

# KMI/ALGO3 — Algoritmy 3

## 4. Rozděl a panuj

Jan Konečný

Katedra informatiky, PřF UP Olomouc

# Kde jsme: sylabus KMI/ALGO3

- Co znamená navrhnout algoritmus — specifikace, invariant, složitost
- Hladové algoritmy — kdy je lokální volba bezpečná
- Hladové algoritmy na grafech — kostry, nejkratší cesty
- **Rozděl a panuj — rekurence, Master theorem**
- Rozděl a panuj II — chytré rozklady
- Dynamické programování I — hledání stavů
- Dynamické programování II — vícerozměrné stavy
- Toky v sítích — max-flow/min-cut jako návrhový nástroj
- Systematické prohledávání — hrubá síla a backtracking
- Branch-and-bound — meze přesného řešení, P a NP
- Adversariální návrh — minimax, alfa-beta
- Když přesné optimum není realistické — aproximace, heuristiky
- Randomizace a syntéza — jak si vybrat metodu

## Jiná strategie než hladovost

- Hladové algoritmy budovaly řešení po *jednom rozhodnutí*.
- Teď velký problém rozložíme na menší problémy *stejného typu*, vyřešíme je rekurzivně a výsledky spojíme.

Otázka zní, jak problém rozdělit, aby byly podproblémy jednodušší a jejich výsledky šly levně složit.

Samotné rozdělení totiž nestačí: kdybychom po rekurzivním volání museli znovu řešit skoro celý původní problém, nic nezískáme.

Úspěšný návrh vyvažuje **tři ceny**:

počet podproblémů   ·   jejich velikost   ·   práce při skládání

## Tři otázky

1. Proč je rekurzivní algoritmus **korektní**?
2. Jak z jeho struktury sestavit **rekurenci** pro dobu běhu?
3. Jak rekurenci **vyřešit** bez hádání podle několika prvních hodnot?

### Struktura metody

1. **Rozděl:** vstup velikosti  $n$  převedeme na menší podproblémy.
2. **Panuj:** podproblémy vyřešíme rekurzivně; malé vstupy řešíme přímo (*základní případy*).
3. **Slož:** z výsledků podproblémů vytvoříme výsledek původní úlohy.

Někdy je skládání téměř zdarma (binární vyhledávání), jindy je hlavním zdrojem složitosti (mergesort), občas právě při něm získáme informaci, která neleží v žádném podproblému (počítání inverzí).

# Rekurence pro dobu běhu

## Schéma rozděl a panuj

Vytváří-li algoritmus  $a$  podproblémů o velikosti přibližně  $n/b$  a mimo rekurzivní volání vykoná práci  $f(n)$ , popisuje jeho dobu běhu

$$T(n) = a T(n/b) + f(n)$$

doplněná konstantními hodnotami pro malá  $n$ .

Tato rovnice popisuje *pouze cenu* výpočtu, o správnosti algoritmu nic neříká:

- $a T(n/b)$  odpovídá rekurzivním voláním,
- $f(n)$  zahrnuje rozdělení, skládání i ostatní práci aktuálního volání.

# Důkaz správnosti sleduje strukturu programu

## Důkazní šablona

**Základ:** algoritmus správně řeší nejmenší přípustné vstupy.

**Indukční předpoklad:** správnost pro všechny menší vstupy, které může algoritmus vytvořit.

**Indukční krok:** rozdělení vytváří platné menší instance a fáze skládání převádí jejich správné výsledky na správný výsledek celé instance.

Zastavení:

Každý podproblém musí být skutečně menší a po konečném počtu dělení dospět k základnímu případu.

Rekurzivní volání se stejnou velikostí vstupu by vedlo k nekonečnému výpočtu.

# Binární vyhledávání

Neklesající pole  $A[1] \leq \dots \leq A[n]$ , hledaná hodnota  $x$ .

Porovnáme  $x$  s prostředním prvkem  $A[m]$ :

- $x = A[m]$  — končíme;
- $x < A[m]$  —  $x$  může být jen v levé části;
- $x > A[m]$  —  $x$  může být jen v pravé části.

