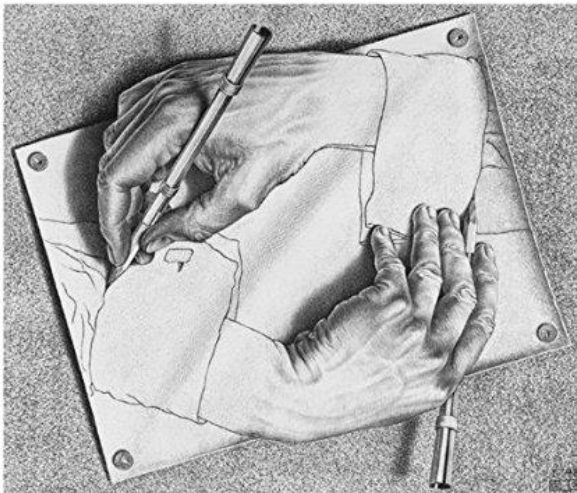


# Computational Complexity

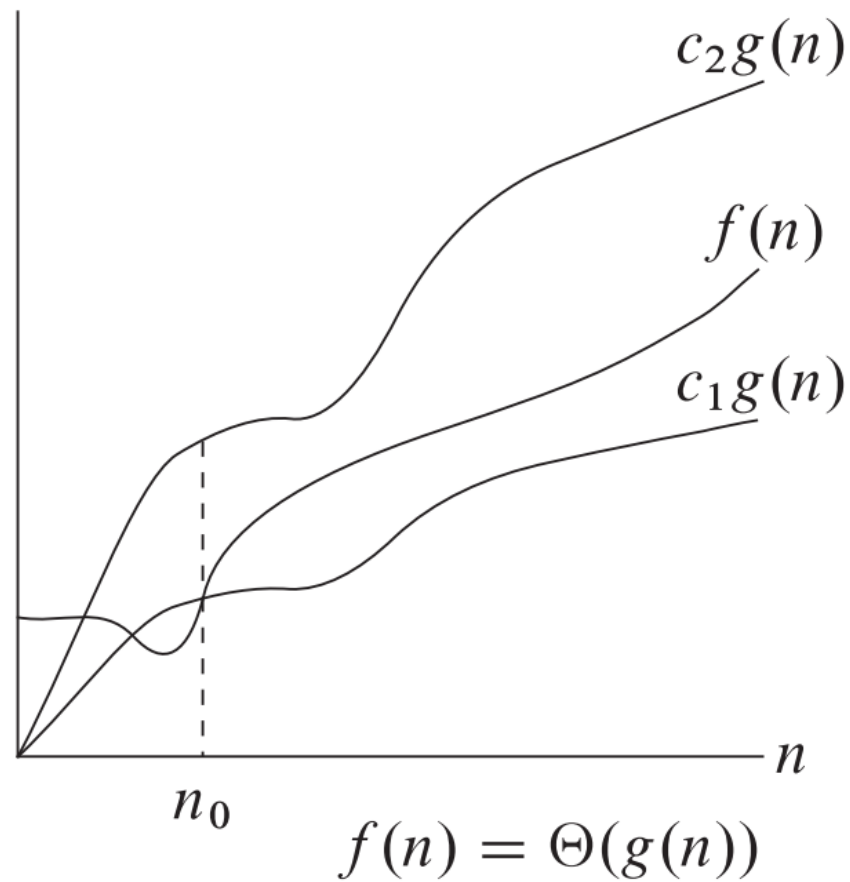


Prof Dr Marko Robnik Šikonja  
Analysis of Algorithms and Heuristic Problem Solving  
Edition 2025

# Asymptotically tight bound $\Theta$

- Given function  $g(n)$ , we denote with  $\Theta(g(n))$  a set of functions:
- $\Theta(g(n)) = \{ f(n); \exists c_1, c_2, n_0 > 0, \forall n > n_0: 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$
- notation used is  $f(n) \in \Theta(g(n))$  and more frequently  $f(n) = \Theta(g(n))$
- $g(n)$  is asymptotically tight bound for  $f(n)$
- assumption:  $g(n)$  is asymptotically positive function

