Základy algoritmizace

7. Vyhledávání a řazení 2

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

Základy algoritmizace

- Dnes:
  - Merge sort
  - Quick sort
  - Heap sort

Merge Sort

Merge Sort

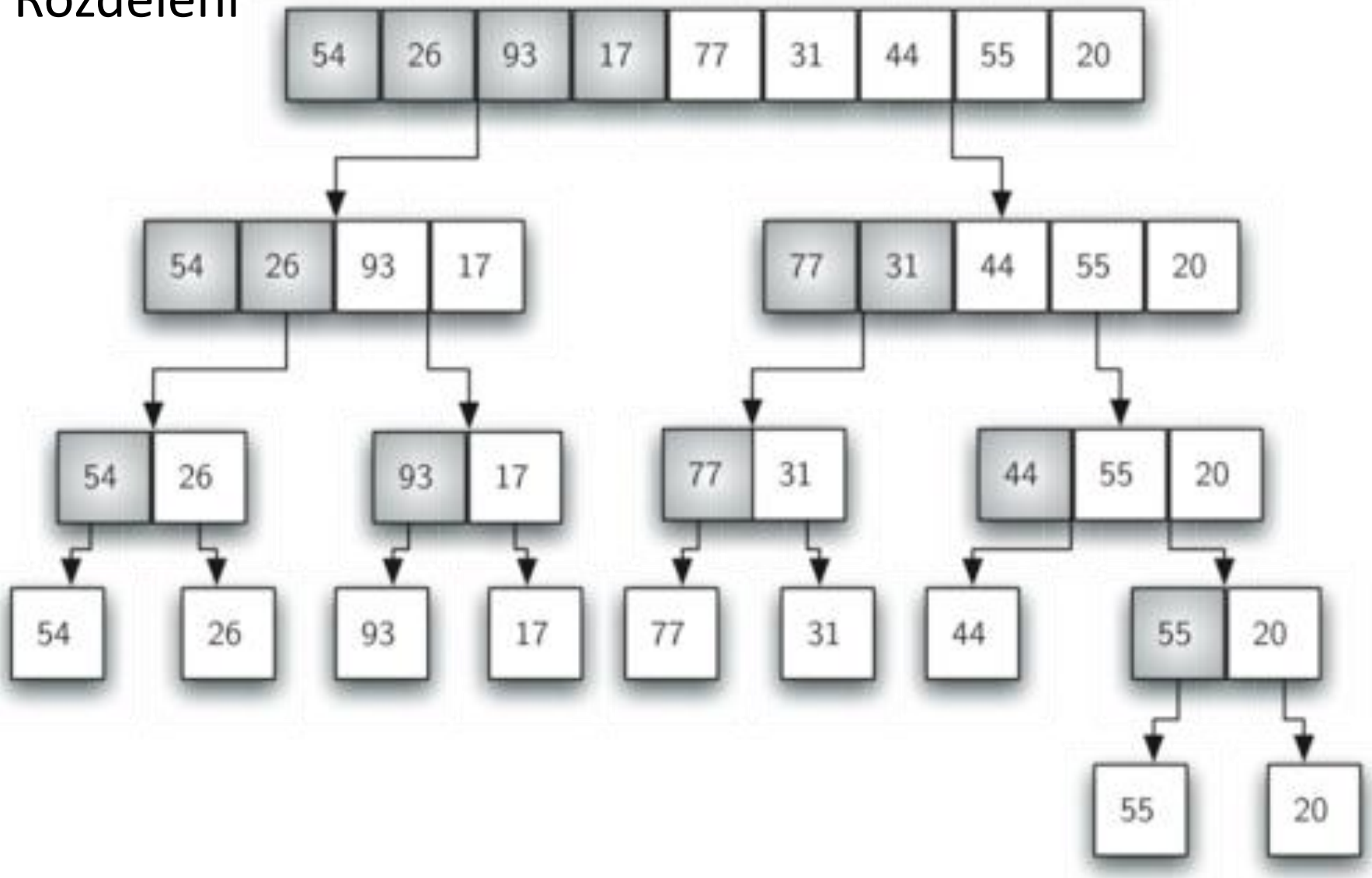
- Řazení pomocí rozdělování a slučování
- Postupně rozděluje seznam na poloviny
- Prázdný nebo jednoprvkový seznam je setříděný
- Pokud máme dvě poloviny setříděné, spojíme je operací **merge**
- Postupně spojujeme setříděné části, dokud nemáme celý seznam
- Je stabilní

