CÁC THUẬT TOÁN SẮP XẾP

BÀI TOÁN SẮP XẾP

- Sắp xếp danh sách học sinh theo tên
- Sắp xếp điểm trung bình xét học bổng
- Sắp xếp danh sách bài hát theo thời lượng
- Sắp xếp thứ tự ưu tiên công việc
- Sắp xếp kết quả thi theo điểm số để xác định giải
- Sắp xếp danh sách điểm thi để xác định điểm trúng tuyển

→ Thuật toán tìm kiếm hiệu quả đối với dữ liệu lớn

NỘI DUNG CHÍNH

SẮP XẾP LỰA CHỌN

SẮP XẾP NỔI BỌT

SẮP XẾP CHÈN

BÀI TOÁN SẮP XẾP

- **Bài toán:** Sắp xếp danh sách gồm n phần tử theo thứ tự không giảm
- **Input:**
  - Dãy gồm n phần tử cần sắp xếp
  - Thứ tự sắp xếp (tăng dần hoặc giảm dần)
- **Output:**
  - Dãy gồm n phần tử đã được sắp xếp theo thứ tự yêu cầu
- **Process:**
  - So sánh các phần tử trong dãy theo một quy tắc xác định
  - Thực hiện các phép hoán đổi hoặc chèn phần tử thích hợp
  - Lặp lại các bước trên cho đến khi dãy được sắp xếp

SẮP XẾP NỔI BỌT

BÀI TOÁN THỰC TẾ

Trong cuộc sống hàng ngày, chúng ta thường gặp tình huống cần phải sắp xếp một tập các đối tượng nào đó.

Ví dụ

Giả sử bạn có 5 viên bi sắt màu xanh như hình dưới đây. Các viên bi sắt này có hình dạng và kích thước giống nhau nhưng trọng lượng khác nhau. Làm thế nào để sắp xếp các viên bi sắt này theo thứ tự khối lượng tăng dần?

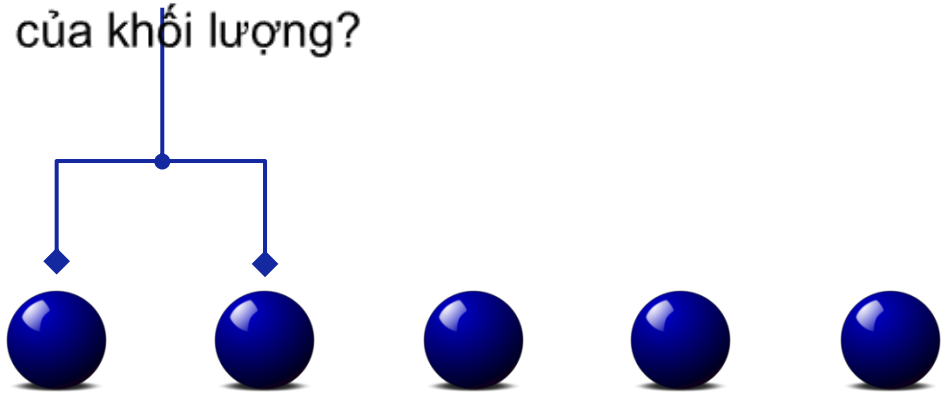
Chưa sắp xếp

Đã sắp xếp

SẮP XẾP NỔI BỌT

Sắp xếp khi bạn chỉ có thể hoán đổi hai phần tử liền kề

- Giả sử chúng ta được cung cấp một công cụ có thể thực hiện những việc sau:
  - Bạn có thể lấy hai viên bi đặt cạnh nhau và so sánh trọng lượng của chúng.
  - Hai viên bi có thể được hoán đổi vị trí cho nhau khi đặt xuống.
- Vì dụng cụ không thể nhớ được khối lượng của các viên bi nên tại một thời điểm chỉ có hai viên bi kề nhau có thể hoán đổi vị trí cho nhau. Do vậy, có thể sử dụng chiến lược đưa viên bi nặng hơn sang bên phải sau mỗi lần so sánh.
- Với chiến lược này, cần bao nhiêu phép so sánh để tất cả các viên bi được sắp xếp theo thứ tự tăng dần của khối lượng?

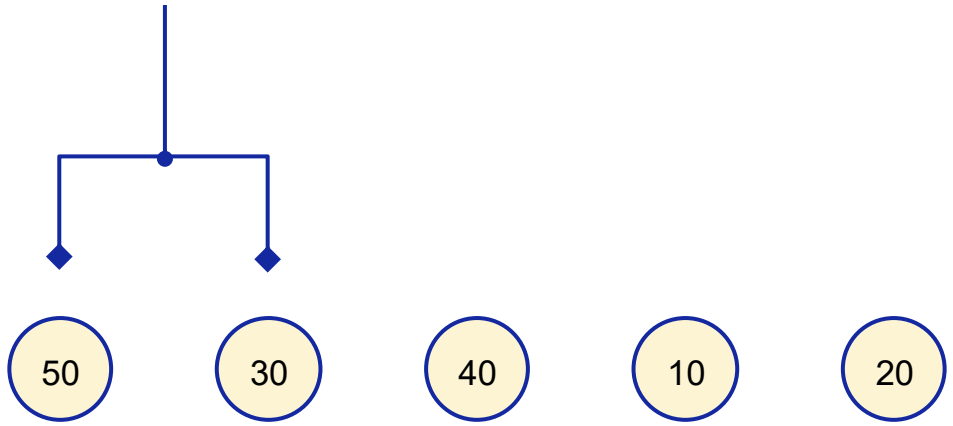


SẮP XẾP NỔI BỌT

1.1. Ví dụ về sắp xếp nổi bọt

Sắp xếp nổi bọt là một thuật toán sắp xếp thực hiện việc so sánh hai phần tử liền kề và hoán đổi vị trí của chúng nếu thứ tự bị đảo ngược (phần tử sau bé hơn phần tử trước).

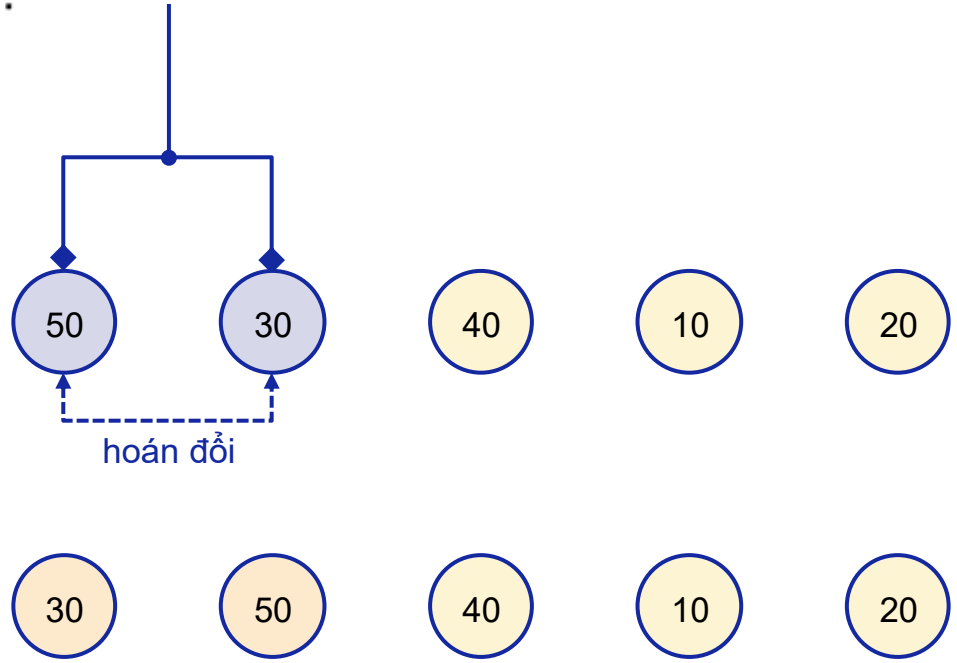
Ví dụ Giả sử chúng ta muốn sắp xếp một danh sách chứa các phần tử [50, 30, 40, 10, 20].



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

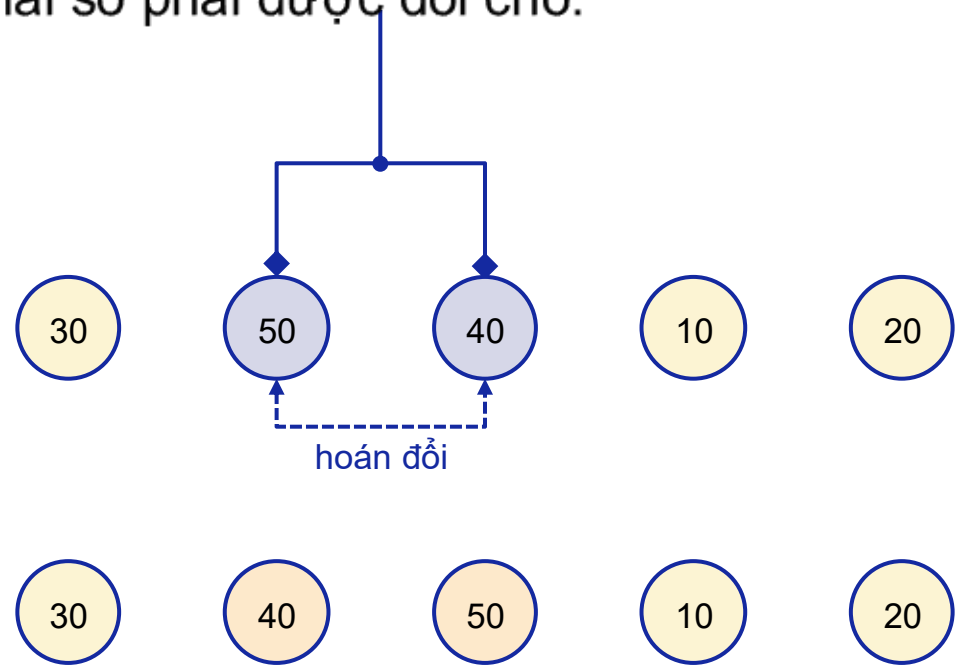
- Đầu tiên, so sánh phần tử thứ nhất 50 với phần tử thứ hai 30. Vì 50 > 30 nên đổi chỗ 2 phần tử.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

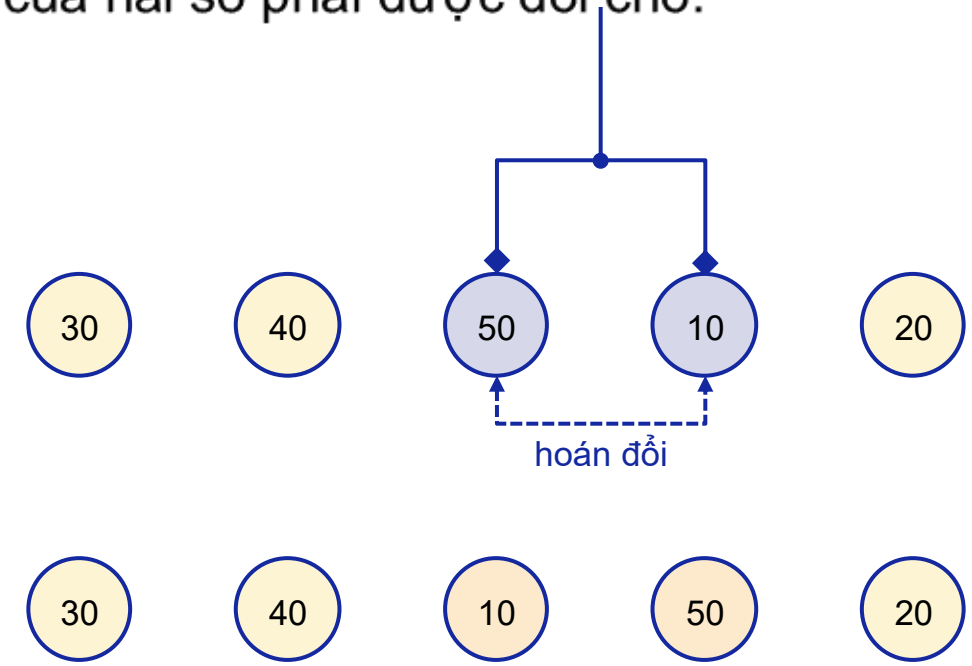
Tiếp theo, so sánh phần tử thứ hai 50 với phần tử thứ ba 40. Vì 50 lớn hơn 40 nên vị trí của hai số phải được đổi chỗ.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

