IB113 Úvod do programování a algoritmizace

Cvičení 8

Jaromír Plhák

Opakování binární vyhledávání

```
def binary_search(needle, haystack):
        lower_bound = 0
        upper_bound = len(haystack) - 1
        while lower_bound <= upper_bound:
                middle = (lower_bound + upper_bound) // 2
                if haystack[middle] == needle:
                        return True
                elif haystack[middle] > needle:
                        upper_bound = middle - 1
                else:
                        lower_bound = middle + 1
        return False
```

Vylepšete tuto funkci tak, aby vracela index pozice, kde se hledaný prvek nachází. Pokud prvek v seznamu není, vraťte -1.

Dobrovolné:
Vylepšete tuto funkci tak, aby vracela seznam indexů pozic, na kterých se hledaný prvek vyskytuje. Pokud prvek v seznamu není ani jednou, vrátíte prázdný seznam.

Domácí úkol 3

- Zadáno v Isu
  - https://is.muni.cz/auth/el/1433/podzim2017/IB113/um/homework_03/
- Je na vás, jaké úkoly implementujete
  - 5 úkolů z easy
    - 2 body za vyřešení
  - 2 úkoly z advanced
    - 5 bodů za vyřešení
- Zadání v angličtině
- V případě nějakých problémů, pište na diskuzní fórum
- Odevzdávejte archiv (.zip, .rar) obsahující všechna řešení
- Deadline čtvrtek 9. 11. 23:59

Funkce nad seznamy - vkládání

- Syntax: list_variable.function()
  - append(x) - vloží objekt x na konec seznamu
  - insert(i, x) - vloží objekt x na index i

```
>>> fruit = ["apple", "coconut"]
>>> fruit.append("date")
>>> fruit
['apple', 'coconut', 'date']
>>> fruit.insert(1, "banana")
>>> fruit
['apple', 'banana', 'coconut', 'date']
```

Funkce nad seznamy - mazání

- remove(x) - smaže první výskyt objektu x ze seznamu
  - Nic nevrací
  - Vyhodí výjimku, pokud x není v seznamu
- pop(i) - smaže objekt na indexu i ze seznamu
  - Smazaný objekt vrátí
  - Vyhodí výjimku, pokud i není v rozsahu

```
>>> fruit = ['apple', 'banana', 'coconut'
    , 'date']
>>> fruit.remove("apple")
>>> fruit.pop(0)
'banana'
>>> fruit
['coconut', 'date']
```

Funkce nad seznamy - vyhledávání

- count(x) - spočítá výskyty objektu x v seznamu
- index(x) - najde první index výskytu objektu x v seznamu

```
>>> fruit = ['apricot', 'peach', 'apple',
    'apple']
>>> fruit.count('apple')
2
>>> fruit.index('peach')
1
```

