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

### 1.1 Zadanie

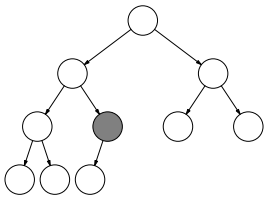
1. Existuje dátová štruktúra, ktorá v konštantnom čase zvláda vloženie prvku aj nájdenie maxima.
   (Prvky netreba vedieť odstraňovať. Ani nájdenie maxima ho z dátovej štruktúry neodstráni.)
2. Existuje dátová štruktúra, ktorá v konštantnom čase zvláda vloženie prvku aj odstránenie maxima.
3. Vektor obsahujúci $n$ prvkov, ktoré majú každý $b$ bajtov, zaberá vždy $O(bn)$ bajtov pamäte.
4. Vyvážený binárny vyhľadávací strom obsahujúci $n$ takých prvkov zaberá $\Theta(bn \log n)$ bajtov pamäte.
5. Pole $A$ má 5 prvkov, sú navzájom rôzne. Existuje algoritmus, ktorý $A$ usporiada pomocou $\leq 6$ porovnaní dvojíc.
6. V úlohe o stabilných manželstvách existuje vždy práve jedno riešenie.
7. Každý algoritmus iterujúci cez všetky podmnožiny množiny $\{0, 1, \ldots, n-1\}$ má časovú zložitosť $\Omega(n!)$.
8. Prioritnú frontu vieme efektívne implementovať ako vyvažovaný binárny vyhľadávací strom.

### 1.2 Riešenie

1. TRUE. Stačí jedna premenná, v ktorej si pamätáme to maximum. (Alebo ak sa nám bridí prvky zahadzovať, môžeme použiť vektor a udržiavať si index do neho, ukazujúci na maximálnu hodnotu.)
2. FALSE. Keby existovala, vedeli by sme s jej pomocou napísať triedenie porovnávaním bežiace v čase $O(n)$.
3. TRUE. Nanajvýš $3bn$ počas zväčšovania veľkosti na dvojnásobok.
4. FALSE. V každom vrchole je len samotný prvok a tri pointre, celý strom teda zaberá len $\Theta(bn)$ bajtov pamäte.
5. FALSE. Na 6 porovnaní vieme rozlíšiť len $2^6 = 64$ možností, ale máme $5! = 120$ možných poradí.
6. FALSE. Ukazovali sme si príklad, kedy záležalo na tom, kto „ide na pytačky“. (Obľubená chyba bola tvrdiť, že vždy existuje viac ako jedno riešenie – to tiež nie je pravda.)
7. FALSE. Obyčajný cyklus cez $[0, 2^n)$ ktorý každú podmnožinu vypíše má časovú zložitosť $\Theta(n \cdot 2^n)$.
8. TRUE. Vieme aj vkladať prvky aj vyberať najväčší v čase logaritmickom od ich počtu.

## 2 Pohľad zvnútra: halda [5+5 bodov]

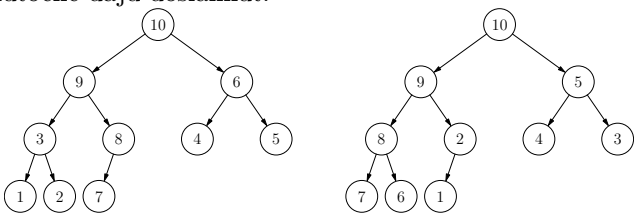
### 2.1 Zadanie



Na obrázku je binárna halda s maximom v koreni. Sú v nej uložené čísla 1 až 10.
a) Doplňte čísla do haldy tak, aby vyfarbené pole malo čo najväčšie číslo.
b) Doplňte čísla do haldy tak, aby vyfarbené pole malo čo najmenšie číslo.
Stručne zdôvodnite, prečo tam väčšie ani menšie číslo od tých vašich byť nemôže.

### 2.2 Riešenie

Vrchol je až v tretej úrovni, preto tam nemôže byť 9 ani 10. Vrchol má syna, preto tam nemôže byť 1. Nasledujúce obrázky ukazujú, že hodnoty 8 a 2 sa už skutočne dajú dosiahnuť.




