1 (True or False) and Justify [20 bodov]

1.1 Zadanie

1. Máme pole, ktoré obsahuje n trojpísmenných kódov letísk. Dá sa toto pole usporiadať v čase lepšom ako $\Theta(n \log n)$?
2. Ak je pri hešovaní počet prvkov ostro menší ako veľkosť tabuľky, v ktorej sú uložené, tak nenastanú žiadne kolízie.
3. Pomocou dokonalej mince sa dá rovnomerne náhodne rozhodnúť medzi troma možnosťami.
4. Z hodnoty x vieme hodnotu x^{34} získať pomocou menej ako 10 násobení.
5. Dané sú dve haldy obsahujúce po n prvkov. Dá sa z nich v $O(n)$ spraviť usporiadané pole obsahujúce všetkých $2n$ prvkov?
6. Dané sú dve haldy obsahujúce po n prvkov. Dá sa z nich v $O(n)$ spraviť halda obsahujúca všetkých $2n$ prvkov?
7. Dané sú dva binárne vyhľadávacie stromy obsahujúce po n prvkov. Dá sa z nich v $O(n)$ spraviť usporiadané pole obsahujúce všetkých $2n$ prvkov?
8. Dané sú dva binárne vyhľadávacie stromy obsahujúce po n prvkov. Dá sa z nich v $O(n)$ spraviť **vyvážený** binárny vyhľadávací strom obsahujúci všetkých $2n$ prvkov?

1.2 Riešenie

1. ÁNO. V $O(n)$ napríklad pomocou triedenia RadixSort (ale dá sa aj priamo použiť CountSort).
2. NIE. Úplná hlúposť: Ak my nevolíme, aké prvky sa na vstupe objavia, nevieme zaručiť, že nenastanú kolízie. Veľkosť hešovacej tabuľky má len vplyv na ich *očakávaný* počet.
3. ÁNO. Dvakrát hodím, znak+znak, znak+hlava a hlava+znak sú moje tri možnosti, hlava+hlava znamená „začni odznova“. Očakávaná časová zložitosť je konštantná.

   (Ale uznáva sa aj odpoveď, že pre žiadne f neexistuje algoritmus s worst case časovou zložitosťou $O(f(n))$, kvôli nesúdeliteľnosti čísel 2 a 3.)
4. ÁNO. Postupne spočítame hodnoty x^2, x^4, x^8, x^{16}, x^{32} a $x^2 \cdot x^{32} = x^{34}$.
5. NIE. Keby to šlo, vedeli by sme to spraviť aj pre dve identické haldy, a teda by sme vedeli v $O(n)$ triediť.
6. ÁNO. Stačí zobrať všetkých $2n$ prvkov a algoritmom z prednášky z nich spraviť haldu.
7. ÁNO. Z každého stromu zostrojíme v $O(n)$ usporiadané pole jeho prvkov. Na dve polia, ktoré takto dostaneme, zavoláme Merge.
8. ÁNO. Zoberieme pole, ktoré sme v predchádzajúcej odpovedi zostrojili, a z neho vyrobíme strom (prostredný prvok do koreňa, rekurzívnym volaním samostatne prerobiť na strom ľavú aj pravú polovicu poľa).

2 Pohľad zvnútra: hrb [10+10 bodov]

2.1 Zadanie

Pole $A[1..n]$ obsahuje hrb. Presnejšie, obsahuje celé čísla, o ktorých sa vie, že najskôr ostro rastú a potom ostro klesajú. A ešte presnejšie, existuje z také, že $1 \leq z \leq n$ a platí $\forall i < z : \ A[i] < A[i+1]$ a $\forall i > z : \ A[i] < A[i-1]$.

Detailne implementujte funkciu, ktorá (pokiaľ možno v lepšom ako lineárnom čase) nájde v poli A maximum. Môžete predpokladať, že pole A už máte načítané v pamäti. Nezabudnite na dôkaz správnosti a odhad časovej zložitosti.

Následne dokážte, že keby sme zo zadania odstránili slová „ostro“ (t.j. zmenili $A[i] < A[i \pm 1]$ na $A[i] \leq A[i \pm 1]$), tak nebude existovať riešenie v lepšom ako lineárnom čase.