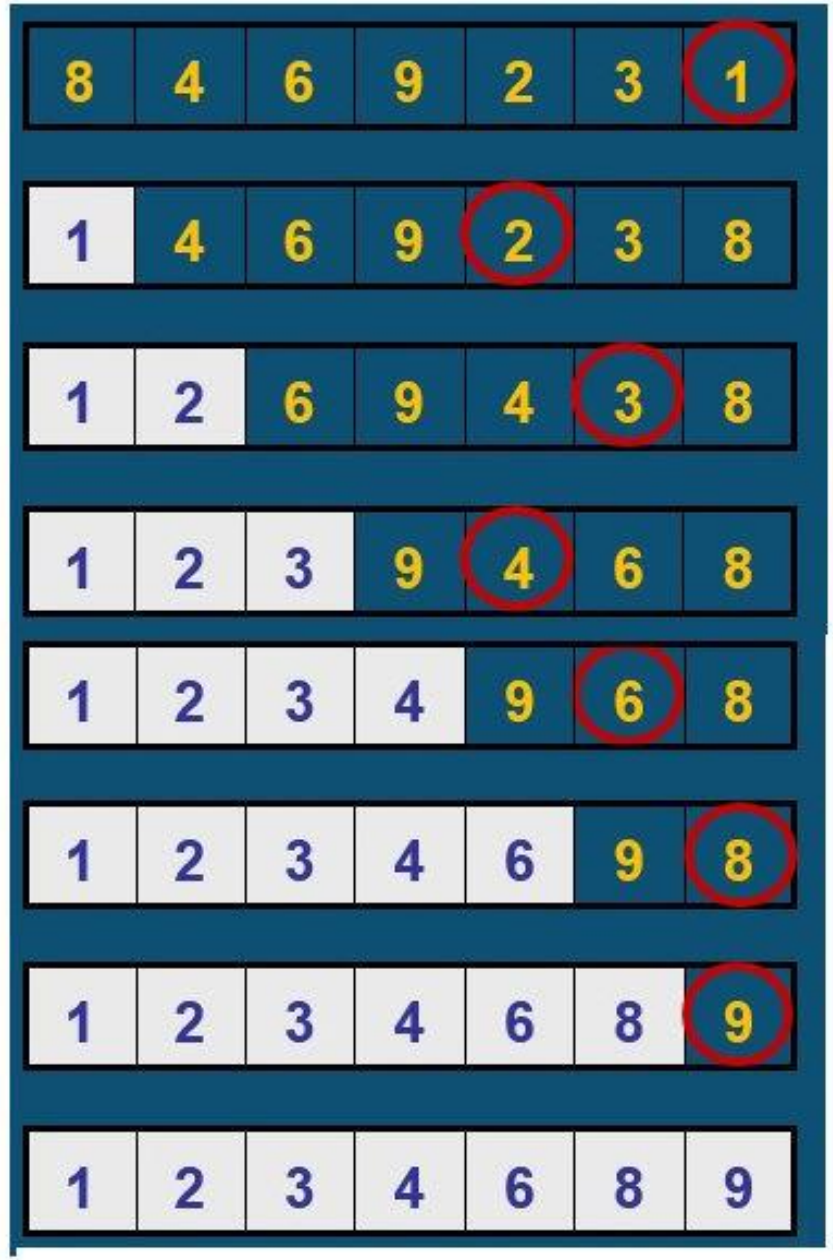
Funkce nad seznamy - řazení

- sort() - vzestupně uspořádá položky seznamu
  - Modifikuje seznam, nevrací nový
- sorted(lst) - vzestupně uspořádá položky seznamu
  - Nemodifikuje vstupní seznam, vrací nový

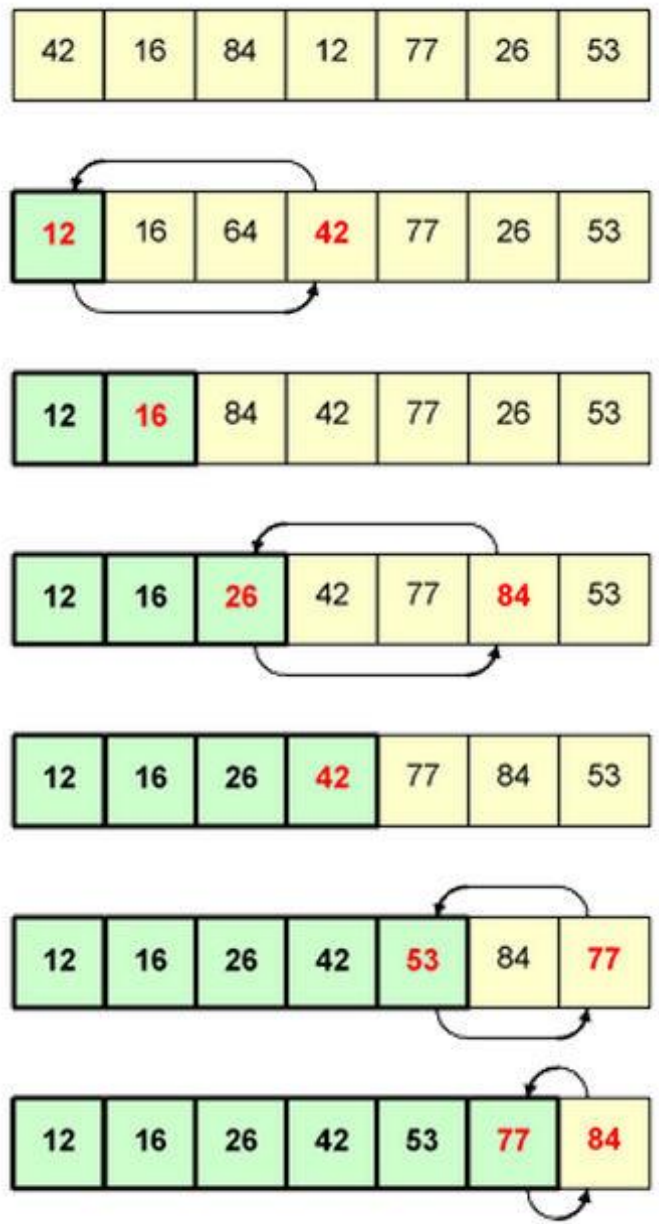
```
>>> fruit = ['apricot', 'peach', 'apple',
    'apple']
>>> fruit.sort()
>>> fruit
['apple', 'apple', 'apricot', 'peach']
>>> sorted(fruit)
['apple', 'apple', 'apricot', 'peach']
```