## 3 Pohľad zvnútra: časová zložitosť v praxi [10+5 bodov]

### 3.1 Zadanie

Anička napísala program, ktorý načíta $n$-prvkové pole čísel, 1000× ho vypíše na výstup a potom spočíta duplikáty v poli tak, že každý prvok postupne porovnáva so všetkými predchádzajúcimi.
Odhadnite, ako dlho bude Aničkin program bežať na súčasnom bežnom počítači pre $n = 10,\ 100,\ 1000,\ \ldots,\ 1\,000\,000$.
Boris spustil Aničkin program na počítači, ktorý má v pivnici. Pre $n = 1\,000\,000$ na ňom tento program bežal asi 1.2 sekundy.
Ako dlho by tento program bežal na Borisovom počítači pre $n = 3\,000\,000$?

### 3.2 Riešenie

Program spraví $n$ načítaní čísla, $1000n$ výpisov a $n(n-1)/2$ porovnaní čísel. Jeho časová zložitosť je teda kvadratická od $n$, pričom ale pre malé $n$ preváži lineárny člen. Porovnanie čísel je rýchle, stačí pár inštrukcií, preto vieme na súčasných počítačoch spraviť zhruba $10^8$ porovnaní za sekundu. Vstup a výstup sú pomalšie, vypísať za sekundu stihneme zhruba $10^7$ čísel.

Odhadované časy behu uvádzame v tabuľke:

| $n$ | $10^1$ | $10^2$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|---|---|
| čas (s) | 0.001 | 0.01 | 0.1 | 1.5 | 60 | 5100 |

(Dôležitejší ako trafiť sa do týchto odhadov bol samozrejme vhodný postup. Z neho muselo vyplynúť, že pre malé $n$ odhady rástli zhruba lineárne, pre veľké $n$ kvadraticky.)

Boris má teda zjavne v pivnici nejaký superpočítač, na tom ale vôbec nezáleží. Pre taký obrovský vstup sa už časová zložitosť správa ako kvadratická funkcia, a teda ak vstup narastie na 3-násobok, čas behu narastie na 9-násobok ($9 = 3^2$), čiže približne na 11 sekúnd.

## 4 Pohľad zvnútra: binárny strom [15 bodov]

### 4.1 Zadanie

Napíšte (nejaký, čím efektívnejší, tým lepšie) algoritmus, ktorý k danému vrcholu binárneho vyhľadávacieho stromu nájde vrchol s nasledujúcou väčšou hodnotou. Inými slovami, napíšte (do detailov) program, ktorý sa spustí vždy, keď inkrementujem iterátor ukazujúci na prvok v `set`e.
(Vstupom je ukazovateľ na vrchol v strome, návratovou hodnotou je ukazovateľ na iný vrchol. Detaily uloženia stromu v pamäti si vhodne zvoľte.)

### 4.2 Riešenie

V každom vrchole stromu budeme mať tri ukazovatele, nazveme ich left, right a up (na ľavého syna, pravého syna a otca). Ak má aktuálny vrchol pravého syna, chceme najľavejší vrchol v jeho podstrome. V opačnom prípade ideme dohora, kým sa nám prvýkrát nestane, že sa vynoríme z ľavého syna (alebo kým neprídeme do koreňa, vtedy žiaden väčší prvok neexistuje. Implementácia v Pythone:

```
next(kde):
    if kde.right:
        kde = kde.right
        while kde.left: kde = kde.left
        return kde
    else:
        while True:
            if not kde.up: return None
            kam = kde.up
            if kde == kam.left: return kam
            kde = kam
```

Keďže ideme len dodola, resp. len dohora v strome, časová zložitosť je nanajvýš priamo úmerná jeho hĺbke – vo vyváženom strome to teda bude $O(\log n)$.
Veľmi obľúbená chyba: Keď zistíme, že pravého syna nemáme, odídeme do otca. No keď zistíme, že pôvodný vrchol je jeho pravým synom, nemôžeme ešte prehlásiť, že väčší prvok neexistuje.

