Triedenie

je činnosť, po skončení ktorej pre všetky dovolené hodnoty indexov poľa platí, že pole[i] ≤ pole[i+1].
Pre triedenie zoznamov (list, a teda aj našich polí) máme v Pythone k dispozícii funkciu **zoznam.sort()** a funkciu **sorted(zoznam)**. Funkcia zoznam.sort() zmení daný zoznam (je na úrovni príkazu). Funkcia sorted(zoznam) vráti nové utriedené pole (funkcia sorted() vystupuje na pravej strane príkazu priradenia, napr. utriedenePole = sorted(zoznam)).
Použitie metódy sort:

```
import pole
poleInt = pole.vytvorIntNahodne(0,100)
poleInt.sort()
pole.vypis(poleInt)
```

Výpis:

```
Počet prvkov poľa: 10
Prvky poľa:
[86, 1, 43, 100, 72, 45, 47, 91, 46, 46]
Prvky poľa:
[1, 43, 45, 46, 46, 47, 72, 86, 91, 100]
```

Použitie funkcie sorted:

```
import pole
poleInt = pole.vytvorIntNahodne(0,100)
poleIntUtr = sorted(poleInt)
print(poleIntUtr)
print(poleInt)
```

Výpis:

```
Počet prvkov poľa: 13
Prvky poľa:
[21, 78, 66, 36, 21, 53, 6, 49, 25, 49, 22, 8, 13]
[6, 8, 13, 21, 21, 22, 25, 36, 49, 49, 53, 66, 78]
[21, 78, 66, 36, 21, 53, 6, 49, 25, 49, 22, 8, 13]
```

Funkcia sort() aj sorted() môžu mať pri volaní ešte parameter key (použitie je nad rámec nášho kurzu) a parameter reverse, ktorý, ak má hodnotu True, pole utriedi zostupne (predvolená hodnota parametra reverse je False, t.j. vzostupné utriedenie poľa).

Príklad T.1
Vytvorte funkciu, ktorá zistí počet rôznych hodnôt v poli.

```
def vratPocetRoznychHodnot(pole):
    pom = sorted(pole)
    pocetRoznych = 1
    for i in range(1,len(pom)):
        if pom[i] != pom[i-1]:
            pocetRoznych += 1
    return pocetRoznych
```

Použitie:

```
poleInt = pole.vytvorIntNahodne()
print("Počet rôznych prvkov v poli: " + str(vratPocetRoznychHodnot(poleInt)))
```

Výpis:

```
Počet prvkov poľa: 10
1. prvok: Adam
2. prvok: Adela
3. prvok: Dáša
4. prvok: Peter
5. prvok: Pavol
```

```
6. prvok: Zuzana
7. prvok: Adela
8. prvok: Zuzana
9. prvok: Pavol
10. prvok: Adela
Prvky poľa:
['Adam', 'Adela', 'Dáša', 'Peter', 'Pavol', 'Zuzana', 'Adela', 'Zuzana', 'Pavol', 'Adela']
Počet rôznych prvkov: 6
```

Je dobré ovládať nejaký triediaci algoritmus, jednak na zdokonalenie algoritmického myslenia, a aj pre prípad, že by daný programovací jazyk nemal priamo implementovaný niektorý z triediacich algoritmov. Triediace algoritmy, ako si ukážeme, možno použiť aj pri iných úlohách, ako len utriedení celej skupiny dát.
Rozlišujeme algoritmy vnútorného a vonkajšieho triedenia. Vnútorné triedenie používame vtedy, ak sa celá množina dát určená na triedenie zmestí do vnútornej pamäte počítača. Údaje sa uložia do poľa a triediaci algoritmus má k nim priamy prístup. Vonkajšie triedenie pracuje s dátami uloženými v súbore na vonkajšej pamäti, pretože sa všetky nezmestili do vnútornej pamäte počítača. Vonkajším triedením sa nebudeme zaoberať.
Algoritmov vnútorného triedenia je viacej a navzájom sa líšia zložitosťou a tomu zodpovedajúcou efektívnosťou triedenia (čím jednoduchší algoritmus, tým menej efektívny). Pri výbere triediaceho algoritmu možno prihliadať aj na veľkosť vstupných údajov, pri niekoľko sto prvkových poliach je najpomalší triediaci algoritmus prakticky rovnako rýchly ako ten najrýchlejší, ktorého napísanie a pochopenie nám môže trvať výrazne dlhší čas.
Najjednoduchšie vnútorné triediace algoritmy majú niekoľko spoločných rysov – zápis algoritmu je krátky a jednoduchý, časová zložitosť je kvadratická (triedenie sa realizuje cyklom v cykle). Sem patria napríklad bublinkové triedenie (triedenie výmenou, Bubble sort), triedenie priamym vkladaním (vsúvaním, Insertion sort), triedenie priamym výberom (Select sort) atď.