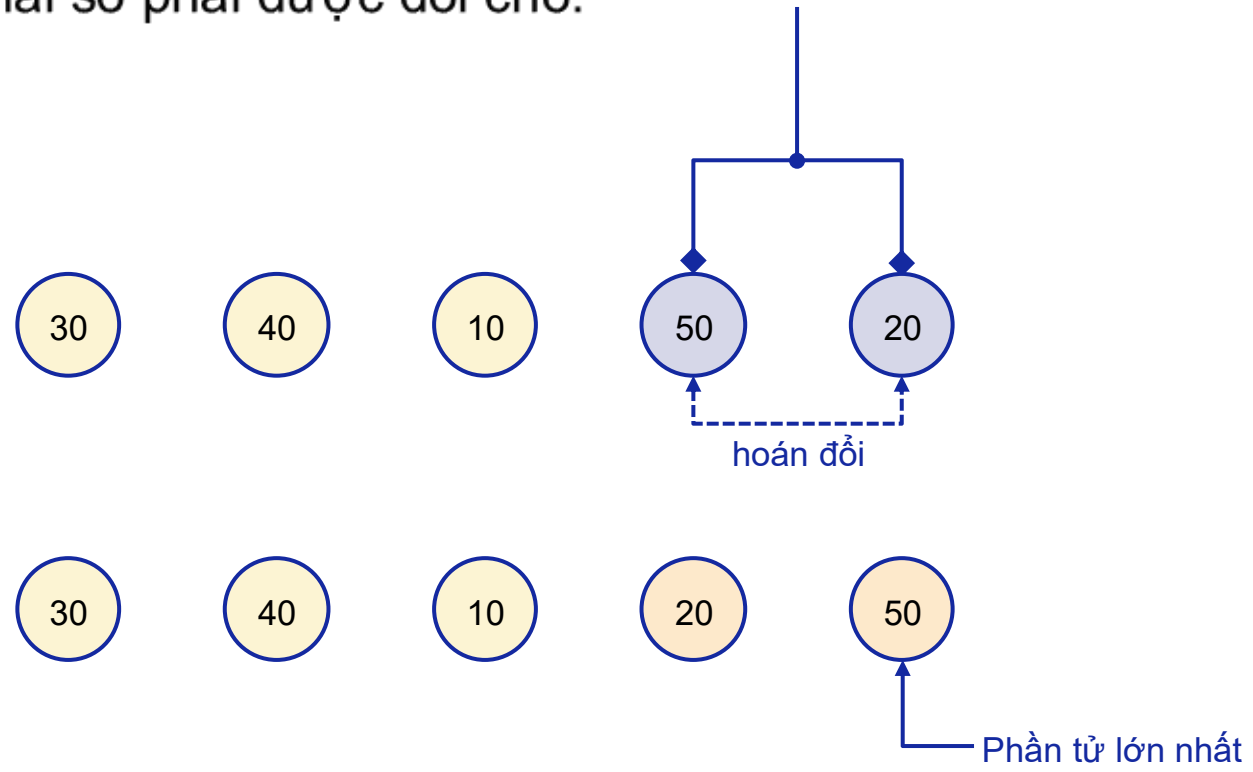
Tiếp theo, so sánh phần tử thứ ba là 50 với phần tử thứ tư là 10. Vì 50 lớn hơn 10 nên vị trí của hai số phải được đổi chỗ.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

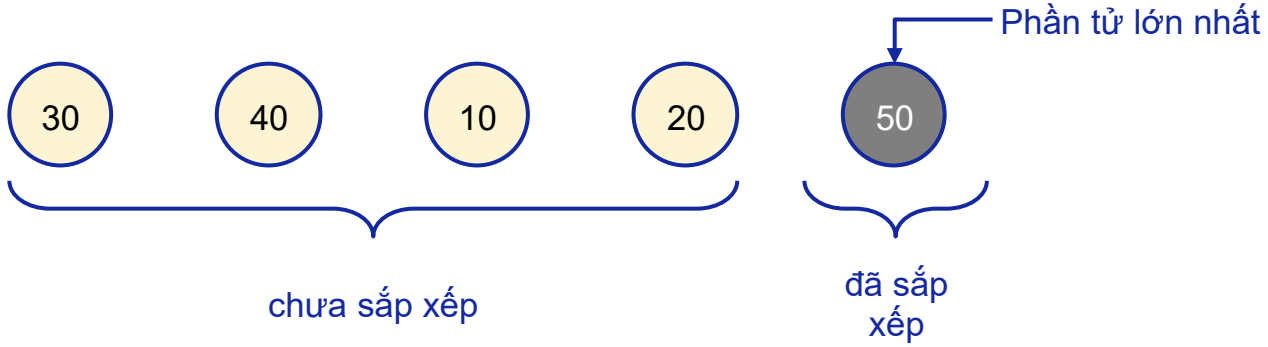
Tiếp theo, so sánh phần tử thứ tư 50 với phần tử cuối cùng 20. Vì 50 lớn hơn 20 nên vị trí của hai số phải được đổi chỗ.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

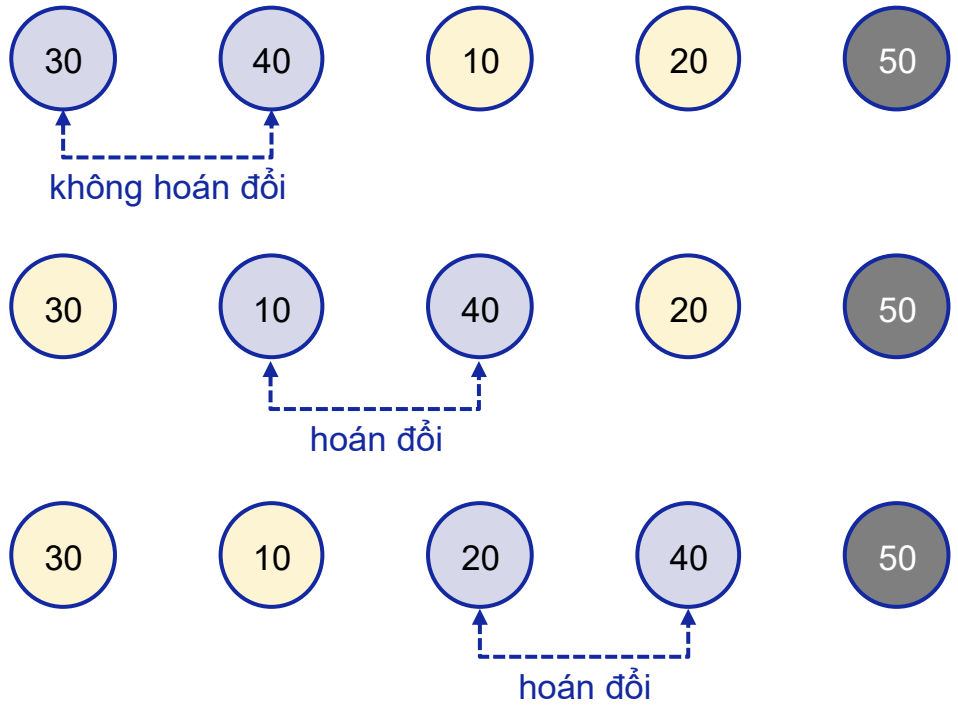
Lưu ý rằng sau bước lặp đầu tiên, phần tử lớn nhất trong danh sách (50) được đặt ở tận cùng bên phải. Phần tử cuối cùng sẽ được loại trừ khỏi bước lặp sắp xếp thứ hai.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

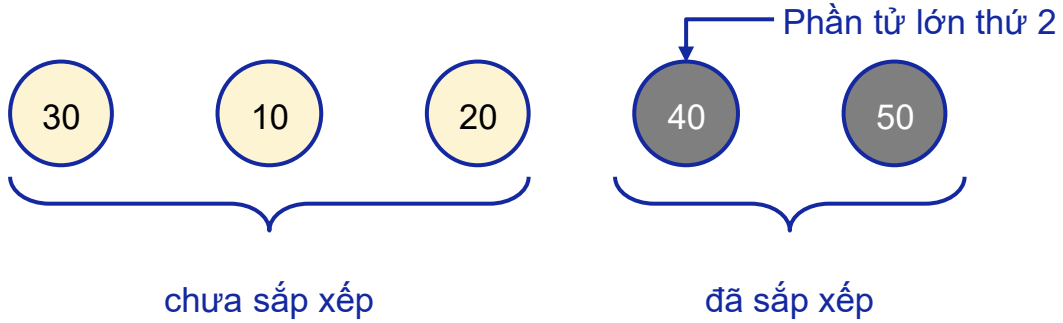
- Tiếp tục thực hiện sắp xếp nổi bọt trên danh sách chưa được sắp xếp (loại trừ phần tử cuối cùng).



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

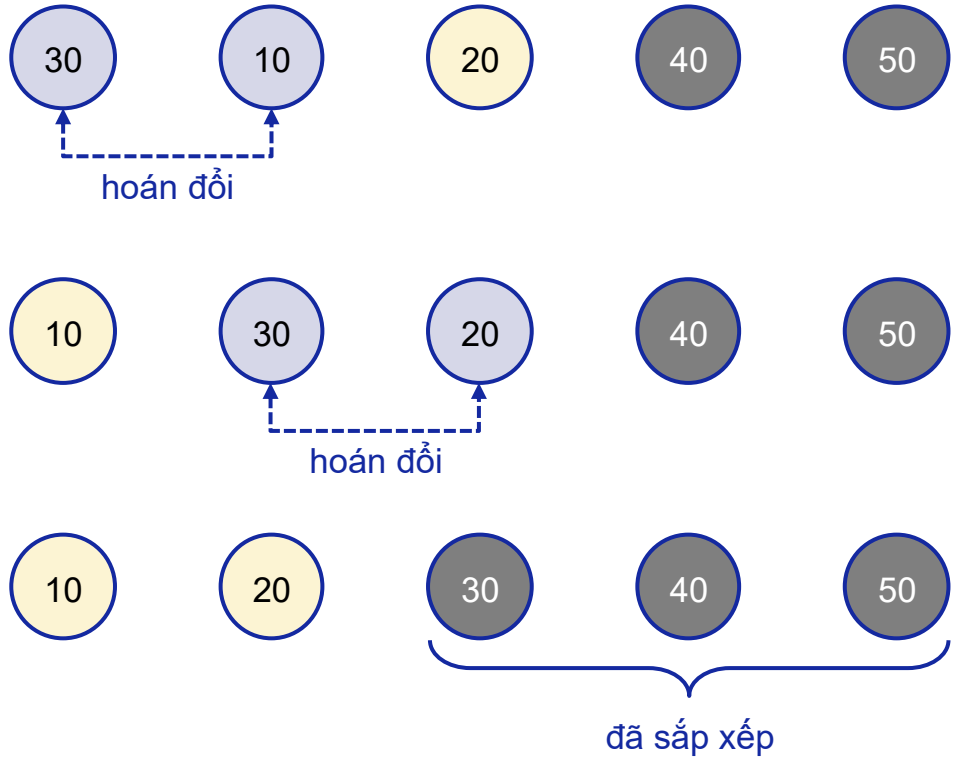
- Trong bước lặp thứ hai, đưa phần từ lớn nhất thứ hai tới cuối dãy.
- Bây giờ phần tử thứ hai này cũng có thể được loại trừ khỏi quá trình sắp xếp tiếp theo.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

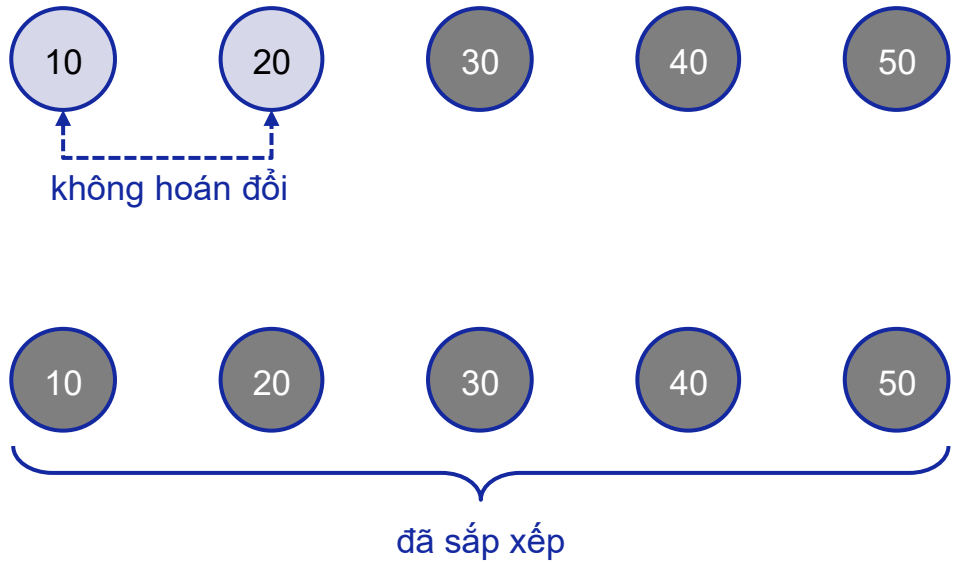
Theo cách tương tự, việc sắp xếp nổi bọt có thể được tiếp tục trong bước lặp thứ ba.



SẮP XẾP NỔI BỌT

1.2. Quá trình sắp xếp nổi bọt

Theo cách tương tự, việc sắp xếp nổi bọt được tiếp tục cho đến khi tất cả các phần tử được sắp xếp.

