Přednáška 6

Dolní mez řazení u algoritmů využívajících porovnávání prvků

Řazení s časovou složitostí O(n)

Řazení

- Viděli jsem několik algoritmů s časovou složitostí O(n log(n)).
  - MERGESORT měl časovou složitost v nejhorším případě O(n log(n))
  - QUICKSORT měl očekávanou časovou složitost O(n log(n))

Můžeme ale řadit rychleji?

Záleží na tom, jak
to chceme udělat…

Popíšeme další možnosti

- Model řazení založený na porovnávání
  - Zahrnuje MergeSort, QuickSort, InsertionSort
  - Víme, že jakýkoliv algoritmus používající tento model musí pro řazení použít nejméně $\Omega(n \log(n))$ kroků.
  - Jaký je náš model počítání?
    - Vstup: pole
    - Výstup: seřazené pole (řazení v jednom poli – řazení na místě)
    - Používané (povolené) operace: porovnání prvků

- Jiný model (opustíme požadavek porovnání prvků a řazení na místě)
  - CountingSort a RadixSort
  - Doba běhu obou $O(n)$

Řazení založené na porovnání prvků

Algoritmy založené na porovnání

- Chceme seřadit pole položek.
- K hodnotám položek nemůžeme přistupovat přímo: můžeme porovnat pouze dvě položky a zjistit, která je větší nebo menší.

Algoritmy založené na porovnání

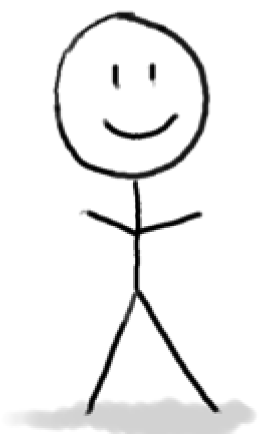
 je zkratka pro

„První věc ve vstupním seznamu
“

Chcete tyto položky seřadit.
Lze je nějak uspořádat (seřadit), ale nevíme, jak.

Je  větší než  ?

Ano



Algoritmus

Od algoritmu chceme, aby nám na výstupu dal seřazenou posloupnost všech prvků.

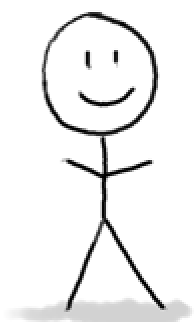
Existuje džin, který ví, jaké je správné pořadí.
Džin může odpovědět ANO / NE na otázky ve formě:
je [toto] větší než [tamto]?

Všechny řadící algoritmy, které jsme viděli, fungují takto.

Např., QuickSort:

Pivot!

7	6	3	5	1	4	2



Je 7 větší než 5 ? **ANO**

Je 6 větší než 5 ? **ANO**

Je 3 větší než 5 ? **NE**

5

atd.

Dolní mez $\Omega(n \log(n))$.

- Věta:
  - Libovolný deterministický řadící algoritmus založený na porovnávání prvků potřebuje $\Omega(n \log(n))$ kroků.
  - Očekávaný počet kroků libovolného pravděpodobnostního algoritmu založeného na porovnání prvků potřebuje $\Omega(n \log(n))$ kroků.
- Jak bychom to mohli dokázat?

Toto pokrývá všechny třídicí algoritmy, které dosud známe !!!

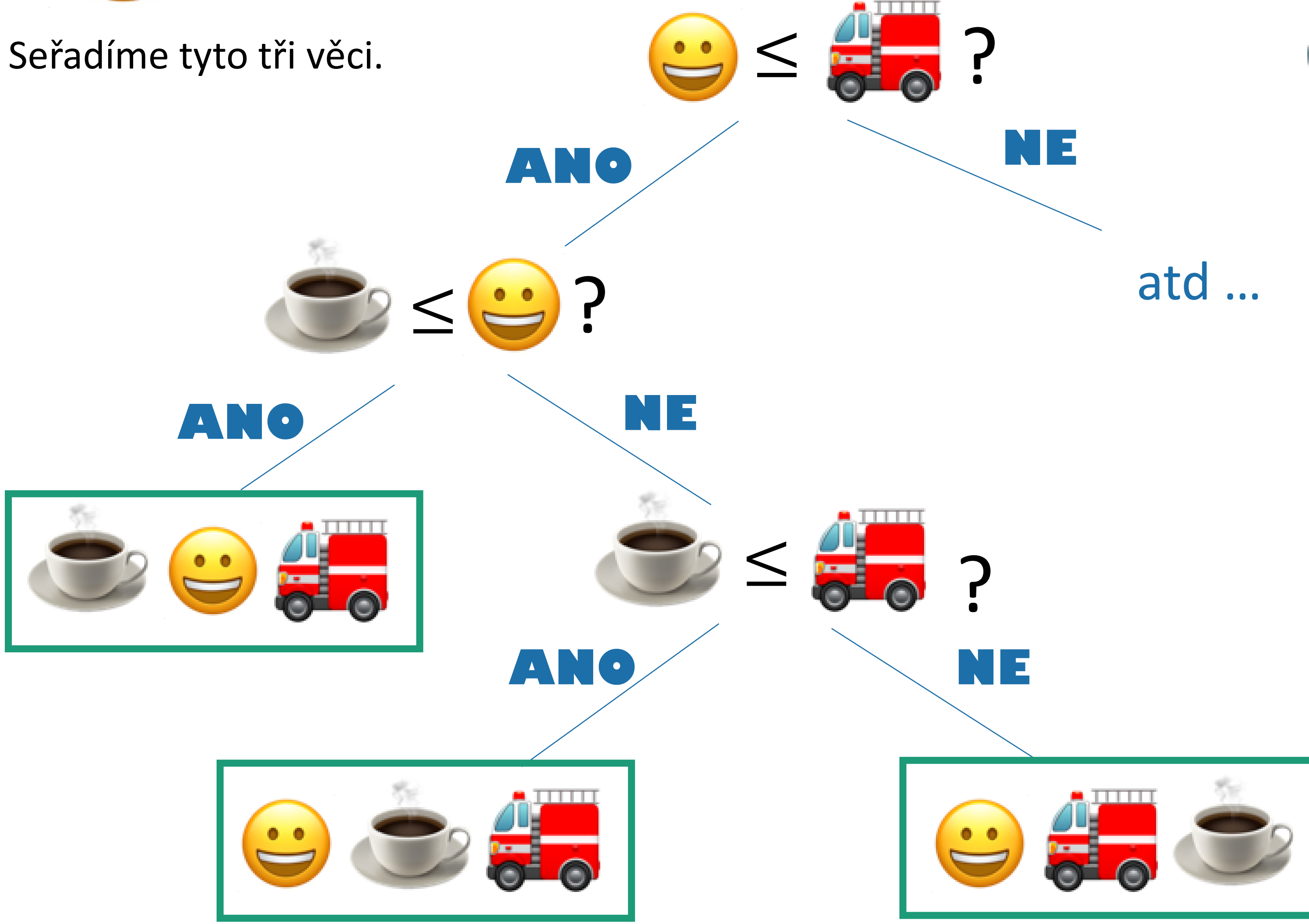
Možnost: Zvážit všechny algoritmy založené na porovnávání, jeden po druhém a analyzovat je.

Ale uděláme to jinak:

Místo toho si ukážeme, že všechny řadící algoritmy založené na porovnání vedou k rozhodovacímu stromu.
Poté budeme analyzovat rozhodovací stromy.

