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

### 1.1 Zadanie

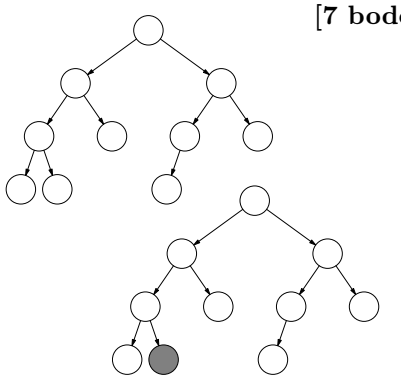
1. Strom z úlohy 2 je vyvážený (v zmysle používanom pri AVL stromoch).
2. Obsah poľa, v ktorom máme uloženú haldu, je jednoznačne určený množinou prvkov, ktoré sú v tejto halde uložené.
3. Tvar vyváženého binárneho stromu je jednoznačne určený množinou prvkov, ktorá je v ňom uložená.
4. Každý binárny strom s $n$ listami má hĺbku $O(\log n)$.
5. Časová zložitosť MergeSortu na poli s $n$ prvkami je $O(n^3)$.
6. Aladár mal pole veľkosti $n$, v ktorom boli celé čísla z rozsahu od 1 po $n^2$. Na usporiadanie tohto poľa použil CountSort: ku každej hodnote si spočítal počet výskytov a podľa toho vyplnil výstupné pole. Jeho algoritmus mal lepšiu časovú zložitosť ako by mal v tejto situácii klasický MergeSort.
7. Dá sa z haldy obsahujúcej $n$ prvkov vyrobiť v čase $O(n)$ (ľubovoľný, nie nutne vyvážený) binárny vyhľadávací strom obsahujúci tých istých $n$ prvkov?
8. Dá sa z (ľubovoľného, nie nutne vyváženého) binárneho vyhľadávacieho stromu obsahujúceho $n$ prvkov vyrobiť v čase $O(n)$ halda obsahujúca tých istých $n$ prvkov?

### 1.2 Riešenie

1. ÁNO. V každom vrchole platí, že sa hĺbky jeho podstromov líšia najviac o 1.
2. NIE. Napríklad (1,2,3) a (1,3,2) sú dve rôzne polia, ale každé z nich predstavuje haldu obsahujúcu prvky 1, 2 a 3.
3. NIE. Napríklad existujú dva rôzne vyvážené stromy obsahujúce prvky 1 a 2.
4. NIE. Ľahko sa dá vyrobiť binárny strom s $n$ listami a hĺbkou až $n-1$. Práve preto je potrebné vyvažovanie.
5. ÁNO. Toto bol chyták. Zápis $O(f(n))$ nám udáva len horný odhad časovej zložitosti, výrok je teda pravdivý. Samozrejme, tesný asymptotický odhad časovej zložitosti MergeSortu je $\Theta(n \log n)$.
6. NIE. Časová zložitosť CountSortu je $\Theta(n+r)$, kde $r$ je rozsah hodnôt, v našom prípade teda dostávame zložitosť $\Theta(n^2)$, čo je horšie ako MergeSort. (Na pamätanie si počtov výskytov potrebujeme použiť pole veľkosti $n^2$, ktoré musíme celé prejsť.)
7. NIE. Z neutriedeného poľa vieme v $O(n)$ vyrobiť haldu, z binárneho vyhľadávacieho stromu vieme v $O(n)$ vyrobiť utriedené pole. Keby bola odpoveď áno, vedeli by sme triediť v čase $O(n)$, to ale nevieme.
8. ÁNO. Vypíšeme do poľa inorder zápis stromu a máme utriedené pole, ktoré zároveň predstavuje haldu.

## 2 Pohľad zvnútra: BST [7 bodov]

### 2.1 Zadanie



Na obrázku je binárny vyhľadávací strom obsahujúci 10 navzájom rôznych prvkov.
Vyfarbite všetky vrcholy, kde sa môže nachádzať tretí najmenší prvok.
Veľmi stručne slovne zdôvodnite správnosť svojej odpovede.

### 2.2 Riešenie

V binárnom vyhľadávacom strome je jednoznačne určené poradie prvkov podľa veľkosti. Keďže ľavý podstrom koreňa obsahuje 5 prvkov a $5 \geq 3$, musí byť tretí najmenší prvok v tomto podstrome. Opakovaním úvahy dostávame jediné riešenie, znázornené na obrázku vpravo.

