1. Řazení quicksort

Je dáno pole celý čísel - { 8, 5, 7, 9, 11, 6, 17, 18 }, které chceme setřídit vzestupně. Uvažujeme algoritmus quicksort z přednášky – klasickou verzi Hoare's partition.

Určete, který z prvků bude prvním pivotem a proveďte první dělení (partition).

Řešení:

Prvním pivotem je číslo 8, po prvním dělení bude posloupnost vypadat následovně { 6, 5, 7, 8, 11, 9, 17, 18 }.

2. Příklad na ADT fronta

ADT fronta je reprezentován kruhovým polem. Popište následující funkce pro práci s tímto ADT:

- vkládání prvků do fronty,
- výběr prvků z fronty,
- ověření, zda fronta je prázdná.

Řešení:

Je dáno pole pevné délky *n* a dva ukazatele – na začátek a konec fronty, ukazatele *head* a *tail*. Při vkládání do fronty vložíme prvek na místo *tail* a ukazatel posuneme. Pokud ukazatel dosáhne hranici *n*, nastavíme ho na začátek pole – index 0. Ukazatel na konci fronty obecně ukazuje na první volné místo v poli. Při výběru prvků z fronty postupujeme obdobně na místě ukazatele na začátek fronty, ukazatele *head*.

To, zda je fronta prázdná zjistíme porovnáním head == tail – fronta je prázdná.

3. Příklad na pravděpodobnost

Jaká je střední hodnota návratové hodnoty randomizovaného algoritmu reprezentovaného metodou `f` pro dané `n` (celé kladné číslo) a `t` z intervalu [0,1]?
Funkce `nextDouble` vrací rovnoměrně rozdělená náhodná čísla z intervalu [0, 1).

```
public static double f(int n, double t) {
      Random r = new Random();
      double[] a = new double[n];
      for(int i = 0; i< n; i++) {
          if(r.nextDouble() >= t) {
              a[i] = 1.0;
          } else {
              a[i] = -2.0;
          }
      }
      double s = 0.0;
      for(int i=0; i<n; i++) s+= a[i];
      return s/n;
}
```

Řešení:

Pravděpodobnost, že a[i] bude 1 je 1-t, tj. v poli máme 1-t jedniček. Pravděpodobnost, že a[i] bude -2 je t, v poli máme t hodnot -2. Jejich součet je $1 * (1-t) + t(-2) = 1 - 3t$.

4. Řazení v O(n)

Použijte stabilní verzi algoritmu counting sort pro řazení pole celých čísel 1, 5, 2, 0, 2, 2. Znázorněte průběh algoritmu: ukažte každou změnu ve výstupním poli.

Řešení:

Zvolíme rozsah vstupních hodnot k = 5 (klíče tedy budou z intervalu 0..k).

Pole výskytů: `[1, 1, 3, 0, 0, 1]`

Pole částečných součtů `C = [1, 2, 5, 5, 5, 6]`

Jednotlivé iterace výstupního pole B pak vypadají následovně (včetně úprav pole C, které nemusí být součástí řešení):

```
B = [?, ?, ?, ?, 2, ?], C = [1, 2, 4, 5, 5, 6]
B = [?, ?, ?, 2, 2, ?], C = [1, 2, 3, 5, 5, 6]
B = [0, ?, ?, 2, 2, ?], C = [0, 2, 3, 5, 5, 6]
B = [0, ?, 2, 2, 2, ?], C = [0, 2, 2, 5, 5, 6]
B = [0, ?, 2, 2, 2, 5], C = [0, 2, 2, 5, 5, 5]
B = [0, 1, 2, 2, 2, 5], C = [0, 1, 2, 5, 5, 5]
```

5. Příklad na rozptylování (hashing)

Uvažte rozptylovací tabulku rozměru M = 7 a následující rozptylovací funkci h: **`h(k) = k mod M`**. Uvažte dále, že pro ukládání do tabulky používáte lineární prohledávání (linear probing) a odpovídající funkci je **`h1(k,i) = (h(k) + i) mod M`**, kde i je pořadí pokusu. Nakreslete obsah tabulky po vložení posloupnosti hodnot: 5, 12, 14, 20, 21, 11.

Rozmyslete, jaký je nejnižší a nejvyšší index prvků tabulky.

Řešení:

Postup plnění tabulky lze zachytit následující řadou:

0	1	2	3	4	5	6
						5

Pokus č.1: ((5 mod 7) + 1) mod 7 = 6

					5	12

Pokus č.1: ((12 mod 7) + 1) mod 7 = 6 – kolize

Pokus č.2: ((12 mod 7) + 2) mod 7 = 0

12	14					5

Pokus č.1: ((14 mod 7) + 1) mod 7 = 1

12	14	20				5

Pokus č.1: ((20 mod 7) + 1) mod 7 = 0 – kolize

Pokus č.2: ((20 mod 7) + 2) mod 7 = 1 – kolize

Pokus č.3: ((20 mod 7) + 3) mod 7 = 2

12	14	20	21			5

Pokus č.1: ((21 mod 7) + 1) mod 7 = 1 – kolize

Pokus č.2: ((21 mod 7) + 2) mod 7 = 2 – kolize

Pokus č.3: ((21 mod 7) + 3) mod 7 = 3

12	14	20	21		11	5

Pokus č.1: ((11 mod 7) + 1) mod 7 = 5