

15. Metody vnitřního řazení

Rozdělení řadících algoritmů

{IVT IV.
3}

- Vnitřní metody řazení
 - Předpokládají, že řazená data jsou celá ve vnitřní paměti počítače
 - ke všem datům je stejně rychlý přístup.
- Vnější metody řazení
 - Řazená data nebo jejich část se nachází ve vnější paměti
 - Soubory na disku, pásmu, proudy dat ze zařízení, sítě, atd.
 - k některým datům je řádově pomalejší přístup než k jiným.

Řazená data

{IVT IV.
3}

Pole celočíselných klíčů

- Velikost pole: **n**
- Index počátku: **0**
- Index posledního prvku: **n-1**
- Index **d**



- Dělící čára – dělí pole na seřazenou a neseřazenou část.
- Prvek na indexu **d** je prvním prvkem neseřazené části pole.



Selection sort

{IVT IV.
3}

- Metoda přímého výběru
- Princip
 - Dělí pole na seřazenou a neseřazenou část
 - V neseřazené části postupně hledá minimální prvky.
 - Nalezená minima vkládá na konec seřazené části.
 - Opakuje tak dlouho, dokud není neseřazená část prázdná.

Selection sort - algoritmus

{IVT IV.
3}

1. Rozděl pole

- seřazená část má velikost 0, neseřazená má velikost n
- Index **d = 0**

2. Dokud je neseřazená část delší než 1

1. Najdi index nejmenšího prvku v seřazené části: **minIndex**
2. Prohod' prvky na indexech **d** a **minIndex**
3. Posuň dělící čáru: **d = d + 1**

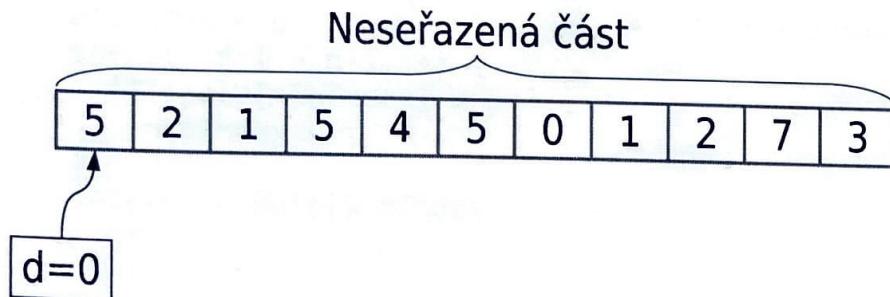
Selection sort - demonstrace

{IVT IV.
3}

Selection sort - demonstrace

{IVT IV.
3}

- Rozděl pole



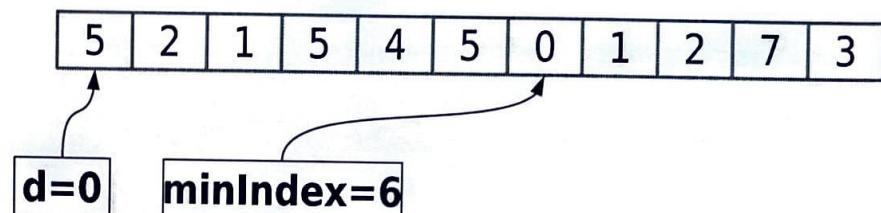
Programování - teorie, 5. ročník

11/124

Programování - teorie, 5. ročník

12/124

- Najdi minimum v neseřazené části



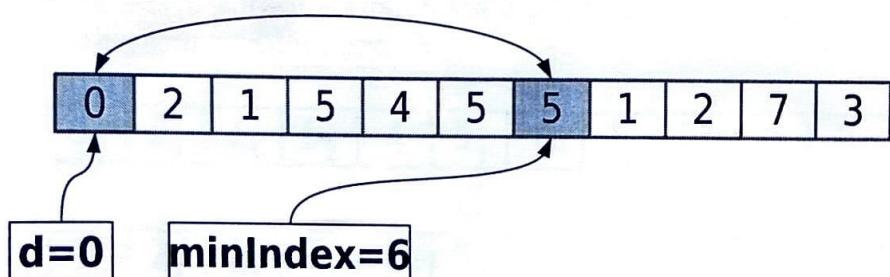
Selection sort - demonstrace

{IVT IV.
3}

Selection sort - demonstrace

{IVT IV.
3}

- Prohod' pole[d] a pole[minIndex]



Programování - teorie, 5. ročník

13/124

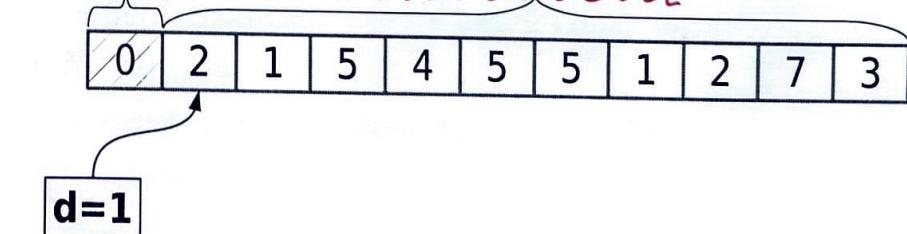
Programování - teorie, 5. ročník

14/124

- Posuň dělící čáru → zmenší neseřazenou část

Seřazená část

Nesřazená část



Selection sort - pseudokód

{IVT IV.
3}

```
algorithm SelectionSort(in: pole, n) {
    for (d: 0 ~ n-2, +) { // proč n-2? poslední první můžeme přeskočit, protože je na spodině mohou
        minIndex ← d
        for (i: d+1 ~ n-1, +) { // proč d+1?
            if (pole[minIndex] > pole[i])
                minIndex ← i
        } // for i
        pole[d] ↔ pole[minIndex]
    } // for d
}
```

Programování - teorie, 5. ročník

15/124

Selection sort - vlastnosti

{IVT IV.
3}

- Časová složitost: $O(n^2)$ - kvadratická
 - Dva vnořené cykly s lineární složitostí
- Prostorová složitost
 - Lineární, in situ.
 - Dodatečné paměťové nároky pro pomocné proměnné: konstantní $O(1)$
- Není přirozený
 - Řadí stejně dlouho nezávisle na charakteru vstupních dat.
- Je sekvenční
 - Prvky navštěvuje postupně jak leží vedle sebe.

Programování - teorie, 5. ročník

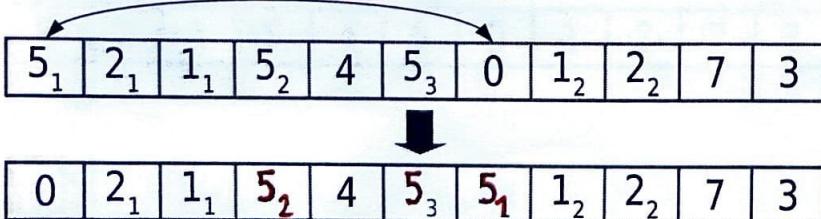
16/124

Selection sort - vlastnosti

{IVT IV.
3}

! • Není stabilní

- Relativní pořadí stejných klíčů může být změněno při nalezení nižšího minima.



Programování - teorie, 5. ročník

17/124

Selection sort

{IVT IV.
3}

- Výhody
 - Snadná implementace
 - Seřazenou část pole lze použít i před dokončením řazení.
- Nevýhody
 - Velká časová složitost
 - Nestabilita
 - Hodí se jen pro malé objemy dat.

