ŘADÍCÍ ALGORITMY

Ing. Karel Johanovský
SPŠ-JIA

Problém

- Vstup: Máte zadáno pole čísel, nebo čehokoliv jiného, co se dá a co má smysl řadit.
- Úkol: Máte najít algoritmus, který seřadí podle zadaného pravidla prvky v poli.

Nyní trochu podrobněji …

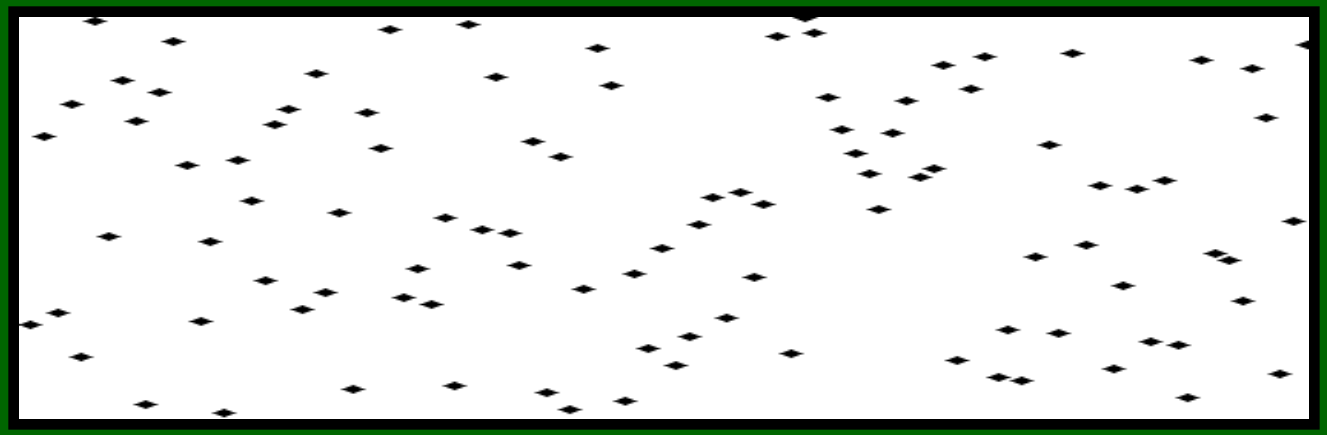
- Máme zadáno POLE, tzn. homogenní strukturu, v níž máme přístup k libovolnému prvku v konstantním čase.
- Máme najít algoritmus, který seřadí prvky podle zadaného PRAVIDLA, (u čísel obvykle velikost), ale obecně by nám měla postačit jakákoliv funkce, která pro dva prvky určí jejich pořadí.

Co nás zajímá …

- U každého algoritmu nás zajímá jeho složitost, tj. jak roste zatížení systému (trvání algoritmu), když měníme naše požadavky (rozsah vstupních dat).
- Složitost řadících algoritmů se zapisuje jako funkce, která vyjadřuje, kolik elementárních operací nad daty se musí vykonat v závislosti na počtu prvků.

