# Algorithms SV worksheet 6

Bálint Molnár

September 28, 2025

1. Consider a variant of a double-ended queue (deque) that supports the following operations:

   - `removeFirst`$(k)$ removes the first $k$ elements from the front of the deque (or clear it if there are less then $k$ elements).
   - `add`$(r)$ removes all elements in the deque that are greater than $r$, then inserts $r$ at the end.

   Design an efficient data structure that supports these operations and analyze their amortized cost.

2. Consider an undirected graph with $n$ vertices, where the edges can be coloured blue or white, and which starts with no edges. The graph can be modified using these operations:

   - `insert_white_edge`$(u, v)$ inserts a white edge between vertices $u$ and $v$.
   - `colour_edges_of`$(v)$ colours blue all the white edges that touch $v$.
   - `colour_edge`$(u, v)$ colours the edge $u \leftrightarrow v$ blue.
   - `is_blue`$(u, v)$ returns `True` if and only if the edge $u \leftrightarrow v$ is blue.

   Give an efficient algorithm that supports these operations, and analyse its amortized cost.

   Extend your algorithm to also support the following operation, and analyse its amortized cost:

   - `are_blue_connected`$(u, v)$ returns `True` if and only if $u$ and $v$ are connected by a blue path.

   Extend your algorithm to also support the following operation, and analyse its amortized cost:

   - `remove_blue_from_component`$(v)$ deletes all blue edges between pairs of nodes in the blue-connected component containing $v$.

   **Note.** It's easy to gloss over difficulties, so be sure to be explicit about all operations. If you change your data structure to answer a later part, make sure your earlier answers are still complete. This is the sort of question you might be asked in a Google interview; the interviewer will be looking for you to take ideas that you have been taught and to apply them to novel situations.

   This question is from Alstrup and Rauhe, via Inge Li Gørt.

3. (a) Write a simple pseudocode for polynomial addition and multiplication. What is their time complexity?

   (b) Prove the following equality for polynomial multiplication:
   Given two polynomials:

   $$A(x) = A_0(x) + A_1(x)x^n$$
   $$B(x) = B_0(x) + B_1(x)x^n$$

   such that $A_0, A_1, B_0, B_1$ are all polynomials with degree $\leq n$ Show that their product can be computed as:

   $$A(x)B(x) = A_1(x)B_1(x)x^{2n} + ((A_1(x)+A_0(x))(B_1(x)+B_0(x)) - A_0(x)B_0(x) - A_1(x)B_1(x))x^n + A_0(x)B_0(x)$$

(c) Use the previous equality to design a divide-and-conquer method for polynomial multiplication that runs in $O(n^{\log_2 3})$ time (where $n$ is the degree of the polynomials).

(d) How would you use this method for multiplying large ($n$-digit) numbers?

4. The Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem. Given a set ($V$) of $n$ cities, a "source" city $s \in V$ and a distance function $d(i, j)$ that defines the cost of travelling between any two cities $i$ and $j$, the goal is to find the shortest possible route that visits each city exactly once starting from city $s$.

   (a) Write a brute-force method to solve the problem. What is the time complexity of your solution?

   (b) Write dynamic programming equations for $\mathrm{DP}(S, v)$ for each $S \subseteq V$ and $v \in V$, where $\mathrm{DP}(S, v)$ is the minimum cost to visit every city in $S$ starting from $v$. What is the time complexity of the resulting DP algorithm?

   (c) Show that the time complexity of the Travelling Salesman problem is $\Omega(n \log n)$.

   *Hint: How many branching clauses (e.g. if-else, switch statements etc.) do you need to enter? The argument is similar to the sorting lower bound.*

5. In a country, there are direct train routes between railway stations, each with a given travel time (The train network is a weighted undirected graph). We define the distance between two nodes as the length of the shortest path between them.

   Train tickets in this country specify only the departure and destination stations. Travelers can take multiple routes between these stations, but with one restriction: at no point during the journey can they travel farther from the destination than they were when they boarded the train (i.e. you cannot take a train from city $C$ to $D$ if $D$ is further away from your destination than $C$. You are allowed to take a train multiple times as long as this constraint is not violated).

   You have a ticket between cities $A$ and $B$. The country has beautiful railway stations, and you want to visit as many of them as possible.

   Design an algorithm that calculates the maximum number of stations you can visit using a ticket between cities A and B.