## Binární vyhledávání

Udržíme interval  $[\ell, r]$ , na začátku  $[1, n]$ . Je-li  $\ell > r$ , hodnota v poli není. Jinak  $m = \lfloor (\ell + r)/2 \rfloor$  a podle porovnání pokračujeme v  $[\ell, m - 1]$ , resp.  $[m + 1, r]$ .

Vždy platí  $\ell \leq m \leq r$ , takže nový interval ztrácí alespoň prvek  $A[m]$  — je *ostře* kratší. Po konečně mnoha krocích tedy buď hodnotu najdeme, nebo se interval vyprázdní.

# Korektnost a složitost

## Věta 1

*Binární vyhledávání vrátí pozici  $x$ , pokud se  $x$  v poli nachází, a jinak správně oznámí její nepřítomnost.*

## Invariant.

*Pokud se  $x$  v poli nachází, existuje jeho výskyt v aktuálním intervalu  $[\ell, r]$ .*

Je-li  $x < A[m]$ , plyne z uspořádání  $A[i] \geq A[m] > x$  pro každé  $i \geq m$  — tuto část lze bezpečně zahodit; případ  $x > A[m]$  je symetrický. V každém kroku se délka ostře zmenší, takže proces skončí.

$$T(n) = T(\lfloor n/2 \rfloor) + \Theta(1) = \Theta(\log n)$$

Zde je  $a = 1$ : rozdělit na dvě části neznamená vždy řešit obě.

# Mergesort: dvě poloviny a lineární spojení

---

```
1 def MergeSort(A):  
2     if  $|A| \leq 1$  then return A  
3     rozděl A na levou polovinu L a pravou polovinu R  
4      $L' \leftarrow \text{MergeSort}(L); \quad R' \leftarrow \text{MergeSort}(R)$   
5     return Merge( $L', R'$ )
```

---

Hlavní částí je operace merge: porovnáváme první dosud nepoužité prvky obou posloupností a menší přesouváme na výstup; po vyčerpání jedné připojíme zbytek druhé.

## Lemma 2 (Korektnost slučování)

*Merge sloučí dvě neklesající posloupnosti o celkové délce  $n$  do jedné neklesající posloupnosti v čase  $\Theta(n)$ .*

## Proč slučování funguje

### Důkaz.

*Invariant: výstup obsahuje právě již zpracované prvky obou vstupů v neklesajícím pořadí a žádný z nich není větší než kterýkoli nezpracovaný prvek.*

Nejmenší nepoužitý prvek musí ležet na začátku jedné ze dvou zbývajících částí — vnitřek seřazené části nemůže obsahovat menší prvek než její začátek.

Výběrem menšího ze dvou začátků invariant zachováme.

Každý prvek přesuneme právě jednou, porovnání je nejvýše  $n - 1$ .

### Věta 3 (Korektnost mergesortu)

*Mergesort vrátí vstupní prvky v neklesajícím pořadí a zachová jejich násobnost.*

### Důkaz.

Indukcí podle  $n$ . Délky 0 a 1 jsou seřazené. Pro  $n > 1$  jsou obě poloviny kratší, takže je rekurze podle indukčního předpokladu seřadí, a lemma je sloučí.

# Stabilita a paměť

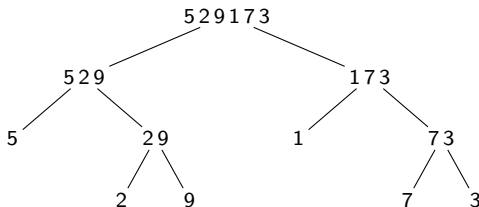
$$T(n) = 2T(n/2) + \Theta(n)$$

- Vybíráme-li při rovnosti nejprve prvek z **levé** poloviny, zachová mergesort vzájemné pořadí prvků se stejným klíčem — je **stabilní**. (Důležité při postupném řazení tabulky podle několika sloupců.)
- Běžná implementace používá pomocné pole velikosti  $\Theta(n)$ ; zásobník rekurze má hloubku  $\Theta(\log n)$ .

Časová složitost tedy není jediný parametr návrhu: za předvídatelný čas platíme dodatečnou paměť.

