Quick Sort

Patří k pravděpodobnostním algoritmům (volba „pivota“ ovlivňuje rychlost)

- Vlastnosti:

- Očekávaná doba běhu O(nlog(n)).
- Doba běhu v nejhorším případě $O(n^2)$.
- V praxi funguje výborně!

Budeme předpokládat, že prvky pole A jsou různé.

Quicksort

Chceme seřadit toto pole.

7	6	3	5	1	2	4

Nejprve vybereme "pivota."

Uděláme to náhodně.

Náhodně volený pivot!

Dále rozdělíme pole na části "větší než 5" nebo "menší než 5"

Tento krok ROZĚLENÍ vyžaduje čas O(n). (Všimněte si, že zde neřadíme každou polovinu

Rozdělíme je zase podobně jako nahoře, tedy:

L = pole s elementy menšími než A[pivot]

R = pole s elementy většími než A[pivot]

1	2	3	4	5	6	7

Bude následovat rekurze na L a R:

Pseudo kód pro postup, který jsme právě viděli

- QuickSort(A):
  - **If** len(A) <= 1:
    - **return**
  - Zvolímě nějaké x = A[i] náhodně. Nazveme ho **pivot**.
  - ROZDĚLÍME dále pole A na:
    - L (menší než x) a
    - R (větší než x)
  - Upravíme A na [L, x, R] (to je, upravíme pole A v tomto pořadí)
  - QuickSort(L)
  - QuickSort(R)

Předpokládáme, že prvky pole A jsou různé. Jak by se postup změnil, kdyby některé prvky byly stejné? Zkuste upravit pseudokód.

Pro rekurzivní algoritmy – pro výpočet doby běhu se využívá tzv.

The Master Theorem

- Předpokládejme $a \geq 1, b > 1,$ a také d jsou konstanty (nezávisí na n).
- Předpokládejme $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Potom

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

Tři parametry:

a : počet podproblémů

b : factor podle kterého se sníží velikost vstupu

d : potřebujeme udělat n^d práce, abychom vytvořili všechny podproblémy a zkombinovali jejich řešení.

Jedi mistr Yoda

Doba běhu?

- $T(n) = T(|L|) + T(|R|) + \ O(n)$

- V ideálním světě …
  - Pokud pivot rozdělí pole přesně na půl …

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

- Podle Master teorému:

$$T(n) \ = \ O(n \log(n)).$$

Jak je Quick Sort rychlý?

Příklad rekurzivního volání

Vyber 5 jako pivot

Rozdělení na každou stranu od 5

Rekurze na [3142], vybereme 3 jako pivot.

Rekurze na [76] a výběr 6 jako pivot.

Rozdělení kolem 3.

Rozdělení na každou stranu 6

Rekurze na [12] a výběr 2 jako pivot.

Rekurze na [4] (hotovo).

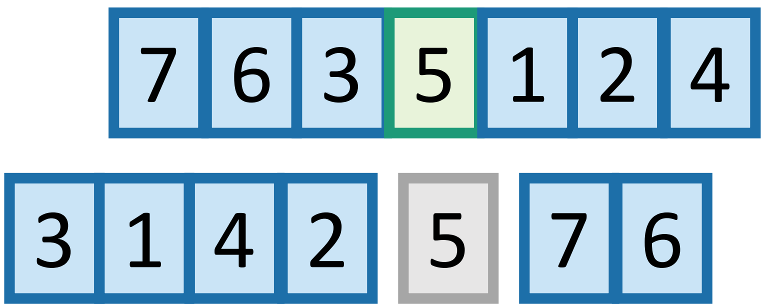
Rekurze na [7], zde je velikost úseku 1, takže jsme hotovi.

Rozdělení kolem 2.

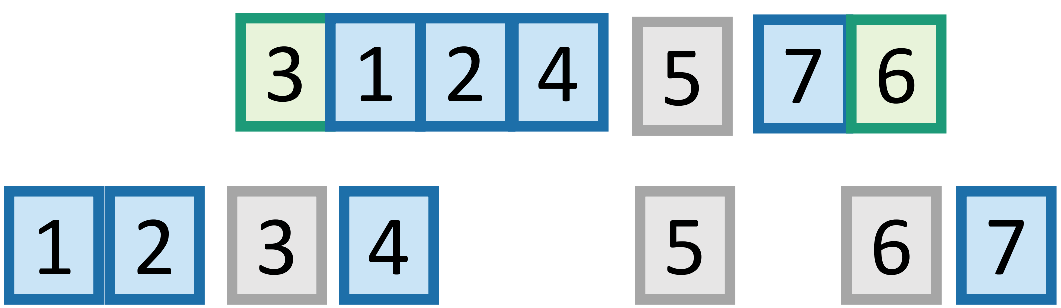
Rekurze na [1] (hotovo).