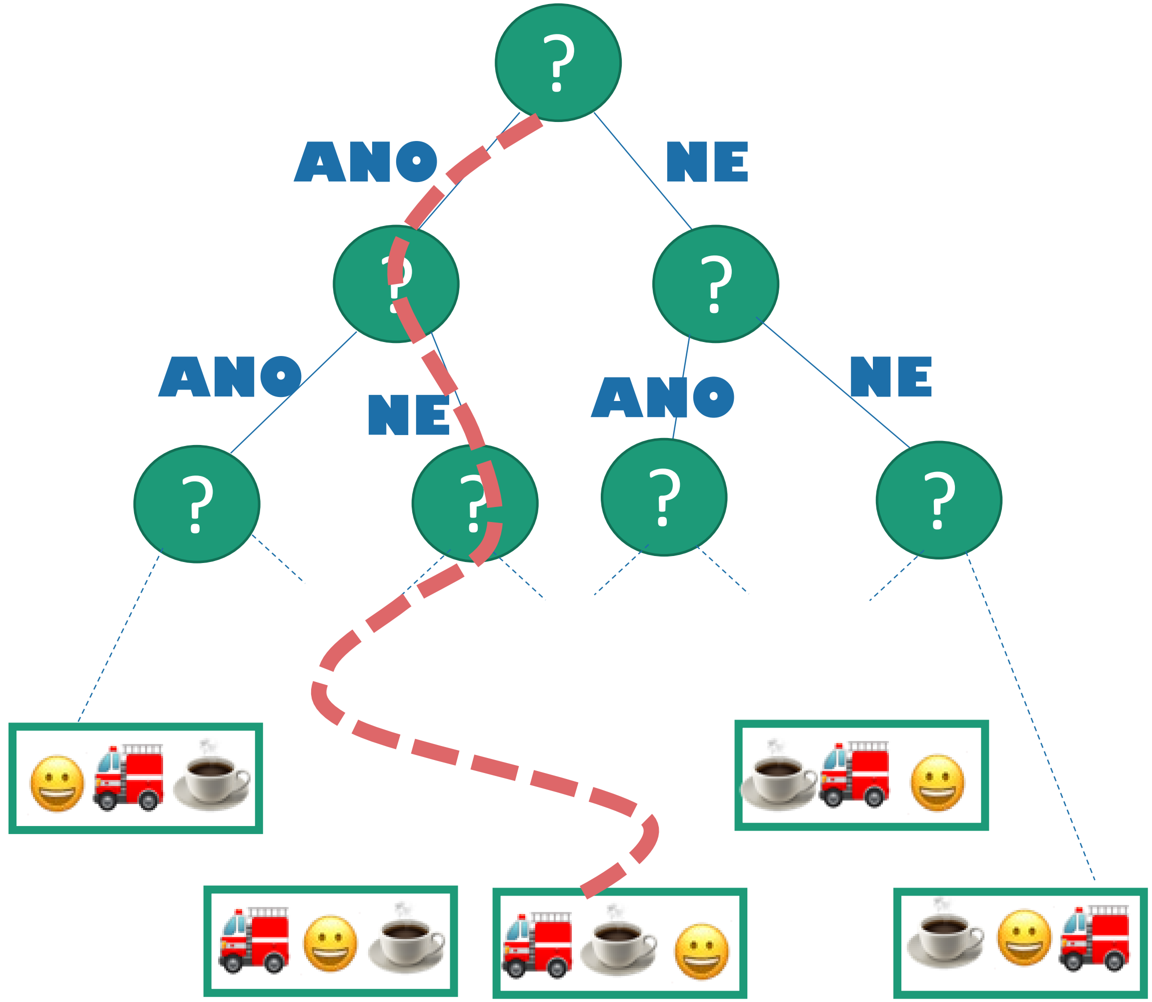
Rozhodovací stromy

Seřadíme tyto tři věci.

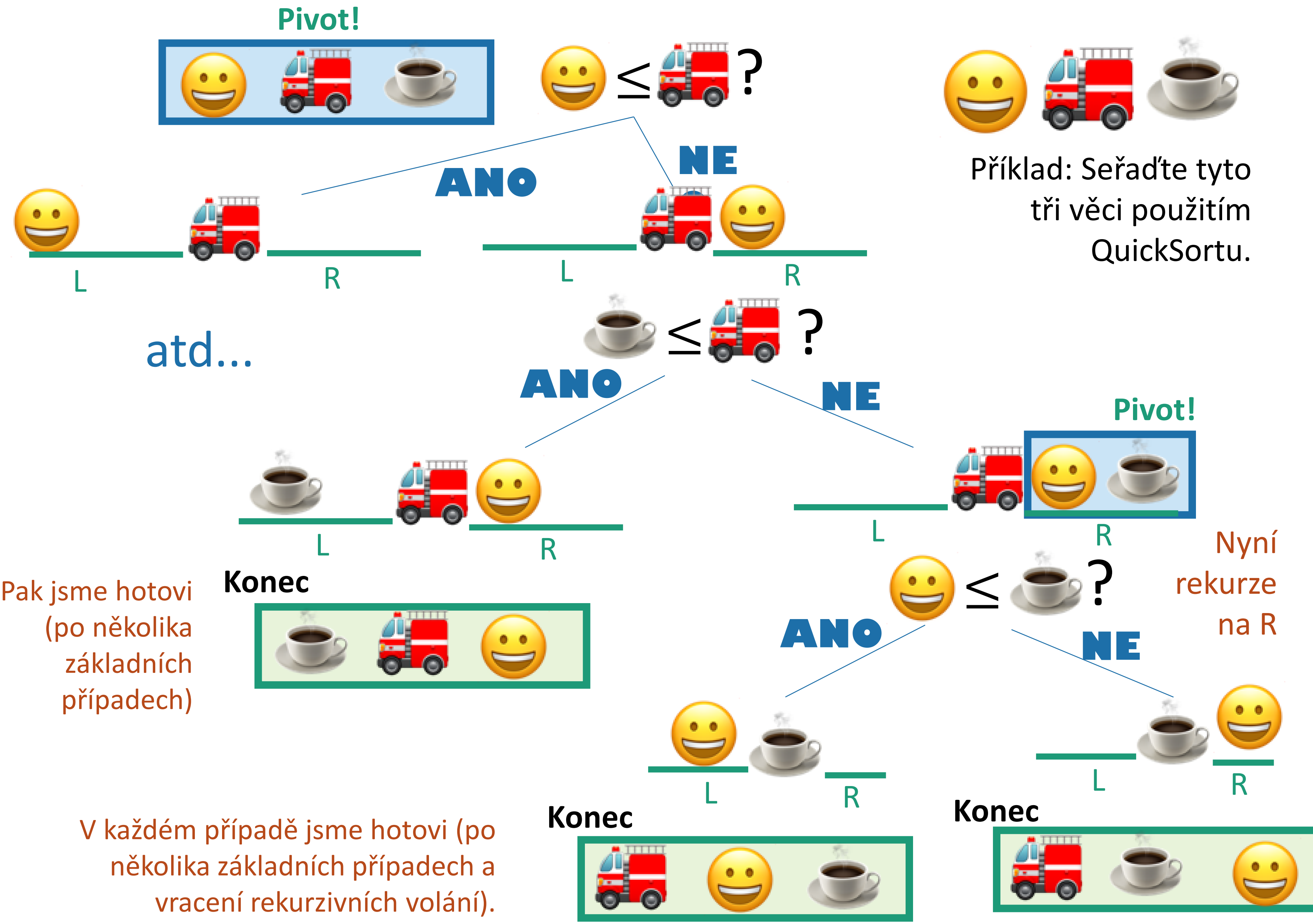


Rozhodovací stromy

- Vnitřní uzly odpovídají otázkám ano / ne.
- Každý vnitřní uzel má dvě děti, jedno pro „ano“ a jedno pro „ne“.
- Listy odpovídají výstupům..
  - V tomto případě všem možným řazení položek.
- Běh algoritmu s konkrétním vstupem odpovídá určité cestě stromem.



Algoritmy založené na porovnání vypadají jako rozhodovací stromy.



Příklad: Seřaďte tyto tři věci použitím QuickSortu.

Nyní rekurze na R

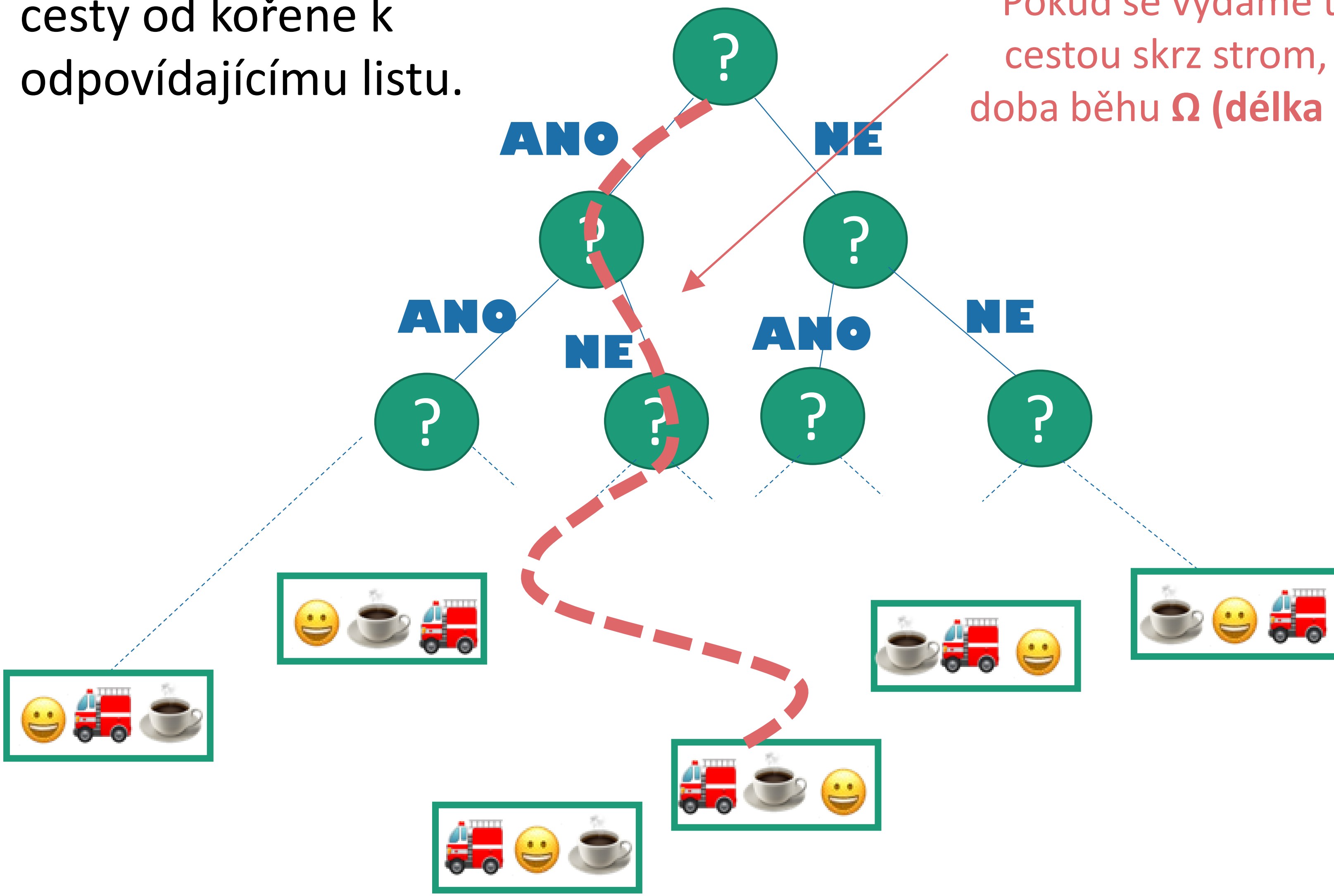
Pak jsme hotovi (po několika základních případech)

V každém případě jsme hotovi (po několika základních případech a vracení rekurzivních volání).

