Sorting

Svarny Petr

Katedra logiky FF UK

21. března 2022

Overview

Zajimavost

Rekurze

Sorting

GTA a JSON

Problém s GTA kvadratičností:
https://www.mattkeeter.com/blog/2021-03-01-happen/
https://nee.lv/2021/02/28/
How-I-cut-GTA-Online-loading-times-by-70/ JSON

GTA a chyba ve čtení JSONu

TL;DR verze:

- Loading GTA trvalo moc dlouho.
- Analýza ukázala, že problém byl načítání JSONu a deduplikace.
- Při načítání ca 63k vstupů docházelo ke kontrole každého prvku!

Python vs C++

Ignorování konstantních faktorů se může vymstít!
TL;DR verze:

- Porovnávání má různou složitost.
- Rozdíl mezi porovnáními:
  - 2-porovnání (e.g. vrací True ptk. jeden objekt je menší než druhý, jinak False)
  - 3-porovnání (vrací -1 pro méně než, +1 pro větší než, 0 jinak)
- Python má dnes dvoj-, C++ má troj-

`https://arxiv.org/abs/1911.12338`

Rekurze - úloha

- Násobení vs sčítání

Není rekurze jako rekurze

- **Lineární** - jen disjunktní, ne současná, rekurzivní volání (faktoriál).
- **Koncová (tail)** - rekurze je poslední příkaz (např. Euclidův algoritmus největšího společného dělitele).
- **Kaskádní** - v tělě jsou alespoň dvě rekurzivní volání (Fibonacci).
- **Vnořená** - argumenty funkce rekurzivně dané! (Ackermannova funkce)

Sorting

`https://www.toptal.com/developers/sorting-algorithms`
Uvědomte si také, jak sami zpracováváte větší množství dat (např. počítání mincí) - často optimalizace na omezenou "RAM".

Divide and conquer

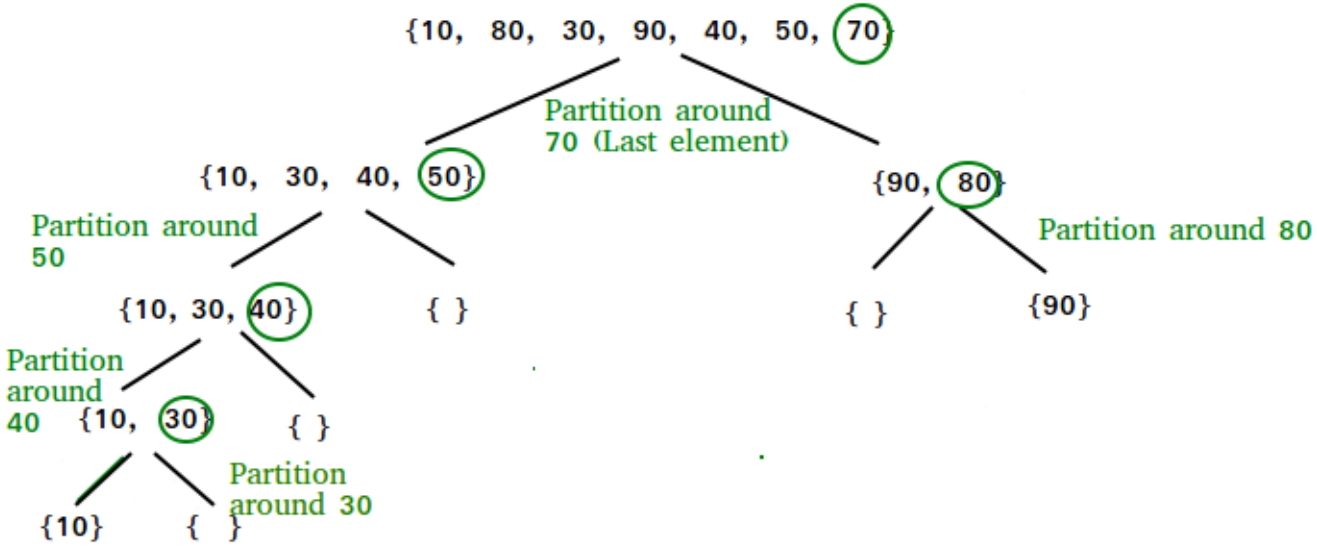
Dělení na stejné díly:

- **QUICK-SORT** - rozděl pomocí určeného prvku na první polovinu dat co je menší než druhá polovina, řazení dělením
- **MERGE-SORT** - rozdělíme na poloviny a při slučování řešíme celkové pořadí, řazení slučováním