Programování - teorie, 5. ročník

18/124

Insert sort

{IVT IV.
3}

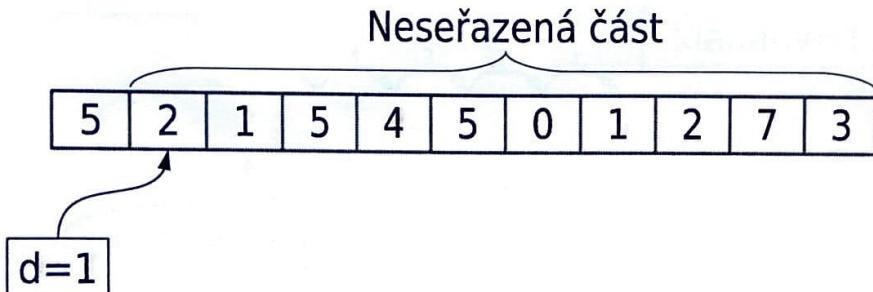
- Metoda vkládání
- Princip
 - Dělí pole na seřazenou a neseřazenou část.
 - Z neseřazené části bere první prvek a vkládá jej na správné místo v seřazené části.
 - Opakuje tak dlouho, dokud není neseřazená část prázdná.
 - Princip řazení karet v ruce.

Programování - teorie, 5. ročník

Insert sort - demonstrace

{IVT IV.
3}

- Rozděl pole



Programování - teorie, 5. ročník

Insert sort - algoritmus

{IVT IV.
3}

1. Rozděl pole
 - seřazená část má velikost 1, neseřazená má velikost n-1,
 - Index d = 1
2. Dokud je neseřazená část delší než 0
 1. Pamatuj si hodnotu prvku na indexu d v proměnné **vkládaný**
 2. Najdi pozici pro vložení prvku v seřazené části a posuň všechny prvky seřazené části od této pozice o jednu pozici doprava.
 3. Vlož **vkládaný** prvek na nalezenou pozici.
 4. Posuň dělící čáru: d = d + 1

19/124

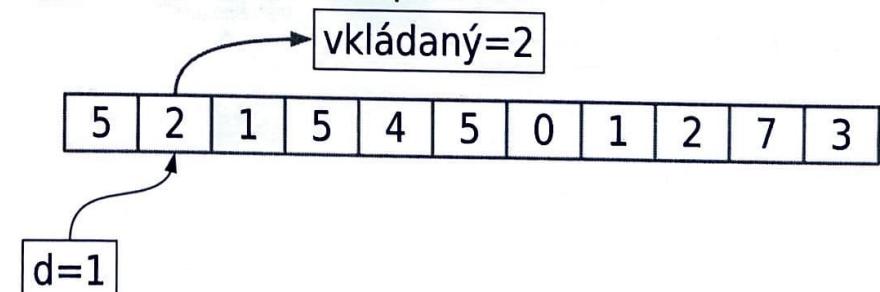
Programování - teorie, 5. ročník

20/124

Insert sort - demonstrace

{IVT IV.
3}

- Pamatuj si prvek na pozici d



21/124

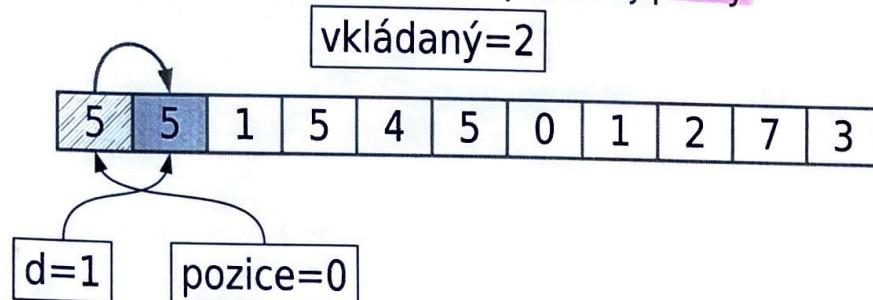
Programování - teorie, 5. ročník

22/124

Insert sort - demonstrace

{IVT IV.
3}

- Hledej pozici pro vložení a posunuj prvky

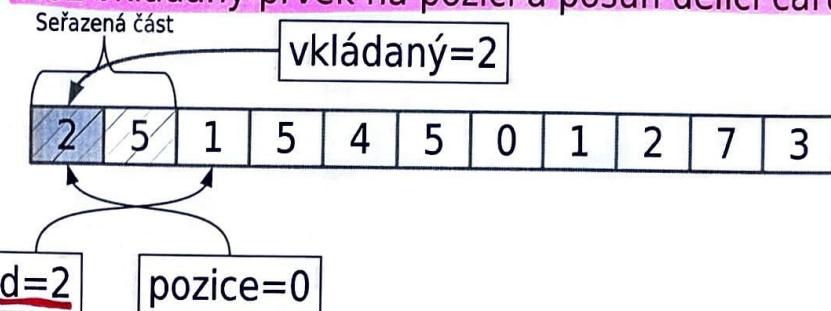


Programování - teorie, 5. ročník

Insert sort - demonstrace

{IVT IV.
3}

- Vlož vkládaný prvek na pozici a posuň dělící čáru



23/124

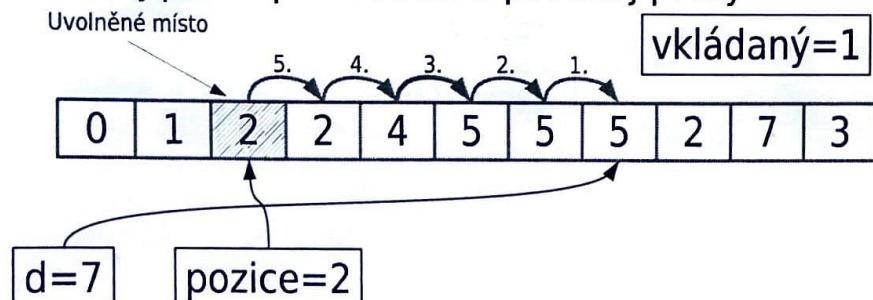
Programování - teorie, 5. ročník

24/124

Insert sort - demonstrace

{IVT IV.
3}

- Hledej pozici pro vložení a posunuj prvky

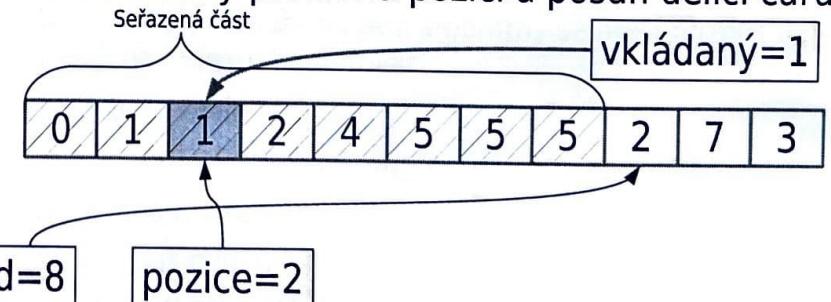


Programování - teorie, 5. ročník

Insert sort - demonstrace

{IVT IV.
3}

- Vlož vkládaný prvek na pozici a posuň dělící čáru



25/124

Programování - teorie, 5. ročník

26/124

Insert sort - pseudokód

{IVT IV.
3}

```
algorithm InsertSort(in: pole, n) {  
    for (d: 1 → n-1, +) {  
        vkladany ← pole[d] // odložení vkládaného  
        i ← d // i je index hledané pozice  
        while((i > 0) AND (pole[i-1] > vkladany)){  
            pole[i] ← pole[i-1]  
            i ← i - 1  
        } // while  
        pole[i] ← vkladany // vložení vkládaného  
    } // for d
```