Otázka: Jaká je doba běhu konkrétního vstupu?

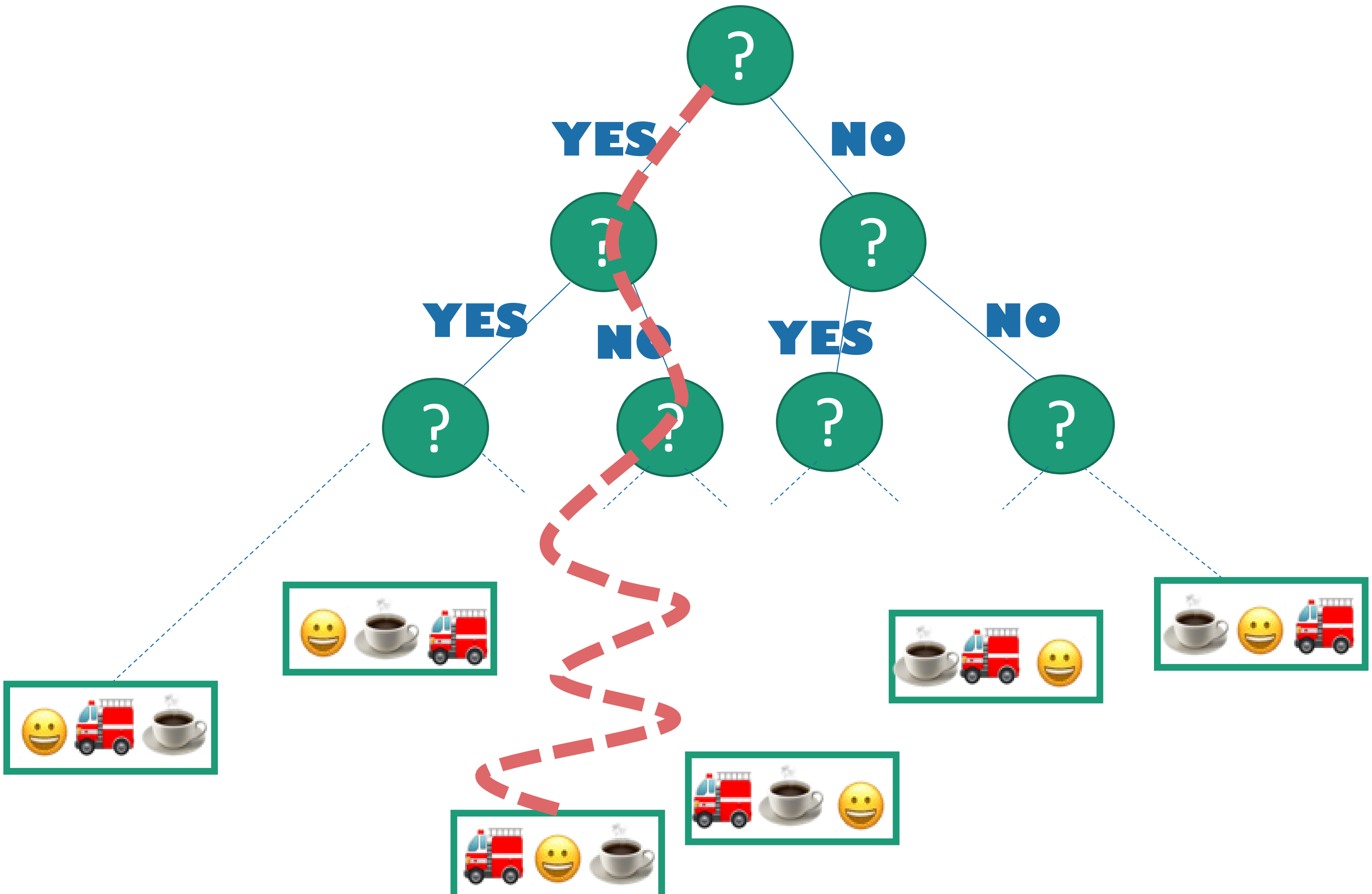
Odpověď: Alespoň délka cesty od kořene k odpovídajícímu listu.

Pokud se vydáme touto cestou skrz strom, je to doba běhu **Ω (délka cesty).**



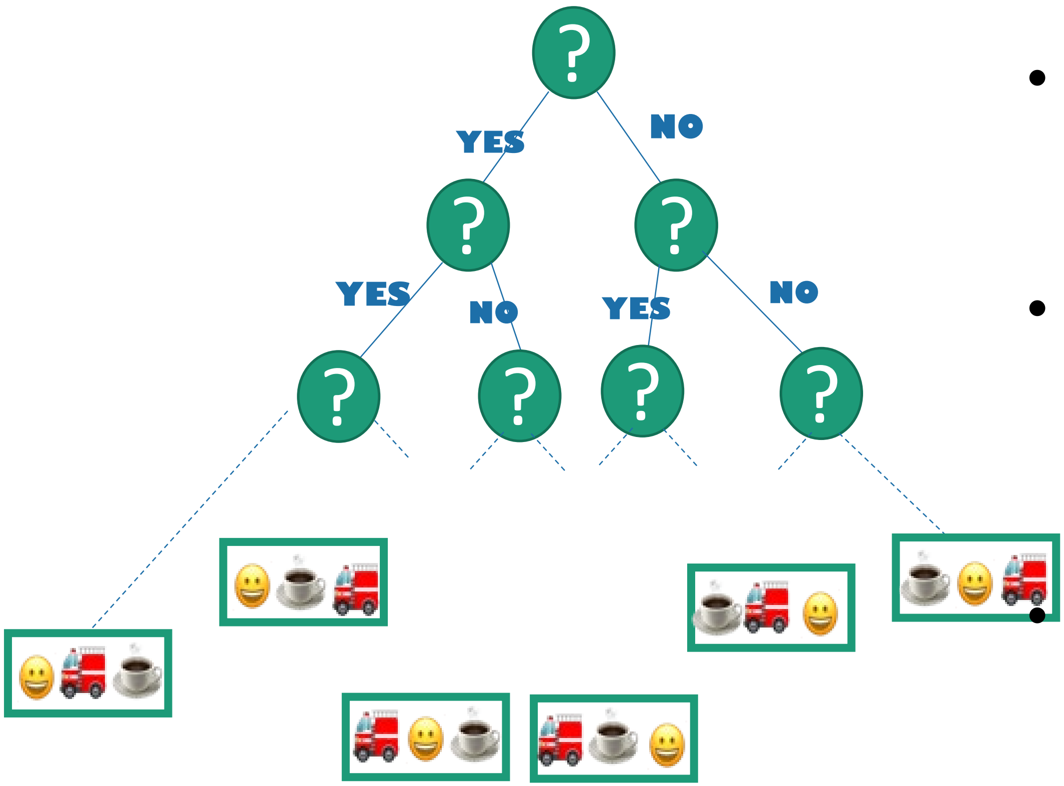
Otázka: Jaká je nejhorší doba běhu?

Odpověď: Alespoň Ω (délka nejdelší cestyy).



Jak dlouhá je nejdelší cesta?

Chceme dokázat: u všech takových stromů je nejdelší cesta přinejmenším ______



- Toto je binární strom s nejméně __n!__ listy.
- Strom s n! listy je nejmělčí a je kompletně vyvážený, takový strom má hloubku __log(n!)__.
- Takže u všech takových stromů je nejdelší cesta přinejmenším log(n!).

- n! je přibližně $(n/e)^n$ (Stirling. approx.*).
- log(n!) je tedy přibližně n log(n/e) = n log(n).

Závěr: nejdelší cesta má délku alespoň n log(n).

* Stirlingova aproximace je trochu komplikovanější, ale nám to stačí pro požadovaný asymptotický výsledek.

Dolní mez $\Omega(n \log(n))$.