BUBBLE SORT

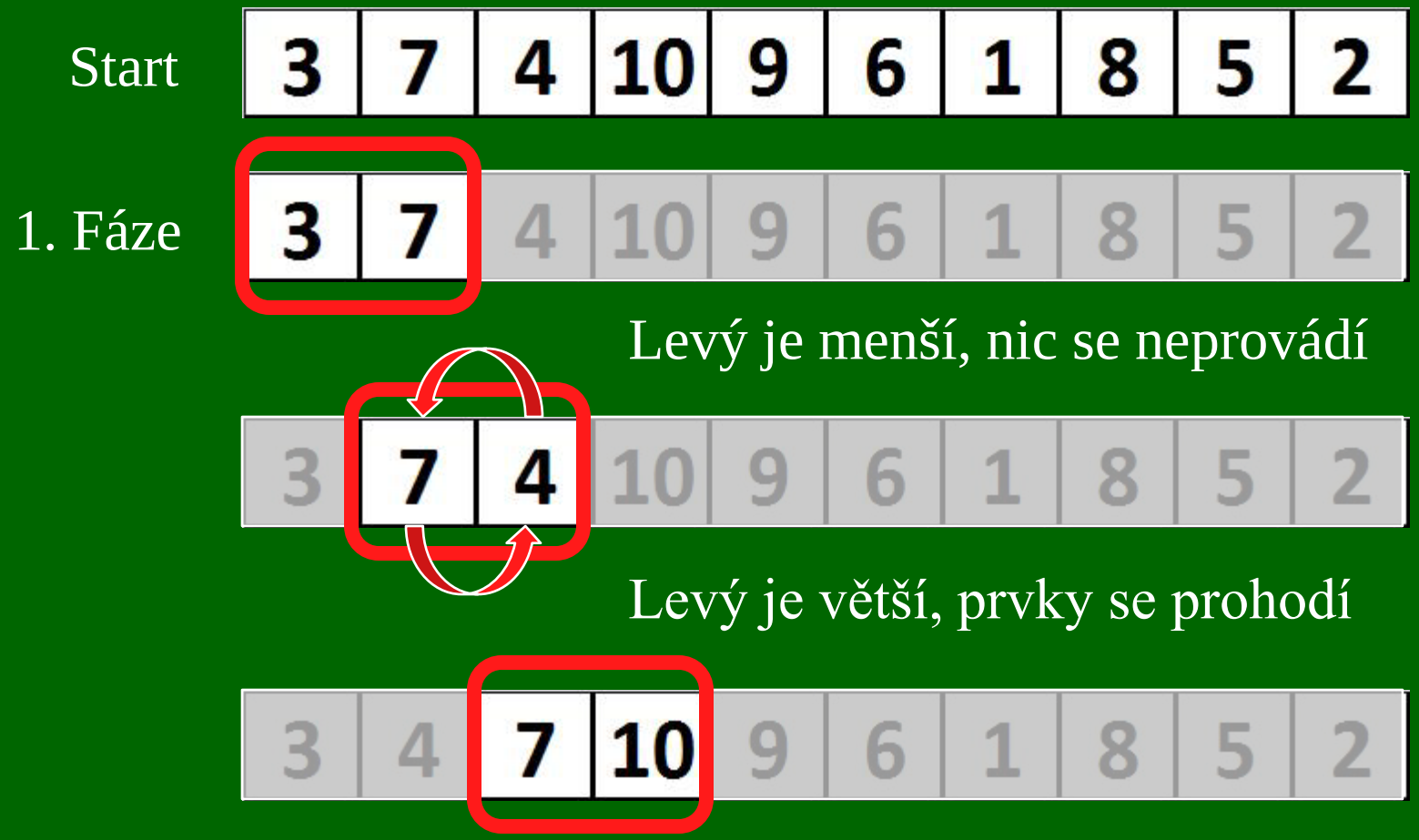
Bublinkové řazení



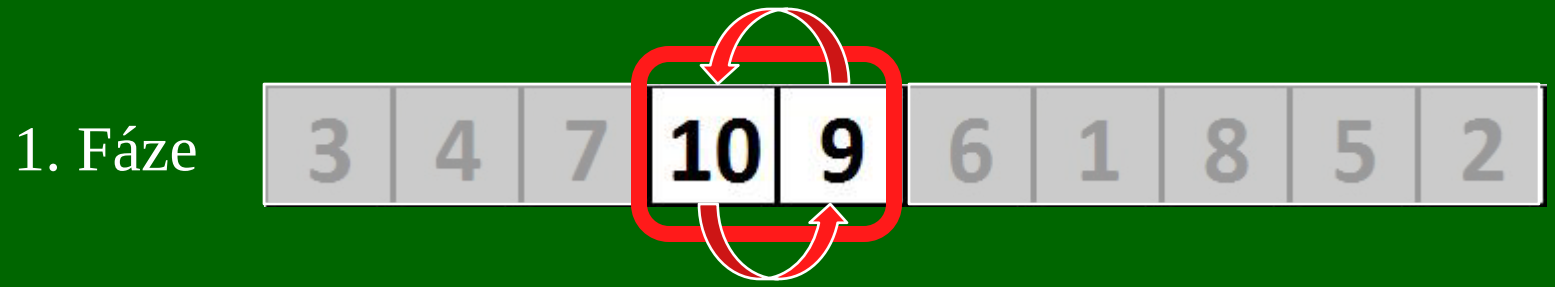
Bubble sort

- Prvním způsobem je tzv. bublinkové řazení (bubble sort).
- Princip spočívá v tom, že se pole projede od začátku a vždy se porovnají dva sousední prvky.
- Pokud se při porovnání zjistí, že je levý prvek větší než pravý, prvky se prohodí.

Bubble sort



Bubble sort



Po proběhnutí 1. fáze máme jistotu, že největší prvek je na svém místě (na konci pole).
Nyní můžeme celý postup opakovat znovu.

Bubble sort v JAVĚ


```
for (int i=0; i<(Pole.length-1); i++)
{
      for (int j=0; j<(Pole.length-1-i); j++)
      {
            if (Pole[j] > Pole[j+1])
            {
                  int pom = Pole[j];
                  Pole[j] = Pole[j+1];
                  Pole[j+1] = pom;
            }
      }
}
```

Bubble sort v C


```
for (i=0; i<(N-1); i++)
{
      for (j=0; j<(N-1-i); j++)
      {
            if (Pole[j] > Pole[j+1])
            {
                  pom = Pole[j];
                  Pole[j] = Pole[j+1];
                  Pole[j+1] = pom;
            }
      }
}
```

Bubble sort - Složitost

- Celkem porovnání:

$$(n-1)+(n-2)+\ldots+2+1=\frac{1}{2}(n^2-n)$$

- Celkem přesunů:

$$\text{Min}: 0$$

$$\text{Max}: \frac{1}{2}(n^2-n)$$

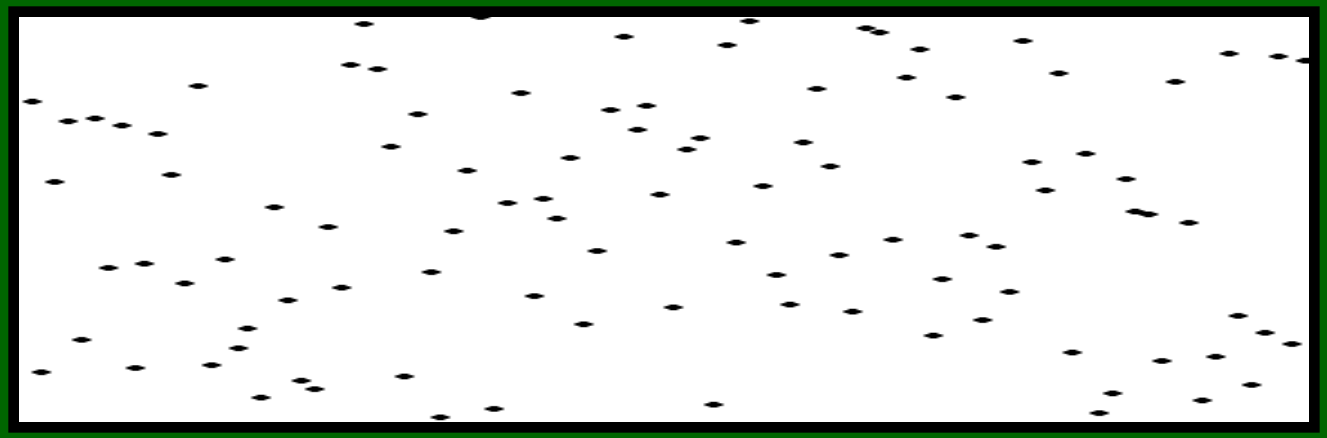
Složitost Bubble sortu je tedy přibližně: N^2

