University of Ljubljana, Faculty of Computer and Information Science

Analysis of Algorithms and Heuristic Problem Solving

Prof Dr Marko Robnik-Šikonja

Ljubljana, February 2023

Lecturer

- Prof Dr Marko Robnik-Šikonja
- marko.robnik@fri.uni-lj.si
- FRI, Večna pot 113, room 2.06, 2nd floor, right from the elevator
- (01) 4798 241
- Contact hour (see webpage)
  - currently, Wednesdays, 13:00 - 14:00 or by arrangement, best to email me
- https://fri.uni-lj.si/en/employees/marko-robnik-sikonja
- Research interests:
  data science, machine learning, artificial intelligence
  natural language processing, network analytics,
  algorithms and data structures

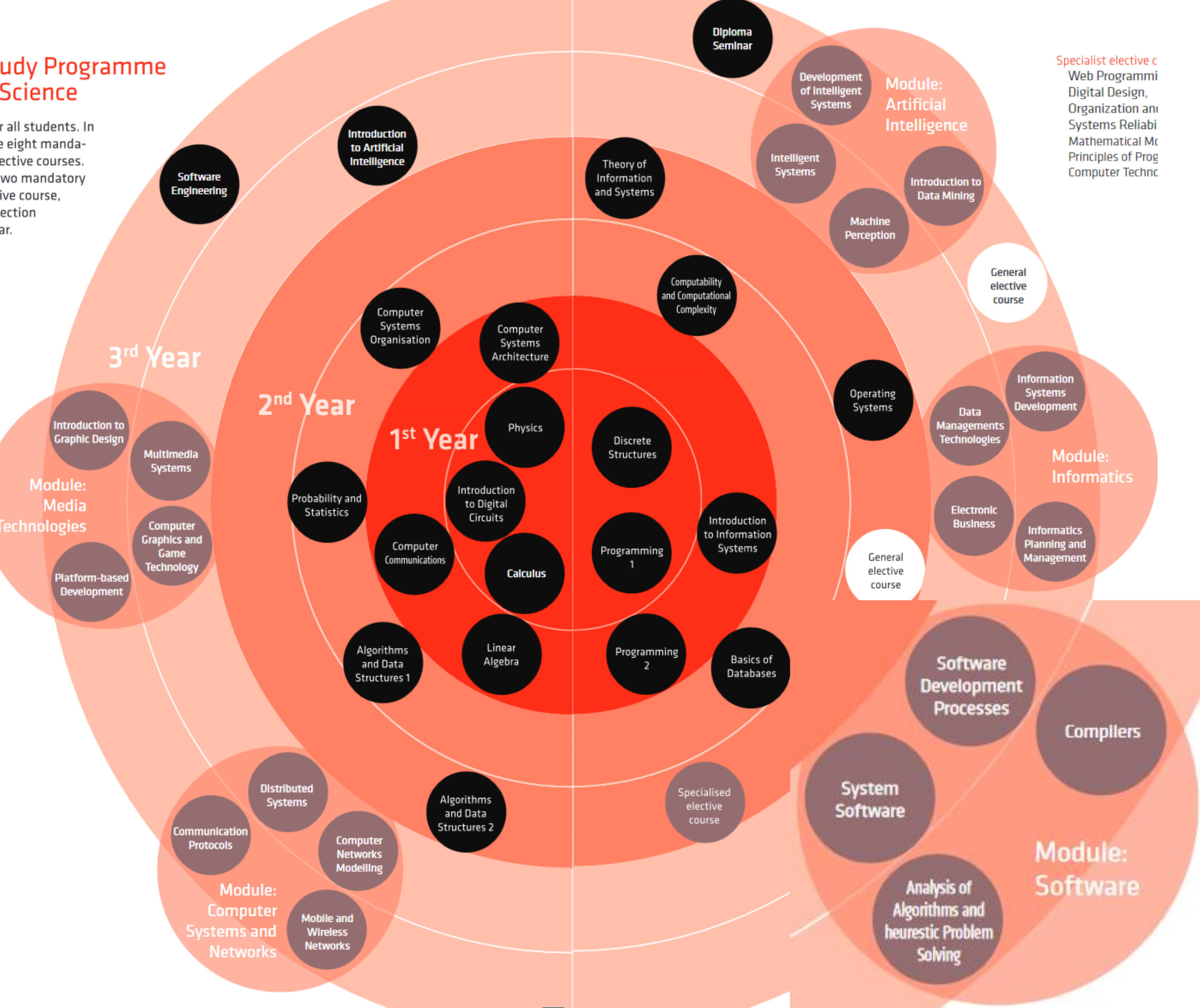
Assistant

- Dr Matej Pičulin
  matej.piculin@fri.uni-lj.si
- Laboratory for Cognitive Modeling
- tutorials mainly in the form of consultations;
  please, prepare questions!

udy Programme
Science

r all students. In
e eight manda-
ective courses.
wo mandatory
ive course,
ection
ar.

