Základy algoritmizace

7. Rekurzivní řazení

doc. Ing. Jiří Vokřínek, Ph.D.
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

Základy algoritmizace

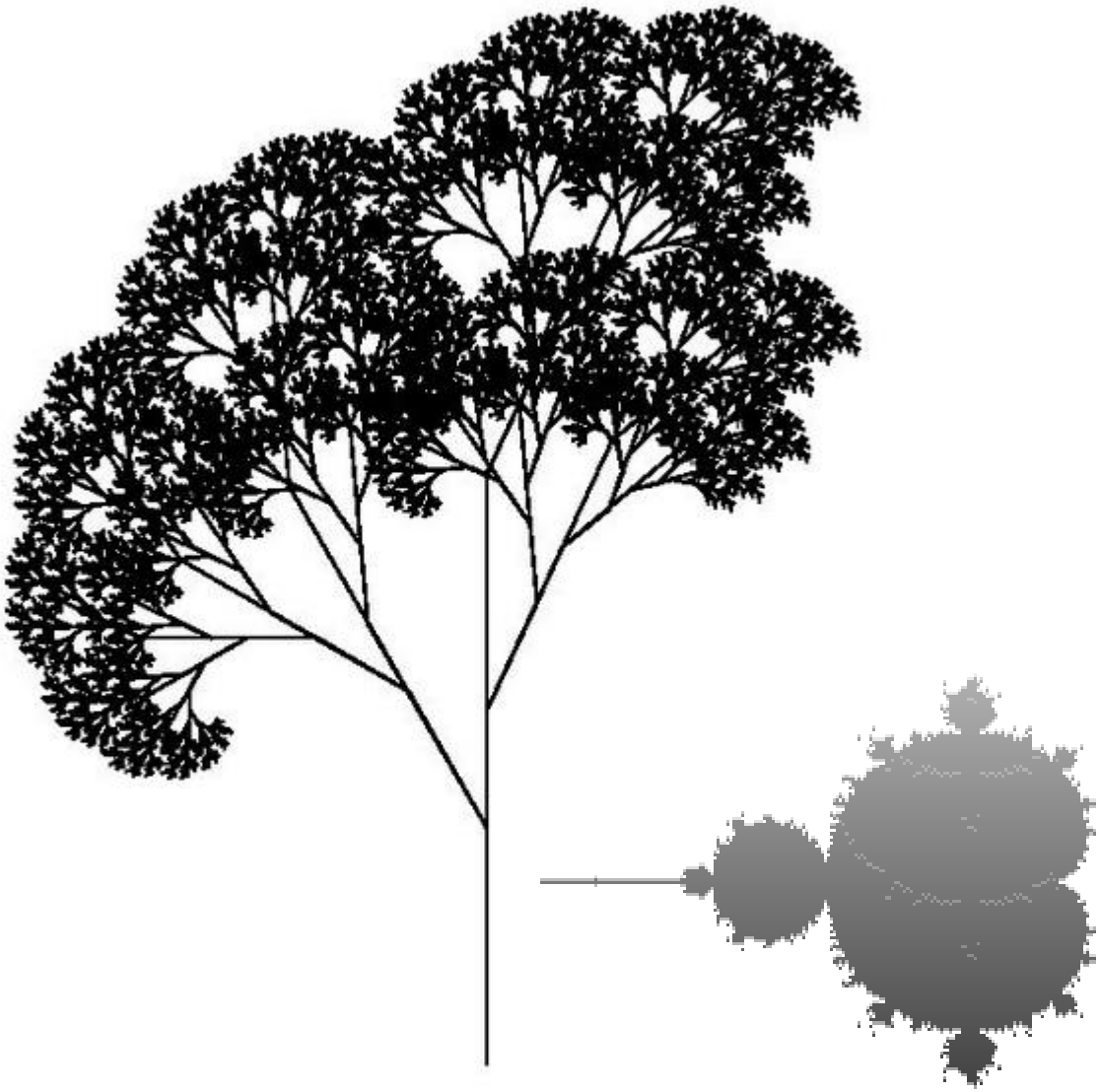
- Dnes:
  - Rekurzivní řazení
    - Merge sort
    - Quick sort

