Níže uvedené úlohy představují přehled otázek, které se vyskytly v tomto nebo v minulých semestrech ve cvičení nebo v minulých semestrech u zkoušky. Mezi otázkami semestrovými a zkouškovými není žádný rozdíl, předpokládáme, že připravený posluchač dokáže zdárně zodpovědět většinu z nich.

Tento dokument je k dispozici ve variantě převážně s řešením a bez řešení.

Je to pracovní dokument a nebyl soustavně redigován, tým ALG neručí za překlepy a jazykové prohřešky, většina odpovědí a řešení je ale pravděpodobně správně :-).

---------- SORT STABILITY ----------

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 nemusí kontrolovat překročení mezí polí

2.
What sequence appears as a result of a stable sorting algorithm sorting the sequence
B_1 C_2 C_1 A_2 B_2 A_1 where $A_1 = A_2 < B_1 = B_2 < C_1 = C_2$?

a) A_1 B_1 C_1 A_2 B_2 C_2
b) A_1 A_2 B_1 B_2 C_1 C_2
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) C_1 C_2 A_1 A_2 B_1 B_2

3.
What sequence appears as a result of a stable sorting algorithm sorting the sequence
C_1 A_2 B_2 C_2 A_1 B_1, where: $A_1 = A_2 < B_1 = B_2 < C_1 = C_2$?

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

4.
Jaká posloupnost vznikne, když stabilní řadící algoritmus seřadí posloupnost
B_1 C_2 A_2 B_2 C_1 A_1, v níž platí $A_1 = A_2 < B_1 = B_2 < C_1 = C_2$?

f) A_1 B_1 C_1 A_2 B_2 C_2
g) A_1 A_2 B_1 B_2 C_1 C_2
h) B_1 C_1 A_1 C_2 A_2 B_2
i) A_2 A_1 B_1 B_2 C_2 C_1
j) A_1 A_2 B_1 B_2 C_1 C_2

5.
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

6.
The input a stable sorting algorithm is the sequence B_1 C_2 A_2 B_2 C_1 A_1. The following holds: $A_1 = A_2 < B_1 = B_2 < C_1 = C_2$. What sequence represents the output of the algorithm?

a) A_1 B_1 C_1 A_2 B_2 C_2
b) A_1 A_2 B_1 B_2 C_1 C_2
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

7.
Stabilní algoritmus řadí do neklesajícího pořadí číselnou posloupnost
$A_2, B_2, A_3, B_3, A_1, B_1$, v níž platí $A_1 = A_2 = A_3 < B_1 = B_2 = B_3$.
Po seřazení vznikne posloupnost

f) $A_1, B_1, A_2, B_2, A_3, B_3$
g) $A_1, A_2, A_3, B_1, B_2, B_3$
h) $B_1, B_2, B_3, A_1, A_2, A_3$
i) $A_2, A_3, A_1, B_2, B_3, B_1$
j) $A_3, A_2, A_1, B_3, B_2, B_1$

8.
Stabilní algoritmus řadí do neklesajícího pořadí číselnou posloupnost
$B_2, A_2, B_1, A_1, B_3, A_3$, v níž platí $A_1 = A_2 = A_3 < B_1 = B_2 = B_3$.
Po seřazení vznikne posloupnost

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) $A_3, A_2, A_1, B_3, B_2, B_1$

9.
A stable algorithm sorts a sequence. The result is sorted in non-decreasing order. The original sequence is: $B_2, A_2, B_1, A_1, B_3, A_3$, and it also holds: $A_1 = A_2 = A_3 < B_1 = B_2 = B_3$. The sorted result is:

k) $A_1, B_1, A_2, B_2, A_3, B_3$
l) $A_1, A_2, A_3, B_1, B_2, B_3$
m) $B_1, B_2, B_3, A_1, A_2, A_3$
n) $A_2, A_3, A_1, B_2, B_3, B_1$
o) $A_3, A_2, A_1, B_3, B_2, B_1$

---------- SELECTION SORT ----------

1.
V určitém problému je velikost zpracovávaného pole s daty rovna rovna $2n^3 \cdot \log(n)$ kde n charakterizuje velikost problému. Pole se řadí pomocí Selection sort-u. Asymptotická složitost tohoto algoritmu nad uvedeným polem je tedy

a) $\Theta(n^2)$
b) $\Theta(n^6 \cdot \log^2(n))$
c) $\Theta(n^3 \cdot \log(n))$
d) $\Theta(n^3 \cdot \log(n) + n^2)$
e) $\Theta(n^5 \cdot \log(n))$