Programování - teorie, 5. ročník

27/124

Insert sort se zarážkou - pseudokód

{IVT IV.
3}

```
algorithm InsertStopSort(in: pole, n) {  
    // n o 1 větší!  
    for (d: 2 → n, +) {  
        pole[0] ← vkladany ← pole[d] // zarážka  
        i ← d // i je index hledané pozice  
        while (pole[i-1] > vkladany) {  
            pole[i] ← pole[i-1]  
            i ← i - 1  
        } // while  
        pole[i] ← vkladany  
    } // for d
```

Programování - teorie, 5. ročník

29/124

Insert sort se zarážkou

{IVT IV.
3}

- Optimalizace procesu vkládání
 - před seřazenou část vložíme vkládaný prvek,
 - v podmínce pak není třeba testovat začátek
 - Ušetříme podstatnou část kódu
- Zarážka
 - Vyrobíme pole o 1 prvek delší
 - Délka pole n je o 1 větší než skutečná délka dat.
 - Alternativně - n znamená délku dat a předpokládám, že pole je o 1 větší (tedy index s hodnotou n je legální).
 - Na pozici 0 budeme vkládat vkládaný prvek - zarážka.
 - Samotná data nyní začínají od indexu 1

Programování - teorie, 5. ročník

28/124

Insert sort se zarážkou

{IVT IV.
3}

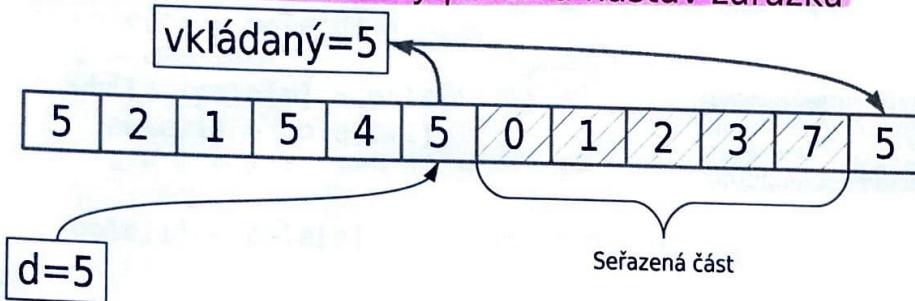
- Zarážka na pozici 0
 - Nevýhodné, protože ostatní algoritmy se musí přizpůsobit a prvek na indexu 0 ignorovat.
- Zarážka za polem
 - Vyrobíme pole o 1 prvek delší
 - Data jdou do indexu **n-2**, zarážku umisťujeme na index **n-1**.
 - Algoritmus nyní musí fungovat opačně.
 - Výhodnější pro spolupráci s ostatními algoritmy.

Programování - teorie, 5. ročník

30/124

Insert sort se zarážkou - demonstrace IVT IV.

- Zapamatuj si vkládaný prvek a nastav zarážku

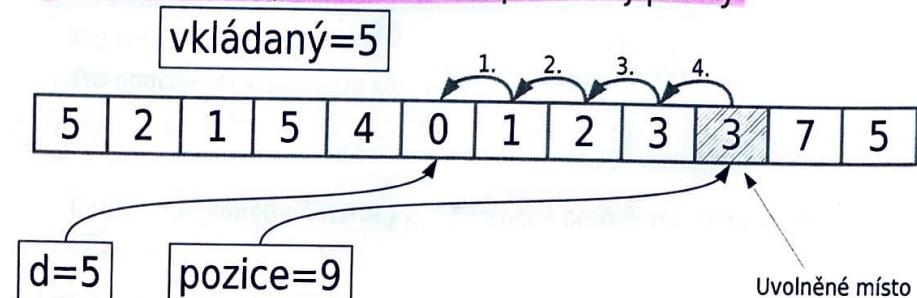


Programování - teorie, 5. ročník

31/124

Insert sort se zarážkou - demonstrace IVT IV.

- Hledej pozici pro vložení a posunuj prvky

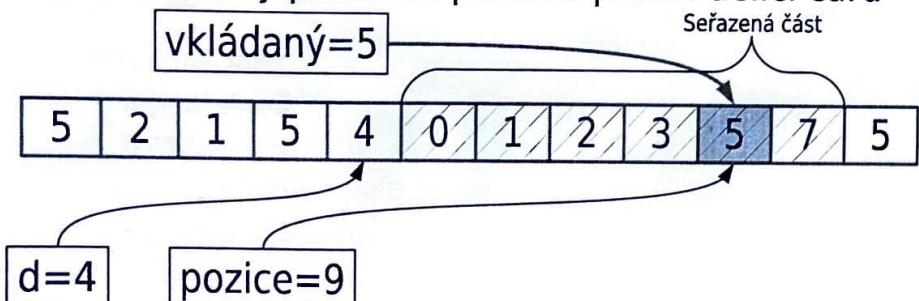


Programování - teorie, 5. ročník

32/124

Insert sort se zarážkou - demonstrace IVT IV.

- Vlož vkládaný prvek na pozici a posuň dělící čáru



Programování - teorie, 5. ročník

33/124

Insert sort se zarážkou na konci IVT IV.

```
algorithm InsertEndStopSort(in: pole, n){//n o 1 větší
    for (d: n-3 ~ 0, -) { // teď jdeme od zadu
        pole[n-1] ← vkládaný ← pole[d] // zarážka
        i ← d // i je index hledané pozice
        while (vkládaný pole[n-1] > pole[i+1]) {
            pole[i] ← pole[i+1]
            i ← i + 1
        } // while
        pole[i]←vkládaný←pole[n-1] //vložení odloženého
    } // for d
}
```

Programování - teorie, 5. ročník

34/124

Insert sort se zarážkou na konci

{IVT IV.
3}

```
algorithm InsertEndStopSort-v2(in: pole, m) {
    for (d: m-2 → 0, -) { // teď jdeme od zadu
        pole[m] ← pole[d] // zarážka
        i ← d // i je index hledané pozice
        while (pole[m] > pole[i+1]) {
            pole[i] ← pole[i+1]
            i ← i + 1
        } // while
        pole[i] ← pole[m] // vložení odloženého
    } // for d
}
```

Programování - teorie, 5. ročník

35/124

m je délka dat,
předpokládám, že
délka pole je o 1 delší

Insert sort - vlastnosti

{IVT IV.
3}

• Časová složitost - kvadratická $O(n^2)$

- Dva vnořené cykly s lineární složitostí
- Pro seřazené pole $O(n)$
- Pro opačně seřazené pole $O(n^2)$

• Prostorová složitost

- Lineární, in situ.
- Dodatečné paměťové nároky pro pomocné proměnné: konstantní $O(1)$

• Je sekvenční

• Je stabilní

Programování - teorie, 5. ročník

36/124

Insert sort - vlastnosti

{IVT IV.
3}

• Je přirozený

- Seřazené pole
 - nejlepší případ
 - Nemusí nic zařazovat
- Opačně seřazené pole
 - nejhorší případ
 - Zařazuje vždy na nejdelší vzdálenost

Programování - teorie, 5. ročník

37/124

Insert sort

{IVT IV.
3}

• Výhody

- Snadná implementace
- Seřazenou část pole lze použít i před dokončením řazení.
- Lze použít i jako metodu vnějšího řazení

• Nevýhody

- Na velké objemy dat existují rychlejší algoritmy

Programování - teorie, 5. ročník

38/124

- na 20. posluchu pole rozkládají jde o Quick sort

Bubble sort

{IVT IV.
3}

- Bublinkové řazení, řazení záměnou
- Princip

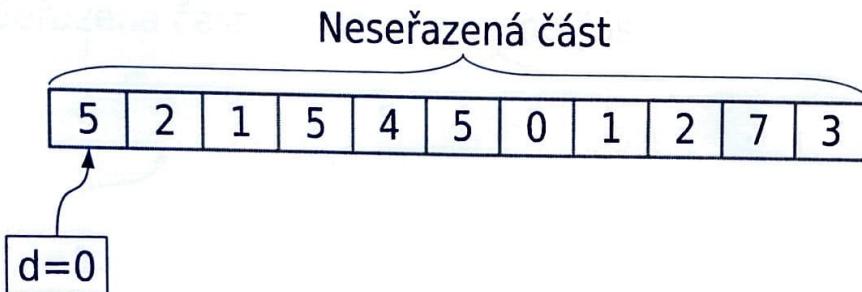
- Dělí pole na seřazenou a neseřazenou část
- V neseřazene části postupně porovnává všechny sousední prvky a prohodí je, když nejsou ve správném pořadí.
- Minimální prvek takto probublá až k seřazene časti.
- Neseřazene část se tímto zkrátí o jedničku.
- Opakuje tak dlouho, dokud není neseřazene část jednoprvková.