Bubble sort - Vylepšení

- Bubble sort se zarážkou
  - Když se neprovede žádná výměna, je seřazeno.
- Ripple sort
  - Pamatuji si pozici první dvojice u které došlo k výměně a v dalším průchodu začínám až na předchozí.
- Shake sort
  - Procházím pole střídavě zprava a zleva.
- Shuttle sort
  - Pokud dojde k výměně, tak vezmu ten menší a protlačuju jej na místo kam patří.

SELECT SORT

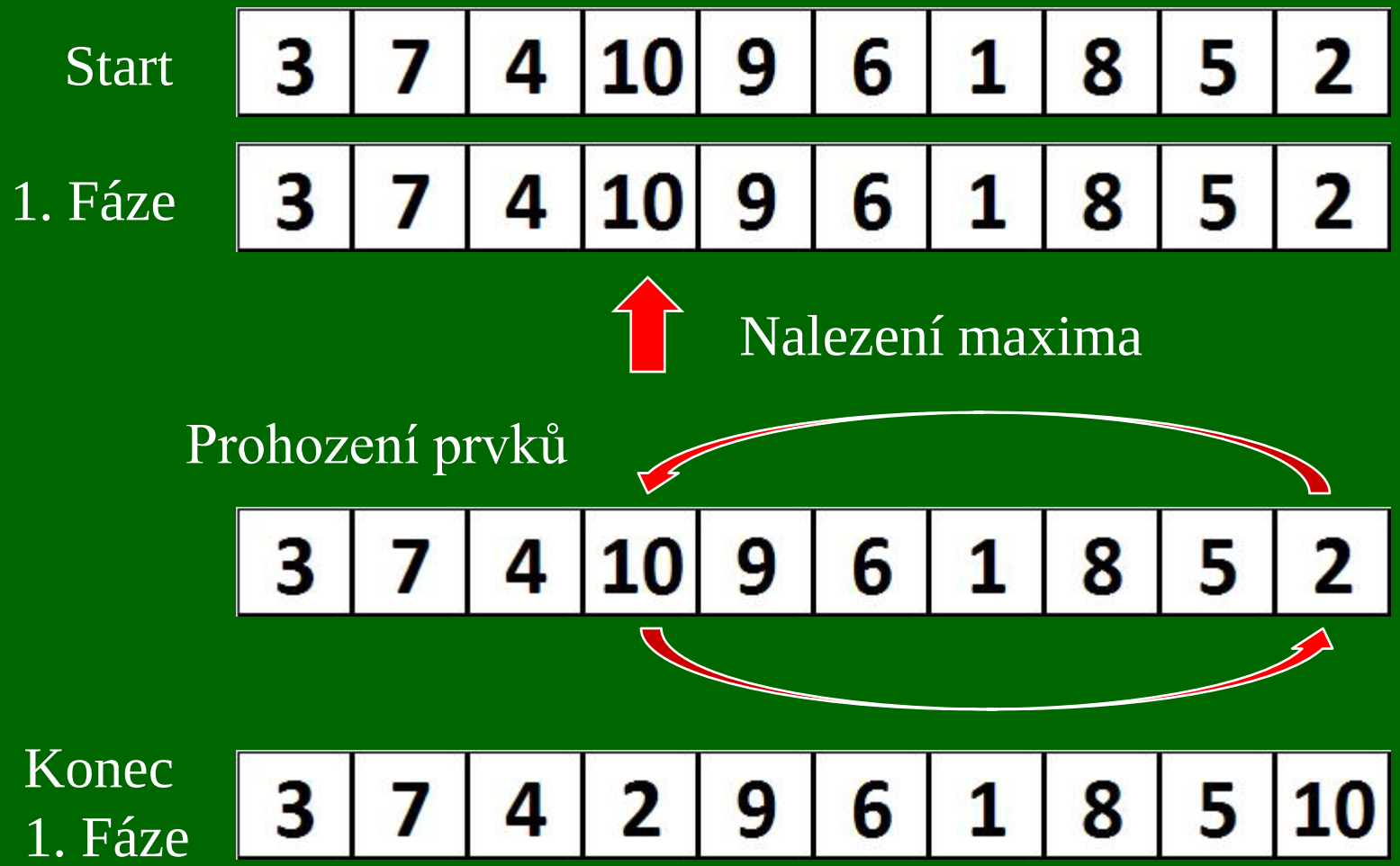
Řazení výběrem minima, nebo maxima



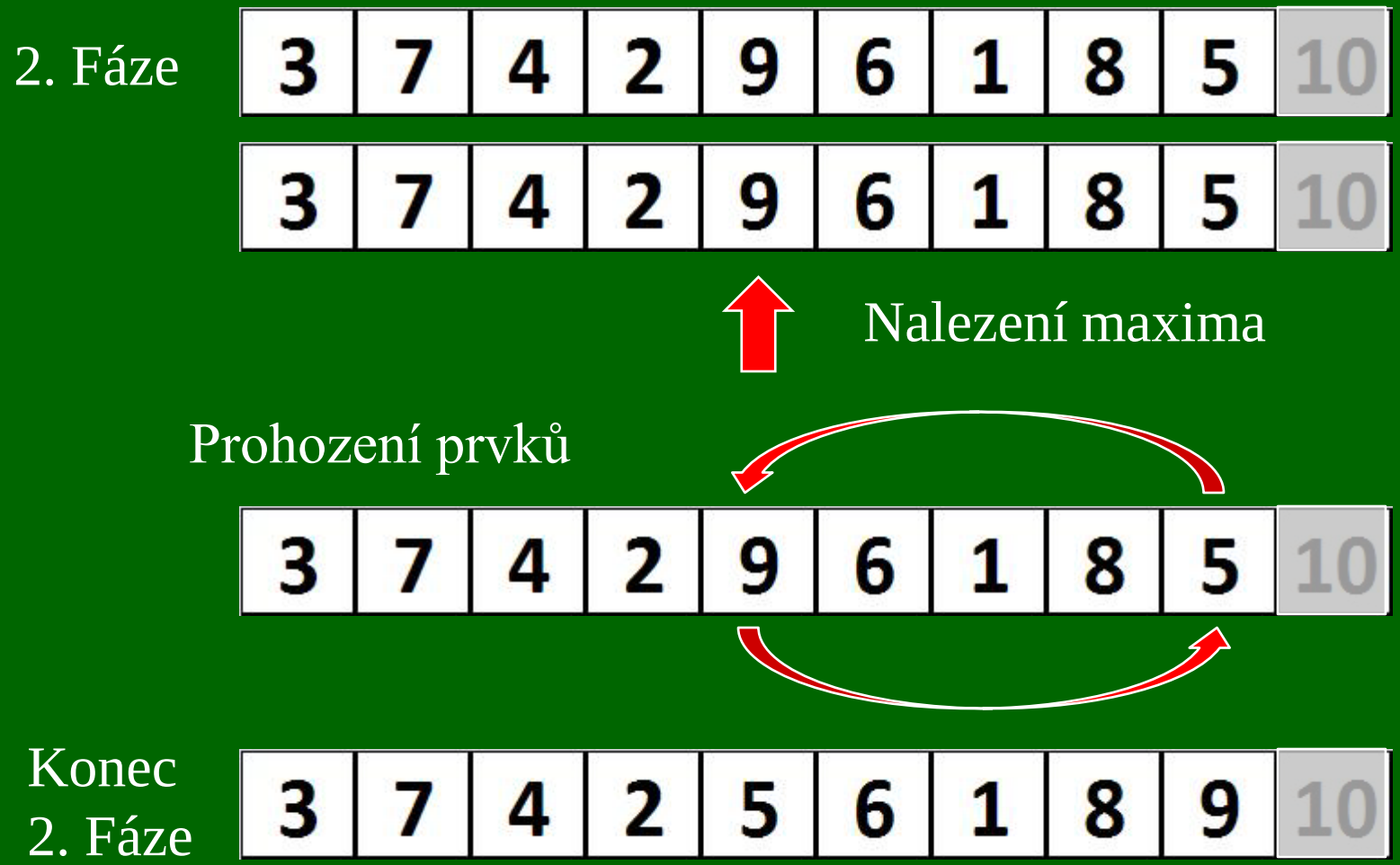
Select sort

- Druhým způsobem je tzv. řazení výběrem (select sort).
- Princip spočívá v tom, že se v poli najde buď maximum, nebo minimum a tento prvek se prohodí s prvkem na posledním, resp. prvním místě.
- Poté postup opakujeme, s vynecháním tohoto posledního / prvního prvku.

Select sort



Select sort



Select sort

- Ve stejném duchu probíhají všechny další fáze.
- Po proběhnutí každé z nich máme vždy nejméně jeden prvek na správné pozici.
- Je tedy jasné, že musí proběhnout N-1 fází, abychom setřídili celé pole.

Select sort v JAVĚ


```
for (int i = 0; i < Pole.length-1; i++)
{
      int max_pos = Pole.length-1-i;
      for (int j = 0; j < Pole.length-i; j++)
      {
            if (Pole[j] > Pole[max_pos])
            {
                  max_pos = j;
            }
      }
      int pom = Pole[Pole.length-1-i];
      Pole[Pole.length-1-i] = Pole[max_pos];
      Pole[max_pos] = pom;
}
```

Select sort v C

```
for (i = 0; i < N-1; i++)
{
      max_pos = N-1-i;
      for (j = 0; j < N-i; j++)
      {
            if (Pole[j] > Pole[max_pos])
            {
                  max_pos = j;
            }
      }
      pom = Pole[N-1-i];
      Pole[N-1-i] = Pole[max_pos];
      Pole[max_pos] = pom;
}
```

