# Algorithms & Data Structures, Class № 5

## Introduction

### Divide and conquer algorithms

1. Divide the problem into a number of sub-problems (let the dividing time be $D(n)$).
2. Conquer the sub-problems by solving them recursively (there are $a$ subproblems to solve, each of size $n/b$, if a problem of size $n$ takes $T(n)$ time to solve, then we spend $a \cdot T(n/b)$ time solving subproblems).
3. *Base case:* If the sub-problems are small enough, just solve them by brute force ($\Theta(1)$).
4. Combine the sub-problem solutions to give a solution to the original problem ($C(n)$).

$$T(n) = \begin{cases} \Theta(1) & \text{for } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

### Recursion solving

Solving methods:

**Substitution method:** Method takes ”Guess and check“ approach for finding the solution, after the guess the ”candidate“ is substituted into recursion function and then confirmed as valid using inductive reasoning

**Iteration method:** This method bases on expansion of recursion equation to the sum of elements, in which each another is based on prior element(s)

**Master theorem:** Given constants $a \geq 1$ and $b > 1$, a non-negative $f(n)$ and $T(n)$ defined with recursion:

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

Then T(n) can be asymptotically limited:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & if \quad f(n) = O(n^{\log_b a - \epsilon}) \\ \Theta(n^{\log_b a} \log^{k+1} n) & if \quad f(n) = \Theta(n^{\log_b a} \log^k n) \\ \Theta(f(n)) & if \quad f(n) = \Omega(n^{\log_b a + \epsilon})^* \end{cases}$$

where $\epsilon > 0$ and $c$ is a certain constant $c < 1$.

*) If $af(n/b) \leq cf(n)$ for constant $0 < c < 1$ for enough big $n$ then $T(n) = \Theta(f(n))$.

## Problem 1

Running time $T(n)$ of processing $n$ data items with a given algorithm is described as follows:

a)

$$T(n) = \begin{cases} 1 & if \quad n = 1 \\ T(n-1) + 1 & if \quad n > 1 \end{cases}$$

b)

$$T(n) = \begin{cases} 1 & if \quad n = 1 \\ 2T(n/2) + n & if \quad n \geq 1 \end{cases}$$

c)

$$T(n) = \begin{cases} 0 & if \quad n = 2 \\ T(\sqrt{n}) + 1 & if \quad n > 2 \end{cases}$$

Derive closed form formulas for $T(n)$.

*Hint*: These properties of logarithms may be useful:

$$a = b^{log_b a}$$

$$log_c(ab) = log_c a + log_c b$$

$$log_b a^n = n \cdot log_b a$$

$$log_b a = \frac{log_c a}{log_c b}$$

$$log_b(1/a) = -log_b a$$

$$log_b a = \frac{1}{log_a b}$$

$$a^{log_b n} = n^{log_b a}$$

## Problem 2

Running time of processing $n$ data items with a given algorithm is described by the recurrence:

$$T(n) = k \cdot T(n/k) + c \cdot n; T(1) = 0$$

Derive a closed form formula for $T(n)$ in terms of $c$, $n$, and $k$. What is the computational complexity of this algorithm in a "Big-Oh" sense? *Hint*: To have the well-defined recurrence, assume that $n = k^m$ with the integer $m = log_k n$ and $k$.

## Problem 2

Use mathematical induction to show that when $n$ is an exact power of $2$, the solution of the recurrence

$$T(n) = \begin{cases} 2 & if \quad n = 2 \\ 2T(n/2) + n & if \quad n = 2^k, \text{ for } k > 2 \end{cases}$$

is $T(n) = n\,lg\,n$.

## Problem 3

Find $T(n)$ using the *Master theorem*:

a) $T(n) = 2T(n/4) + 1$

b) $T(n) = 2T(n/4) + \sqrt{n}$

c) $T(n) = 2T(n/4) + n$

d) $T(n) = 2T(n/4) + n^2$

e) $T(n) = 5T(n/2) + n^2$

f) $T(n) = 27T(n/3) + n^3 lg\,n$

g) $T(n) = 5T(n/2) + n^3$

*h) $T(n) = 27T(n/3) + n^3/lg\,n$