Radix sort

1.

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

2.

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

3.

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

4a.

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.

4b.

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.

5a.

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.

5b.

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.

6.

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.

7.
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

8.
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

9.
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?

10a.
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

10b.
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

11a.
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.

11b.
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.

12.
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.

13.
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;
  }
}
```