## 3 Pohľad zvnútra: náhodná fronta [20 bodov]

### 3.1 Zadanie

Popíšte, ako by ste implementovali dátovú štruktúru *náhodná fronta*, ktorá podporuje operácie „vlož prvok“ a „vyber náhodný prvok“. Pri výbere musia mať všetky prvky rovnakú pravdepodobnosť, že budú vybrané. Snažte sa dosiahnuť čo najlepšiu časovú zložitosť pomalšej z operácií.
Môžete predpokladať, že máte k dispozícii funkciu generujúcu vhodné náhodné čísla. Ak neviete úlohu riešiť vo všeobecnosti, pridajte si predpoklad, že v našej fronte sa nikdy naraz neobjaví viac ako 100 000 prvkov.

### 3.2 Riešenie

Väčšina riešiteľov upravovala strom alebo haldu, čo viedlo k menej alebo viac podrobne popísaným riešeniam s časovou zložitosťou každej operácie $\Theta(\log n)$. Toto je však zbytočne pomalé a zároveň zbytočne zložité.
Obe operácie sa dajú robiť v konštantnom čase a stačí nám na ne obyčajné pole (dynamicky meniace veľkosť ako vektor). Vloženie nového prvku implementujeme tak, že ho pridáme na koniec poľa. Podobne ako u vektoru, ak nám na to nestačí alokovaná pamäť, vyrobíme si dvakrát väčšiu a všetky prvky prekopírujeme.
Na výber prvku si vygenerujeme náhodné číslo od 0 po $n-1$, kde $n$ je aktuálny počet prvkov, a použijeme ho ako index do poľa. Takto nájdený prvok vypíšeme, na jeho miesto presunieme posledný prvok, aby sme nemali dieru, a pole o jedno políčko skrátime (t.j. zmenšíme počítadlo prvkov, alokovaná pamäť ostáva rovnaká).

## 4 Pohľad zvnútra: jackpot [10+3 bodov]

### 4.1 Zadanie

Na digitálnom displeji pred kasínom sa zobrazuje výška jackpotu. Každú sekundu narastie jackpot o 1 euro. Aby bol stále jackpot korektne zobrazený, je potrebné každú sekundu niektoré segmenty na displeji zhasnúť a niektoré iné rozsvietiť. Displej má $d$ cifier, každá cifra je zobrazená pomocou klasických 7 segmentov. Práve niekto vyhral jackpot, takže $d-1$ cifier nesvieti a na poslednej pozícii svieti 0.
Nájdite asymptotický odhad počtu segmentov, ktoré sa dokopy prepnú počas nasledujúcich $n$ sekúnd. (Môžete predpokladať, že $n \leq 10^d$.) Potom uveďte, čo sa zmení, keď vynecháme podmienku, že na začiatku je na displeji nula.

### 4.2 Riešenie

Ľahko je dokázateľný horný odhad $O(nd)$: v každom z $n$ krokov sa na každej z $d$ pozícii zmení nanajvýš 7 segmentov, čo je konštanta. Troška presnejší horný odhad je $O(n \log n)$. Na ten si treba všimnúť, že po $n$ krokoch máme číslo s $O(\log n)$ ciframi, a teda sa nám v každom kroku zmenilo $O(\log n)$ segmentov.
Zjavný dolný odhad je, že v každom kroku sa zmení aspoň jeden segment, teda dokopy máme $\Omega(n)$ zmien. (Veľká omega znamená „aspoň rádovo“.)

Medzi horným a dolným odhadom, ktoré zatiaľ máme, je však rozdiel. Ktorý z nich, ak vôbec niektorý, je tesný?

Spočítajme počet zmenených segmentov ináč: po cifrách. Posledná cifra sa zmení v každom kroku. Predposledná v každom desiatom. Tá pred ňou v každom stom kroku. A tak ďalej. A každá zmena cifry predstavuje zmenu najviac 7 segmentov. Celkový počet zmien segmentov počas $n$ krokov teda vieme zhora odhadnúť hodnotou $\sum_{i=0}^{d-1} \lfloor 7n/10^i \rfloor$.
A túto sumu vieme zhora odhadnúť tak, že zanedbáme celé časti a budeme sumovať až do nekonečna, čím dostávame $7n \cdot \sum_{i=0}^{\infty} 1/10^i = 7n \cdot (10/9)$. Celkový počet zmenených segmentov počas prvých $n$ krokov je preto $O(n)$, a teda $\Theta(n)$.