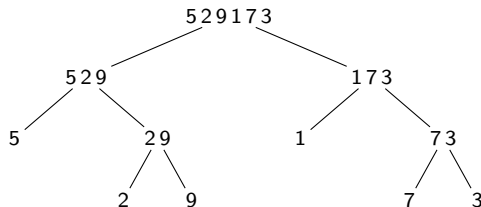
# Mergesort krok za krokem

Vstup  $\langle 5, 2, 9, 1, 7, 3 \rangle$ . Nejprve se *jen dělí* (žádná práce), teprve při návratu se slučuje:



# Mergesort krok za krokem

Vstup  $\langle 5, 2, 9, 1, 7, 3 \rangle$ . Nejprve se *jen dělí* (žádná práce), teprve při návratu se slučuje:



Zpětný chod (výsledky slučování):

29, 259, 37, 137, 123579

## Příklad slučování

Slučujeme  $L = \langle 2, 5, 9 \rangle$  a  $R = \langle 1, 3, 7 \rangle$ :

| krok | čelo $L$ | čelo $R$ | bereme     | výstup |
|------|----------|----------|------------|--------|
| 1    | 2        | 1        | z $R$      | 1      |
| 2    | 2        | 3        | z $L$      | 12     |
| 3    | 5        | 3        | z $R$      | 123    |
| 4    | 5        | 7        | z $L$      | 1235   |
| 5    | 9        | 7        | z $R$      | 12357  |
| 6    | 9        | —        | zbytek $L$ | 123579 |

- Porovnání bylo  $5 = n - 1$  — horní mez z lemmatu.
- Kdykoli je jedna část vyčerpaná, *zbytek druhé se jen připojí* bez porovnávání.
- Při rovnosti bereme z **levé** části — právě tím je mergesort *stabilní*.

Myšlenku slučování popsal už roku 1945 John von Neumann na stroji EDVAC — mergesort patří k vůbec prvním publikovaným netriviálním algoritmům.

## Jak řešit rekurence: 1. rozvinutí

Položme  $T(1) = c$  a  $T(n) = 2T(n/2) + dn$ .

$$T(n) = 2(2T(n/4) + d\frac{n}{2}) + dn = 4T(n/4) + 2dn$$

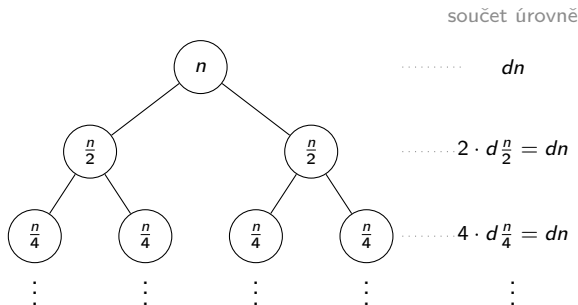
Po  $k$  úrovních

$$T(n) = 2^k T(n/2^k) + k dn.$$

Rekurze končí pro  $n/2^k = 1$ , tedy  $k = \log_2 n$ :

$$T(n) = n T(1) + dn \log_2 n = \Theta(n \log n).$$

## 2. Rekurzivní strom



$\log_2 n$  vnitřních úrovní, každá s prací  $dn$ , a  $n$  listů s cenou  $c$

**Nerozhoduje jen hloubka rekurse**, ale *celková práce celé úrovně*. U binárního vyhledávání má každá úroveň jediný aktivní uzel; u mergesortu počet uzlů roste, ale vstupy se zmenšují.

### 3. Substituční metoda

Navrhujeme kandidátní mez a dokážeme ji indukci.

Ukažme  $T(n) \leq c n \log_2 n$  pro mocniny dvou  $n \geq 2$ .

**Proč ne už od  $n = 1$ ?** Pravá strana je pro  $n = 1$  rovna  $c \cdot 1 \cdot \log_2 1 = 0$  a nerovnost  $T(1) \leq 0$  nezachrání žádné  $c$ . Asymptotická tvrzení ale směřují začínat od libovolného pevného  $n_0$ .

**Základ  $n = 2$ :** pravá strana je  $2c$ ; stačí  $c \geq T(2)/2$ .

**Krok:** pro mocninu dvou  $n \geq 4$  je  $n/2 \geq 2$ , tedy

$$T(n) \leq 2c \frac{n}{2} \log_2 \frac{n}{2} + dn = cn(\log_2 n - 1) + dn \leq cn \log_2 n,$$

pokud  $c \geq d$ . Celkem  $c \geq \max\{T(2)/2, d\}$ .

