Řadicí algoritmy

Domrachev Danil
(xdomra00)

8. května 2022

Teorie

Algoritmus – konečná množina jednoduchých determinovaných kroků vedoucích k požadovanému cíli.

Teorie

Algoritmus – konečná množina jednoduchých determinovaných kroků vedoucích k požadovanému cíli.

Libovolný algoritmus lze zapsat pomocí tří komponent

Teorie

Algoritmus – konečná množina jednoduchých determinovaných kroků vedoucích k požadovanému cíli.

Libovolný algoritmus lze zapsat pomocí tří komponent

- Sekvence

Teorie

Algoritmus – konečná množina jednoduchých determinovaných kroků vedoucích k požadovanému cíli.

Libovolný algoritmus lze zapsat pomocí tří komponent

- Sekvence
- Selekce

Teorie

Algoritmus – konečná množina jednoduchých determinovaných kroků vedoucích k požadovanému cíli.

Libovolný algoritmus lze zapsat pomocí tří komponent

- Sekvence
- Selekce
- Iterace

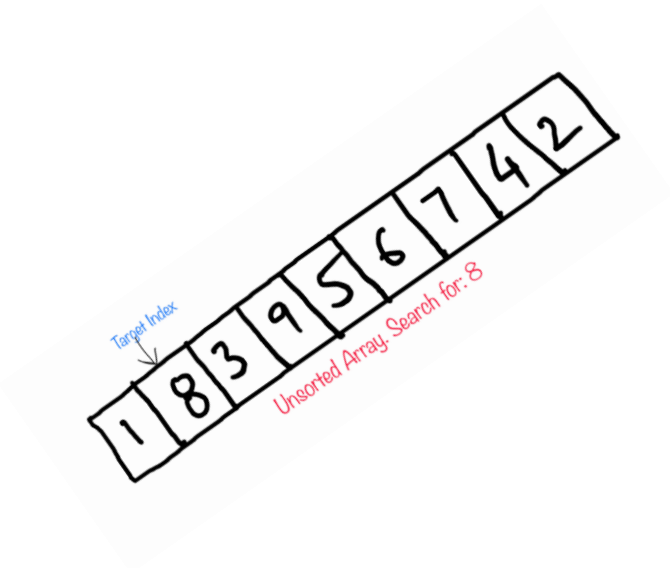
Proč vůbec něco řadit?

Obvykle samotné řazení není konečným cílem, slouží k realizaci dalších algoritmů jako:

Proč vůbec něco řadit?

Obvykle samotné řazení není konečným cílem, slouží k realizaci dalších algoritmů jako:

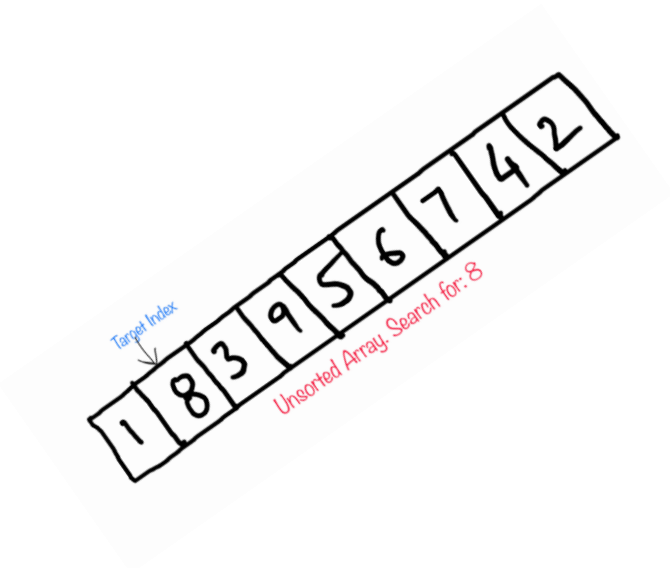
Algoritmy vyhledávání,

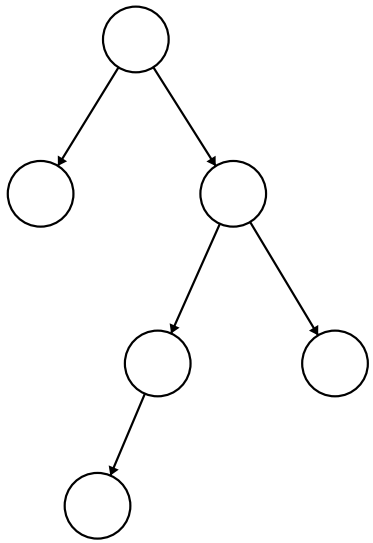


Proč vůbec něco řadit?

Obvykle samotné řazení není konečným cílem, slouží k realizaci dalších algoritmů jako:

Algoritmy vyhledávání, algoritmy datových struktur atd.

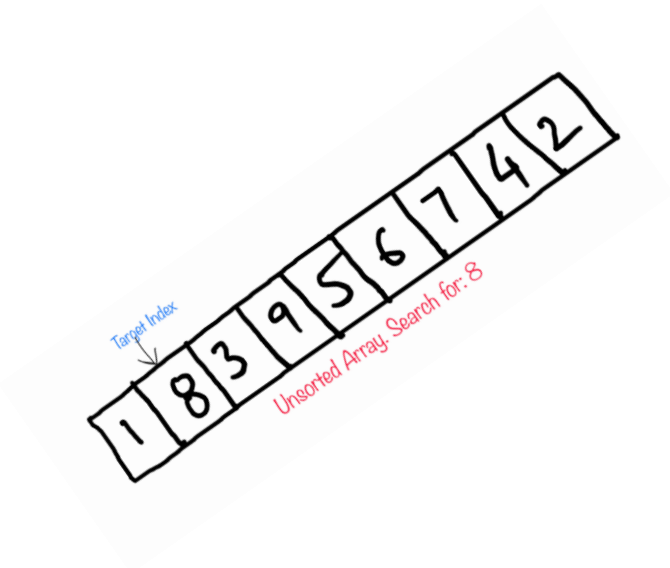


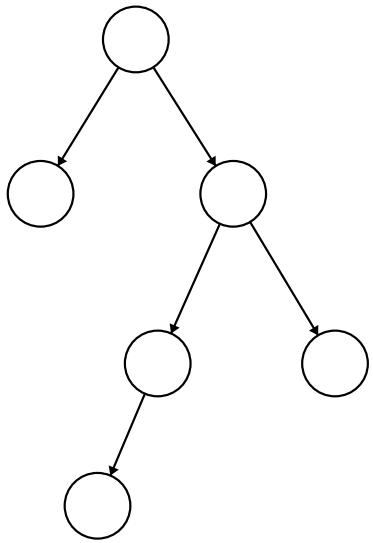


Proč vůbec něco řadit?

Obvykle samotné řazení není konečným cílem, slouží k realizaci dalších algoritmů jako:

Algoritmy vyhledávání, algoritmy datových struktur atd.





Využitím seřazeného pole zvyšujeme jejich efektivitu

Druhy řadicích algoritmů

Name	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity	Stability
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	O(1)	Stable
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	O(1)	Unstable
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	O(1)	Stable
Merge Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	O(n)	Stable
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	O(log(n))	Unstable
Heap Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	O(1)	Unstable
Counting Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	O(k)	Stable
Radix Sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	O(n+k)	Stable

https://medium.com/@bill.shantang/8-classical-sorting-algorithms-d048eec3fdab

Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová

Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová
- Způsob seřazení – obvykle porovnávím, ale existují i jiné, např. Counting Sort

Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová
- Způsob seřazení – obvykle porovnávím, ale existují i jiné, např. Counting Sort
- Paralelita – dává možnost využít více procesorů současně pro zrychlení algoritmu

Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová
- Způsob seřazení – obvykle porovnávím, ale existují i jiné, např. Counting Sort
- Paralelita – dává možnost využít více procesorů současně pro zrychlení algoritmu
- Přizpůsobivost – schopnost řadit rychleji na specificky předzpracovávaných datech

Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová
- Způsob seřazení – obvykle porovnávím, ale existují i jiné, např. Counting Sort
- Paralelita – dává možnost využít více procesorů současně pro zrychlení algoritmu
- Přizpůsobivost – schopnost řadit rychleji na specificky předzpracovávaných datech
- Prostorová náročnost – spotřeba paměti

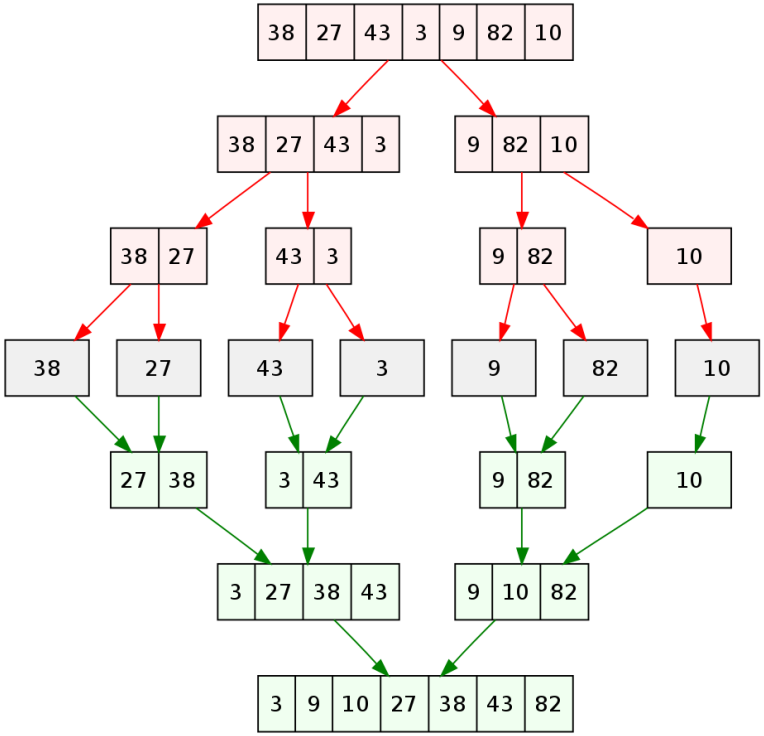
Vlastnosti řadicích algoritmů

- Stabilita – pořadí elementů se stejným klíčem se zachová
- Způsob seřazení – obvykle porovnávím, ale existují i jiné, např. Counting Sort
- Paralelita – dává možnost využít více procesorů současně pro zrychlení algoritmu
- Přizpůsobivost – schopnost řadit rychleji na specificky předzpracovávaných datech
- Prostorová náročnost – spotřeba paměti
- Časová náročnost – rychlost seřazení

Merge sort

Základní myšlenka: Rozděl a panuj

Drobíme pole až zůstanou skupiny ze dvou či jednoho prvku. Pak je postupně dáváme ve spravném pořadí do větších celků.



Merge sort – Realizace

Algoritmus 1: Help Function Merge()

Input : *array*, *first*, *last*

```
1:  temp_array = [ ]
2:  middle = (first + last)/2
3:  begin = first
4:  end = middle + 1
5:  for i = last to end do
6:      if (begin ≤ middle) & ((end > last) || (array[begin] < array[end])) then
7:          temp_array[i] = array[begin]
8:          begin = begin + 1
9:      end if
10:     else
11:         temp_array[i] = array[end]
12:         end = end + 1
13:     end if
14: end for
15: for i = begin to end do
16:     array[i] = temp_array[i]
17: end for
```

Merge sort – Realizace

Algoritmus 2: Main Function MergeSort()

Input : *array*, *first*, *last*

1: **if** *first* $<$ *last* **then**
2: *MergeSort*(*array*, *first*, (*first* + *last*)/2)
3: *MergeSort*(*array*, (*first* + *last*)/2 + 1, *last*)
4: *Merge*(*array*, *first*, *last*)
5: **end if**