Algoritmy řazení dat

- Zpracování dat
- Úloha řazení/třídění dat
- Třídění dat v poli
- Třídění výběrem
- Třídění vkládáním
- Třídění přímou výměnou

ZPRACOVÁNÍ DAT

Základní algoritmy zpracování sady dat

- Data
  - záznamy o studentech fakulty (jméno, příjmení, datum narození, datum přijetí, průběh studia)
  - kniha jízd – datum, SPZ vozidla, počet ujetých kilometrů, čerpání phm atd.
  - objednávky firmy – objednatel, položky objednávky

- Základní operace
  - přidat záznam (studenta)
  - odstranit záznam (například v případě chybného založení)
  - modifikovat aktuálně zaznamenané údaje
  - vyhledat konkrétní záznam (na základě stanoveného kritéria)
  - seřadit záznamy dle požadovaného kritéria (dle jména, dle studijních výsledků)

- Uchování záznamů aktuálně zpracovávaných programem
  - **pole**
  - soubor
  - další struktury jako je seznam

ŘAZENÍ DAT V POLI

Úloha řazení (třídění) dat

- Budeme se zabývat problematikou, jak uspořádat data dle zadaného kritéria

- Uvažujeme sadu dat/hodnot libovolného (konkrétního) typu
- Předpokládáme, že pro hodnoty daného typu je definována relace „uspořádání“ s operací ≤
  - Reálná, celá čísla
  - Komplexní čísla
  - Textové řetězce
  - Body roviny
  - Záznamy o studentech fakulty
- Naším cílem je z dané sady hodnot vytvořit uspořádanou posloupnost a_1, a_2 … a_n (n je počet hodnot) tak, aby pro každé dvě sousední hodnoty a_i, a_{i+1} platilo $a_i \leq a_{i+1}$
  - Takto jsme zavedli požadavek na vzestupné uspořádání hodnot
  - Obdobně bychom mohli zavést sestupné uspořádání

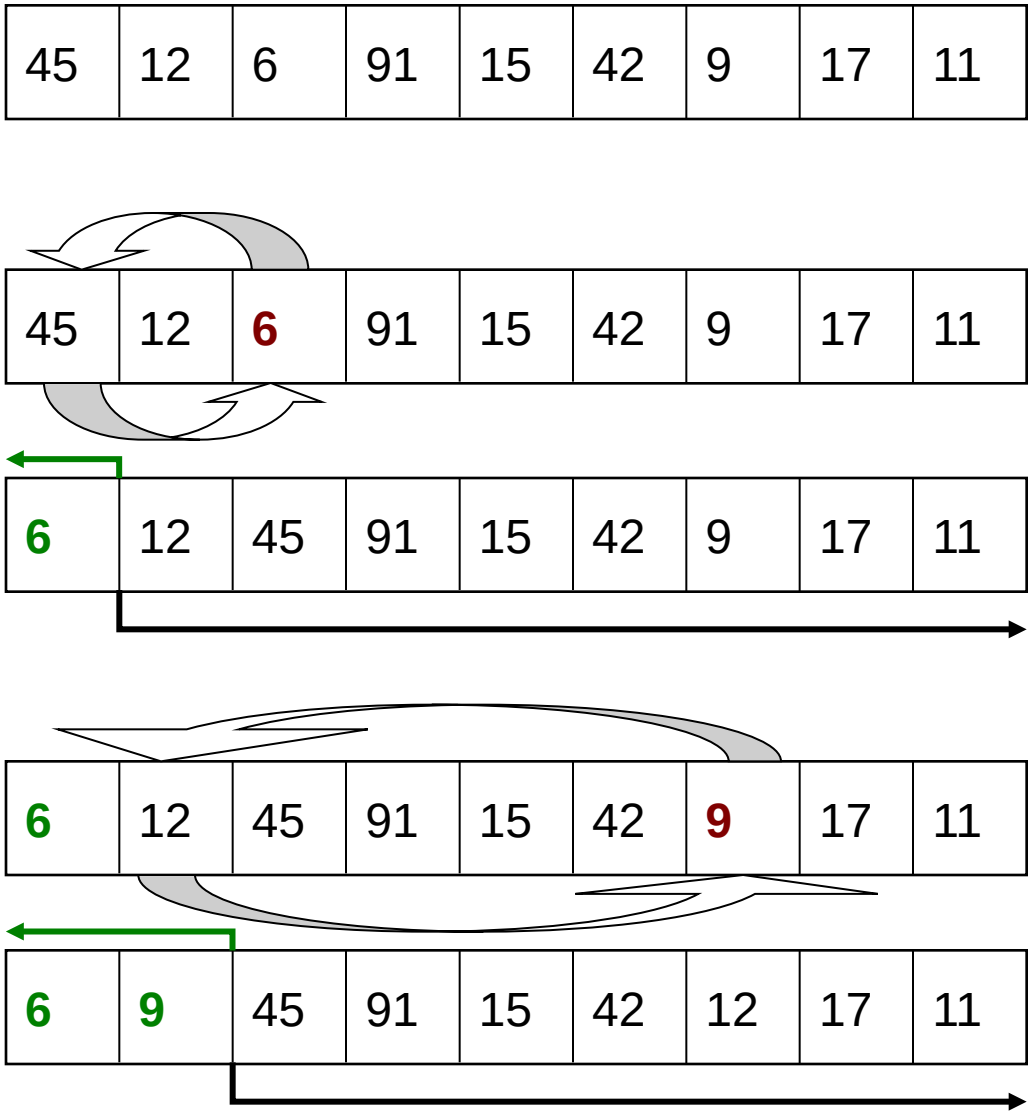
- Různé metody řazení
  - Metody pro řazení dat v poli, v souboru
  - Přímé metody, nepřímé metody

