Vyhledání extrému v poli

- použito v algoritmech řazení
- hledání maxima

```
int[] pole = new int[pocet]
int max = pole[0];
int id;
for(int i =1; i< pole.Length; i++) {      // nikoli 0
      if (Pole[i] > max) {
            max = pole[i];
            id = i;
      }
}
```

počátek algoritmu: první prvek = maximum

nové maximum

- Minimum = stejný princip

Algoritmy řazení

Úvod

- jedna ze základních činností při práci s daty (až 25% času)

Základní terminologie

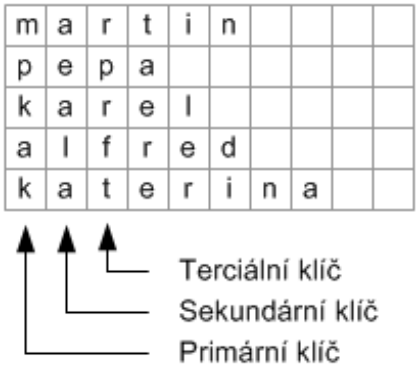
- **sort**
  - třídit, roztřiďovat, rozdělovat, zařadit (podle druhů)
  - uspořádat, seřadit
- **řazení** = přeskupování prvků množiny tak, aby vytvořily určitou posloupnost, ve které platí mezi prvky vztah „menší – větší“
  - „seřaď studenty podle data narození“
- **třídění** = činnost; prvky z velké množiny jsou rozdělovány do menších množin (podskupin) podle nějaké společné vlastnosti
  - „roztřiď studenty podle data narození do 10 skupin“
- řazení a třídění se běžně zaměňuje, obojí = „přeskupit posloupnost“
- **směr řazení**
  - vzestupně → `prvek[i] <= Prvek[i+1]` pro všechna i
  - sestupně → `prvek[i] >= Prvek[i+1]` pro všechna i
- **klíč**
  - = položka záznamu, podle které se řadí
  - při řazení neelementárních prvků („záznam“ = datová struktura s více (různými) položkami – více později)

věk:	21	13	89	4
Jméno:	Alois	Pepa	Jirka	Lenka
hmotnost:	88,6	68,2	110	30,8

- příklad třídění, klíč = věk

4	13	21	89
Lenka	Pepa	Alois	Jirka
30,8	68,2	88,6	110

1) primární, sekundární, terciární řazení – položky jsou seřazeny podle primárního klíče, vzniklé podskupiny podle sekundárního atd.



- stabilita třídění = metoda neporuší integritu jednoho záznamu

- různé metody řazení – různé rychlosti, paměťové nároky, vhodné pro různě velké množiny prvků
- poznámky:
  - dále diskutované algoritmy → nad polem:

```
int[] pole = {3,89,1,55,12};
```

  - efektivita algoritmů diskutována později
  - všechny algoritmy mají několik variant, lze je vylepšit

Řazení přímým výběrem – Select sort

- též řazení s výběrem mezního prvku
- Princip:
  1. V poli nalezneme prvek s nejmenší hodnotou a zapamatujeme si jeho pořadí
  2. Vyměníme tento prvek s prvkem na prvním místě
  3. Postupně opakujeme body 1. a 2. nad zbývající, nesetříděnou částí pole
- Průběh

```
3,89,1,55,12          1,89,3,55,12
1,89,3,55,12          1,3,89,55,12
1,3,89,55,12          1,3,12,55,89
1,3,12,89,55          1,3,12,55,89
1,3,12,55,89
```

- řešení v C#:

```
static void SelectSort(int[] pole)
{
	int min, temp;

	for (int i = 0; i < pole.Length; i++)
	{
		min = i;	// pozice min prvku
		// nalezeni minima
		for (int j = i + 1; j < pole.Length; j++)
		{
			if (pole[j] < pole[min])
			{
				min = j;
			}
		}
		// prohozeni prvku
		temp = pole[i];
		pole[i] = pole[min];
		pole[min] = temp;
	}
}
```

Řazení přímým vkládáním – Insert sort

- postup je používán i v běžném životě → řazení karet
- Princip:
  1. máme setříděno `k` prvků
  2. pro `(k+1)`. prvek nalezneme místo v již setříděné posloupnosti → zbytek posloupnosti se odsune
  3. Postupně opakujeme body 1. a 2. nad zbývající, nesetříděnou částí posloupnosti
- Průběh

// pred tridenim	po trideni
3,89,1,55,12	3,89,1,55,12
3,89,1,55,12	1,3,89,55,12
1,3,89,55,12	1,3,55,89,12
1,3,55,89,12	1,3,12,55,89

