

ŘADÍCÍ ALGORITMY

Karel Johanovský
SPŠ-JIA

Problém

- Úvod

- Bubble sort
- Select sort
- Insert sort
- Quick sort
- Závěr

- 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 ...

- Úvod

- Bubble sort
- Select sort
- Insert sort
- Quick sort
- Závěr

- 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 ...

- Úvod

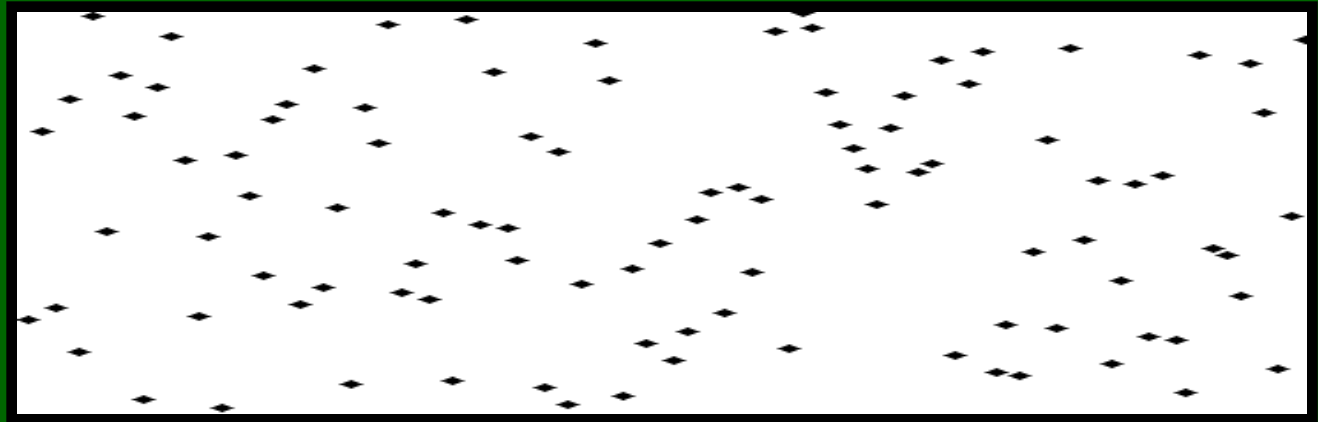
- Bubble sort
- Select sort
- Insert sort
- Quick sort
- Závěr

- 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.

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

BUBBLE SORT

Bublínkové řazení



Bubble sort

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

- 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

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

Start

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

1. Fáze

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

Levý je menší, nic se neprovádí

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

Levý je větší, prvky se prohodí

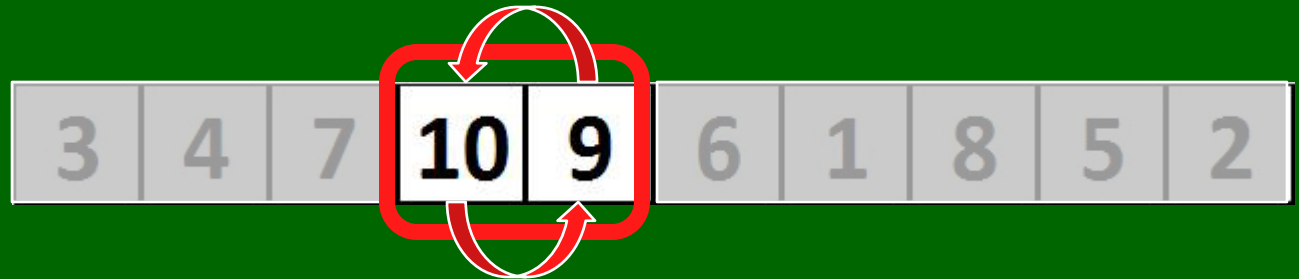
3	4	7	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

Levý je menší, nic se neprovádí

Bubble sort

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

1. Fáze



Konec

1. Fáze



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

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

```
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

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