Select sort - Složitost

- Celkem porovnání:

$$(n-1)+(n-2)+\ldots+2+1=\frac{1}{2}\left(n^2-n\right)$$

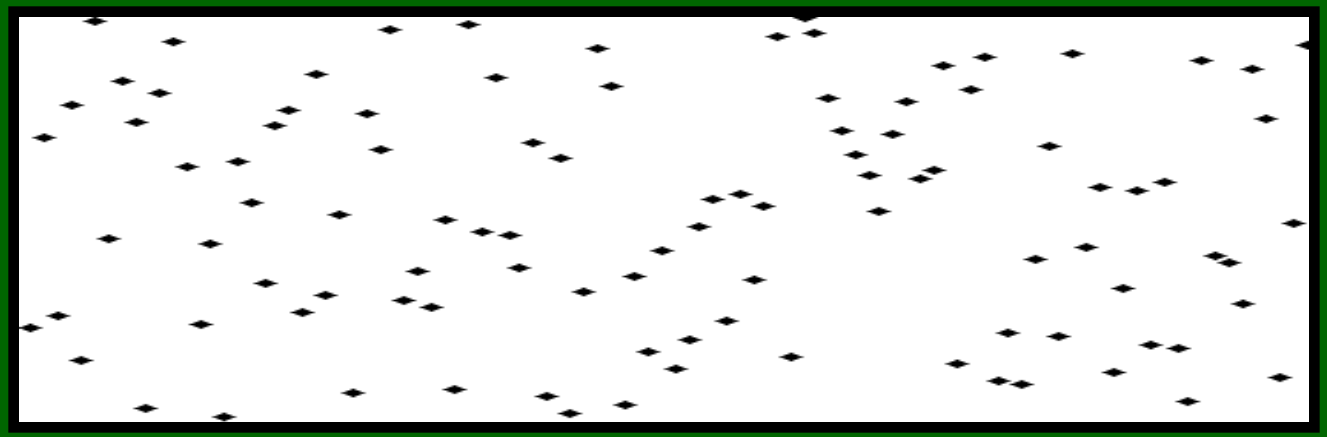
- Celkem přesunů:

$$\sum_{k=1}^{n-1} 3 = 3(n-1)$$

Složitost Select sortu je tedy přibližně: N^2

INSERT SORT

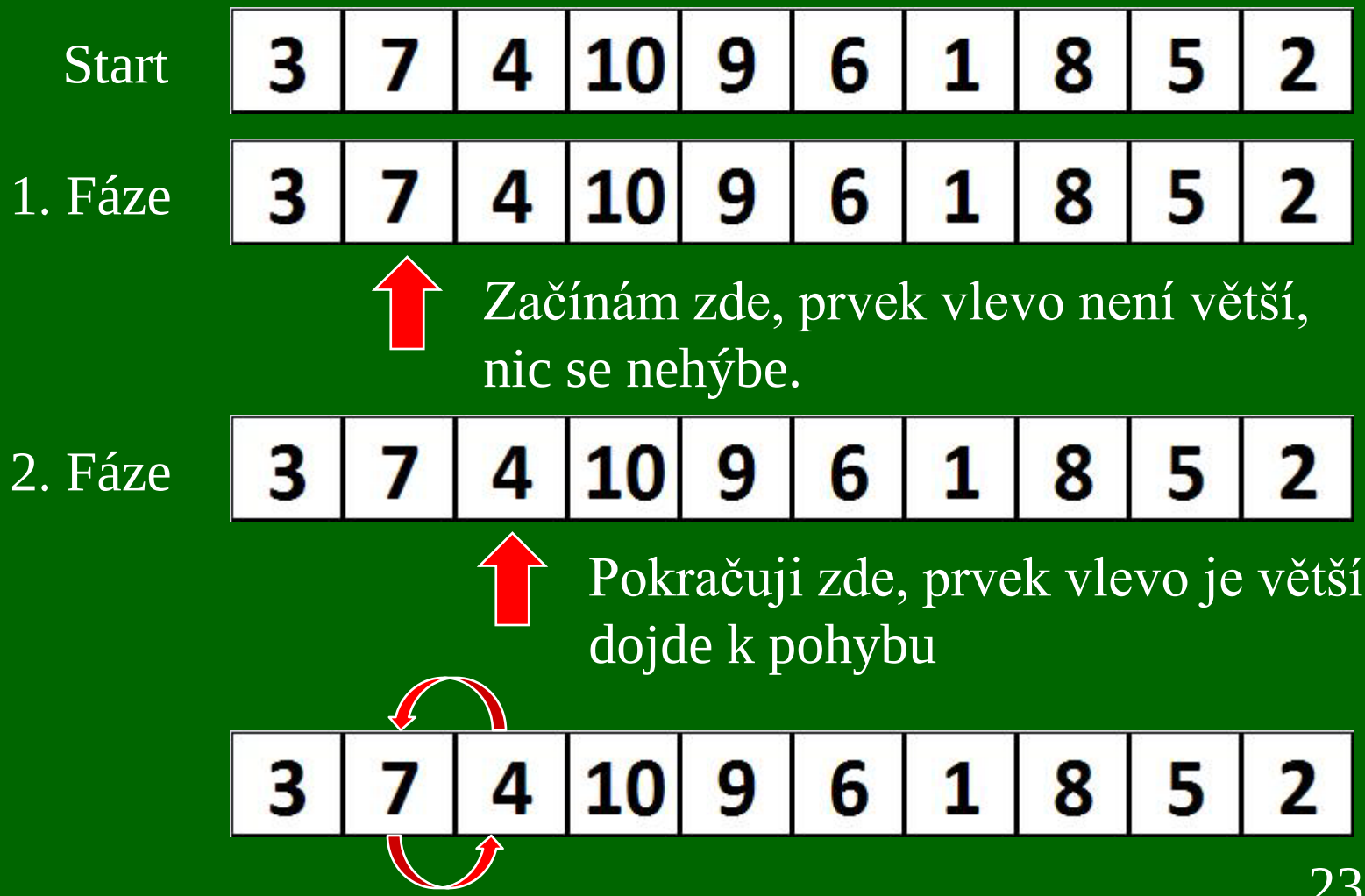
Řazení vkládáním na adekvátní pozici



Insert sort

- Třetím způsobem je řazení vkládáním (insert sort).
- Princip spočívá v tom, že se postupně prochází prvky a každý nesetříděný se zařadí na správné místo.
- K tomu je nutné mít část pole již setříděné, to se řeší tak, že první prvek se považuje za setříděný a začne se od druhého.

