# Řazení I
# cvičení

# 1.

- Stabilní řazení je charakterizováno následujícím tvrzením

a) řazení nemá asymptotickou složitost větší než $\Theta(n \cdot \log(n))$
b) řazení nemá asymptotickou složitost menší než $\Theta(n \cdot n)$
c) řazení nemění pořadí prvků s různou hodnotou
d) řazení nemění pořadí prvků se stejnou hodnotou
e) kód řazení kontroluje překročení mezí polí

# 2a.

- Jaká posloupnost vznikne, když stabilní řadící algoritmus seřadí posloupnost $A_2\ C_2\ B_2\ B_1\ C_1\ A_1$, v níž platí $A_1 = A_2 < B_1 = B_2 < C_1 = C_2$?

a) $A_1\ B_1\ C_1\ A_2\ B_2\ C_2$
b) $A_2\ A_1\ B_2\ B_1\ C_2\ C_1$
c) $B_1\ C_1\ A_1\ C_2\ A_2\ B_2$
d) $A_2\ A_1\ B_1\ B_2\ C_2\ C_1$
e) $A_1\ A_2\ B_1\ B_2\ C_1\ C_2$

# 2b.

- Jaká posloupnost vznikne, když stabilní řadící algoritmus seřadí posloupnost $B_2, A_2, B_1, A_1, B_3, A_3$, v níž platí $A_1 = A_2 = A_3 < B_1 = B_2 = B_3$?

a) $B_1, A_1, B_2, A_2, B_3, A_3$
b) $A_1, B_1, A_2, B_2, A_3, B_3$
c) $A_2, A_1, A_3, B_2, B_1, B_3$
d) $A_1, A_2, A_3, B_1, B_2, B_3$
e) $A_3, A_2, A_1, B_3, B_2, B_1$

# 3.

- Změňte kód Insert Sortu tak, aby přestal být stabilní:

```
for (i = 1; i < n; i++) {
  insVal = a[i];
  j = i-1;
  while ((j >= 0) && (a[j] > insVal))
     { a[j+1] = a[j]; j--; }
  a[j+1] = insVal;
}
```

# 4.

- Řadíme pole **a**[1..*N*]. Máme funkci **flip**(*n*1, *n*2), která v konstantním čase obrátí pořadí prvků v poli **a** mezi indexy *n*1 a *n*2 včetně.

Dále máme funkci **isSorted**(*n*1, *n*2), která usek pole **a**[*n*1..*n*2] v konstantním čase otestuje a vrátí

| | |
|---|---|
| 0, | když jsou v úseku všechny hodnoty stejné, |
| –1, | když je úsek uspořádán v neklesajícím pořadí, |
| 1, | když je úsek uspořádán v nerostoucím pořadí, |
| 2 | ve všech ostatních případech. |

# 4a.

- K prvkům pole lze přistupovat pouze prostřednictvím uvedených dvou funkcí.

- Víme, že řazené pole je téměř vzestupně seřazené a jen prvek s minimální hodnotu, který je jediný, se nachází na neznámém místě pole. Napište kód algoritmu, který pole seřadí v čase O(*N*).

# 4b.

- Víme, že řazené pole je téměř vzestupně seřazené a pouze pro jeden jeho prvek platí **a**[$k$] > **a**[$k$+1], neznáme však hodnotu $k$. Napište kód algoritmu, který pole seřadí v čase O($N$).

# 4c.

- Pokud se podaří pomocí funkcí flip a isSorted implementovat porovnání dvou prvků a výměnu dvou prvků, můžeme pak již pro řazení použít libovolný standardní algoritmus. Napište implementaci obou operací:
- Funkce compare(**a**, *k*1, *k*2) vrátí –1, 0, 1, podle toho zda platí **a**[*k*1] < **a**[*k*2], a[k1] == **a**[*k*2], **a**[*k*1] > **a**[*k*2].
- Funkce swap(**a**, *k*1, *k*2) navzájem vymění v poli **a** prvky na pozicích *k*1 a *k*2.