```
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

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

- Celkem porovnání:

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

- Celkem přesunů:

Min : 0

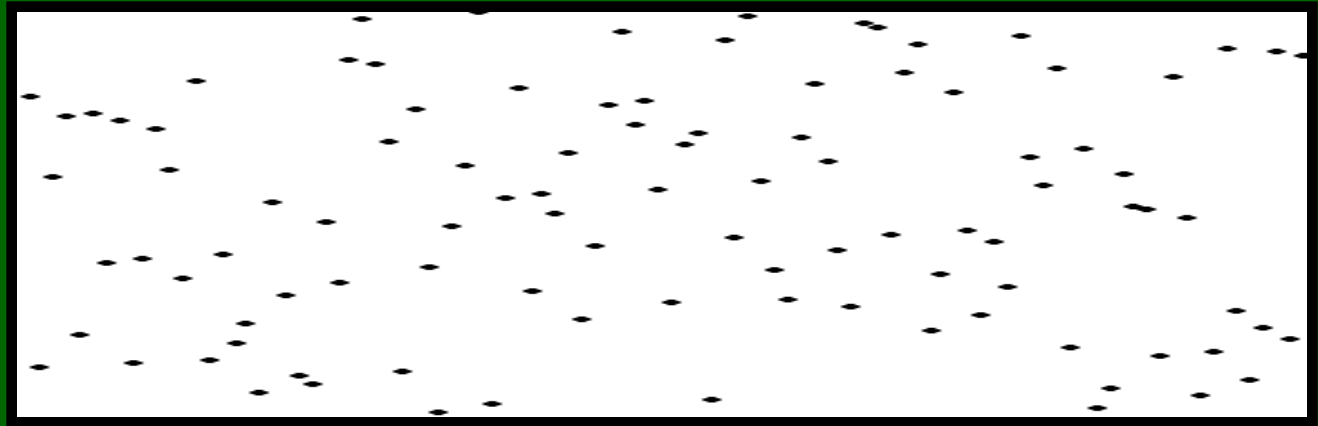
$$\text{Max} : \frac{1}{2}(n^2 - n)$$

Složitost Bubble sortu
je tedy přibližně: N^2

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

SELECT SORT

Řazení výběrem minima, nebo maxima



Select sort

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

- 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

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

Start

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

1. Fáze

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Nalezení maxima

Prohození prvků

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Konec

1. Fáze

3	7	4	2	9	6	1	8	5	10
---	---	---	---	---	---	---	---	---	----

Select sort

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

2. Fáze

3	7	4	2	9	6	1	8	5	10
---	---	---	---	---	---	---	---	---	----

3	7	4	2	9	6	1	8	5	10
---	---	---	---	---	---	---	---	---	----



Nalezení maxima

Prohození prvků

3	7	4	2	9	6	1	8	5	10
---	---	---	---	---	---	---	---	---	----



Konec
2. Fáze

3	7	4	2	5	6	1	8	9	10
---	---	---	---	---	---	---	---	---	----

Select sort

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

- 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

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

```
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

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

```
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

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

- Celkem porovnání:

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

- Celkem přesunů:

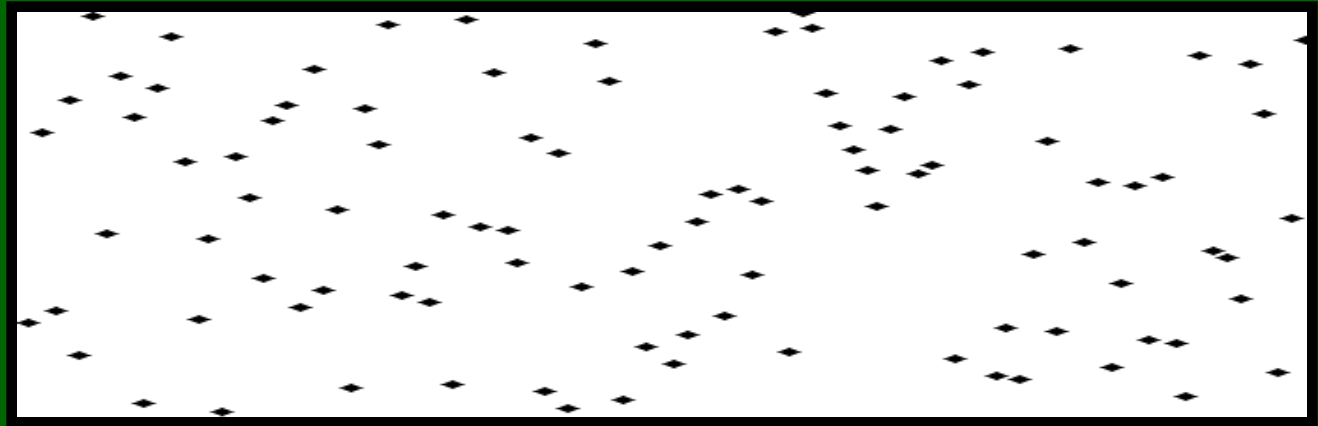
$$\sum_{k=1}^{n-1} 3 = 3(n-1)$$

Složitost Select sortu
je tedy přibližně: N^2

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

INSERT SORT

Řazení vkládáním na adekvátní pozici



Insert sort

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

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

Insert sort

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

Start

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

1. Fáze

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Začínám zde, prvek vlevo není větší,
nic se nehýbe.

2. Fáze

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Pokračuji zde, prvek vlevo je větší,
dojde k pohybu

3	7	4	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

Insert sort

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

3. Fáze

3	4	7	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Pokračuji zde, prvek vlevo není větší, nic se nehýbe

4. Fáze

3	4	7	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---



Pokračuji zde, prvek vlevo není větší, nic se nehýbe

5. Fáze

3	4	7	10	9	6	1	8	5	2
---	---	---	----	---	---	---	---	---	---

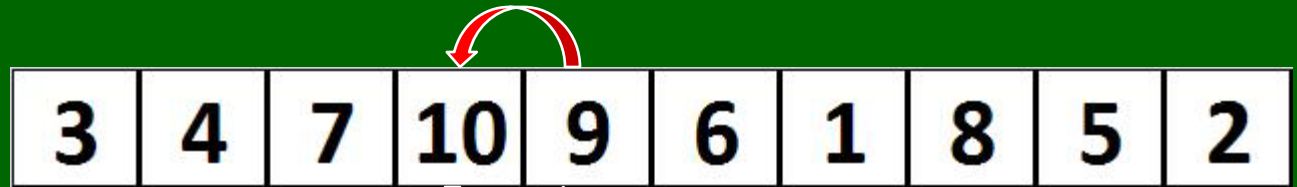


Pokračuji zde, prvek vlevo je větší, dojde k pohybu

Insert sort

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

6. Fáze



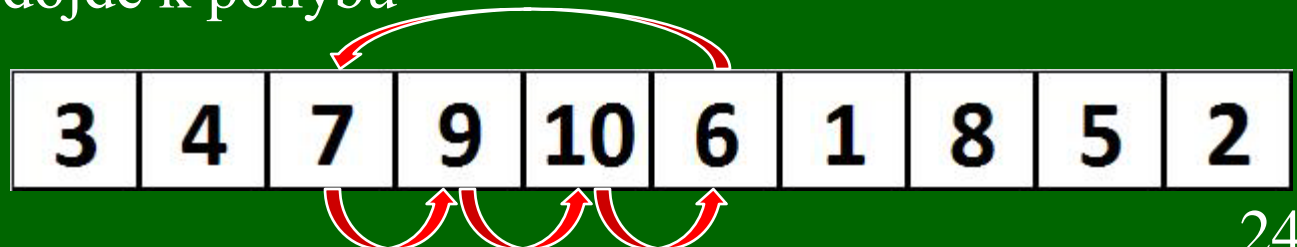
Pokračuji zde, prvek vlevo není větší, nic se nehýbe