# Bacha!

1. Základ nelze vždy „ošetřit zvětšením  $c$ “: pravá strana může být v základu nulová; pak je nutné indukci začít později.
2. Musíme dokazovat *konkrétní nerovnost*, ne jen symbolicky dosadit  $O(n \log n)$  do rekurence.

Někdy je navíc nutné indukční hypotézu **zesílit**, například na

$$T(n) \leq c n \log n - e n.$$

Dokazujeme silnější tvrzení, protože slabší se v indukčním kroku neudrží.

## Proč je zesílení hypotézy někdy nutné

Zkusme dokázat, že  $T(n) = 2T(n/2) + 1$  je  $O(n)$ , tedy  $T(n) \leq cn$ :

$$T(n) \leq 2c\frac{n}{2} + 1 = cn + 1 > cn.$$

Indukční krok *neprojde*, přestože mez platí.

## Proč je zesílení hypotézy někdy nutné

Zkusme dokázat, že  $T(n) = 2T(n/2) + 1$  je  $O(n)$ , tedy  $T(n) \leq cn$ :

$$T(n) \leq 2c\frac{n}{2} + 1 = cn + 1 > \textcolor{red}{cn}.$$

Indukční krok *neprojde*, přestože mez platí.

Zesilme hypotézu na  $T(n) \leq cn - b$  pro vhodné  $b > 0$ :

$$T(n) \leq 2\left(c\frac{n}{2} - b\right) + 1 = cn - 2b + 1 \leq cn - b \quad \text{pro } b \geq 1.$$

Nyní krok projde a  $cn - b \leq cn$  dává původní tvrzení.

## Proč je zesílení hypotézy někdy nutné

Zkusme dokázat, že  $T(n) = 2T(n/2) + 1$  je  $O(n)$ , tedy  $T(n) \leq cn$ :

$$T(n) \leq 2c\frac{n}{2} + 1 = cn + 1 > \textcolor{red}{cn}.$$

Indukční krok *neprojde*, přestože mez platí.

Zesílme hypotézu na  $T(n) \leq cn - b$  pro vhodné  $b > 0$ :

$$T(n) \leq 2\left(c\frac{n}{2} - b\right) + 1 = cn - 2b + 1 \leq cn - b \quad \text{pro } b \geq 1.$$

Nyní krok projde a  $cn - b \leq cn$  dává původní tvrzení.

Selže-li tedy indukční krok o aditivní konstantu, je namísto hypotézu **zpřísnit**, ne zeslabit.

# Master theorem

## Věta 4 (Master theorem pro polynomiální $f$ )

*Nechť  $T(n) = a T(n/b) + \Theta(n^d)$ , kde  $a \geq 1$ ,  $b > 1$ ,  $d \geq 0$  jsou konstanty. Potom*

- 1. je-li  $a < b^d$ , pak  $T(n) = \Theta(n^d)$ ;*
- 2. je-li  $a = b^d$ , pak  $T(n) = \Theta(n^d \log n)$ ;*
- 3. je-li  $a > b^d$ , pak  $T(n) = \Theta(n^{\log_b a})$ .*

- Počet uzlů se na každé úrovni násobí  $a$ , práce jednoho uzlu se dělí  $b^d$ ; rozhoduje poměr  $q = a/b^d$ .
- $q < 1$ : dominuje kořen;  $q = 1$ : úrovně jsou vyrovnané, přibude  $\log n$ ;  $q > 1$ : dominují listy, kterých je  $n^{\log_b a}$ .

Obecnější verze pro libovolné  $f$  (s podmínkou pravidelnosti): skripta, věta 4.4.

# Tři případy na příkladech

## Příklad 5

|                                                     |                            |
|-----------------------------------------------------|----------------------------|
| $T_1(n) = 2T_1(n/2) + \Theta(n^2) = \Theta(n^2)$    | dominuje skládání          |
| $T_2(n) = 2T_2(n/2) + \Theta(n) = \Theta(n \log n)$ | vyrovnané úrovně           |
| $T_3(n) = 4T_3(n/2) + \Theta(n) = \Theta(n^2)$      | dominuje počet podproblémů |

**Zaokrouhlení.** Master theorem platí i pro podproblémy velikosti  $\lfloor n/b \rfloor$  či  $\lceil n/b \rceil$  — asymptotický řád se nezmění.