Jaký je čas běhu?

- Budeme počítát počet srovnání, která algoritmus provádí.
  - Dá nám to dobrou představu o době běhu.
- Kolikrát jsou porovnávány jakékoliv dvě položky?



V předchozím příkladu bylo vše porovnáno s 5 jednou v prvním kroku ... a nikdy znovu.



Ale ne všechno bylo srovnáváno s 3.
5 byla, stejně jako 1,2 a 4. Ale ne 6 nebo 7.

Každá dvojice položek je porovnána buď 0, nebo 1 krát. Který to je?

7	6	3	5	1	2	4

Předpokládejme, že čísla v poli jsou ve skutečnosti čísla 1,…, n

- **Zda jsou nebo ne a,b porovnávány** je náhodná proměnná, která závisí na volbě pivota. Řekněme

$$X_{a,b} = \begin{cases} \mathbf{1} & \textbf{když } a \textbf{ a } b \textbf{ jsou někdy porovnávány} \\ \mathbf{0} & \textbf{když } a \textbf{ and } b \textbf{ nejsou nikdy porovnávány} \end{cases}$$

- V Předchozím případě $\mathbf{X_{1,5} = 1}$, protože položka 1 a položka 5 byly porovnávány.
- Ale $\mathbf{X_{3,6} = 0}$, protože položka 3 a položka 6 NEBYLY porovnávány.

Spočítáme počet porovnávání

- Celkový počet porovnávání během algoritmu je

$$\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{a,b}$$

- Očekávaný počet porovnávání je

$$E\left[\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{a,b}\right] = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[\,X_{a,b}]$$

pomocí linearity očekávání.

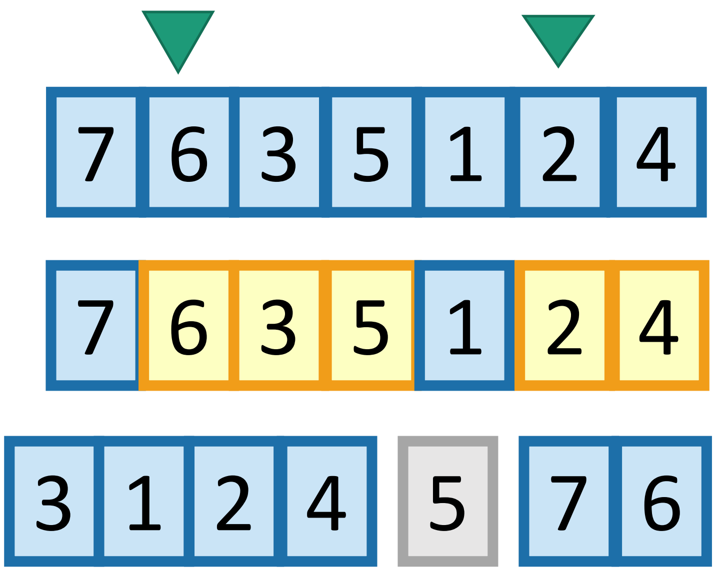
Počet porovnání

očekávaný počet porovnávání :

$$\sum_{a=1}^{n-1} \sum_{b=a+1}^{n} E[\,X_{a,b}]$$

- Musíme tedy jen přijít na $E[\ X_{a,b}\ ]$
- $E\left[X_{a,b}\right] = P(X_{a,b} = 1) \cdot 1 + P(X_{a,b} = 0) \cdot 0 = P(X_{a,b} = 1)$
  - (pomocí definice očekávání)
- Musíme tedy přijít na:

$P(X_{a,b} = 1)$ = pravděpodobnost, že ***a*** a ***b*** jsou porovnávány.



Řekněme, že a = 2 a b = 6. Jaká je pravděpodobnost, že 2 a 6 budou porovnávány?

