Cvičení z Algoritmů 3, 20. 10.

Vyřešte následující úkoly. Jejich řešení věnujte alespoň 90 minut (nebo méně, pokud se Vám je povede vyřešit dříve). Správná řešení zveřejním a okomentuji za týden. Není nutné mi nic posílat. V případě potřeby mě můžete kontaktovat mailem a můžeme si dohodnout konzultaci na Zoomu.

Úkol 1.
Na papír simulujte algoritmus pro rychlé násobení polynomů pro výpočet součinu $1 + 10x + x^2$ s $6 + x$.

Úkol 2.
Vymyslete algoritmus se složitostí $O(n \log n)$, který v posloupnosti různých přirozených čísel (uložené v poli)

$$a_1, a_2 \dots, a_n$$

spočítá počet dvojic (a_i, a_j), pro které $a_i > a_j$ ale $i < j$. (Tedy spočítá počet inverzí v poli). Nápověda: *MergeSort*.

Řešení

Úkol 1.
Zjistíme, že potřebujeme počítat FFT se 4 koeficienty.

$\underline{\text{FFT}([1, 10, 1, 0], n)}$
Máme

$$\begin{aligned} A_S &\leftarrow [1, 1], \\ A_L &\leftarrow [10, 0] \end{aligned}$$

Pomocí rekurzivních zavolání (viz níže) dostaneme

$$\begin{aligned} S &\leftarrow [2, 0] \\ L &\leftarrow [10, 10]. \end{aligned}$$

Pro $k = 0, k + n/2 = 2$ máme

$$\begin{aligned} x &\leftarrow 1 \\ R[0] &\leftarrow 2 + 1 \cdot 10 = 12 \\ R[2] &\leftarrow 2 - 1 \cdot 10 = -8 \end{aligned}$$

Pro $k = 1, k + n/2 = 3$ máme

$$\begin{aligned} x &\leftarrow i \\ R[1] &\leftarrow 0 + i \cdot 10 = 10i \\ R[3] &\leftarrow 0 - i \cdot 10 = -10i \end{aligned}$$

Celkově tedy $\text{FFT}([1, 10, 1, 0], n)$ vrací $[12, 10i, -8, -10i]$. Pro rekurzivní zavolání $\text{FFT}([1, 1], 2)$

máme

$$\begin{aligned} S &\leftarrow [1] \\ L &\leftarrow [1]. \end{aligned}$$

a tedy pro $k = 0, k + n/2 = 1$ máme

$$\begin{aligned} x &\leftarrow 1 \\ R[0] &\leftarrow 1 + 1 \cdot 1 = 2 \\ R[1] &\leftarrow 1 - 1 \cdot 1 = 0 \end{aligned}$$

Pro druhé rekurzivní zavolání, tedy $\textsc{fft}([10, 0], 2)$ máme

$$\begin{aligned} S &\leftarrow [10] \\ L &\leftarrow [0]. \end{aligned}$$

a tedy pro $k = 0, k + n/2 = 1$ máme

$$\begin{aligned} x &\leftarrow 1 \\ R[0] &\leftarrow 10 + 1 \cdot 0 = 10 \\ R[1] &\leftarrow 10 - 1 \cdot 0 = 10 \end{aligned}$$

$\underline{\textsc{fft}([6, 1, 0, 0], n)}$

$$\begin{aligned} A_S &\leftarrow [6, 0], \\ A_L &\leftarrow [1, 0] \end{aligned}$$

Pomocí rekurzivních zavolání (která už nevypisujeme, jsou analogická $\textsc{fft}([10, 0], 2)$ výše) dostaneme

$$\begin{aligned} L &\leftarrow [6, 6] \\ S &\leftarrow [1, 1]. \end{aligned}$$

Pro $k = 0, k + n/2 = 2$ máme

$$\begin{aligned} x &\leftarrow 1 \\ R[0] &\leftarrow 6 + 1 \cdot 1 = 7 \\ R[2] &\leftarrow 6 - 1 \cdot 1 = 5 \end{aligned}$$

Pro $k = 1, k + n/2 = 3$ máme

$$\begin{aligned} x &\leftarrow i \\ R[1] &\leftarrow 6 + i \cdot 1 = 6 + i \\ R[3] &\leftarrow 6 - i \cdot 1 = 6 - i \end{aligned}$$

Celkově tedy $\textsc{fft}([6, 1, 0, 0], n)$ vrací $[7, 6 + i, 5, 6 - i]$.

Součin (po souřadnicích) $[12, 10i, -8, -10i] \cdot [7, 6 + i, 5, 6 - i]$ je $[84, -10 + 60i, -40, -10 - 60i]$.

$\underline{\textsc{fft-reverse}([84, -10 + 60i, -40, -10 - 60i], 4)}$

Z rekurzivních zavolání (která už nerozepisujeme, už je to rutina) dostaneme

$$\begin{aligned} S &\leftarrow [44, 124] \\ L &\leftarrow [-20, 120i] \end{aligned}$$

Pro $k = 0, k + n/2 = 2$ máme

$$\begin{aligned} x &\leftarrow 1 \\ R[0] &\leftarrow 44 + 1 \cdot (-20) = 24 \\ R[2] &\leftarrow 44 - 1 \cdot (-20) = 64 \end{aligned}$$

Pro $k = 1, k + n/2 = 3$ máme

$$\begin{aligned} x &\leftarrow -i \text{ (tady vidime rozdil oproti FFT, tam bychom měli } i) \\ R[1] &\leftarrow 124 + (-i) \cdot 120i = 24 \\ R[3] &\leftarrow 124 - (-i) \cdot 120i = 4 \end{aligned}$$

Výsledek tedy je $[24, 244, 64, 4]$, který když podělíme po složkách 4 dostaneme $[6, 61, 16, 1]$, a to už jsou koeficienty výsledného polynomu.

Úkol 2.

(*Je dobrý nápad připomenout si před čtením řešení úkolu algoritmus MergeSort*)

Nápad: rozdělíme pole $a_1, a_2, \ldots, a_n$ na poloviny, které označíme je L (levá) a P (pravá). Potom je počet inverzí celkem roven počtu inverzí v L + počtu inverzí v P + počtu inverzí, ve kterých je prvek v P menší než nějaký prvek v L. Abychom mohli počítat inverze tohoto druhu, bylo by výhodné mít L a P uspořádány vzestupně. Pak bychom totiž mohli snadno počítat, kolik prvků v L je větších než daný konkrétní prvek v P a nemuset přitom procházet celé L (stačilo by najít nejlevější prvek v L, který je větší). Potřeba mít prvky v L a P uspořádány a nutnost procházet P a L a porovnávat jejich prvky nás dovede k třídícímu algoritmu Mergesort, kde se toto děje během spojování již setříděných polovin pole (tj. v proceduře Merge). Stačí tuto proceduru trochu upravit, aby počítala i počty inverzí, a dostaneme tak algoritmus pro náš problém (rekurzivní volání v Mergesortu i upravená procedura Merge tak budou vracet počty nalezených inverzí, které jenom sečteme a dostaneme výsledek). Jak upravit proceduru Merge? V momentě, kdy porovnáváme prvek l z pole L s prvkem p z pole P a zjistíme, že $p < l$, pak počet nalezených inverzí zvedneme o počet prvků, které zbývají ke slití v poli L.

Z předchozího popisu jistě zvládnete algoritmus rekonstruovat podrobněji. Podrobnější popis lze také nalézt v knize *Kleinberg, Tardes: Algorithm Design.* na straně 221.