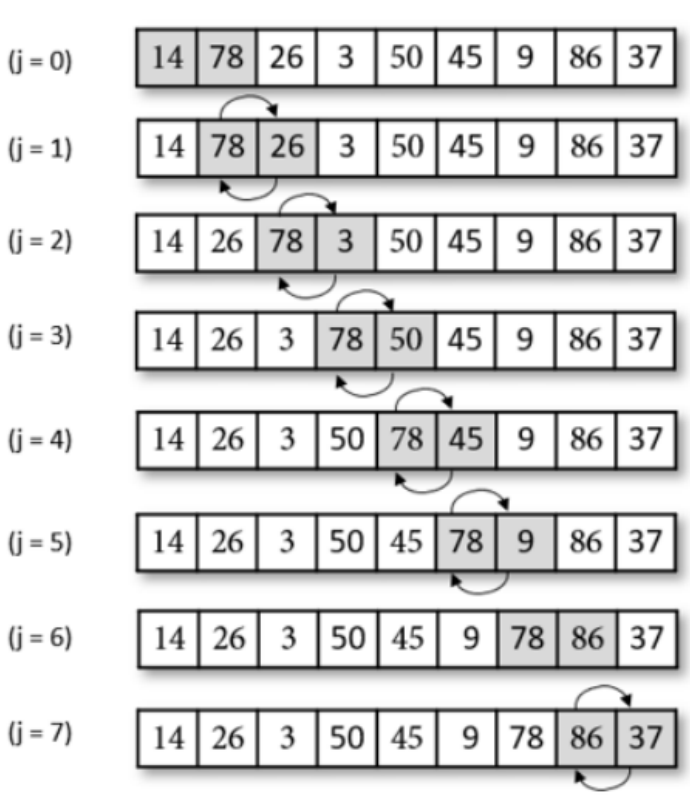


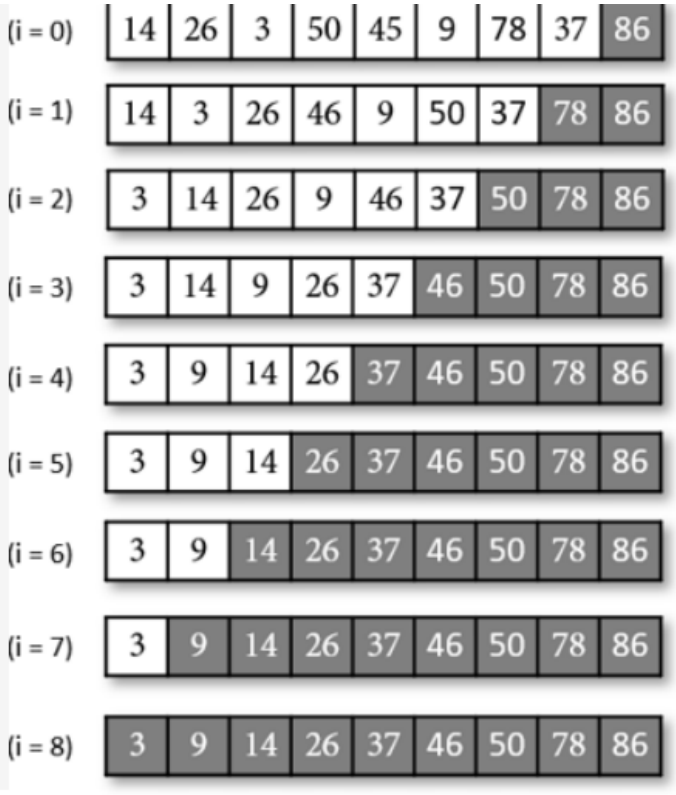
SẮP XẾP NỔI BỌT

1.2. Ý tưởng thuật toán

Thực hiện (n) bước: từ i = 0 tới (n-1)

- Tại bước thứ i, phần tử lớn nhất thứ i di chuyển tới vị trí (n-i-1)





SẮP XẾP NỔI BỌT

1.1. Hoán đổi giá trị của hai phần tử

Hai thao tác cơ bản trong thuật toán sắp xếp là thao tác so sánh và hoán đổi. Hoạt động hoán đổi có thể được thực hiện như hàm sau.

```
1 def swap1(S, x, y):
2     t = S[x]
3     S[x] = S[y]
4     S[y] = t
```

```
1 S = [10, 20]
2 print(S)
3 swap1(S, 0, 1)
4 print(S)
```

```
[10, 20]
[20, 10]
```

Dòng 1 ~ 4

- Hàm swap () nhận một danh sách S và vị trí x và y của các phần tử danh sách làm tham số đầu vào.
- Lưu giá trị của S [x] vào một biến tạm thời t và lưu giá trị của S [y] vào S [x].
- Giá trị của S [x] được lưu trữ trong biến t sau đó được lưu trữ trong S [y].

SẮP XẾP NỔI BỌT

1.2. Hoán đổi trong Python

Trong Python, bạn có thể gán giá trị cho nhiều biến cùng một lúc, vì vậy bạn có thể hoán đổi giá trị của hai biến đơn giản hơn như sau.

```
1 def swap2(S, x, y):
2     S[x], S[y] = S[y], S[x]
```

```
3 S = [10, 20]
4 print(S)
5 swap2(S, 0, 1)
6 print(S)
```

```
[10, 20]
[20, 10]
```

Dòng 1 ~ 2

- Thay các giá trị của S [y] và S [x] thành S [x] và S [y], tương ứng.
- Tại thời điểm này, cần lưu ý rằng hai giá trị có thể được hoán đổi vì giá trị bên phải được chuyển đổi thành một tuple và được gán cho giá trị bên trái.

SẮP XẾP NỔI BỌT

2.2. Thực thi sắp xếp nổi bọt

```
1  def bubblesort(S):
2      n = len(S)
3      for i in range(n):
4          print(S)
5          for j in range(n - 1):
6              if S[j] > S[j + 1]:
7                  S[j], S[j + 1] = S[j + 1], S[j]
```

```
số 8  S = [50, 30, 40, 10, 20]
9  bubblesort(S)
10 print(S)
```

```
[50, 30, 40, 10, 20]
[30, 40, 10, 20, 50]
[30, 10, 20, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```

Dòng 4

- Để kiểm tra hoạt động của sắp xếp bong bóng, trạng thái của S được xuất ra mỗi khi vòng lặp thứ i được thực hiện.

TRẮC NGHIỆM

Q1. Có bao nhiêu thao tác hoán đổi đã được thực hiện trong thủ tục sắp xếp nổi bọt dưới đây?

```
1  def bubblesort(S):
2      n = len(S)
3      for i in range(n-1):
4          for j in range(n-i-1):
5              if S[j] > S[j+1]:
6                  S[j], S[j+1] = S[j+1], S[j]
```