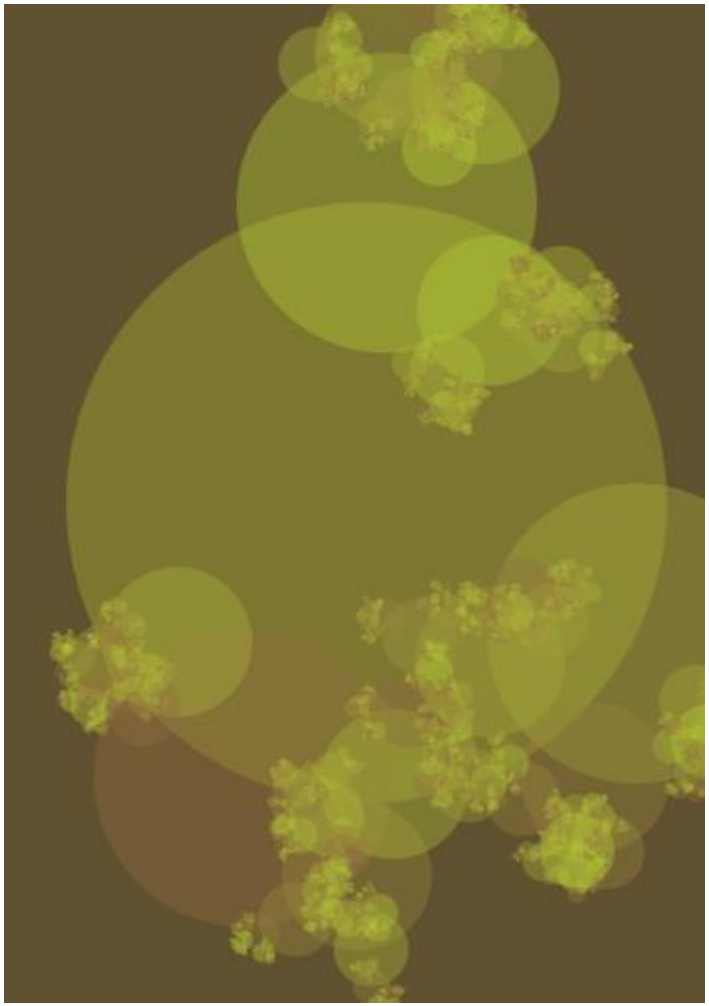
Rekurzivní algoritmy

- Rekurzivní algoritmus předepisuje výpočet *„shora dolů“* v závislosti na velikosti vstupních dat
  - Pro nejmenší (nejjednodušší) vstup je výpočet určen přímo
  - Pro obecný vstup je výpočet předepsán s využitím **téhož** algoritmu pro **menší vstup**
- Výhodou rekurzivních funkcí je jednoduchost a přehlednost

```
def recursion(n):
    # do something before recursion
    if n is "trivial":
        return "solution" # or just do nothing
    else:
        sol = recursion(n - 1)
    # do something after recursion
    return n + sol # construct solution
```

Rekurzivní algoritmy





Rekurzivní řazení

- Posloupnost je seřazená právě tehdy, když
  - $|A| < 2$,
  - $|A| \geq 2, val(a_1) \leq val(a_2)$ a posloupnost $\langle a_2 \ldots a_n \rangle$ neobsahuje prvek a_1 a je seřazená

- Rekurzivní algoritmus
  - Jednoprvková množina je seřazená
  - Víceprvkovou množinu rozdělíme a seřadíme po částech
  - Řešení vzniká
    - správným rozdělením množiny
    - Správným spojením výsledků

Jak by vypadala rekurzivní implementace algoritmů z předchozích přednášek?