Selection sort

- Vstup: seznam hodnot, které se dají uspořádat
  - (Matematicky: na hodnotách existuje uspořádání)
- Vstupní seznam je **pomyslně** rozdělený na dvě části: seřazenou a neseřazenou
- Nejprve je seřazená část prázdná a neseřazená část je celý vstupní seznam
- V každém kroku algoritmus najde **minimum** neseřazené části a vymění ho s nejlevějším prvkem neseřazené části
  - Tím ho vlastně přidá na konec (napravo) seřazené části
  - Seřazená část se buduje zleva doprava
- Jakmile je neseřazená část prázdná, algoritmus končí
- Výstup: uspořádaný seznam hodnot vstupního seznamu



Zdroj: http://freefeast.info/general-it-articles/selection-sort-pseudo-code-of-selection-sortselectionsort-in-data-structure/



Zdroj: http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/09/array-sorting.html

Úkol

- Naprogramujte funkci selection_sort(lst), kde lst je seznam hodnot
- Využijte testovací výpisy před každou iterací cyklu pro kontrolu průběhu řazení

>>> selection_sort ([9 , 8, 4, 6, 1])

[9, 8, 4, 6, 1]

[1, 8, 4, 6, 9]

[1, 4, 8, 6, 9]

[1, 4, 6, 8, 9]