```
1  S = [50, 30, 40, 10, 20]
2  bubblesort(S)
3  print(S)
```

```
[50, 30, 40, 10, 20]
[30, 40, 10, 20, 50]
[30, 10, 20, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50]
```

Bài tập lập trình

Thực hành sắp xếp nổi bọt (Thực hành 10.1 – LMS)

THỜI GIAN: 20h30 – 20h45

SẮP XẾP NỔI BỌT

Mở rộng

- Độ phức tạp thời gian của thuật toán sắp xếp?
- Thuật toán sắp xếp nổi bọt sử dụng 2 vòng lặp for lồng nhau và số lần so sánh cho mỗi vòng lặp là

$$N - 1, N - 2, \ldots 2, 1, 0.$$

- Số phép so sánh trong thuật toán sắp xếp nổi bọt là $T(N) = (N - 1) + (N - 2) + \ldots + 1 = N(N-1) / 2$.
- Độ phức tạp thời gian của thuật toán sắp xếp nổi bọt là $O(N^2)$.
- Thuật toán liên tục thực hiện việc đổi chỗ nếu dãy được sắp theo thứ tự ngược lại.

SẮP XẾP LỰA CHỌN

SẮP XẾP LỰA CHỌN – BÀI TOÁN THỰC TẾ

2.2. Sắp xếp khi sử dụng cân thăng bằng

- Sử dụng một cân thăng bằng. Bạn có thể thực hiện các thao tác sau:
  - Lấy hai viên bi bất kỳ và so sánh trọng lượng của chúng.
  - Hai viên bi có thể được hoán đổi vị trí khi đặt lại.
- Cân thăng bằng cũng không thể nhớ được khối lượng của các viên bi.
- Ý tưởng: tìm viên bi có trọng lượng nhỏ nhất và đổi chỗ cho viên bi đầu tiên, và tiếp tục với viên bi nhỏ nhất thứ 2….
- Với chiến lược này, có bao nhiêu phép so sánh để sắp xếp tất cả các viên bi?

SẮP XẾP LỰA CHỌN

2.1. Ví dụ về sắp xếp lựa chọn

- Sắp xếp lựa chọn là một thuật toán sắp xếp tìm phần tử nhỏ nhất trong danh sách và đặt nó ở phía tận cùng bên trái.
- Để làm điều này, nó sử dụng một chiến lược so sánh phần tử đầu tiên của danh sách chưa được sắp xếp với phần còn lại của danh sách và hoán đổi vị trí nếu phần tử được so sánh là nhỏ hơn.

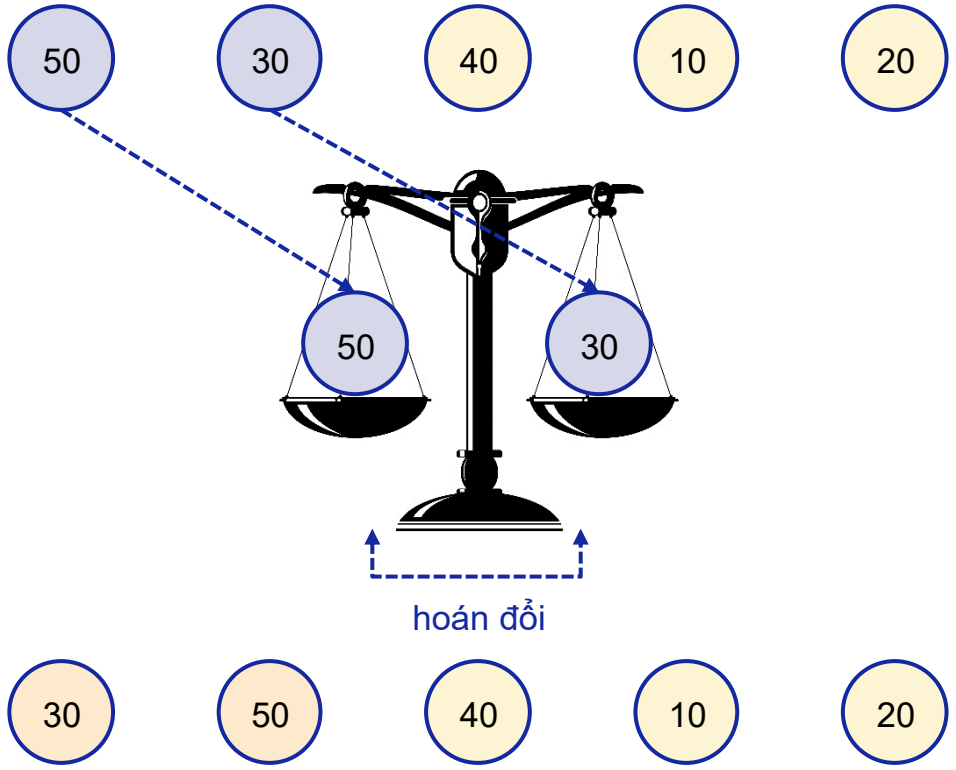
Ví dụ

Giả sử chúng ta muốn sắp xếp một danh sách gồm các phần tử [50, 30, 40, 10, 20].

SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

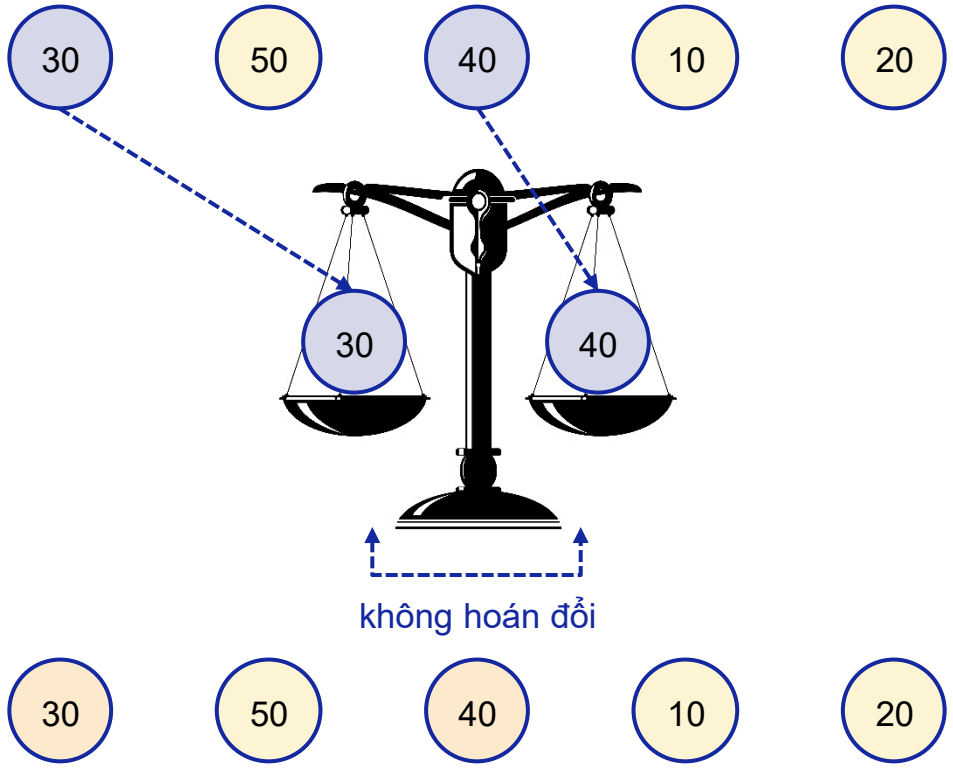
Đầu tiên, so sánh phần tử đầu tiên 50 với phần tử thứ hai 30. Vì 50 lớn hơn 30 nên hoán đổi vị trí của hai phần tử.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

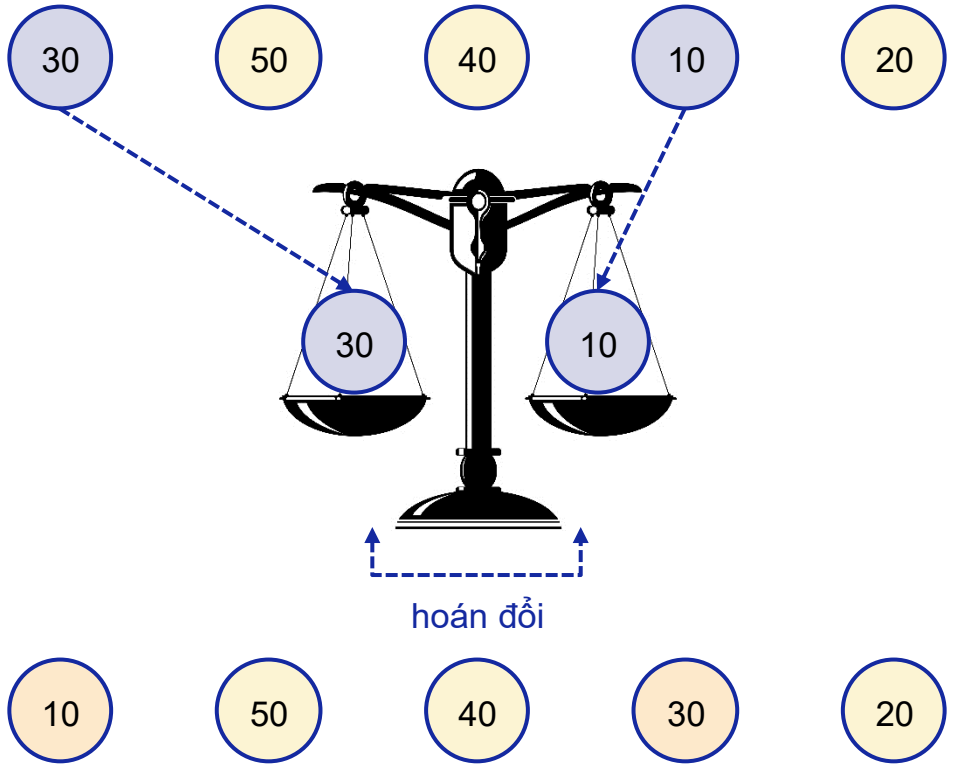
Tiếp theo, so sánh phần tử đầu tiên 30 với phần tử thứ ba 40. Vì 30 nhỏ hơn 40 nên không cần đổi vị trí của chúng.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

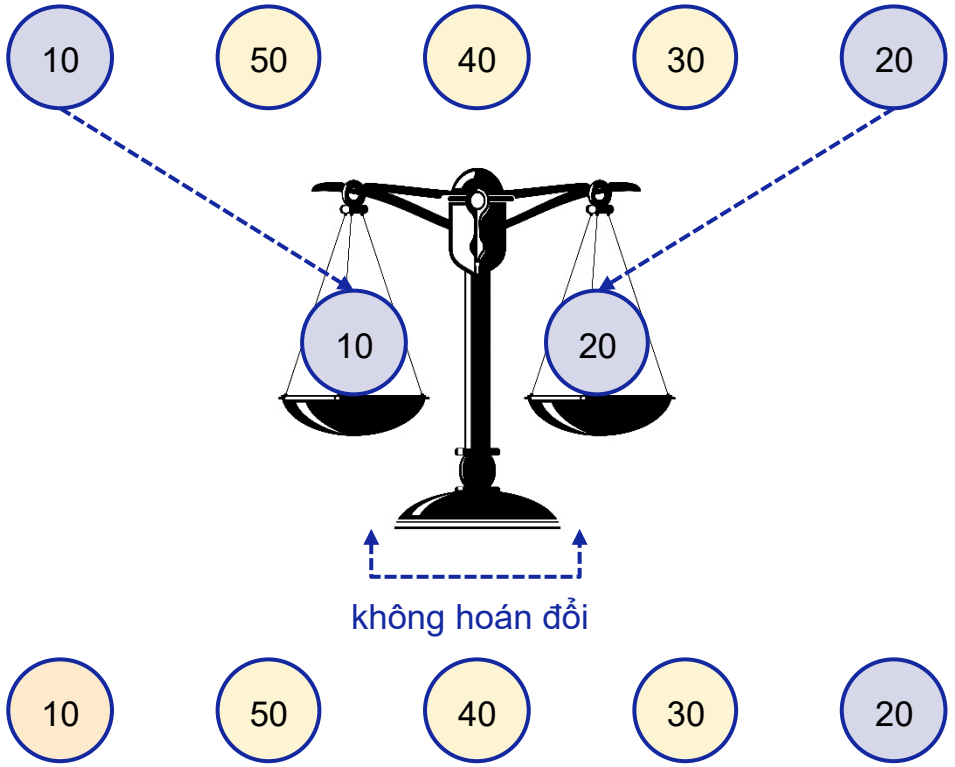
Tiếp theo, so sánh phần tử đầu tiên 30 với phần tử thứ tư 10. Vì 10 nhỏ hơn 30 nên hoán đổi vị trí của hai phần tử.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

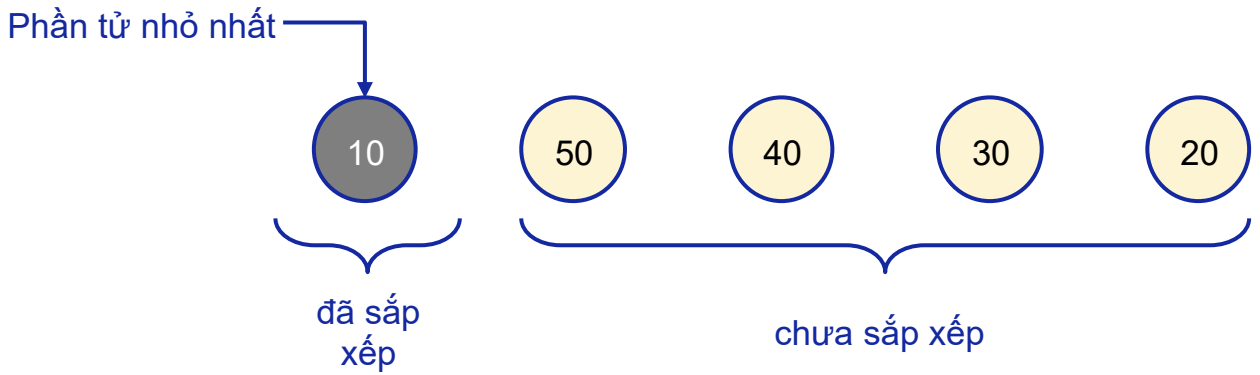
Tiếp theo, so sánh phần tử đầu tiên 10 với phần tử thứ năm 20. Vì 10 nhỏ hơn 20 nên không cần thay đổi vị trí của chúng.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

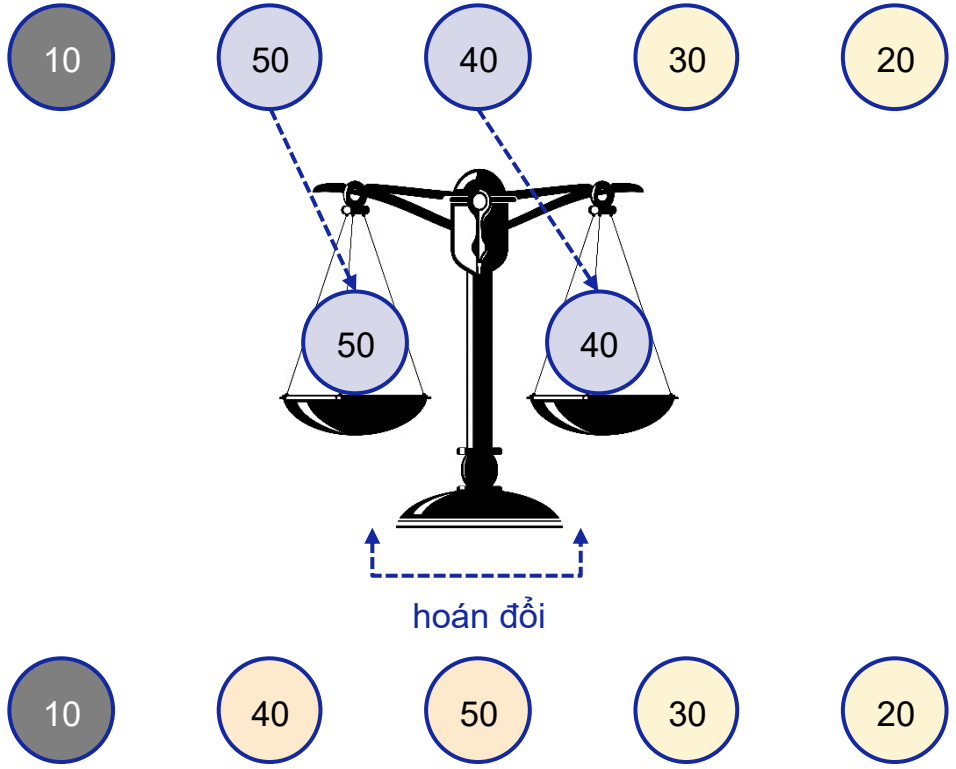
Lưu ý rằng trong sắp xếp lựa chọn, sau bước lặp đầu tiên, phần tử nhỏ nhất 10 được đặt ở đầu danh sách. Phần tử đầu tiên được loại trừ khỏi quá trình sắp xếp lựa chọn trong bước lặp tiếp theo.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

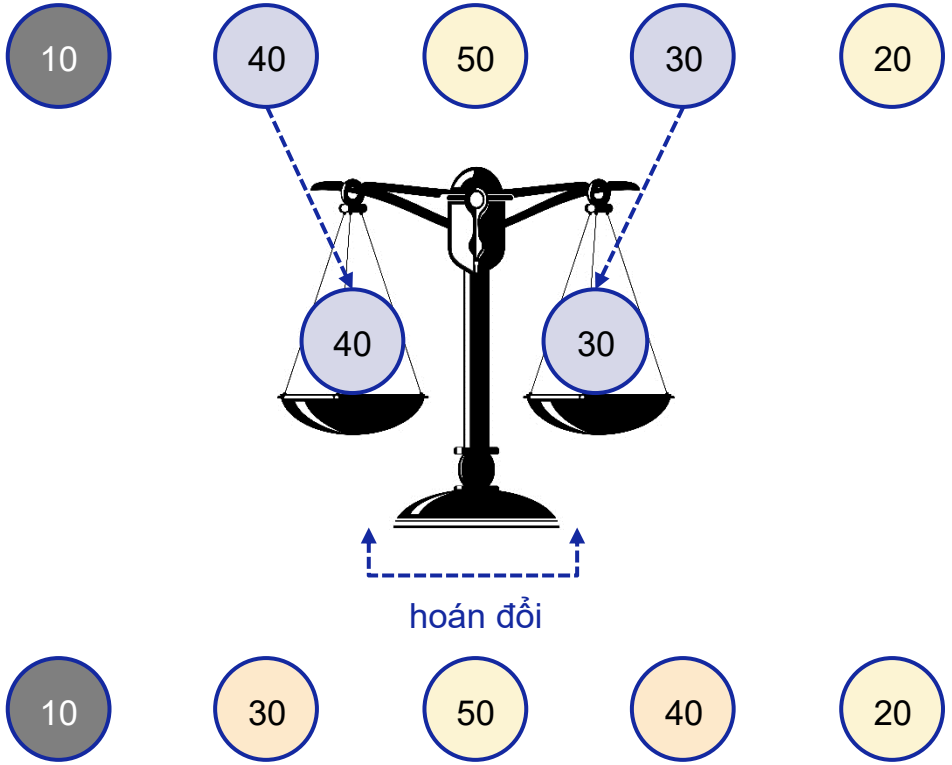
- Việc sắp xếp lựa chọn có thể tiếp tục trên danh sách chưa được sắp xếp (loại trừ phần tử đầu tiên).



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

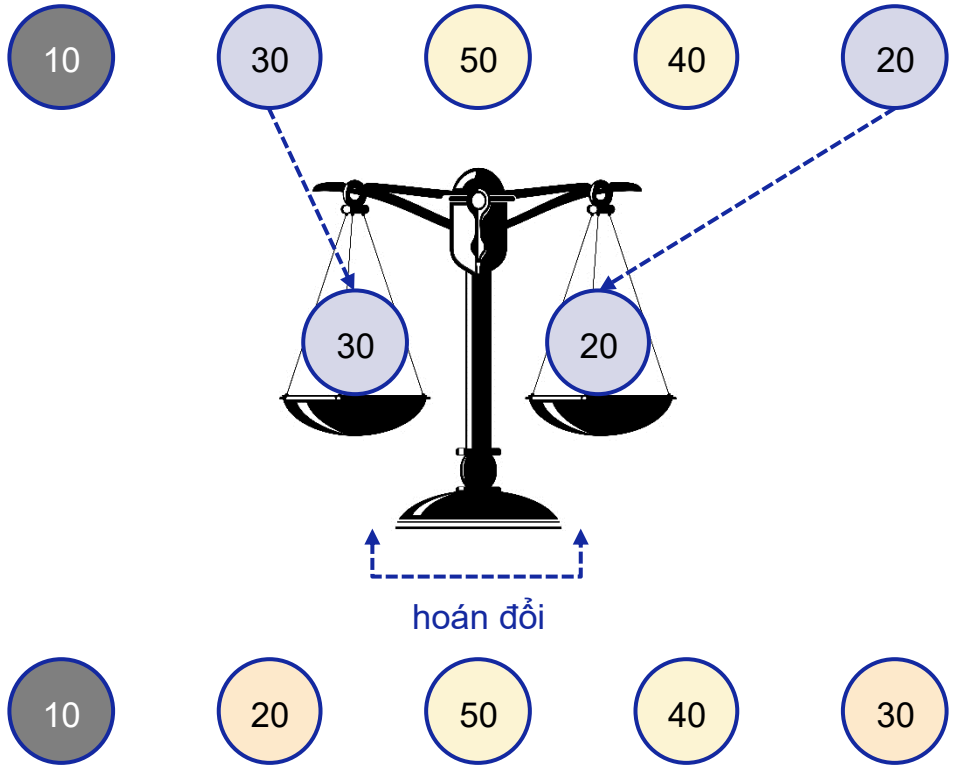
- Việc sắp xếp lựa chọn có thể tiếp tục trên danh sách chưa được sắp xếp (loại trừ phần tử đầu tiên).



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

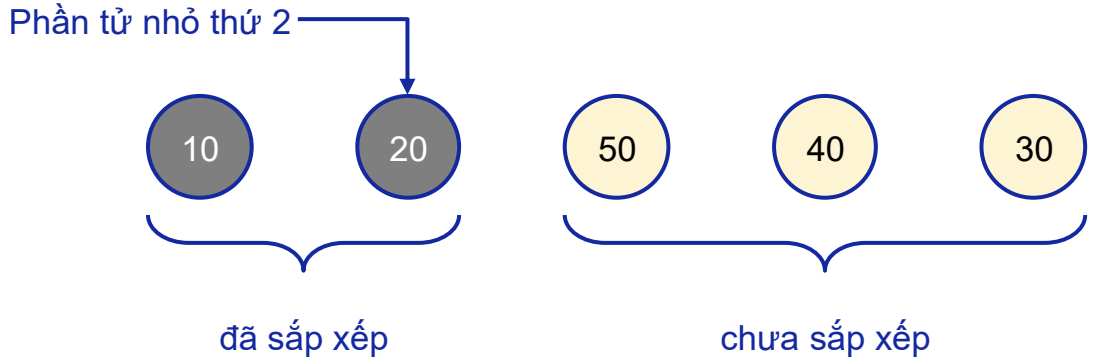
- Việc sắp xếp lựa chọn có thể tiếp tục trên danh sách chưa được sắp xếp (loại trừ phần tử đầu tiên).



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

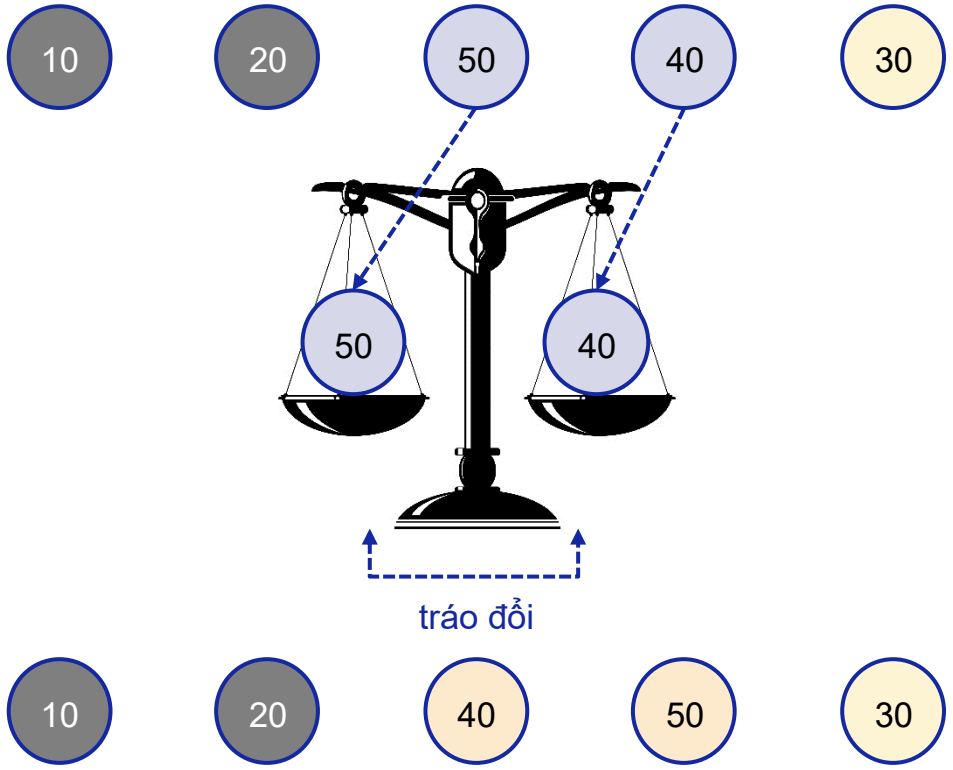
- Trong lần sắp xếp thứ hai tìm thấy phần tử nhỏ thứ hai



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

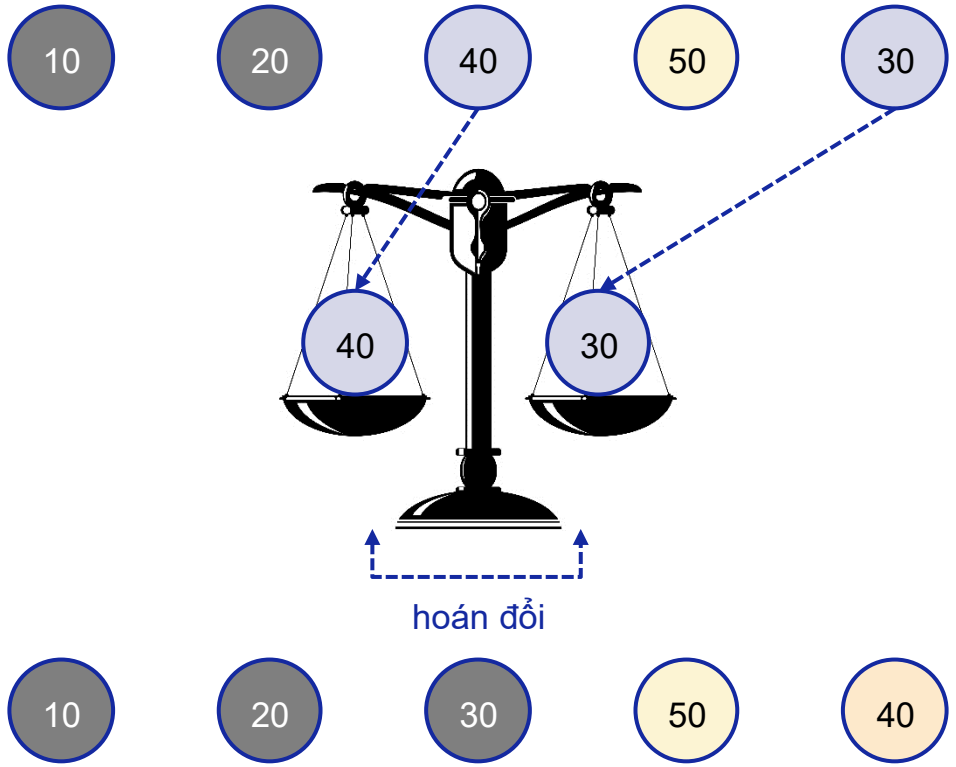
Bây giờ phần tử thứ hai được loại trừ khỏi việc sắp xếp.



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

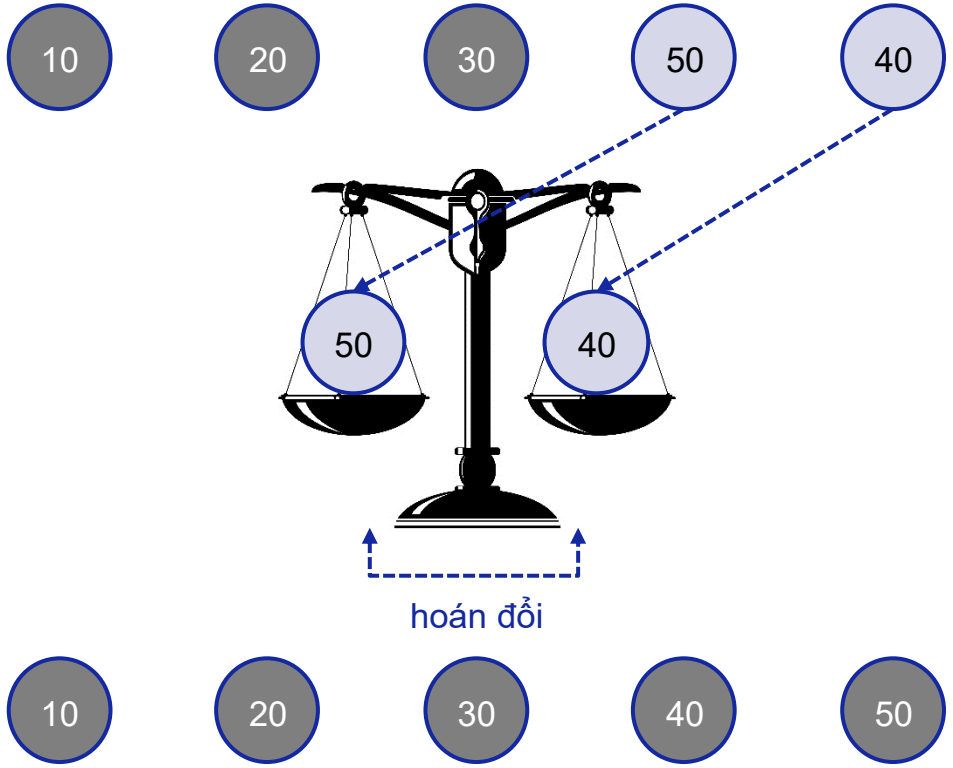
- Việc sắp xếp lựa chọn có thể được tiếp tục trên danh sách chưa được sắp xếp (loại trừ danh sách đã sắp xếp).



SẮP XẾP LỰA CHỌN

2.2. Quá trình sắp xếp lựa chọn

- Việc sắp xếp lựa chọn có thể được tiếp tục trên danh sách chưa được sắp xếp (loại trừ danh sách đã sắp xếp).



SẮP XẾP LỰA CHỌN

3.4. Cài đặt