Programování - teorie, 5. ročník

Bubble sort - demonstrace

{IVT IV.
3}

- Rozděl pole



Programování - teorie, 5. ročník

Bubble sort - algoritmus

{IVT IV.
3}

1. Rozděl pole

- seřazena část má velikost 0, neseřazena má velikost n
- Index d = 0

2. Dokud je neseřazena část delší než 1

1. V neseřazene časti postupně porovnej všechny sousední prvky. Pokud nejsou ve správném pořadí, prohod' je.
2. Posuň dělící čáru: d = d + 1

39/124

Programování - teorie, 5. ročník

40/124

Bubble sort - demonstrace

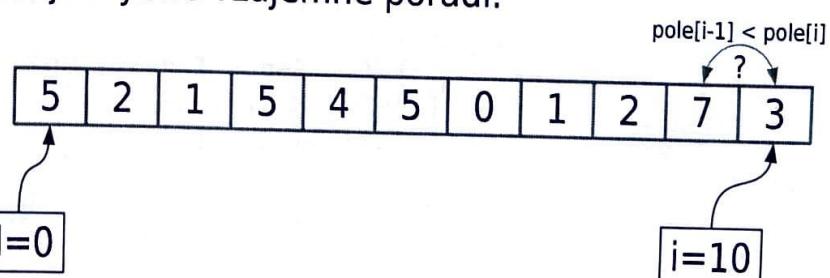
{IVT IV.
3}

- Rozděl pole

Bubble sort - demonstrace

{IVT IV.
3}

- Projdi neseřazenu část a prohod' dvojice, které mají chybné vzájemné pořadí.



41/124

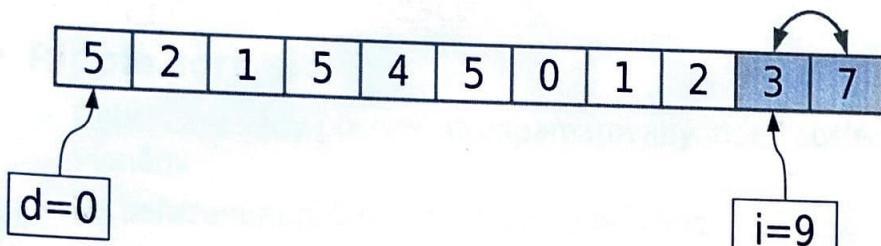
Programování - teorie, 5. ročník

42/124

Bubble sort - demonstrace

{IVT IV.
3}

- Projdi neseřazenou část a prohod' dvojice, které mají chybné vzájemné pořadí.



Programování - teorie, 5. ročník

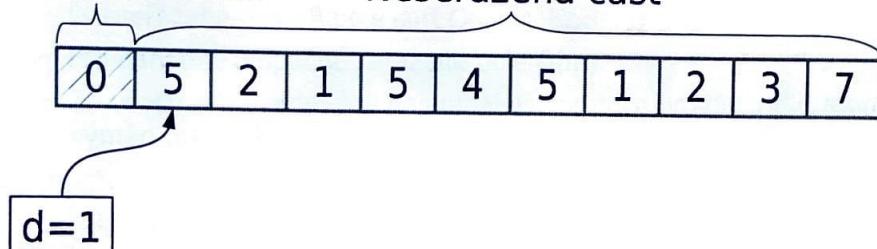
Bubble sort - demonstrace

{IVT IV.
3}

- Posuň dělící čáru

Seřazená část

Neseřazená část

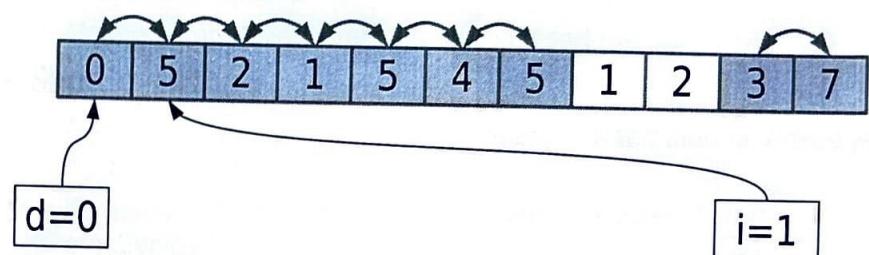


Programování - teorie, 5. ročník

Bubble sort - demonstrace

{IVT IV.
3}

- Projdi neseřazenou část a prohod' dvojice, které mají chybné vzájemné pořadí.



43/124 Programování - teorie, 5. ročník

44/124

Bubble sort - pseudokód

{IVT IV.
3}

```
algorithm BubbleSort(in: pole, n) {
    // posun zarázky
    for (d: 0 .. n-2, +) {
        // „probublávání“ menších prvků zprava doleva
        for (i: n-1 .. d+1, -) {
            if (pole[i-1] > pole[i]) {
                pole[i] ↔ pole[i-1] // výměna prvků
            } // if
        } // for i
    } // for d
}
```

45/124

Programování - teorie, 5. ročník

46/124

Bubble sort - varianty

{IVT IV.
3}

- V základní verzi se zbytečně testují i již seřazené části pole → optimalizace

Ripple sort

- Dělící čáru vždy posune na zapamatovaný index poslední výměny.
- Na seřazeném poli mu stačí jediný průchod.

Bubble sort - varianty

{IVT IV.
3}

Shaker sort

- Dva vnitřní cykly - jeden posouvá minimum doleva, druhý posouvá maximum doprava
- Možno kombinovat s variantou Ripple sort

Shuttle sort

- Dojde-li k výměně, bublá s menším prvkem tak dlouho, dokud jej nezařadí na správné místo.
- Kombinace Bubble sortu a Insert sortu (náš Insert sort je efektivnější).

Bubble sort - vlastnosti

{IVT IV.
3}

Časová složitost - kvadratická $O(n^2)$

- Dva vnořené cykly s lineární složitostí
- Pro seřazené pole: Ripple sort $O(n)$, základní $O(n^2)$
- Pro náhodné a opačně seřazené pole $O(n^2)$
- Ze všech zde uvedených nejpomalejší, protože provádí příliš mnoho výměn (nejdražší operace)

Prostorová složitost

- Lineární, in situ.
- Dodatečné paměťové nároky pro pomocné proměnné: konstantní $O(1)$

Bubble sort - vlastnosti

{IVT IV.
3}

Přirozenost

- Základní verze není
- Optimalizované verze jsou

Je sekvenční

Je stabilní

- Při správné implementaci.

Bubble sort - vlastnosti

{IVT IV.
3}

- Výhody
 - Snadná implementace
 - Ripple sort je nejrychlejší pro rozpoznání seřazeného pole.
- Nevýhody
 - Nejpomalejší z běžných řadících algoritmů.
 - Příliš mnoho výměn prvků.
 - V praxi se (téměř) nepoužívá.