# 4d.

- Když podle bodu 4c. implementujeme některý stabilní řadící algoritmus, bude jeho stabilita porušena tím, že jsme se omezili poze na funkce compare a swap, které manipulují polem neobvyklým způsobem?

# 5.

- Insert sort řadí pole obsahující prvky 1 2 4 3 (v tomto pořadí). Jaký bude celkový počet vzájemných porovnání prvků pole při tomto řazení?

# 6.

- Nahradíme-li Selection sort Insert sort-em, pak asymptotická složitost řazení

a) nutně klesne pro každá data
b) nutně vzroste pro každá data
c) může klesnout pro některá data
d) může vzrůst pro některá data
e) zůstane beze změny pro libovolná data

# 7.

- V určitém problému je velikost zpracovávaného pole s daty rovna rovna $3n^2 \cdot \log(n^2)$ kde $n$ charakterizuje velikost problému. Pole se řadí pomocí Insert sort-u. Jaká je symptotická složitost tohoto algoritmu nad uvedeným polem?

# 8.

- Insert sort řadí do neklesajícího pořadí pole o $n$ prvcích, kde v první polovině pole jsou jen dvojky, a ve druhé polovině jen jedničky. Jediné nesprávné označení asymptotické složitosti výpočtu je

a) $\Theta(n)$
b) $\Omega(n)$
c) $O(n^2)$
d) $\Theta(n^2)$
e) $\Omega(n^2)$

# 9.

- Quick sort řadí pole **4 6 1 3 8 5**, pivotní hodnota je 4. Jak bude pole rozděleno na „malé“ a „velké“ hodnoty po jednom průchodu polem?

a) 1 3 4 5 / 6 8
b) 1 / 3 4 6 8 5
c) 3 4 1 6 / 8 5
d) 3 1 6 / 4 8 5
e) 3 1 / 6 4 8 5

# 10a.

- Proveďte jeden průchod uvedeným polem, který v Quick Sortu rozdělí prvky na „malé“ a „velké“. Jako pivotní hodnotu zvolte hodnotu posledního prvku v poli. Napište, v jakém pořadí budou po tomto průchodu prvky v poli uloženy, a vyznačte, kde bude pole rozděleno na „malé“ a „velké“.

| 6 | 10 | 8 | 5 | 7 | 2 | 3 | 9 | 1 | 4 |
|---|---|---|---|---|---|---|---|---|---|

# 10b.

- Opakujte předchozí úlohu, s tím, že za pivotní hodnotu zvolíte hodnotu prvního prvku v poli.

Zadané pole se nemění:

| 6 | 10 | 8 | 5 | 7 | 2 | 3 | 9 | 1 | 4 |
|---|---|---|---|---|---|---|---|---|---|

# 11a.

- Quick sort řadí pole s $n$ prvky, přičemž první polovina pole obsahuje pouze hodnoty 50, druhá polovina pole obsahuje pouze hodnoty 100. Asymptotická složitost tohoto řazení je

**a)** $\Theta(n)$ **b)** $O(n)$ **c)** $O(n \cdot \log(n))$ **d)** $\Theta(n^2)$ **e)** $\Omega(n^2)$

# 11b.

- Quick sort řadí pole s $n$ prvky stejné hodnoty , až na jediného v polovině pole, který je větší. Asymptotická složitost tohoto řazení je

a) $\Theta(n)$
b) $O(n+\log(n))$
c) $\Theta(n+1)$
d) $O(n \cdot \log(n))$
e) $\Omega(n^2)$

# 12.

- Určete, jak dlouho bude  Quick sort řadit pole o n prvcích v případě, že všechny prvky budou mít stejnou hodnotu.

# 13.

- Popište nerekurzivní verzi algoritmu Quick-sort s využitím vlastního zásobníku. Předpokládejte, že dělení na „malé“ a „velké“ hodnoty v aktuálně řazeném úseku bude probíhá stejně jako v rekurzivní verzi a neuvádějte jeho algoritmus explicitně.