Date: 2025-10-24
Version: 2025-10-24 01:46

Exercise Sheet 2 for Effiziente Algorithmen (Winter 2025/26)

Hand In: *Until 2025-10-31 18:00, on ILIAS.*

Problem 1

10 + 10 points

Prove the following statements, each for $n \to \infty$.

a) Let $P(n) = a_0 + a_1 n + \cdots + a_k n^k$ be any polynomial with $a_k > 0$.

Then $P(n) \sim a_k n^k$ holds.

b) For all $\alpha, \varepsilon \in \mathbb{R}_{>0}$ we have $n^\alpha \in o(n^{\alpha+\varepsilon})$.

Problem 2

10 + 10 + 20 + 20 points

Prove or disprove the following claims, each for $n \to \infty$.

a) $2^{2n} \sim 4^n$.

b) $\sqrt[n]{n} = o(1)$.

c) For arbitrary functions $f, g : \mathbb{N} \to \mathbb{R}$ it holds that $f(n) = O(g(n))$ or $g(n) = O(f(n))$.

d) $(n+m)^4 \in \Theta(n^4)$ for $m \in \mathcal{O}(1)$.

Problem 3

10 + 20 + 20 + 10 points

In the lecture, several solutions for the Maximum Subarray Problem were presented. In this exercise, you are to develop a solution for the 2-dimensional variant of the problem. The definition of the 2D Maximum Subarray Problem is as follows:

Given an integer $n \times n$ matrix M, where $M[i,j]$ with $0 \leq i,j \leq n$ gives the value of the matrix in the i-th row and j-th column. Let $z_k(x_1, x_p) = \sum_{i=x_1}^{x_p} M[i,k]$ be the sum of the rows x_1 through x_p of the k-th column of the matrix. Furthermore, let $S(x_1, x_p, y_1, y_q) = \sum_{i=y_1}^{y_q} z_i(x_1, x_p)$ be the sum of the submatrix of M consisting of columns y_1 through y_q, each with rows x_1 through x_p. Determine the coordinates of the maximal submatrix $S_{\max}$ in M, i.e.

$$S_{\max} = \left\{(x_1, x_p, y_1, y_q) \;\middle|\; \forall (x_1', x_p', y_1', y_q') : S(x_1, x_p, y_1, y_q) \geq S(x_1', x_p', y_1', y_q')\right\}.$$

You can find the code base for this exercise on the course website. In addition, the code base is also embedded in this PDF (the latter may not work with some PDF viewers):

SubArrayProblem.java

Note: $S_{\max}$ returns a set; in the code and in this exercise it suffices to return one possible solution.

a) The `bruteForce` method of the `SubArrayProblem` class returns a (very naive) solution for the 2D Maximum Subarray Problem. Provide the smallest possible upper bound in Big-O notation for the `bruteForce` algorithm. Justify your answer.

b) Describe in text an algorithm that solves the problem in time $\mathcal{O}(n^3)$.

   Hint: You may use the $\Theta(n)$ solution for the 1-dimensional problem from the lecture. Justify the running time of your solution.

c) Implement your solution from (b) in a method `efficient` in the class `SubArrayProblem`.

d) In the class `SubArrayProblem`, create a suitable experiment to empirically determine the running times of `bruteForce` and `efficient`. Plot a comparison of the results and describe them.

Problem 4

30 points

We consider a stack S that supports the following operations:

1. PUSH(S, x): Pushes element x onto the stack S.
2. POP(S): Removes the top element from the stack S.
3. MULTIPOP(S, k): Removes the top k elements from the stack S.

`Push` and `Pop` have running time $O(1)$, i.e., a sequence of n `Push` or `Pop` operations has running time $\Theta(n)$. `MultiPop` removes the (at most) top k elements of the stack S of size s in a sequence of `Pop` operations and therefore has cost $\min(s, k)$.

We now consider a sequence of `Push`, `Pop`, and `MultiPop` operations on an initially empty stack. Since the stack has height at most n, `MultiPop` has running time $O(n)$ in the worst case. Accordingly, a sequence of n of these three operations has running time at most $O(n^2)$. However, this upper bound is not very close to the actual cost.

Determine the amortized cost using the potential method: Determine a suitable potential Φ that assigns to the stack D_i, which results after the i-th stack operation, a non-negative number $\Phi(D_i)$. Using this, compute the amortized cost of the three stack operations and infer a tighter upper bound for the cost of n operations than the original $O(n^2)$.