Algorithm analysis

Presentation by Antonio-Gabriel Sturzu

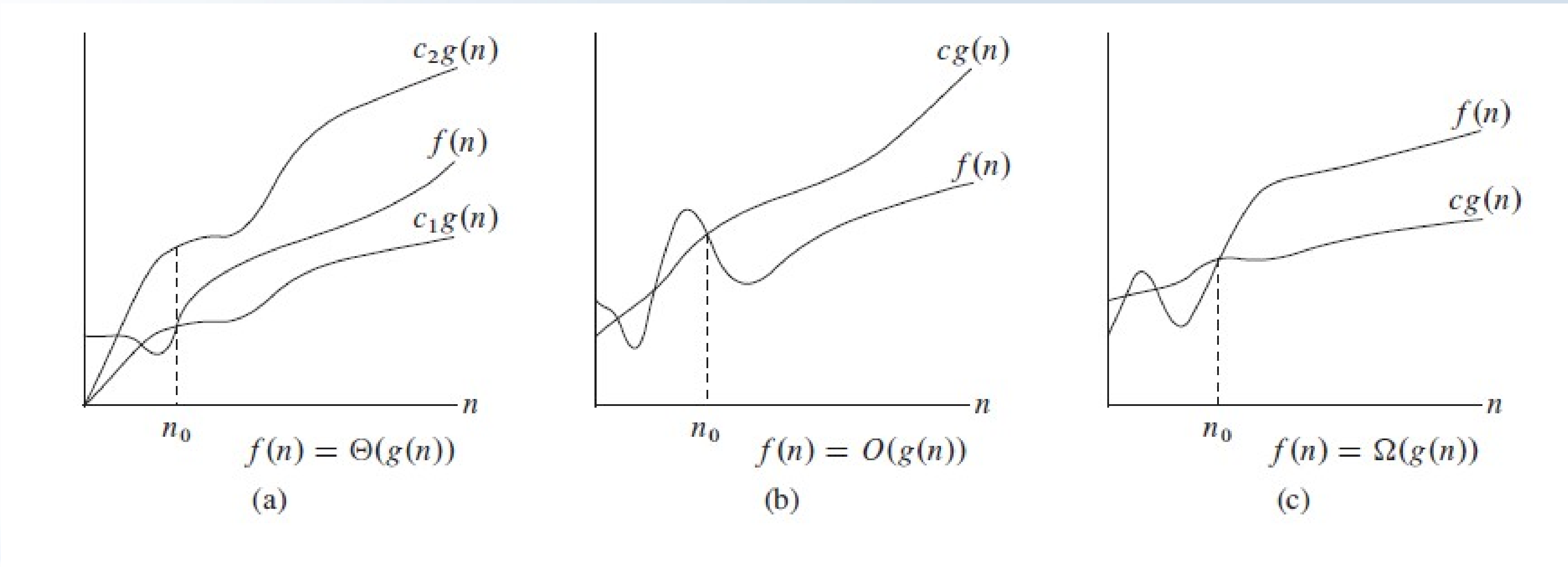
® 1&1 Internet AG 2010

Agenda

- Asymptotic notation
- Examples
- Substitution method
- Recursion tree method
- Master Theorem
  - Examples

® 1&1 Internet AG 2010

Big O, Θ and Ω notation



® 1&1 Internet AG 2010

Big O, Θ and Ω notation

- In practice we are mainly interested in a tight upper bound for the worst case scenario and this is where the big O notation comes to the rescue
- $O(g(n))=\{ f(n)$: there exist constants $c>0$ and $n_0>0$ such that $0<=f(n)<=cg(n)$ for all $n>=n_0 \}$
- $\Omega(g(n))=\{ f(n)$: there exist constants $c>0$ and $n_0>0$ such that $0<=cg(n)<=f(n)$ for all $n>=n_0 \}$
- $\Theta(g(n))=O(g(n)) \cap \Omega(g(n))$

Examples

- In all the examples when we say $f(n)=O(g(n))$ we actually mean $f(n)\in O(g(n))$ because O is a set
- $2n^2= O(n^3)$ ($c=1$, $n_0=2$)
- $\log_2 n= O(\log_3 n)$ ($c=\log_2 3$, $n_0=2$)
- $n^2=\Omega(n)$ ($c=1$, $n_0=1$)
- $2^{n+10}= \Theta(2^n)$ ($c=2^{10}$, $n_0=1$)

® 1&1 Internet AG 2010

Examples

- A nice trick is that in practice we can ignore the lower order terms of an asymptotically positive function because for a sufficient larger n they will get dominated by the highest order term
- Also the coefficient of the highest order term can be ignored because it will only modify by a constant factor the c constant in the big O notation