```python
def selection_sort(nums):
    n = len(nums)
    for i in range(0, n-1):
        for j in range(i+1, n):
            if a[i] < a[j]:
                a[i], a[j] = a[j], a[i]
        print(f"Vong lap {i+1}: {nums}")
a = [1,3,2,5,4]
selection_sort(a)

```

```
Vong lap 1: [5, 1, 2, 3, 4]
Vong lap 2: [5, 4, 1, 2, 3]
Vong lap 3: [5, 4, 3, 1, 2]
Vong lap 4: [5, 4, 3, 2, 1]
```

Nhận xét

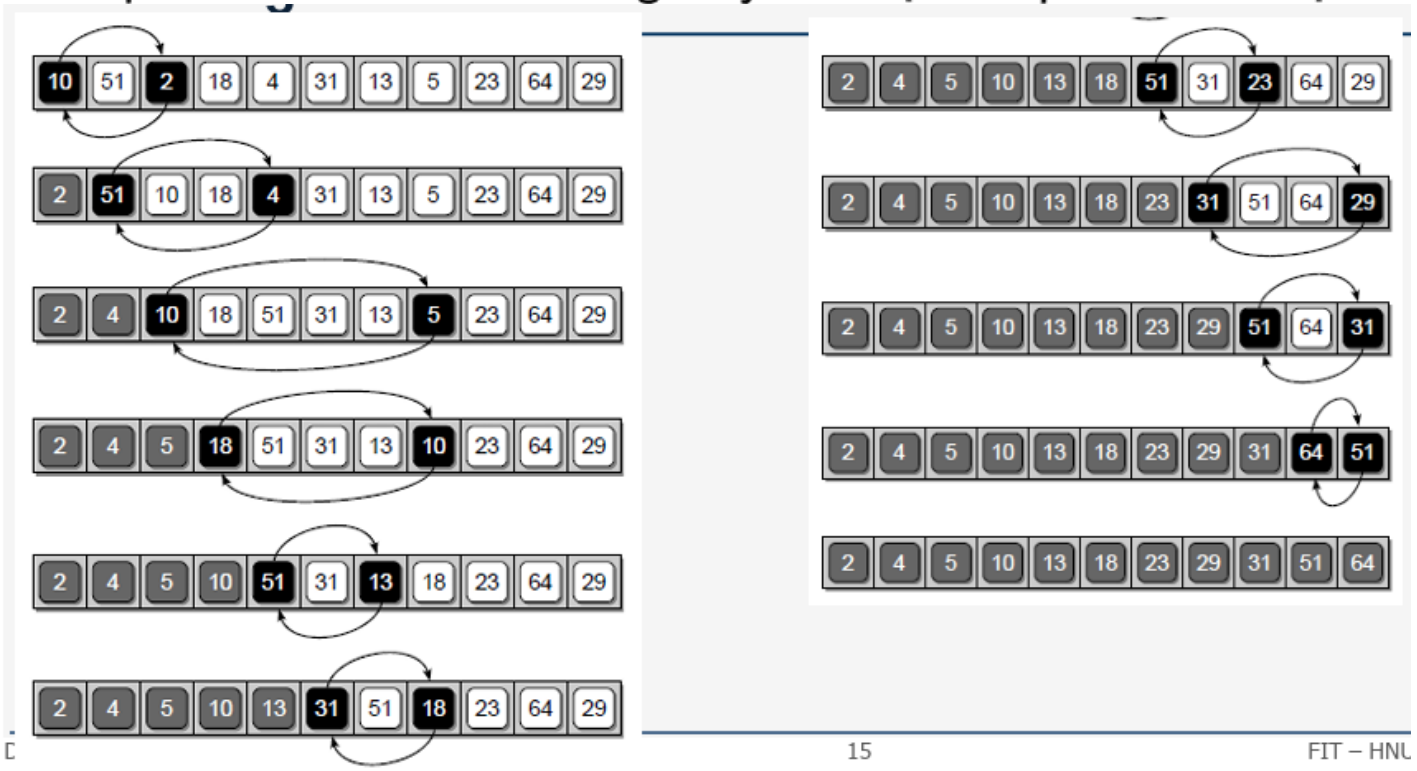
- Đổi giá trị nhiều lần để đặt được số nhỏ nhất trong dãy còn lại vào vị trí thứ i => Hạn chế đổi giá trị bằng cách tìm giá trị nhỏ nhất trong dãy còn lại, đổi giá trị của phần tử nhỏ nhất với phần tử ở vị trí thứ i.

SẮP XẾP LỰA CHỌN

2.2. Ý tưởng thuật toán

Thực hiện n bước, từ i = 0 tới (n-1). Tại bước thứ i:

- Tìm phần tử nhỏ thứ i trong dãy;
- Đổi chỗ phần tử nhỏ nhất trong dãy còn lại với phần tử ở vị trí thứ i.



SẮP XẾP LỰA CHỌN

3.4. Cài đặt