**Master theorem není univerzální.** V tomto tvaru jej nelze použít na  $T(n) = T(n-1) + n$ , na nestejně velké podproblémy, ani na rekurenci, v níž se počet podproblémů mění s  $n$ .

## Malý katalog k procvičení

| rekurence              | $a$ | $b$ | $d$ | výsledek                           |
|------------------------|-----|-----|-----|------------------------------------|
| $T(n) = T(n/2) + 1$    | 1   | 2   | 0   | $\Theta(\log n)$ (bin. vyhl.)      |
| $T(n) = 2T(n/2) + n$   | 2   | 2   | 1   | $\Theta(n \log n)$ (mergesort)     |
| $T(n) = 2T(n/2) + 1$   | 2   | 2   | 0   | $\Theta(n)$ (průchod stromem)      |
| $T(n) = 3T(n/2) + n$   | 3   | 2   | 1   | $\Theta(n^{\log_2 3})$ (Karatsuba) |
| $T(n) = 4T(n/2) + n$   | 4   | 2   | 1   | $\Theta(n^2)$                      |
| $T(n) = 2T(n/2) + n^2$ | 2   | 2   | 2   | $\Theta(n^2)$                      |
| $T(n) = 7T(n/2) + n^2$ | 7   | 2   | 2   | $\Theta(n^{\log_2 7})$ (Strassen)  |

- Poslední řádek je **Strassenovo násobení matic**: sedm součinů poloviční velikosti místo osmi, tedy  $n^{2,807\dots}$  místo  $n^3$ . Stejný princip jako u Karatsubova algoritmu (příští přednáška).
- Řádky s  $a = b^d$  (mergesort) jsou „na hraně“: stačí drobná změna  $f(n)$  a výsledek se překlápí.

## Je mergesort optimální?

Zatím umíme říct „ $O(n \log n)$  stačí“. Zajímavější otázka zní:

*Nemůže existovat třídící algoritmus podstatně rychlejší?*

Odpověď závisí na tom, **co algoritmus smí dělat**. Omezme se na **porovnávací třídění**: o vstupu se dozvídáme *výhradně* dotazy tvaru „je  $a_i < a_j$ ?“.

### Definice 6 (Rozhodovací strom)

Binární strom, v jehož vnitřních vrcholech je jedno porovnání a jehož listy jsou možné výstupní permutace. Běh algoritmu na konkrétním vstupu je cesta z kořene do listu; počet porovnání je její délka.

Jeden algoritmus, jedno  $n$  — jeden pevný strom. Jeho *výška* je počet porovnání v nejhorším případě.

## Dolní mez $\Omega(n \log n)$

### Věta 7

*Každý porovnávací třídící algoritmus potřebuje v nejhorším případě  $\Omega(n \log n)$  porovnání.*

### Důkaz.

Aby byl algoritmus správný, musí pro vstupy z navzájem různých prvků umět vydat *každou* z  $n!$  permutací — jinak by existoval vstup, který nikdy neseřadí. Strom má tedy alespoň  $n!$  listů.

Binární strom výšky  $h$  má nejvýše  $2^h$  listů, takže

$$2^h \geq n! \implies h \geq \log_2(n!).$$

Ze Stirlingovy formule  $n! \geq (n/e)^n$  plyne

$$h \geq n \log_2 n - n \log_2 e = \Omega(n \log n).$$

Mergesort je tedy asymptoticky optimální.

## Kde dolní mez neplatí

Předpoklad byl „jediná operace je porovnání dvou prvků“. Kdo jej poruší, může být rychlejší:

- **Counting sort** — klíče z malého rozsahu  $\{0, \dots, k\}$ ; místo porovnávání *indexujeme* do pole. Čas  $O(n + k)$ .
- **Radix sort** — třídí po číslicích,  $O(d(n + k))$  pro  $d$ -ciferné klíče.