Specialist elective c
Web Programmi
Digital Design,
Organization an
Systems Reliabi
Mathematical Mc
Principles of Prog
Computer Technc



Objectives

- Students shall become acquainted with
  - the analysis of algorithms, at foremost computational complexity,
  - techniques for efficient solving of difficult problems, requiring optimization techniques and approximations.
- Practical use of theoretical knowledge on (almost) real-world problems.
- Increase the problem-solving toolbox with
  - new techniques for analysis of algorithms,
  - heuristic optimization algorithms.
- For a given optimization problem, students shall be able to
  - select one of the appropriate methods,
  - construct a solution prototype.

"Thinking outside of the box is difficult for some people. Keep trying."

Lectures and tutorials

- Lectures:
  - introduction to the topic, discussion,
  - some examples,
  - broader view of the topic.
- Tutorials:
  - exercises,
  - assignments motivated by practical use,
  - assistant presents the assignments, helps with tips, moderates discussion so…
  - … come prepared and pose questions.
  - Introduce some problem solving tools and useful software.

Syllabus

- 1st part:
  - computational complexity,
  - analysis of algorithms,
  - some problems turn out to be too difficult for solving exactly, so we need approximation methods and heuristic approaches,
- 2nd part:
  - heuristic programming,
  - introduction to some heuristic approaches using
    - operation research approaches,
    - population techniques
    - metaheuristics
  - how to approach real-world problems.

More details

Lecture topics:

1. Analysis of recursive algorithms: recursive tree method, substitution method, solution for divide and conquer approach, Akra-Bazzi method.
2. Probabilistic analysis: definition, analysis of stochastic algorithms.
3. Randomization of algorithms.
4. Amortized analysis of algorithm complexity.
5. Solving linear recurrences.
6. Analysis of multithreaded, parallel and distributed algorithms.
7. Linear programming for problem solving.
8. Combinatorial optimization, local search, simulated annealing.
9. Metaheuristics and stochastic search: guided local search, variable neighbourhood search, and tabu search.
10. Memetic algorithms, particle swarm optimization, grey wolf, whales, bees, etc.
11. Differential evolution.
12. Machine learning for combinatorial optimization.
13. Many (almost) practical problems; interspersed within other topics

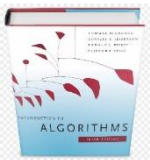
Obligations

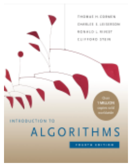
- 5 quizzes checking continuous work; obtaining at least 50% of points altogether is necessary,
- 5 assignments of different difficulty, practical and theoretical assignments, written reports, one assignment is in the form of competition and public presentation,
- written exam.

Learning materials

- learning materials in the eClassroom
  http://ucilnica.fri.uni-lj.si
- practical work in open-source system R,
- optionally in Python, java, C/C++

Readings





- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein: *Introduction to Algorithms, 3rd or 4th edition*. MIT Press, 2009, 2022
- M. Gendreau, J-Y. Potvin (Eds.): *Handbook of Metaheuristics*, 2nd edition. Springer 2010

Further readings:

- R. Sedgewick, P. Flajolet: *An Introduction to the Analysis of Algorithms.* Addison-Wesley, 1995
- scientific papers, some on eClassroom

Review of existing knowledge on computational complexity

Find the computational complexity

```
s=0 ;
for (i=1; i <= n ; i++)
  s=s+a[i];
```

```
s=0 ;
for (i=1; i <= n ; i++)
  for (j=1; j <= n ; j++)
    s = s+t[i][j];
```

```
s=0 ;
for (i = 1; i <= n ; i += m)
  s = s + a[i];
```

```
for (i=1; i <= n ; i++)
   for (j=1 ; j <= n ; j++)
      for (k=1 ; k <= n ; k++)
         if  (i + j + k < a)
            G[i][j] = A[i][j]+B[i][k]*C[k][j];
```