2.2 Riešenie prvej podúlohy

Upravíme binárne vyhľadávanie: V každom kroku sa pozrieme na prostredné dva prvky. Ak je ľavý menší, je hrb v pravej polovici, ak je pravý menší, je hrb v ľavej polovici. Keď nám už zostane len jeden prvok, vyhrali sme. Každú iteráciu spravíme v konštantnom čase. A keďže každou iteráciou zmenšíme pole približne na polovicu, bude iterácií $O(\log n)$ a taká je aj časová zložitosť nášho riešenia.
V nižšie uvedenom programe používam polo-otvorený interval $[vlavo, vpravo)$. Teda ak z označuje index, na ktorom je maximum, tak platí invariant $vlavo \leq z < vpravo$.

```
A[n+1] = mínus nekonečno
vľavo, vpravo = 1, n+1
kým vpravo - vľavo > 1:
    t = (vľavo + vpravo) / 2
    ak A[t-1] < A[t]: vľavo = t
    inak: vpravo = t

odpoveď = A[vľavo]
```

2.3 Riešenie druhej podúlohy

Intuitívne, ak sa program nepozrie na všetky políčka poľa A, tak pred ním hľadané maximum vieme schovať. Presnejšie, ukážeme, ako oklamať každý program, ktorý sa pozrie na najviac $n-2$ políčok poľa A.
Upraveným podmienkam zodpovedá ako pole „samé nuly“, tak aj každé pole „samé nuly a medzi nimi niekde jedna jednotka“. Vždy, keď sa nás program opýta na nejaké políčko poľa A, povieme mu, že je tam 0. Keď program skončí, vyberie si nejakú hodnotu z. Potom ale existuje $z' \neq z$ také, že na $A[z']$ sa nás program nikdy nespýtal. A teda pre vstupné pole „samé nuly, len na pozícii z' jednotka“ dá tento program nesprávnu odpoveď.
Každý korektný program sa teda musí v najhoršom prípade pozrieť na $n-1$ políčok poľa A, a teda jeho časová zložitosť musí byť aspoň lineárna, q.e.d.

3 Pohľad zvnútra: porovnanie trojice [10+10 bodov]

3.1 Zadanie

Budeme sa hrať nasledovnú hru. Mám pole a v ňom $n \geq 3$ rôzne drahých vecí, očíslovaných od 1 do n. V každom kole hry mi poviete tri rôzne čísla vecí a ja vám poviem, ktorá z tých troch je najdrahšia a ktorá najlacnejšia. Cieľom hry je usporiadať veci od najdrahšej po najlacnejšiu.

Príklad: Mám veci s cenami $(10, 47, 1, 2, 42)$. Poviete mi čísla $1, 4, 5$. Odpoviem: najdrahšia je vec 5 a najlacnejšia vec 4.

a) Dokážte, že neexistuje stratégia, ktorá zaručuje výhru v tejto hre a pritom vždy položí nanajvýš $7n$ otázok.

b) Nájdite a detailne popíšte (alebo naprogramujte) stratégiu, ktorou túto hru vyhráte, pričom položíte $O(n \log n)$ otázok.

3.2 Riešenie prvej podúlohy

Ak by takáto stratégia existovala, tak potom existuje aj algoritmus, ktorý ľubovoľné n-prvkové pole usporiada pomocou nanajvýš $3 \cdot 7n = 21n$ porovnaní. Totiž pomocou troch porovnaní vieme pre danú trojicu zistiť najmenší a najväčší prvok v nej.
A z prednášky vieme, že takýto triediaci algoritmus neexistuje – porovnaní treba aspoň rádovo $n \log n$. Preto nemôže existovať ani dotyčná stratégia.