® 1&1 Internet AG 2010

Examples

- A clear example that proves this is that given $f(n)= a_p n^p + a_{p-1}n^{p-1}+.......+a_1 n+a_0$, $a_p>0$ and all the coefficients are constant then $f(n)=\theta(n^p)$
- In practice a constant will be expressed by $\theta(1)$ or $O(1)$ since it is a degree 0 polynomial
- The above notation could also mean a constant function

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- Guess the form of the solution
- Verify by mathematical induction
- Solve for constants

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- Example:
- $T(n)=4T(n/2)+n$ (we assume that $T(1)=1$)
- We guess that the upper bound is $O(n^3)$
- Now begins the induction phase
- First we prove the base case
- The base case is that $T(1)=O(n^3)$
- It is easy to see that the inductive assumption holds for $c>=1$

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- The general case is to prove that $T(n)<=cn^3$
- We use the inductive hypothesis that $T(m)<=cm^3$ for $m<n$
- Using this we have $T(n)=4T(n/2)+n<=4c(n/2)^3+n$
- Then $T(n)<=(c/2)n^3+n<=cn^3$
- From here we get the condition $(c/2)n^3-n>=0$ and for any $n>=1$ we have $(c/2)n^2>=1$ and from here $c>=2/n^2$
- We can choose $c=2$ and $n_0=1$ and the proof is complete

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- Sometimes the upperbound that we guess isn't tight enough
- We guess that $T(n)=O(n^2)$
- First we try to prove the base case $T(1)=O(n^2)$ for $c>=1$
- We will try to prove that $T(n)<=cn^2$
- The inductive hypothesis is $T(k)<=ck^2$, $k<n$
- $T(n)=4T(n/2)+n<=4c(n/2)^2+n= cn^2+n=O(n^2)$

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- The proof is wrong because we are trying to prove that $T(n)<=cn^2$
- From the inductive hypothesis we have that $T(n)<=cn^2+n$
- If we impose $cn^2+n<=cn^2$ it implies that $n<=0$ which is false !
- So we need to make a different inductive assumption in order to prove that $T(n)=O(n^2)$

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- We assume that $T(k) \le ck^2 - k$, $c > 0$, $k < n$
- $T(n) = 4T(n/2) + n \le 4c(n/2)^2 - 4n/2 + n = cn^2 - n \le cn^2$ for every $c > 0$ and $n > 0$
- We choose $c = 2$ and $n_0 = 1$
- All that is left to do is to prove the base case $T(1) \le 2 - 1 = 1$ which holds and the proof is complete

® 1&1 Internet AG 2010

Substitution method for solving recurrences

- Sometimes if the base case doesn't hold we can take advantage of the Big O definition that says that we can prove the inequality for $n>=n_0$ which we can conveniently choose

® 1&1 Internet AG 2010

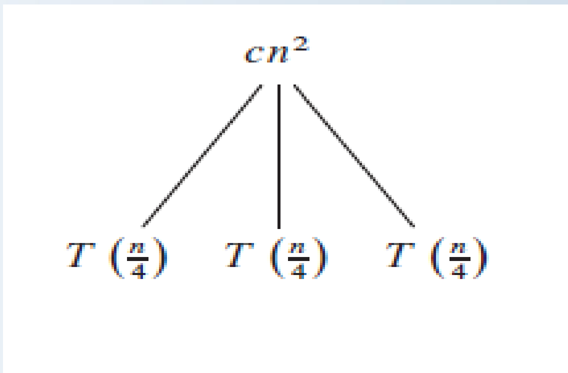
Recursion tree method

- Models the cost of a recursive algorithm
- Is intuitive but sometimes it can be somewhat imprecise
- Each node in the tree represents the cost of a subproblem in the recursive invocation of the algorithm
- Can be used as a guess for the substitution method if we were sloppy in the recursion tree method