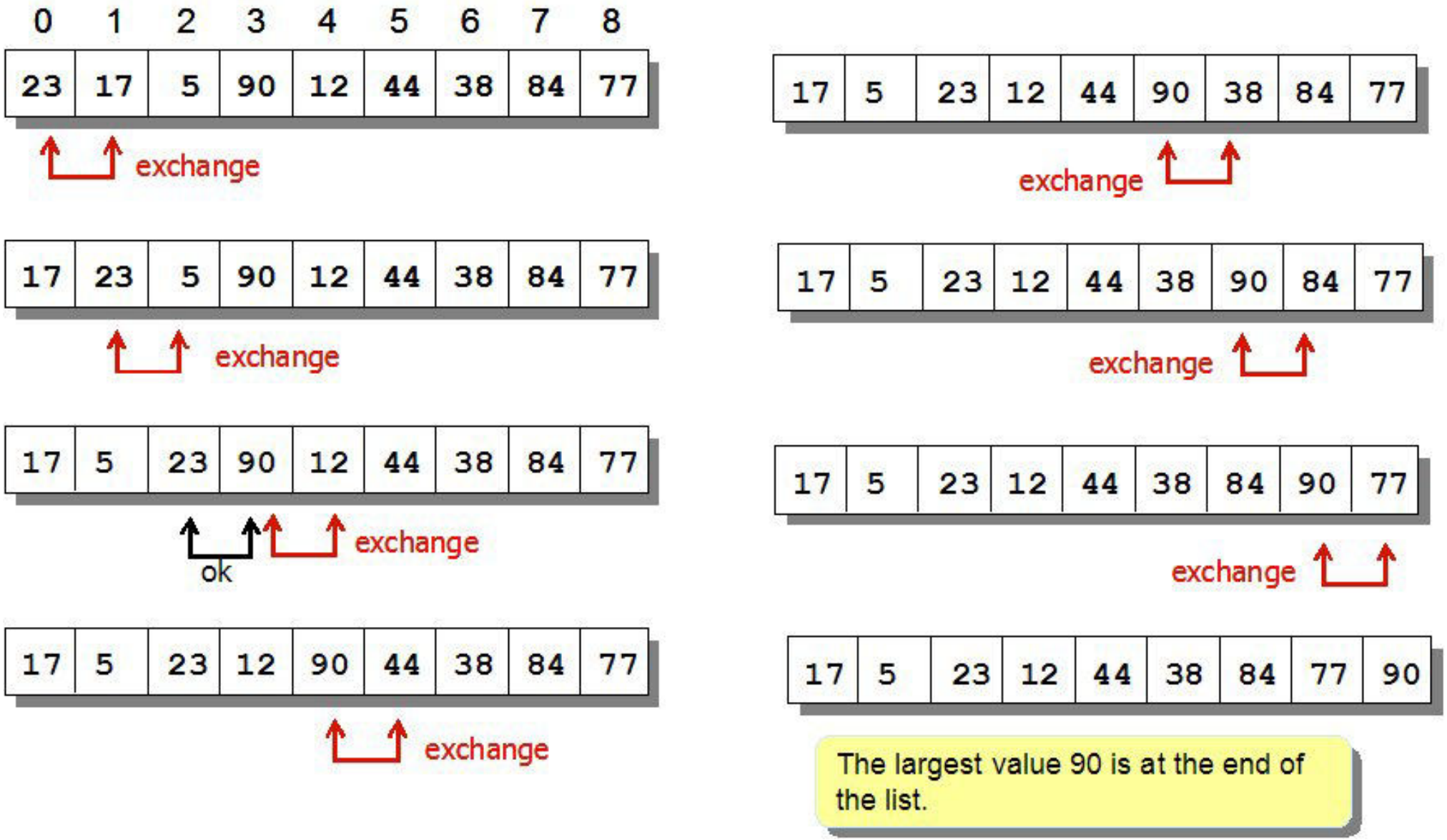
[1, 4, 6, 8, 9]

Finish : [1, 4, 6, 8, 9]

- Můžete měnit vstupní seznam

Bubble sort - myšlenka

- Vstup: seznam hodnot, na kterých existuje uspořádání
- Algoritmus prochází seznam od začátku a porovná **každou dvojici sousedních prvků**
  - Pokud jsou sousedi v opačném pořadí, vymění je
  - Pokud jsou sousedi ve správném pořadí, nechá je tak
- Pokud nastala alespoň jedna výměna, seznam se bude procházet **znova od začátku**
- Po každé iteraci **největší prvek** „probublá“ **na konec** seznamu
- Výstup: uspořádaný seznam hodnot vstupního seznamu



Zdroj: http://www.csit.parkland.edu/~mbrandyberry/CS1Java/Lessons/Lesson28/BubbleSort.htm

