Kỹ Thuật Lập Trình
Part 05

Do Phu Thinh

Table of Contents

Tổng Quan Các Giải Thuật Sắp Xếp

- ✗ "Some" famous sorting algorithms

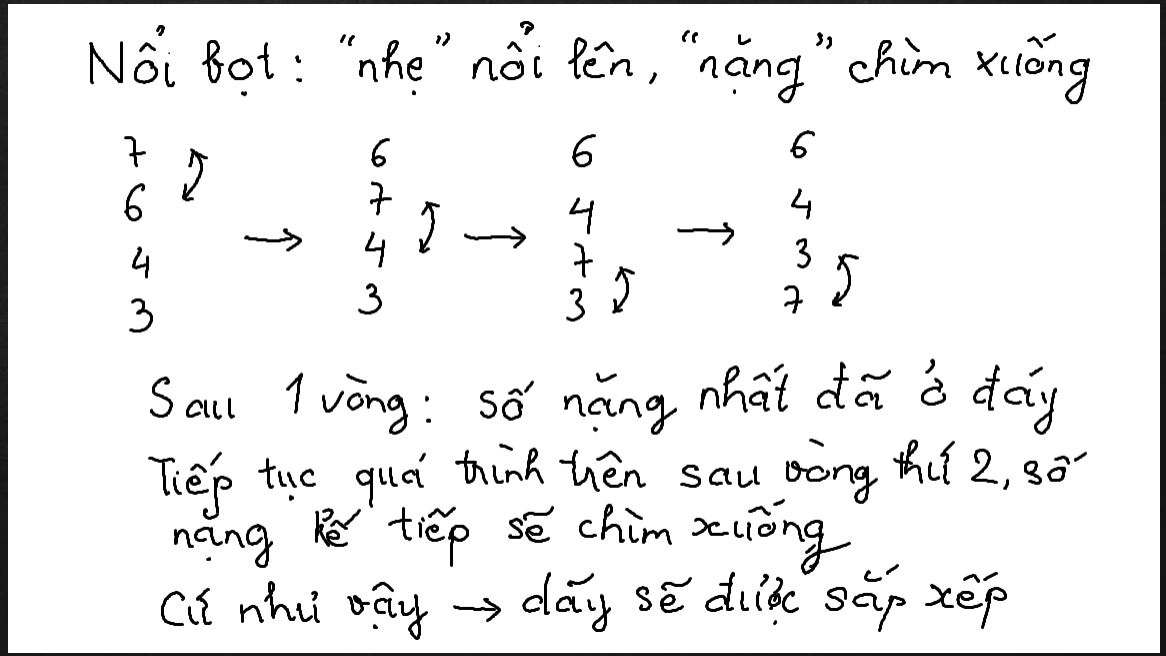
Advanced sorting algorithms. Most built-in sorting algorithms in programming languages are based on these ones

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Learn these algos since it's quite simple and lay foundations for other advanced algos

Bubble Sort

- Giải thuật sắp xếp nổi bọt: dựa trên ý tưởng "nặng" chìm xuống và "nhẹ" nổi lên



Bubble Sort

✗ Implement hàm one_pass_sort:

Lưu ý hàm này xảy ra in-place nên không cần return kết quả (kết quả sẽ cập nhật lên chính đối số đưa vào)

```
def one_pass_sort(li):
    for i in range(len(li)-1):
        if li[i] > li[i+1]:
            li[i], li[i+1] = li[i+1], li[i]
```

Ví dụ: li = [7,6,4,3]

one_pass_sort(li) → [6, 4, 3, 7]

one_pass_sort(li) → [4, 3, 6, 7]

one_pass_sort(li) → [3, 4, 6, 7]