Programování - teorie, 5. ročník

Quick sort

{IVT IV.
3}

- Metoda rychlého řazení
 - Sir C. A. R. Hoare, 1962
- Princip
 - Zcela odlišný od předchozích algoritmů.
 - Princip „rozděl a panuj“.
 - Rekurzivní algoritmus
 - Lze vytvořit i nerekurzivní verzi, ale vyžaduje implementaci vlastní datové struktury zásobník.

51/124 Programování - teorie, 5. ročník

52/124

Quick sort - algoritmus

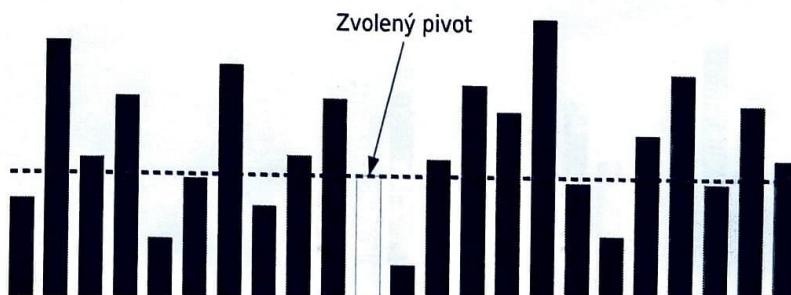
{IVT IV.
3}

1. Pokud je pole delší než 1 prvek
 1. Vyber z pole pivot
 2. Rozděl pole podle velikosti pivota
 1. Prvky menší než pivot vyměňuj s prvky většími než pivot tak, aby menší byly v poli vlevo a větší vpravo.
 - Pivot nyní dělí pole přibližně na dvě poloviny. V levé části jsou prvky menší než pivot. Ostatní prvky jsou v pravé části.
 3. Rekurzivně zpracuj stejným způsobem levou a pravou část pole.

Programování - teorie, 5. ročník

Quick sort - dělení podle pivota

{IVT IV.
3}

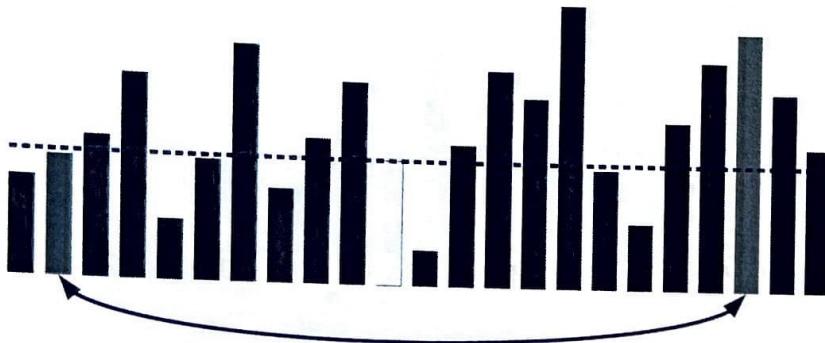


53/124 Programování - teorie, 5. ročník

54/124

Quick sort - dělení podle pivota

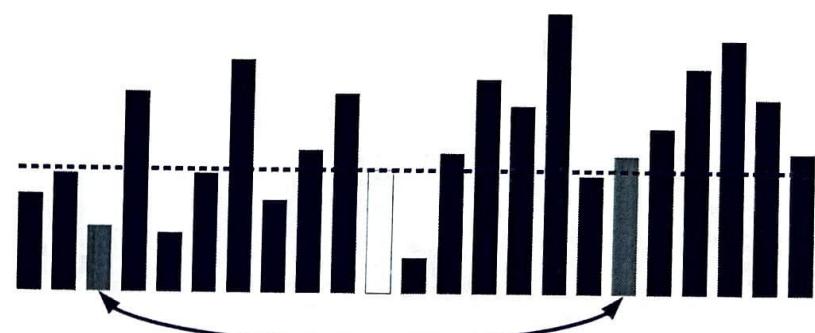
{IVT IV.
3}



Programování - teorie, 5. ročník

Quick sort - dělení podle pivota

{IVT IV.
3}

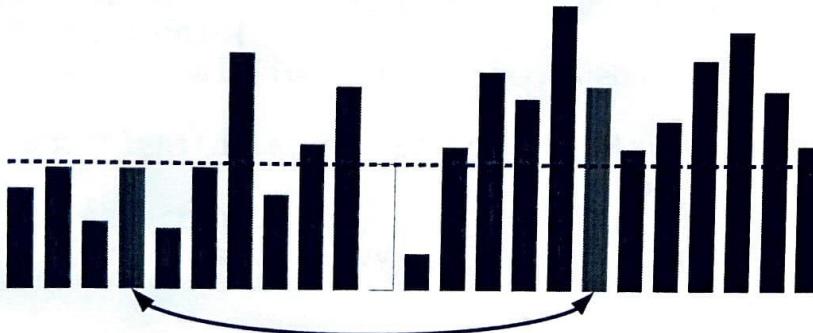


55/124 Programování - teorie, 5. ročník

56/124

Quick sort - dělení podle pivota

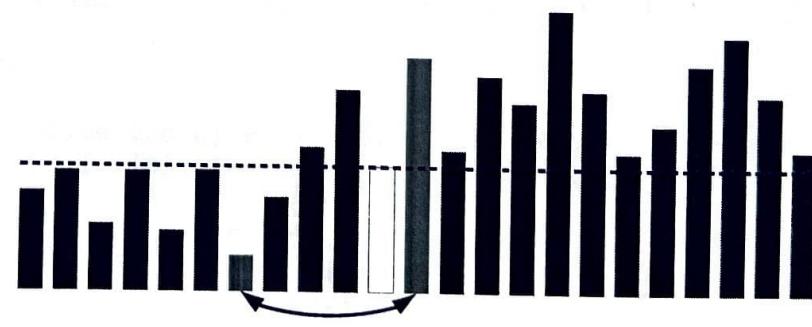
{IVT IV.
3}



Programování - teorie, 5. ročník

Quick sort - dělení podle pivota

{IVT IV.
3}

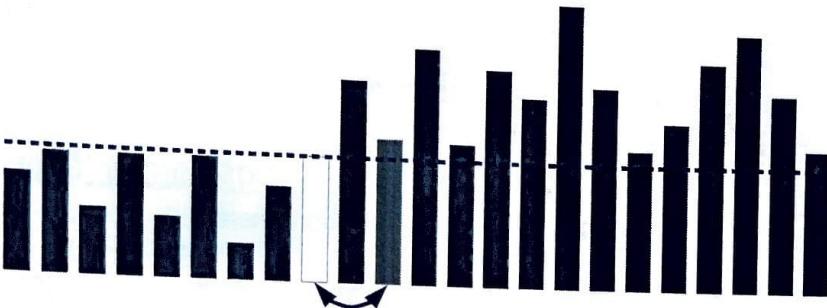


57/124 Programování - teorie, 5. ročník

58/124

Quick sort - dělení podle pivota

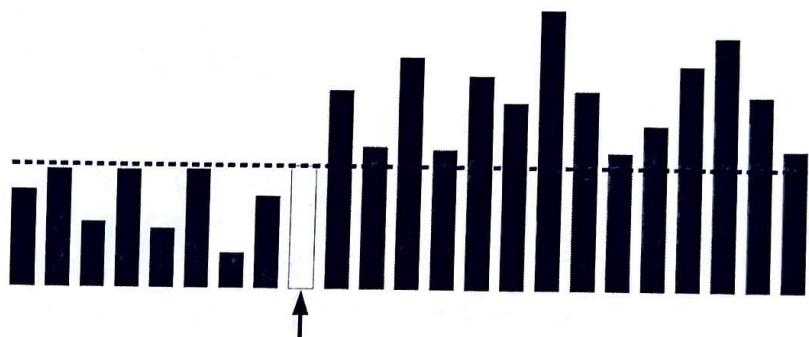
{IVT
3 IV.}



Programování - teorie, 5. ročník

Quick sort - dělení podle pivota

{IVT
3 IV.}



59/124 Programování - teorie, 5. ročník

60/124

Quick sort - pseudokód

{IVT
3 IV.}

```
algorithm QuickSort(in: pole, od, po) {
    if (od < po) {
        pivot ← pole[(od + po)/2]; // nebo jiný výběr
        rozdeleni(pole, od, po, pivot, &levy, &pravy);
        QuickSort(pole, od, pravy);
        QuickSort(pole, levy, po);
    } // if
}
```

