INTRODUCTION TO DATA STRUCTURES AND ALGORITHM

DATA STRUCTURES AND ALGORITHMS

ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

Nội dung

- Thuật toán & Độ phức tạp của thuật toán
- Ví dụ

Thuật tóan

•* Xây dựng cấu trúc không thể tách rời thuật toán

=> Ngoài 1 số tiêu chuẩn lựa chọn cấu trúc dữ liệu, cần xác định thêm yêu cầu thao tác cụ thể trên dữ liệu để chọn 1 cấu trúc phù hợp nhất, các thuật toán thực hiện trên dữ liệu đạt hiệu quả cao nhất.

Thuật toán

- **Thuật toán**: Là tập hợp (dãy, bước) hữu hạn các chỉ thị (hành động) được định nghĩa rõ ràng và có thể thực thi nhằm giải quyết một bài toán cụ thể nào đó. Quá trình hành động theo các bước này phải dừng và cho được kết quả như mong muốn.

- **Ví dụ**: Thuật toán tính tổng tất cả các số nguyên dương nhỏ hơn n gồm các bước sau:

Bước 1: S=0, i=1;
Bước 2: nếu i<n thì s=s+i;
Ngược lại: qua bước 4;
Bước 3:
i=i+1;
Quay lại bước 2;
Bước 4: Tổng cần tìm là S.

Sự cần thiết của thuật toán

- Tại sao sử dụng máy tính để xử lý dữ liệu?
  - Nhanh hơn.
  - Nhiều hơn.
  - Giải quyết những bài toán mà con người không thể hoàn thành được.
- Làm sao đạt được những mục tiêu đó?
  - Nhờ vào sự tiến bộ của kỹ thuật: tăng cấu hình máy ⇨ chi phí cao ☹
  - Nhờ vào các thuật toán hiệu quả: thông minh và chi phí thấp ☺

"Một máy tính siêu hạng vẫn không thể cứu vãn một thuật toán tồi!"

Các tính chất của thuật toán

- Tính chất cơ bản của thật toán:
  - Tính rõ ràng (xác định): các câu lệnh minh bạch (không mập mờ), được sắp xếp theo thứ tự nhất định và có khả năng thực thi.
  - Tính kết thúc (dừng): là sự hữu hạn các bước tính toán.
  - Tính chính xác (đúng): để đảm bảo kết quả tính toán hay các thao tác mà máy tính thực hiện được là chính xác.

Các tính chất của thuật toán

- Ngoài ra còn có những tính chất sau:
  - Tính hiệu quả: tính hiệu quả của thuật toán được đánh giá dựa trên một số tiêu chuẩn như khối lượng tính toán, không gian và thời gian khi thuật toán được thi hành.
  - Tính tổng quát: phải áp dụng được cho mọi trường hợp của bài toán.
  - Tính khách quan: được viết bởi nhiều người trên máy tính nhưng kết quả phải như nhau.
  - Tính phổ dụng: có thể áp dụng cho một lớp các bài toán có đầu vào tương tự nhau.

Biểu diễn Thuật toán

- Dạng ngôn ngữ tự nhiên
- Dạng lưu đồ (sơ đồ khối)
- Dạng mã giả
- Ngôn ngữ lập trình

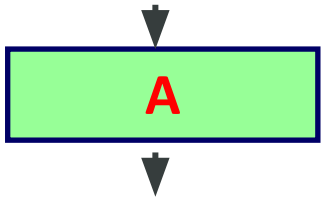
Biểu diễn bằng Ngôn ngữ tự nhiên

- NN tự nhiên thông qua các bước được tuần tự liệt kê để biểu diễn thuật toán.

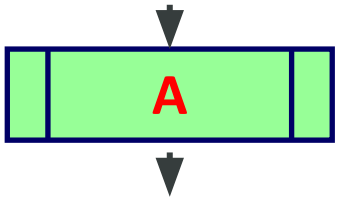
- Ưu điểm:
  - Đơn giản, không cần kiến thức về về cách biểu diễn (mã giả, lưu đồ,...)
- Nhược điểm:
  - Dài dòng, không cấu trúc.
  - Đôi lúc khó hiểu, không diễn đạt được thuật toán.