3.3 Riešenie druhej podúlohy

Použijeme napríklad MergeSort. Vždy, keď potrebujeme porovnať veci $A[i]$ a $A[j]$, zvolíme si k rôzne od i a j a opýtame sa na trojicu i, j, k. Z odpovede vieme vždy určiť, ktorá z vecí $A[i]$ a $A[j]$ je lacnejšia: $A[i]$ je lacnejšia ako $A[j]$ vtedy, keď je vec i najlacnejšia zo zvolených troch, a ešte vtedy, keď je i prostredná a j najdrahšia.
Časová zložitosť je evidentne $O(n \log n)$ a taký je aj počet otázok, ktoré položíme. Nasleduje implementácia v C++, v ktorej vhodne naprogramujem porovnávaciu funkciu pre knižničné triedenie.

```
#include <algorithm>
#include <iostream>

bool my_less(int i, int j) {
    int k=1, x, y;
    while (k==i || k==j) ++k;
    std::cout << i << " " << j << " " << k << ": ";
    std::cin >> x >> y; // nacita najmensi a najvacsi z nasich troch
    return (x==i || y==j);
}

int main() {
    int n; std::cin >> n;
    int B[n];
    for (int i=0; i<n; ++i) B[i] = i+1;
```

```
    std::sort(B,B+n,my_less);
    for (int i=0; i<n; ++i) std::cout << B[i] << "\n";
}
```

4 Pohľad zvonka: zlato [15 bodov]

4.1 Zadanie

Veštica Teodora, ktorá sa nikdy nemýli, vyveštila ceny zlata na najbližších n dní: o i dní bude gram zlata stáť $C[i]$ centov. Teraz máte kapitál 100 eur = 10 000 centov. Môžete ľubovoľne veľa krát nakupovať a predávať zlato, a to ľubovoľne malé časti gramu. Ako pseudokód alebo kus programu napíšte čo najefektívnejší algoritmus, ktorý zistí, koľko najviac peňazí sa dá mať o n dní. Nezabudnite na dôkaz správnosti a odhad časovej zložitosti. Zaokrúhľovacie chyby ignorujte.

Príklad: Ak budú v najbližších dňoch ceny zlata $(4200, 4000, 4355, 4700, 3950)$, tak najlepšie je nakúpiť o 2 dni 2.5 gramu zlata a o ďalšie 2 dni všetko predať. Výsledný kapitál: $2.5 \times 4700 = 11750$ centov, teda 117.5 eura.

4.2 Riešenie

Po chvíli experimentovania pochopíme, že vždy, keď cena zlata ide začať stúpať, sa nám ho oplatí (za všetko, čo máme) nakúpiť. A naopak, keď stúpanie ceny skončí, chceme všetko zlato, ktoré máme, predať.
Priamo na základe tejto myšlienky sa dá aj spraviť dôkaz správnosti, aj naprogramovať riešenie s lineárnou časovou zložitosťou. Ukážeme však ešte jeden, jednoduchší pohľad.
Každú noc sa cena zlata mení. Ak rastie, chceme túto noc vlastniť zlato, aby sme na tom zarobili. Ak nerastie, chceme mať peniaze, aby sme neboli stratoví. Správnosť tejto stratégie je očividná.
Implementácia je tiež zjavná:

```
def zlato(pole C[1..n]):
    majetok = 10000
    pre i od 2 po n:
        zisk = max(1, C[i]/C[i-1])
        majetok *= zisk
    return majetok
```

5 Pohľad zvonka: anonymný list [15+10 bodov]

5.1 Zadanie

Máte časopis $\mathcal{C}$, t. j. veľmi dlhý reťazec znakov. Chcete z neho vystrihnúť niektoré písmená a z tých poskladať správu $\mathcal{S}$. (Pre jednoduchosť predpokladajme, že z druhej strany každého textu je vytlačená nejaká fotka, takže vystrihnutím písmena nijaké iné nestratíte.)

a) Na vstupe je dané malé číslo k a polia C a S, predstavujúce dané dva reťazce $\mathcal{C}$ a $\mathcal{S}$. Každý prvok v poliach C a S je číslo z množiny $\{1, 2, \ldots, k\}$: rôzne čísla predstavujú rôzne znaky.

   Ako pseudokód alebo kus programu napíšte čo najefektívnejší algoritmus, ktorý zistí, či sa daná správa dá z daného časopisu vyrobiť. Ak áno, váš program by mal pre každé písmeno správy vypísať, koľké písmeno časopisu naň použiť. Nezabudnite na dôkaz správnosti a odhad časovej zložitosti.