2.
Pole n různých prvků je uspořádáno od druhého prvku sestupně, první prvek má nejmenší hodnotu ze všech prvků v poli.

Zatrhněte všechny možnosti, které pravdivě charakterizují asymptotickou složitost Selection Sortu (třídění výběrem) pracujícího nad tímto konkrétním polem.
$O(n)$ $\Omega(n)$ $\Theta(n)$ $O(n^2)$ $\Omega(n^2)$ $\Theta(n^2)$

3.
Implementujte Selection sort pro jednosměrně zřetězený spojový seznam.

---------- INSERT SORT ----------

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

a) 3
b) 4
c) 5
d) 7
e) 8

2.
Insert sort sorts the array containing elements 1 2 4 3 (in this order). The total number of comparisons of elements in this particular case will be:

f) 3
g) 4
h) 5
i) 7
j) 8

3.
Insert Sort sorts a sequence 4 3 1 2. Number of element comparisons during the sort is

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

4.
Insert Sort sorts a sequence 2 1 4 3. Number of comparisons it performs during the search is

a) 1
b) 3
c) 4
d) 5
e) 7

5.
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

6.

Čtyři prvky řadíme pomocí Insert Sortu. Celkový počet vzájemných porovnání prvků je

a) je alespoň 4
b) je nejvýše 4
c) je alespoň 3
d) může být až 10
e) je vždy roven 4*(4−1)/2 = 6.

Řešení Nejmenší možný počet testů je:

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.
Asymptotická složitost tohoto algoritmu nad uvedeným polem je tedy

a) $O(n^2 \cdot \log(n))$
b) $O(n^2 \cdot \log(n^2))$
c) $O(n^4 \cdot \log^2(n))$
d) $O(n^2 \cdot \log(n) + n^2)$
e) $O(n^2)$

8.

Insert sort řadí (do neklesajícího pořadí) pole o n prvcích, kde jsou stejné všechny hodnoty kromě poslední, která je menší. Jediné nesprávné označení asymptotické složitosti výpočtu je

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

9.

Insert sort řadí pole seřazené sestupně. Počet porovnání prvků za celý průběh řazení bude

a) n
b) n−1
c) n*(n−1)/2
d) n*n
e) $\log_2(n)$

10.

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

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

11.

V poli velikosti n mají všechny prvky stejnou hodnotu kromě posledního, který je menší.
Zatrhněte všechny možnosti, které pravdivě charakterizují asymptotickou složitost Insert Sortu (třídění vkládáním) pracujícího nad tímto konkrétním polem.
$O(n)$ $\Omega(n)$ $\Theta(n)$ $O(n^2)$ $\Omega(n^2)$ $\Theta(n^2)$

12.

nsert sort řadí do neklesajícího pořadí pole o n prvcích kde hodnoty od třetího do posledního prvku rostou a hodnoty prvních dvou prvků jsou stejné a v poli největší. Jediné nepravdivé označení asymptotické složitosti výpočtu je

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

13.
Insert sort řadí do neklesajícího pořadí pole o *n* prvcích, v němž jsou stejné všechny hodnoty kromě první a poslední, které jsou větší a navzájem stejné. Jediné nepravdivé označení asymptotické složitosti výpočtu je

f) $O(n)$
g) $\Theta(n)$
h) $\Omega(n^2)$
i) $O(n^2)$
j) $\Omega(n)$

14.
We perform Insert sort on an array of length *n*.The array values grow from the beginning up to the middle of the array and then they decrease towards the array end. Asymptotic complexity of this process on the given array is

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

15.
Napište rekurzivní verzi algoritmu Insert sort. Při návrhu algoritmu postupujte metodou shora dolů.

16.
Implementujte Insert sort pro obousměrně zřetězený spojový seznam.

---------- QUICK SORT ----------

1.
Proveďte (v mysli nebo na papíře) 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“.

2.
Opakujte předchozí úlohu, s tím, že za pivotní hodnotu zvolíte hodnotu prvního prvku v poli

3.
Quick sort řadí následující čtyřprvkové pole čísel.
2 4 1 3
Jako pivotní hodnotu volí první v pořadí tj. nejlevější. Jak bude pole rozděleno na „malé“ a „velké“ hodnoty po jednom průchodu polem? (Svislá čára naznačuje dělení.)
a) **1 | 4 2 3**

b) **1 2 | 3 4**
c) **1 2 | 4 3**
d) **2 1 | 3 4**

4.
Quick sort řadí následující čtyřprvkové pole čísel.
3 2 1 4
Jako pivotní hodnotu volí první v pořadí tj. nejlevější. Jak bude pole rozděleno na „malé“ a „velké“ hodnoty po jednom průchodu polem? (Svislá čára naznačuje dělení.)