Programování - teorie, 5. ročník

61/124

Quick sort - pseudokód

{IVT
3 IV.}

```
algorithm rozdeleni(in:pole,od,po,pivot, out:levy,pravy)
{ levy ← od; pravy ← po // indexy prvků pro výměnu
do {
    // hledání prvků pro výměnu zleva a zprava
    while (pole[levy] < pivot AND levy < po) ++levy
    while (pivot < pole[pravy] AND pravy > od) --pravy
    // výměna prvků a posun indexů
    if (levy < pravy) { pole[levy] ↔ pole[pravy] }
    if (levy ≤ pravy) { ++levy; --pravy }
} while (levy < pravy)
}
```

Programování - teorie, 5. ročník

62/124

Quick sort

{IVT IV.
3}

- Pivot

- Lze použít jakýkoli prvek pole, ale kvalita výběru dramaticky ovlivňuje celkovou časovou složitost algoritmu.
- Ideálně medián
 - prvků pole s menší hodnotou než medián je přibližně stejně, jako prvků s větší hodnotou
- Medián je obtížné nalézt
 - Přesné nalezení mediánu má lineární složitost, což je neakceptovatelné.

Programování - teorie, 5. ročník

63/124

Quick sort

{IVT IV.
3}

- Pivot

- První nebo poslední prvek pole.
 - Snazší implementace, ale silně nevýhodné pro částečně seřazené posloupnosti $\rightarrow O(n^2)$
- Prostřední prvek pole
 - V praxi častá volba. U specifických případů opět vede k $O(n^2)$.
- Náhodný prvek pole
- Medián z několika pseudonáhodně vybraných prvků.
 - Výhodné u velkých objemů dat.

Programování - teorie, 5. ročník

64/124

Quick sort - vlastnosti

{IVT IV.
3}

- Časová složitost **$O(n \log n)$**

- **$O(n \log n)$** pro průměrný případ
 - Ideálně, když je pivot vždy medián
- **$O(n^2)$** pro nejhorší případ
 - Když pivot systematicky dělí pole na silně asymetrické části.

- Prostorová složitost

- Dodatečné paměťové náklady na zásobník pro rekurzi **$O_s(\log n)$**
- V nejhorším případě to ale může být až **$O(n)$**

Programování - teorie, 5. ročník

97/124

Quick sort - vlastnosti

{IVT IV.
3}

- Přirozenost závisí na dělení a výběru pivota

- Nejhorší případ nemusí být opačně seřazené pole.

- Sekvenční

- Jde naprogramovat sekvenčně - pivot jako první prvek, ale má to dříve zmiňované důsledky.

- V naší implementaci **není sekvenční** výběr pivota.

- Není stabilní

- V průběhu rozdělování snadno dojde k výměně pozic stejných klíčů.

Programování - teorie, 5. ročník

98/124

Quick sort - vlastnosti

{IVT IV.
3}

- Výhody
 - V průměru pro náhodná data je suverénně nejrychlejší.
- Nevýhody
 - Nestabilní
 - Citlivé na pečlivou implementaci
 - Rekurze - větší paměťové nároky

Programování - teorie, 5. ročník

Merge sort

{IVT IV.
3}

- Řazení sléváním
 - John von Neumann, 1945
 - Lze řadit jak mezi vnitřní, tak mezi vnější metody řazení !
- Princip
 - Princip „rozděl a panuj“
 - Rekurzivní algoritmus
 - Rekurzivně dělí pole na poloviny
 - Seřazené poloviny slévá do větších seřazených částí

99/124

Programování - teorie, 5. ročník

100/124

Merge sort - algoritmus

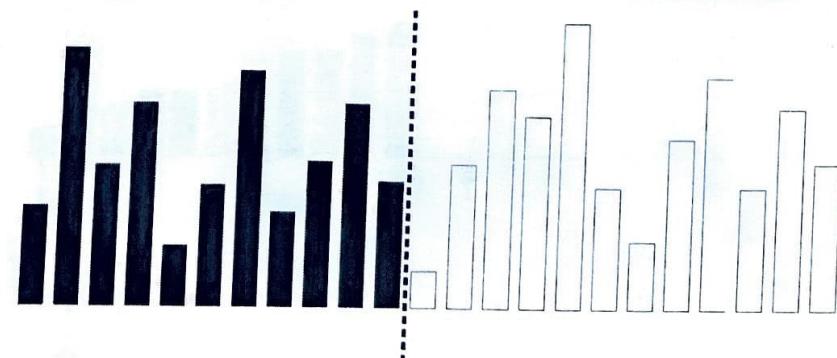
{IVT IV.
3}

1. Prázdné a jednoprvkové pole je již seřazeno.
2. Pokud je pole delší než 1 prvek
 1. Rozděl pole na poloviny podle indexů
 2. Seřaď levou polovinu
 - Rekurzivní aplikace algoritmu na levou polovinu pole
 3. Seřaď pravou polovinu
 - Rekurzivní aplikace algoritmu na pravou polovinu pole
 4. Sléváním vytvoř z obou polovin seřazené pole
 - Použij k tomu pomocné pole stejné délky, jako vstupní pole.