Merge Sort

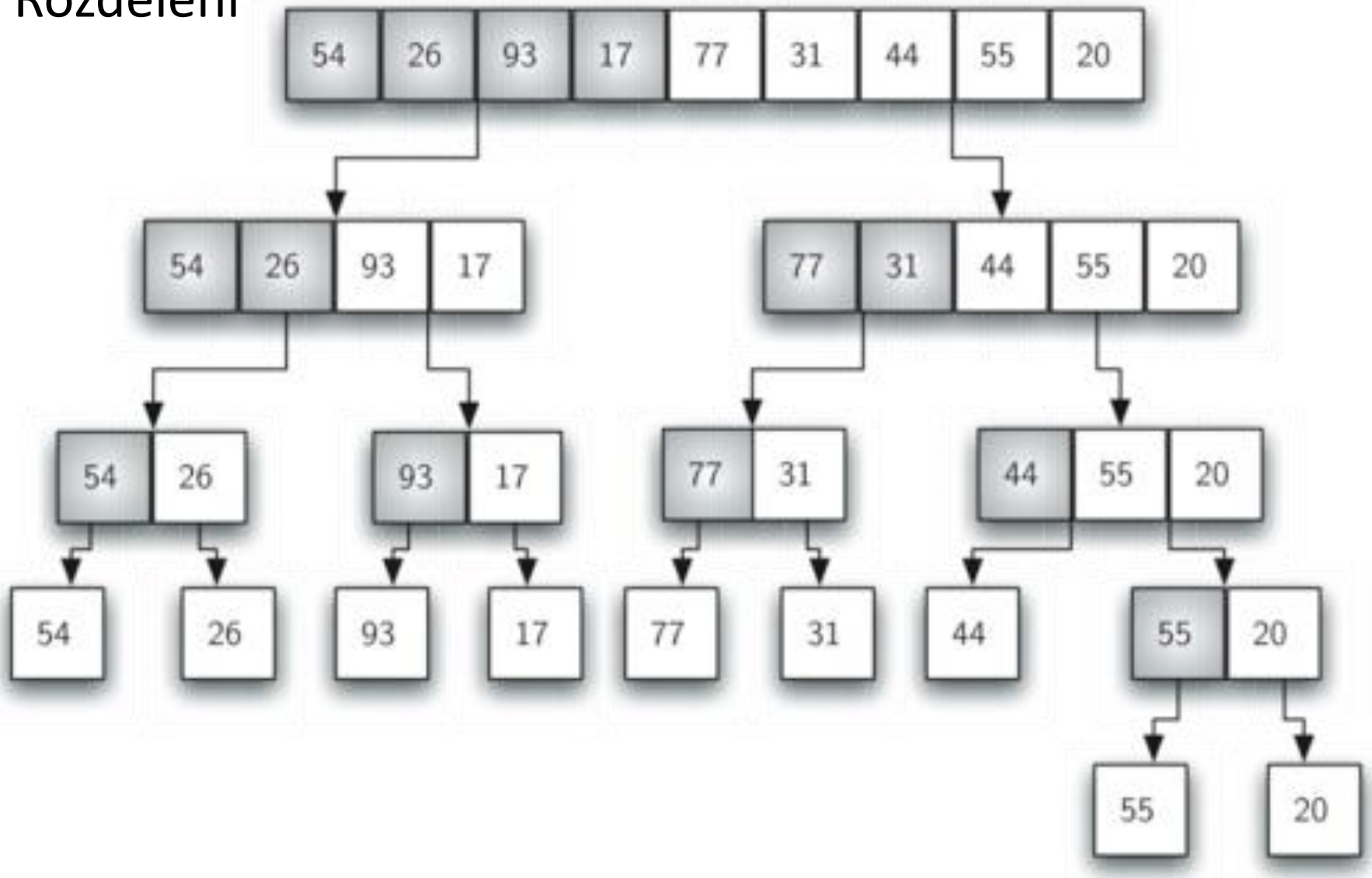
Merge Sort

- Řazení pomocí rozdělování a slučování
- Postupně rozděluje seznam na poloviny
- Prázdný nebo jednoprvkový seznam je setříděný
- Pokud máme dvě poloviny setříděné, spojíme je operací **merge**
- Postupně spojujeme setříděné části, dokud nemáme celý seznam
- Je stabilní

- Rekurzivní algoritmus – výsledek vzniká pomocí volání stejného algoritmu na menší části