T.2 Naprogramujte triedenie výmenou – bublinkové triedenie (Bubble sort).

Už samotné bublinkové triedenie môže mať veľa variant, od dvoch vnorených cyklov s pevným počtom opakovaní, až po vonkajší cyklus s opakovaním, pokým dochádza k výmene pri prechode poľom. Porovnávať tiež môžeme prvky pole[i-1] a pole[i], alebo pole[i] a pole[i+1], ísť od prvého alebo posledného prvku poľa a pod.

T.2.1 Vytvorte triediaci algoritmus, ktorý bude porovnávať dva vedľa seba stojace prvky, a ak je ľavý väčší ako pravý, vymení ich. Určte potrebný počet prechodov poľom.
Napríklad

Pôvodné pole	**5**	**3**	**4**	**2**	**1**
Pri 1. prechode sa vymenili	5 a 3	5 a 4	5 a 2	5 a 1	5
Po 1. prechode	3	4	2	1	5
Po 2. prechode	3	2	1	4	5
Po 3. prechode	2	1	3	4	5
Po 4. prechode	1	2	3	4	5

Z príkladu vidieť, že poľom treba prejsť pri N prvkoch N-1 krát. Hodnota N je dĺžka poľa, teda N = len(pole).

```
def utriedBubble(pole):
  """ Bubble sort
      po prvom prechode je najväčší prvok na svojom mieste
  """
  for cp in range(1,len(pole)):                    # cp - číslo prechodu poľom: prvý, druhý,..., len(pole)-1
    for i in range(0,len(pole)-cp):
      if pole[i]>pole[i+1]:
        pole[i], pole[i+1] = pole[i+1], pole[i]
  return pole
```

Pri uvedenej procedúre sa poľom prejde len(pole)-1 krát, kde len(pole) udáva počet prvkov poľa (vonkajší cyklus). Pri každom prechode poľom (vnútorný cyklus) sa porovnávajú dva vedľa seba ležiace prvky (pole[i] a pole[i+1]) a ak je ľavý prvok väčší ako pravý, vymenia sa. Po prvom prechode poľom musí byť na svojom mieste najväčší

prvok v poli, po druhom druhý najväčší atď. čo umožňuje skracovať vnútorný cyklus o hodnotu cp (po prvom prechode poľom už netreba porovnať posledný prvok, po druhom prechode ani predposledný prvok atď.).
Po len(pole)-1 prechodoch poľom (vonkajší cyklus) je na svojich miestach sprava počet-1 prvkov a teda pole je utriedené (najmenší prvok – prvý zľava, nemá inú možnosť, len byť na prvom mieste v poli, keďže všetky väčšie prvky boli presunuté doprava).