✗ Iteration through one_pass_sort several times

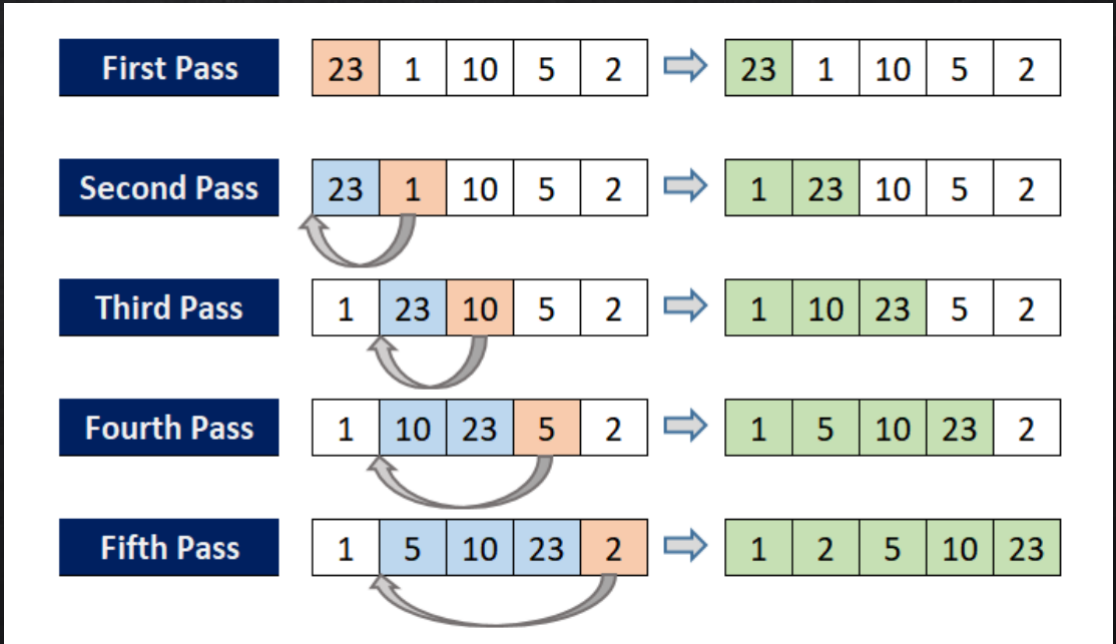
```
def bubble_sort(li):
    for i in range(len(li)-1):
        one_pass_sort(li)
```

Ví dụ: li = [7,6,4,3]

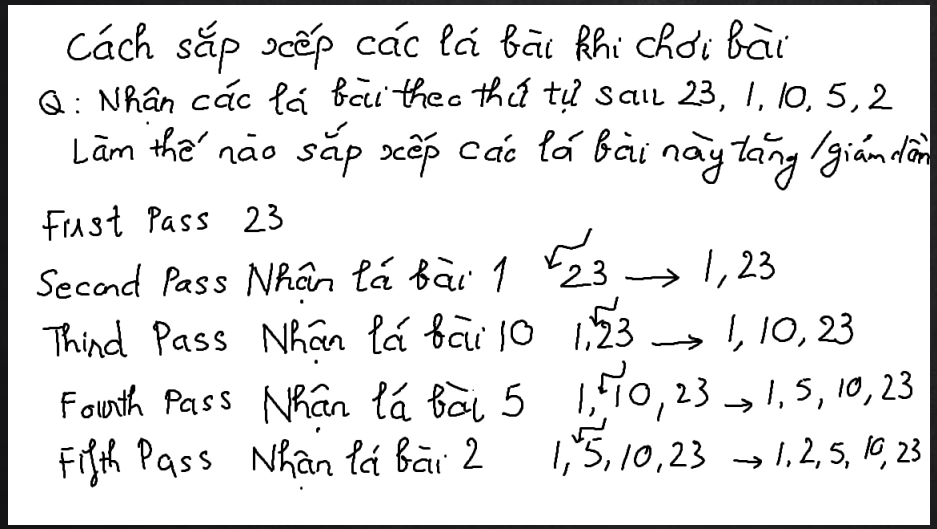
bubble_sort(li) → [3, 4, 6, 7]

Insertion Sort

- Giải thuật sắp xếp chèn: dựa trên ý tưởng xếp bài khi nhận một lá bài mới sẽ chèn vào vị trí phù hợp đảm bảo các tấm bài được sắp xếp theo thứ tự tăng dần



Insertion Sort



- ✗ Step 1: Xác định vị trí phù hợp để chèn vào, gọi vị trí này found_idx
- ✗ Step 2: Dịch sang phải 1 đơn vị tất cả các lá bài từ vị trí found_idx
- ✗ Step 3: Chèn lá bài mới được chia vào vị trí found_idx
- ✗ Thực hiện lại tất cả các step trên cho tất cả các lá bài được chia

Insertion Sort – Implementation

✗ Step 1: Xác định vị trí phù hợp để chèn vào, gọi vị trí này found_idx

Question: Cho một dãy tăng dần, và cho một số bất kỳ, tìm vị trí index phù hợp sao cho khi đưa giá trị num vào index này thì dãy vẫn giữ tính chất tăng dần

```
def find_position(li,num):
    for  i in range(len(li)):
        if num < li[i]:
            return i
    return len(li)
```

```
li = [2,5,7,10]
num = 6
```

```
find_position(li,num)
```

```
2
```

```
li = [2,5,7,10]
num = 12
```

```
find_position(li,num)
```

```
4
```