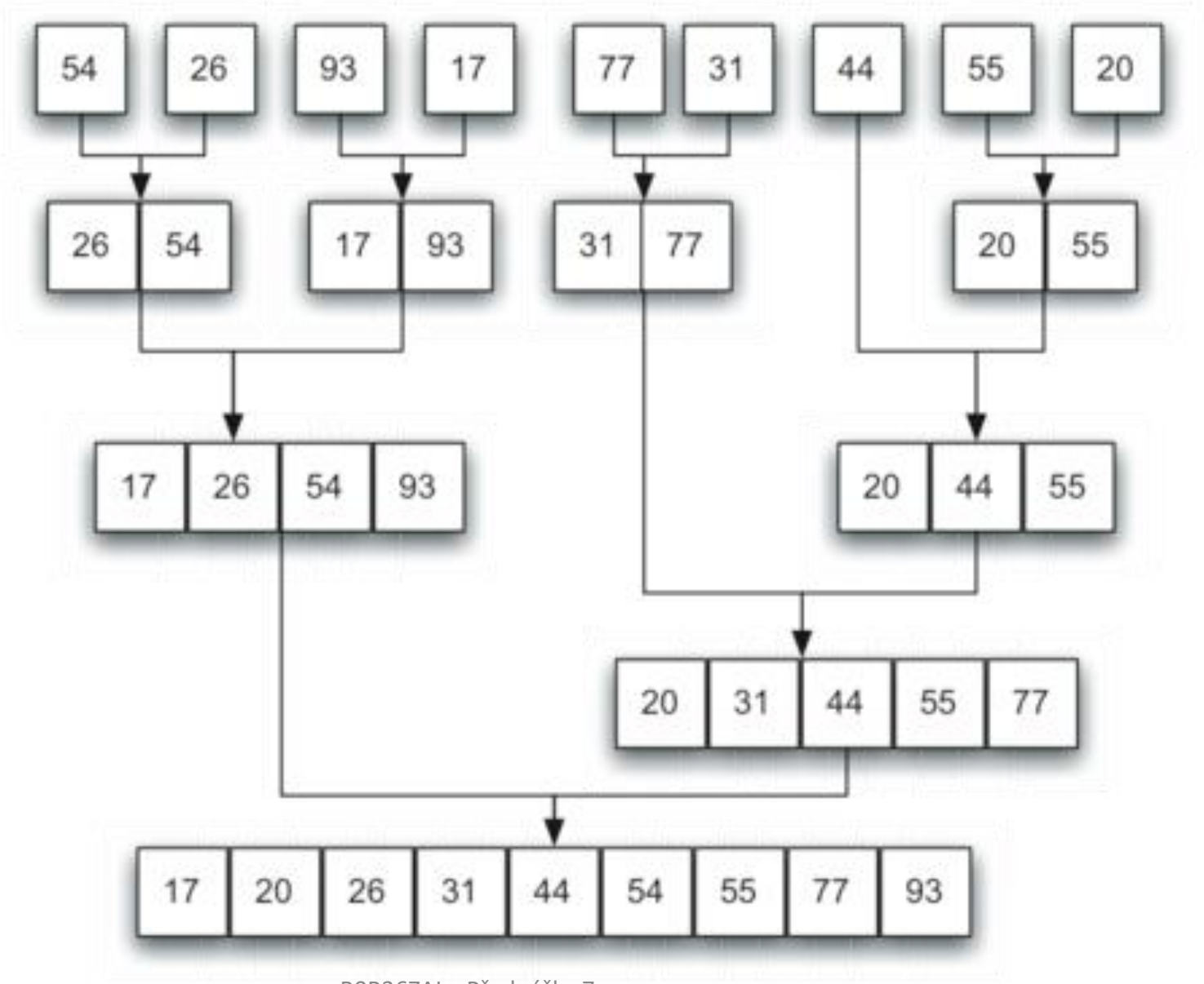
Merge Sort

- Rozdělení



Merge Sort

- Sloučení



Merge Sort

```
def mergeSort(array):
    if len(array) < 2:
        return array
    m = len(array) // 2
    return merge(mergeSort(array[:m]), mergeSort(array[m:]))


def merge(leftPart, rightPart):
    result = []
    i = j = 0
    while i < len(leftPart) and j < len(rightPart):
        if leftPart[i] <= rightPart[j]:
            result.append(leftPart[i])
            i += 1
        else:
            result.append(rightPart[j])
            j += 1
    result += leftPart[i:]
    result += rightPart[j:]
    return result
```

Merge Sort

- Rozdělování končí na jednoprvkové množině
  - Ta je z definice seřazená
- Vhodný na data se sekvenčním přístupem (např. data větší než operační paměť)
- Vhodný pro spojové seznamy
- Vhodný pro paralelní zpracování