a) **1 | 2 4 3**
b) **1 2 | 3 4**
c) **1 2 | 4 3**
d) **2 3 | 1 4**
e) **1 2 3 | 4**

5.
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) $\Omega(n^2)$

6.
Quick sort řadí pole s n prvky, v němž jsou všechny hodnoty stejné, až na jedinou hodnotu v polovině pole, která 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) $\Omega(n^2)$

7.
Na vstupu je neseřazené pole prvků pole[počet]. Bez toho, abyste toto pole seřadili, napište funkci, která v něm nalezne nalezne *k*-tý nejmenší prvek (= *k*-tý v seřazeném poli). Použijte metodu rozděl a panuj jako při řazení Quick-sortem, ale celé pole neřaďte. Hodnota *k* bude vstupním parametrem Vašeho algoritmu.

8.
Napište nerekurzivní verzi algoritmu Quick-sort. Při návrhu algoritmu postupujte metodou shora dolů.

---------- MERGE SORT ----------

1.
Merge sort
a) lze napsat tak, aby nebyl stabilní
b) je stabilní, protože jeho složitost je Θ (n*log(n))
c) je nestabilní, protože jeho složitost je Θ(n*log(n))
d) je rychlý (Θ(n*log(n))) právě proto, že je stabilní
e) neplatí o něm ani jedno předchozí tvrzení

2a.
V určitém problému je velikost zpracovávaného pole s daty rovna rovna $2n-2$, kde n charakterizuje velikost problému. Pole se řadí pomocí Merge Sort-u (řazení sléváním). S použitím $\Theta/O/\Omega$ symboliky vyjádřete asymptotickou složitost tohoto algoritmu nad uvedeným polem v závislosti na hodnotě n. Výsledný vztah předložte v co nejjednodušší podobě.

2b.
V určitém problému je velikost zpracovávaného pole s daty rovna rovna $2n^3$,kde n charakterizuje velikost problému. Pole se řadí pomocí Merge sort-u. S použitím $\Theta/O/\Omega$ symboliky vyjádřete asymptotickou složitost tohoto algoritmu nad uvedeným polem v závislosti na hodnotě n. Výsledný vztah předložte v co nejjednodušší podobě.

3.
Merge sort řadí pole, které obsahuje $n^3 + \log(n)$ položek. Asymptotická složitost tohoto řazení je
a) $\Theta(n^3 + \log(n))$
b) $\Theta(n^2 \cdot \log(n^3))$
c) $\Theta(n^3 \cdot \log(n))$
d) $\Theta(n \cdot \log(n))$
e) $\Theta(n^3 + n^2)$

4a.
Merge sort, který rekurzivně dělí řazený úsek vždy na poloviny, řadí pole šesti čísel: { 8, 1, 7, 6, 4, 2 }. Situace v aktuálně řazeném poli (ať už to bude původní pole nebo pomocné) bude těsně před provedením posledního slévání (merging) následující:
a) 1 7 8 2 4 6
b) 1 2 4 6 7 8
c) 8 7 6 4 2 1
d) 1 2 4 7 8 6
e) 2 4 6 1 7 8

4b.
Merge sort, který rekurzivně dělí řazený úsek vždy na poloviny, řadí pole šesti čísel: { 9, 5, 8, 7, 2, 0 }. Situace v aktuálně řazeném poli (ať už to bude původní pole nebo pomocné) bude těsně před provedením posledního slévání (merging) následující:
a) 0 2 7 5 8 9
b) 0 2 5 7 8 9
c) 2 5 7 0 8 9
d) 7 8 9 0 2 5
e) 5 8 9 0 2 7

5.
Merge sort řadí pole obsahující prvky 2 3 4 1 (v tomto pořadí). Celkový počet vzájemných porovnání prvků pole při tomto řazení bude

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

6.
Merge sort sorts the array containing elements 2 3 4 1 (in this order). The total number of comparisons of elements in this particular case will be:

f) 3
g) 4

h) 5
i) 6
j) 8

7.
Merge sort řadí pole, které obsahuje $n^3 – 5n$ položek. Asymptotická složitost tohoto řazení je

a) $\Theta(n^3 – \log(n))$
b) $\Theta(n^2 \cdot \log(n))$
c) $\Theta(n \cdot \log(n))$
d) $\Theta(n^3 – n)$
e) $\Theta(n^3 \cdot \log(n))$