To je přesně pravděpodobnost, že buď 2 nebo 6 je nejprve vybráno jako pivot ze zvýrazněných položek.

Pokud by například bylo vybrána 5, pak by 2 a 6 byly odděleny a nikdy by se navzájem neviděly.

Počet porovnání

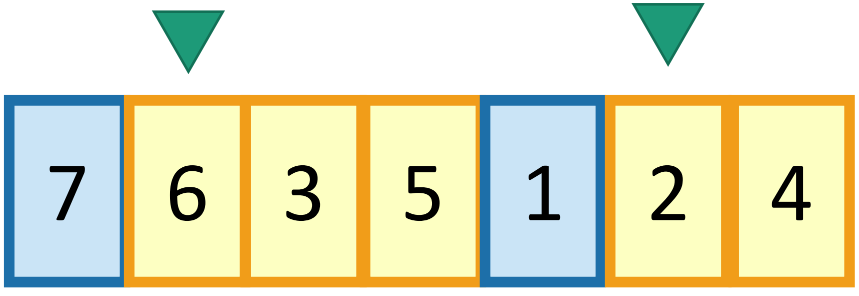
$$P(\,X_{a,b} = 1\,)$$

= pravděpodobnost a, b jsou porovnávány

= pravděpodobnost, že jedno z čísel a, b je vybráno jako první ze všech čísel mezi b – a +1.

2 výběry z b-a+1...

$$= \frac{2}{b - a + 1}$$



Dohromady…

Očekávaný počet porovnávání

- $E\left[\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} X_{a,b}\right]$ Toto je očekávaný počet porovnávání v celém algoritmu
- $= \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} E[\, X_{a,b}]$ linearity of expectation
- $= \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} P(\, X_{a,b} = 1)$ definice očekávání
- $= \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} \frac{2}{b-a+1}$ odvození, které jsme právě udělali

- To není hezký součet, výsledek ale můžeme spočítat.
- Zjistíme, že je to méně než 2n ln(n).

Jsme skoro u konce

- Viděli jsme, že E[počet porovnání] = O(n log(n))
- Je to stejné jako E[doba běhu]?

- V tomto případě, **ano**.
- Tvrdíme, že čas k provedení porovnávání dominuje (je větší než) době běhu.

- QuickSort(A):
  - **If** len(A) <= 1:
    - **return**
  - Vyber náhodně x = A[i]. Nazveme ho pivot.
  - Rozdělíme zbytek A do:
    - L (menší než x) a
    - R (větší než x)
  - Nahradíme A polem [L, x, R] (to je, uspořádáme pole A v tomto pořadí)
  - QuickSort(L)
  - QuickSort(R)

Co jsme zjistili?

