1. **Co splňuje řadící algoritmus, když o něm říkáme že je datově citlivý ?**

Jeho složitost jse mění v závislosti na uspořádání prvků, které má řadit.
Např. Bubble Sort seřadí 5 4 2 1 vzestupně se složitosní $O(n^2)$ a 1 2 4 5 se složitostí $\Omega(n)$.
Rychleji zpracuje částečně seřazenou posloupnost.

2. **Co splňuje řadící algoritmus, když o něm říkáme že je stabilní ?**

Zachováva pořadí stejných prvků. Napr. Bubble Sort, Shake Sort, Insetion Sort jsou stabilní.

3. **Za jakých podmínek lze použít bucketSort s lineární složitostí ?**

BucketSort s lineární složitostí lze použít pokud bude rovnoměrné rozložení prvků v "kýblech". BucketSort lze použít pouze tehdy, pokud znám předem minimum a maximum řazených prvků, aby bylo možný předem určit intervaly pro přidělení po kýblu a počet intervalů je lineární.

4. **Za jakých podmínek lze použít countingSort s lineární složitostí ?**

Třídí na principu pomocného pole, kterému se říká *sčítací pole*, někdy také *indexovací pole*. To bude dlouhé tak, jak je dlouhý rozsah prvků v původním poli a budou v něm zatím samé nuly. Původní pole se lineárně projede a prvky v něm jsou brány jako indexy do sčítacího pole. Pro každý prvek si ve sčítacím poli zvýšíme hodnotu (započítáme ho). Potom druhým lineárním průchodem projedeme sčítací pole a podle počtů na jednotlivých indexech do původního pole zleva zapisujeme indexy zpět jako prvky.

5. **Jaký je smysl Heapify v binární haldě ? Popište slovně/pseudokódem jak funguje. Fázy tvorby haldy můžete vynechat. Určete složitost heapify a heapysort.**

Heapify – zařadí prvek na správne místo, složitost O(n.logn)

Vstup: *Pole A a index i takový, že binární podstromy s kořeny v $A[\text{LEFT}(i)]$ a $A[\text{RIGHT}(i)]$ jsou binární haldy (splňují H-vlastnost), ale přitom $A[i] < A[\text{LEFT}(i)]$ nebo $A[i] < A[\text{RIGHT}(i)]$.*

```
procedure HEAPIFY(A, i)
{
    l ← LEFT(i);
    r ← RIGHT(i);
    if (i ≤ Heap_Size(A) & A[l] > A[i])
        then Largest ← l else Largest ← i;
    if (r ≤ Heap_Size(A) & A[r] > A[Largest])
        then Largest ← r;
    if (Largest ≠ i)
        then {A[i] ↔ A[Largest]; HEAPIFY(A, Largest)}
}
```

HeapSort

```
procedure HeapSort(A)
{
    Build_Heap(A);
    for (i ← length(A) downto 2)
        do { A[1] ↔ A[i];
             Heap_Size(A) ← Heap_Size(A) − 1;
             Heapify(A, i)
           }
}
```

Věta

Časová složitost $t_{\mathrm{HS}}(n) = O(n \log n)$.

HeapSort - neboli řazení haldou je jeden z nejlepších obecných algoritmů řazení, založených na porovnávání prvků.

6. Definujte prostorovou složitost problému a časovou složitost :

Časová složitost

- délka vstupu – počet buněk, který vstup zabírá
- délka výpočtu pro konkrétní vstup – počet provedení instrukcí, které algoritmus vykoná než se zastaví
- je funkce, kterou označujeme

$$T_{Alg.}(n): N \to N$$

kde *n* je délka vstupu

Prostorová složitost

- velikost paměti algoritmu pro konkrétní vstup – je číslo *p+1*, kde *p* je *max* z adres buněk navštívených během výpočtu

$$S_{Alg.}: N \to N$$