- Věta:
  - Libovolný deterministický řadící algoritmus založený na porovnávání prvků musí mít $\Omega(n \log(n))$ kroků.

- Rekapitulace důkazu:
  - Libovolný deterministický algoritmus založený na porovnání lze reprezentovat jako rozhodovací strom s $n!$ listy.
  - Nejhorší dobou chodu je nejméně hloubka rozhodovacího stromu.
  - Všechny rozhodovací stromy s $n!$ listy mají hloubku $n\log(n)$.
  - Takže jakýkoli řadící algoritmus založený na porovnávání prvků musí mít nejhorší dobu běhu alespoň $\Omega(n\log(n))$.

Poznámka:
Co pravděpodobnostní algoritmy?

- Například, QuickSort?
- Věta:
  - Očekávaný počet kroků libovolného pravděpodobnostního algoritmu založeného na porovnání prvků musí být $\Omega(n \log(n))$ kroků.
- Důkaz:
  - (stejné myšlenky jako u deterministických algoritmů)

Zkuste sami!

To je tedy špatná zpráva

- Věta:
  - Libovolný deterministický řadící algoritmus založený na porovnávání prvků potřebuje $\Omega(n \log(n))$ kroků.

- Věta:
  - Očekávaný počet kroků libovolného pravděpodobnostního algoritmu založeného na porovnání prvků je $\Omega(n \log(n))$ kroků.

Ale také,
MergeSort je optimální!

- To je jedna z nejlepších věcí na dolních mezích, jako je tato: víme, kdy můžeme vyhlásit vítězství!

Můžeme seřadit prvky lépe?

- Existuje jiný model výpočtu, ve kterém můžeme řadit rychleji než nlog(n)?
- Kromě řadících algoritmů založených na porovnávání

Použijeme jiný model výpočtu

- Položky, které třídíme, mají spíše smysluplné hodnoty.

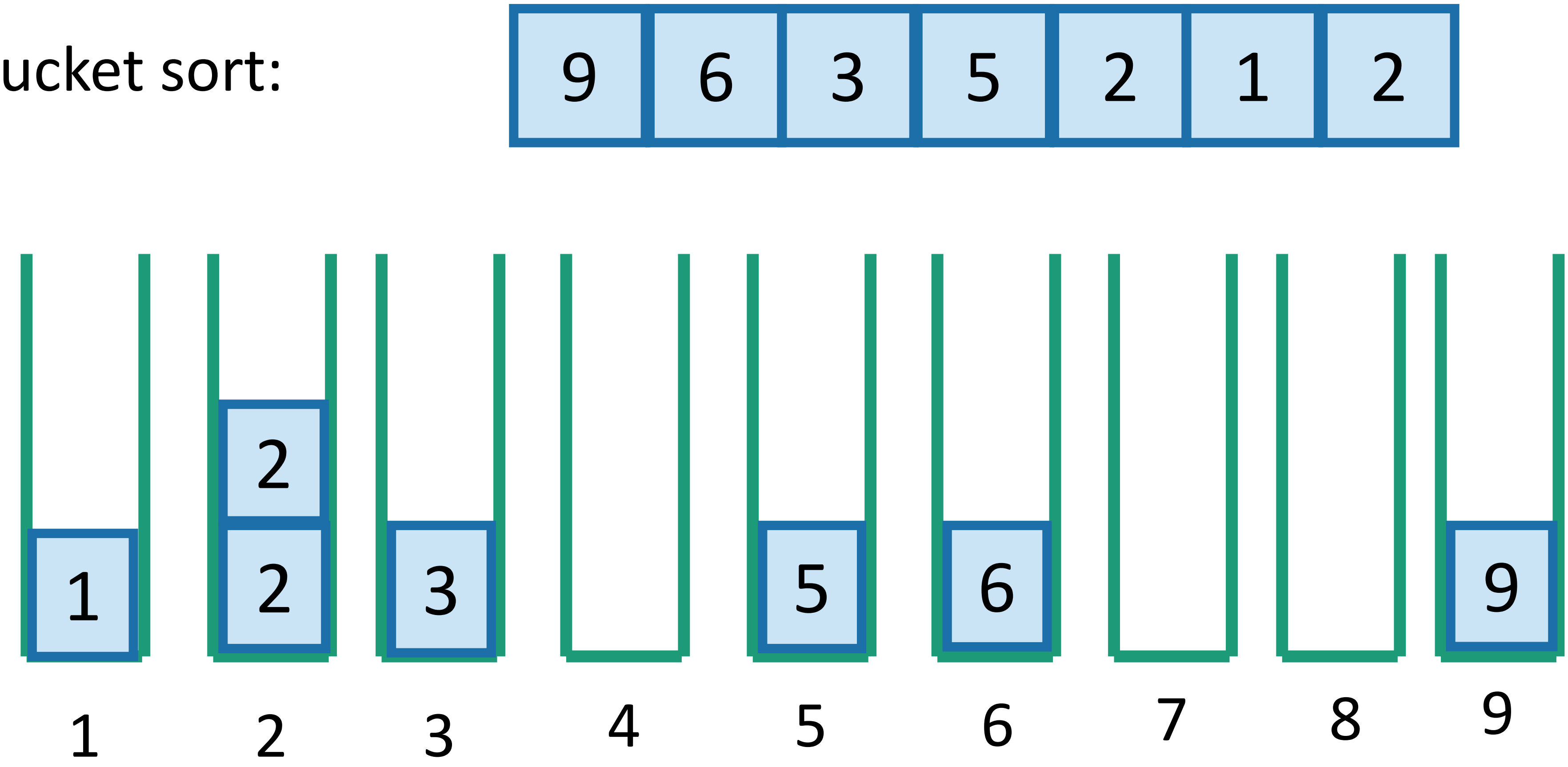
9	6	3	5	2	1	2

místo

Proč by to mohlo pomoci?

Implementujeme segmenty jako propojené seznamy. Uvnitř kbelíků bude fungovat fronta. Jako první dovnitř, první ven. To bude užitečné později.

Bucket sort:



Zřetězíme kbelíky!

SETŘÍDĚNO!

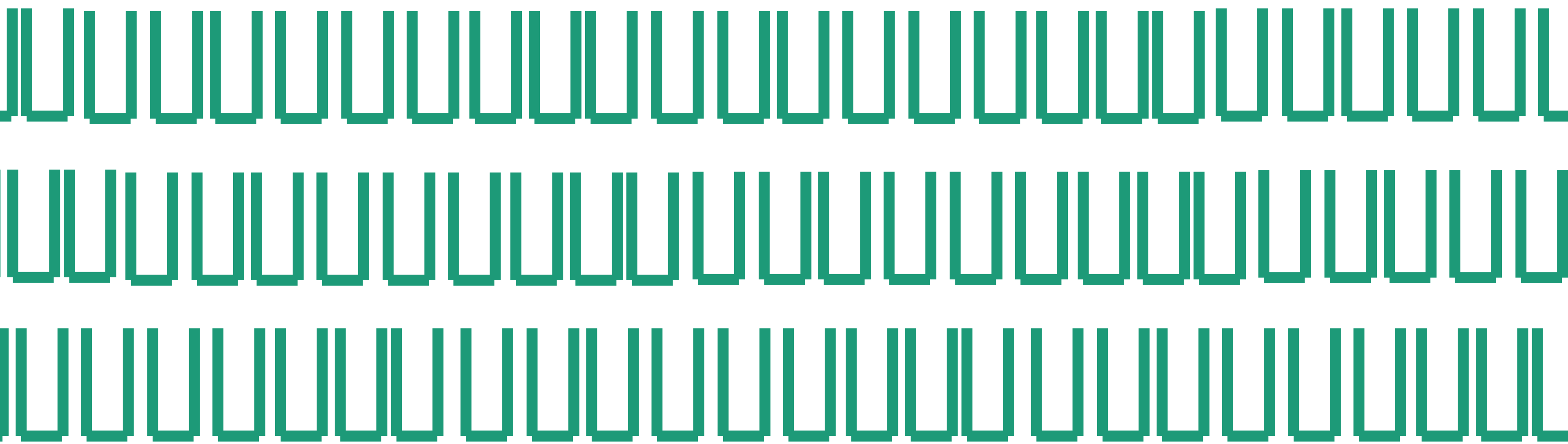
V čase O(n).

Předpoklady

- Potřebujeme vědět, do kterého kbelíku něco dát.
  - Předpokládáme, že můžeme položky hodnotit přímo, nejen porovnáním
- Potřebujete vědět, jaké hodnoty se mohou objevit předem.

2	12345	13	2^{1000}	50	100000000	1

- Je třeba předpokládat, že takových hodnot není příliš mnoho.