(Stačilo uviesť aj kvalifikovaný odhad, že zjavne je zmien na inej ako poslednej pozícii dokopy výrazne menej ako zmien na poslednej pozícii, a tých je $\Theta(n)$.)

Vyššie uvedená úvaha by takmer bez zmeny platila aj keby sa nezačínalo počítať od nuly. Len dolné celé časti by sme museli zmeniť na horné, tým však suma narastie len zanedbateľne: nanajvýš o $d$. V takomto prípade vieme teda o počte zmien segmentov povedať, že je $O(n+d)$.

## 5 Pohľad zvonka: krówky [20 bodov]

### 5.1 Zadanie

Mišof je závislý na krówkach. Musí každú minútu zjesť jednu, inak umrie v bolestivých kŕčoch. Krówky nakupuje úplne čerstvé od poľských šmelinárov. Čerstvá krówka vydrží $d$ minút, potom stvrdne a vyláme si na nej zuby. Na vstupe je číslo $d$, číslo $n$ a pole obsahujúce ceny krówiek počas najbližších $n$ minút. Nájdite efektívny algoritmus, ktorý spočíta najmenšiu celkovú cenu krówiek pre Mišofa. Dokážte jeho správnosť a odhadnite časovú zložitosť.
(Na plný počet bodov stačí čokoľvek použiteľné pre $d, n \leq 10^6$. Bonusové body za asymptoticky optimálne riešenie!)

Príklad: pre $d = 3$ a ceny $(10, 1, 5, 2, 5, 10, 10)$ optimálnu cenu 18 dosiahne takto: 1. minútu kúpi 1 krówku a zje ju, 2. minútu ich kúpi 4 (viac nestihne zjesť, kým stvrdnú) a počas 4. minúty dokúpi ďalšie 2 (tie zje počas 6. a 7. minúty).

### 5.2 Riešenie

Pre každú minútu potrebujeme určiť, kedy kúpiť krówku, ktorá bude v danú minútu zjedená. Na výber máme (nanajvýš) $d+1$ možností: buď v aktuálnu minútu, alebo v jednu z $d$ predchádzajúcich. A zjavne sa oplatí spomedzi cien za týchto $d+1$ minút vybrať tú najnižšiu.

Priamočiare riešenie, ktoré zakaždým prezrie všetky ceny prichádzajúce do úvahy, má časovú zložitosť $\Theta(nd)$. Je korektné, takže nejaké body zaň boli, ale do optimálneho má ďaleko.

Na plný počet bodov stačilo riešenie s časovou zložitosťou $O(n \log d)$. Použijeme dátovú štruktúru multiset – t.j. usporiadaná množina, ktorá môže obsahovať násobné prvky. Implementovaná je ako vyvažovaný binárny vyhľadávací strom, všetky operácie s ňou teda vieme robiť v čase logaritmickom od počtu prvkov v nej. V tejto dátovej štruktúre si budeme pamätať ceny za posledných $d+1$ minút. Vždy, keď ideme spracúvať nasledujúcu minútu, pridáme do multisetu jej cenu, vyberieme odtiaľ cenu pred $d+1$ minútami, a následne nájdeme najmenšiu zo zapamätaných cien.

Bonusové body: Existuje viacero komplikovanejších riešení, ktoré majú časovú zložitosť $O(n)$. Jedna možnosť je nasekať si zadané pole cien na úseky dĺžky $d+1$ a v každom spočítať prefixové aj sufixové minimá. Z týchto informácií už vieme v konštantnom čase určiť minimum ľubovoľne umiestneného úseku dĺžky $d+1$. Iná možnosť je použiť dátovú štruktúru deque (obojsmerná fronta) a v tej si pamätať len tie záznamy (index,cena), ktoré ešte majú šancu niekedy byť hľadaným minimom. V každom okamihu bude tento zoznam zároveň usporiadaný aj podľa indexu aj podľa ceny vzostupne. Rozmyslite si, ktoré hodnoty to presne sú, aj ako presne sa obsah našej deque zmení po prečítaní ďalšej ceny.