8.
Merge sort řadí pole, které obsahuje $2n^3 + 3n^2$ položek. Asymptotická složitost tohoto řazení je

e) $\Theta(2n^3 + \log(n))$
f) $\Theta(n^2 \cdot \log(n^3))$
g) $\Theta(n^3 \cdot \log(n))$
h) $\Theta(n \cdot \log(n))$
i) $\Theta(n^3 + n^2)$

9.
Merge sort, který dělí řazený úsek vždy na poloviny, řadí pole šesti čísel: { 2, 9, 4, 8, 1, 6 }.
Situace v aktuálně zpracovaném poli (ať už to bude původní pole nebo pomocné) bude těsně před provedením posledního slévání (merging) následující:

a) 2 4 9 1 6 8
b) 1 2 4 6 8 9
c) 2 9 4 8 1 6
d) 9 8 6 4 2 1
e) 1 9 8 6 4 2

10.
Merge sort, který dělí řazený úsek vždy na poloviny, řadí pole šesti čísel: { 3, 6, 1, 7, 2, 5 }.
Situace v aktuálně řazeném poli (ať už to bude původní pole nebo pomocné) bude těsně před provedením posledního slévání (merging) následující:

a) 1 2 3 5 6 7
b) 3 6 1 7 2 5
c) 7 6 5 3 2 1
d) 1 3 6 2 5 7
e) 1 7 6 5 3 2

11.
Implementujte nerekurzivní variantu Merge sortu s využitím vlastního zásobníku. Vaše implementace nemusí usilovat o maximální rychlost, stačí, když dodrží asymptotickou složitost Merge sortu. Jednotlivé operace zásobníku neimplementujte, předpokládejte, že jsou hotovy, a pouze je vhodně volejte.

12.
Merge Sort lze naprogramovat bez rekurze („zdola nahoru“) také tak, že nejprve sloučíme jednotlivé nejkratší možné úseky, pak sloučíme delší úseky atd. Proveďte to.

---------- **HEAP** ----------

1.
Jedna z následujících posloupností představuje haldu uloženou v poli. Která?

a) 9 5 4 6 3
b) 5 4 2 3 9
c) 3 8 9 5 6
d) 5 1 8 9 1
e) 1 3 6 5 4

2.
Jedna z následujících posloupností představuje haldu uloženou v poli. Která?

a) 3 2 6 7 4
b) 9 3 1 6 4
c) 2 4 8 7 9
d) 6 3 8 7 3
e) 2 7 4 8 1

3.
Pouze jediná z následujících posloupností představuje haldu uloženou v poli. Označte ji.

a) 2 5 6 4 9
b) 2 6 5 4 9
c) 2 6 5 9 4
d) 2 5 4 9 6
e) 2 9 4 5 6

4.
Only one of the following sequences represents a heap stored in an array. Which one?

f) 2 5 6 4 9
g) 2 6 5 4 9
h) 2 6 5 9 4
i) 2 5 4 9 6
j) 2 9 4 5 6

5.
Pouze jediná z následujících posloupností představuje haldu uloženou v poli. Která?

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

6.
Only one of the following sequences represents a heap stored in an array. Which one?

f) 1 4 5 3 8
g) 1 4 3 8 5
h) 1 5 4 3 8
i) 1 5 4 8 3
j) 1 8 3 4 5

7.
One of the following sequences represents a heap stored in an array. Which one is it?

a) 9 5 4 6 3
b) 5 4 2 3 9
c) 3 8 9 5 6
d) 5 1 8 9 1
e) 1 3 6 5 4

8.
Pouze jediná z následujících posloupností představuje haldu. Která?

a) 39475
b) 34975
c) 43579
d) 35749
e) 45379

9.
Only one of the following sequences represents a heap. Which one?

f) 3 9 4 7 5
g) 3 4 9 7 5
h) 4 3 5 7 9
i) 3 5 7 4 9
j) 4 5 3 7 9

10.
Pouze jediná z následujících posloupností představuje haldu. Která?

a) 29386
b) 32689
c) 26398
d) 29863
e) 36289

11.
Only one of the following sequences represents a heap. Which one?

f) 2 9 3 8 6
g) 3 2 6 8 9
h) 2 6 3 9 8
i) 2 9 8 6 3
j) 3 6 2 8 9

12.
Heap sort řadí pole obsahující hodnoty 4 3 1 2. V první fázi řazení z těchto hodnot vytvoří haldu. Ta bude mít tvar

a) 1 2 4 3
b) 2 1 3 4
c) 3 1 2 4
d) 2 1 3 4
e) 2 4 1 3

13.
Implementujte prioritní frontu. Je to datový typ, který má čtyři operace
Init() -- vytvoří prázdnou prioritní frontu

Empty() -- vrací true, je-li fronta prázdná, jinak vrací false
Insert(prvek) -- vloží prvek do fronty
ExtractMin() -- vrátí prvek s nejmenší hodnotou ve frontě