Programování - teorie, 5. ročník

Merge sort - dělení podle indexů

{IVT IV.
3}



101/124 Programování - teorie, 5. ročník

102/124

Merge sort - seřazení částí

{IVT
3 IV.}

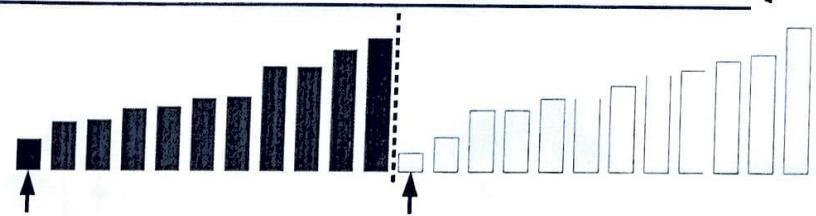


Programování - teorie, 5. ročník

103/124

Merge sort - slévání seřazených částí

{IVT
3 IV.}

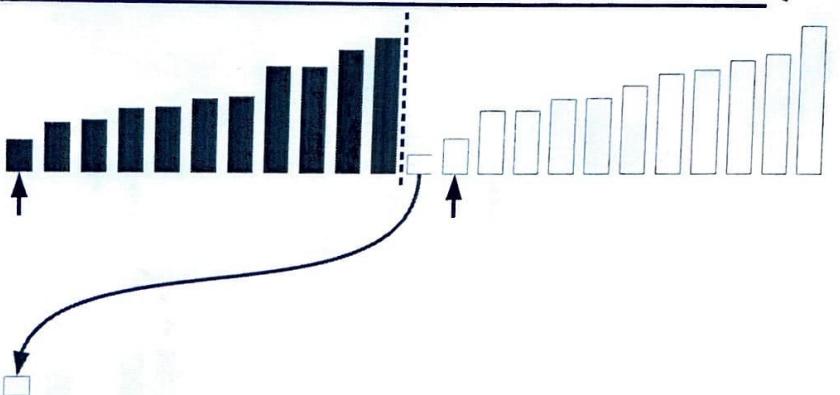


Programování - teorie, 5. ročník

104/124

Merge sort - slévání seřazených částí

{IVT
3 IV.}

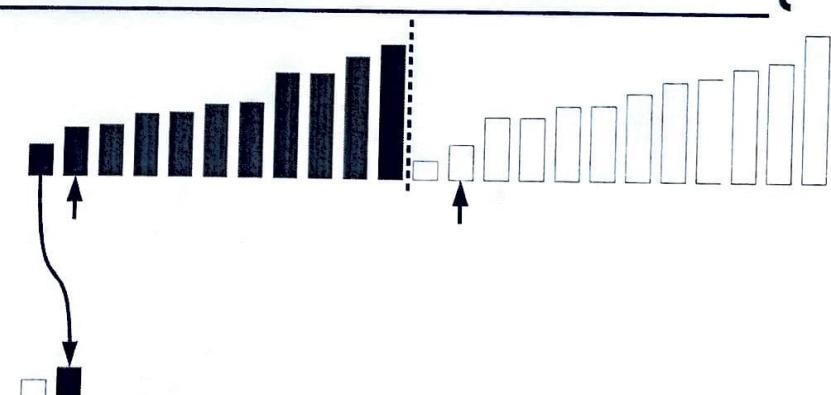


Programování - teorie, 5. ročník

105/124 Programování - teorie, 5. ročník

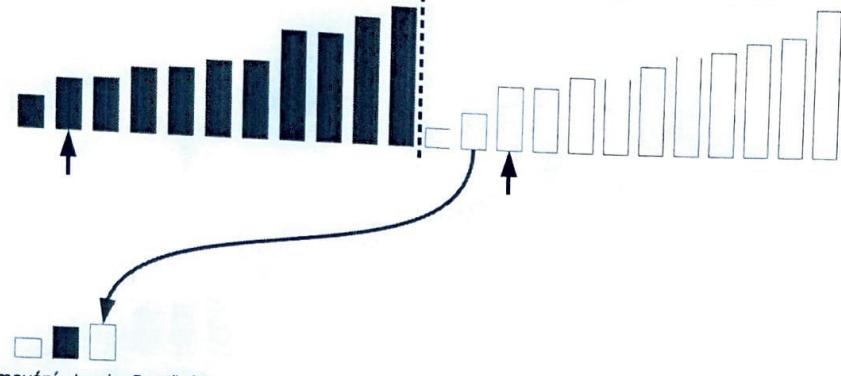
Merge sort - slévání seřazených částí

{IVT
3 IV.}



106/124

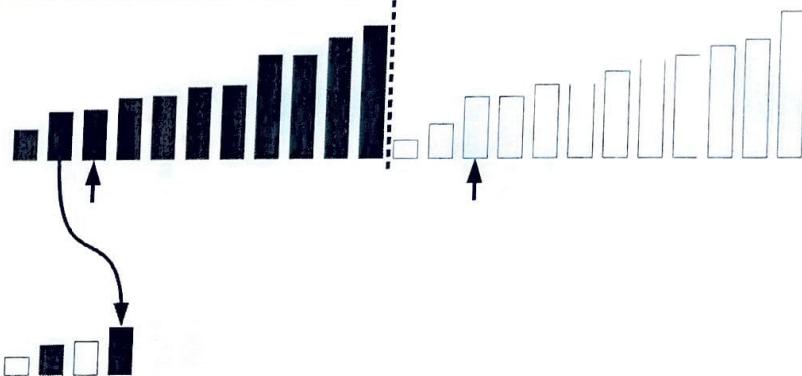
Merge sort - slévání seřazených částí $\left\{ \begin{matrix} \text{IVT} \\ \text{IV.} \\ 3 \end{matrix} \right\}$



Programování - teorie, 5. ročník

107/124

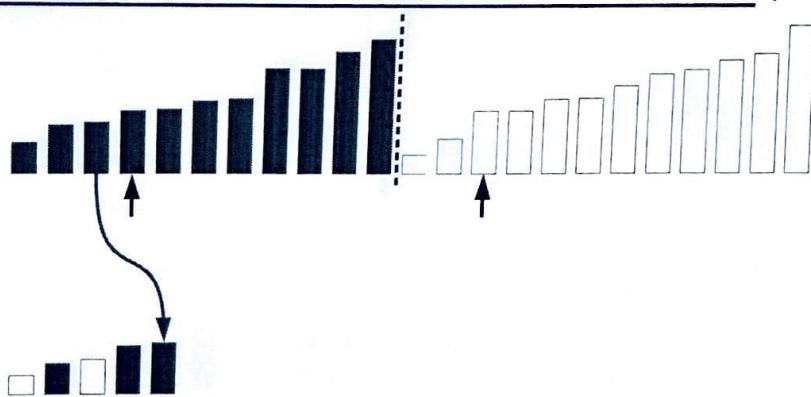
Merge sort - slévání seřazených částí $\left\{ \begin{matrix} \text{IVT} \\ \text{IV.} \\ 3 \end{matrix} \right\}$



Programování - teorie, 5. ročník

108/124

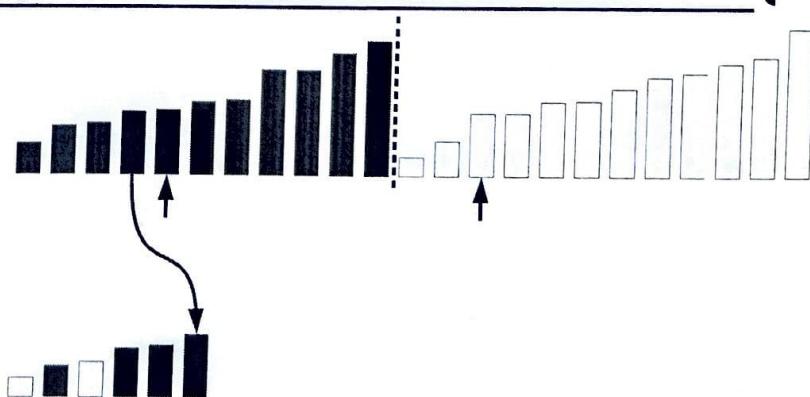
Merge sort - slévání seřazených částí $\left\{ \begin{matrix} \text{IVT} \\ \text{IV.} \\ 3 \end{matrix} \right\}$



Programování - teorie, 5. ročník

109/124

Merge sort - slévání seřazených částí $\left\{ \begin{matrix} \text{IVT} \\ \text{IV.} \\ 3 \end{matrix} \right\}$



Programování - teorie, 5. ročník

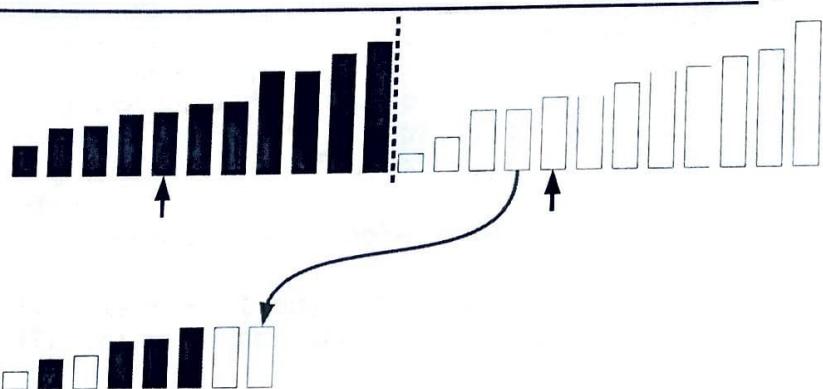
110/124

Merge sort - slévání seřazených částí $\left\{ \begin{smallmatrix} \text{IVT} \\ 3 \\ \text{IV.} \end{smallmatrix} \right\}$