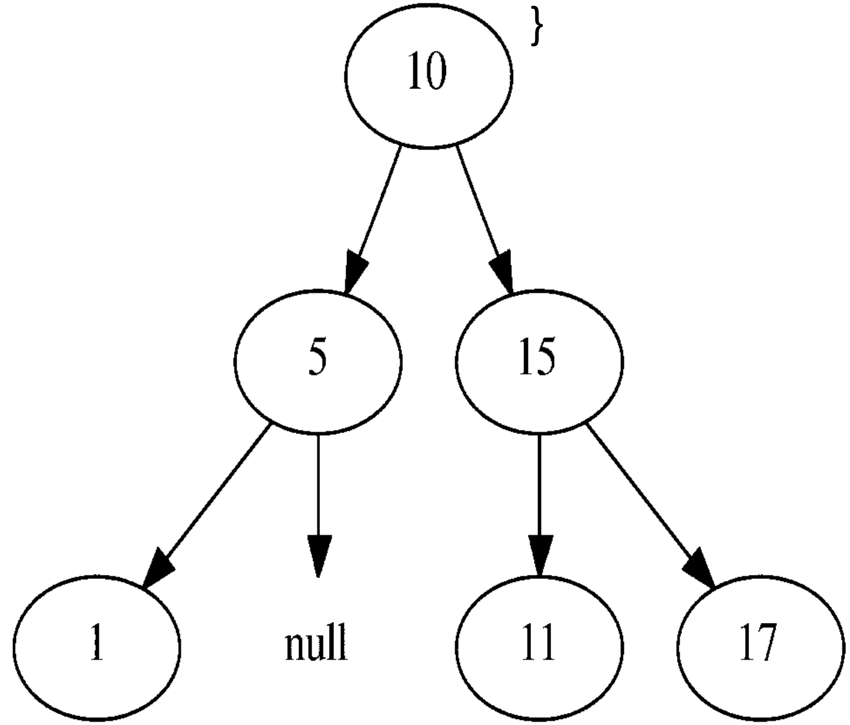
```
for (i=1; i <= n ; i++)
   if  (i < a)
      for (j=1 ; j <= n ; j++)
        for (k=1 ; k <= n ; k++)
            G[i][j] = A[i][j]+B[i][k]*C[k][j];
```

```
int i = n ;
int r =0 ;
while (i >1) {
    r = r+1 ;
    i = i / 2 ;
}
```

```
public static void loopRek(int m, int n)
{
      if (n == 1)
       System.out.println("+") ;
     else
       for (int i=0; i < m ; i++)
          loopRek(m, n-1) ;
}
```

```
struct Node {
    int key ;
    Node left, right ;
}
```

```
public static void infix(Node p)
{
   if (p != null)  {
       infix(p.left) ;
       System.out.print(p.key) ;
       infix(p.right) ;
   }
}
```



first determine the parameter of complexity

```
max = a[1] ;
for (i=2 ; i <= n ; i++)
  if (max < a[i])
    max = a[i] ;
System.out.print(max) ;
```

```
max = a[1] ;
for (i=2 ; i <= n ; i++)
  if (max < a[i]) {
    max = a[i] ;
    veryComplexOperation(max)
  }
System.out.print(max) ;
```

```
void p(int n, int m) {
  int i,j,k ;
  if (n > 0) {
    for (i=0 ; i < m ; i++)
      for (j=0 ; j < m ; j++)
        if (i < j – a)
          for (k=0 ; k < m ; k++)
            System.out.println(i + j * k) ;
    p(n/m, m) ;
  }
}
```

Analysis of algorithms

- How complex is the algorithm?
- How many resources it requires?
- How much time, memory, etc. will the computer need?

- Resources: time, memory, network accesses, other hardware

A simple model of computer - RAM

- RAM – abstract uniprocessor machine with random access to the memory (RAM –Random-Access Machine)
- operations and their price (execution time, memory, etc.):
- typical operations: arithmetical and logical operations, memory operations, control
- each operation uses a constant amount of time
- integers and floating-point numbers
- numbers use a limited amount of memory; for example number n takes at most c $\log_2(n)$ bits, where constant c >= 1 (what if it is not constant)
- we assume constant time for some other operations as well, e.g., logarithms, exponents, trigonometrical operations
- we do not consider parallelism, pipelines, memory hierarchies
- RAM is (good enough) approximation for real world computers

Input size

- define for each problem separately
  - size of an input, e.g., array
  - number of bits in input
  - size of graph (nodes, edges)
  - number of steps taken,
  - etc.