Lưu đồ

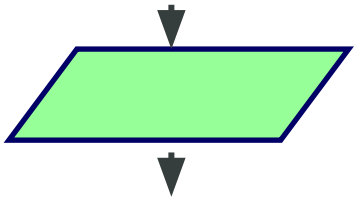
- Là hệ thống các nút, cung hình dạng khác nhau thể hiện các chức năng khác nhau.



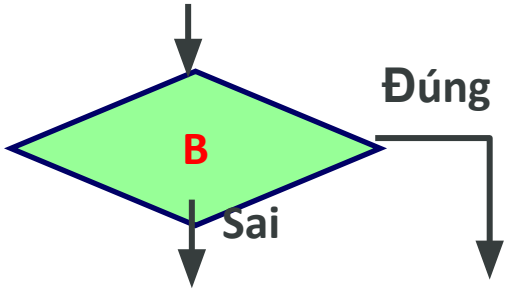
Thực hiện A



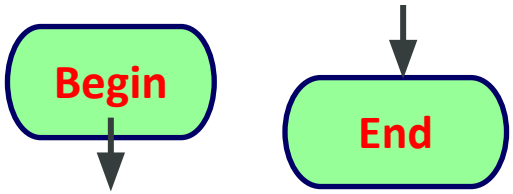
Gọi hàm A



Vào / Ra dữ liệu

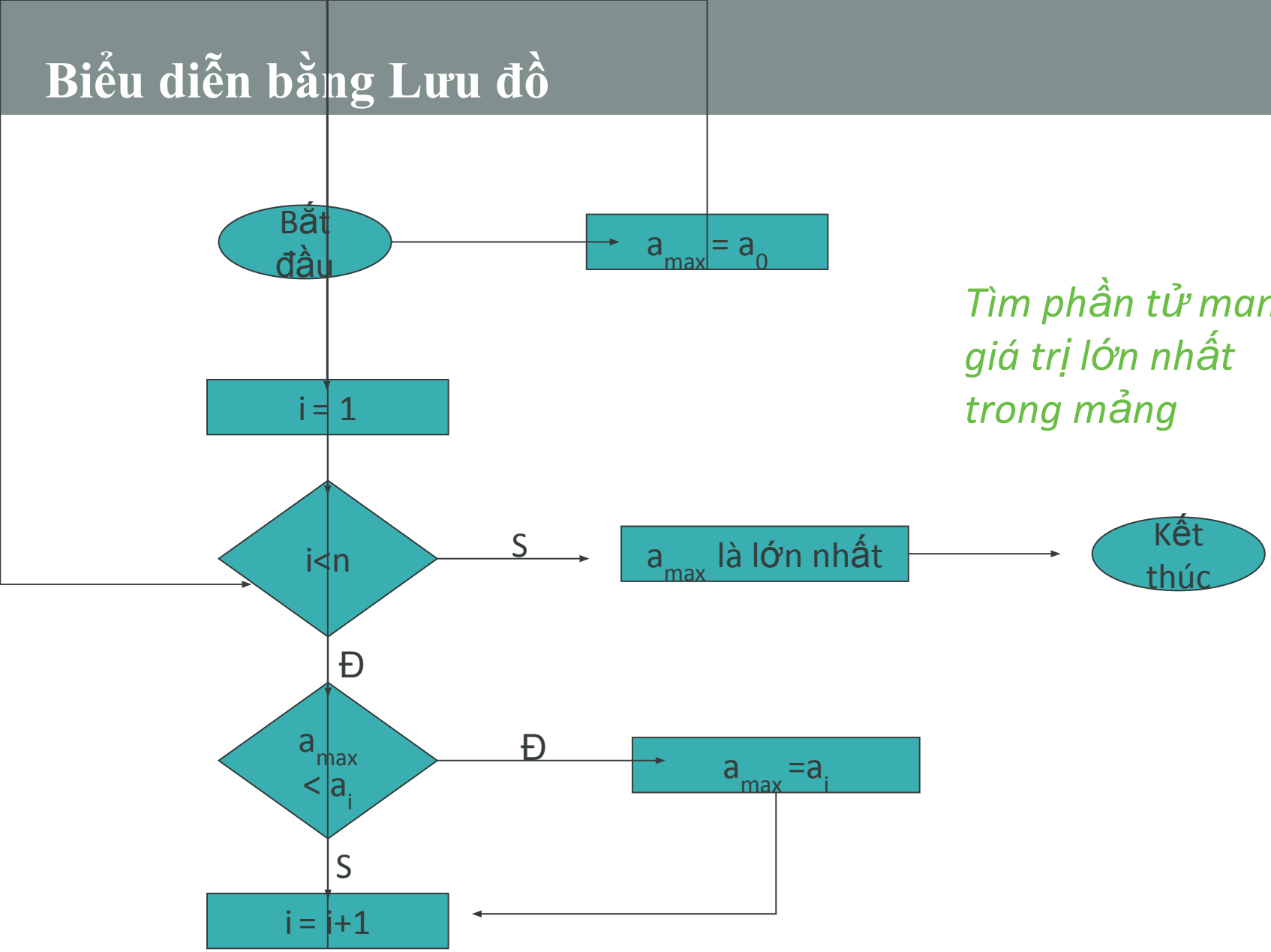


Điều kiện rẻ nhánh B



Nút giới hạn bắt đầu / kết thúc chương trình

Biểu diễn bằng Lưu đồ



Tìm phần tử mang giá trị lớn nhất trong mảng

Biểu diễn bằng Mã giả

- Ngôn ngữ tựa ngôn ngữ lập trình:
  - Dùng cấu trúc chuẩn hóa, chẳng hạn tựa Pascal, C.
  - Dùng các ký hiệu toán học, biến, hàm.
- Ưu điểm:
  - Đỡ cồng kềnh hơn lưu đồ khối.
- Nhược điểm:
  - Không trực quan bằng lưu đồ khối.

Biểu diễn bằng Mã giả

- **Một số quy ước**

1. Các biểu thức toán học
2. Lệnh gán: "=" (A□B)
3. So sánh: "==", "!="
4. Khai báo hàm (thuật toán)

Thuật toán *<tên TT> (<tham số>)*

Input*: <dữ liệu vào>*

Output*: <dữ liệu ra>*

<Các câu lệnh>

End

Biểu diễn bằng Mã giả

5. Các cấu trúc:
   Cấu trúc chọn:
   if … **then** … [**else** …] **fi**
   Vòng lặp:
   while … **do**
   do … **while** (…)
   for … **do** … **od**
6. Một số câu lệnh khác:
   Trả giá trị về: **return** [giá trị]
   Lời gọi hàm: <Tên>(tham số)

Biểu diễn bằng Mã giả