Insert sort



Insert sort

3. Fáze

Pokračuji zde, prvek vlevo není větší, nic se nehýbe

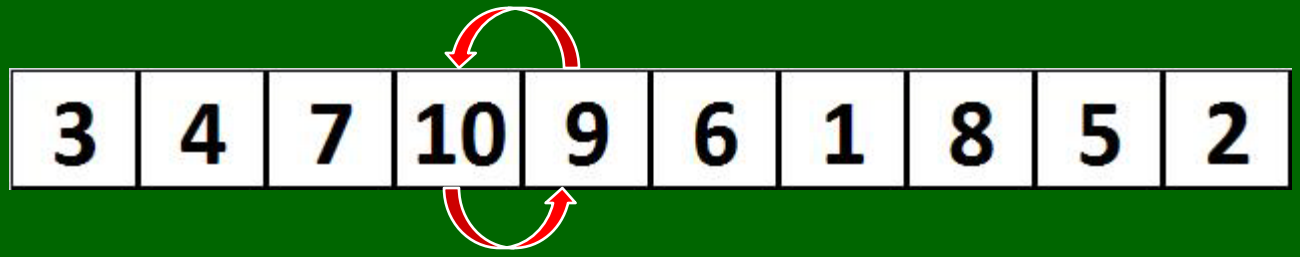
4. Fáze

Pokračuji zde, prvek vlevo není větší, nic se nehýbe

5. Fáze

Pokračuji zde, prvek vlevo je větší, dojde k pohybu

Insert sort

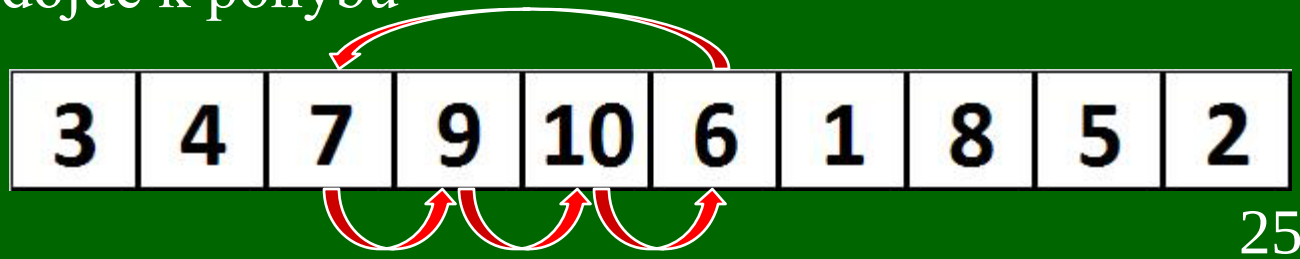


6. Fáze

Pokračuji zde, prvek vlevo není větší, nic se nehýbe

7. Fáze

Pokračuji zde, prvek vlevo je větší, dojde k pohybu



Insert sort

Konec 7. Fáze

atd…

Všechny prvky se v průběhu jednotlivých fází postupně posouvají blíže a blíže svým správným místům.

Dokud ale neproběhne všech N-1 fází nemůžeme s jistotou říci, že je posloupnost setříděná.

Insert sort v JAVĚ


```
for (int i = 1; i < Pole.length ; i++)
{
      int ins = Pole[i];
      int j = i-1;
      while((j >= 0) && (Pole[j] > ins))
      {
            Pole[j+1] = Pole[j];
            j--;
      }
      Pole[j+1] = ins;
}
```

Insert sort v C


```
for (i = 1; i < N ; i++)
{
      ins = Pole[i];
      j = i-1;
      while((j >= 0) && (Pole[j] > ins))
      {
            Pole[j+1] = Pole[j];
            j--;
      }
      Pole[j+1] = ins;
}
```

Insert sort - Složitost

- Celkem porovnání:

$$\text{Min} : n - 1$$

$$\text{Max} : \frac{n^2 - n}{2}$$

- Celkem přesunů:

$$\text{Min} : 2(n - 1)$$

$$\text{Max} : \frac{n^2 + n - 2}{2}$$

Složitost Insert sortu je tedy přibližně: N^2

Insert sort - Vylepšení

- Binární insert sort
  - Vychází z klasického insert sortu, ale využívá toho že levá půlka je již seřazena a proto mohu k vyhledání správné pozice použít bisekci.
  - Je to nejrychlejší algoritmus z kvadratických řadících algoritmů.

- Úvod
- Bubble sort
- Select sort
- Insert sort
- **Quick sort**
- Závěr

Quick sort

Sir Charles Antony Richard Hoare

C. A. R. Hoare: Quicksort. Computer Journal, Vol. 5, 1, 10-15 (1962)

Quick sort

- Posledním způsobem řazení, který je zde vysvětlen je tzv. rychlé řazení (quick sort).
- Základní myšlenkou je rozdělení řazené posloupnosti čísel na dvě přibližně stejné části.
- V jedné části jsou čísla větší a ve druhé menší, než nějaká zvolená hodnota nazývaná pivot).