---------- **HEAP SORT** ----------

1.
Heap sort řadí pole obsahující hodnoty 4 3 1 2. V první fázi řazení z těchto hodnot vytvoří haldu. Ta bude mít tvar

f) 1 2 4 3
g) 2 1 3 4
h) 3 1 2 4
i) 2 1 3 4
j) 2 4 1 3

2.
Heap sort sorts the array containing elements 4 3 1 2 (in this order). In the first phase of sorting process, the element are rearranged to form a heap. The particular form of this heap is:

k) 1 2 4 3
l) 2 1 3 4
m) 3 1 2 4
n) 2 1 3 4
o) 2 4 1 3

3.
Heap sort řadí pole obsahující hodnoty 3 4 2 1. V první fázi řazení z těchto hodnot vytvoří haldu. Ta bude mít tvar

a) 2 3 1 4
b) 3 1 2 4
c) 1 3 2 4
d) 2 1 3 4
e) 2 4 1 3

4.
Heap sort sorts the array containing elements 3 4 2 1 (in this order). In the first phase of sorting process the element are rearranged to form a heap. The particular form of this heap is:

f) 2 3 1 4
g) 3 1 2 4
h) 1 3 2 4
i) 2 1 3 4
j) 2 4 1 3

5.
Heap sort řadí pole. Poté, co v první své fázi vytvoří haldu, je v poli uložena posloupnost:
3 7 5 8 9 Algoritmus dále pokračuje v práci. Po zařazení prvního prvku (=3) na jeho definitivní místo, je situace v poli:

a) 5 7 8 9 3
b) 3 9 8 5 7
c) 7 5 8 9 3

d) 5 7 9 8 3
e) 3 5 7 8 9

6.
Heap sort řadí pole. Poté, co v první své fázi vytvoří haldu, je v poli uložena posloupnost:
2 6 3 9 8
Algoritmus dále pokračuje v práci. Po zařazení prvního prvku (= 2) na jeho definitivní místo, je situace v poli:

a) 6 3 8 9 2
b) 2 3 6 9 8
c) 3 6 8 9 2
d) 2 6 3 8 9
e) 3 9 6 8 2

7.
Předložíme-li Heap sort-u vzestupně seřazenou posloupnost délky n, bude složitost celého řazení
a) $\Theta(n)$, protože Heap sort vytvoří haldu v čase $\Theta(n)$
b) $\Theta(n^2)$, protože vytvoření haldy bude mít právě tuto složitost
c) $\Theta(n \cdot \log_2(n))$, protože vytvoření haldy bude mít právě tuto složitost
d) $\Theta(n \cdot \log_2(n))$, protože zpracování haldy bude mít právě tuto složitost
e) $\Theta(n)$, protože vytvoření i zpracování haldy bude mít právě tuto složitost

8.
Heap sort
a) není stabilní, protože halda (=heap) nemusí být pravidelným stromem
b) není stabilní, protože žádný rychlý algoritmus ($\Theta(n*\log(n))$ nemůže být stabilní
c) je stabilní, protože halda (=heap) je vyvážený strom
d) je stabilní, protože to zaručuje pravidlo haldy
e) neplatí o něm ani jedno předchozí tvrzení

9a.
Následující posloupnost představuje haldu o čtyřech prvcích uloženou v poli.
a) 1 3 4 2
b) 1 4 2 3
c) 1 2 4 3
d) 2 3 4 1

9b.
Následující posloupnost představuje haldu o čtyřech prvcích uloženou v poli
a) 1 3 4 2
b) 1 3 2 4
c) 1 4 2 3
d) 2 3 1 4

10a.
Následující posloupnost čísel představuje haldu uloženou v poli. Proveďte první krok fáze řazení v Heap Sortu (třídění haldou), tj.
a) zařaďte nejmenší prvek haldy na jeho definitivní místo v seřazené posloupnosti a
b) opravte zbytek tak, aby opět tvořil haldu.
1 5 2 17 13 24 9 19 23 22

Řešení

10b.

Následující posloupnost čísel představuje haldu uloženou v poli. Proveďte první krok fáze řazení v Heap Sortu (třídění haldou), tj.
a) zařaďte nejmenší prvek haldy na jeho definitivní místo v seřazené posloupnosti a
b) opravte zbytek tak, aby opět tvořil haldu.
4 8 11 12 9 18 13 17 21 25

11.
Zadanou posloupnost v poli seřaďte ručně pomocí heap sortu. Registrujte stav v poli po každém kroku. Nejprve po každé opravě částečné haldy od prostředku pole směrem k začátku, tj. při první fázi. Potom také po každé opravě haldy a přenesení prvku z vrcholu haldy na za konec haldy.