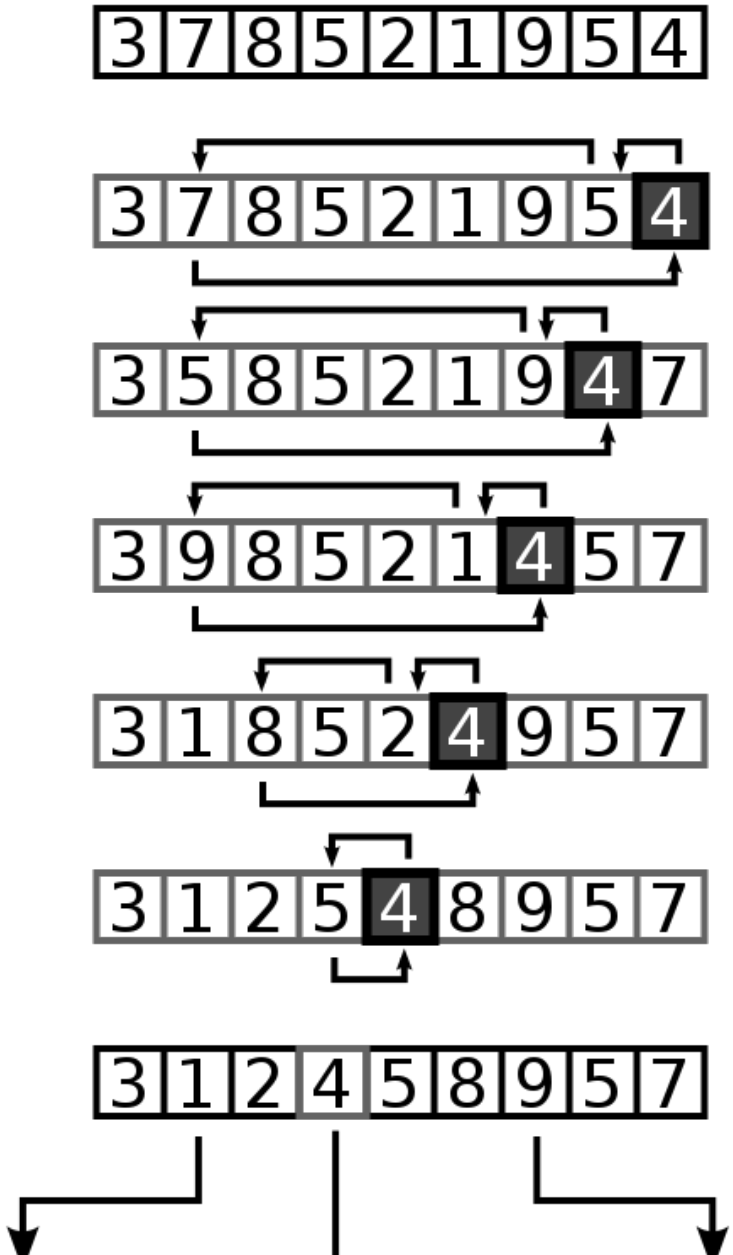
Dělení na prvek a zbytek:

- **SELECTION-SORT** - ze vstupní posloupnosti najít minimální prvek a vložíme do čela výstupní posloupnosti – řazení přímým výběrem
- **INSERTION-SORT** - ze vstupní posloupnosti oddělíme, který potom zařadíme do výsledku na správné místo – řazení zatřiďováním

Quick-sort


{10, 80, 30, 90, 40, 50, 70}
Partition around
70 (Last element)
{10, 30, 40, 50}
{90, 80}
Partition around
50
Partition around 80
{10, 30, 40}
{ }
{ }
{90}
Partition
around
40
{10, 30}
{ }
Partition
around 30
{10}
{ }

Quick-sort


3 7 8 5 2 1 9 5 4
3 7 8 5 2 1 9 5 4
3 5 8 5 2 1 9 4 7
3 9 8 5 2 1 4 5 7
3 1 8 5 2 4 9 5 7
3 1 2 5 4 8 9 5 7
3 1 2 4 5 8 9 5 7

Quick-sort

```
def quick_sort(arr, i, j):
    if i<j:
        pivot = select_pivot(arr, i, j)
        mid = partition(arr, i, j, pivot)
        quick_sort(arr, i, mid)
        quick_sort(arr, mid+1, j)
```