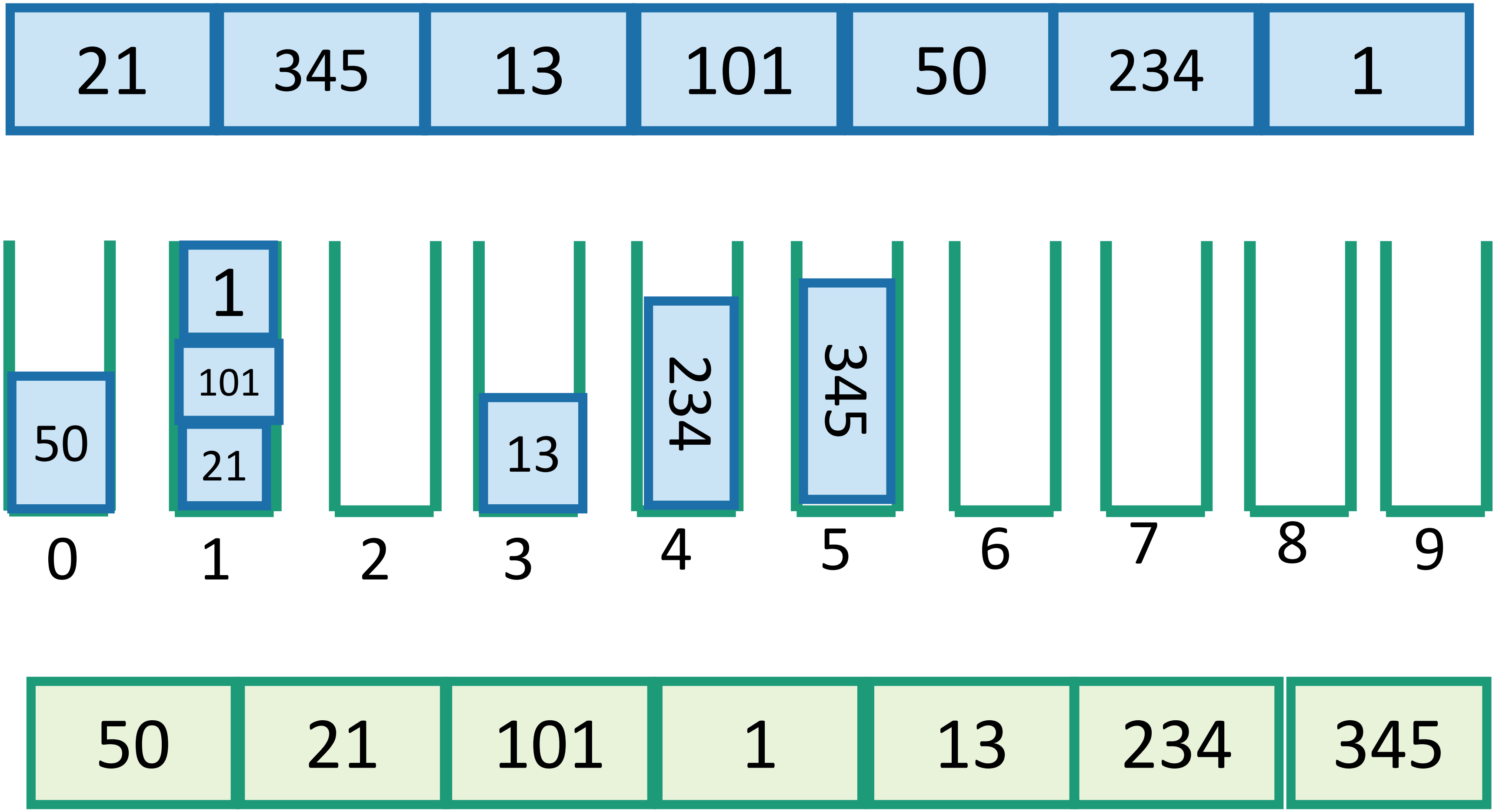


RadixSort

- Pro třídění celých čísel do velikosti M.
    - nebo obecněji pro lexikografické třídění řetězců
- Může využívat méně místa než Bucket sort

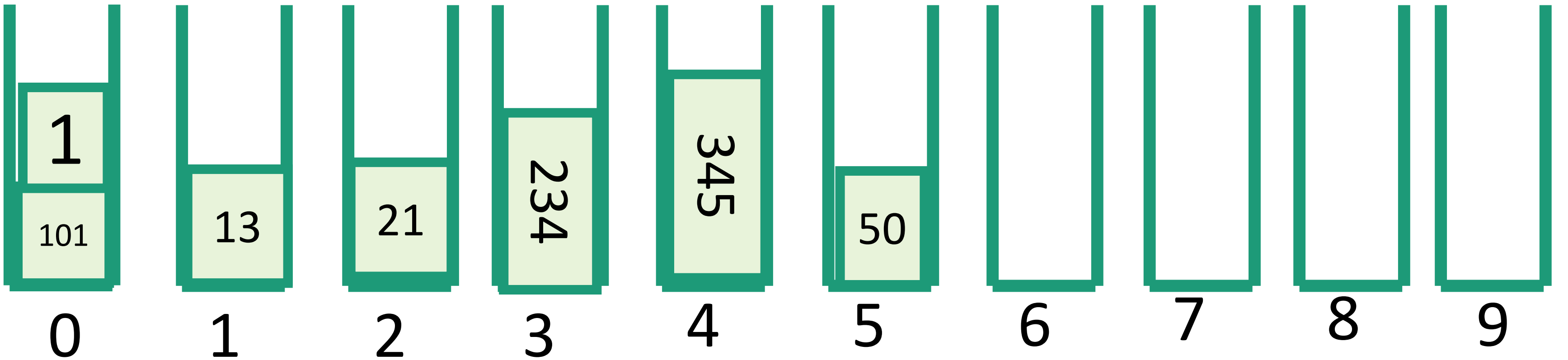
- Myšlenka: Začneme nejprve nejméně významnou číslicí, potom na další nejméně významnou číslici atd.

Krok 1: Radix sort na nejméně významné číslici



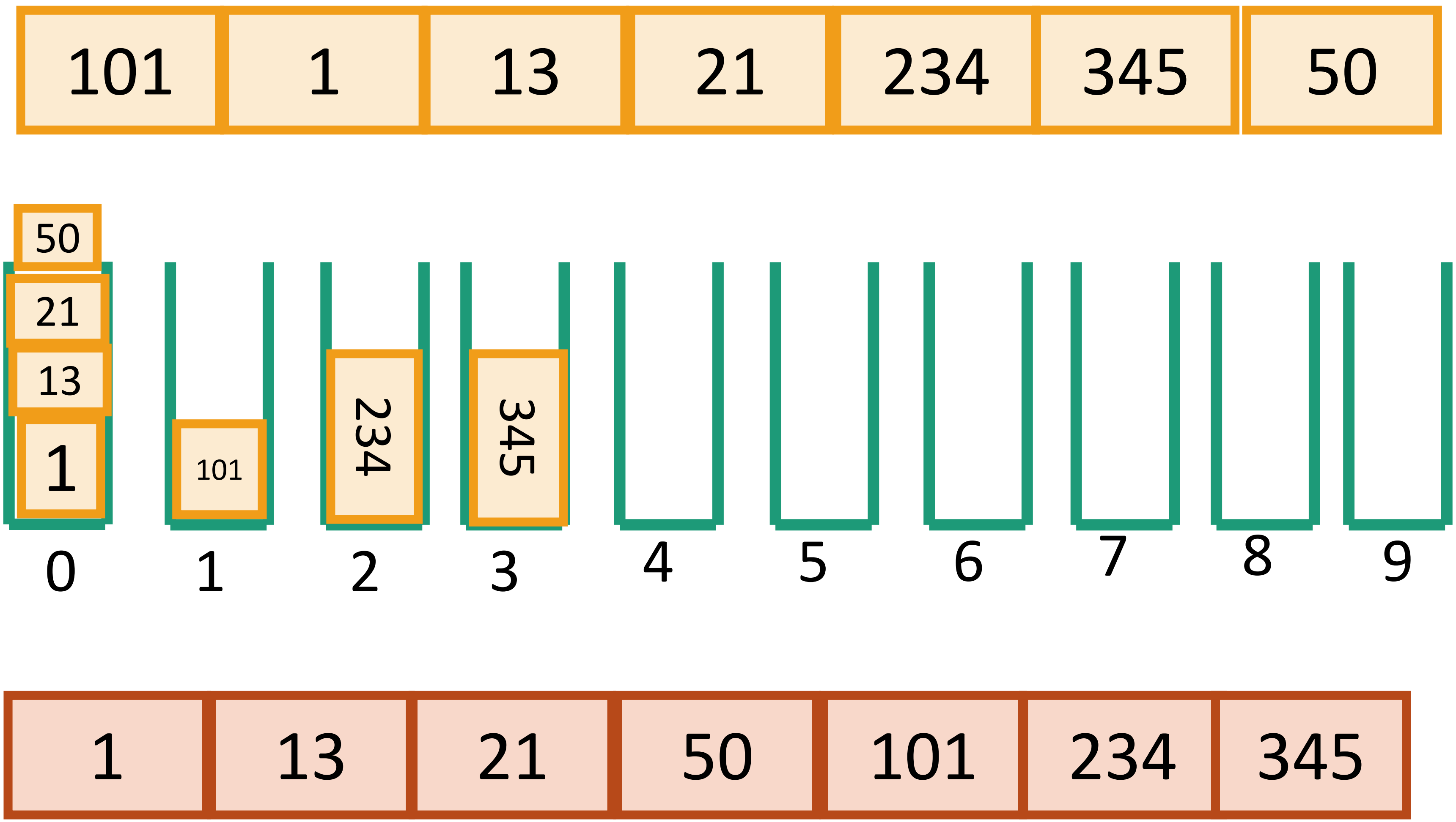
Step 2: Radix sort na 2. nejméně významné číslici

50	21	101	1	13	234	345



101	1	13	21	234	345	50

Krok 3: Radix sort na 3. nejméně významné číslici



Funguje to!!

