Lecture 1: Introduction

1. Why study algorithms

Thuật toán là gì?

Thuật toán là một tập các nguyên tắc hay công thức nhằm giải quyết một bài toán nào đó.

Tại sao cần biết thuật toán?

- Thuật toán là một thành phần quan trọng trong khoa học máy tính
- Thuật toán ảnh hưởng đến tất cả các nhánh trong khoa học máy tính
- Thuật toán đóng vai trò then chốt trong sự phát triển của công nghệ hiện đại.
- Cung cấp các góc nhìn mới về các lĩnh vực bên khác
- Cung cấp luyện tập khả năng giải quyết vấn đề

2. Integer Multiplication

Input

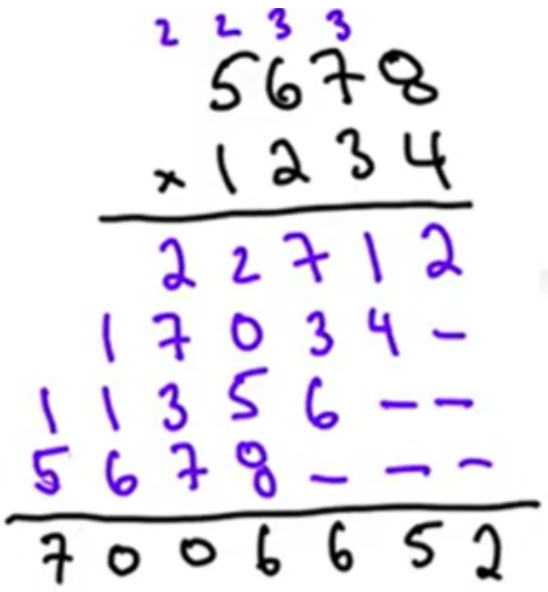
Hai số tự nhiên x và y

Output

Kết quả của phép nhân x và y

Thuật toán “Lớp 3”

Đây là cách chúng ta được học cách thực hiện phép nhân hồi tiểu học, bằng cách lấy từng số của x nhân với y theo quy tác như sau:



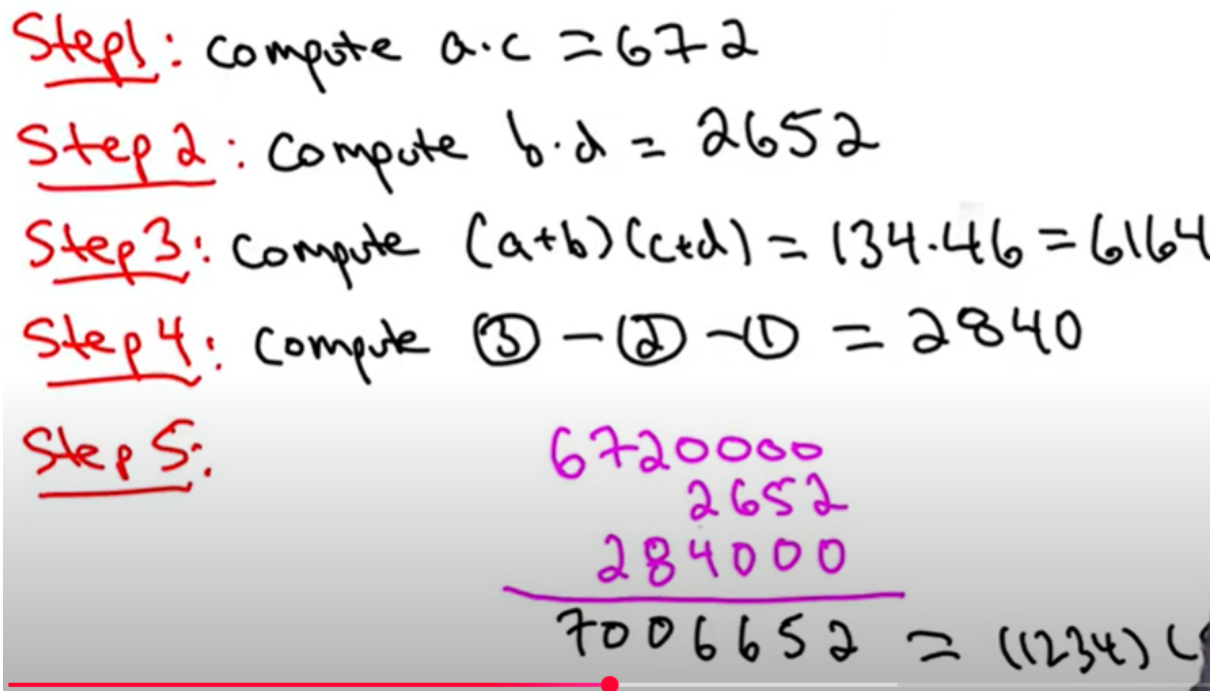
Phương pháp này tốn tối đa $2.n^n$ phép tính cơ bản (hai số có một chữ số nhân với nhau và một vài phép cộng có nhớ)

