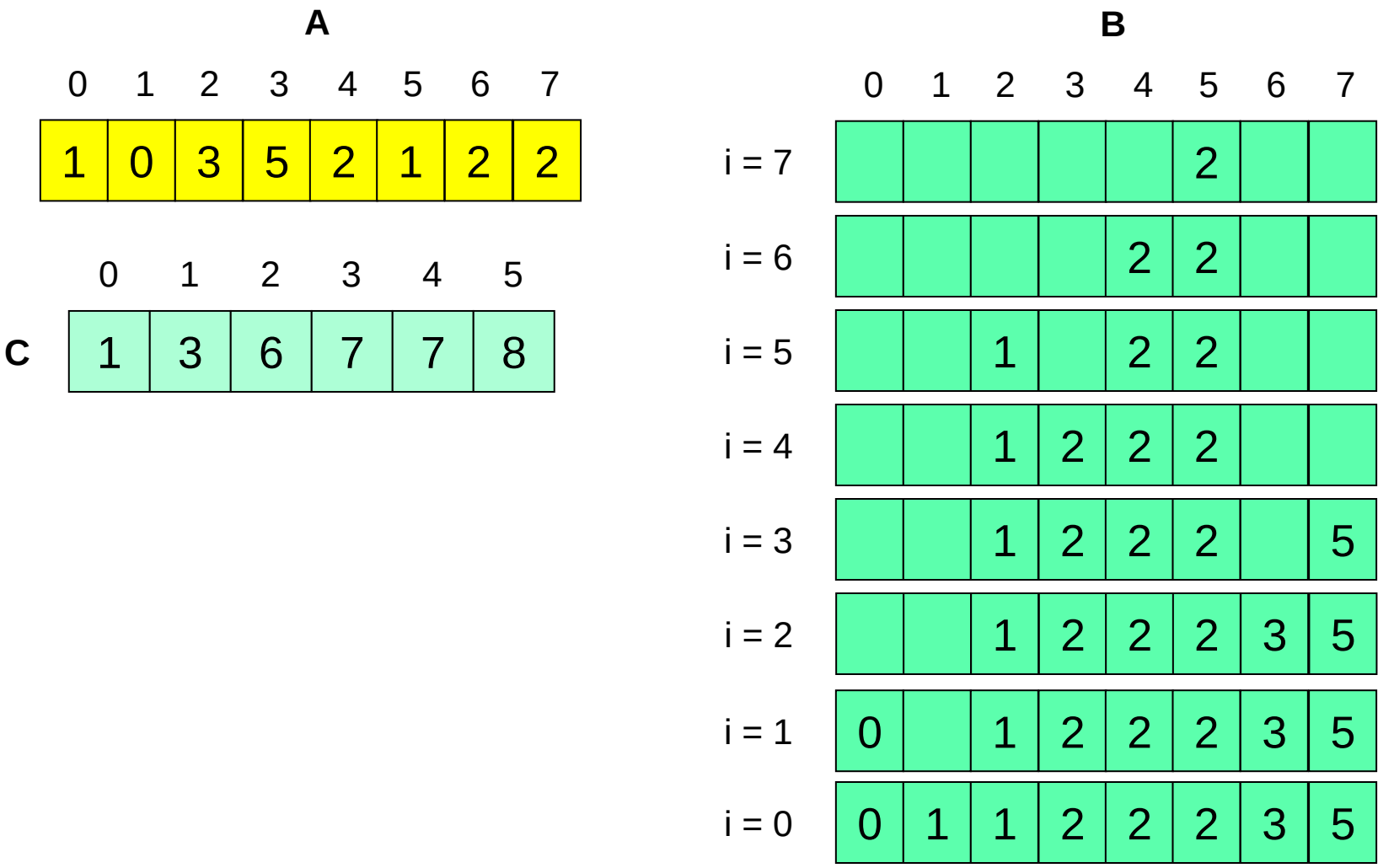
Giải thuật đưa các phần tử vào vị trí thích hợp

- *Đầu vào*:
  - Dãy A gồm n phần tử a_0, a_1,…, a_{n-1} nằm trong khoảng từ 0 đến k;
  - Mảng C gồm k+1 phần tử, trong đó c_i là số phần mà bé hơn hoặc bằng i
- *Đầu ra*: dãy B là một hoán vị được sắp xếp của dãy A.
- *Nội dung giải thuật*:
  - Đưa lần lượt các phần tử từ dãy A sang dãy B theo thứ tự từ cuối về đầu, tức là phần tử a[i] (với i = n-1, n-2, …, 0) sẽ được đưa vào vị trí b[c[a[i]]-1], vì có c[a[i]] phần tử nhỏ hơn hoặc bằng a[i].
  - Lưu ý: mỗi lần đưa được a[i] vào 1 vị trí, thì số phần tử nhỏ hơn hoặc bằng a[i] sẽ giảm đi 1.

