ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DATA STRUCTURES AND ALGORITHMS

Basic definitions and notations

CONTENT

- Overview of data structures and algorithms
- Pseudo codes
- Complexity analysis
- Big-O notations
- First example

Overview

- Data structures
  - The way we store and organize data in the memory to facilitate access and modifications

Overview

- Data structures
  - The way we store and organize data in the memory to facilitate access and modifications
- Algorithms
  - Sequence of computational steps that transform the input into the output

Overview

- Data structures
  - The way we store and organize data in the memory to facilitate access and modifications
- Algorithms
  - Sequence of computational steps that transform the input into the output
- Goal
  - How to design and implement efficiently data structures and algorithms for solving computation problems
- Applications
  - Database management systems
  - Combinatorial optimization
  - Artificial Intelligence, computer vision and machine learning
  - Operating systems
  - etc.

Pseudo code

- Describe algorithms in a simple way (free of programming language syntactic details)

```
Assignment
  x = <expression>;
  x ← <expression>;
```

```
For loop
  for i = 1 to n do{
     . . .
  }
```

```
While loop
 while i ≠ 100 do{
    . . .
 }
```

```
Condition
  if a < b then {
    . . .
  }
```

```
Procedures, funtions

  proc(a,b,x){
     . . .
     return ans;
  }
```

```
max(a[1..n]){
  ans = a[1];
  for i = 2 to n do
    if ans < a[i] then
      ans = a[i];
  return ans;
}
```

Pseudo code

- There might be several algorithms for sorting a sequence

```
selectionSort(a[1..n]){
   for k = 1 to n do{
    min = k;
    for j = k+1 to n do{
      if a[min] > a[j] then
        min = j;
    }
    swap(a[k],a[min]);
  }
}
```

```
insertionSort(a[1..n]){
   for k = 2 to n do{
     last = a[k];
     j = k;
   while(j > 1 and a[j-1] > last){
     a[j] = a[j-1];
     j--;
   }
   a[j] = last;
  }
}
```

Complexity analysis

- Analyze the efficiency of algorithms
  - Running times
  - Memory used
- Running times analysis
  - Experiments
  - Basic operations analysis

Complexity analysis

- Experiments
  - Write a program implementing the algorithm
  - Run the program in a specified machine with different inputs
  - Plot the (clock) running times

Complexity analysis

- Experiments
  - Write a program implementing the algorithm
  - Run the program in a specified machine with different inputs
  - Plot the (clock) running times

- Limitations of experimental running time measurements
  - Need to write the program
  - Clock running time depends on hardware configurations

Complexity analysis

- Measure running times in term of number of ***primitive operations*** executed (function of the input size)
- Identify the input size
  - Number of bits used for representing the input data
  - Or (high level) number of items of a given sequence, number of nodes, edges of a given graph, etc.
- Identify primitive operations to be analyzed

```
s = 0;
for i = 1 to n do
   s = s + a[i];
```

Primitive operations are assignment instructions → running time is $T(n) = n+1$

Complexity analysis

```
1.  insertionSort(a[1..n]){
2.    for j = 2 to n do{
3.      key = a[j];
4.      i = j-1;
5.      while i > 0 and a[i] > key do{
6.         a[i+1] = a[i];
7.         i = i - 1;
8.      }
9.      a[i+1] = key;
10.   }
11. }
```

Line	cost	times
2	c_2	n
3	c_3	n-1
4	C_4	n-1
5	C_5	$\sum_{j=2}^{n} t_j$
6	C_6	$\sum_{j=2}^{n} (tj - 1)$
7	C_7	$\sum_{j=2}^{n} (tj - 1)$
9	c_9	n-1

Denote t_j: number of times the while loop test (line 5) is executed for each value of j (outer for loop)

Running time $T(n) = c_2 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(tj - 1) + c_9(n-1)$

Complexity analysis

Running time $T(n) = c_2 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(tj - 1) + c_9(n-1)$

- Best case: the sequence is already sorted, $t_j = 1$ (j = 2,…,n)

→ $T(n)$ has the form $an + b$ (linear)

- Worst case, the sequence is in reverse sorted order, $t_j = j$ (j = 2,…, n)

→ $T(n)$ has the form $an^2 + bn + c$ (quadratic)

