Za domácí úkol máte příklady 5 a 6. Důkladně si přečtěte zadání!

Věta 1 (Linearita střední hodnoty)**.** Nechť X a Y jsou náhodné veličiny a α a β jsou reálná čísla. Potom $E[\alpha X + \beta Y] = \alpha E[X] + \beta E[Y]$.

Lemma 1 (O džbánu)**.** Čekáme-li na náhodný jev, který nastane s pravděpodobností p, dočkáme se ve střední hodnotě po $1/p$ nezávislých pokusech.

Příklad 1 (Quickselect)**.** Uvažme následující randomizovaný algoritmus pro hledání k-tého nejmenšího prvku. Náhodně vyberme pivota a nechť l je počet prvků menších než pivot. Jestliže $k \leq l$, pak rekurzivně hledejme k-tý prvek mezi prvky menší než pivot, jinak hledejme $(k-l)$-tý prvek mezi prvky, které jsou větší nebo rovny pivotu. Jaká je složitost tohoto algoritmu v nejhorším případě? Jaká je očekávaná složitost?

Příklad 2. Quicksort může (a prakticky se občas i stává) potřebovat lineární prostor pro zásobník, což může způsobit i pád programu. Jak můžeme upravit quicksort, aby mu vždy stačil logaritmicky velký zásobník?

Příklad 3. Quicksort má očekávanou složitost $\mathcal{O}(n \log n)$, ale v nejhorším případě může výpočet trvat $\mathcal{O}(n^2)$. Jak jednoduše upravit quicksort, aby jeho složitost byla $\mathcal{O}(n \log n)$ i v nejhorším případě, ale nedošlo k výraznému zpomalení v průměrném případě?

Příklad 4. Jak rychle umíte setřídit n čísel z rozsahu $0, \ldots, n^3 - 1$.

Příklad 5. Jaká je průměrná časová složitost bublinkového třídění? Přesněji: pokud dostaneme na vstupu náhodnou permutaci čísel 1 až n, jaká bude střední hodnota doby výpočtu? Předpokládáme, že algoritmus se zastaví, jakmile během jednoho průchodu nic neprohodí.

Příklad 6. Určete časovou složitost v nejhorším případě následujících variant třídících algoritmů založených na hledání souvislých rostoucích podposloupností a jejich slévání.

1. Nejprve jedním průchodem rozdělme vstupní posloupnost na maximální souvislé rostoucí podposloupnosti a poté opakujme následující postup. Mějme k rostoucích podposloupností a pro všechna $i = 1, \ldots, \lfloor k/2 \rfloor$ slijme $(2i-1)$-tou a $(2i)$-tou podposloupnost. Pokud i je liché, tak poslední podposloupnost nezměníme.
2. Nejprve najdeme první maximální rostoucí podposloupnost, tj. maximální k takové, že prvních k prvků je v rostoucím pořadí. Poté opakovaně hledáme další maximální rostoucí podposloupnost a sléváme s první.
3. Rostoucí podposloupnosti (resp. indexy hrajních prvků) bude postupně dávat do zásobníku. V každé fázi algoritmu dáme na vrchol zásobníku ze vstupní posloupnosti další maximální souvislou rostoucí podposloupnost a aplikujeme následující pravidla, dokud je možné některé aplikovat. Nechť k je vždy aktuální počet podposloupností na zásobníku, jejichž délka je značena $l_1, \ldots, l_k$, kde l_1 je délka podposloupnosti na vrcholu zásobníku.
   - Jestliže $k \geq 2$ a $l_2 \leq l_1$, tak slijeme podposloupnosti 1 a 2.
   - Jestliže $k \geq 3$ a $l_3 \leq l_1 + l_2$, tak slijeme podposloupnost 2 s kratší z 1 a 3.

   Všimněme si, že vždy sléváme dvě sousední podposloupnosti a slitou vždy vracíme na tu pozici zásobníku, kde se nacházeli původní dvě podposloupnosti a jedno uvolněné místo vynecháme. Poté, co projdeme celou vstupní posloupnost, tak ještě dotřídíme podposloupnosti na zásobníku postupným sléváním dvou podposloupností na vrcholu zásobníku.