```
1  def selectionsort2(S):
2      n = len(S)
3      for i in range(n - 1):
4          print(S)
5          smallest = i
6          for j in range(i + 1, n):
7              if S[j] < S[smallest]:
8                  smallest = j
9          S[i], S[smallest] = S[smallest], S[i]
```

```
10 S = [50, 30, 40, 10, 20]
11 selectionsort2(S)
12 print(S)
```

```
[50, 30, 40, 10, 20]
[10, 30, 40, 50, 20]
[10, 20, 40, 50, 30]
[10, 20, 30, 50, 40]
[10, 20, 30, 40, 50]
```

Dòng 4

- Để kiểm tra hoạt động của sắp xếp lựa chọn, trạng thái của S được xuất ra mỗi khi vòng lặp thứ i được thực hiện.

SẮP XẾP LỰA CHỌN

Mở rộng

- Độ phức tạp thời gian của thuật toán sắp xếp là gì?
- Vì sắp xếp lựa chọn sử dụng 2 vòng for lồng nhau, độ phức tạp thời gian là O (N^2).
- Tuy nhiên số lượng phép hoán đổi ít hơn.
- Dãy số đã được sắp xếp: 3, 7, 9 , 11; nhưng thuật toán vẫn thực hiện với độ phức tạp O (N^2) => cải tiến ????

Thực hành sắp xếp lựa chọn (Thực hành 10.2 – LMS)

THỜI GIAN: 21h – 21h15

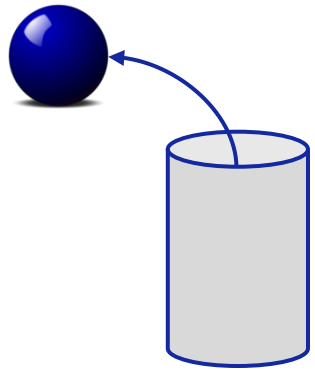
SẮP XẾP CHÈN

SẮP XẾP CHÈN

2.3. Sắp xếp khi bạn chỉ có thể nhận được một viên bi tại một thời điểm

- Sử dụng cân thăng bằng. Nhưng có những hạn chế:
  - Các viên bi sắt ở trong thùng và chỉ có thể lấy ra từng viên một.
  - Bất cứ khi nào một viên bi được lấy ra, nó phải luôn được đặt ở vị trí đã được sắp xếp.
- Trong trường hợp này, không thể so sánh tất cả các viên bi cùng một lúc, nhưng vì đã có các viên bi đã được sắp xếp, nên viên bi mới có thể được tìm và đặt vào theo đúng thứ tự.
- Với chiến lược này, có bao nhiêu phép so sánh để xếp tất cả các viên bi theo thứ tự sắp xếp theo khối lượng?

Đã sắp xếp



SẮP XẾP CHÈN

3.1. Ví dụ về sắp xếp chèn

Sắp xếp chèn là một thuật toán sắp xếp trong đó thực hiện việc thêm từng phần tử từ danh sách chưa được sắp xếp vào danh sách đã sắp xếp. Để làm điều này, nó sử dụng một chiến lược để tìm vị trí cần chèn bằng cách so sánh phần tử sẽ được thêm vào với lần lượt từng phần tử trong danh sách được sắp xếp.

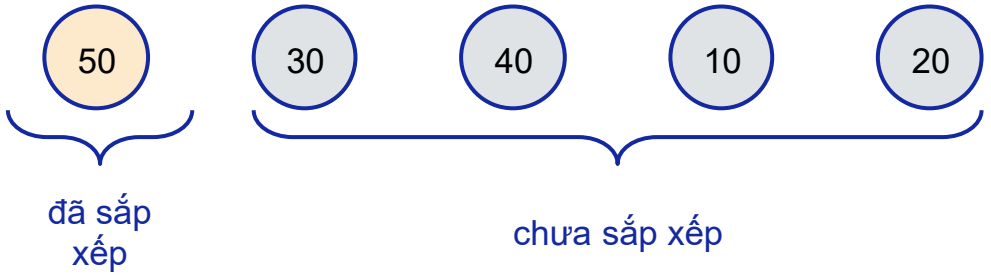
Ví dụ Giả sử chúng ta muốn sắp xếp một danh sách gồm các phần tử [50, 30, 40, 10, 20].

50 30 40 10 20

SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

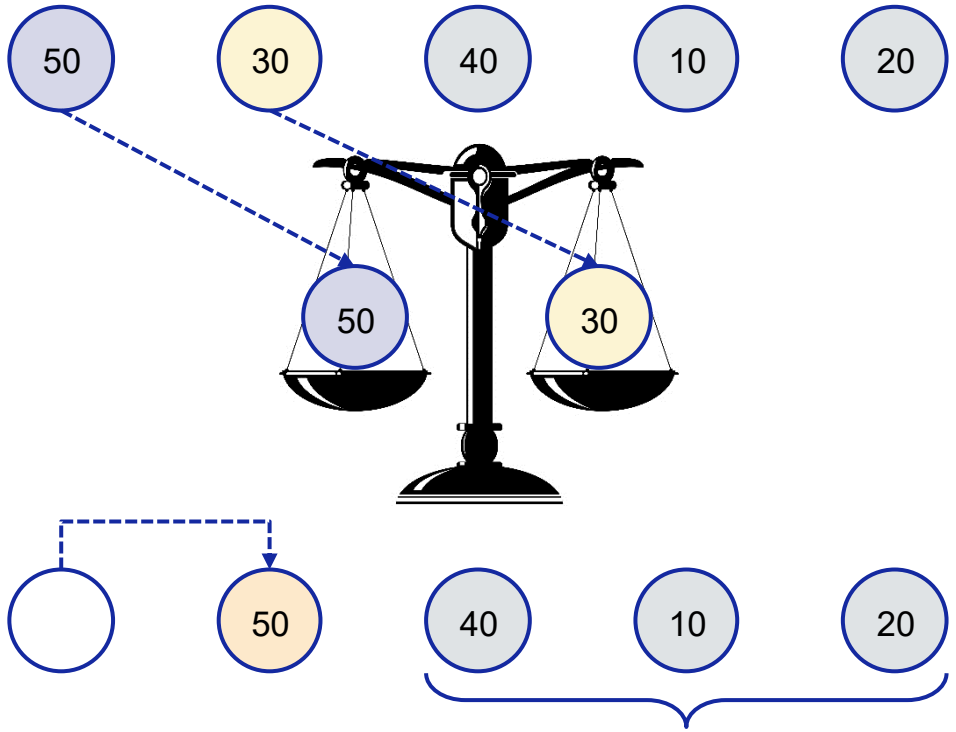
- Đầu tiên, phần tử 50 có thể được coi là một danh sách đã được sắp xếp vì chỉ có một phần tử.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

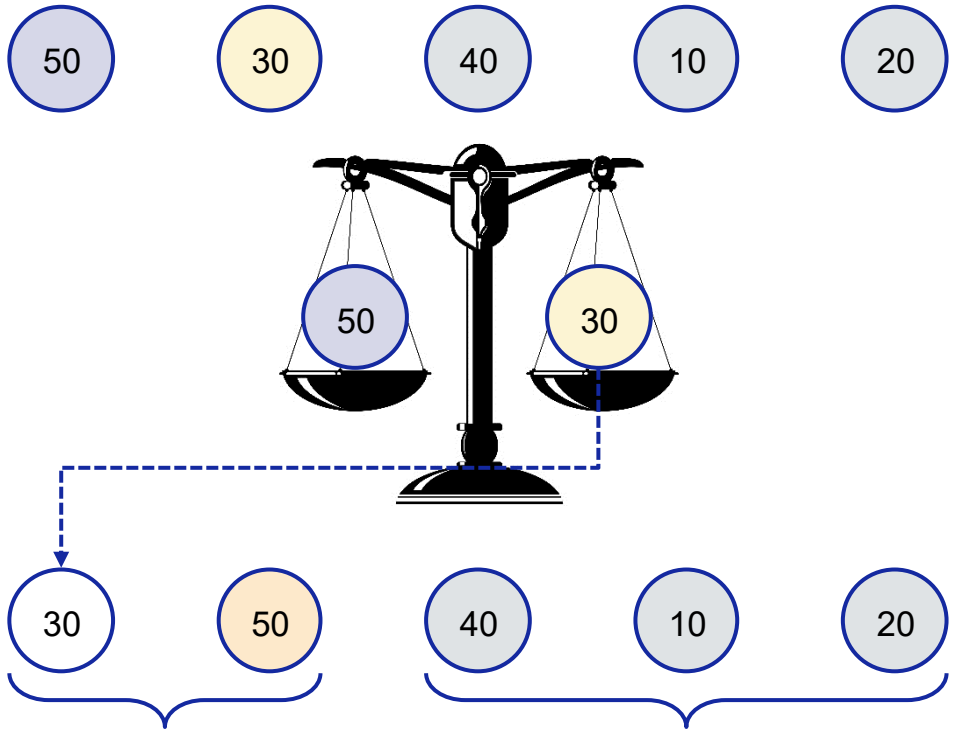