Quick sort

- Pokud je tato hodnota zvolena dobře, jsou obě části přibližně stejně velké.
- Pokud budou obě části samostatně seřazeny, je seřazené i celé pole.
- Obě menší části se pak REKURZIVNĚ řadí stejným postupem.

Pozn.: Rekurze - termín je nejspíše odvozen z latinského slovesa: *recurso* (vrátit se) nebo podstatného jména: *recursus* (návrat, zpětný běh). V programování se tímto termínem označuje případ kdy funkce volá sama sebe (obvykle na jednodušší zadání).

Quick sort

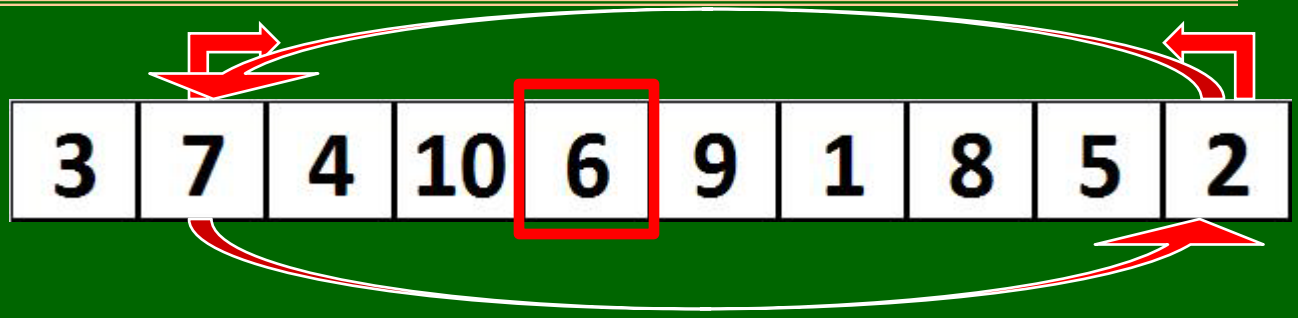
Start

Volba pivota

Posun ukazatelů

Nyní z obou konců pole vysílám proti sobě ukazatele. Levý se posunuje dokud nachází prvky menší než pivot, pravý se posunuje dokud nachází prvky větší než pivot.

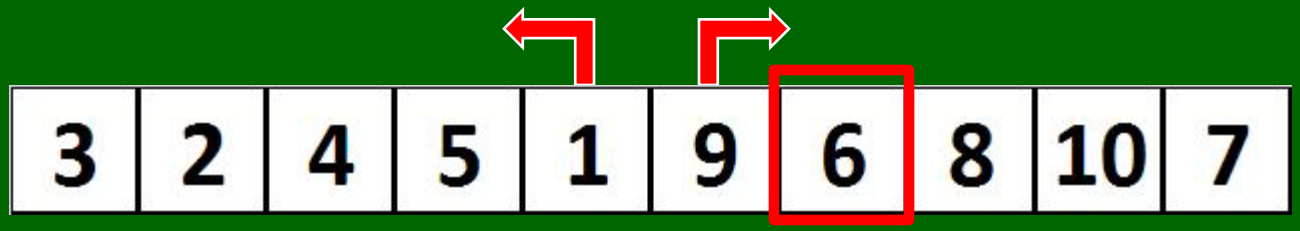
Quick sort



Jakmile se oba ukazatele zastaví, a platí se nepřekřížily, tzn. pozice lévého < pozice pravého, prvky na které ukazují se prohodí a oba ukazatele se posunou na další prvek.

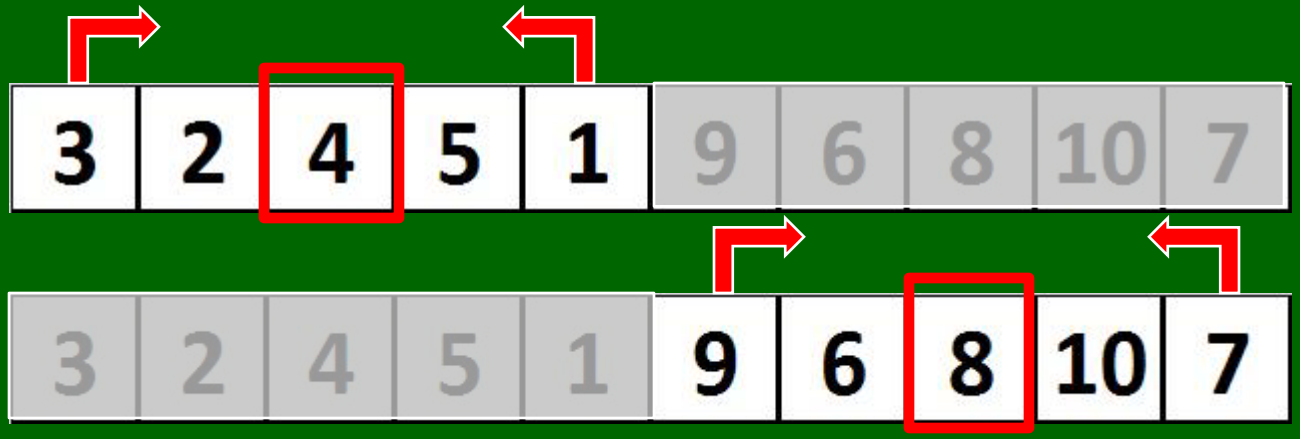
Nyní se znovu opakuje posun obou ukazatelů a hledání a prohazování prvků větších než pivot, s prvky menšími než pivot.

