جامعة طرابلس ـ كلية تقنية المعلومات

Design and Analysis Algorithms

تصميم وتحليل خوارزميات

ITGS301

المحاضرة الثالثة: Lecture 3

Tripoli University - Faculty of Information Technology
Marwa Solla
m.solla@uot.edu.ly

خريف 2024

LIMIT TECHNIQUE
FOR COMPARING GROWTH RATES

Another way of checking if a function $f(n)$ grows faster or slower than another function $g(n)$ is to divide $f(n)$ by $g(n)$ and take the limit $n \to \infty$ as follows

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

We use limits as n tends to infinity. That is,

f grows slower than g as x $\longrightarrow \infty$

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, $\quad f(n) = O(g(n))$.

f grows faster than g as x $\longrightarrow \infty$

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, $\quad f(n) = \Omega(g(n))$.

f and g are growth at the same rate

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$, and $C \neq 0$ $\quad f(n) = \Theta(g(n))$.

USING THE LIMIT METHOD: EXERCISE 1

- Compare growth rate of n^2 and $n^2 - 7n - 30$
- $\lim_{n\to\infty} \frac{n^2-7n-30}{n^2}$
- $= \lim_{n\to\infty} (1 - \frac{7}{n} - \frac{30}{n^2})$
- $= 1$
- So $n^2 - 7n - 30 \in \Theta(n^2)$

Examples:

$f(n) = \sqrt{n}$ $\qquad g(n) = 2n$

$$\frac{f(n)}{g(n)} = \frac{\sqrt{n}}{2n} = \frac{1}{2\sqrt{n}} \xrightarrow[n\to\infty]{} 0$$

$5n^2 - 4n - 100$ $\quad g(n) = n^2$

$$\frac{f(n)}{g(n)} = \frac{5n^2-4n-100}{n^2} = 5 - \frac{4}{n} - \frac{100}{n^2} \xrightarrow[n\to\infty]{} 5$$

Examples:

$f(n) = \sqrt[3]{n} \qquad g(n) = \sqrt{n}$

$$\frac{f(n)}{g(n)} = \frac{\sqrt[3]{n}}{\sqrt{n}} = \frac{n^{\frac{1}{3}}}{n^{\frac{1}{2}}} = n^{\frac{1}{3}-\frac{1}{2}} = n^{-\frac{1}{6}} = \frac{1}{n^{\frac{1}{6}}} \xrightarrow[n\to\infty]{} 0$$

$f(n) = n^2 \qquad g(n) = n\log n$

$$\frac{f(n)}{g(n)} = \frac{n^2}{n\log n} = \frac{n}{\log n} \xrightarrow[n\to\infty]{} \infty$$

Analysis of Time Complexity

(1) Determine the **input size** (n)

(2) Determine the **basic operations**

(3) Let $\boldsymbol{c(n)}$ be the **maximum count** of the **basic operations** as function of n

(4) Let $\boldsymbol{d(n)}$ be the **minimum count** of the **basic operations** as function of n

(5) The **upper bound** of the time complexity is $O(c(n))$

(6) The **lower bound** of the time complexity is $\Omega(d(n))$

(7) If $O(c(n)) = \Omega(d(n))$, then the **exact bound** of the time complexity is $\Theta(c(n))$

Main Rules of Asymptotic Notations

1. Drop constant factors
   - ✓ $6n - 3 = O(n)$
   - ✓ $2n^2 + 1000 = O(n^2)$
   - ✓ $4n \log n + 10 = O(n \log n)$
2. Drop lower-order terms
   - ✓ $n^3 + n^2 + n + 1 = O(n^3)$
   - ✓ $n + \log n = O(n)$
   - ✓ $n \log n + n = O(n \log n)$
   - ✓ $\log n + \log\log n = O(\log n)$

Big Oh Rules:

1. Ignore constant factors.
2. IF we have 2 functions f1(n) , f2(n) and f1(n) = O(g1(n)) , f2(n) = O(g2(n)) then

   f1(n) * f2(n) = O (g1(n) * g2(n)).

Ex: f1(n) = O(n^2) and f2(n) = O(n)

f1(n) * f2(n) = O(n^2 * n)

= O(n^3)

3. if we have 2 functions f1(n) , f2(n) and f1(n) = O(g1(n)) , f2(n) = O(g2(n)) then

f1(n) + f2(n) = Max(g1(n) , g2(n))

= O (g1(n) + g2(n)).

Ex: f1(n) = $O(n^2)$ and f2(n) = $O(n^3)$

f1(n) + f2(n) = Max($O(n^2)$, $O(n^3)$)

= $O(n^2) + O(n^3)$

= $O(n^3)$.

Analysis of Time Complexity

Counting the Number of Operations

1. The running time equals the number of primitive operations (steps) executed before termination.
2. Each operation takes a certain time.

❑ *Analysis of Loops:*

- Simple Loops: The running time of a **for** loop is at most the running time of the **statements inside the loop** times **the number of iterations**.