7. Fáze



Pokračuji zde, prvek vlevo je větší, dojde k pohybu



Insert sort

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

Konec
7. Fáze

3	4	6	7	9	10	1	8	5	2
---	---	---	---	---	----	---	---	---	---

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

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

```
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

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

```
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

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

- Celkem porovnání:

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

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

- Celkem přesunů:

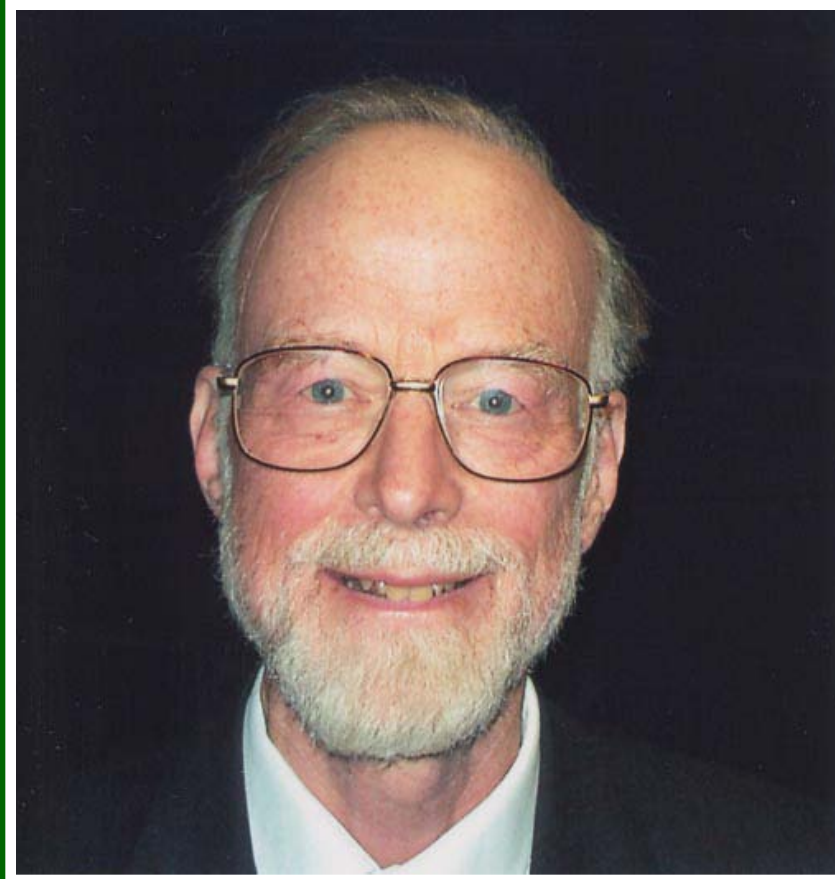
$$\text{Min} : 2(n - 1)$$