⇒ Số lượng phép tính cần phải thực hiện tăng khi độ dài x và y tăng

Karatsuba multiplication

Input ví dụ: x = 5678 và y =1234.

Tách x thành a = 56, b = 78 và tách y thành c = 12, d = 34. Thuật toán Karatsuba hoạt động như sau:



Giải thích:

Cách tách x và y như trên có thể được tổng quát hóa như sau:

$x = 10^{n/2}.a + b$

$y = 10^{n/2}.c + d$

Vậy

$$x.y = (10^{n/2}.a + b).(10^{n/2}.c + d) = 10^{n}.ac + 10^{n/2}.(ad + bc) + bd$$

Vậy thay vì thực hiện một phép nhân 4 chữ số nhân 4 chữ số chúng ta cần thực hiện 4 phép nhân 2 chữ số nhân 2 chữ số. Áp dụng tư tưởng đệ quy, mỗi phép nhân 2 chữ số có thể suy về các phép 1 chữ số nhân 1 chữ số, mà ở đây các phép này được coi là tốn 1 phép tính cơ bản.

Ở phép toán ở giữa, thay vì thực hiện tư tưởng đệ quy cho ad và bc rồi lấy tổng, chúng ta có một cách tiếp cận nhanh hơn đó là đệ quy để tính tích (a+b).(c+d) vì:

$$(a + b).(c + d) = ac + ad + bc + bd.$$

Do đã biết ac và bd từ 2 phép đệ quy còn lại nên chúng ta có thể tính ad + bc như sau:

$$ad + bc = (a + b).(c + d) - ac - bd$$

Vậy chúng ta giảm được từ 4 phép đệ quy xuống chỉ còn 3 phép đệ quy

3. Merge Sort Motivation

Đặc điểm

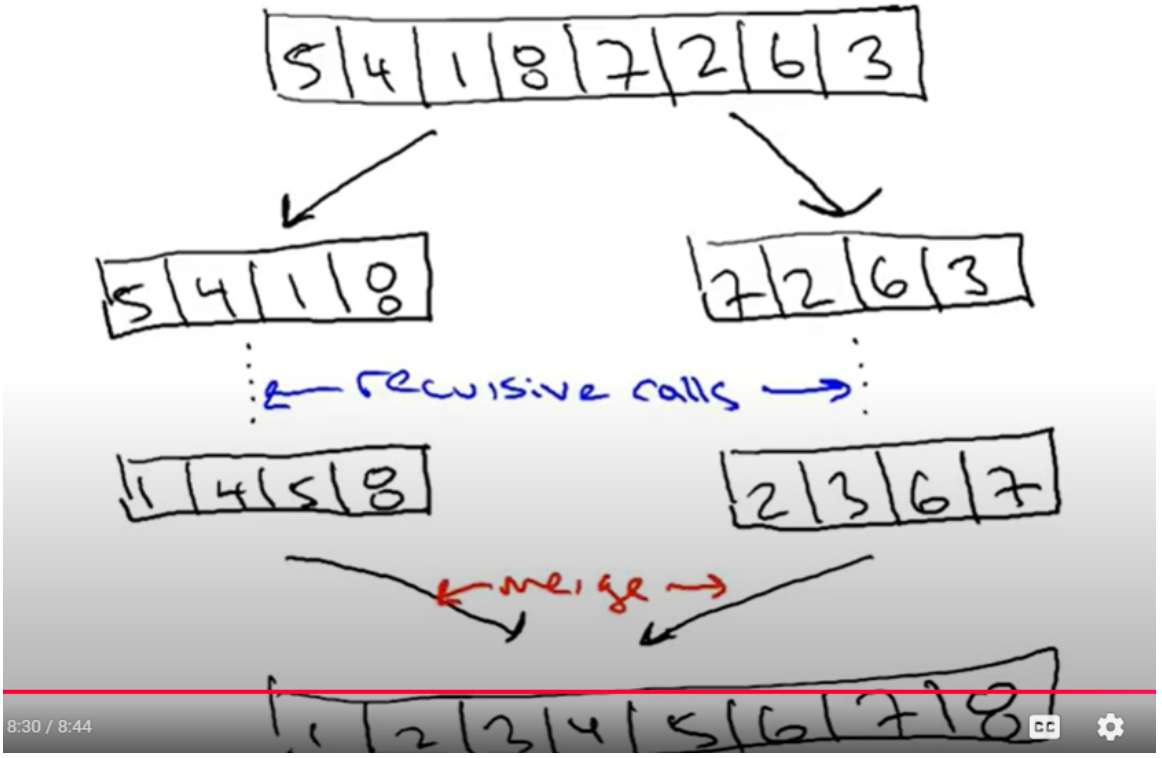
- Ví dụ trực quan của phương pháp chia để trị
- Cải thiện so với các thuật toán sắp xếp khác như: Selection, Insertion, Bubble
- Phù hợp trong việc giải thích cách phân tích độ phức tạp thuật toán (trường hợp tệ nhất, tiệm cận, ...)
- Cung cấp tổng quát về phương pháp Master (Master method)

Bài toán

Input: một mảng n phần tử, không được sắp xếp

Output: mảng đầu vào sau khi đã sắp xếp các phần tử

Cách thuật toán hoạt động:



Cài đặt

- Áp dụng đệ quy

```
Merge(A, B):
        n = A.siz
        i = 1, j = 1
        for k: 1 -> n:
                if A[i] > B[j]
                        C[k] = B[j]
                        j++
                else
                        C[k] = A[i]
                        i++
        return C
```

```
Merge_Sort(A):
        n = A.size
        if n == 1:
                return A
        else:
                B = Merge_Sort(A[1 -> n/2])
                C = Merge_Sort(A[n/2 + 1 -> n])
                A = Merge(B, C)
                return A
```

Phân tích thời gian chạy

Claim: Với mỗi mảng đầu vào gồm n phần tử, thuật toán merge sort thực hiện việc sắp xếp các phần tử sử dụng tối đa $6nlog_2n + 6n$ phép tính toán.

Chứng mình:

- Trong thuật toán merge sort, ở mỗi giai đoạn chúng ta cần phải chia đôi mảng hiện tại ra để sắp xếp cho đến khi đến trạng thái cơ sở là mảng chỉ có 1 phần tử. Mỗi mảng sẽ được chia làm hai phần, từ đó ta tính được có $log_2n + 1$ lần chia.
- Ở mỗi bước đệ quy, chúng ta đề cần phải thực hiện việc gộp mảng, việc này tống 6n phép tính như có thể đếm trong hàm merge ở trên.

⇒ Tổng số phép tính cần thực hiện là:

$$6n(log_2n + 1) = 6nlog_2n + 6n$$

4. Guiding Principles for Analysis of Algorithms

Có 3 nguyên tắc quan trọng chính trong quá trình phần tích thuật toán:

1. Trường hợp tệ nhất (Worst case): chính là tổng thời gian chạy của thuật toán
2. Không cần chú ý đến các yếu tố bất biến: các yếu tố này phụ thuộc vào kiến trúc, ngôn ngữ, compiler đang sử dụng. Ngoài ra chúng không quá ảnh hưởng đến đánh giá chung của thuật toán.

3. Tiệm cận (Asymptotic analysis): tập chung vào thời gian chạy cho các trường hợp với kích cỡ đầu vào n lớn

Thế nào là thuật toán “Nhanh”

Thuật toán được coi là nhanh tương đương với khi trường hợp tệ nhất của nó tăng chậm khi kích cỡ đầu vào tăng dần.