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

Více o rekurzi na příští přednášce

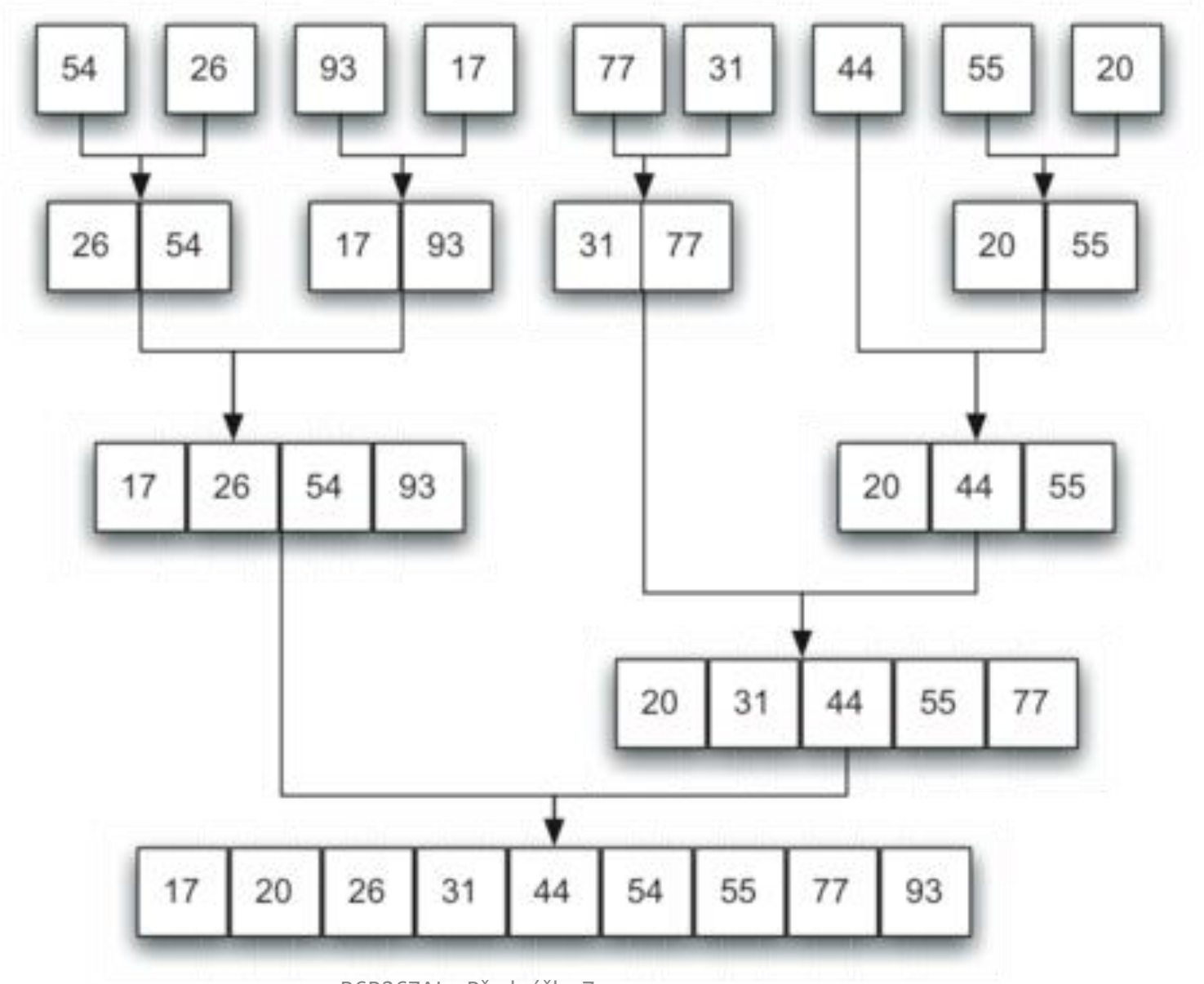
Merge Sort

- 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
```

Quick Sort

Quick Sort

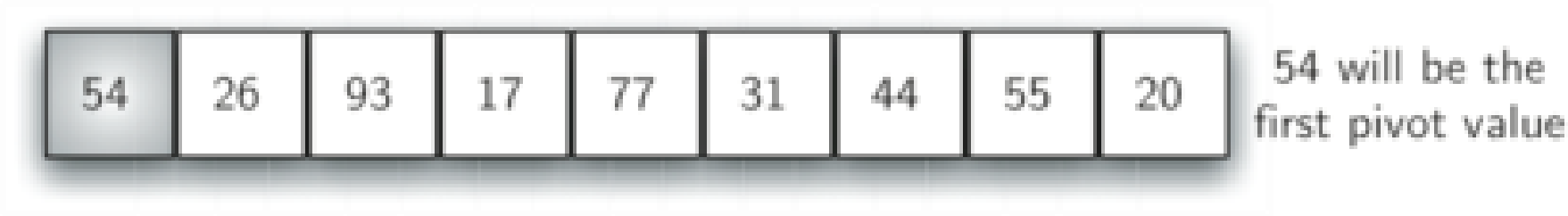
- Řazení pomocí rozdělování a slučování
- Rozděluje seznam na základě **pivot**a *x*