- Očekáváná doba běhu Quick Sortu je $O(n\log(n))$

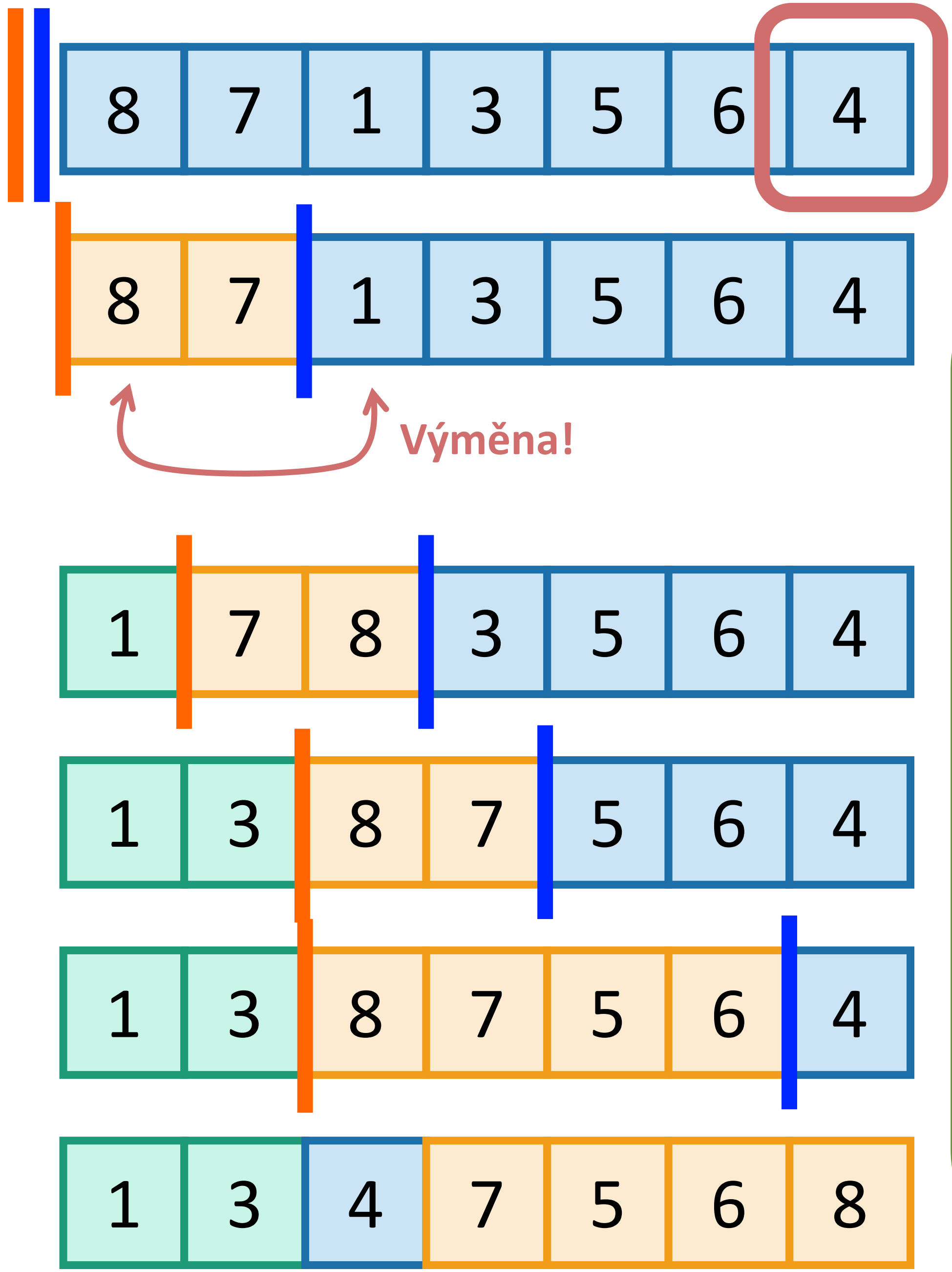
Nejhorší doba běhu

- Předpokládejme, že „protivník“ pro vás vybírá „náhodné“ pivoty.
- Pak by mohla být doba běhu $O(n^2)$
  - To je, zvolili by implementaci SlowSort
  - V praxi se to obvykle nestává

Jak bychom to měli implementovat?

- Náš pseudokód je snadno pochopitelný a analyzovatelný, ale není to dobrý způsob implementace tohoto algoritmu.
    - QuickSort(A):
        - **If** len(A) <= 1:
            - **return**
        - Vyber náhodně x = A[i]. Nazveme ho pivot.
        - Rozdělíme zbytek A do:
            - L (menší než x) a
            - R (větší než x)
        - Nahradíme A polem [L, x, R] (to je, uspořádáme pole A v tomto pořadí)
        - QuickSort(L)
        - QuickSort(R)
- Místo toho jej implementujte na místě (bez samostatných L a R.)
    - Zde je několik maďarských lidových tanečníků, kteří ukazují, jak se to dělá : https://www.youtube.com/watch?v=ywWBy6J5gz8

Lepší způsob rozdělení



Pivot

Vybereme jej náhodně a poté ji vyměníme za poslední prvek, takže je pivot na konci.