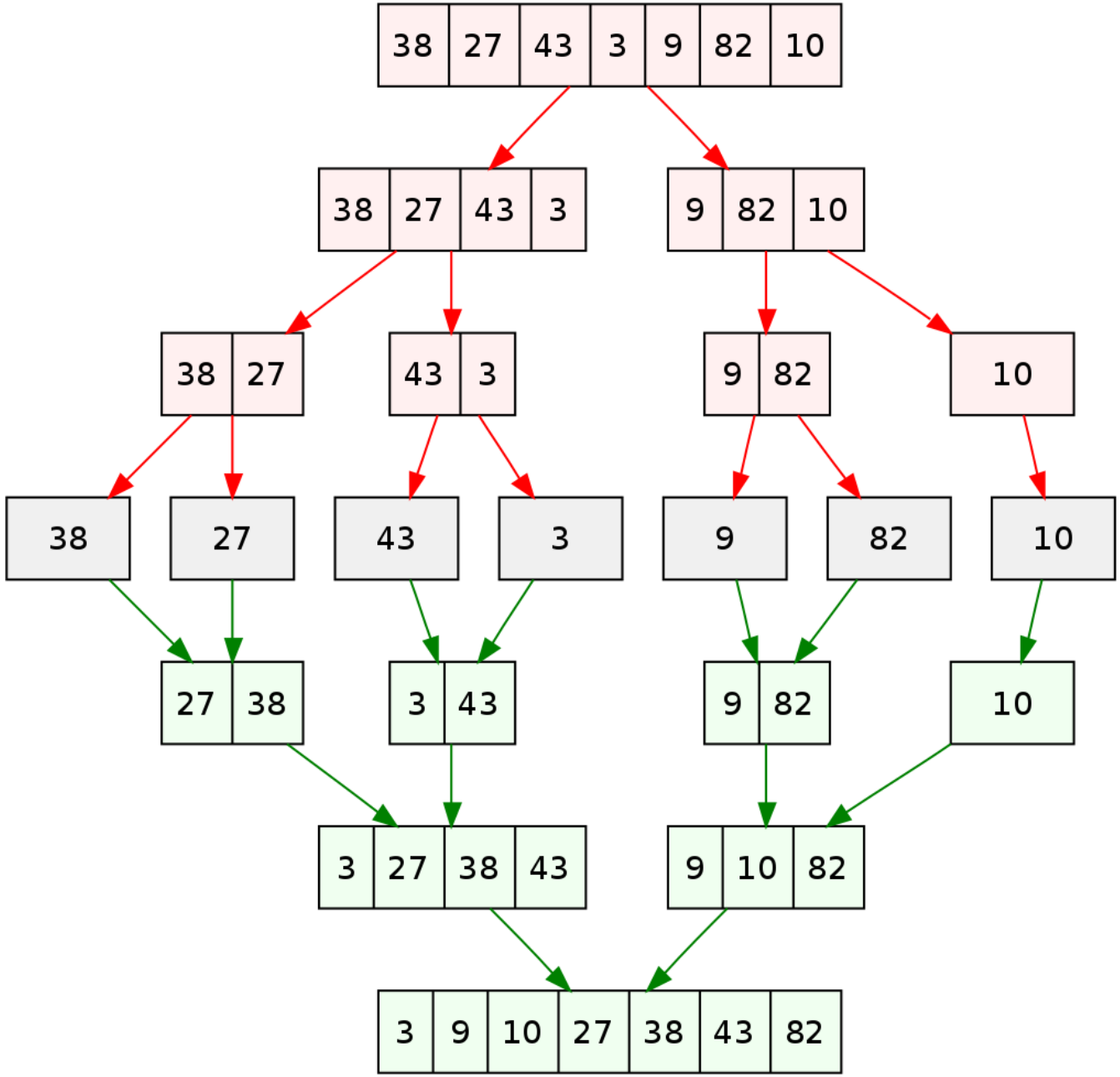
Quick-sort

```
def quick_sort(arr, i, j):
    while i<j:
        pivot = select_pivot(arr, i, j)
        mid = partition(arr, i, j, pivot)
        quick_sort(arr, i, mid)
        i = mid + 1
```

Quick-sort vlastnosti

- Časová: $O(nlogn)$
- Prostorová: $O(logn)$
- Může však mít i $O(n^2)$ pokud se zvolí špatný pivot (a tedy následné dělení).