- **Ví dụ**: Tìm phần tử lớn nhất trong mảng một chiều.

$a_{max} = a_0$;
i = 1;
while (i<n)
  if ($a_{max} < a_i$) $a_{max} = a_i$;
    i++;
end while;

Biểu diễn bằng Ngôn ngữ lập trình

- Dùng ngôn ngữ máy tính (C, Pascal,...) để diễn tả thuật toán, CTDL thành câu lệnh.
- Kỹ năng lập trình đòi hỏi cần học tập và thực hành (nhiều).
- Dùng phương pháp tinh chế từng bước để chuyển hoá bài toán sang mã chương trình cụ thể.

Độ phức tạp của Thuật toán

- Một thuật toán hiệu quả:
  - Chi phí cần sử dụng tài nguyên thấp: Bộ nhớ, thời gian sử dụng CPU, …
- Phân tích độ phức tạp thuật toán:
  - **N** là khối lượng dữ liệu cần xử lý.
  - Mô tả độ phức tạp thuật toán qua một hàm **f(N)**.
  - Hai phương pháp đánh giá độ phức tạp của thuật toán:
    - Phương pháp thực nghiệm.
    - Phương pháp xấp xỉ toán học.

```
sum = 0;

for(i= 0; i < n; i++)

  sum = sum + i;

return sum;
```

Phương pháp Thực nghiệm

- Cài thuật toán rồi chọn các bộ dữ liệu thử nghiệm.
- Thống kê các thông số nhận được khi chạy các bộ dữ liệu đó.
- *Ưu điểm*: Dễ thực hiện.
- *Nhược điểm*:
  - Chịu sự hạn chế của ngôn ngữ lập trình.
  - Ảnh hưởng bởi trình độ của người lập trình.
  - Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí.
  - Phụ thuộc vào phần cứng.