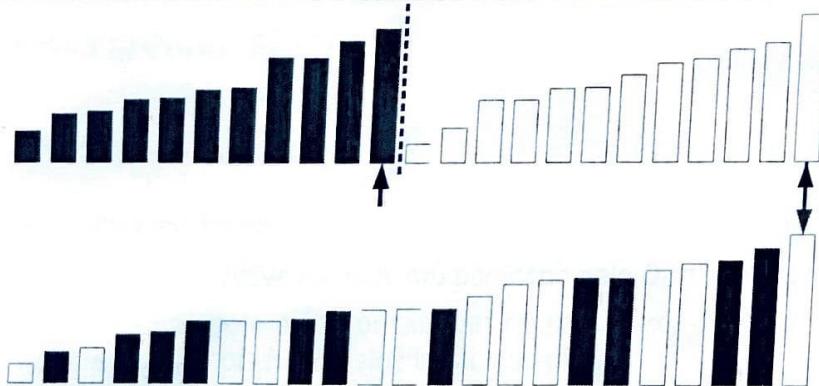
111/124

Merge sort - slévání seřazených částí $\left\{ \begin{smallmatrix} \text{IVT} \\ 3 \\ \text{IV.} \end{smallmatrix} \right\}$



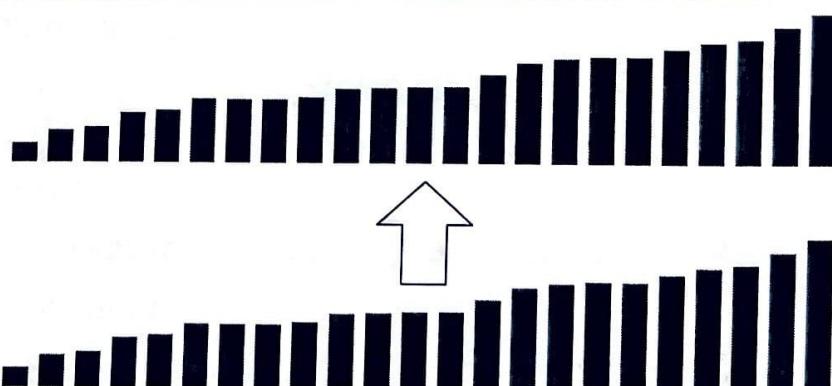
112/124

Merge sort - slévání seřazených částí $\left\{ \begin{smallmatrix} \text{IVT} \\ 3 \\ \text{IV.} \end{smallmatrix} \right\}$



113/124

Merge sort - slévání seřazených částí $\left\{ \begin{smallmatrix} \text{IVT} \\ 3 \\ \text{IV.} \end{smallmatrix} \right\}$



114/124

Merge sort - pseudokód

{IVT IV.
3}

```
algorithm MergeSort(in: pole, pompole, od, po) {
    if (od < po) { // od≥po → jednoprvkové nebo prázdné
        stred ← (od + po)/2
        MergeSort(pole, pompole, od, stred)
        MergeSort(pole, pompole, stred+1, po)
        slevani(pole, pompole, od, po)

        // kopie prvků pompole mezi indexy od a po (včetně)
        pole[od,po] ← pompole[od,po] // cyklus!
    } // if
}
```

Programování - teorie, 5. ročník

115/124

Merge sort - pseudokód

{IVT IV.
3}

```
algorithm slevani(in: pole, pompole, od, po) {
    levy ← pom ← od; stred ← (od+po)/2; pravy ← stred+1
    while (levy ≤ stred AND pravy ≤ po) {
        if (pole[levy] ≤ pole[pravy])
            pompole[pom++] ← pole[levy++]
        else
            pompole[pom++] ← pole[pravy++]
    } // while
    while (levy ≤ stred) pompole[pom++] ← pole[levy++]
    while (pravy ≤ po) pompole[pom++] ← pole[pravy++]
}
```

Programování - teorie, 5. ročník

116/124

Merge sort - vlastnosti

{IVT IV.
3}

- Časová složitost **O(n log n)**
 - Průměrná i nejhorší
 - Rekurzivní dělení pole: **O(log n)**
 - Slévání: **O(n)**
- Prostorová složitost
 - Dodatečné paměťové náklady pro pomocné pole $O_s(n)$ ⇒ není in situ.
 - Tip: pomocné pole může obsahovat pouze indexy do řazeného pole ⇒ výhoda při řazení složitějších struktur.

Programování - teorie, 5. ročník

117/124

Merge sort - vlastnosti

{IVT IV.
3}

- Není přirozený
 - Pracuje přibližně stejně dlouho pro libovolně seřazená data
- Je sekvenční
 - Pro sekvenční struktury lepší než Quick sort.
- Je stabilní
 - Slévání neprohazuje prvky se stejným klíčem.

Programování - teorie, 5. ročník

118/124

Merge sort - vlastnosti

{IVT IV.
3}

- Výhody
 - Pro některá data je výhodou, že je sekvenční a stabilní.
 - Snadno paralelizovatelný.
- Nevýhody
 - Vyžaduje dodatečné pole stejné délky jako má řazené pole.
 - Pomalejší než Quick sort, či Heap sort
 - Provádí více přesunů v paměti.

Otázky

{IVT IV.
3}

- Co znamená pojem vnitřní metody řazení?
- Jak funguje ??? sort (princip, algoritmus a náčrt kódu)? + varianty
- Jaké jsou vlastnosti jednotlivých řadících algoritmů?
- Je Selection sort stabilní? Proč? Uveďte příklad.
- Jakou má Quick sort časovou složitost? Za jakých situací?

Otázky

{IVT IV.
3}

- Jakou má Quick sort prostorovou složitost? Proč?
- Jak vybírat pivoty u algoritmu Quick sort?
- Kdy je lepší použít Merge sort místo Quick sortu?
- Je rychlejší Merge sort nebo Quick sort? Kdy a proč?
- U kterého řazení počet průchodů a počet porovnání nezávisí na pořadí prvků ve vstupním poli? Jak se tato vlastnost nazývá?

Otázky

{IVT IV.
3}

- Napište prvních 5 průchodů při řazení algoritmem [Selection | Insert | Bubble | Ripple | Skaker] Sort nad zadáným polem. Elementární kroky v rámci jednoho průchodu naznačte šipkami.

5	8	1	5	4	9	0	1	2	7	3
---	---	---	---	---	---	---	---	---	---	---

Otázky

{IVT IV.
3}

- Kolik cyklů má řazení [Selection | Insert | Bubble | Shaker] sort, k čemu tyto cykly slouží, o jaké cykly jde (for, while, do-while?) a jaké jsou jejich meze.
- Ve kterém řazení se používá zarážka? Proč se používá? Co obsahuje a kde je umístěna?
- Napište (pseudo)kód, který hledá v části pole minimum. V jakém řazení se tento algoritmus používá a jak?

Programování - teorie, 5. ročník

Otázky

{IVT IV.
3}

- V proměnné **x** je uložena hodnota některého z prvků pole **p** ležící mezi indexy **A** a **B** včetně. Napište algoritmus, který nastaví indexy **L** a **R** tak, aby **L** ukazoval na první prvek zleva, který je větší nebo roven **x** a **R** ukazoval na první prvek zprava, který je menší než **x**. V poli se pohybujte pouze mezi indexy **A** a **B**. V jakém řadícím algoritmu by tento algoritmus nalezl uplatnění?

123/124 Programování - teorie, 5. ročník

124/124