Inicializujeme a

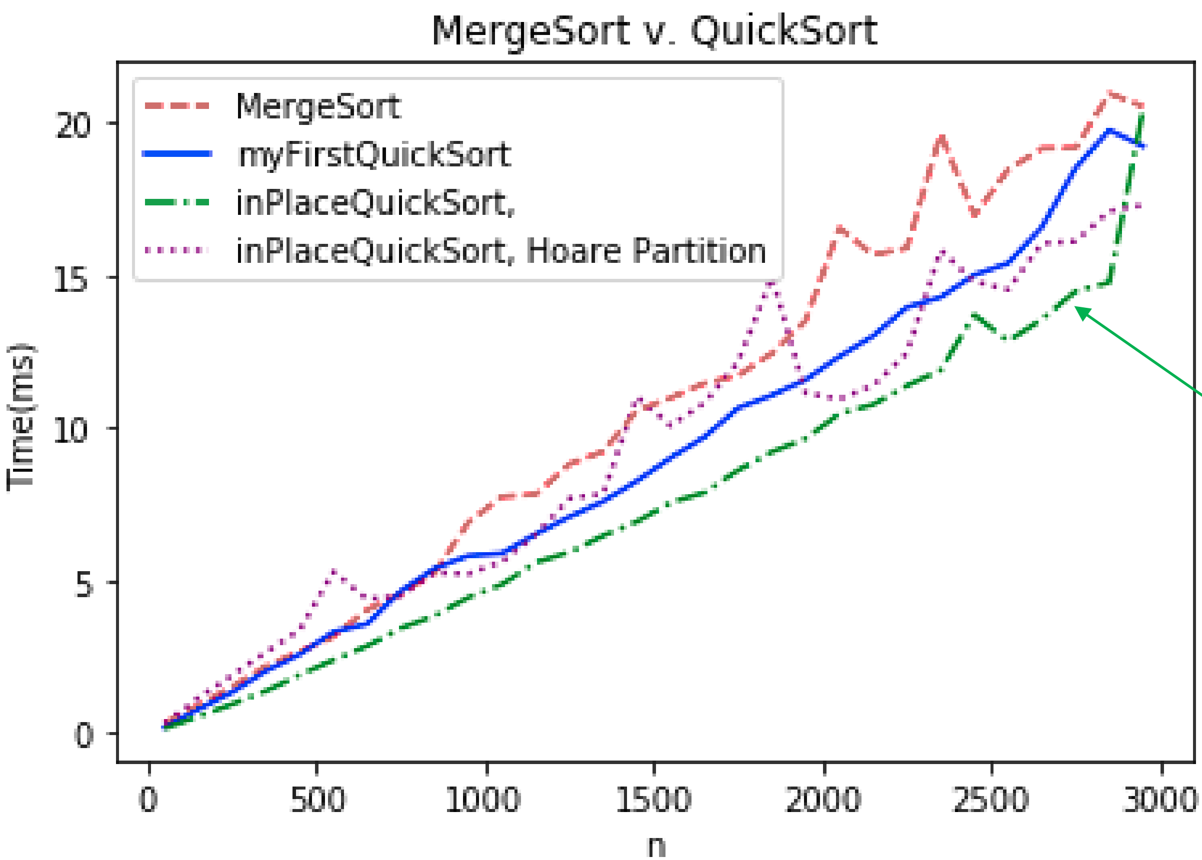
Krok dopředu.

Když vidí něco menšího než je pivot, **vyměníme** elementy před pruhy a inkrementujeme oba pruhy.

Opakujeme až do konce a umístíme pivot na správné místo.

QuickSort vs. chytřejší QuickSort vs. Mergesort?

- Všechny se zdají docela srovnatelné …



Funkce rozdělování pole na místě využívá méně místa a v této implementaci je také rychlejší.

QuickSort vs MergeSort

	QuickSort (náhodný pivot)	MergeSort (deterministický)
Doba běhu	• Nejhorší případ: $O(n^2)$ • Očekávaný: $O(n \log(n))$	Nejhorší případ: $O(n \log(n))$
Využití	• Java pro primitivní typy • C qsort • Unix • g++	• Java pro objekty • Perl
Na místě? (S $O(\log(n))$ paměti navíc)	Ano, snadno	Ne snadno* chceme-li zachovat stabilitu i běh. (Ale docela snadno, pokud můžete obětovat runtime).
Stabilní?	Ne	Ano
Jiný klad	Využití mezipaměti, pokud je implementován pro pole	Merge Sort jeopravdu efektivní, když je implementován na spojených seznamech

To jsme probrali

Spíše pro zajímavost

Co jsme probrali

- QuickSort (s náhodným pivotem) je randomizovaný řadící algoritmus.

- Jak měříme dobu běhu náhodného algoritmu?
  - Očekávaná doba běhu
  - Nejhorší doba běhu

Příště

- Můžeme řadit rychleji než $\Theta(n\log(n))$??