Analysis of Algorithms
Script of Lecture 2

Amalia Duch

September 25, 2019

Contents

- Quick reminder on asymptotic notations $\mathcal{O}$, Ω, Θ.
- Rules to calculate the cost of iterative algorithms.
- Examples
- Master theorems to calculate the cost of recursive algorithms.
- Examples

Analysis of Iterative Algorithms

- The cost of elementary operations (see previous class) is $\Theta(1)$.
- Sequential composition: Given two fragments of code s_1, s_2 with cost f_1 and f_2 respectively, the cost of the fragment:

```
s 1 ;
s 2 ;
```

  is $f_1 + f_2$

- Conditional composition: Let A be an expression with cost to evaluate f_a, and s_1 and s_2 two fragments of code with cost f_1 and f_2 respectively, the cost, in worst case, of the fragment:

```
if (A) {
        s1;
  }
else {
        s2;
  }
```

  is: $f_a + \max\{f_1, f_2\}$

- Iterative Composition: Let A be an expression with cost to evaluate it at the i-th iteration g_i, and it is a code fragment with cost to the $i-th$ iteration f_i, the cost, if not worse, of the fragment:

```
while (A) {
        s;
  }
```

  if n iterations are performed is: $\sum_{i=1}^{n}(f_i + g_i) = O(n(f+g))$ with $f = \max\{f_i\}$ and $g = \max\{g_i\}$.

Examples

Calculation of the cost of different loops.

Analysis of recursive algorithms

The cost (in worst case, average, $\ldots$) of a recursive algorithm $T(n)$ satisfies a recurrent equation: this is, $T(n)$ will depend on the value of T for smaller values of n. Frequently, the recurrence has one of the following forms:

$$T(n) = a \cdot T(n-c) + g(n),$$
$$T(n) = a \cdot T(n/b) + g(n).$$

The first one corresponds to algorithms that have a non recursive part with cost $g(n)$ and do a recursive calls with subproblems of size $n-c$, with c a constant.

The second one corresponds to algorithms with a non recursive part of cost $g(n)$ that do a recursive calls with subproblems of size (approximately) n/b, with $b > 1$.

Theorem 1. *Let $T(n)$ be the cost (worst case, average case, ...) of a recursive algorithm that satisfies the recurrence:*

$$T(n) = \begin{cases} f(n) & \text{if } 0 \leq n < n_0 \\ a \cdot T(n-c) + g(n) & \text{if } n \geq n_0, \end{cases}$$

where n_0 is a constant, $c \geq 1$, $f(n)$ is an arbitrary function and $g(n) = \Theta(n^k)$ for a constant $k \geq 0$.

Then

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } a < 1 \\ \Theta(n^{k+1}) & \text{if } a = 1 \\ \Theta(a^{n/c}) & \text{if } a > 1. \end{cases}$$

Theorem 2. *Let $T(n)$ be the cost (worst case, average case, ...) of a recursive algorithm that satisfies the recurrence:*

$$T(n) = \begin{cases} f(n) & \text{if } 0 \leq n < n_0 \\ a \cdot T(n/b) + g(n) & \text{if } n \geq n_0, \end{cases}$$

where n_0 is a constant, $b > 1$, $f(n)$ is an arbitrary function and $g(n) = \Theta(n^k)$ for a constant $k \geq 0$.

Let $\alpha = \log_b a$. Then

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } \alpha < k \\ \Theta(n^k \log n) & \text{if } \alpha = k \\ \Theta(n^\alpha) & \text{if } \alpha > k. \end{cases}$$

Examples

- Binary search: $\Theta(\log(n))$ in the worst case.
- Fast exponentiation: $\Theta(\log(n))$.
- Fibonacci numbers: $\mathcal{O}(2^n)$ and $\Omega(2^{(n/2)})$.