Dolní mez tedy vždy platí *vůči zvolenému modelu výpočtu*. Změníme-li povolené operace, změní se i mez.

Podobně na modelu závisí test prvočíselnosti ze slajdů k přednášce 1 (velikost vstupu vs. hodnota čísla) a nastane znovu u Karatsuby (bitové operace vs. jedno násobení).

## Co se opravdu používá v knihovnách

Knihovní implementace nejsou čistý mergesort ani čistý quicksort:

- **Introsort** (C++ `std::sort`) — quicksort, ale při příliš hluboké rekurzi přepne na heapsort. Získá rychlost quicksortu a záruku  $O(n \log n)$ .
- **Timsort** (Python; Java pro pole objektů) — mergesort, který nejprve najde už seřazené úseky (*runs*) a slučuje je. Na částečně setříděných datech běží až v  $O(n)$ .
- **Malé úseky** se všude řadí *insertion sortem* — pro  $n \lesssim 16$  má nejmenší režii, přestože je kvadratický.

Asymptotická složitost tedy určí vhodné kandidáty; o výběru mezi nimi rozhodují konstanty, paměťová hierarchie a typický tvar dat.

# Počítání inverzí: informace mezi polovinami

## Definice 8 (Inverze)

Pro posloupnost  $A[1], \dots, A[n]$  je **inverzí** dvojice indexů  $(i, j)$  taková, že  $i < j$  a  $A[i] > A[j]$ .

Například  $(2, 4, 1, 3)$  má tři inverze:  $(1, 3)$ ,  $(2, 3)$ ,  $(2, 4)$ , tj. dvojice hodnot  $(2, 1)$ ,  $(4, 1)$ ,  $(4, 3)$ .

Přímé vyzkoušení všech dvojic trvá  $\Theta(n^2)$ . Lze využít dělení na poloviny?

# Počítání inverzí: informace mezi polovinami

## Definice 8 (Inverze)

Pro posloupnost  $A[1], \dots, A[n]$  je **inverzí** dvojice indexů  $(i, j)$  taková, že  $i < j$  a  $A[i] > A[j]$ .

Například  $(2, 4, 1, 3)$  má tři inverze:  $(1, 3)$ ,  $(2, 3)$ ,  $(2, 4)$ , tj. dvojice hodnot  $(2, 1)$ ,  $(4, 1)$ ,  $(4, 3)$ .

Přímé vyzkoušení všech dvojic trvá  $\Theta(n^2)$ . Lze využít dělení na poloviny?

Každá inverze je právě jednoho ze tří typů:

1. oba indexy **vlevo**;
2. oba indexy **vpravo**;
3. **příčná**: první vlevo, druhý vpravo.

První dva počty získáme rekurzivně.

## Zesílený návratový údaj

Kdyby rekurzivní volání vracelo *pouze počet*, neměli bychom dost informací. Necháme je proto vracet **zároveň seřazenou polovinu**.

### Počítání inverzí

Jednoprvková posloupnost: vrátíme 0 a ji samotnou. Jinak rozdělíme na  $L$ ,  $R$ , rekurzí získáme  $c_L$ ,  $c_R$  a seřazené poloviny.

Při slučování: kdykoli je aktuální prvek pravé části menší než aktuální prvek levé části, tvoří inverzi se *všemi* dosud nepoužitými prvky levé části — jejich počet přičteme.

## Zesílený návratový údaj

Kdyby rekurzivní volání vracelo *pouze počet*, neměli bychom dost informací. Necháme je proto vracet **zároveň seřazenou polovinu**.

### Počítání inverzí

Jednoprvková posloupnost: vrátíme 0 a ji samotnou. Jinak rozdělíme na  $L$ ,  $R$ , rekurzí získáme  $c_L$ ,  $c_R$  a seřazené poloviny.

Při slučování: kdykoli je aktuální prvek pravé části menší než aktuální prvek levé části, tvoří inverzi se *všemi* dosud nepoužitými prvky levé části — jejich počet přičteme.

**Všechny přičteme najednou**, protože jsou seřazené: je-li pravý prvek  $r$  menší než nejmenší zbývajících levý prvek  $\ell$ , je menší i než každý další. Žádnou dvojici nezapočítáme dvakrát, protože ji účtujeme právě ve chvíli, kdy přesouváme její *pravý* prvek.