® 1&1 Internet AG 2010

Recursion tree method

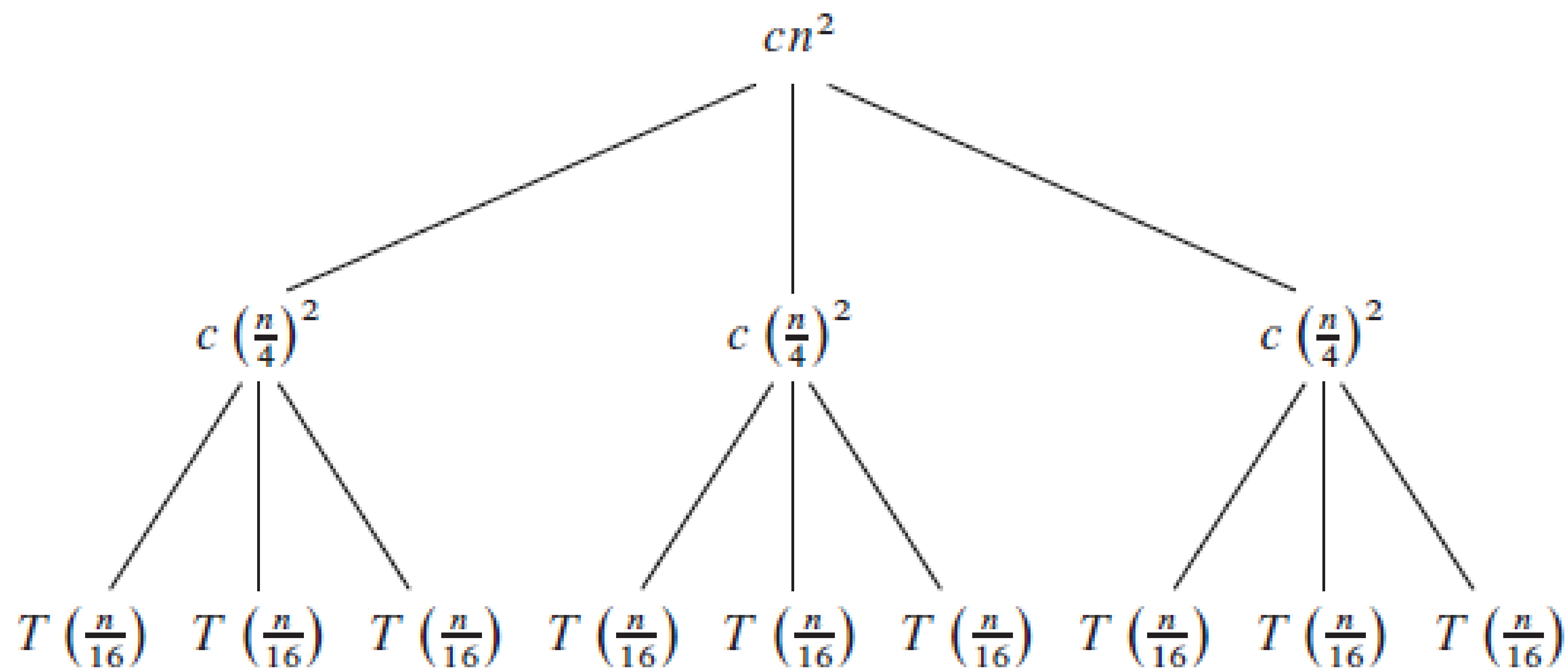
- Example
- $T(n) = 3T(n/4) + \Theta(n^2)$
- First we write $T(n) = 3T(n/4) + cn^2$
- In the first recursive invocation of the algorithm the tree looks like this:



® 1&1 Internet AG 2010

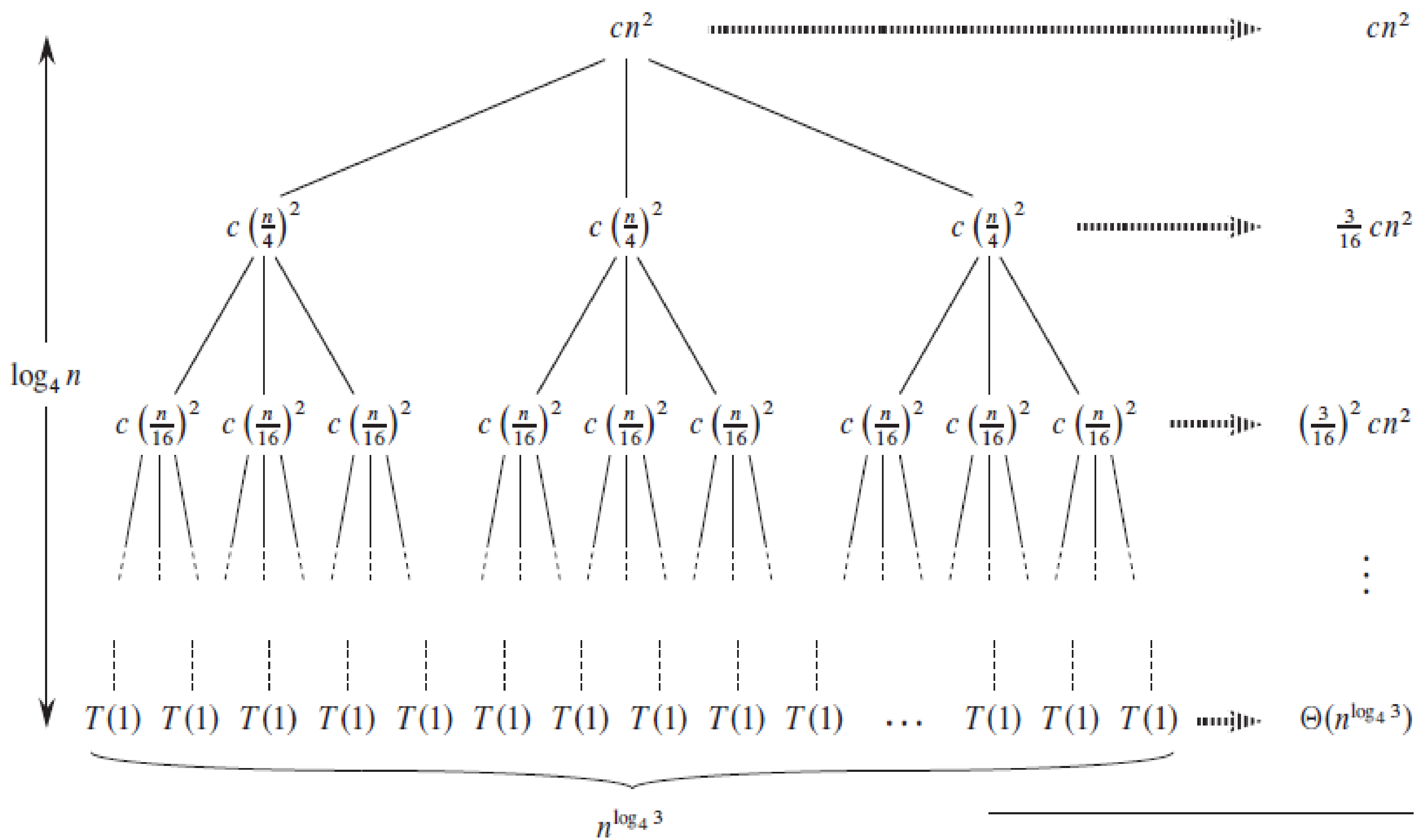
Recursion tree method

- In the next recursive call the tree looks like this:



® 1&1 Internet AG 2010

Recursion tree method



® 1&1 Internet AG 2010

Recursion tree method

- The number of subproblems at level j is 3^j
- The work done by a subproblem at level j is $c(n/4^j)^2$ which is equivalent to $cn^2(1/16)^j$
- The total work done at level j will be $cn^2(3/16)^j$
- At the last level the number of nodes is $3^{\log4(n)}$ which is equivalent to $n^{\log4(3)}$

® 1&1 Internet AG 2010

Recursion tree method

- Therefore the total work done will be

$$cn^2 \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i + \theta\left(n^{\log_4 3}\right) < cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \theta\left(n^{\log_4 3}\right)$$

$$= \frac{1}{1-\frac{3}{16}} cn^2 + \theta\left(n^{\log_4 3}\right) = \frac{16}{13} cn^2 + \theta\left(n^{\log_4 3}\right) = O\left(n^2\right)$$

® 1&1 Internet AG 2010

Master Theorem

- Let a>=1 and b>1 be constants and let f(n) be a nonnegative function with T(n)=aT(n/b)+f(n)
- Then T(n) has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

® 1&1 Internet AG 2010

Examples

- $T(n)=2T(n/2) + n$
- We have a=2, b=2, f(n)=n and we are in case 2
- So $T(n)=\theta(n\lg n)$
- $T(n)=8T(n/2) + n^2$
- We have a=8, b=2, $f(n)=n^2$ and we are in case 1
- So $T(n)=\theta(n^3)$
- $T(n)=3T(n/4) + n\lg n$
- We have a=3, b=4, f(n)=nlgn and we are in case 3
- So $T(n)=\theta(n\lg n)$

® 1&1 Internet AG 2010

Examples

- A more tricky example
- $T(n)=2T(\sqrt{n})+\log_2 n$
- The idea is to make a change of variable
- Consider $m=\log_2 n$ then the recurrence becomes
- $T(2^m)=2T(2^{m/2})+m$
- Let $S(m)=T(2^m)$. The relation becomes $S(m)=2S(m/2)+m$
- The time complexity will be $\theta(\lg n \lg(\lg n))$

® 1&1 Internet AG 2010