Ukážeme si v předmětu PJV

- Možnost využití v kombinaci jinými algoritmy
  - Pro seznamy menší než zvolené *n* využijeme jiný algoritmus
  - Můžeme definovat maximální počet rozdělení (zanoření) v závislosti na velikosti vstupních dat
- Merge sort + insertion sort = timsort

Python 2.3, Java SE 7, Android, …

Quick Sort

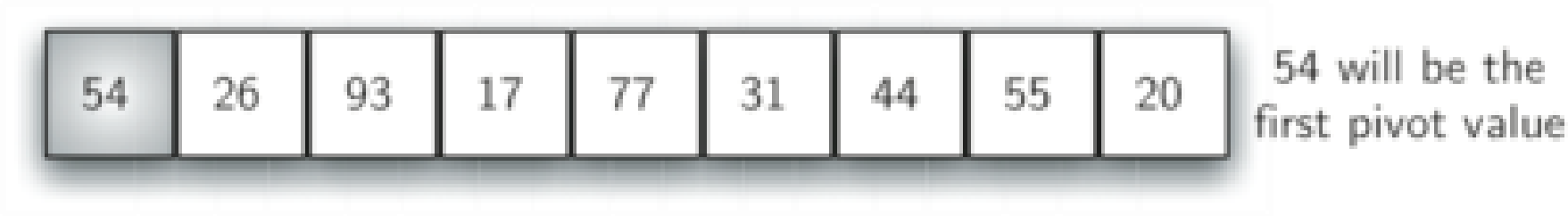
Quick Sort

- Řazení pomocí rozdělování a slučování
- Rozděluje seznam na základě **pivot**a *x*
- Nalevo od prvku *x* umístíme všechny prvky menší než *x*
- Napravo od prvku *x* umístíme všechny prvky větší než *x*
- Rozdělení se opakuje pro seznam nalevo a napravo od *x*
- Postup opakujeme dokud nerozdělujeme jednoprvkové úseky
- Není stabilní

- Rekurzivní algoritmus – výsledek vzniká pomocí volání stejného algoritmu na menší části