Insertion Sort – Implementation

✗ Step 2: Dịch sang phải 1 đơn vị tất cả các lá bài từ vị trí found_idx

```
def shift_to_right(li,idx):
    # extend one element more for li
    li.append(None)
    for i in range(len(li)-1,idx,-1):
        li[i] = li[i-1]
```

Ví dụ: li = [2,5,4,3,6], và found_idx = 2 → dịch 4,3,6 qua phải 1 vị trí

```
li = [2,5,4,3,6]
found_idx = 2
shift_to_right(li,found_idx)
```

```
li
```

```
[2, 5, 4, 4, 3, 6]
```

Insertion Sort – Implementation

- ✗ Step 3: Chèn lá bài mới được chia vào vị trí found_idx (straightforward)
- ✗ Thực hiện quy trình lặp step 1, step 2, step 3 hoàn thiện insertion sort

```python
def find_position(li,num):
    for i in range(len(li)):
        if num < li[i]:
            return i
    return len(li)

def shift_to_right(li,idx):
    # extend one element more for li
    li.append(None)
    for i in range(len(li)-1,idx,-1):
        li[i] = li[i-1]

def insertion_sort(li_input):
    output_li = []
    for v in li_input:
        found_idx = find_position(output_li,v)
        shift_to_right(output_li,found_idx)
        output_li[found_idx] = v
    return output_li
```

```python
li = [2,5,4,3,3,4,6]
output_li = insertion_sort(li)
```

```
output_li
```

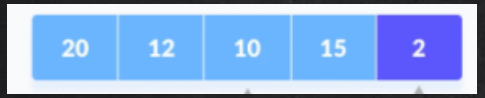
```
[2, 3, 3, 4, 4, 5, 6]
```

Selection Sort

- Ý tưởng: Tìm số nhỏ nhất (selection) trong phần còn lại để sắp xếp

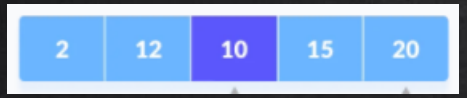
Ví dụ: Sắp xếp dãy [20, 12, 10, 15, 2]

Step 1: tìm số nhỏ nhất trong [20,12,10,15,2] và hoán đổi số nhỏ nhất này với số đầu tiên trong list [20,12,10,15,2]



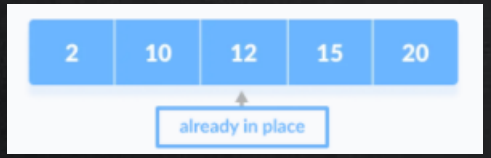
[2, 12, 10, 15, 20]

Step 2: tìm số nhỏ nhất trong [12,10,15,20] và hoán đổi số nhỏ nhất này với số thứ hai trong list [2,12,10,15,20]



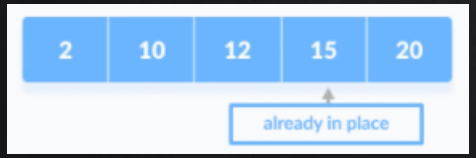
[2, 10, 12, 15, 20]

Step 3: tìm số nhỏ nhất trong [12,15,20] và hoán đổi số nhỏ nhất này với số thứ ba trong list [2,12,10,15,20]



[2, 10, 12, 15, 20]

Step 4: tìm số nhỏ nhất trong [15,20] và hoán đổi số nhỏ nhất này với số thứ tư trong list [2,10,12,15,20]



[2, 10, 12, 15, 20]

Selection Sort – Implementation

✗ Viết hàm số tìm index của minimum value trong given list

```
def find_min_idx(input_li):
    min_value = input_li[0]
    min_idx = 0
    for i in range(1,len(input_li)):
        v = input_li[i]
        if v < min_value:
            min_value = v
            min_idx = i
    return min_idx
```

```
li = [2,5,4,3,3,1,4,6]
find_min_idx(li)
```

```
5
```

✗ Thực hiện vòng lặp mỗi bước lặp swapping vị trí cần sắp xếp với phần tử nhỏ nhất trong phần còn lại của list

```
def selection_sort(input_li):
    for i in range(len(input_li)):
        min_idx = i + find_min_idx(li[i:])
        li[i], li[min_idx] = li[min_idx], li[i]
```

```
li = [2,5,4,3,3,1,4,6]
selection_sort(li)
```

```
li
```

```
[1, 2, 3, 3, 4, 4, 5, 6]
```