- Nalevo od prvku *x* umístíme všechny prvky menší než *x*
- Napravo od prvku *x* umístíme všechny prvky větší než *x*
- Rozdělení se opakuje pro seznam nalevo a napravo od *x*
- Postup opakujeme dokud nerozdělujeme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

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

Více o rekurzi na příští přednášce

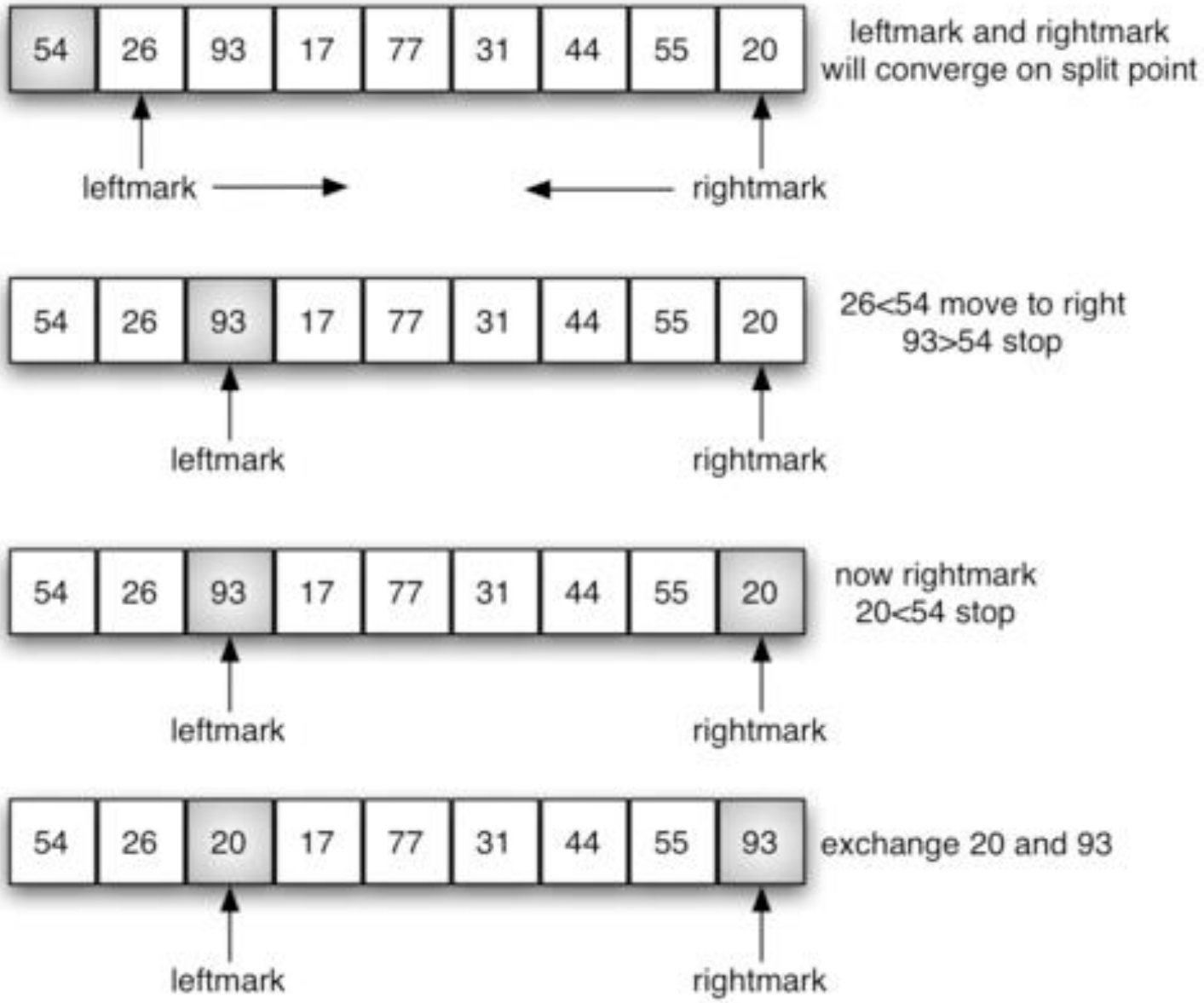
Quick Sort

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

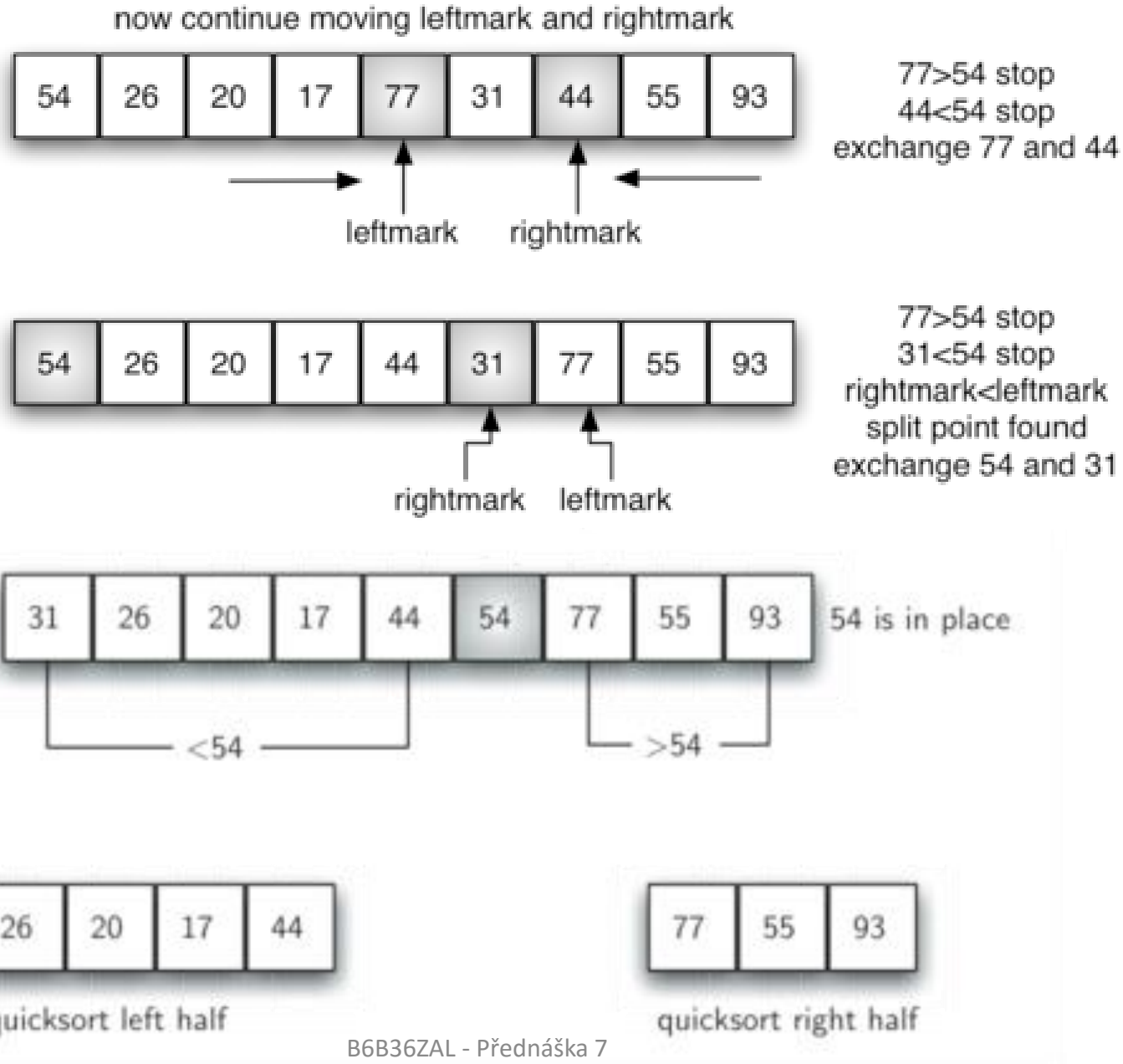


- Postupné přerovnávání prvků a hledání bodu pro rozdělení

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
>54
54 is in place
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
```