23 29 27 4 28 17 1 24 6 30 19

12.
Vysvětlete, jak je nutno modifikovat Heap sort, aby po jeho skončení pole obsahovalo prvky seřazené vzestupně. Algoritmus musí být stejně rychlý, nejen rekurzivně, a nesmí používat žádné další datové struktury nebo proměnné.

---------- RADIX SORT ----------

1.
We perform Radix sort on an array of n strings, the length of each string is k. Asymptotic complexity of this process is

a) $\Theta(k^2)$
b) $\Theta(n^2)$
c) $\Theta(k \cdot n)$
d) $\Theta(n \cdot \log(n))$
e) $\Theta(k \cdot n \cdot \log(n))$

2a.
Následující posloupnost řetězců je nutno seřadit pomocí Radix Sortu (přihrádkového řazení). Proveďte první průchod (z celkových čtyř průchodů) algoritmu danými daty a napište, jak budou po tomto prvním průchodu seřazena.

IIYI PIYY YIII YPPP YYYI PYPP PIPI PPYI

2b.
Následující posloupnost řetězců je nutno seřadit pomocí Radix Sortu (přihrádkového řazení). Proveďte první průchod (z celkových čtyř průchodů) algoritmu danými daty a napište, jak budou po tomto prvním průchodu seřazena.
GKKG KEEG GKGG KGKK KGEE KEEG EGEE GEEG

3.
Pole A obsahuje téměř seřazené řetězce (např. z 99% seřazené), pole B obsahuje řetězce stejné délky, ale zcela neseřazené. Radix sort seřadí
a) A asymptoticky rychleji než B
b) B asymptoticky rychleji než C
c) A asymptoticky stejně rychle jako B, ale použije více paměti pro řazení A
d) A asymptoticky stejně rychle jako B, ale použije více paměti pro řazení B
e) A asymptoticky stejně rychle jako B a použití paměti bude stejné

4.

V určitém okamžiku průběhu Radix sortu dostaneme k dispozici pomocná pole z1, k1, d1 (značení podle přednášky, pole se indexují od 0). Máme určit, podle kterého znaku právě řazení probíhá. To určit

a) lze, index tohoto znaku je uložen v k1[0]
b) lze, index tohoto znaku je uložen v d1[0]
c) lze, index tohoto znaku je uložen v z1[0]
d) nelze, protože algoritmus tuto informaci v d1 soustavně přepisuje
e) nelze, pole z1, k1, d1 tuto informaci neregistrují

5.

V určitém okamžiku průběhu Radix sortu dostaneme k dispozici pomocná pole z1, k1, d1 (značení podle přednášky). Máme určit, zda v tomto okamžiku již byl dokončen jeden z průchodů algoritmu vstupním polem (nezáleží na tom, který) nebo zda je nutno číst ještě nějaké další prvky, než se aktuální průchod ukončí. To určit

a) nelze, protože neznáme pořadí prvků ve vstupním poli
b) nelze, protože některé prvky v poli d1 mohou být stejné
c) nelze, protože některé prvky v polích z1, k1 mohou být stejné
d) lze sečtením délek všech seznamů uložených v poli d1
e) lze sečtením rozdílů odpovídajících si prvků v polích k1 a z1

6a.

Radix sort řadí pole řetězců {“dda”, “bab”, “ddc”, “aaa”, “bcd”, “dbc”, “bbb”, “add”, “ccd”, “dab”, “bbc” }. napište, jak budou zaplněna jednotlivá pomocná pole (z, k, d, podle přednášky) po prvním průchodu algoritmu tj, po seřazení podle posledního znaku.

6b.

Radix sort řadí pole řetězců {"dad", "caa", "cad", "aac", "bca", "dbc", "bbd", "ddc", "cda", "dac", "bbc" }. napište, jak budou zaplněna jednotlivá pomocná pole (z, k, d, podle přednášky) po prvním průchodu algoritmu tj, po seřazení podle posledního znaku.

7a.

Radix sort řadí pole řetězců. V aktuálním okamžiku provádí průchod podle 3 znaku od konce a ještě jej nedokončil. Obsah jednotlivých pomocných polí je uvedený (značení odpovídá přednášce):

z1	a	b	c	**k1**	a	b	c	**d1**	1	2	3	4	5	6	7	8	9	10
	3	**10**	**5**		**1**	**2**	**5**		**0**	**0**	**8**	**0**	**0**	**0**	**2**	**1**	**0**	**7**

Napište, jak se obsah bude postupně měnit po zpracování každého z dalších řetězců:
“abbac”(9), ”aaaaa”(4), ”bbbac”(6), čísla v závorkách uvádějí pozici řetězce ve vstupním poli.