Example 1 : O(n) Loops

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

Analyzing :

```
sum = 0;        excuted only 1 time  :: O(1)
for( i = 0; i < n; i++)
          // i = 0; executed only once: O(1)
                    // i < n;  n + 1 times       O(n)
                    // i++         n times       O(n)
total time of the loop heading:
          O(1) + O(n) + O(n) = O(n)

sum = sum + i;  // executed n times,          O(n)
```

The time required for this algorithm equals: O(1) + O(n) + O(n) = O(n).

Example 2 O(n) Loops

```
int sum = 0;
int i = 0;
while (i < n) {
   sum++;
   i++;
}
```

Analyzing :

```
int sum = 0;          // 1 time
int i = 0;            // 1 time
while (i < n) {       // n+1 times
  sum++;              // n times
  i++;                // n times
}
```

Hence, T(n) = 3*n+3 = O(n)

Example 3 O(1) Loops

A loop or recursion that runs a constant number of times is considered as O(1).

```
Int sum = 0;
for (int i = 1; i <= 10; i++) {
    sum = sum + a[i]
}
```

○ Nested Loop:

Time complexity of nested loops is equal to the number of times the innermost statement is executed.

Example 4 O(n^2) Loops

```
sum = 0;
for( i = 0; i < n; i++)
  for( j = 0; j < n; j++)

     sum++;
```

The running time = O(1) + O(n*n) + O(n)

$= O(1) + O(n^2) + O(n)$

$= O(n^2)$

Consecutive program fragments

The total running time is the maximum of the running time of the individual fragments

Example 5 $O(n^2)$ Loops

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


sum = 0;
for( i = 0; i < n; i++)
   for( j = 0; j < 2n; j++)
        sum++;
```

If statement

```
IF Condition
    S1;
else
    S2;
```

The running time is the maximum of the running times of **S1** and **S2**.

Exercises

What is time complexity of following ?

1.
```
if (a[i] == x)
        return 1;
   else
        return -1;
```

2.
```
sum = 0;        for( i = 0; i < 2n; i++)
   for( j = 0; j < n ; j++)
     for( k = 0; k < n; k++)
       sum++;
```

Exercises

3.
```
int sum = 0;
int i = 0;
while (i < n) {
    int a = 0;
    while (a < i) {
        sum++;
        a++;
    }
  i++;
}
```

4.
```
Val=0
for (int i = 1; i < n; i = i*2)
      Val= Val + i // some O(1) operation
```

Exercises

What is time complexity of fun()?

```
int fun(int n){
  int count = 0;
  for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j++) {
      count = count + 1;
    }
  }
  return count;
}
```

Worst and Best Case Analysis

Worst Case Analysis

- ✓ In worst case analysis, we calculate upper bound on running time of an algorithm.
- ✓ We must know the case that causes maximum number of operations to be executed.

Example 6: Worst Case Analysis of Linear Search

- ✓ For Linear Search, the worst case happens when the element to be searched (x) is not present in the array.
- ✓ In this case, the algorithm compares it with all the elements of A one by one.
- ✓ Therefore, worst case time complexity of linear search would be O(n).

```
// INPUT: an array A[1..n] of n integers and an interger x
// OUTPUT: Index i if A[i] = x for 1 <= i <= n, and 0 otherwise
int LinearSearch(int A[], int n, int x) {
  for (int i = 1; i <= n; i++) {
    if (A[i] == x) return i;
  }
  return 0;
}
```

Worst and Best Case Analysis

Best Case Analysis

- ✓ In best case analysis, we calculate lower bound on running time of an algorithm.
- ✓ We must know the case that causes minimum number of operations to be executed.

Example 7: Best Case Analysis of Linear Search

- ✓ In the linear search algorithm, the best case occurs when x is present at the first location.
- ✓ The number of operations in the best case is constant (not dependent on n).
- ✓ So, time complexity in the best case would be $\Omega(1)$

```
// INPUT: an array A[1..n] of n integers and an interger x
// OUTPUT: Index i if A[i] = x for 1 <= i <= n, and 0 otherwise
int LinearSearch(int A[], int n, int x) {
  for (int i = 1; i <= n; i++) {
    if (A[i] == x) return i;
  }
  return 0;
}
```

Logarithms and properties

In algorithm analysis we often use the notation "log n" without specifying the base

Binary logarithm: $\lg n = \log_2 n$

Natural logarithm: $\ln n = \log_e n$

$\lg^k n = (\lg n)^k$

$\lg \lg n = \lg(\lg n)$

$$\log x^y = y \log x$$

$$\log xy = \log x + \log y$$

$$\log \frac{x}{y} = \log x - \log y$$

$$a^{\log_b x} = x^{\log_b a}$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Some Simple Summation Formulas

- **Arithmetic series:** $$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$
- **Geometric series:** $$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$
  - **Special case: x < 1:** $$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$
- **Harmonic series:** $$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$
- **Other important formulas:** $$\sum_{k=1}^{n} \lg k \approx n \lg n$$ $$\sum_{k=1}^{n} k^p = 1^p + 2^p + \ldots + n^p \approx \frac{1}{p+1} n^{p+1}$$

$1^2 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n+1)/6$

The End .