- Dále budou uvedeny tři různé algoritmy –
  - Přímé algoritmy pro řazení číselných hodnot v poli
  - Klíčem podle kterého budeme porovnávat dvě položky pole bude příslušná hodnota
  - Číselné hodnoty pro porovnání použijeme relační operátory

ALGORITMY TŘÍDĚNÍ

Řazení přímým výběrem

- Pole číselných hodnot
- V prvním kroku hledáme prvek na první pozici
- Na první pozici patří minimální prvek z celého rozsahu od indexu 0 do n-1
- Vyhledáme minimální prvek a zajistíme jeho přesun na první pozici (výměnou dvou prvků pole)
- Po prvním kroku
- Ve druhém kroku hledáme prvek na druhou pozici – minimální prvek od druhého do n-tého prvku v poli



Řazení přímým výběrem

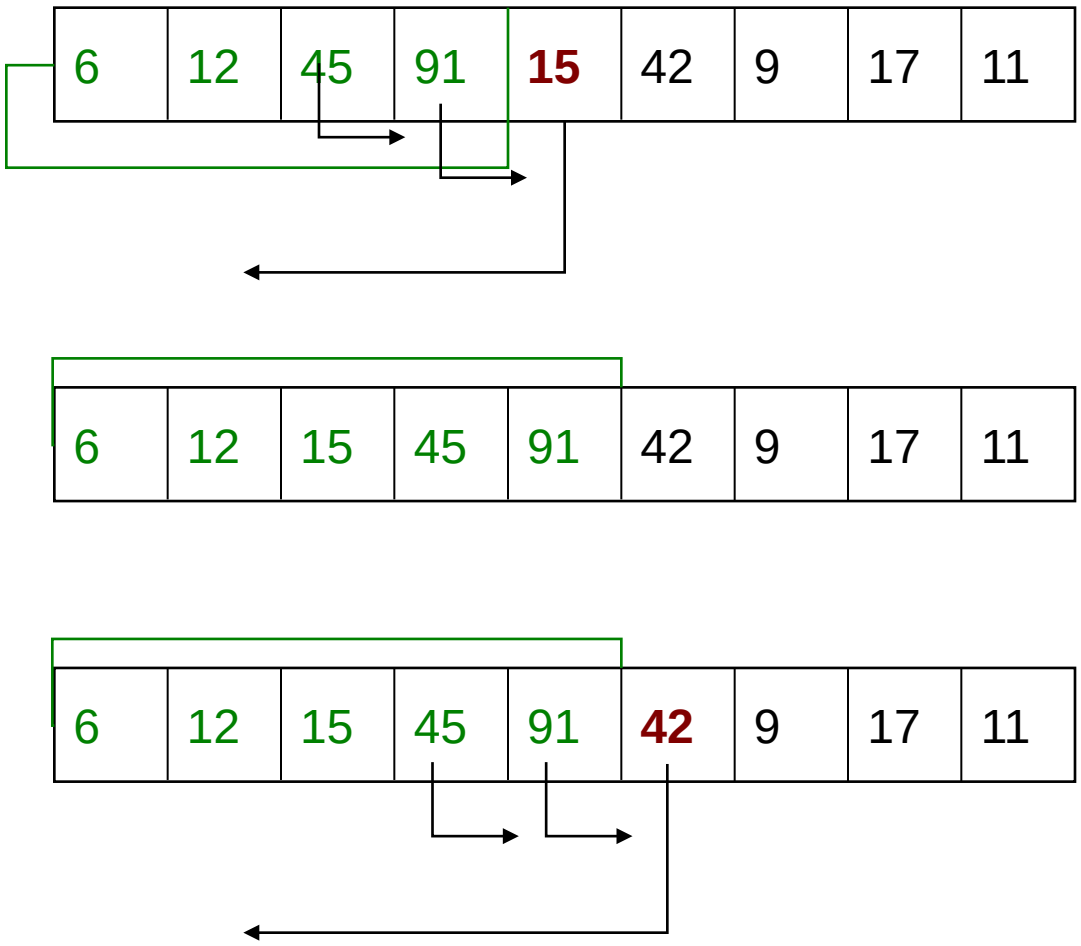
```
for (int i = 0; i < n-1; i++){
  imin = i;
  for (int j = i+1; j < n; j++) {
    if (a[j] < a[imin]) imin = j;
  }
  if (i != imin) {
    pom = a[i];
    a[i] = a[imin];