$$\Theta(g(n))$$



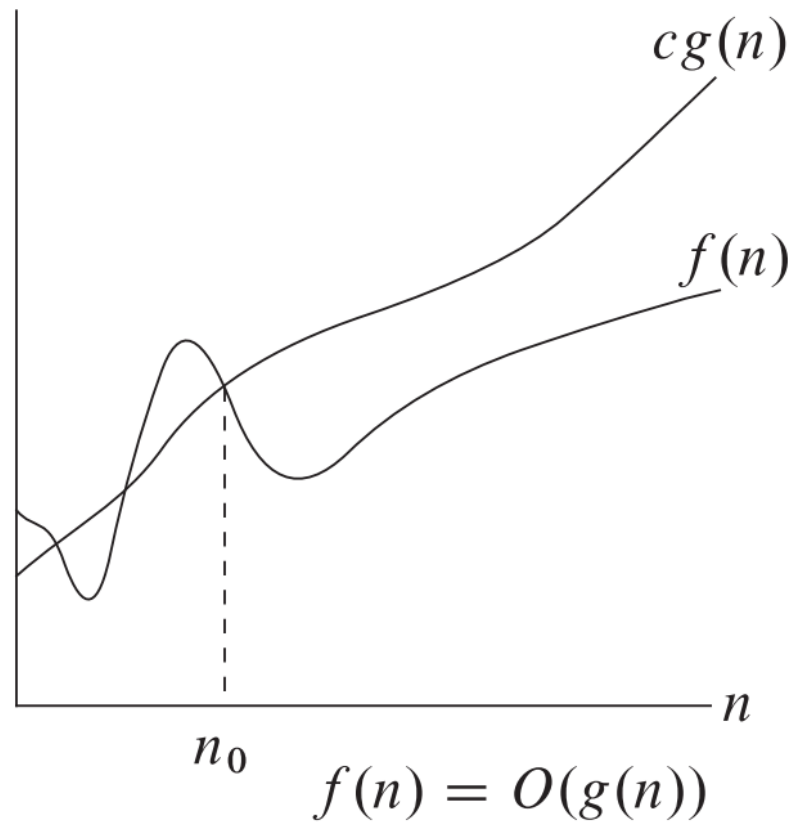
# An example

- Let us show that  $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- find  $c_1, c_2, n_0$
- Home work:
  - show that  $an^2 + bn + c = \Theta(n^2)$
  - show for all polynomials  $p(n), p(n) = \sum_{i=0}^d a_i n^i, a_d > 0$ , that  $p(n) = \Theta(n^d)$
  - show  $6n^3 \neq \Theta(n^2)$
- we denote constant function as  $\Theta(n^0) = \Theta(1)$

# Asymptotical upper bound $O$

- for functions  $g(n)$  we write  $O(g(n))$  to be a set of functions for which the following holds:
- $O(g(n)) = \{ f(n); \exists c, n_0 > 0, \forall n > n_0: 0 \leq f(n) \leq cg(n) \}$
- we use notation  $f(n) \in O(g(n))$  or more frequently  $f(n) = O(g(n))$
- $g(n)$  is asymptotical upper bound for  $f(n)$
- attention! the literature tend to be imprecise in this notation
- use also as an anonymous function, for example  
 $T(n) = 2 T(n/2) + O(n)$

$O(g(n))$



# Alternative definitions

- for upper bound

$$f(n) = O(g(n)) \iff \lim_{n \rightarrow \infty} \frac{|f(n)|}{g(n)} < \infty \text{ and the limit exists}$$