7b.

Radix sort řadí pole řetězců. V aktuálním okamžiku provádí průchod podle 4 znaku od konce a ještě jej nedokončil. Obsah jednotlivých pomocných polí je uvedený (značení odpovídá přednášce):

z1	a	b	c	**k1**	a	b	c	**d1**	1	2	3	4	5	6	7	8	9	10	11
	8	**0**	**3**		**1**	**0**	**2**		**0**	**0**	**2**	**0**	**0**	**0**	**9**	**7**	**1**	**0**	**0**

Napište, jak se obsah bude postupně měnit po zpracování každého z dalších řetězců:
“baaab”(11), ”ccaba”(4), ”ababa”(6), ”babab”(5), ”bbbbc”(10), čísla v závorkách uvádějí pozici řetězce ve vstupním poli.

8.

Profesor Vylepšil je toho mínění, že by se Radix sort dal využít i pro řazení kladných celých čísel. Navrhuje následující postup. Každé číslo celé bude interpretovat jako posloupnost čtyř znaků reprezentujích jeho zápis v soustavě o základu 256. Jinými slovy, hodnotu každého bytu tohoto čísla bude považovat za samostatný znak. Například číslo 1697043971 se rovná $101 \cdot 256^3 + 38 \cdot 256^2 + 214 \cdot 256^1 + 3 \cdot 256^0$,
takže jej lze v tomto návrhu zapsat jako <101><38><214><3>, kde každé číslo společně s jeho závorkou interpretujeme jako jeden samostatný symbol.
Uvažte, jaké změny je potřeba učinit v kódu Radix sortu, aby bylo možno tento návrh impelementovat a posuďte, nakolik je relevantní pro řazení různě velkých polí celých čísel.

9.
Profesor Omezil se zadíval na datové struktury Radix sortu a prohlásil, že ukazatelé na začátky seznamů (pole Z na přednášce) jsou zbytečné. Navrhuje, aby každý seznam byl cyklický, to jest, aby poslední prvek seznamu odkazoval na první prvek. První prvek pak bude vždy dostupný jako následník posledního prvku. Ukazatele na poslední prvky seznamů je však vhodno v paměti udržovat díky tomu, že se na konec seznamů neustále průběžně přidávají prvky a seznamy se prodlužují.
My si o návrhu profesora Omezila myslíme, že pokud bude fungovat, omezí sice poněkud velikost použité paměti, ale zato zvýší čas provádění kódu, protože referenci na začátek každého seznamu bude nutno průběžně v poli následníků přepisovat, zatímco v původní podobě se při každém průchodu daty zaznamenávají jen jednou.
Implementujte navrhované změny, rozhodněte zda budou funkční a pokud ano, za domácí úkol porovnejte praktické rychlosti obou variant.

---------- COUNTING SORT ----------

1
Uvažujme pole obsahující 1000 navzájem různých čísel v pohyblivé řádové čárce. Counting sort se pro toto pole

a) hodí, protože toto řazení má lineární složitost
b) hodí, protože toto řazení má sublineární složitost
c) hodí, protože čísla lze převést na řetězce
d) nehodí, protože čísla v poli nemusí být celá
e) nehodí, protože čísla jsou navzájem různá

2.
Counting sortem řadíme pole čísel:
8 14 14 7 11 11 6 3 12 11 2 12 14 9 8
Jaký bude počáteční obsah pole četností pro tato data?

3a.
Counting sortem řadíme pole čísel:
50 88 87 87 93 87 23 53 70 89 53 62.
Jaký bude nejmenší a největší index v poli četností?

a) 50 62
b) 0 62
c) 0 93
d) 23 93
e) 50 93

3b.
Counting sortem řadíme pole čísel:
56 54 20 13 44 75 84 39 31 34 68.
Jaký bude nejmenší a největší index v poli četností?

a) 20 56
b) 20 84
c) 1 56
d) 56 84
e) 56 68

4a.

Řadíme 14 celých čísel. Těsně předtím, než se začne plnit výstupní pole v Counting sortu, je obsah původního pole četností následující (slabě psaná čísla jsou indexy):

10	11	12	13	14	15	16	17	18	19
1	**3**	**4**	**8**	**10**	**12**	**12**	**12**	**13**	**14**

Napište, jak vypadá výsledné uspořádané pole, za předpokladu, že všechna pole začínají indexem 1.