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

Složitost Insert sortu je
tedy přibližně: N^2

Quick sort

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



Sir Charles Antony Richard Hoare

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

Quick sort

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

- 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

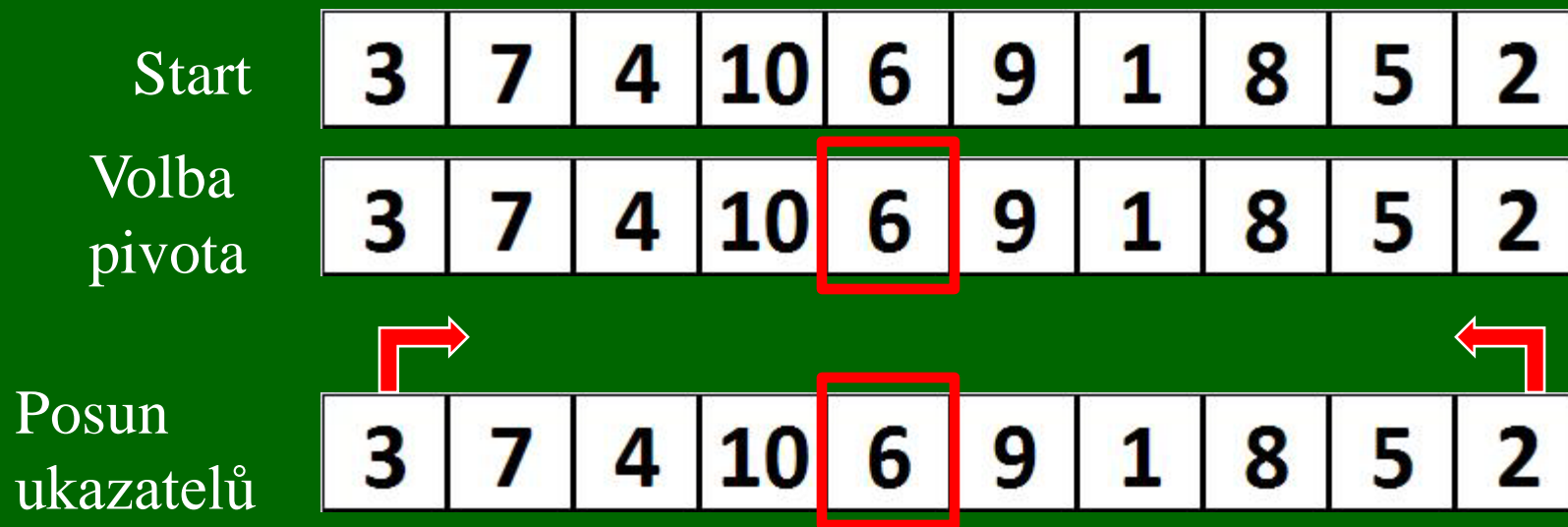
- Úvod
- Bubble sort
- Select sort
- Insert sort
- Quick sort
- Závěr

- 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

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

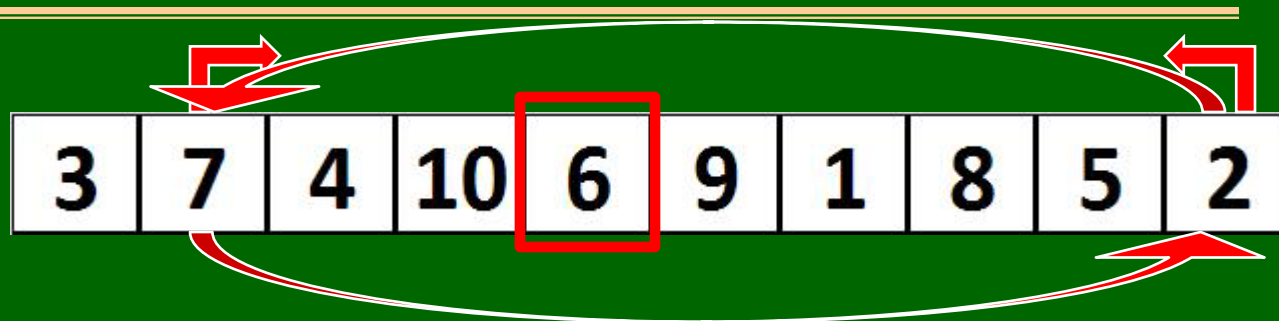


Nyní z obou konců pole vysílám proti sobě ukazatele. Levý se posunuje dokud nachází prvky menší než pivot, pravý se posunuje dokud nachází prvky větší než pivot.