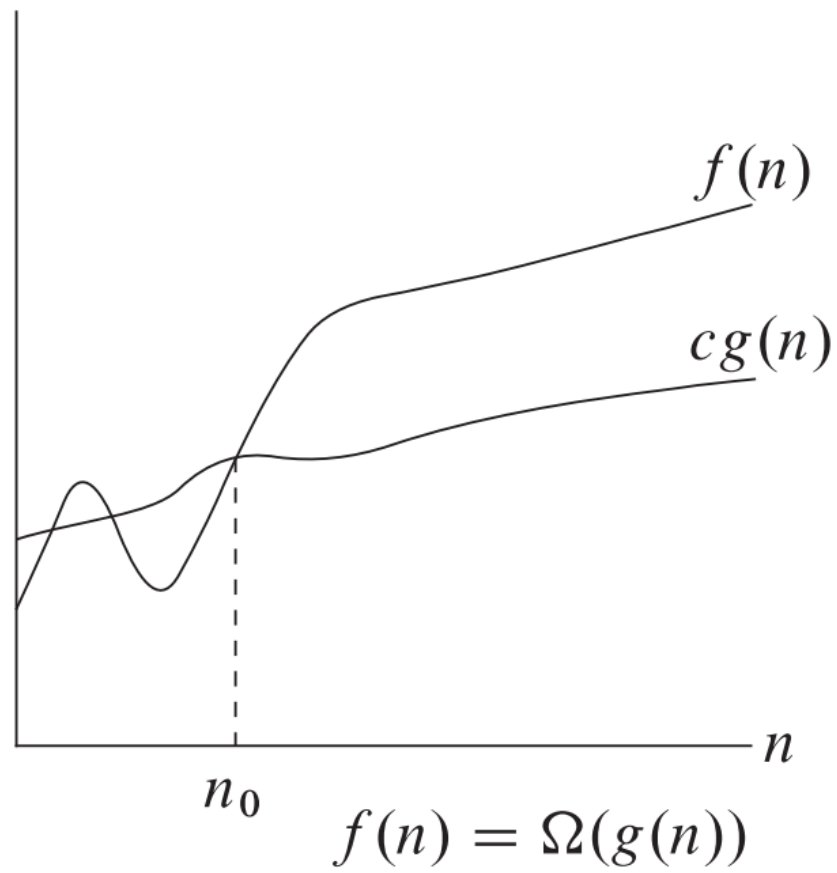
# Examples

- Show  $\frac{1}{2} n^2 - 3n = O(n^2)$
- Show at home:
  - $an^2 + bn + c = O(n^2)$
  - $an + c = O(n^2)$

# Asymptotical lower bound $\Omega$

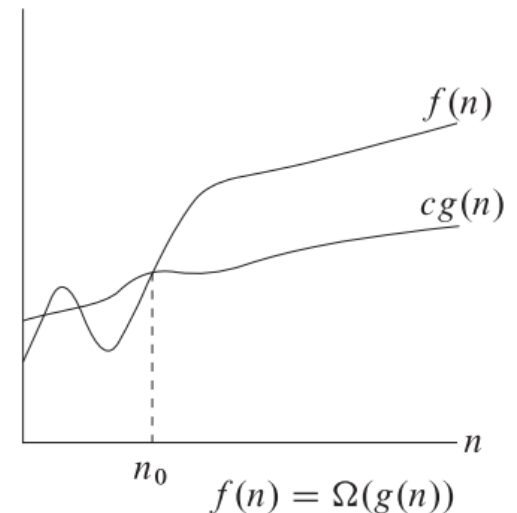
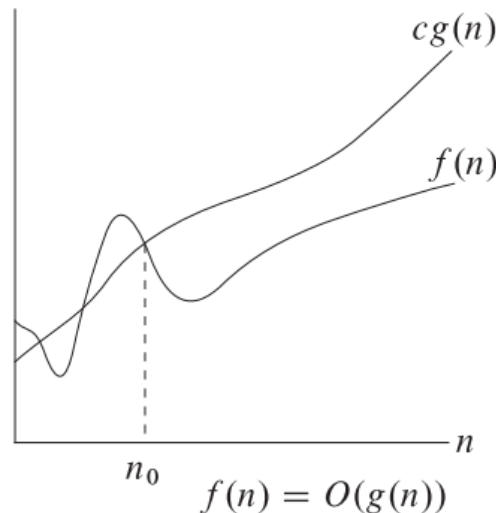
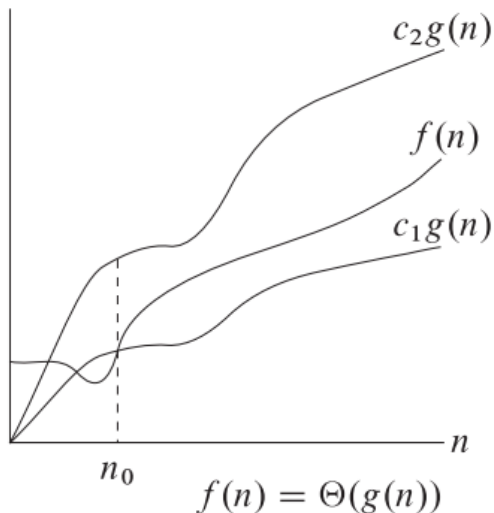
- For function  $g(n)$  we write  $\Omega(g(n))$  to be a set of functions:
- $\Omega(g(n)) = \{ f(n); \exists c, n_0 > 0, \forall n > n_0: 0 \leq cg(n) \leq f(n) \}$
- notation  $f(n) \in \Omega(g(n))$  or more frequently  $f(n) = \Omega(g(n))$
- $g(n)$  is asymptotical lower bound for  $f(n)$
- attention, the literature might be imprecise