Execution time

- number of steps of the abstract machine
- for simpler analysis, we assume that each line of pseudocode requires a constant time (except function calls),
- so line i requires c_i time

An example: insertion sort

- execution time depends on input (number of elements, their initial positions)
- time: number of steps of abstract machine
- for the sake of simplicity, we assume a constant execution time for each line of pseudo-code, i.e., line i takes c_i time, where c_i is constant larger or equal zero
- idea: iteratively increase the sorted part of an array, by inserting unsorted elements into the already sorted part

Pseudocode

```
InsertionSort(A) {
  for j = 2 to A.length
    key = A[j] ;
     // insert A[j] into sorted array A[1..j-1]
    i = j-1 ;
    while i > 0 and A[i] > key
          A[i+1] = A[i] ;
          i = i -1 ;
    A[i+1] = key ;
```

Count the operations

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$\quad key = A[j]$	c_2	$n - 1$
$\quad$ // Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n - 1$
$\quad i = j - 1$	c_4	$n - 1$
$\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$\quad\quad A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$\quad\quad i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$\quad A[i+1] = key$	c_8	$n - 1$

Sum together

$$
\begin{aligned}
T(n) \;=\; & c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
& + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \; .
\end{aligned}
$$

- number of operations depends on the input

Best case

- the best case is when the array is already sorted, then $t_j = 1$, for $j = 2,3,...,n$ and we get a linear dependency on n

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) .
\end{aligned}
$$

Worst case

- worst case occurs when the array is sorted in reversed order, then $t_j = j$, for j=2,3, ..., n and we get

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \ . \end{aligned}$$

which can be expressed as a quadratic dependency

$T(n) = an^2 + bn + c$

Analysis

- we mostly analyze worst and average case complexities; why?
- we are rarely interested in actual constant and settle for the order of growth,
- in this case only the fastest growing terms are important, others are asymptotically unimportant,
- the worst case for the insertion sort is $\Theta(n^2)$

Differences between the orders of complexity

		n				
		10				
		100				
		1.000				
		10.000				
		100.000				
		1.000.000				

Differences between the orders of complexity

	√n	n				
	3	10				
	10	100				
	31	1.000				
	100	10.000				
	316	100.000				
	1.000	1.000.000				

Differences between the orders of complexity

$log_{10}(n)$	**√n**	**n**				
1	3	10				
2	10	100				
3	31	1.000				
4	100	10.000				
5	316	100.000				
6	1.000	1.000.000				

Differences between the orders of complexity

$\log_{10}(n)$	**√n**	**n**	**$n \cdot \log_{10}(n)$**			
1	3	10	10			
2	10	100	200			
3	31	1.000	3.000			
4	100	10.000	40.000			
5	316	100.000	500.000			
6	1.000	1.000.000	6.000.000			

Differences between the orders of complexity

$\log_{10}(n)$	$\sqrt{n}$	n	$n \cdot \log_{10}(n)$	n^2		
1	3	10	10	100		
2	10	100	200	10.000		
3	31	1.000	3.000	1.000.000		
4	100	10.000	40.000	10^8		
5	316	100.000	500.000	10^{10}		
6	1.000	1.000.000	6.000.000	10^{12}		

Differences between the orders of complexity

$\log_{10}(n)$	$\sqrt{n}$	n	$n \cdot \log_{10}(n)$	n^2	n^3	
1	3	10	10	100	1000	
2	10	100	200	10.000	1.000.000	
3	31	1.000	3.000	1.000.000	10^9	
4	100	10.000	40.000	10^8	10^{12}	
5	316	100.000	500.000	10^{10}	10^{15}	
6	1.000	1.000.000	6.000.000	10^{12}	10^{18}	

Differences between the orders of complexity

$\log_{10}(n)$	$\sqrt{n}$	n	$n \cdot \log_{10}(n)$	n^2	n^3	2^n
1	3	10	10	100	1000	1024
2	10	100	200	10.000	1.000.000	$1.25 \cdot 10^{30}$
3	31	1.000	3.000	1.000.000	10^9	10^{301}
4	100	10.000	40.000	10^8	10^{12}	$2 \cdot 10^{3.010}$
5	316	100.000	500.000	10^{10}	10^{15}	$10^{30.103}$
6	1.000	1.000.000	6.000.000	10^{12}	10^{18}	$10^{301.030}$