Quick sort

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



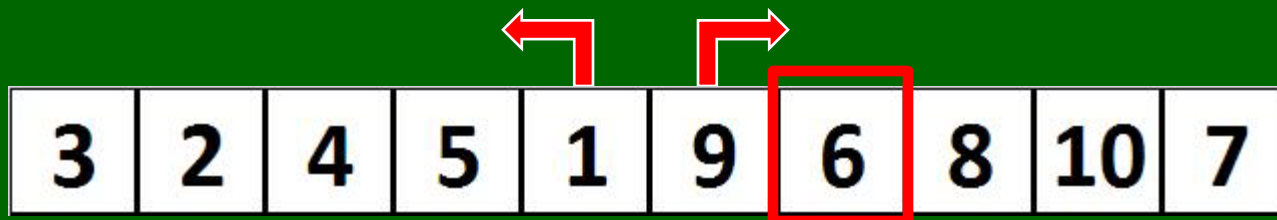
Jakmile se oba ukazatele zastaví, a platí se nepřekřížily, tzn. pozice levého < pozice pravého, prvky na které ukazují se prohodí a oba ukazatele se posunou na další prvek.



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

Quick sort

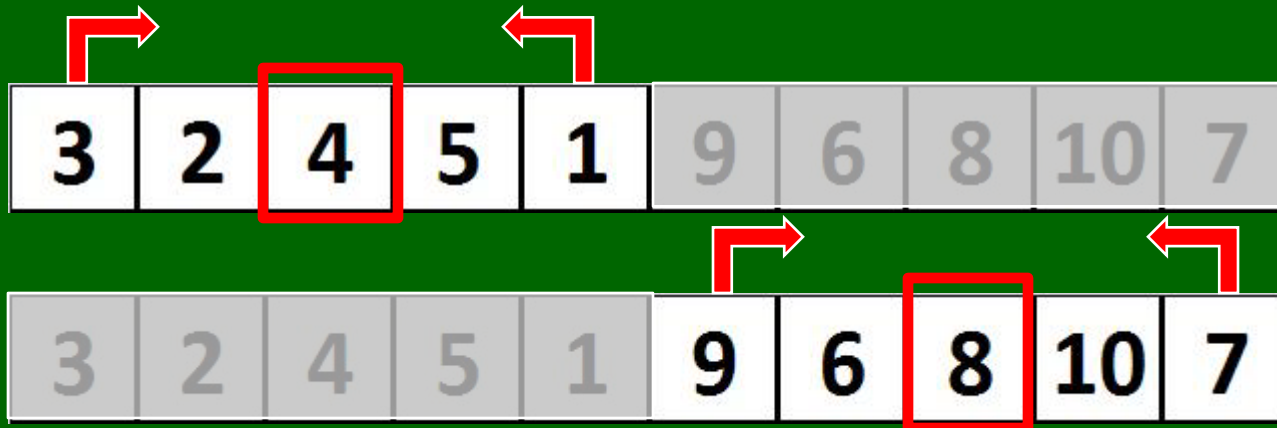
- Úvod
- Bubble sort
- Select sort
- Insert sort
- Quick sort
- Závěr



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

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

```
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

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

```
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

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

- 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 $N \cdot \log(N)$.

Řadící algoritmy – porovnání efektivity

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

N	N^2	$N \cdot \log(N)$	$\frac{N^2}{N \cdot \log(N)}$	Zpomalení (1~1s)
1	1	0,0	-	-
10	100	33,2	3,0	3 sec
100	10000	664,4	15,1	15 sec
1000	1000000	9965,8	100,3	1 min 40 sec
10000	100000000	132877,1	752,6	12 min 33 sec
100000	10000000000	1660964,0	6020,6	1 hod 40 min 21 sec
1000000	1000000000000	19931568,6	50171,7	13 hod 56 min 12 sec
10000000	100000000000000	232534966,6	430042,9	4 dny 23 hod 27 min 23 sec

Zdroje a odkazy

- Úvod
- Bubble sort
- Select sort
- Insert sort
- Quick sort
- [Závěr](#)

[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

DĚKUJI ZA POZORNOST