$$\Omega(g(n))$$



# Relations between asymptotical bounds

- for functions  $f(n)$  and  $g(n)$  it holds:
- $f(n) = \Theta(g(n))$  if and only if  $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$



# Imprecise boundaries, notations $o$ and $\omega$

- $o(g(n)) = \{ f(n); \forall c > 0, \exists n_0 > 0, \forall n > n_0: 0 \leq f(n) < cg(n) \}$
  - e.g.,  $7n = o(n^2)$  in  $3n^2 \neq o(n^2)$
  - $o(g(n))$  is an imprecise upper bound
  - $f(n) = o(g(n)) \leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- 
- $\omega(g(n)) = \{ f(n); \forall c > 0, \exists n_0 > 0, \forall n > n_0: 0 \leq cg(n) < f(n) \}$
  - e.g.,  $n^2 = \omega(n)$  and  $3n \neq \omega(n)$
  - $\omega(g(n))$  is an imprecise lower bound
  - $f(n) = \omega(g(n)) \leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

# Properties of asymptotic bounds<sup>1/2</sup>

- transitivity

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

- reflexivity

$$f(n) = \Theta(f(n))$$

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

# Properties of asymptotic bounds 2/2

- symmetry

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$

- transpose symmetry

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$$

- analogy with numbers

$$f(n) = O(g(n)) \quad a \leq b$$

$$f(n) = \Omega(g(n)) \quad a \geq b$$

...

- but not trichotomy

e.g., between two numbers exactly one of the following relations holds  $a < b$ ,  $a = b$ ,  $a > b$

why not for asymptotic function bounds?

# Divide and conquer algorithms

- Idea:
  - **divide** the problem into several (equal) parts
  - (recursively) **conquer (solve)** each of the sub problems
  - **combine** sub problem solutions
- An example: maximum subarray problem

# Maximum subarray problem

// maximal subarray of array A[low...high] crossing the point mid

```
findMaxCrossingSubarray(A, low, mid, high) {  
    leftSum =  $-\infty$  ; sum = 0 ;  
    for (i = mid ; i >= low ; i--) {  
        sum = sum + A[i] ;  
        if (sum > leftSum) {  
            leftSum = sum ;  
            maxLeft = i ;  
        }  
    }  
    rightSum =  $-\infty$  ; sum = 0 ;  
    for (j = mid + 1 ; j <= high ; j++) {  
        sum = sum + A[j] ;  
        if (sum > rightSum) {  
            rightSum = sum ;  
            maxRight = j ;  
        }  
    }  
    return (maxLeft, maxRight, leftSum + rightSum) ;  
}
```

// maximal subarray of array A[low...high]

findMaxSubarray(A, low, high) {

**if** (low == high) // boundary condition

**return** (low, high, A[low]) ;

**else** {

        mid = (low + high) / 2 ;

        (leftLow, leftHigh, leftSum) = findMaxSubarray(A, low, mid) ;

        (rightLow, rightHigh, rightSum) = findMaxSubarray(A, mid+1, high) ;

        (crossLow, crossHigh, crossSum) = findMaxCrossingSubarray(A, low, mid, high) ;

**if** (leftSum >= rightSum && leftSum >= crossSum)

**return** (leftLow, leftHigh, leftSum) ;

**else if** (rightSum >= leftSum && rightSum >= crossSum)

**return** (rightLow, rightHigh, rightSum) ;

**else return** (crossLow, crossHigh, crossSum) ;

    }

}

# Kadane algorithm

- idea: for each position compute the maximum subarray result for the subarray ending at given position

```
findMaxSubarrayKadane(A) {  
    maxEndingHere = 0 ;  
    maxSoFar = 0 ;  
    for (i=1 ; i <= A.length ; i ++) {  
        maxEndingHere = max(0, maxEndingHere + A[i]) ;  
        maxSoFar = max(maxSoFar, maxEndingHere) ;  
    }  
    return maxSoFar ;  
}
```