    a[imin] = pom;
  }
}
```

- Pole a hodnot primitivního datového typu
- V poli je n hodnot
- Indexy od `0` do `n-1`
- Pomocná proměnná `pom` je stejného datového typu jako položky pole

- **Přímý výběr** – postupně vybíráme minimální prvek v daném rozsahu pole a přesouváme jej na správné místo

Řazení přímým vkládáním

- Každý prvek se postupně vkládáme do zleva vzestupně setříděné posloupnosti
- Obecný i-tý krok
- Prvek a_i vkládáme do setříděné posloupnosti a_0 .. a_{i-1}
- Každý větší prvek odsuneme vpravo
- Pokud nalezneme prvek, který je menší nebo roven vkládané hodnotě, vložíme hodnotu za něj
- Tento postup aplikujeme pro všechny prvky posloupnosti



Řazení přímým vkládáním

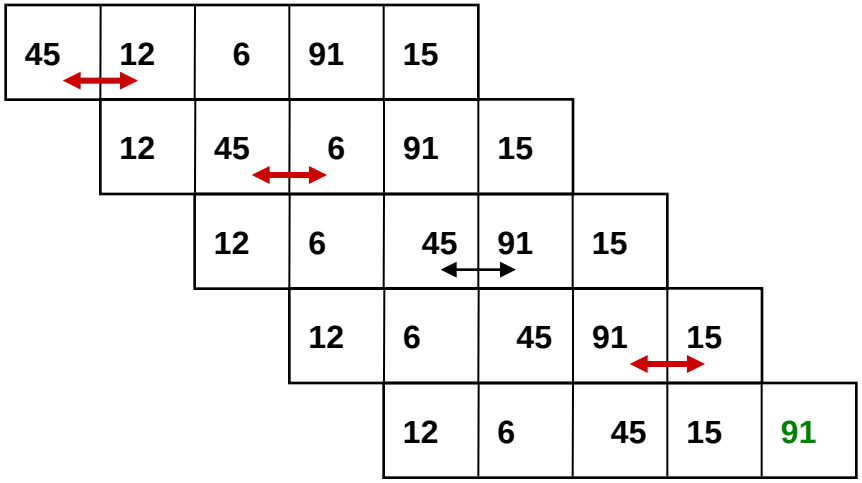
```
...

for (int i = 1; i<n; i++){
  pom = a[i];
  j = i - 1;
  while ((j >= 0 ) && (a[j] > pom)){
    a[j+1] = a[j];
    j--;
  }
  a[j+1] = pom;
}
```