Heap Sort

Heap Sort

- Řazení pomocí převodu na binární haldu
  - Binární strom, kde každý prvek je větší než jeho následovníci
  - Je vyvážený
  - Každý prvek je co nejvíce vlevo
- Implementujeme pomocí pole, kde potomci prvku *n* jsou na pozici 2n+1 a 2n+2
- Vyjímáním max. elementu budujeme setříděný seznam
  - Pokud není halda prázdná, zaměníme první a poslední prvek
  - „spravíme“ haldu
- Není stabilní

Heap Sort

- Tvorba haldy
  - Halda – pole prvků $h_1, \dots, h_n$, kde $h_{n/2-1}, \dots, h_n$ jsou listy
  - Je třeba zajistit vlastnosti haldy – pro $h_{n/2-2}$ až h_1 kontrolujeme přípustnost – výměna prvků
- Po výměně prvku je třeba opět kontrola haldy
  - Kontrolujeme přípustnost pro h_1
- Zajištění přípustnosti
  - pokud je daný prvek menší než některý z jeho následníků, vyměníme je a kontrolujeme přípustnost vyměněného následníka

Heap Sort

```
    def heapSort(array):
      for start in range((len(array)-2)//2, -1, -1):
        movedown(array, start, len(array)-1)

      for end in range(len(array)-1, 0, -1):
        array[end], array[0] = array[0], array[end]
        movedown(array, 0, end - 1)

def movedown(array, start, end):
  root = start
  while True:
   child = root * 2 + 1
   if child > end: break
   if child + 1 <= end and array[child] < array[child + 1]:
     child += 1
   if array[root] < array[child]:
     array[root], array[child] = array[child], array[root]
     root = child
   else:
     break
```

Heap Sort

- Vstup

```
[26,81,54,1,93,65,58,74,36,18,9,95,75,58,61,42]
```

- Heap

```
[95,93,75,74,81,65,61,42,36,18,9,26,54,58,58,1]
```

- Výstup

```
[1,9,18,26,36,42,54,58,58,61,65,74,75,81,93,95]
```

Řazení

- Porovnání algoritmů
  - Časová náročnost (best, average, worst)

Porovnávací algoritmy nemohou být lepší než $n \cdot log\ n$

  - Paměťová náročnost (memory)
  - Stabilita řazení
- Vše v „big O“ notaci

Více se tomuto tématu budeme věnovat v druhé půlce semestru

Algoritmus	Best	Average	Worst	Memory	Stable
Merge sort	n log n	n log n	n log n	n	Yes
Quick sort	n log n	n log n	n^2	log n	No
Heap sort	n log n	n log n	n log n	1	No

Základy algoritmizace

- Dnes:
  - Řazení
    - Merge sort
    - Quick sort
    - Heap sort

Příště rekurze