Proč to funguje?

Původní pole:

21	345	13	101	50	234	1

Pole po seřazení podle 1. číslice

5**0**	2**1**	10**1**	**1**	1**3**	23**4**	34**5**

Pole po seřazení podle 2 číslic

1**01**	**01**	**13**	**21**	2**34**	3**45**	**50**

Pole po seřazení podle 3 číslic

001	**013**	**021**	**050**	**101**	**234**	**345**

Seřazené pole

Dokážeme, že je to správné …

- Jaká je indukční hypotéza?

Původní pole:

21	345	13	101	50	234	1

Pole po seřazení podle 1. číslice.

50	21	101	1	13	234	345

Pole po seřazení podle 2 číslic.

101	01	13	21	234	345	50

Pole po seřazení podle 3 číslic.

001	**013**	**021**	**050**	**101**	**234**	**345**

Seřazené pole

RadixSort je správný

- Induktivní hypotéza (IH):
  - Po k-té iteraci, pole je seřazeno podle prvních k nejméně významných číslic.
- Základní případ:
  - „Seřazeno podle 0 nejméně významných číslic“ znamená dosud neuspořádané, takže IH platí pro k = 0.
- Induktivní krok:
  - Uděláme následně
- Závěr:
  - Indukční hypotéza platí pro všechna k, takže po poslední iteraci je pole seřazeno podle všech číslic. Tedy, pole je seřazeno!

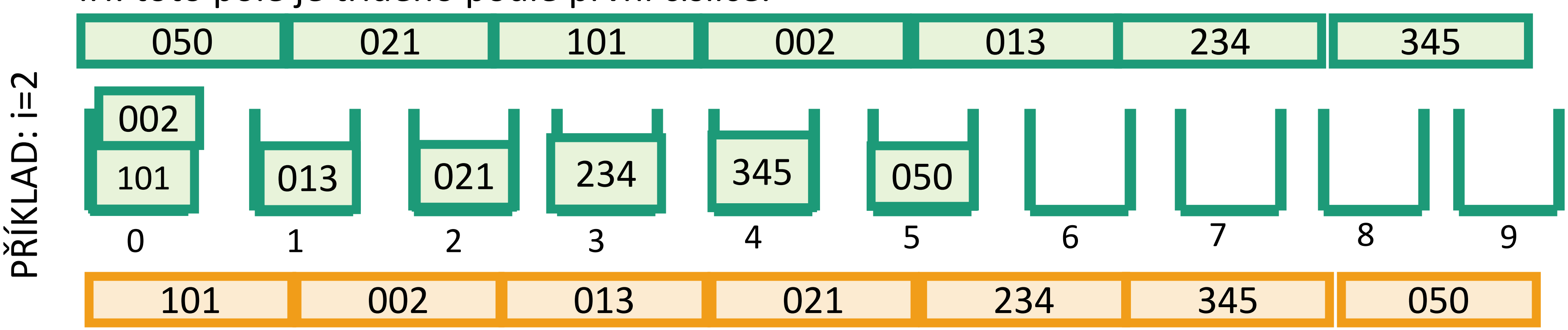
Induktivní hypotéza:

Po k-té iteraci je pole seřazeno podle prvních k nejméně významných číslic.

Induktivní krok

- Potřebujeme ukázat: pokud IH platí pro k=i-1, potom platí také pro k=i.
  - Předpokládejme, že po i-1 –té iteraci je pole seřazeno podle i-1 prvních nejméně významných číslic.
  - Potřebujeme ukázat, že po i-té iteraci je pole seřazeno podle prvních i nejméně významných číslic.

IH: toto pole je tříděno podle první číslice.



Chceme ukázat: toto pole je seřazeno podle 1. a 2 číslice (nejméně významné, tj. zleva.

Schéma důkazu...

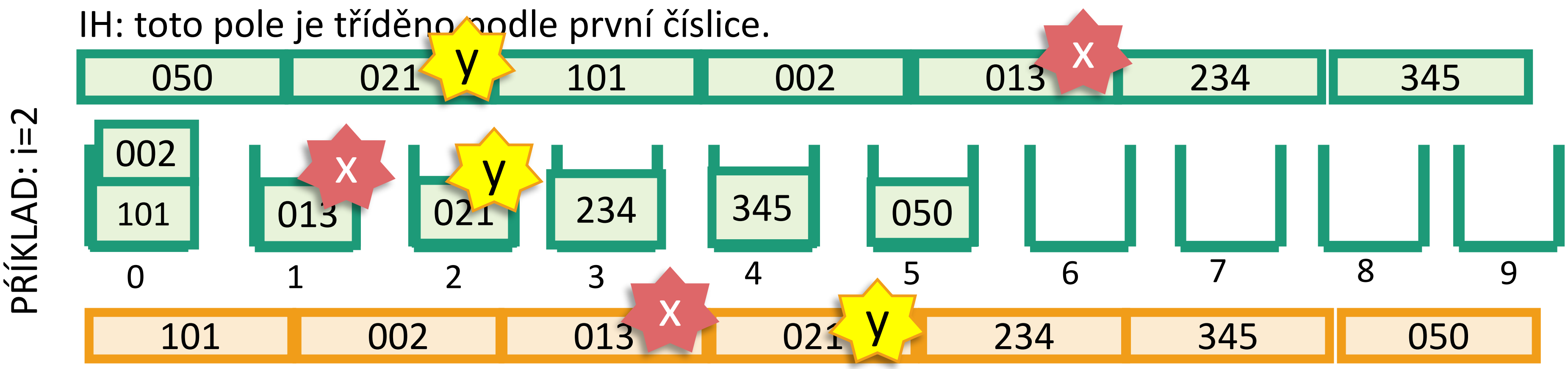
důkaz dále

Chceme ukázat: po *i*-té iteraci je pole seřazeno podle prvních *i* nejméně významných číslic.

- Nechť $x=[x_d x_{d-1} \ldots x_2 x_1]$ a $y=[y_d y_{d-1} \ldots y_2 y_1]$ jsou libovolné x,y.
- Předpokládejme, že $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Chceme ukázat, že x se objeví před y na konci i-té iterace.
- PŘÍPAD 1: $x_i<y_i$
  - x je v dřívější přihrádcenež y.

Tedy, chceme ukázat, že pro libovolná x a y, pokud x patří před y, pak také dáme x před y.

IH: toto pole je tříděno podle první číslice.



Chceme ukázat: toto pole je tříděno podle 1. a 2. číslice.

Schéma důkazu…

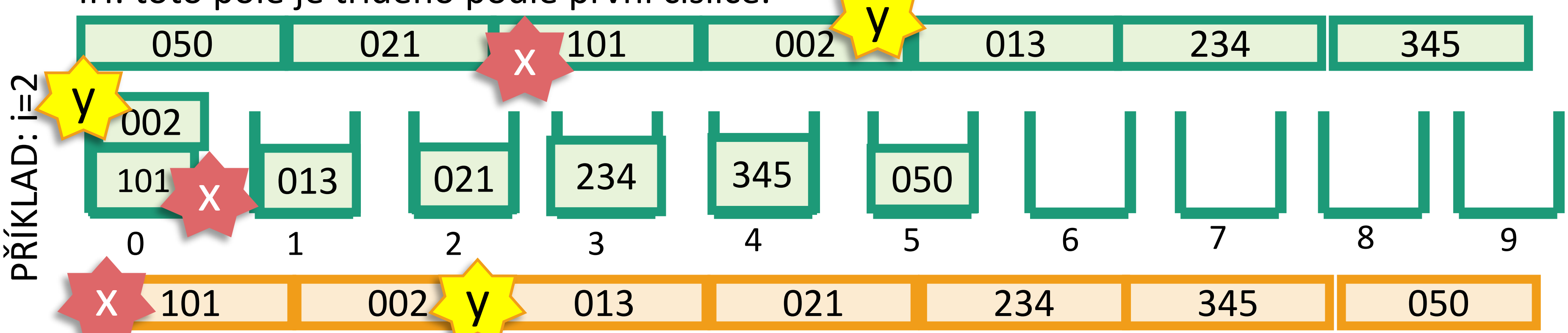
důkaz dále

Chceme ukázat: po *i*-té iteraci je pole seřazeno podle prvních *i* nejméně významných číslic

- Nechť $x=[x_d x_{d-1} \ldots x_2 x_1]$ a $y=[y_d y_{d-1} \ldots y_2 y_1]$ jsou libovolné x,y.
- Předpokládejme, že $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Chceme ukázat, že x se objeví před y na konci i-té iterace.
- PŘÍPAD 1: $x_i < y_i$
  - x je v dřívějším kbelíku než y.
- PŘÍPAD 2: $x_i = y_i$
  - $[x_{i-1} \ldots x_2 x_1] < [y_{i-1} \ldots y_2 y_1]$,
  - x a y ve stejné přihrádce, ale x bylo vloženo do přihrádky jako první.