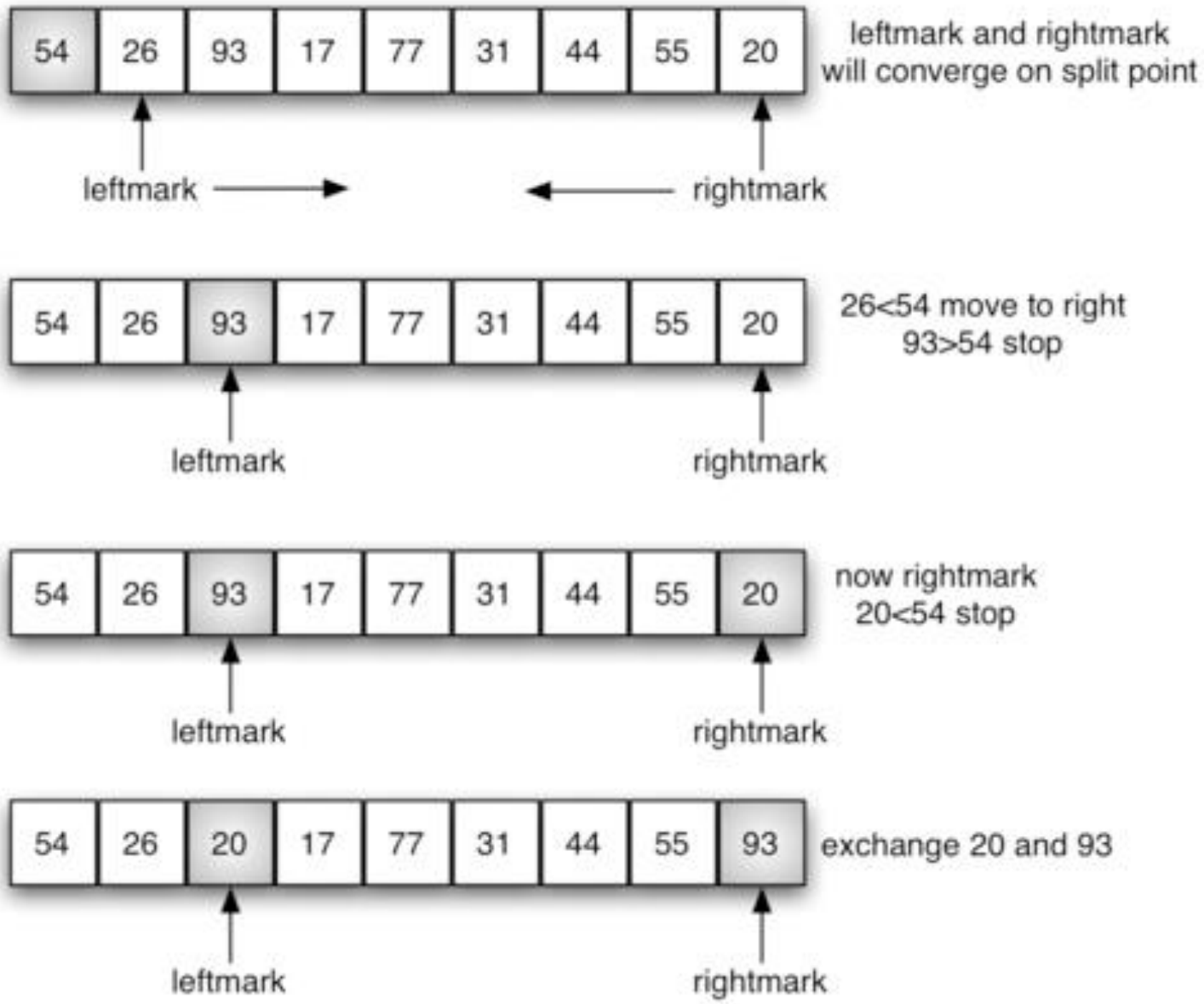
Quick Sort

- Výběr pivota – ovlivňuje výkonost řazení
  - První prvek
  - Medián ze tří (první, střední, poslední)