Phương pháp Xấp xỉ

- Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
- *Ưu điểm*: Ít phụ thuộc môi trường cũng như phần cứng hơn.
- *Nhược điểm*: Phức tạp.
- Các trường hợp độ phức tạp quan tâm:
  - Trường hợp tốt nhất (phân tích chính xác)
  - Trường hợp xấu nhất (phân tích chính xác)
  - Trường hợp trung bình (mang tích dự đoán)

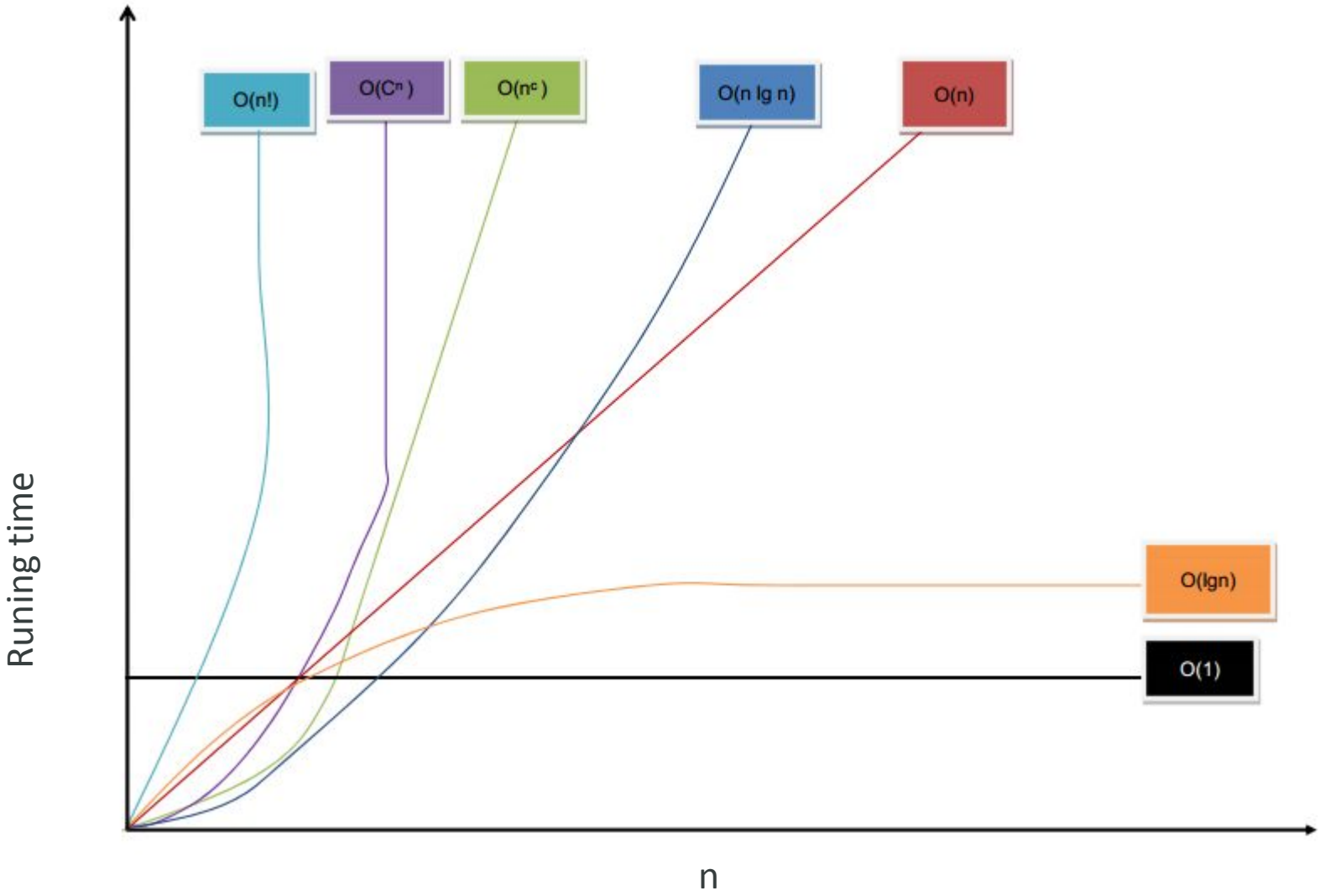
Sự phân lớp theo độ phức tạp của thuật toán

- **Sử dụng ký hiệu BigO**
  - Hằng số : O(c)
  - logN : O(logN)
  - N : O(N)
  - NlogN : O(NlogN)
  - N^2 : $O(N^2)$
  - N^3 : $O(N^3)$
  - 2^N : $O(2^N)$
  - N! :O(N!)