## 5 Pohľad zvonka: jedenkrát zlato [15 bodov]

### 5.1 Zadanie

V poli $Z[0..n-1]$ máme od veštice Teodory presné ceny zlata (v eurách za gram) na dnes a nasledujúcich $n-1$ dní. Máme 1000 eur. Aby sme neboli podozriví, môžeme *len raz* za všetky peniaze (alebo prípadne len ich časť) zlato nakúpiť a niekedy *neskôr* (alebo prípadne hneď v ten istý deň) ho zas predať. Navrhnite algoritmus, ktorý zistí, koľko najviac peňazí môžeme mať na konci. Dokážte jeho správnosť a odhadnite časovú zložitosť. (Môžete sa bez rozpisovania odvolávať na algoritmy a dôkazy z prednášok.)
Príklad: ak $Z = (30, 29, 33, 37, 35)$, tak je najlepšie nakúpiť zajtra zlato za všetky peniaze a o dva dni neskôr ho predať.

### 5.2 Riešenie

Ideálne by bolo nakúpiť v globálnom minime a predať v globálnom maxime. To ale nemusí ísť – čo ak je globálne maximum skôr ako globálne minimum?
Priamočiare riešenie s časovou zložitosťou $O(n^2)$ vyskúša všetky dvojice dní.
Lepšie riešenie: postupne budeme skúšať všetky možnosti, kedy predať. Keď už vieme, kedy predať, kedy nakúpiť? Musí to byť v niektorý skorší deň. A spomedzi dotyčných dní nás samozrejme zaujíma ten, kedy je to najlacnejšie.
Tu sa síce ponúka použiť nejakú fancy dátovú štruktúru typu set, ale je to úplne zbytočné, keďže to ide aj jednoduchšie – stačí nám jediná premenná, v ktorej si budeme pamätať doteraz najlepšiu cenu na nákup.
Implementácia v Pythone:

```
najlepsi_vystup = vstupny_kapital
najlepsi_nakup = Z[0]

for den_predaja in range(1,n):

    # skusime nakupit najlacnejsie ako vieme a potom v dany den predat
    najlepsi_vystup = max( najlepsi_vystup, (vstupny_kapital / najlepsi_nakup) * Z[den_predaja] )

    # pre buduce dni uz budeme moct nakupovat aj v tento den
    najlepsi_nakup = min( najlepsi_nakup, Z[den_predaja] )

print( najlepsi_vystup )
```

Neuveriteľne obľúbená chyba: my v skutočnosti hľadáme dvojicu indexov $i$, $j$ takú, že $Z[j]/Z[i]$ je maximálne – ale veľa ľudí s radosťou hľadalo dvojicu ktorá maximalizuje $Z[j] - Z[i]$.

## 6 Pohľad zvonka: kozmonaut [5 × 5 bodov]

### 6.1 Zadanie

Howard ide na vesmírnu stanicu. Treba mu nabaliť čo najviac jedla. V ponuke je $n$ typov túb s jedlom. Každá tuba $i$-teho typu má kladnú celočíselnú hmotnosť $m_i$ (v gramoch) a kladné reálne číslo $c_i$ udávajúce počet kalórií. Každej tuby si Howard môže nabrať, koľko kusov chce, dokopy sa ale musí zmestiť do celkového hmotnostného limitu $M$, lebo viac ruská raketa nevynesie. Zistite, koľko najviac kalórií môže Howard zobrať so sebou.
Príklad: pre $M = 65000$, $n = 3$ a typy vecí s $(m_i, c_i) = (30000, 1000), (16000, 120), (4047, 1)$ Howard vezme 2 veci prvého typu a 1 vec tretieho typu. Celková hmotnosť bude 64047, celkové množstvo kalórií 2001.

a) Napíšte (ako pseudokód alebo kus programu) rekurzívny algoritmus skúšajúci všetky možnosti ako vybrať jedlo. (Začne napr. tým, že pre $n$-tý typ tuby postupne rekurzívne vyskúša možnosti „ešte jednu takúto vezmem“ a „už žiadnu takúto nevezmem“.) Zdôvodnite, že váš program pre ľubovoľný možný vstup naozaj skončí.
b) Pridajte do predchádzajúceho algoritmu memoizáciu tak, aby vznikol algoritmus s časovou zložitosťou polynomiálnou od počtu typov túb $n$. Odhadnite jeho časovú zložitosť.
c) Uveďte ekvivalentný algoritmus, ktorý túto úlohu rieši pomocou dynamického programovania.
d) Existujú situácie, kedy sa môžeme pozrieť na hmotnosti túb a množstvá kalórií a z nich priamo usúdiť, že niektoré tuby určite v optimálnom riešení nepoužijeme. Pre nasledujúcu sadu typov túb nájdite dva typy túb, ktoré Howard určite brať nebude (bez ohľadu na $M$). Vyslovte kritérium, ktoré ste použili, všeobecne. Dokážte jeho správnosť.
Máme $n = 7$ typov túb s $(m_i, c_i) = (3200, 1001), (447, 1), (3000, 1000), (1600, 120), (1310, 98), (2320, 84), (415, 1)$.
e) Dokážte, že nefunguje pažravý algoritmus, ktorý typy túb usporiada zostupne podľa hodnoty $c_i/m_i$ a následne zoberie každej z nich čo najviac kusov.