Quick sort



Po prvním průchodu máme v levé části pole prvky menší než pivot a v pravé části prvky větší než pivot.
Poté, pokud jsou obě nově vzniklé části nenulové, zavoláme na ně rekurzivně opět stejný postup.

2. Fáze



Quick sort v JAVĚ


```
public static void quicksort(int l, int r, int[] Pole)
{
        int i = l;
        int j = r;
        int pivot = Pole[((l + r) / 2)];
        do
        {
                while(Pole[i] < pivot) i++;
                while(Pole[j] > pivot) j--;
                if (i <= j)
                {
                        int pom = Pole[i];
                        Pole[i] = Pole[j];
                        Pole[j] = pom;
                        i++;
                        j--;
                }
        }
        while(i < j);
        if ((j - l) > 0) quicksort(l, j, Pole);
        if ((r - i) > 0) quicksort(i, r, Pole);
}
```

Quick sort v C


```
void quicksort(int l, int r, int* Pole)
{
        int i = l;
        int j = r;
        int pivot = Pole[((l + r) / 2)];
        do
        {
                while(Pole[i] < pivot) i++;
                while(Pole[j] > pivot) j--;
                if (i <= j)
                {
                        pom = Pole[i];
                        Pole[i] = Pole[j];
                        Pole[j] = pom;
                        i++;
                        j--;
                }
        }
        while(i < j);
        if ((j - l) > 0) quicksort(l, j, Pole);
        if ((r - i) > 0) quicksort(i, r, Pole);
}
```

Quick sort - Složitost

- Celkem přesunů a porovnání:

$$\text{Min} : n \cdot (\log_2(n))$$
$$\text{Avg} : n \cdot (\log_2(n))$$
$$\text{Max} : n^2$$

Složitost Quick sortu je sice stále v tom nejhorším případě rovna N^2, ale průměrná složitost objevující se na běžných datech je N.log(N).

Quick sort – další varianta

- Quick sort – varianta Lomuto
  - Autor: Nico Lomuto (1984)
  - Postupně umisťuje prvky menší jak pivot do levé půlky pole, která se tímto zvětšuje.
  - Vykazuje méně porovnání, ale více přesunů než Hoareho varianta.
  - Opět zavisí na volbě pivota => může degradovat na N^2.

Ostatní řadící algoritmy

- Counting sort
  - Využívá znalosti minima a maxima v poli z čehož sestaví pole četností jednotlivých hodnot.
  - Pole četností se přepočítá na pole posledních indexů a pomocí toho je poté jedním průchodem sestaveno seřazené pole.
  - Nevýhody: potřebuje další pole, pouze diskrétní hodnoty.
- Heap sort
  - Využívá bin. haldu, což je struktura, která rekurzivně splňuje vlastnost, že každý rodič, je vyšší než oba jeho potomci.
  - Velmi efektivní, zaručena složitost N.log(N).

Ostatní řadící algoritmy

- Merge sort
  - Pracuje na bázi slévání již seřazených částí pole za pomoci dodatečného pole velikosti N.
  - Nejprve je pole rekurzivně půleno až na jednotlivé prvky a poté při návratu z rekurze se slévají jednotlivá pole do výsledného seřazeného.
- Shell sort
  - Podobný jako insert sort, ale neporovnává prvky vedle sebe, ale s určitou mezerou, která se postupně zmenšuje.
  - Jde o to jak vhodně zvolit mezeru (2.2)

Řadící algoritmy – Porovnání čas [s]

Počet prvků	10	100	1000	10000	100000	1000000	10000000
Bubble sort	0,000006	0,000232	0,023946	0,938401	89,42081	9164,938	neměřeno
Bubble with stop	0,000003	0,000087	0,008912	0,889644	90,12325	9264,003	neměřeno
Ripple sort	0,000002	0,000094	0,009115	0,925881	92,65036	9772,114	neměřeno
Shake sort	0,000003	0,000089	0,008305	0,820023	81,96399	8334,661	neměřeno
Shuttle sort	0,000003	0,000053	0,004941	0,514714	48,92581	5027,231	neměřeno
Select sort	0,000003	0,000059	0,004967	0,491035	49,28691	5002,091	neměřeno
Insert sort	0,000003	0,000033	0,002842	0,280647	28,73029	2969,268	neměřeno
Insert sort - Binary	0,000003	0,000035	0,002334	0,213047	23,01424	2439,217	neměřeno
Quick sort - Hoare	0,000002	0,000014	0,000171	0,002114	0,026368	0,300333	3,415378
Quick sort - Lomuto	0,000003	0,000018	0,000215	0,002829	0,038349	0,433496	5,037954
Counting sort	0,000005	0,000011	0,000069	0,000701	0,014526	0,244312	2,945509
Heap sort	0,000004	0,000019	0,000243	0,003289	0,042771	0,731834	12,74019
Merge sort	0,000003	0,000023	0,000289	0,003725	0,052477	0,551276	6,378464
Shell sort	0,000003	0,000017	0,000261	0,003842	0,064737	0,647795	7,978711

Zdroje a odkazy

[1] – Datové struktury a algoritmy – přednášky ČVUT

[2] - http://en.wikipedia.org/wiki/Bubble_sort

[3] - http://en.wikipedia.org/wiki/Selection_sort

[4] - http://en.wikipedia.org/wiki/Insertion_sort

[5] - http://en.wikipedia.org/wiki/Quick_sort

[6] - http://en.wikipedia.org/wiki/C._A._R._Hoare

[7] – http://www.algoritmy.net

DĚKUJI ZA POZORNOST