Độ phức tạp tăng dần

Linear Search O(N) vs Binary Search O(logN)

Độ phức tạp



Ví dụ về thuật toán

- Two algorithms for computing the Factorial
- Which one is better?

```
int factorial (int n) {
	if (n <= 1) return 1;
	else return n * factorial(n-1);
}
```

```
int factorial (int n) {
	if (n<=1) return 1;
	else {
		fact = 1;
		for (k=2; k<=n; k++)
			fact *= k;
		return fact;
	}
}
```

Examples of famous algorithms

- Shunting-yard algorithm
- Constructions of Euclid
- Newton's root finding
- Fast Fourier Transform
- Compression (Huffman, Lempel-Ziv, GIF, MPEG)
- DES, RSA encryption
- Simplex algorithm for linear programming
- Shortest Path Algorithms (Dijkstra, Bellman-Ford)
- Error correcting codes (CDs, DVDs)
- TCP congestion control, IP routing
- Pattern matching (Genomics)
- Search Engines

Example: Max Subsequence Problem

- Cho dãy các số nguyên (bao gồm cả số âm) $A_0, A_1, \ldots, A_{n-1}$, tìm tổng dương lớn nhất của dãy con (subsequence) $A_i, \ldots, A_j$.
- Bạn muốn có một dãy con liên tiếp với tổng dương lớn nhất?
- Ví dụ: -2, **11, -4, 13,** -5, -2
- Kết quả: 20 (subseq. A_1 -> A_3).
- Sau đây ta sẽ thảo luận về 4 thuật toán khác nhau với độ phức tạp lần lượt là: $O(n^3)$, $O(n^2)$, O(nlogn), and O(n).
- Với $n = 10^6$, thuật toán 1 có thể tốn > 10 năm; thuật toán 4 sẽ chỉ tốn chưa tới một phần giây.

Thuật toán 1

- Given $A_0, \ldots, A_n$, find the maximum value of $A_i + A_{i+1} + \ldots + A_j$
- 0 if the max value is negative

```
int maxSum = 0;

for(int i = 0; i < a.size( ); i++ )

    for(int j = i; j < a.size( ); j++ ) {

            int thisSum = 0;
            for(int k = i; k <= j; k++ )
                thisSum += a[ k ];
            if (thisSum > maxSum)
                maxSum = thisSum;
        }
return maxSum;
```

Time complexity: $O(n^3)$

Thuật toán 2

- Idea: Given sum from i to j-1, we can compute the sum from i to j in constant time.
- This eliminates one nested loop, and reduces the running time to $O(n^2)$.

```
into maxSum = 0;

for( int i = 0; i < a.size( ); i++ )

    int thisSum = 0;

    for( int j = i; j < a.size( ); j++ ) {
        thisSum += a[ j ];
        if( thisSum > maxSum )
            maxSum = thisSum;
    }
return maxSum;
```

Thuật toán 3

- This algorithm uses divide-and-conquer paradigm.
- Suppose we split the input sequence at midpoint.
- The max subsequence is entirely in the left half, entirely in the right half, or it straddles the midpoint.
- Example:

left half		right half					
4	-3	5	-2	-1	2	6	-2

- Max in left is 6 (A_0 through A_2); max in right is 8 (A_5 through A_6). But straddling max is 11 (A_0 thru A_6).

Thuật toán 3 (tt)

- Ví dụ:

Left half				Right hand			
4	-3	5	-2	-1	2	6	-2

- Max subsequences in each half found by recursion.
- How do we find the straddling max subsequence?
- Key Observation:
  - Left half of the straddling sequence is the max subsequence ending with -2.
  - Right half is the max subsequence beginning with -1.
- A linear scan lets us compute these in O(n) time.

Thuật toán 3: phân tích

- The divide and conquer is best analyzed through recurrence:

$$T(1) = 1$$

$$T(n) = 2T(n/2) + O(n)$$

- This recurrence solves to $T(n) = O(n \log n)$.