- Pole a hodnot primitivního datového typu
- V poli je n hodnot
- Indexy od `0` do `n-1`
- Pomocná proměnná `pom` je stejného datového typu jako položky pole

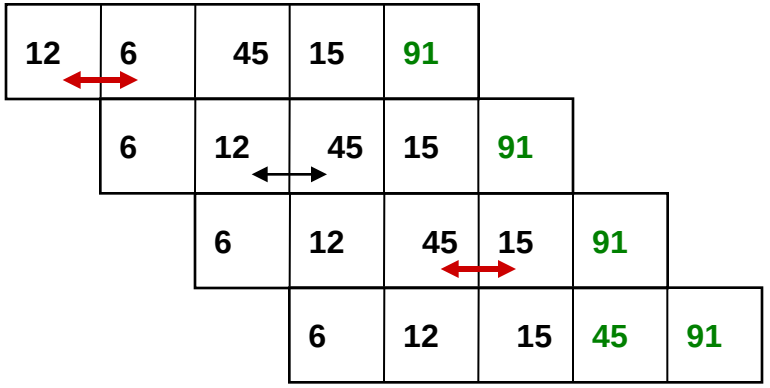
- **Přímé vkládání** – prvky postupně vkládáme do vzestupně setříděné podposloupnosti vytvářené průběžně od indexu 0 v příslušném poli

Řazení přímou výměnou

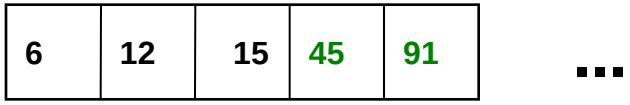
- Projdeme celé pole a porovnáme vždy dva sousední prvky – pokud neodpovídají uspořádání, potom je navzájem vyměníme
- Výsledkem celého průchodu polem bude, že přinejmenším na poslední místo se dostane největší hodnota celého pole



- Udělejme totéž znovu
- Při druhém průchodu postačí projít pole do předposledního prvku



- Pokud uvedený postup/průchod zopakujeme pole (n-1)x (s postupně snižující se horní hranicí indexu pole), potom výsledkem celé operace bude uspořádané pole



Řazení přímou výměnou

```
...

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

- Pole a hodnot primitivního datového typu
- V poli je n hodnot
- Indexy od `0` do `n-1`
- Pomocná proměnná `pom` je stejného datového typu jako položky pole

- **Přímá výměna** – postupně testujeme a vzájemně vyměňujeme sousední prvky v poli

Řazení přímou výměnou

- Předchozí varianta algoritmu třídění přímou výměnou je velmi neefektivní
- Postup má některé přirozené vlastnosti, které můžeme použít pro zvýšení efektivity
- Při každém průchodu pole zaznamenáme
  - Zda nastala nějaká výměna
  - Kde k příslušné výměně došlo
- Takto zaznamenané informace použijeme při dalším průchodu
  - Další průchod provedeme pouze v případě, že při minulém (průchodu) došlo k výměně; v opačném případě je pole setříděné
  - Při dalším průchodu bude horní hranice indexu odvozena od místa poslední výměny při průchodu předešlém (od poslední výměny do konce je pole setříděné)

Řazení přímou výměnou

```
...
vymena = n-1;
while (vymena > 0) {
  kam = vymena;
  vymena = 0;
  for (int i = 0; i < kam; i++){
    if (a[i] > a[i+1]){
      pom = a[i];
      a[i] = a[i+1];
      a[i+1] = pom;
      vymena = i;
    }
  }
}
```

- Pole a hodnot primitivního datového typu
- V poli je n hodnot
- Indexy od `0` do `n-1`
- Pomocná proměnná `pom` je stejného datového typu jako položky pole

- **Přímá výměna** – postupně testujeme a vzájemně vyměňujeme sousední prvky v poli

- Takto upravený algoritmus využívá přirozených vlastností použité metody