Tedy, chceme ukázat, že pro libovolná x a y, pokud x patří před y, pak také dáme x před y.

IH: toto pole je tříděno podle první číslice.



Chcete ukázat: toto pole je tříděno podle 1. a 2. číslice.

Chci ukázat: po *i*-té iteraci je pole seřazeno podle prvních *i* nejméně významných číslic.

- Nechť $x=[x_d x_{d-1} \ldots x_2 x_1]$ and $y=[y_d y_{d-1} \ldots y_2 y_1]$ jsou libovolná x,y.
- Předpokládejme, že $[x_i x_{i-1} \ldots x_2 x_1] < [y_i y_{i-1} \ldots y_2 y_1]$.
- Chceme ukázat, že x se objeví před y na konci i-té iterace.
- PŘÍPAD 1: $x_i < y_i$.
  - x se objeví v dřívější přihrádce než y, takže x se po i-té iteraci objeví před y.
- PŘÍPAD 2: $x_i = y_i$.
  - x a y končí ve stejné přihrádce.
  - V tomto případě, $[x_{i-1} \ldots x_2 x_1] < [y_{i-1} \ldots y_2 y_1]$, tedy, podle indukční hypotézy se x objeví před y po i-1 -té iteraci.
  - Poté bylo x vloženo do přihrádky dříve než y, takže také x bude vyjmuto z přihrádky dříve než y.
    - Připomeňme, že kbelíky jsou fronty FIFO.
  - Takže x se objeví před y v i-té iteraci.

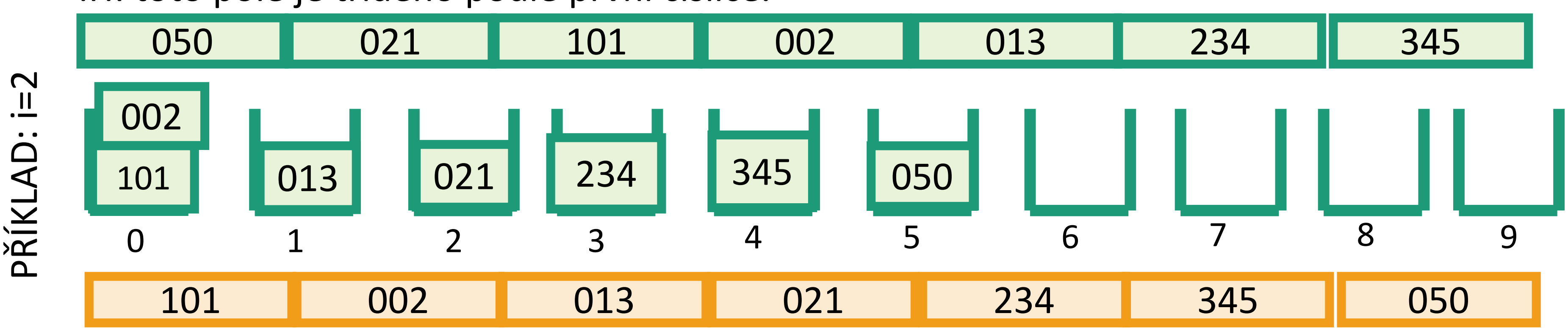
Induktivní krok

Induktivní hypotéza :

Po k-té iteraci je pole seřazeno podle prvních k nejméně významných číslic.

- Potřebujeme ukázat: pokud IH platí pro k=i-1, potom platí také pro k=i.
  - Předpokládejme, že po i-1 –té iteraci je pole seřazeno podle prvních i-1 nejméně významných číslic.
  - Je třeba ukázat, že po i-té iteraci je pole seřazeno podle prvních i nejméně významných číslic.

IH: toto pole je tříděno podle první číslice.



Chceme ukázat: toto pole je tříděno podle 1. a 2. číslice.

RadixSort je korektní

- Induktivní hypotéza:
  - Po k-té iteraci je pole seřazeno podle prvních k nejméně významných číslic.
- Základní případ:
  - “Seřazeno podle 0 nejméně významných číslic “znamená nezařazeno, takže IH platí pro k = 0.
- Induktivní krok:
  - UDĚLÁNO
- Závět:
  - Indukční hypotéza platí pro všechna k, takže po poslední iteraci je pole seřazeno podle všech číslic. Proto je pole seřazeno!

Jaký je čas běhu?

Pro Radix Sort čísla základ -10.

- Předpokládejme, že třídíme n d-ciferných čísel.

Např. n=7, d=3:

021	345	013	101	050	234	001

1. Kolik iterací tam je?

2. Jak dlouho trvá každá iterace?

3. Jaká je celková doba běhu?

Jaký je čas běhu?

Pro Radix Sort čísla základ -10.

- Předpokládejme, že třídíme n d-ciferných čísel.

Např. n=7, d=3:

021	345	013	101	050	234	001

1. Kolik iterací tam je?
   - d iterací
2. Jak dlouho trvá každá iterace?
   - Čas na inicializaci 10 přihrádek plus čas na vložení n čísel do 10 přihrádek. O(n).
3. Jaká je celková doba běhu?
   - O(n.d)

To nevypadá příliš impozantně

- Abychom seřadili n celých čísel, každé z nich je v {1,2,…,n}…
- d = $\lfloor \log_{10}(n) \rfloor + 1$
  - Například:
    - n = 1234
    - $\lfloor \log_{10}(1234) \rfloor + 1 = 4$
  - Vysvětlení dále.
- Doba = $O(nd) = O(n \log(n))$.
  - ? viz - MergeSort!

Můžeme řadit rychleji?

- RadixSort se základem 10 (počet přihrádek) se nejeví jako dobrý nápad …
- Ale co když změníme základ? (Řekněme základ r)
- Uvidíme, že nastane kompromis :
  - Větší r znamená více přihrádek
  - Větší r znamená méně číslic

Příklad: základ 100

Původní pole:

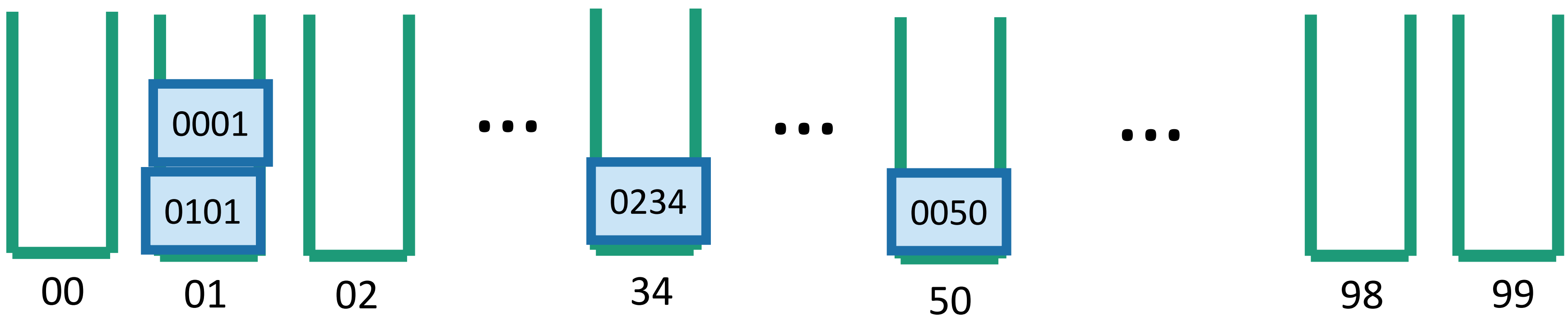
21	345	13	101	50	234	1

Příklad: základ 100

Původní pole:

0021	0345	0013	0101	0050	0234	0001

100 přihrádek:

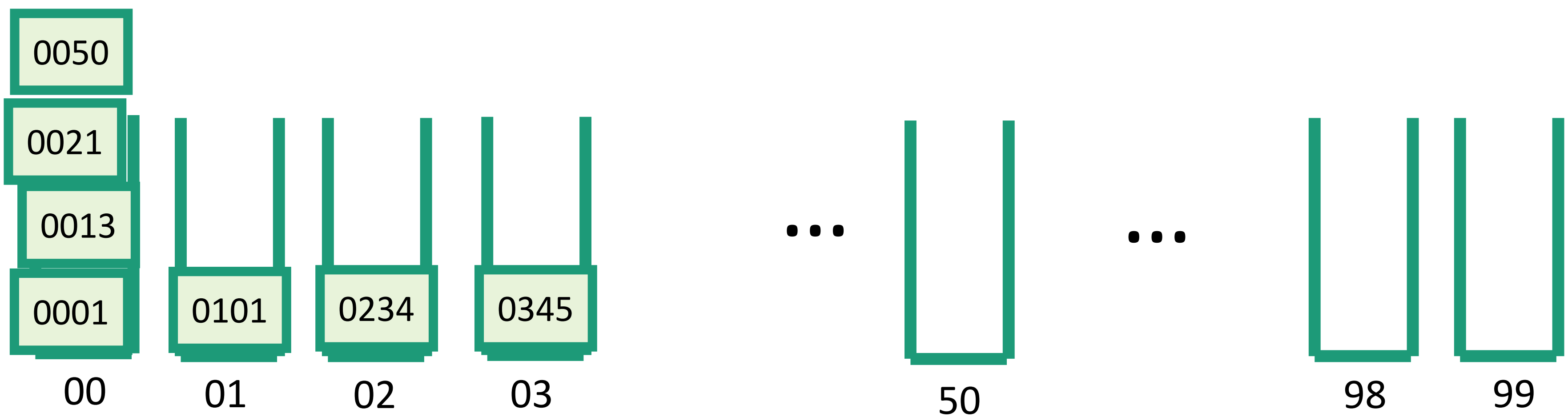


0101	0001	0013	0021	0234	0345	0050

Příklad: základ 100

0101	0001	0013	0021	0234	0345	0050

100 přihrádek:



0001	0013	0021	0050	0101	0234	0345

Seřazeno!