Thuật toán 3 (tt)

```
// Find the maximum possible sum in arr[l..r]
// such that arr[mid] is part of it
int maxCrossingSum(int arr[], int l, int mid, int r) {
    // Include e lements on left of mid.
    int sum = 0;
    int left_sum = INT_MIN;
    for (int i = mid; i >= l; i--) {
        sum = sum + arr[i];
        if (sum > left_sum) left_sum = sum;
    }
     // Include elements on right of mid
    sum = 0;
    int right_sum = INT_MIN;
    for (int i = mid+1; i <= r; i++) {
        sum = sum + arr[i];
        if (sum > right_sum) right_sum = sum;
    }
    // Return sum of elements on left and right of mid
    return left_sum + right_sum;
}
```

Thuật toán 3 (tt)

```
int max(int a, int b) { return (a > b)? a : b; }
int max(int a, int b, int c) { return max(max(a, b), c); }

// Returns sum of maximum sum subarray in arr[l..r]

int maxSubArraySum(int arr[], int l, int r) {

   // Base Case: Only one element

   if (l == r) return arr[l];

   int mid = (l + r)/2;

    /* Return maximum of following three possible cases

      a) Maximum subarray sum in left half

      b) Maximum subarray sum in right half

      c) Maximum subarray sum such that the subarray crosses the midpoint */

   return max(maxSubArraySum(arr, l, mid),

              maxSubArraySum(arr, mid+1, r),

              maxCrossingSum(arr, l, mid, r));
```

Thuật toán 4

Ví dụ: 2, 3, -2, 1, -5, 4, 1, -3, 4, -1, 2

```
int maxSum = 0, thisSum = 0;
for( int j = 0; j < a.size( ); j++ ) {
    thisSum += a[ j ];
    if ( thisSum > maxSum )
        maxSum = thisSum;
    else if ( thisSum < 0 )
        thisSum = 0;
}
return maxSum;
```

- Time complexity clearly $O(n)$
- But why does it work? I.e. proof of correctness

Proof of Correctness

- Max subsequence cannot start or end at a negative A_i.
- More generally, the max subsequence cannot have a prefix with a negative sum.

  Ex: -2 11 -4 13 -5 -2
- Thus, if we ever find that A_i through A_j sums to < 0, then we can advance i to j+1
  - Proof. Suppose j is the first index after i when the sum becomes < 0
  - The max subsequence cannot start at any p between i and j. Because A_i through A_{p-1} is positive, so starting at i would have been even better.

Thuật toán 4 (tt)

- The algorithm resets whenever prefix is < 0. Otherwise, it forms new sums and updates maxSum in one pass

Bài tập thực hành

Write program and analyze complexity:

- Perform polynomial additional and multiplication
- Determine the 2nd largest element in an array – the kth largest element in an array.
- Find longest increasing subsequence:
  - contiguous case: 2 algorithms with different complexity
    0, 8, 4, 12, 24, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15
  - uncontiguous case:
    0, 8, 4, 12, 2, 4, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

Câu hỏi và Bài tập

1. Trình bày tầm quan trọng của CTDL&GT?

2. Các tiêu chuẩn để đánh giá CTDL&GT?

3. Khi xây dựng giải thuật có cần quan tâm tới CTDL không? Tại sao?

4. Sử dụng các kiểu dữ liệu cơ bản trong C, xây dựng CTDL để lưu trữ đa thức có bậc tự nhiên n ($0 \le n \le 100$) trên trường số thực :

$$(a_i, x \in R)$$

$$fn(x) = \sum_{i=0}^{n} a_i x^i$$

Với CTDL đã được xây dựng, trình bày thuật toán và cài đặt chương trình để thực hiện các công việc sau:

Câu hỏi và Bài tập

- - Nhập xuất đa thức.

  - Tính giá trị của đa thức tại x_0 nào đó.

  - Tính tổng tích của 2 đa thức.

5. Tương tự như bài tập 4, nhưng đa thức trong trường số hữu tỷ Q (các số a_i và x là các phân số có tử số và mẫu số là các số nguyên).

6. Sử dụng kiểu dữ liệu cấu trúc trong C, xây dựng CTDL để lưu trữ trạng thái của các cột đèn giao thông (có 3 đèn: xanh, đỏ, vàng). Với CTDL đã được xây dựng, trình bày thuật toán và cài đặt chương trình để minh họa hoạt động của 2 cột đèn trên 2 tuyến đường giao nhau tại một ngã tư.

Chúc các em học tốt!