
Divide-and-Conquer

Practical Exercises

Departamento de Engenharia Informática (DEI)
Faculdade de Engenharia da Universidade do Porto (FEUP)

Spring 2025

Exercise 1

Recall the **maximum sum subarray problem**:

Given any one-dimensional array $A[1..n]$ of integers, the **maximum sum subarray problem** tries to find a contiguous subarray of A , starting with element i and ending with element j , with the largest sum: $\max_{x=i}^j A[x]$ with $1 \leq i \leq j \leq n$. The return is the sum of the maximum subarray, for which i and j are the indices of the first and last elements of this subsequence (respectively).

Input example: $A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]$

Expected result: $[0, 0, 0, 1, 1, 1, 1, 0, 0]$, as subsequence $[4, -1, 2, 1]$ ($i = 3, j = 6$) produces the largest sum, 6.

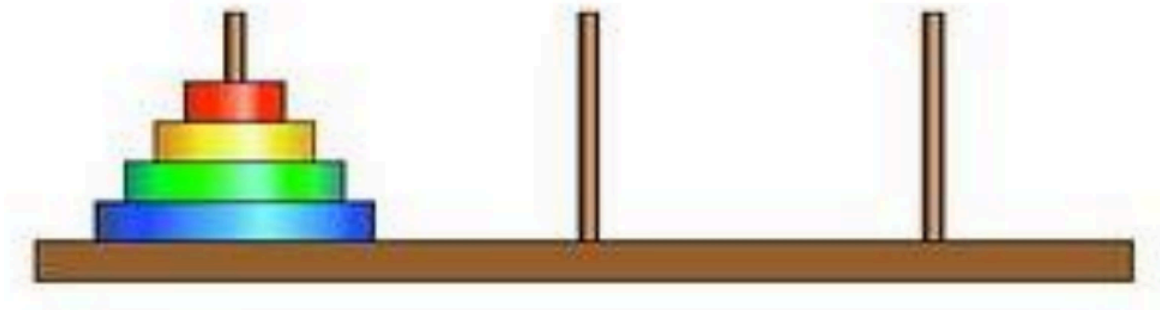
- Propose in pseudo-code a divide-and-conquer strategy for this problem.
- Using the master theorem, indicate and justify the time complexity of the proposed algorithm.
- Implement *maxSubsequenceDC* using the proposed algorithm.

```
int maxSubsequenceDC(int A[], unsigned int n , int &i, int &j)
```

Exercise 2

In the **Hanoi towers problem**, the goal is to move a stack of n disks of decreasing size from one peg to another, with the following constraints:

- Three pegs are available: A, B and C. The stack of n disks begins at one of the pegs.
- Only one disk can be moved at a time.
- At any time, the disk stack of any peg must be ordered in decreasing size order, with the largest disk at the bottom and the smallest one at the top.



- Propose an algorithm in pseudo-code that solves this problem using a divide-and-conquer strategy. The solution must minimize the number of disk movements.
- Using induction, prove that for n pegs, at most $(2^n - 1)$ moves are needed. Using this result, indicate and justify the algorithm's time complexity, with respect to the number of disks, n .
- Implement the *hanoiDC*, which implements the proposed algorithm.

```
std::string hanoiDC(unsigned int n, char src, char dest)
```

Input example: $n = 4$, $\text{src} = \text{'A'}$, $\text{dest} = \text{'B'}$

Expected result:

"A→C,A→B,C→B,A→C,B→A,B→C,A→C,A→B,C→B,C→A,B→A,C→B,A→C,A→B,C→B"

Exercise 3

Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose if your problem instances exhibited large values of n ?