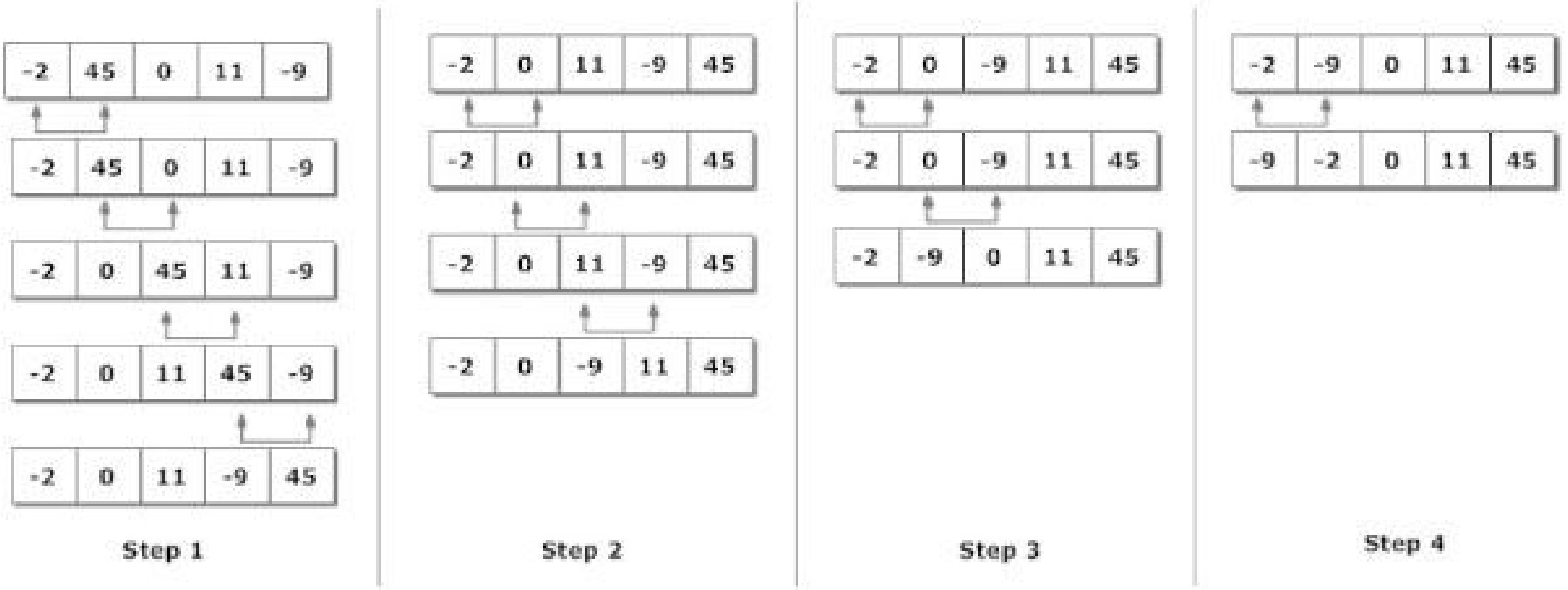


Figure: Working of Bubble sort algorithm

Zdroj: http://www.programiz.com/article/bubble-sort-algorithm-programming

Úkol

- Naprogramujte funkci bubble_sort(lst), kde lst je seznam hodnot
- Využijte testovací výpisy před každou iterací cyklu pro kontrolu průběhu řazení

>>> bubble_sort ([9 , 8, 4, 6, 1])

[9, 8, 4, 6, 1]

[8, 4, 6, 1, 9]

[4, 6, 1, 8, 9]

[4, 1, 6, 8, 9]