### 6.2 Riešenie podúlohy a)

Rekurzívna funkcia bude mať dva parametre: koľko ešte máme nevyužitej hmotnosti a spomedzi koľkých typov túb ešte vyberáme. Návratová hodnota bude maximálny počet kalórií, ktorý v danej situácii môžeme ešte mať z túb, ktoré naberieme.

```
# globalne premenne N, M_max, M[0..N-1], C[0..N-1]

def najlepsi_zisk(vaha,veci):
    if veci==0: return 0
    odpoved = najlepsi_zisk(vaha,veci-1)
    if M[veci-1] <= vaha:
        odpoved = max( odpoved, C[veci-1] + najlepsi_zisk(vaha-M[veci-1],veci) )
    return odpoved

print najlepsi_zisk(M_max,N)
```

Všimnite si, že oproti verzii, kedy z každej veci máme k dispozícii len jeden kus, sa program takmer vôbec nelíši. Jediný rozdiel je v tom, že ak tubu vyberieme, použijeme rekurzívne volanie `najlepsi_zisk(vaha-M[veci-1],veci)` namiesto `...,veci-1)` – teda si ponecháme možnosť vybrať ďalšiu rovnakú tubu.

### 6.3 Riešenie podúlohy b)

```
# globalne premenne N, M_max, M[0..N-1], C[0..N-1]
memo = {}

def najlepsi_zisk(vaha,veci):
    if veci==0: return 0
    if (vaha,veci) in memo: return memo[ (vaha,veci) ]
    odpoved = najlepsi_zisk(vaha,veci-1)
    if M[veci-1] <= vaha:
        odpoved = max( odpoved, C[veci-1] + najlepsi_zisk(vaha-M[veci-1],veci) )
    memo[ (vaha,veci) ] = odpoved
    return odpoved

print najlepsi_zisk(M_max,N)
```

Časová zložitosť je zjavne $O(Mn)$.

### 6.4 Riešenie podúlohy c)

```
# globalne premenne N, M_max, M[0..N-1], C[0..N-1], memo[0..M][0..N]

for vaha in range(M+1):
    for veci in range(N+1):
        if veci==0:
            memo[vaha][veci] = 0
        else:
            odpoved = memo[vaha][veci-1]
            if M[veci-1] <= vaha:
                odpoved = max( odpoved, C[veci-1] + memo[ vaha-M[veci-1] ][veci] )
            memo[vaha][veci] = odpoved

print memo[M_max][N]
```

### 6.5 Riešenie podúlohy d)

Definujme, že typ túb $(m_1, c_1)$ je *aspoň tak výhodný* ako typ $(m_2, c_2)$, ak súčasne platí $m_1 \leq m_2$ aj $c_1 \geq c_2$.
Tvrdenie: Ak máme medzi ponúkanými typmi túb dva typy také, že $(m_1, c_1)$ je aspoň tak výhodný ako $(m_2, c_2)$, tak existuje optimálne riešenie, v ktorom druhý typ vôbec nepoužijeme.
Dôkaz: Zoberme ľubovoľné optimálne riešenie. Všetky tuby druhého typu (ak tam nejaké sú) môžeme vymeniť za tuby prvého typu: celkový počet kalórií tým neklesne, hmotnosť tým nestúpne.
Pre inštanciu zo zadania týmto kritériom vieme vylúčiť typy vecí $(447, 1)$ a $(2320, 84)$.

### 6.6 Riešenie podúlohy e)

Prvý typ $m_1 = 1000$, $c_1 = 1000$, druhý typ $m_2 = 700$, $c_2 = 699$, maximálna povolená hmotnosť $M = 2100$. Pažravý algoritmus vezme dve tuby prvého typu a už sa mu nič iné nezmestí. Optimálne je vziať tri tuby druhého typu.