Použitie:

```
poleInt = pole.vytvorIntNahodne()
poleIntUtr = utriedBubble(poleInt[:])
print("Pôvodné pole:", poleInt)
print("Utriedené pole:", poleIntUtr)
```

T.2.2 Analyzujte nasledujúci triediaci algoritmus:

```
def utriedBubbleX(pole):
    """ Bubble sort
        po prvom prechode je najmenší prvok na svojom mieste
    """
    for cp in range(len(pole)-1):              # cp nadobúda hodnoty: 0, 1, 2,..., len(pole)-2
        for i in range(len(pole)-1, cp, -1):
            if pole[i] < pole[i-1]:
                pole[i], pole[i-1] = pole[i-1], pole[i]
    return pole
```

T.2.3* Vytvorte verziu bublinkového triedenia, v ktorej, ak už nenastala výmena pri prechode poľom, triedenie sa ukončí.

T.3 Naprogramujte triedenie výberom – Selection sort.

Triedenie výberom sa môže realizovať cez minimá alebo maximá. Opis algoritmu triedenia cez minimá: nájdeme najmenší prvok poľa a vymeníme ho s prvkom na prvom mieste v poli, potom vyberieme druhý najmenší a vymeníme ho s prvkom na druhej pozícii v poli atď. až po posledný prvok poľa (stačí predposledný prvok poľa, najväčší prvok musí už byť na poslednom - správnom mieste). Inou možnosťou je nájsť najväčší prvok poľa a vymeniť ho s posledným, následne vybrať druhý najväčší a vymeniť s predposledným prvkom poľa atď.

```
def selectionSort(pole):
    """ Selection sort cez minimá
        nájde najmenší prvok a vymení ho s prvým, potom druhý najmenší a vymení ho s druhým prvkom v poli atď.
    """
    dh = 0                                              # nastavenie dolnej hranice prehľadávanej oblasti poľa
    while dh < len(pole)-1:                             # pokiaľ je čo prehľadávať
        inxmin = dh                                     # počiatočná hodnota indexu minima
        for i in range(dh + 1, len(pole)):              # prehľadávanie zvyšnej časti poľa
            if pole[i] < pole[inxmin]:
                inxmin = i                              # nový index minima v prehľadávanej oblasti
        pole[dh], pole[inxmin] = pole[inxmin], pole[dh] # výmena
        dh += 1                                         # zvýšenie dolnej hranice prehľadávanej oblasti poľa
    return pole
```

Použitie:

```
poleInt = pole.vytvorIntNahodne()
poleIntUtr = selectionSort(poleInt[:])
pole.vypis(poleIntUtr)
print("Kontrola:")
poleInt.sort()
pole.vypis(poleInt)
```

Poznámka:
Skupinu príkazov `dh = 0, while dh < len(pole)-1:` a `dh += 1` možno nahradiť for-cyklom.

Variant s využitím štandardných funkcií min() a index() pre zoznamy:

```
def selectionSortFu(pole):
    """ Selection sort cez minimá využívajúci štandardné funkcie min() a index() pre zoznamy
    """
    dh = 0                                                  # nastavenie dolnej hranice prehľadávanej oblasti poľa
    while dh < len(pole)-1:                                 # pokiaľ je čo prehľadávať
        lokmin = min(pole[dh:])                             # lokmin je minimum v časti pole[dh:]
        inxmin = pole[dh:].index(lokmin)                    # index prvého výskytu lokmin v prehľadávanej časti poľa
        pole[dh], pole[dh+inxmin] = pole[dh+inxmin], pole[dh]       # výmena
        dh += 1                                             # posunutie dolnej hranice prehľadávanej oblasti poľa
    return pole
```

Selection sort cez maximá:

```
def utriedCezMaxima(pole):
    """ Selection sort cez maximá
        nájde najväčší prvok poľa a vymení ho s posledným v poli, potom druhý najväčší a vymení ho
        s druhým prvkom od konca poľa atď.
    """
    hh = len(pole)                                          # nastavenie hornej hranice prehľadávanej oblasti poľa
    while hh > 0:                                           # pokiaľ je čo prehľadávať
        inxmax = 0                                          # počiatočná hodnota indexu maxima!!!
        for i in range(1,hh):                               # prehľadávanie zvyšnej časti poľa
            if pole[i]>pole[inxmax]:
                inxmax = i                                  # nový index maxima v prehľadávanej oblasti
        hh -= 1                                             # posunutie hornej hranice prehľadávanej oblasti poľa!!!
        pole[inxmax], pole[hh] = pole[hh], pole[inxmax]     # výmena až teraz (prečo?)
    return pole
```