Cài đặt giải thuật

```
for i = n-1 downto 0
   b[c[a[i]]-1] = a[i];
   c[a[i]]--;
```

Ví dụ


A
0 1 2 3 4 5 6 7
1 0 3 5 2 1 2 2
C
0 1 2 3 4 5
1 3 6 7 7 8
B
0 1 2 3 4 5 6 7
i = 7
2
i = 6
2 2
i = 5
1 2 2
i = 4
1 2 2 2
i = 3
1 2 2 2 5
i = 2
1 2 2 2 3 5
i = 1
0 1 2 2 2 3 5
i = 0
0 1 1 2 2 2 3 5

Độ phức tạp của giải thuật đếm

- Giải thuật này có độ phức tạp là O(k).

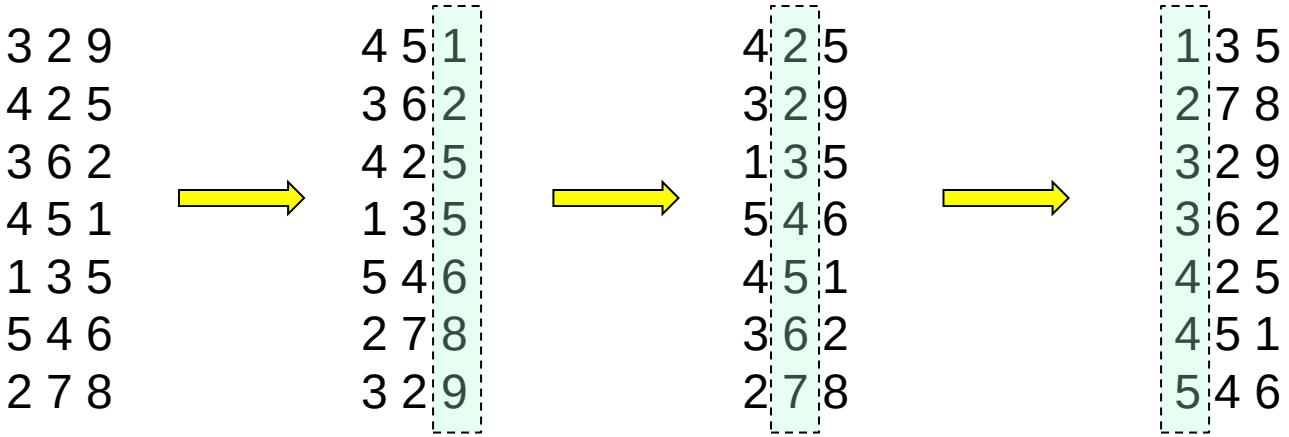
Sắp xếp cơ số (Radix sort)

- *Điều kiện dãy phần tử*: giải thuật này yêu cầu dãy phần tử phải là dãy số nguyên có cùng số chữ số, tức là dãy a_0, a_1,…, a_{n-1} cần sắp xếp giả sử đều có d chữ số.

Sắp xếp cơ số (Radix sort)

- *Ý tưởng giải thuật:* giải thuật sẽ tiến hành trong d bước, i = 1, 2, ..., d, trong đó ở bước thứ i sẽ sắp xếp các chữ số thứ i của dãy n số.
- *Lưu ý:* thứ tự các chữ số được sắp xếp là từ phải qua trái, tức là từ chữ số có bậc nhỏ nhất đến chữ số có bậc lớn nhất.

Ví dụ:


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

Độ phức tạp của giải thuật SXCS

- Giải thuật này có độ phức tạp là O(n).

Tóm tắt các giải thuật sắp xếp

- Các giải thuật sắp xếp cơ bản:
  - **Ưu điểm**: ý tưởng đơn giản, dễ cài đặt
  - **Nhược điểm**: độ phức tạp của giải thuật cao ($O(n^2)$) → tốc độ sắp xếp chậm
- Các giải thuật sắp xếp nâng cao:
  - **Ưu điểm**: độ phức tạp của giải thuật thấp ($O(n.log(n))$) → tốc độ sắp xếp nhanh
  - **Nhược điểm**: ý tưởng phức tạp, cài đặt cũng phức tạp hơn

Tóm tắt các giải thuật sắp xếp

- Các giải thuật sắp xếp tuyến tính:
  - **Ưu điểm**: ý tưởng đơn giản, dễ cài đặt, độ phức tạp của giải thuật thấp (O(n)).
  - **Nhược điểm**: đòi hỏi đầu vào đặc biệt, khó áp dụng cho dãy bất kỳ.

Bài tập

- Bài 1: cài đặt các giải thuật sắp xếp cơ bản cho danh sách được cài đặt bằng móc nối đơn thẳng
- Bài 2: cài đặt các giải thuật sắp xếp nâng cao cho danh sách được cài đặt bằng móc nối đơn thẳng

Thank you!