Merge-sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Merge-sort

```
def merge_sort(arr, i, k):
    if i<k:
        j = floor((i+k)/2)
        merge_sort(arr, i, j)
        merge_sort(arr, j+1, k)
        merge(arr, i, j, k)
```

Merge-sort vlastnosti

- Časová: $\Theta(nlogn)$
- Prostorová: $O(n)$

Selection-sort

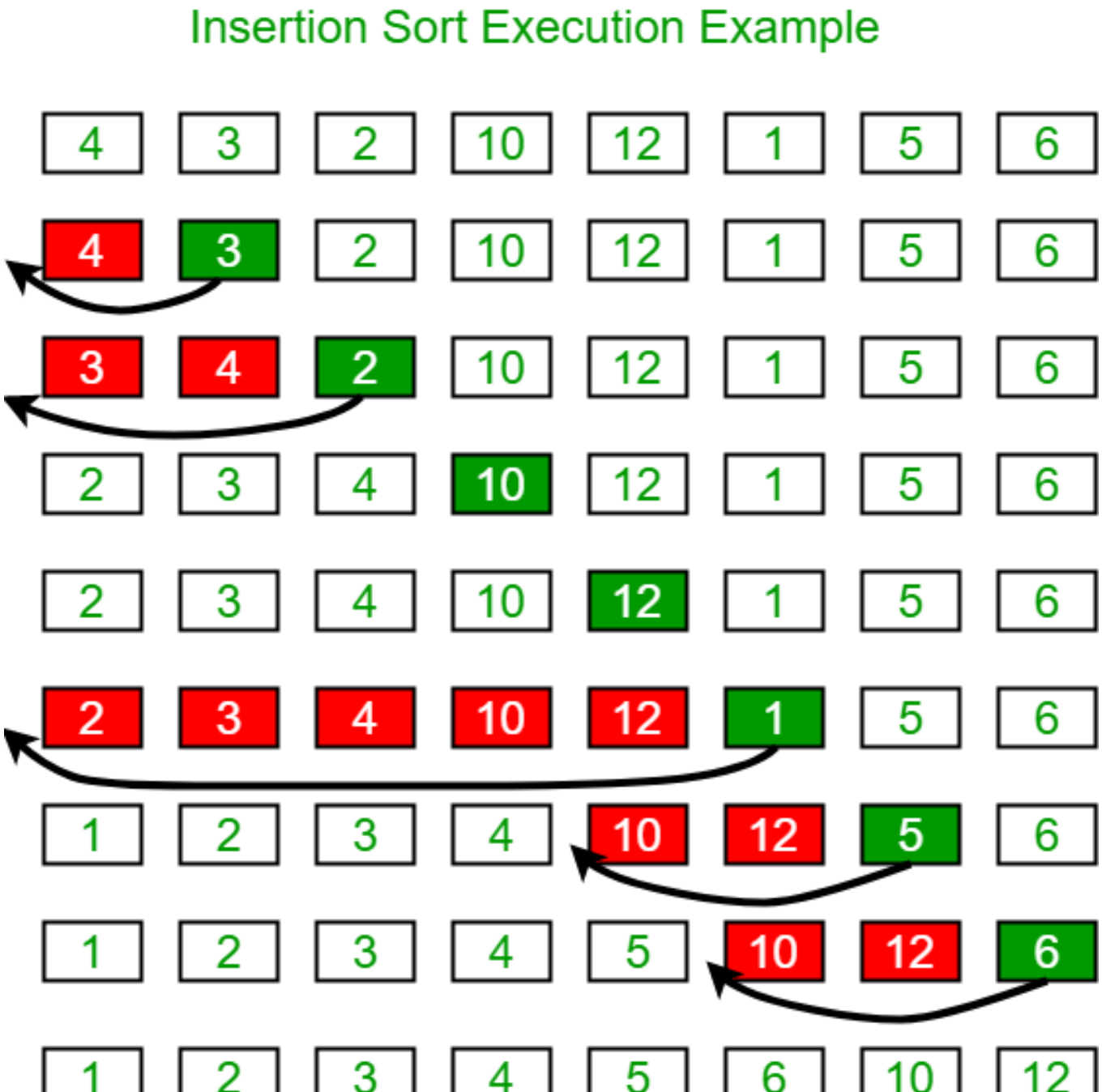
```
for i in range(len(A)):
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    A[i], A[min_idx] = A[min_idx], A[i]
```

Selection-sort vlastnosti

- Časová: $\Theta(n^2)$
- Prostorová: $O(1)$

Insertion-sort



Insertion-sort

```
for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key
```

Insertion-sort vlastnosti

- Časová: $O(n^2)$
- Prostorová: $O(1)$
- Používáno v Quick-sort implementacích.
- "Opak" selection-sort, hledá od prvku zpět.