## 6 Pohľad zvonka: reklamy v telke [4 × 5 bodov]

### 6.1 Zadanie

Predávaš zubné kefky zahnuté do pravého uhla, ale moc nejdú na odbyt. Ešteže televízia Matfýza v tomto predvianočnom období nepretržite vysiela bloky reklám. O každom bloku $i$ je známy počet ľudí $P[i]$, ktorí ho budú sledovať. Chcel by si si dať pustiť reklamu na svoje kefky v niektorých blokoch. Pritom sleduješ dva ciele. Prvý: aby neliezla ľuďom príliš na nervy. Ten dosiahneš tak, že medzi každými dvomi jej vysielaniami bude aspoň jeden blok, v ktorom tvoja reklama vysielaná nebude. Druhý cieľ je, aby ju dokopy videlo čo najviac ľudí. Treba teda vybrať takú množinu blokov, aby súčet ich hodnôt $P$ bol najväčší možný.
Príklad: pre $P = (70\,000, 30\,000, 100\,000, 4\,500, 9\,000, 470\,000)$ by sme vybrali prvý, tretí a šiesty blok.

a) Dokážte alebo vyvráťte správnosť pažravého algoritmu: Kým máme na výber, tak v každom kroku vyberieme ten blok, ktorý nesusedí so žiadnym skôr vybratým blokom a zo všetkých takých má najviac divákov.
b) Napíšte (ako pseudokód alebo kus programu) rekurzívny algoritmus skúšajúci všetky možnosti ako vybrať povolenú sadu blokov. Odhadnite jeho časovú zložitosť zhora. (Bonusové body za asymptoticky tesný odhad!)
c) Pridajte do predchádzajúceho algoritmu memoizáciu tak, aby vznikol algoritmus s časovou zložitosťou polynomiálnou od počtu blokov $n$. Odhadnite jeho časovú zložitosť.
d) Uveďte ekvivalentný algoritmus, ktorý túto úlohu rieši pomocou dynamického programovania.

### 6.2 Riešenie a)

Najjednoduchší protipríklad je pole $(2, 3, 2)$, na ktorom pažravý algoritmus zoberie 3, ale optimálne je zobrať 2+2.

### 6.3 Riešenie b)

Predpokladáme, že vstup máme v globálnom poli `P[0..n-1]`. Napíšeme rekurzívnu funkciu `vyries(x)`, ktorá vyskúša všetky možnosti, ako vybrať povolenú podmnožinu spomedzi prvých `x` blokov – teda z poľa `P[0..x-1]`. Návratovou hodnotou je najväčší možný súčet vybraných prvkov. Fungovanie našej funkcie je jednoduché: prvé rekurzívne volanie zodpovedá možnosti, že som posledný prvok aktuálneho úseku nevybral, druhé volanie zodpovedá možnosti, že som ho vybral (a teda ten bezprostredne pred ním musím preskočiť a na výber už mám len `x-2` prvkov).

```
def vyries(x):
    if x<=0:
        return 0
    else:
        return max( vyries(x-1), P[x-1] + vyries(x-2) )
```

Určite vieme časovú zložitosť tohto algoritmu zhora odhadnúť ako $O(2^n)$, lebo zakaždým urobíme dve rekurzívne volania a zakaždým sa zmenší ich argument, takže hĺbka stromu rekurzie nikde neprekročí $n$.
(Pozor: Dá sa dokázať dolný odhad $\Omega(2^{n/2})$, to ale nie je ani rádovo tá istá funkcia! V exponente konštanty zanedbať nesmieme. Uvedomte si, že $2^{n/2} = \sqrt{2^n}$.)

### 6.4 Riešenie bonusovej otázky

Ak označíme $T(n)$ časovú zložitosť funkcie `vyries` pre pole veľkosti $n$, tak vidíme, že funkcia $T$ zjavne spĺňa rekurenciu $T(n) = T(n-1) + T(n-2) + O(1)$, odkiaľ $T(n) = \Theta(F_n)$, kde $F_n$ je $n$-té Fibonacciho číslo. Teda po zaokrúhlení $T(n) = O(1.618034^n)$, čo je o dosť lepší horný odhad.

### 6.5 Riešenie c)

Pridáme memoizáciu: majme globálne pole `memo[0..n-1]` inicializované na -1. Upravíme našu funkciu nasledovne:

```
def vyries(x):
    if x<=0:
        return 0
    else:
        # ak sme to este nepocitali, spocitame a ulozime si vysledok
        if memo[x] == -1: memo[x] = max( vyries(x-1), P[x-1] + vyries(x-2) )
        return memo[x]
```

### 6.6 Riešenie d)

```
best[-1] = best[0] = 0
# postupne pre kazde i od 1 po n spocitame vysledok
for x in [1,2,...,n]: best[x] = max( best[x-1], P[x-1] + best[x-2] )
```

V oboch posledných podúlohách je evidentné, že časová zložitosť je $\Theta(n)$, keďže každú z $n$ potrebných hodnôt vypočítame v konštantnom čase.