   Príklad: Vstup $\mathcal{C} = abeceda$ a $\mathcal{S} = baca$ môže byť zadaný ako $k = 5$, $C = (1, 2, 5, 3, 5, 4, 1)$ a $S = (2, 1, 3, 1)$.
   Jeden správny výstup pre tento vstup je postupnosť 2,7,4,1.

b) Na vstupe sú reťazce $\mathcal{C}$ a $\mathcal{S}$. Predpokladáme, že vstupné reťazce sú v tom istom európskom jazyku, teda používajú nejakú malú podmnožinu Unicode. (Každé $\mathcal{C}[i]$ a $\mathcal{S}[i]$ je jedno písmeno, trebárs v kódovaní UTF-8.)

   Ako pseudokód alebo kus programu napíšte čo najefektívnejší algoritmus, ktorý zistí počet k rôznych písmen v $\mathcal{C} + \mathcal{S}$ a vyrobí polia čísel C a S, ktoré budú spolu s číslom k vstupom pre podúlohu a).

5.2 Riešenie prvej podúlohy

Pre každé i od 1 po k si spravíme prázdnu frontu.
Prejdeme C, pričom pre každé i vložíme i do fronty číslo $C[i]$. Po skončení tejto fázy teda máme pre každé i vo fronte číslo i všetky indexy do C, na ktorých sa toto číslo vyskytuje.
Teraz môžeme zostrojovať riešenie. Spravíme si prázdny vektor, v ktorom ho budeme vyrábať. (Alebo stačí aj statické pole veľkosti $|S|$.) Pre každé i sa pozrieme do fronty číslo $S[i]$. Ak je prázdna, už sa nám minuli príslušné písmená, riešenie teda neexistuje. V opačnom prípade z fronty vyberieme jeden index, kde sa hľadané písmeno vyskytlo, a pridáme ho na koniec zostrojovaného riešenia.
Časová zložitosť algoritmu je zjavne lineárna od $|C| + |S|$, a teda optimálna.

5.3 Riešenie druhej podúlohy

Použijeme hešovanie. (Technická poznámka: Keďže v UTF-8 kódovaní znaky používané v konkrétnom jazyku tvoria viac-menej súvislý úsek, obyčajné „modulo veľkosť tabuľky“ bude výbornou hešovacou funkciou.) Prejdeme znaky $\mathcal{C}$ aj $\mathcal{S}$, pričom hešovaciu tabuľku použijeme ako (neusporiadané) asociatívne pole: ku každému znaku si budeme pamätať číslo, ktoré sme mu priradili. Vždy, keď spracúvame ďalší znak, pozrieme sa, či sa už niekedy vyskytol. Ak nie, zvýšime k a aktuálnu hodnotu k priradíme dotyčnému znaku.
Priamo počas tohto procesu sa dá vyrábať aj polia C a S. Ale rovnako dobré je najskôr v jednom prechode zistiť k a vyrobiť asociatívne pole „znak → malé číslo“ a potom v druhom prechode pomocou tohto poľa „preložiť“ $\mathcal{C}$ a $\mathcal{S}$ na C a S.
Očakávaná časová zložitosť tohto riešenia je lineárna od $|\mathcal{C}| + |\mathcal{S}|$.
Alernatívne riešenie: použijeme usporiadané asociatívne pole (napr. map v C++). Spracovanie každého znaku teraz bude mať zaručenú časovú zložitosť $\Theta(\log k)$, čo je stále dostatočne dobré.

Nasleduje kompletná implementácia v Pythone. Obe podúlohy sú spojené do jednej a namiesto front sú použité zásobníky.

```
from sys import stdin, exit
from collections import defaultdict
C, S = [ x.strip() for x in stdin.readlines() ]
pozicie = defaultdict(list)
for i,c in enumerate(C): pozicie[c].append(i+1)
odpoved = []
for s in S:
    if len(pozicie[s])==0: print "neda sa" ; exit()
    odpoved.append( pozicie[s].pop() )
for x in odpoved: print x
```