## Výsledek a obecný princip

### Věta 9

*Upravený mergesort vrátí přesný počet inverzí v čase  $\Theta(n \log n)$ .*

### Důkaz.

Indukcí podle délky: rekurzivní volání spočítají první dva typy, argument výše ukazuje, že slučování spočítá každou příčnou inverzi právě jednou. Mimo dvě rekurzivní volání provedeme jedno lineární sloučení, tedy  $T(n) = 2T(n/2) + \Theta(n)$ .

### Zesílený návratový údaj

Rekurzivní volání nemusí vracet jen odpověď požadovanou zadáním. Může vrátit **víc**, a tím umožnit levné skládání.

Zde spolu s počtem inverzí vrací seřazenou posloupnost; díky této pomocné informaci je spojování lineární, a ne kvadratické.

# Počítání inverzí krok za krokem

Vstup  $\langle 2, 4, 1, 3, 5 \rangle$ ; dělíme na  $L = \langle 2, 4 \rangle$  a  $R = \langle 1, 3, 5 \rangle$ .

**Rekurze:**  $L$  nemá inverzi ( $c_L = 0$ ),  $R$  také ne ( $c_R = 0$ ); vrátí se seřazené  $\langle 2, 4 \rangle$  a  $\langle 1, 3, 5 \rangle$ .

**Slučování** — účtujeme, kdykoli bereme prvek z *pravé* části:

| krok | zbytek $L$ | zbytek $R$ | bereme  | přičteme | výstup |
|------|------------|------------|---------|----------|--------|
| 1    | 24         | 135        | 1 z $R$ | +2       | 1      |
| 2    | 24         | 35         | 2 z $L$ | 0        | 12     |
| 3    | 4          | 35         | 3 z $R$ | +1       | 123    |
| 4    | 4          | 5          | 4 z $L$ | 0        | 1234   |
| 5    | —          | 5          | 5 z $R$ | +0       | 12345  |

Celkem  $0 + 0 + 3 = 3$  příčné inverze, konkrétně  $(2, 1)$ ,  $(4, 1)$ ,  $(4, 3)$ .

## Počítání inverzí krok za krokem

Vstup  $\langle 2, 4, 1, 3, 5 \rangle$ ; dělíme na  $L = \langle 2, 4 \rangle$  a  $R = \langle 1, 3, 5 \rangle$ .

**Rekurze:**  $L$  nemá inverzi ( $c_L = 0$ ),  $R$  také ne ( $c_R = 0$ ); vrátí se seřazené  $\langle 2, 4 \rangle$  a  $\langle 1, 3, 5 \rangle$ .

**Slučování** — účtujeme, kdykoli bereme prvek z *pravé* části:

| krok | zbytek $L$ | zbytek $R$ | bereme  | přičteme | výstup |
|------|------------|------------|---------|----------|--------|
| 1    | 24         | 135        | 1 z $R$ | +2       | 1      |
| 2    | 24         | 35         | 2 z $L$ | 0        | 12     |
| 3    | 4          | 35         | 3 z $R$ | +1       | 123    |
| 4    | 4          | 5          | 4 z $L$ | 0        | 1234   |
| 5    | —          | 5          | 5 z $R$ | +0       | 12345  |

Celkem  $0 + 0 + 3 = 3$  příčné inverze, konkrétně  $(2, 1)$ ,  $(4, 1)$ ,  $(4, 3)$ .

**Podstatný je krok 1:** bereme 1 z pravé části, zatímco v levé zbývají *dva* prvky. Oba jsou větší, protože  $L$  je seřazená — započítáme je tedy *najednou*, ne po jednom.

## K čemu je počet inverzí dobrý

- **Míra neuspořádanosti.** 0 inverzí = seřazeno,  $\binom{n}{2}$  = obrácené pořadí.
- **Porovnávání pořadí (Kendallovo  $\tau$ ).** Dva recenzenti seřadí tytéž filmy; počet inverzí mezi jejich pořadími měří neshodu. Odtud plyne i jednoduchý doporučovací systém: hledej uživatele s malým počtem inverzí vůči mně.
- **Odhad práce insertion sortu.** Ten provede přesně tolik výměn, kolik je inverzí — proto je na „skoro seřazených“ datech tak rychlý (a proto jej používá Timsort).