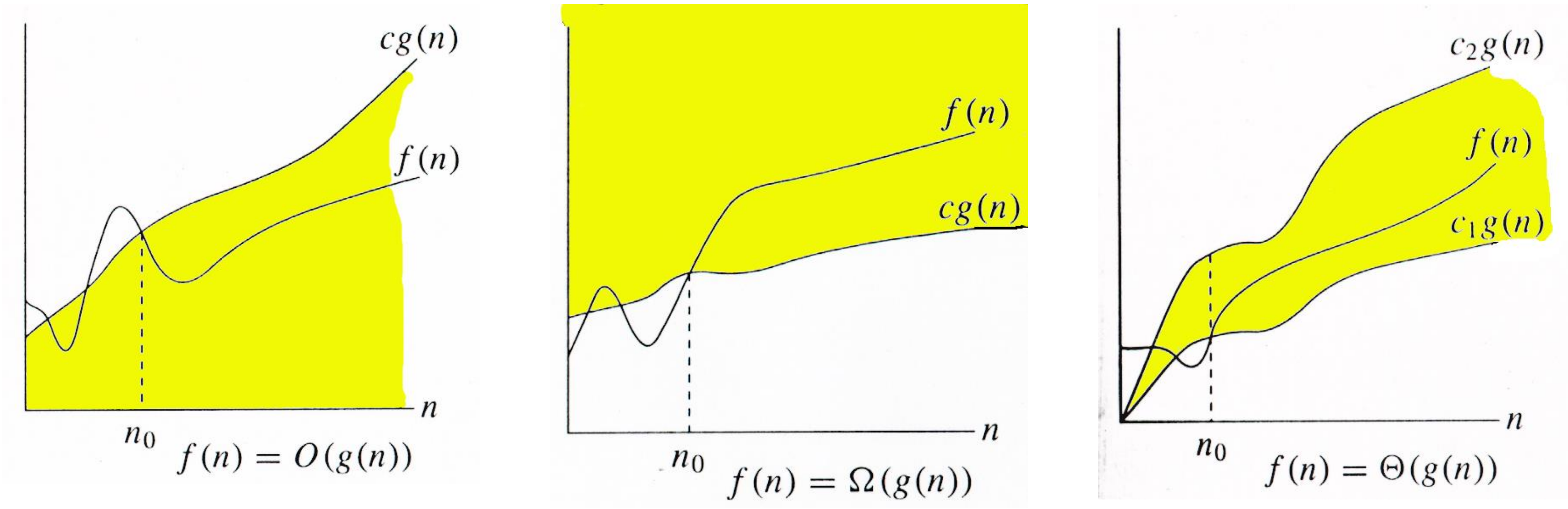
Complexity analysis

- **Growth rate**: dominant term (e.g., n^2 in $an^2 + bn + c$) describes the order of growth of the running time
- Growth rate is an indicator on how fast the running time increases when the input size increases
- Some frequent growth rates

Logarithmic algorithms	Log n
Linear algorithms	n
Quadratic algorithms	n^2
Polynomial algorithms	n^k
Exponential algorithms	c^n

Big O - notation

- Let g(n) be a function from *N* to *R*
  - $O(g(n)) = \{f(n) \mid \exists\, c > 0 \text{ and } n_0 \text{ s.t. } 0 \le f(n) \le cg(n)\ \forall n \ge n_0\}$
  - $\Omega(g(n)) = \{f(n) \mid \exists\, c > 0 \text{ and } n_0 \text{ s.t. } 0 \le cg(n) \le f(n)\ \forall n \ge n_0\}$
  - $\Theta(g(n)) = \{f(n) \mid \exists\, c_1, c_2 > 0 \text{ and } n_0 \text{ s.t. } c_1 g(n) \le f(n) \le c_2 g(n)\ \forall n \ge n_0\}$



Big O - notation

- Examples
  - $10^3n^2 + 2n + 10^6 \in O(n^2)$
  - $10^3n^2 + 2n + 10^6 \in O(n^3)$
  - $10^3n^2 + 2n + 10^6 \in \Theta(n^2)$
  - $10^3n^2 + 2n + 10^6 \in \Omega(n)$
  - $10^3n^2 + 2n + 10^6 \in \Omega(nlogn)$

Big O - notation

- Let *f* and *g* are non-negative valued functions from *N* to *R*
  - If $f(n) \in \Theta(g(n))$, then $f(n) \in \Omega(g(n))$ and $f(n) \in O(g(n))$
  - If $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, then $f(n) \in \Theta(g(n))$
  - If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f(n) \in O(g(n))$

First example: max subsequence

- Given an integers sequence $a = (a_1, a_2, \ldots, a_n)$. A subsequence of a is defined to be $a_i, a_{i+1},\ldots,a_j$. The weight of a subsequence is the sum of its elements. Find the subsequence having the highest weight
- Example: a = 2, -10, **11, -4, 13**, -5, 2 then **11, -4, 13** is the subsequence having the highest weight

First example : max subsequence

```
maxSubSeq3(a[1..n]){
  ans = -∞;
  for i = 1 to n do{
    for j = i to n do{
       s = 0;
       for k = i to j do
          s = s + a[k];
       if s > ans then
         ans = s;
    }
  }
  return ans;
}
```

Naïve: running time $O(n^3)$

```
maxSubSeq2(a[1..n]){
  ans = -∞;
  for i = 1 to n do{
    s = 0;
    for j = i to n do{
       s = s + a[j];
       if s > ans then
         ans = s;
    }
  }
  return ans;
}
```

Improvement: Running time $O(n^2)$

First example : max subsequence

- Dynamic programming
  - Denote s[i]: weight of the largest (highest weight) subsequence of a_1, . . ., a_i such that the last element is a_i.

```
maxSubSeq1(a[1..n]){
  s[1] = a[1];
  ans = s[1];
  for i = 2 to n do{
     if s[i-1] > 0 then
        s[i] = s[i-1] + a[i];
     else s[i] = a[i];
     if ans < s[i] then
       ans = s[i];
  }
  return ans;
}
```

Best algorithm: Running time O(n)

- s[1] = a_1
- s[i] = $\begin{cases} s[i-1] + a_i, & \text{if } s[i-1] > 0 \\ a_i, & \text{otherwise} \end{cases}$

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY

SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!