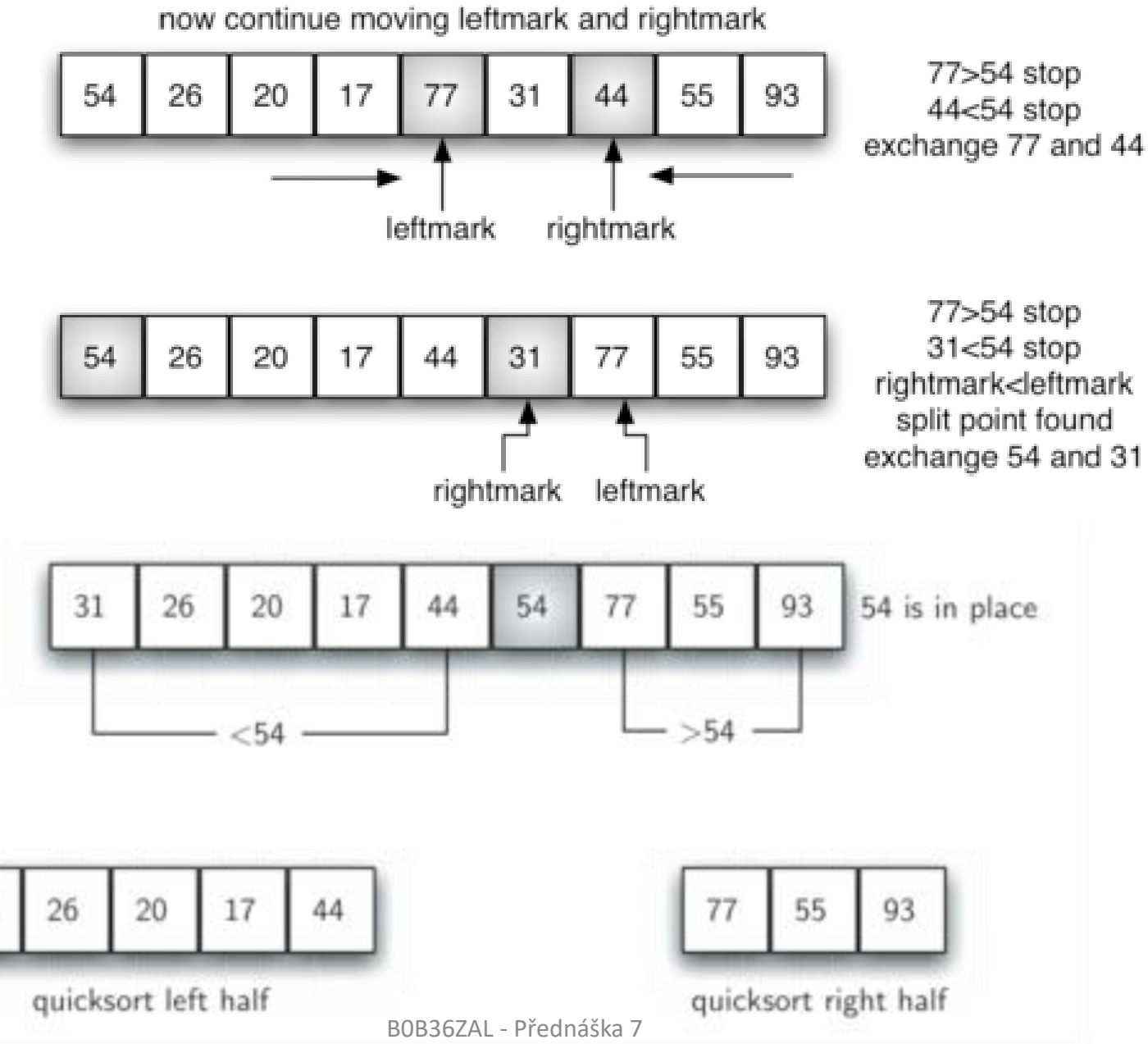


- Postupné přerovnávání prvků a hledání bodu pro rozdělení

Quick Sort


54 26 93 17 77 31 44 55 20
leftmark and rightmark will converge on split point
leftmark
rightmark
54 26 93 17 77 31 44 55 20
26<54 move to right
93>54 stop
leftmark
rightmark
54 26 93 17 77 31 44 55 20
now rightmark
20<54 stop
leftmark
rightmark
54 26 20 17 77 31 44 55 93
exchange 20 and 93
leftmark
rightmark

Quick Sort


now continue moving leftmark and rightmark
54 26 20 17 77 31 44 55 93
leftmark
rightmark
77>54 stop
44<54 stop
exchange 77 and 44
54 26 20 17 44 31 77 55 93
rightmark
leftmark
77>54 stop
31<54 stop
rightmark<leftmark
split point found
exchange 54 and 31
31 26 20 17 44 54 77 55 93
54 is in place
<54
>54
31 26 20 17 44
quicksort left half
77 55 93
quicksort right half

Quick Sort

```
def quickSort(array):
    quickSortHelper(array,0,len(array)-1)

def quickSortHelper(array,first,last):
    if first<last:
        splitpoint = partition(array,first,last)
        quickSortHelper(array,first,splitpoint-1)
        quickSortHelper(array,splitpoint+1,last)
```

Quick Sort

```
def partition(array,first,last):
    pivotvalue = array[first]
    left = first+1
    right = last

    while True:
        while left <= right and array[left] <= pivotvalue:
            left += 1
        while right >= left and array[right] >= pivotvalue:
            right -= 1
        if right < left:
            break
        else:
            array[left],array[right] = array[right], array[left]

    array[first],array[right] = array[right],array[first]
    return right
```

Quick Sort

- Rozdělování končí na jednoprvkové množině
  - Ta je z definice seřazená
- Vhodný na „in-memory“ data – hodně náhodných přístupů v rámci celé množiny dat
- Využívá záměn v rámci vstupní datové struktury (nepotřebuje pomocné uložení dat)
- Považován za „nejrychlejší“
- Citlivý na volbu pivota
  - Mnoho alternativních způsobů výběru
- Možná detekce „špatné situace“ a změna algoritmu za běhu
- Quick sort + heap sort = introsort
  - Změna alg. po překročení logaritmického počtu zanoření

C++ Standard Library

Základy algoritmizace

- Dnes:
  - Rekurzivní řazení
    - Merge sort
    - Quick sort

Příště hledání v grafech