Místo kvadratického počítání dvojic tak dostáváme  $\Theta(n \log n)$ ; stačilo změnit, co rekurze vrací.

# Jak navrhovat dělení

1. **Stejný typ podproblému.** Rekurze má řešit instance se stejnou specifikací, pouze menší.
2. **Pokrok k základnímu případu.** Velikosti všech rekurzivních vstupů musí klesat.
3. **Úplnost rozdělení.** Každá část původního řešení musí být zahrnuta v některém podproblému nebo ve fázi skládání.
4. **Vyváženost.** Dvě poloviny vedou k logaritmické hloubce; rozdělení 1 a  $n - 1$  vede k lineární.
5. **Cena skládání.** Výsledky musí jít spojit levněji než opětovným vyřešením původní instance.
6. **Dostatečný návratový údaj.** Někdy je potřeba vracet pomocnou informaci.

# Jak navrhovat dělení

1. **Stejný typ podproblému.** Rekurze má řešit instance se stejnou specifikací, pouze menší.
2. **Pokrok k základnímu případu.** Velikosti všech rekurzivních vstupů musí klesat.
3. **Úplnost rozdělení.** Každá část původního řešení musí být zahrnuta v některém podproblému nebo ve fázi skládání.
4. **Vyváženost.** Dvě poloviny vedou k logaritmické hloubce; rozdělení 1 a  $n - 1$  vede k lineární.
5. **Cena skládání.** Výsledky musí jít spojit levněji než opětovným vyřešením původní instance.
6. **Dostatečný návratový údaj.** Někdy je potřeba vracet pomocnou informaci.

## Příklad 10 (Nevyvážené dělení)

$T(n) = T(n - 1) + \Theta(n)$  dá po rozvinutí  $\Theta(n + (n - 1) + \dots + 1) = \Theta(n^2)$ .  
*Samotné použití rekurze rychlost nezaručuje.*

## Shrnutí

Rozděl a panuj je způsob návrhu algoritmu, v němž menší *nezávislé* instance řešíme stejnou metodou a jejich výsledky cíleně skládáme.

- Korektnost dokazujeme indukcí kopírující rekurzivní strukturu.
- Složitost zapisujeme rekurencí, v níž oddělíme rekurzivní volání od práce aktuálního uzlu.
- Rekurenci chápeme rozvinutím a stromem; Master theorem je zkratka pro pravidelné vyvážené případy.
- Vhodné skládání dokáže započítat i informaci *přes hranici* podproblémů (příčné inverze).

Příště se podíváme na dělení, které sníží nejen velikost podproblémů, ale i jejich počet.

# K procvičení

1. Přepište binární vyhledávání na polootevřený interval  $[\ell, r)$ . Podmínka prázdnoty? Nové intervaly? Proč se vždy zmenší?
2. Rekurzivním stromem vyřešte  $T(n) = T(n/2) + \Theta(1)$ .
3. Pomocí Master theoremu:  $T_1(n) = 3T_1(n/2) + n$ ,  $T_2(n) = 4T_2(n/2) + n^2$ .
4. Jaký je největší možný počet inverzí posloupnosti délky  $n$ ?
5. Proč zůstává mergesort  $\Theta(n \log n)$  i pro dělení na  $\lfloor n/2 \rfloor$  a  $\lceil n/2 \rceil$ ?
6. Navrhněte algoritmus typu rozděl a panuj pro hledání minima i maxima s nejvýše  $3n/2 - 2$  porovnáními.
7. Najděte největší součet souvislého úseku v čase  $O(n \log n)$ .  
*Nápověda:* optimální úsek leží celý vlevo, celý vpravo, nebo překračuje místo rozdělení.

# Literatura

 T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein.  
*Introduction to Algorithms.*  
MIT Press (2001).

 S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani.  
*Algorithms.*  
McGraw–Hill (2006).

 J. Kleinberg, É. Tardos.  
*Algorithm Design.*  
Addison–Wesley (2006), kap. 5.