- Vì phần tử thứ hai 30 nhỏ hơn 50 trong danh sách đã sắp xếp, nên 50 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

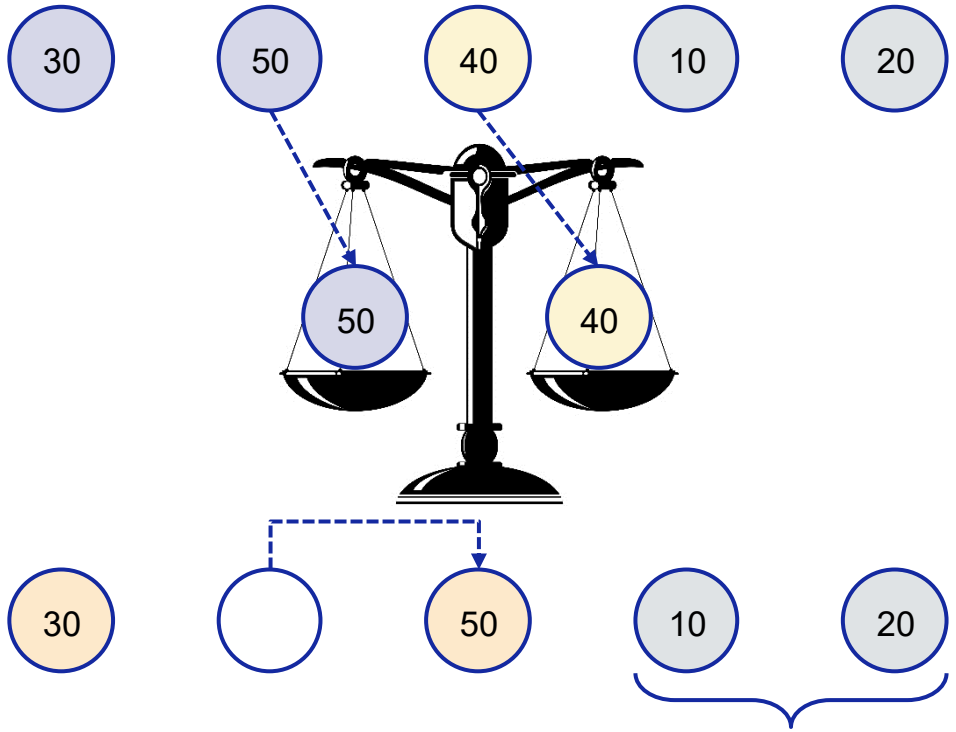
Sau khi di chuyển 50, 30 được chèn vào vị trí đó vì không còn phần tử nào để so sánh.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

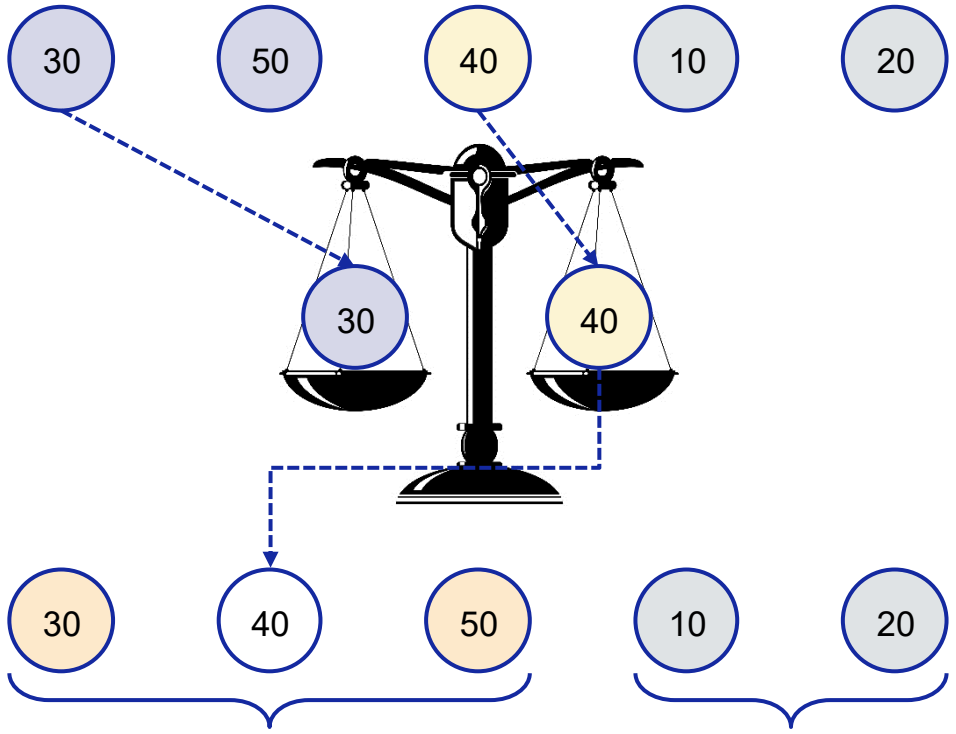
Vì phần tử thứ ba 40 nhỏ hơn 50 trong danh sách đã sắp xếp, nên 50 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

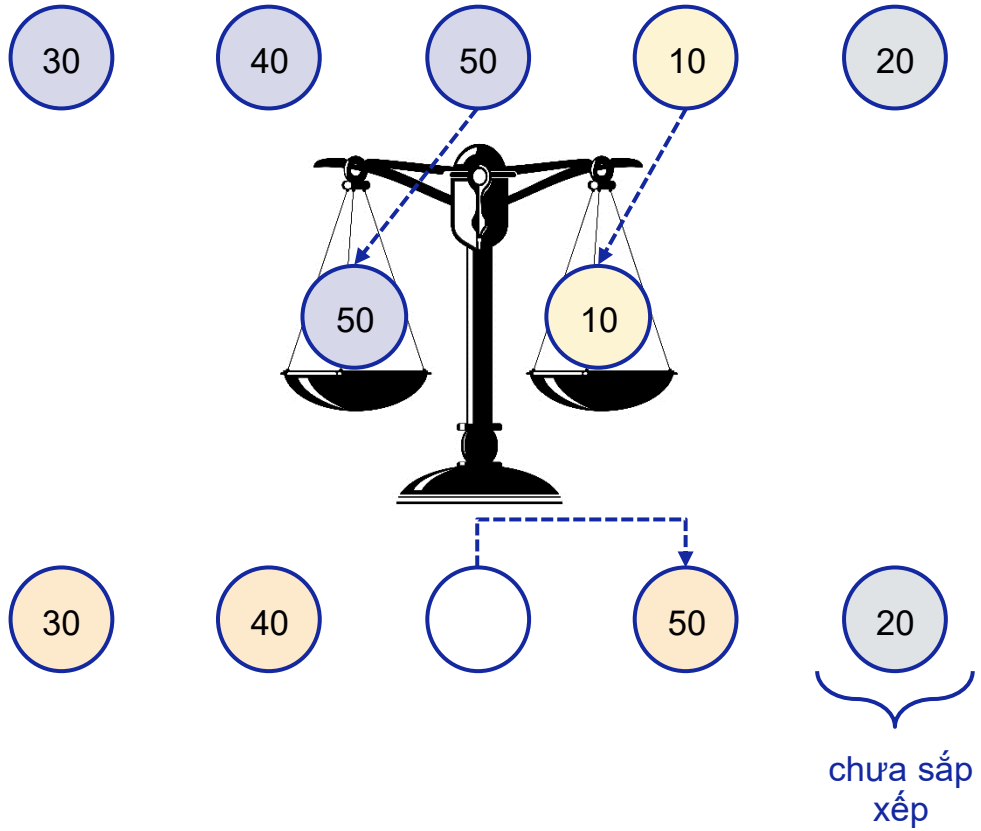
Trong danh sách đã sắp xếp, vì phần tử 30 nhỏ hơn 40 nên 40 được chèn vào vị trí hiện tại.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

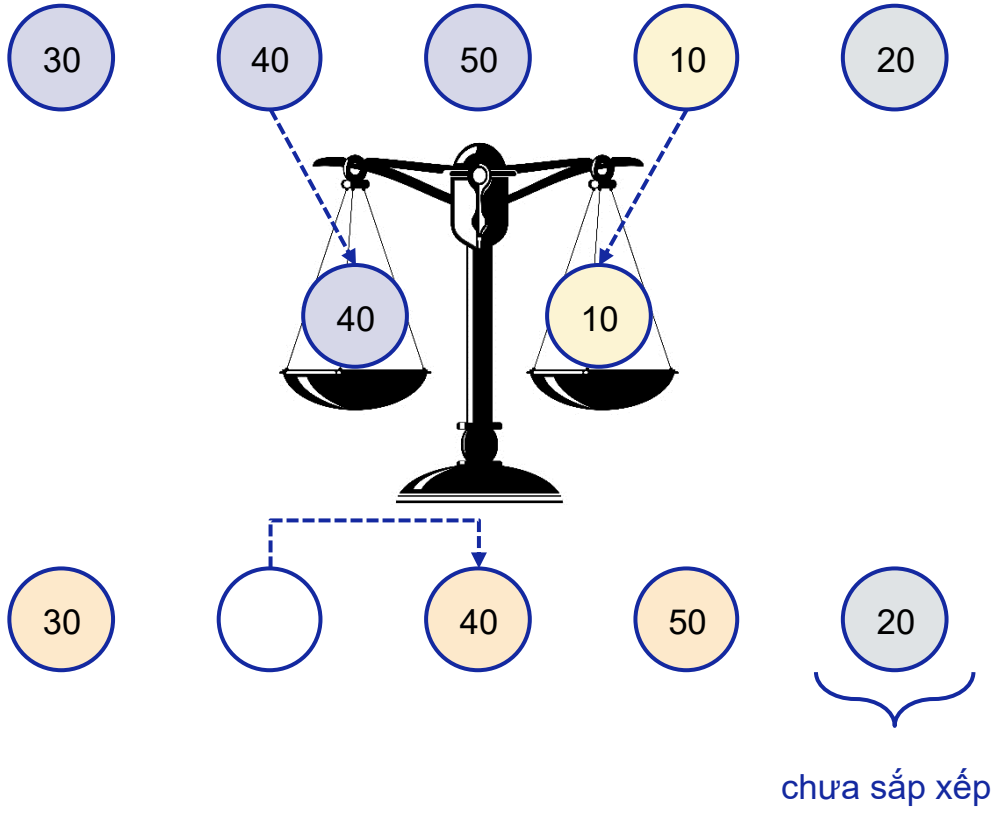
- Trong danh sách đã sắp xếp, vì phần tử trước 50 lớn hơn 10 nên phần tử 50 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

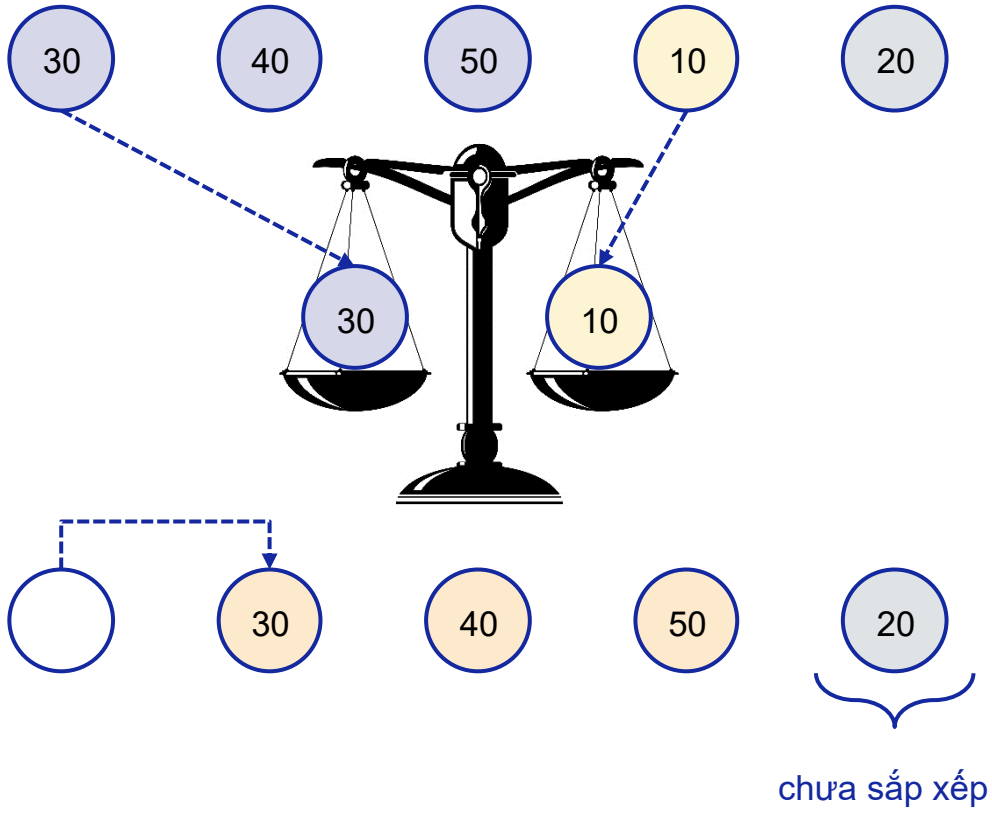
Trong danh sách đã sắp xếp, vì phần tử 40 trước đó lớn hơn 10 nên 40 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

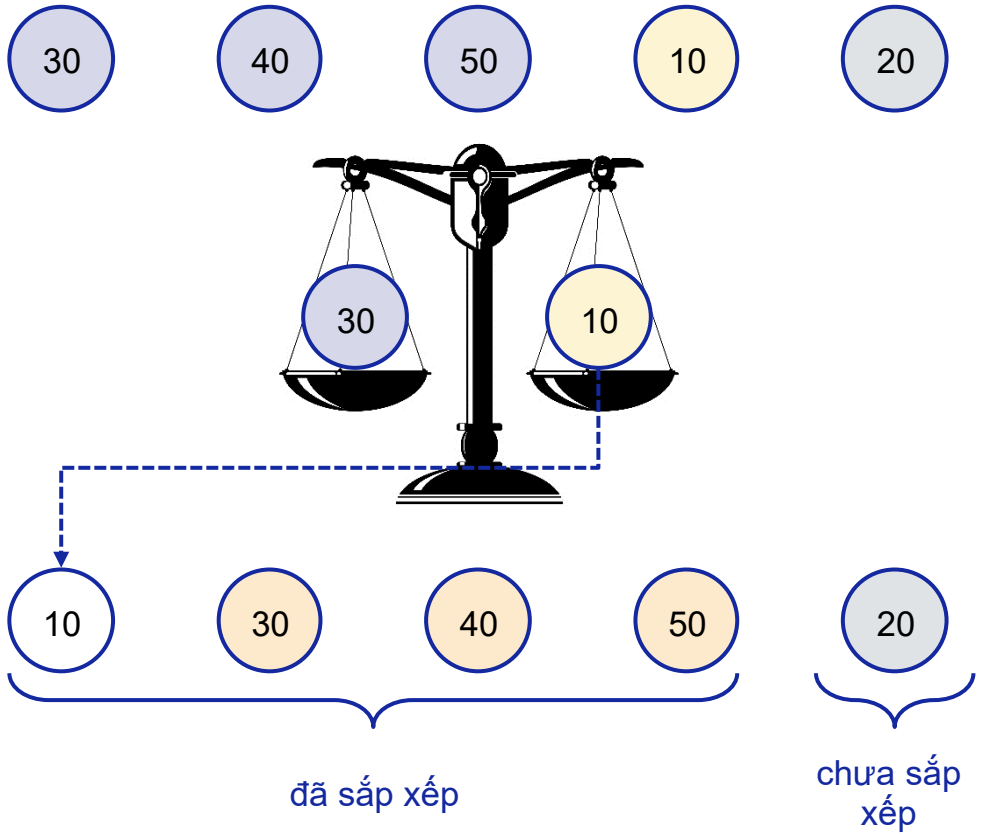
- Trong danh sách đã sắp xếp, vì phần tử trước 30 lớn hơn 10 nên 30 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

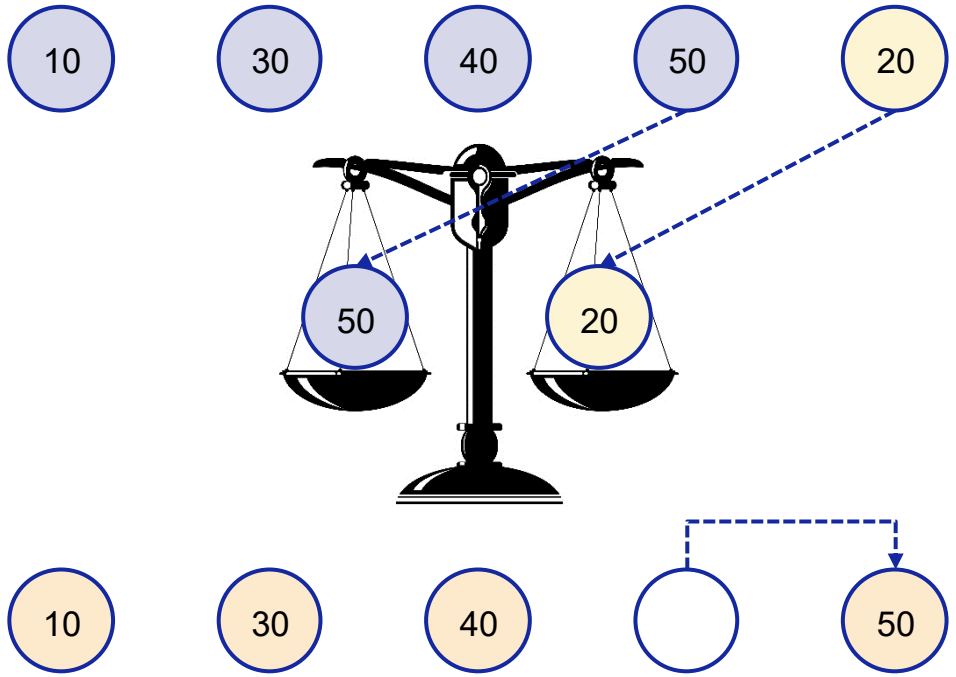
10 được chèn tại vị trí đó vì không còn phần tử nào để so sánh trong danh sách đã sắp xếp.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

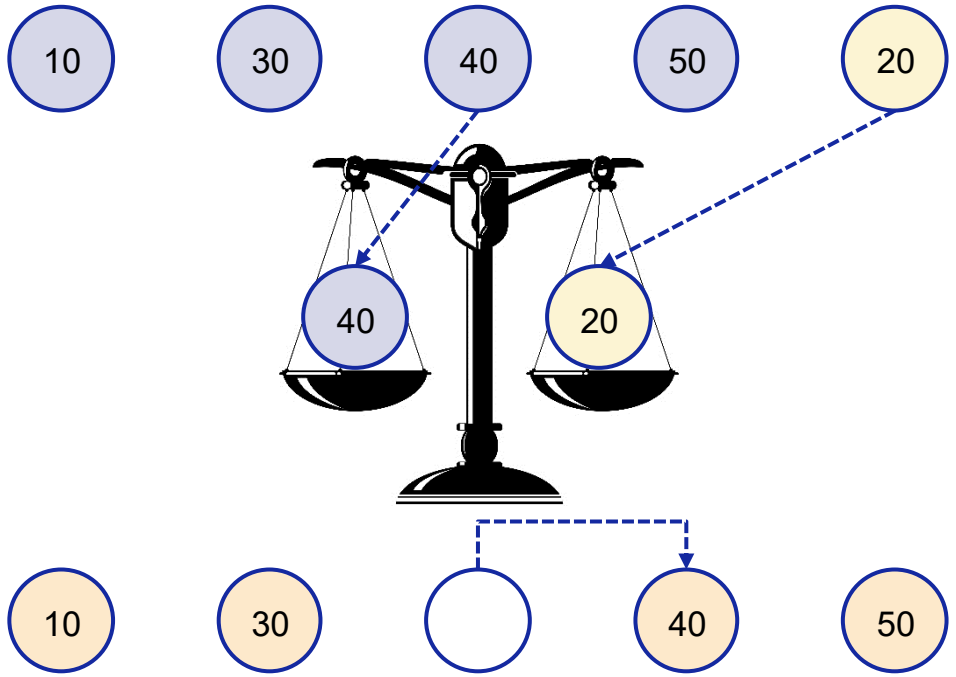
Vì phần tử cuối cùng 20 nhỏ hơn 50 trong danh sách đã sắp xếp, nên 50 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

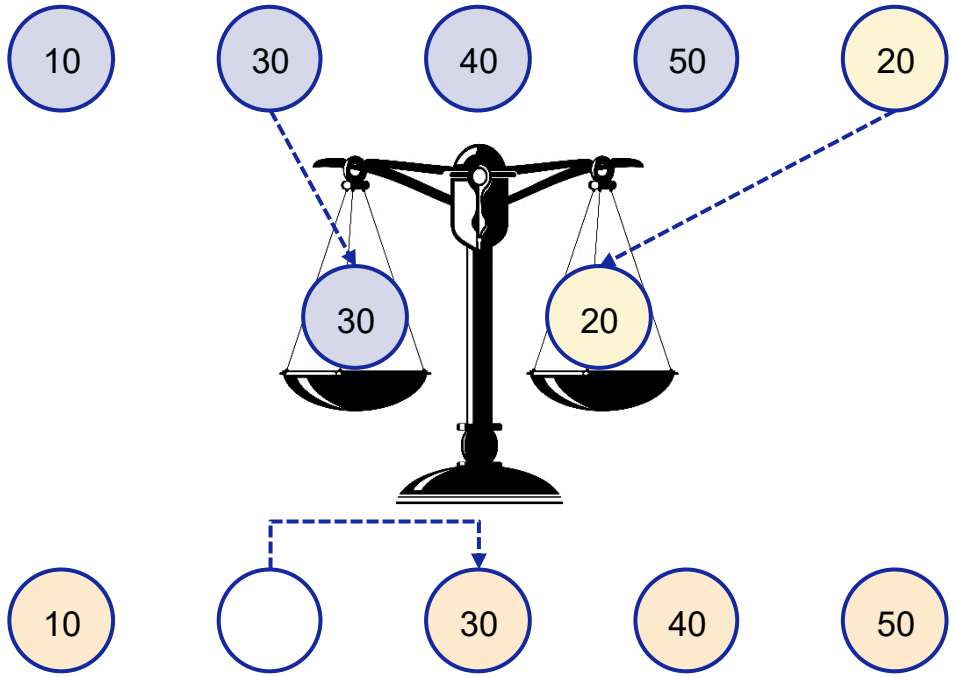
Trong danh sách đã sắp xếp, vì phần tử trước 40 lớn hơn 20 nên 40 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

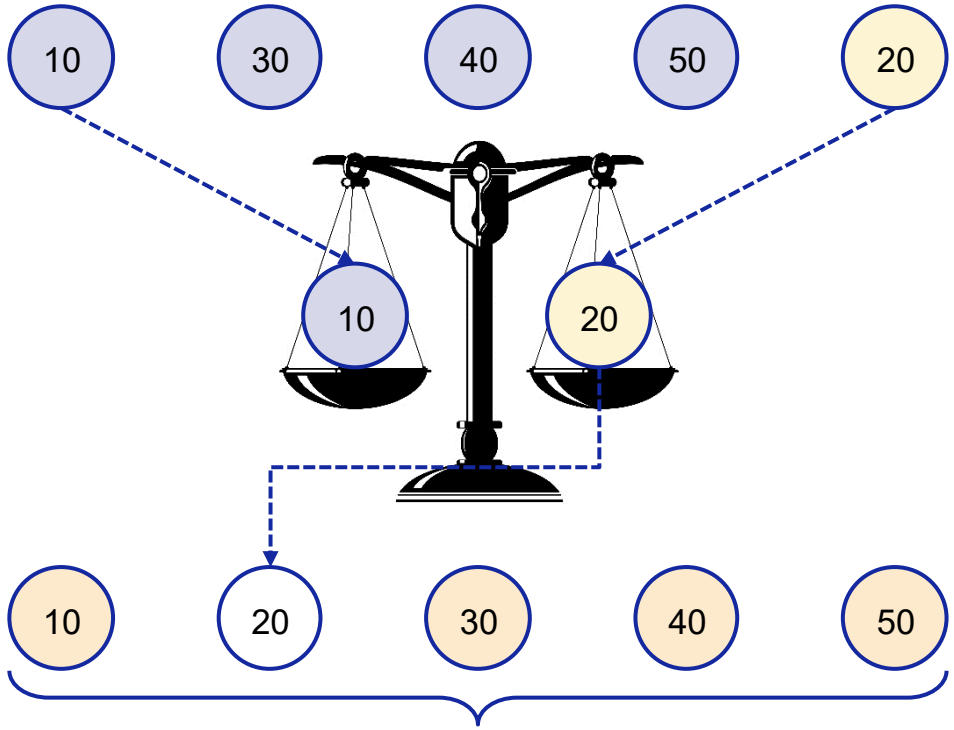
Trong danh sách đã sắp xếp, vì phần tử trước 30 lớn hơn 20 nên 30 được chuyển sang phải một vị trí.



SẮP XẾP CHÈN

3.2. Quá trình sắp xếp chèn

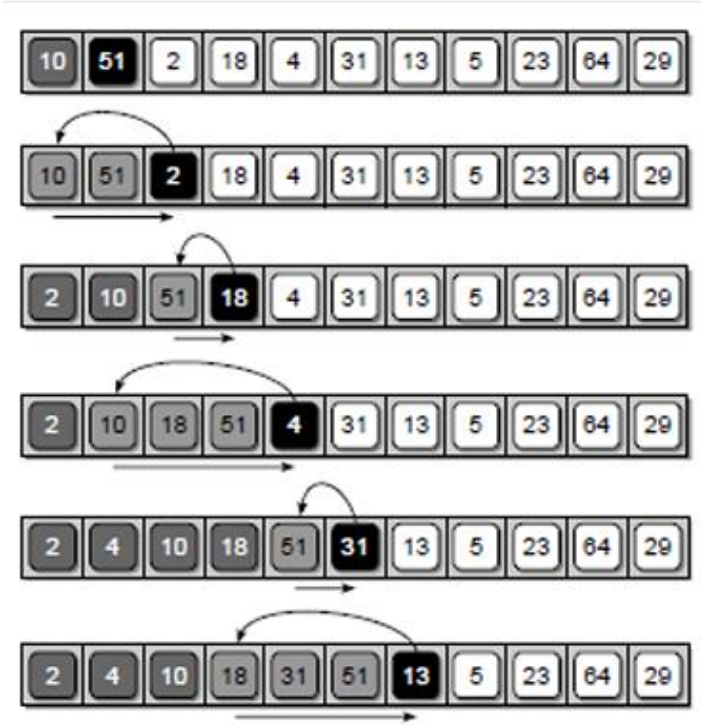
Trong danh sách đã sắp xếp, vì phần tử trước đó 10 nhỏ hơn 20 nên 20 sẽ được chèn vào vị trí đó. Chúng ta đã chèn tất cả các phần tử nên toàn bộ danh sách là danh sách đã được sắp xếp.

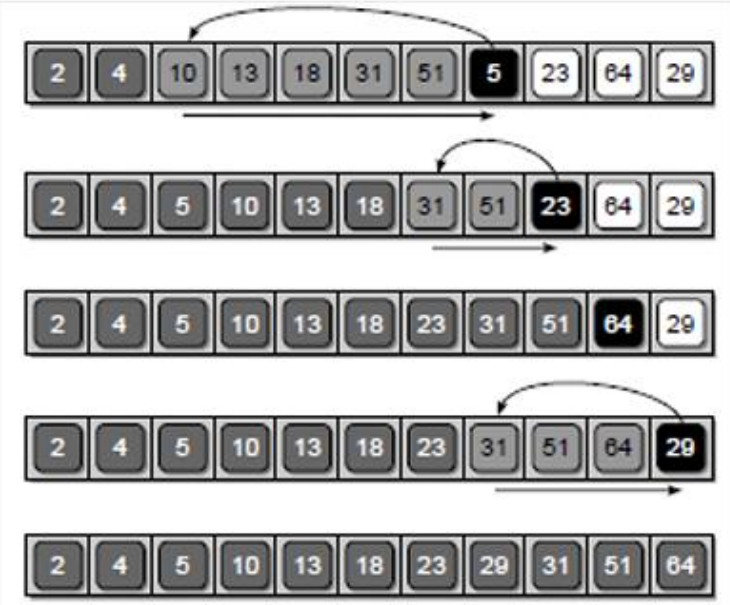


SẮP XẾP CHÈN

3.2. Ý tưởng thuật toán

- Thực hiện n-1 bước, từ i = 1 tới n-1.
- Tại bước thứ i :
  - Tìm vị trí để đặt phần tử thứ i vào danh sách đã được sắp xếp trước đó (bằng cách dịch các phần tử sang phải trong quá trình tìm kiếm)
  - Đặt phần tử thứ i vào vị trí đúng.





SẮP XẾP CHÈN

3.2. Cài đặt thuật toán

