

1. Napište bublinkové třídění jako program pro RAM, popište formát vstupu a výstupu a vyjádřete přesně (počtem instrukcí) časovou a paměťovou složitost programu v nejlepším i nejhorším případě.

Formát vstupu/výstupu: V buňce 0 délka vstupu (n), v dalších buňkách jednotlivá čísla. Po skončení programu budou na stejném místě čísla setříděná.

Program:

```

B := 1      Index vnějšího cyklu
C := B + 1  Index vnitřního cyklu
             Pokud je pořadí správné, neprohazujeme
cyklus: if [B] <= [C] then goto neproh
         D := [B]      Jinak prohodíme čísla na idx B a C
         [B] := [C]
         [C] := D
neproh:  C := C + 1    Další krok vnitřního cyklu
         if C <= [0] then goto cyklus
         C := 0        Další krok vnitřního cyklu
         B := B + 1
         if B < [0] then goto cyklus
halt     Konec programu, vnější cyklus prošel všechny prvky

```

Časová složitost v nejhorším případě: $3 + (n - 1) \cdot (3 + \frac{n}{2} \cdot 6)$ (3 instrukce na začátku/konci, $n - 1$ iterací vnějšího cyklu, 3 instrukce na jeho režii, $n - B$ iterací vnitřního cyklu, 6 instrukcí na cyklus, když vždy prohazujeme)

Časová složitost v nejlepším případě: $3 + (n - 1) \cdot (3 + \frac{n}{2} \cdot 3)$ (vstup je už setříděný a neprohazujeme)

Prostorová složitost ve všech případech: $n + 4$

2. Vyberte si nějaký strojový/assembly jazyk reálného počítače a porovnejte ho s jazykem instrukcí RAM.

V obecnosti se RAM od strojových jazyků reálných počítačů liší především

- sloučením konceptu registrů a paměti,
- prací s libovolnými operandy (strojové jazyky typicky jeden vstupní operand berou zároveň jako výstupní, některé umožňují zapisovat výsledky jen do registrů),

- a zjednodušeným modelem rychlosti vykonání instrukce (žádné cache, pipeline, rozdíly v cenách instrukcí...).

Jinak jde ale o model asi nejbližší dnešním počítačům.

3. Uvažte RAM s jednotkovou cenou instrukcí a bez omezení na velikost buňek.

a) Najděte způsob, jak libovolných n čísel omezené velikosti zakódovat do jedné buňky.

Čísla naší omezené velikosti budou mít omezeně dlouhý binární zápis, řekněme b bitů. (*Omezím se na přirozená čísla, celá čísla lze řešit např. znaménkovým bitem.*) Takových čísel můžeme pomocí bitových operací „naskládat“ do jedné buňky neomezené množství. Z místa A můžeme pak dostat číslo I pomocí $(A \gg (b * I)) \& (b \text{ jedniček})$.

b) Jaké aritmetické operace jdou na takto zakódovaných „vektorech“ provádět v konstantním čase?

Taková (přirozená) čísla můžeme v konstantním čase po složkách sčítat a odečítat, dokud nepřetečou.

Násobení těchto „vektorů“ už násobení po složkách nedá, jeho výsledek je ale stejně užitečný. Když si představíme každý „vektor“ jako polynom v 2^b (nebo ekvivalentně jako číslo v 2^b -ární soustavě), dostaneme násobením „vektorů“ součin polynomů.

c) Vymyslete, jak transformovat libovolný program na RAMu s omezenou velikostí buňky do této neomezené varianty tak, aby měl konstantní paměťovou složitost.

Stačí program přepsat tak, aby se operátory prováděly na omezeném počtu pracovních registrů a každý přístup do paměti přepsat na přístup do dílčí části jedné buňky reprezentující paměť jako „vektor“.

- d) Najděte reprezentaci dvou množin vektorů A, B takovou, že skalární součin $a \cdot b$ (pro libovolné $a \in A, b \in B$) lze vypočítat v čase $\mathcal{O}(1)$. Použijte tuto reprezentaci k násobení matic v čase $\mathcal{O}(n^2)$.

Skalární součin dvou vektorů $a, b \in \mathbb{N}^n, a \cdot b = \sum a_i b_i$ lze získat z polynomů $a(x) = a_1 x^0 + \dots + a_n x^{n-1}, b(x) = b_n x^0 + \dots + b_1 x^{n-1}$ jako koeficient u x^{n-1} v jejich součinu: Sečtou se v něm právě násobky členů z a a b , jejichž mocniny x se sečtou na $n - 1$, tedy $a_1 x^0$ a $b_1 x^{n-1}, a_2 x^1$ a $b_2 x^{n-2}, \dots$

Stačí použít reprezentaci z a), kdy ale každý vektor z B zapíšeme v opačném pořadí. Podle polynomového pohledu je pak koeficient u x^{n-1} roven n -tému prvku výsledného „vektoru“.

Pro nepřírozeně rychlý algoritmus násobení matic už pak stačí vědět, že $(A \cdot B)_{ij} = A_{i*} \cdot B_{*j}$.

4. Minule jste dokázali, že existují dvojice funkcí neporovnatelné inkluzí tříd \mathcal{O} . Nyní dokažte, že to platí i pokud se omezíme jen na neklesající funkce.

Možné řešení je najít funkci f , kde pro každé c a dostatečně velké n platí, že $\frac{f(n+1)}{f(n)} > c$. Za g pak můžeme vzít stejnou funkci posunutou o půl kroku.

$$f(n) = \lfloor n \rfloor!, g(n) = \lceil n \rceil!$$

5. Navrhněte grafovou reprezentaci jízdního řádu a popište algoritmus, který najde pro výchozí zastávku, čas odjezdu a maximální čas příjezdu množinu zastávek, do kterých se lze dopravit (tzv. izochronu).

Algoritmů na hledání cest v jízdním řádě je celá řada a s nimi existuje i řada reprezentací. Asi nejjednodušší (pro použití s existujícími algoritmy) je tzv. *time-expanded graph*, kde se vezme popis dopravní sítě (obdobný mapě) a pak se vytvoří pro každý relevantní čas kopie každé zastávky.