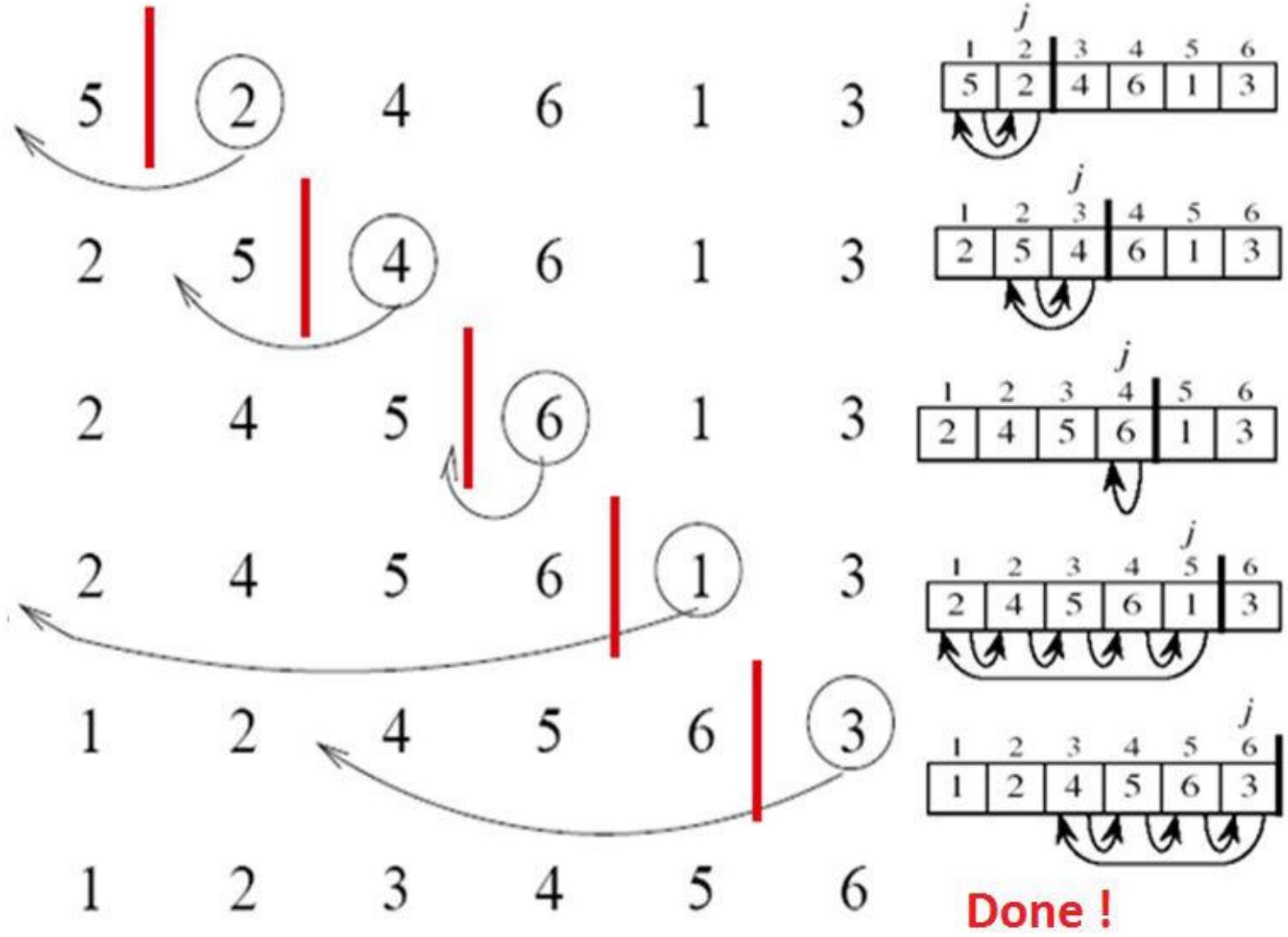
[1, 4, 6, 8, 9]

Finish : [1, 4, 6, 8, 9]