```python
def insertionsort2(S):
    n = len(S)
    for i in range(1, n):
        print(S)
        x = S[i]
        j = i - 1
        while j >= 0 and S[j] > x:
            S[j + 1] = S[j]
            j -= 1
        S[j + 1] = x
```

```python
S = [50, 30, 40, 10, 20]
insertionsort2(S)
print(S)
```

```
[50, 30, 40, 10, 20]
[30, 50, 40, 10, 20]
[30, 40, 50, 10, 20]
[10, 30, 40, 50, 20]
[10, 20, 30, 40, 50]
```

SẮP XẾP CHÈN

Mở rộng

- Độ phức tạp thời gian của thuật toán sắp xếp là gì?
- Trong trường hợp sắp xếp chèn, số lượng so sánh khác nhau giữa trường hợp xấu nhất và trường hợp tốt nhất.
- Trường hợp tốt nhất là một mảng đã được sắp xếp. Trong trường hợp này, độ phức tạp về thời gian trở thành O (N) vì vị trí chèn chỉ có thể được tìm thấy bằng một phép so sánh cho mỗi bước lặp.
- Trường hợp xấu nhất là một mảng được sắp xếp theo thứ tự ngược lại. Trong trường hợp này, độ phức tạp về thời gian là O (N^2) như trong sắp xếp nổi bọt/lựa chọn, vì vị trí chèn phải được so sánh với tất cả các phần tử của danh sách được sắp xếp cho mỗi lần lặp.

Thực hành sắp xếp chèn (Thực hành 10.3 – LMS)

THỜI GIAN: 21h30 – 22h