4b.

Řadíme 15 celých čísel. Těsně předtím, než se začne plnit výstupní pole v Counting sortu, je obsah původního pole četností následující (slabě psaná čísla jsou indexy):

10	11	12	13	14	15	16	17	18	19
0	**1**	**2**	**3**	**7**	**9**	**9**	**13**	**14**	**15**

Napište, jak vypadá výsledné uspořádané pole, za předpokladu, že všechna pole začínají indexem 1.

5.

Na začátku Counting sortu je v poli četností uložena právě četnost výskytu každé hodnoty ve vstupním poli. V průběhu řazení se obsah pole četností průběžně mění. Rozhodněte, zda je možno po dokončení celého řazení rekonstruovat z modifikovaného pole četností původní (a ovšem i výsledné) četnosti jednotlivých řazených hodnot.

6.

Níže je uveden jednoduchý kód funkce, která implementuje Counting sort, jejím vstupem je neseřazené pole *a*, jejím výstupem je seřazené pole *b*. Obě pole jsou deklarována mimo funkci. Funkce ale musí používat pole četností, které sama deklaruje. Ve stávající implementaci toto pole začíná od nulového indexu a končí na indexu rovném maximální hodnotě v řazeném poli. To je paměťově nevýhodné řešení, protože počáteční část pole četností se nevyužije, pokud je minimální hodnota v řazeném poli větší než nula. Dále toto řešení předpokládá, že všechny řazené hodnoty jsou nezáporné, což je ještě vážnější omezení.

Modifikujte daný kód tak, aby velikost pole četností byla minimální, to jest rovna rozdílu maximální a minimální hodnoty v řazeném poli zvětšené o 1. Zároveň funkce musí být schopna zpracovávat i záporná čísla ve vstupním poli.

```
void countSort0(int a[], int [] b) {
 if (a.length != b.length) return;
 int min = a[0];
 int max = a[0];
 int val;

 // get min and max value in input array
 for (int i = a.length%2; i < a.length; i += 2 )
   if (a[i] < a[i+1]) {
     if (a[i] < min) min = a[i];
     if (a[i+1] > max) max = a[i+1];
   }
   else {
     if (a[i+1] < min) min = a[i+1];
     if (a[i] > max) max = a[i];
   }
 // frequency array
 int [] freq = new int[max+1];
 for (int i = 0; i < freq.length; i++ )
    freq[i] = 0;        // unnecessary in java

 // fill frequencies
 for (int i = 0; i < a.length; i++ )
   freq[a[i]]++ ;
 for (int i = min+1; i <= max; i++ )
   freq[i] += freq[i-1];

 // sort
```

```
 for (int i = a.length-1; i >= 0; i-- ) {
   val = a[i];
   freq[val]--;
   b[freq[val]] = val;
 }
}
```

7.

Je dána posloupnost A={ a_1, a_2,..., a_n } uložená v poli *p*, pro níž platí:

- 0 <= a_i <= 20 000 pro všechna *i*
- žádná z hodnot v posloupnosti se neopakuje více než 50-krát
- 10 000 <= n <= 20 000

Napište algoritmus, který tuto posloupnost uspořádá v lineárním čase, přičemž použije maximálně tři průchody posloupností a využije pomocnou paměť o velikosti O(n). Uspořádaná posloupnost bude nakonec uložena v původním poli *p*. Úplná deklarace datových stuktur (2 body), algoritmus

Porovnání řazení

1.

Následující dvojici řazení je možno implementovat tak, aby byla stabilní

a) Heap sort a Insertion sort
b) Selection sort a Quick sort
c) Insertion sort a Merge sort
d) Heap sort a Merge sort
e) Radix sort a Quick sort

2.

Poskytneme-li Quick Sort-u a Merge sort-u stejná data k seřazení, platí

a) Quick sort bude vždy asymptoticky rychlejší než Merge Sort
b) Merge sort bude vždy asymptoticky rychlejší než Quick Sort
c) někdy může být Quick sort asymptoticky rychlejší než Merge Sort
d) někdy může být Merge sort asymptoticky rychlejší než Quick Sort
e) oba algoritmy budou vždy asymptoticky stejně rychlé

3.

Společná vlastnost Insert sort-u, Select sort-u a Bubble sort-u je následující:

a) všechny tyto algoritmy jsou vždy stabilní
b) všechny tyto algoritmy jsou vždy nestabilní
c) všechny tyto algoritmy mají složitost $O(n \cdot \log_2(n))$
d) všechny tyto algoritmy mají složitost $O(n^2)$
e) všechny tyto algoritmy mají složitost $\Theta(n^2)$