Příklad: základ 100

Původní pole

0021	0345	0013	0101	0050	0234	0001

0101	0001	0013	0021	0234	0345	0050

0001	0013	0021	0050	0101	0234	0345

Seřazené pole

Základ 100:

- d=2, tak máme jen 2 iterace. vs.
- 100 přihrádek

Základ 10:

- d=3, takže 3 iterace.
- 10 přihrádek

Větší základ znamená více přihrádek, ale méně iterací.

Obecná doba běhu RadixSort

- Řekněme, že chceme třídit :
  - n celých čísel,
  - Maximální hodnota M,
  - základ r.
- Počet iterací RadixSort:
  - Stejné jako počet číslic, základ r, od celého čísla x maximální velikosti M..
  - To je $\mathrm{d} = \lfloor \log_r(M) \rfloor + 1$

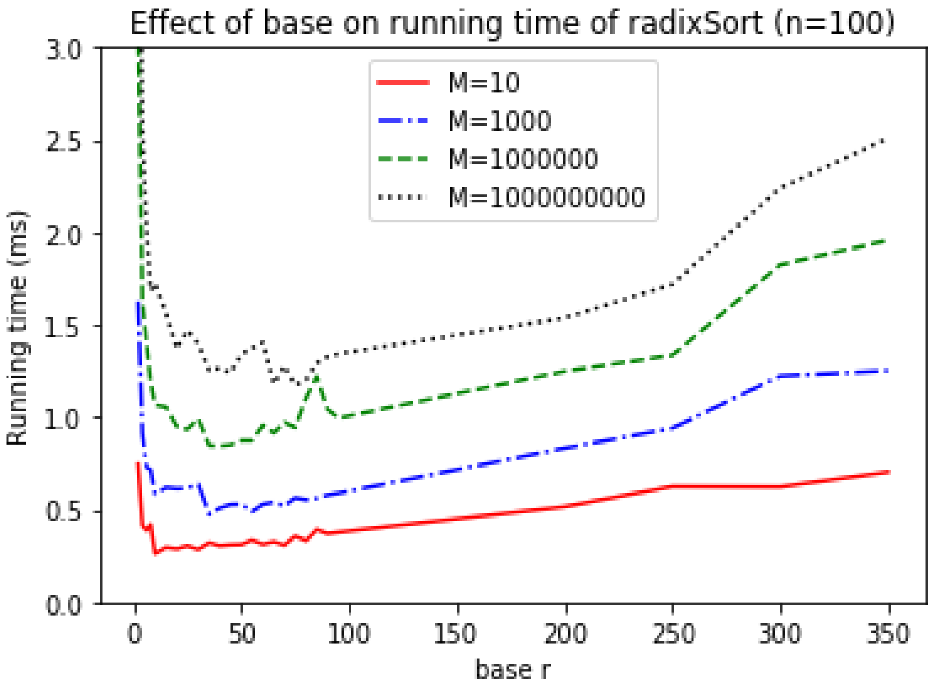
Přesvědčte se, že toto je ten správný vzorec pro d.

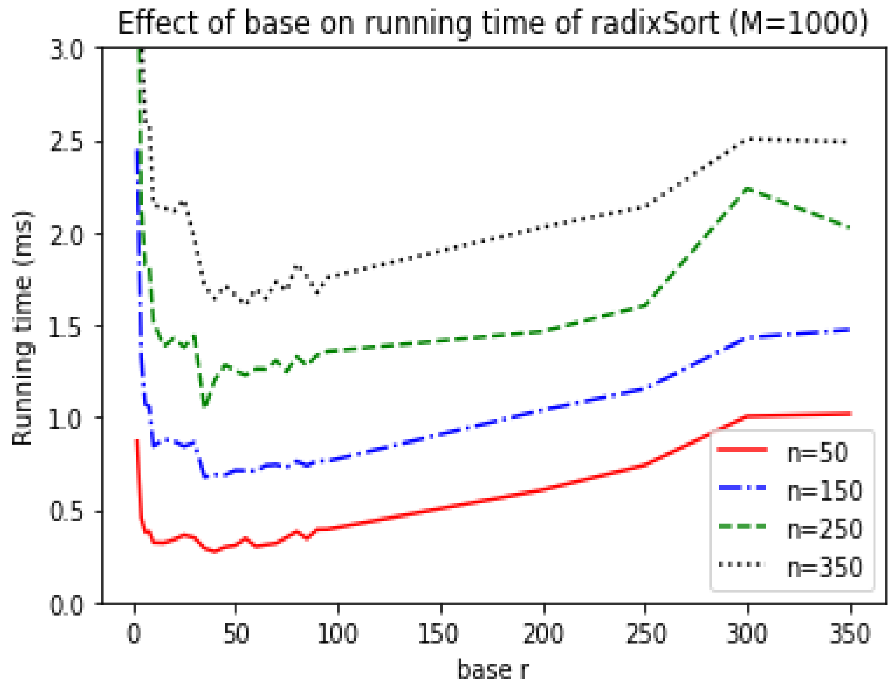
- Čas na iteraci:
  - Inicializujeme kbelíky r, vložíme do nich n položek
  - $O(n+r)$ celková doba.
- Celková doba:
  - $O\big(d \cdot (n+r)\big) = O\big((\lfloor \log_r(M) \rfloor + 1) \cdot (n+r)\big)$

Doba běhu: $O\left(\left(\lfloor \log_r(M) \rfloor + 1\right) \cdot (n+r)\right)$

Kompromisy

- Máme dáno n, M, jak máme volit r?
- Z experimentů to vypadá, že můžeme volit nějakou vhodnou hodnotu:





Rozumná volba : r=n

- Doba běhu:

$$O\left(\left(\lfloor \log_r(M) \rfloor + 1\right) \cdot (n + r)\right)$$

Intuitivně: zhruba rovnost n a r zde.

- Zvolíme n=r:

$$O\left(n \cdot \left(\lfloor \log_n(M) \rfloor + 1\right)\right)$$

Volba r = n je docela dobrá. Jaká volba r optimalizuje asymptotickou dobu běhu? Co když mi také záleží na prostoru? (paměti?)

Doba běhu RadixSort s r = n

- Chcete-li seřadit n celých čísel o velikosti maximálně M, je čas
  - $O\left(n \cdot \left(\lfloor \log_n(M) \rfloor + 1 \right)\right)$
- Takže doba běhu (ve smyslu n) tedy závisí na tom, jak velké M je ve smyslu n :
  - Pokud $M \leq n^c$ pro nějakou konstantu c, potom je to O(n).
  - Pokud $M = 2^n$, potom je to $O\left(\frac{n^2}{\log(n)}\right)$
- Počet potřebných přihrádek je r=n.

Co jsme se dozvěděli?

Sem můžeme dát jakoukoli konstantu místo 100.

- RadixSort umí řadit n celých čísel o velikosti maximálně n^{100} v čase O(n) a potřebuje dostatek místa pro uložení O(n) celých čísel.
- Pokud mají naše celá čísla mnohem větší velikost než n (například 2^n), možná bychom RadixSort neměli používat.
- Záleží na tom, jak si vybereme základ.

Rekapitulace

- Závisí na modelu výpočtu, jak bude obtížné řazení.
- Jak rozumný je model výpočtu, je na diskusi.

- Model řazení založený na porovnávání prvků
  - To zahrnuje MergeSort, QuickSort, InsertionSort
  - Jakýkoliv algoritmus založený na tomto modelu musí použít nejméně $\Omega(n \log(n))$ operací. ☹
  - Dokáže však zpracovat libovolné srovnatelné objekty. ☺
- Pokud třídíme malá celá čísla (nebo jiná rozumná data):
  - BucketSort a RadixSort
  - Oba pracují v čase $O(n)$ ☺
  - Pokud budou celá čísla příliš velká, může zabrat více prostoru a/nebo být pomalejší ☹

Příště

- Binární vyhledávací stromy!
- Vyvážené binární vyhledávací stromy!