- poznámka: Posunutí karet → snadné; posunutí pole = cyklus → trvá dlouho

- řešení v C#:

```
static void InsertSort(int[] pole)
{
      int j, temp;

      for (int i = 1; i < pole.Length; i++)
      {
            temp = pole[i];
            j = i;
            while ((pole[j - 1] > temp) && (j > 0))
            {
                  pole[j] = pole[j - 1];
                  j--;
            }
            pole[j] = temp;
      }
}
```

Řazení zaměňováním – Bubble sort

- modifikace Select sort
- Princip:
  1. Porovnáváme vždy dva sousední prvky směrem od konce posloupnosti
  2. Pokud `Pole[i+1] < Pole[i]` → prohodíme je; konec na prvním prvku
     - nejmenší prvek → na první pozici – „probublá“ posloupností
     - částečně se setřídí posloupnost
  3. Postupně opakujeme body 1. a 2. → konec u druhého prvku v pořadí atd.
- Průběh

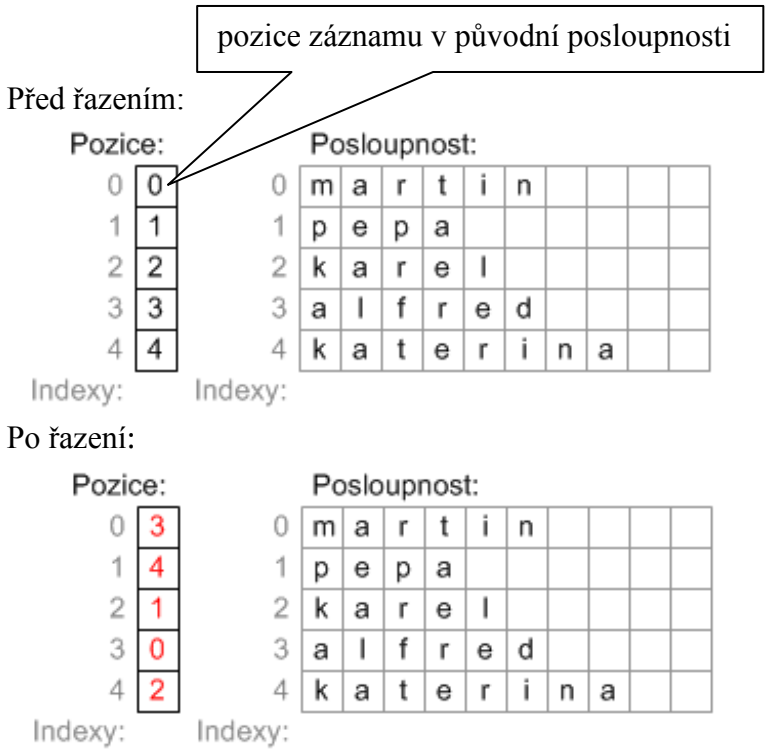
```
// porovnani          // prohozeni
3,89,1,55,12          3,89,1,12,55
3,89,1,12,55          3,89,1,12,55
3,89,1,12,55          3,1,89,12,55
3,1,89,12,55          1,3,89,12,55      // probublala 1
1,3,89,12,55          1,3,89,12,55
1,3,89,12,55          1,3,12,89,55
1,3,12,89,55          1,3,12,89,55      // probublala 3
1,3,12,89,55          1,3,12,55,89
1,3,12,55,89          1,3,12,55,89
1,3,12,55,89          1,3,12,55,89      // probublala 12
1,3,12,55,89
```

- praktické řešení viz cvičení – umět ale i na zkoušku!

Závěrečné poznámky

- další metody – Quick sort, Merge sort, Radix sort…
  - pro cca 13 a více položek → Quick sort efektivnější než zmíněné metody
  - pro < 13 → Quick sort horší

- Složité záznamy fyzicky netřídíme



Algoritmy vyhledávání

Úvod

- vyhledávání = nachází se hledaný prvek v posloupnosti? Výstup:
  - nalezeno/nenalezeno (`bool`)
  - index (pozice) prvku
- výstupní informace index nalezeného prvku (`i`):
  - `i >= 0`: prvek se nachází v posloupnosti na pozici i
  - `i = -1`: prvek se nenachází
- prvek je v poli vícekrát, vracíme:
  - první výskyt
  - všechny výskyty (výstupní informace = pole)
- dle organizace vstupní posloupnosti:
  - neuspořádaná posloupnost – nutné tzv. sekvenční vyhledávání
  - uspořádaná (setříděná) posloupnost – lze i tzv. binární vyhledávání
- dle povahy hledaného prvku
  - elementární prvek (jedno číslo v poli `int`)
  - subposloupnost („text v textu“)