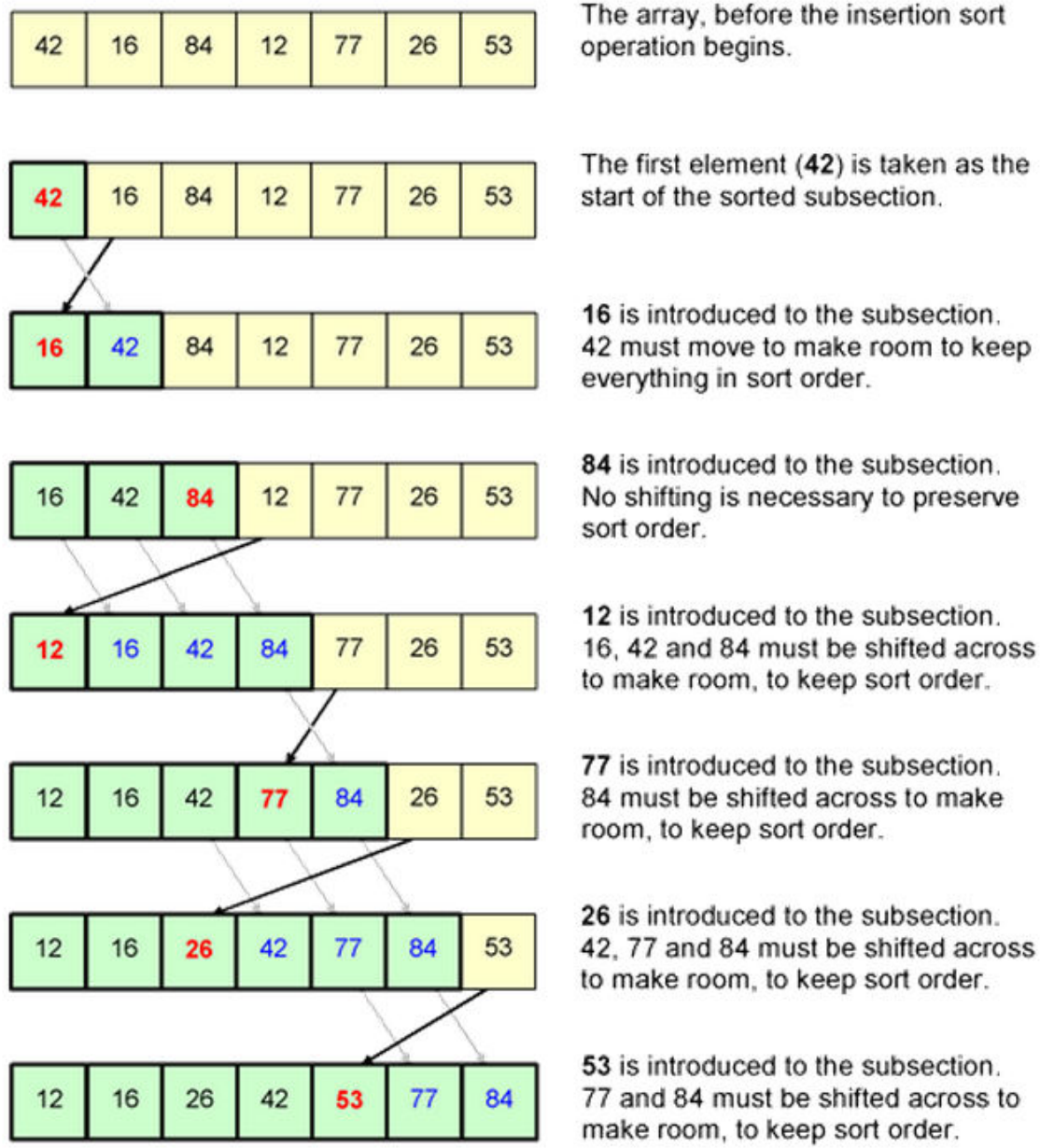
- Můžete měnit vstupní seznam

Insertion sort - myšlenka

- Vstup: seznam hodnot, které se dají uspořádat
  - (Matematicky: na hodnotách existuje uspořádání)
- Vstupní seznam je **pomyslně** rozdělený na dvě části: seřazenou a neseřazenou
- Nejprve seřazená část obsahuje první prvek seznamu a neseřazená část je zbytek vstupního seznam
- V každém kroku algoritmus vezme první prvek neseřazené části a vloží ho na správné místo seřazené části
- Jakmile je neseřazená část prázdná, algoritmus končí
- Výstup: uspořádaný seznam hodnot vstupního seznamu



Zdroj: http://freefeast.info/general-it-articles/insertion-sort-pseudo-code-of-insertion-sort-insertion-sort-in-data-structure/



Zdroj: http://www.cs.rmit.edu.au/online/blackboard/chapter/05/documents/contribute/chapter/09/array-sorting.html

Úkoly

- Naprogramujte funkci insertion_sort(lst), kde lst je seznam hodnot
- Využijte testovací výpisy před každou iterací cyklu pro kontrolu průběhu řazení

>>> insertion_sort ([9 , 8, 4, 6, 1])

[9, 8, 4, 6, 1]

[8, 9, 4, 6, 1]

[4, 8, 9, 6, 1]

[4, 6, 8, 9, 1]

Finish : [1, 4, 6, 8, 9]

- Můžete měnit vstupní seznam