Úlohy na precvičenie:

- napíšte a odladťte triedenie výberom cez maximá s využitím štandardných funkcií pre zoznamy
- vytvorte a použite funkciu, ktorá efektívne vypíše k (0 < k ≤ len(pole)) najväčších prvkov poľa,
- vytvorte a použite funkciu, ktorá efektívne vypíše k (0 < k ≤ len(pole)) najmenších prvkov poľa,
- * vytvorte a použite funkciu, ktorá efektívne vypíše k (0 < k ≤ len(pole)) rôznych najväčších prvkov poľa,
- * vytvorte a použite funkciu, ktorá efektívne vypíše k (0 < k ≤ len(pole)) rôznych najmenších prvkov poľa,
- * napíšte a odladťte triedenie výberom cez minimum a maximum súčasne (otestujte pre polia [3,1,2] a [2,3,1]).

Nepovinné príklady rozširujúce učivo o triedení:

T.4* Naprogramujte triedenie priamym vkladaním - Insertion sort.

Ako už naznačuje názov algoritmu, postupne berieme druhý, tretí, štvrtý až posledný prvok poľa a vkladáme ich na „správne“ miesto vľavo od pôvodnej pozície vkladaného prvku. „Správne“ miesto pre vkladaný prvok je také, že vľavo sú už len menšie čísla a vpravo (po pôvodnú pozíciu) väčšie alebo rovné hodnote vkladaného prvku. Pole je v tejto časti čiastočne utriedené.

Napríklad

Prvok:	1.	2.	3.	4.	5.
Index:	0	1	2	3	4
Pôvodné pole	5	3	4	2	1
2. prvok na „svoje“ miesto	3	5	4	2	1
3.prvok na „svoje“ miesto	3	4	5	2	1
4.prvok na „svoje“ miesto	2	3	4	5	1
5.prvok na „svoje“ miesto	1	2	3	4	5

Všimnite si, že pole je zľava po vkladaný prvok čiastočne utriedené (možno využiť v špecifických úlohách).

Jedna z viacerých alternatív:

```
def insertionSort(pole):
    """ Insertion sort
        zoberie druhý prvok poľa a vloží ho na „svoje miesto“ vľavo,
        potom tretí prvok poľa a vloží ho na „svoje miesto“ vľavo atď.
    """
    for i in range(1, len(pole)):          # vložiť prvok s indexom 1, potom 2, atď. až s posledným indexom
        vlozit = pole[i]
        inxkam = i-1
        while inxkam >= 0 and pole[inxkam] > vlozit:
            pole[inxkam+1] = pole[inxkam]
            inxkam -= 1
        pole[inxkam+1] = vlozit            # inxkam je o jednu pozíciu vľavo od miesta, kam treba vložiť „vlozit“
    return pole
```

Použitie:

```
poleInt = pole.vytvorIntNahodne(0,100)
poleIntUtr = insertionSort(poleInt[:])
pole.vypis(poleIntUtr)
```

Výpis:

```
Počet prvkov poľa: 30
Prvky poľa:
[58, 22, 61, 14, 18, 83, 12, 86, 5, 35, 89, 98, 31, 86, 8, 19, 99, 52, 76, 68, 65, 83, 86, 62, 20, 89, 81, 69, 61, 89]
Prvky poľa:
[5, 8, 12, 14, 18, 19, 20, 22, 31, 35, 52, 58, 61, 61, 62, 65, 68, 69, 76, 81, 83, 83, 86, 86, 86, 89, 89, 89, 98, 99]
```

*Úlohy na precvičenie:

- napíšte program, ktorý bude vypisovať priebežné výsledky dobiehajúcich pretekárov do cieľa (po zadaní novej hodnoty z klávesnice vypíše utriedené dovtedy vložené hodnoty; počet pretekárov môže byť známy).