Sekvenční vyhledávání

První výskyt prvku

- první výskyt prvku = stop algoritmu

- požadavek: metoda, bude vracet pozici prvku
  - nalezeno → pozice (0 až `pole.Length-1`)
  - nenalezeno → záporné číslo (-1)
- řešení v C#:

```
static int SeqSearch(int[] pole, int prvek)
{
      int index;
      for(int i = 0; i < pole.Length; i++) {
            if (pole[i] == prvek) {
                  index = i;
                  return index;
            }
      }
      return -1;
}
static void Main(string[] args)
{
      int[] pole = { 3, 89, 1, 55, 12 };
      int pozice = SeqSearch(pole, 55);
      if (pozice == -1)
            Console.WriteLine("Prvek nebyl nalezen");
      else
            Console.WriteLine("Prvek je na pozici {0}", pozice);
}
```

Všechny výskyty prvku

- Nutno prohledat celé pole
- Návratová hodnota = pole, prvky → pozice (indexy) výskytu prvku
  - Nenalezen žádný prvek → `null`
- problém: výstupní pole nutno vytvořit před vyhledáváním, ale velikost (počet nalezených výskytů) známo po vyhledávání. Řešení:
  - vytvoření pomocného pole o max velikosti
  - po skončení hledání (znám počet výskytů) → vytvořím správné pole a pomocné → kopie

- řešení v C#:

```
static int[] SeqSearchAll(int[] pole, int prvek)
{
      int dalsi = 0;    // indexy v poli nalezenych vyskytu
      int[] tempPole = new int[pole.Length];
      for (int i = 0; i < pole.Length; i++)
      {
            if (pole[i] == prvek)
                  tempPole[dalsi++] = i;  // ulozime pozici
      }
      // vytvorime nove pole o spravnem poctu prvku
      if (dalsi > 0)
      {
            int[] indexy = new int[dalsi];
            for (int i = 0; i < indexy.Length; i++)
                  indexy[i] = tempPole[i];
            return indexy;
      }
      else
            return null;
}
```

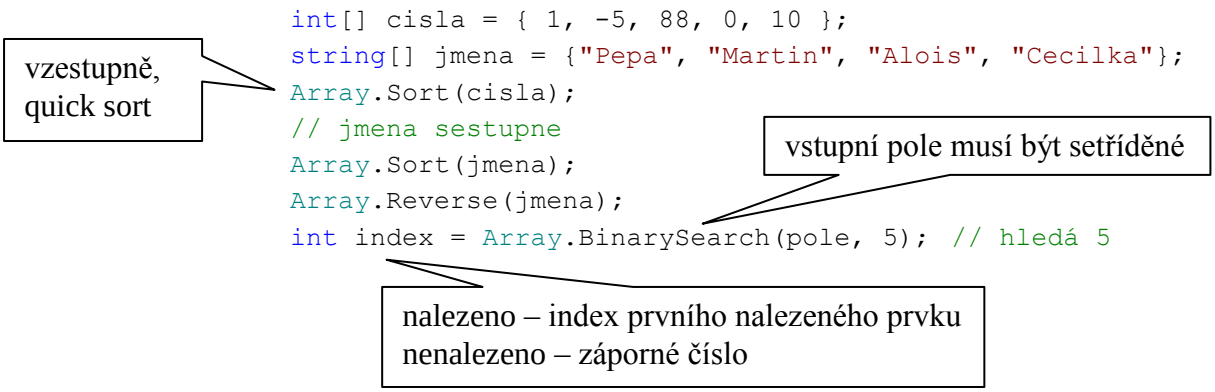
Binární vyhledávání

- Více hledání nad stejnými daty → vyplatí se data setřídit
- Pak lze hledat metodou půlení intervalu = binární vyhledávání → je rychlejší než sekvenční
- Výstup všech výskytů nemá smysl!!!
- řešení v C:

```
int BinarySearch(int[] pole, int prvek)
{
      int stred, dolni = 0, horni = pole.Length - 1;

      while (horni >= dolni)
      {
            stred = (horni + dolni) / 2;
            if (prvek < pole[stred])
            {
                  horni = stred - 1;
            }
            else if (prvek > pole[stred])
            {
                  dolni = stred + 1;
            }
            else
            {
                  // prvek nalezen
                  return stred;
            }
      }
      return -1;  // nenalezen
}
```

Řazení a vyhledávání v C# prakticky



- Metody třídy `Array` umí pracovat s poli (skoro) jakéhokoli typu