# Four approaches to the analysis of divide-and-conquer algorithms

- substitution method:
  - guess the solution
  - using induction find the constants and prove the solution is valid (requires some practice and knowledge of some tricks)
- recursive tree:
  - draw recursion tree and sum complexity level-wise and altogether;
  - prove with induction that the result is correct
- master theorem
- Akra-Bazzi theorem

# Master theorem

- for divide and conquer algorithms
- assume constants  $a \geq 1, b > 1$ , a function  $f(n)$
- $T(n)$  is defined for nonnegative integers with recurrent equation

$$T(n) = aT(n/b) + f(n),$$

where  $n/b$  is either  $\lfloor n/b \rfloor$  or  $\lceil n/b \rceil$ .  $T(n)$  has the following asymptotic bounds

$$\left\{ \begin{array}{ll} T(n) = \theta(n^{\log_b a}) & ; f(n) = O(n^{\log_b a - \varepsilon}) \text{ for constant } \varepsilon > 0 \\ T(n) = \theta(n^{\log_b a} \log n) & ; f(n) = \theta(n^{\log_b a}) \\ T(n) = \theta(f(n)) & ; f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ for constant } \varepsilon > 0, \\ & \text{if } af\left(\frac{n}{b}\right) \leq cf(n), \text{ for constant } c < 1, \text{ and all large enough } n \end{array} \right\}$$

# Using the master theorem

- examples when it works
- and when it doesn't

# Akra-Bazzi theorem

(Mohamad Akra and Louay Bazzi, 1998)

Let

$$T(x) = \begin{cases} \theta(1) & ; 1 \leq x \leq x_0 \\ \sum_{i=1}^k a_i T(b_i x) + f(x) & ; x > x_0 \end{cases}, \text{ where}$$

- real number  $x \geq 1$ ,
- constant  $x_0 \geq 1/b_i$  and  $x_0 \geq 1/(1-b_i)$  for  $i = 1, 2, \dots, k$
- $a_i$  is a positive constant for  $i = 1, 2, \dots, k$
- $b_i$  is constant  $0 < b_i < 1$  for  $i = 1, 2, \dots, k$
- $k \geq 1$  is an integer constant
- $f(x)$  is nonnegative function satisfying polynomial growth condition: there exist positive constants  $c_1$  and  $c_2$  such that for all  $x \geq 1$  and for  $i = 1, 2, \dots, k$ , for all  $u$  for which  $b_i x \leq u \leq x$  it holds  $c_1 f(x) \leq f(u) \leq c_2 f(x)$ .  
Alternatively: if  $|f'(x)|$  is upper bounded by polynomial of  $x$ , then  $f(x)$  satisfies polynomial growth condition.
- real number  $p$  is the only solution of equation  $\sum_{i=1}^k a_i b_i^p = 1$

Then the solution of the recursion is

$$T(x) = \theta(x^p (1 + \int_1^x \frac{f(u)}{u^{p+1}} du)).$$

# Akra-Bazzi theorem – the strong form

Let

$$T(x) = \begin{cases} \theta(1) & ; 1 \leq x \leq x_0 \\ \sum_{i=1}^k a_i T(b_i x + h_i(x)) + f(x) & ; x > x_0 \end{cases}, \text{ where}$$

- real number  $x \geq 1$ ,
- constant  $x_0 \geq \max(b_i, 1/b_i)$  for  $i = 1, 2, \dots, k$
- $a_i$  is a positive constant for  $i = 1, 2, \dots, k$
- $b_i$  is constant  $0 < b_i < 1$  for  $i = 1, 2, \dots, k$
- $k \geq 1$  is an integer constant
- $|f(x)| = O(x^c)$  for any  $c \in \mathbb{N}$
- $|h_i(x)| = O(\frac{x}{\log^2 x})$
- real number  $p$  is the only solution of equation  $\sum_{i=1}^k a_i b_i^p = 1$

Then the solution of the recursion is

$$T(x) = \theta(x^p (1 + \int_1^x \frac{f(u)}{u^{p+1}} du)).$$