8. Rozděl a panuj

Známá strategie *Divide et Impera* (někdy překládáno spíše jako „Rozeštvi a panuj“) pochází z dob antického Říma[1], kdy panovník zasel nesvár mezi místní kmeny (čímž je rozdělil a rozeštval) a následně přišel, urovnal napětí, aby je mohl opět ovládnout a panovat a aby byli všichni spokojení.

Tato strategie ovšem přetvrává (ač třeba V jiných oblastech) a pomáhá řešit ty problémy, které se dají rozdělit na menší jednodušší podproblémky.

Jak tedy algoritmus typu *rozděl a panuj* pracuje? Mějme problém, který má tu vlastnost, že když jej rozdělíme na vhodné podproblémy, které mají stejný charakter, a ty vyřešíme, složením jejich řešení můžeme získat řešení původního problému. Algoritmus tedy bude rekurzivně volat sám sebe, než se dostane k podproblému nějaké konstantní velikosti, který už umí vyřešit triviálně, a pak se začne z rekurze vracet a skládat jednotlivá dílčí řešení.

Příklad 1 – MergeSort:

Tento třídící algoritmus pracuje na principu, že vstup rozdělíme na dvě (skoro) stejně velké části, které rekurzivním voláním setřídíme, a nakonec výsledné dvě posloupnosti slijeme do jedné.

Algoritmus:

1. *Vstup:* posloupnost $x_1, \ldots, x_n$.
2. Pokud $n \leq 1 \Rightarrow$ vrátíme vstup.
3. $y_1, \ldots, y_{\lfloor n/2 \rfloor} \leftarrow$ *MergeSort* $(x_1, \ldots, x_{\lfloor n/2 \rfloor})$.
4. $z_1, \ldots, z_{\lceil n/2 \rceil} \leftarrow$ *MergeSort* $(x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n)$.
5. Vrátíme $\mathit{Merge}(y_1, \ldots, y_{\lfloor n/2 \rfloor}; z_1, \ldots, z_{\lceil n/2 \rceil})$.

Na slití dvou setříděných posloupností do jedné používáme funkci Merge:

$\mathit{Merge}(y_1, \ldots, y_a; z_1, \ldots, z_b)$**:**

1. $i \leftarrow 1, j \leftarrow 1, k \leftarrow 1$.
2. Dokud $k \leq a + b$:
3. Je-li $(j > b)$ nebo $(i \leq a) \& (y_i < z_j) \Rightarrow x_k \leftarrow y_i, k++, i++$.
4. Jinak $\Rightarrow x_k \leftarrow z_j, k++, j++$.
5. Vrátíme $x_1, \ldots, x_n$.

Pozorování: *Merge* trvá $\Theta(n)$, neboť každou ze slévaných posloupností projdeme právě jednou.

Časová složitost MergeSortu:

Rozdělování a slévání nám trvá lineárně dlouho, takže pro časovou složitost MergeSortu platí tato rekurentní rovnice:

$$T(n) = 2 \cdot T(n/2) + c \cdot n.$$

[1] Ač se touto strategií římští panovníci řídili, není nalezen žádný antický zdroj výroku; ten je připisován až renesančnímu N. Machiavellimu.

Přitom $T(n)$ zde značí čas strávený na vstupu délky n a c je nějaká vhodná konstanta. Pro jednoduchost BÚNO předpokládejme, že n je mocnina dvojky. Zároveň si jednotky zvolme tak, že bude platit $T(1) = 1$. Můžeme si tedy rekurentní vztah rozepsat následovně:

$$\begin{aligned} T(n) &= 2 \cdot (2T(n/4) + c \cdot n/2) + c \cdot n = 4T(n/4) + 2cn = \\ &= 4(2T(n/8) + c(n/4)) + 2cn = 8T(n/8) + 3cn = \ldots \\ &= 2^k T(n/2^k) + kcn. \end{aligned}$$

Pokud zvolíme $k = \log_2 n$, získáme:

$$T(n) = 2^{\log_2 n} \cdot T(n/2^{\log_2 n}) + \log_2 n \cdot c \cdot n = n \cdot T(1) + c \cdot n \cdot \log_2 n = \Theta(n \log n).$$

Ke stejnému výsledku můžeme ale dojít také úplně jinou cestou. Představme si strom rekurzivních volání. Každý vrchol má dva syny (dělíme vstup na dvě části), v nichž jsou vstupy poloviční velikosti. V každém vrcholu trávíme čas lineární s velikostí jeho vstupu, součet velikostí vstupů přes každou hladinu je n a hloubka stromu musí být $\Theta(\log n)$. Vyjde nám tedy, že $T(n) = \Theta(n \log n)$.[2]

Paměťová složitost MergeSortu:

$M(n) = d \cdot n + M(n/2) = d \cdot n + d \cdot n/2 + d \cdot n/4 + \ldots \leq 2dn = \Theta(n)$.

Tento vztah platí pro nějakou vhodnou konstantu d. Můžeme ho opět nahlédnout například ze stromu rekurzivních volání. Podívejme se na libovolnou cestu od kořene do listu. V jednotlivých vrcholech potřebujeme paměti přesně $d \cdot n/2^k$, kde k je číslo hladiny. Když tyto hodnoty sečteme přes všechny vrcholy na této cestě, výsledek bude konvergovat k $2dn$, což dává paměťovou složitost $\Theta(n)$.

Závěr: Mergesort běží V čase $\Theta(n \log n)$ a paměti $\Theta(n)$. Lineární paměťová složitost není výhodná, ale na druhou stranu se tento algoritmus velmi hodí například na třídění lineárních spojových seznamů.

Přiklad 2 – Násobení čísel:

Pokud násobíme dvě čísla X a Y (obě délky n; pokud bylo jedno kratší, tak ho doplníme nulami zleva tak, aby byla obě čísla stejně dlouhá) způsobem, který nás učili na základní škole, výsledný algoritmus má časovou složitost $\Theta(n^2)$. Protože se jedná o dost častou operaci, zamysleme se, zda by nešla zrychlit. Nasměrujme naše úvahy na postup *rozděl a panuj*. Rozdělíme každého činitele na dvě stejně dlouhé části. Pro jednoduchost předpokládejme, že toto rozštěpení činitele proběhne vždy bez zbytku:

$$X = A \cdot 10^{n/2} + B, \qquad Y = C \cdot 10^{n/2} + D.$$

Zde A, B, C, D jsou už jen $(n/2)$-ciferná čísla. Původní součin získáme jako:

$$XY = (A \cdot 10^{n/2} + B)(C \cdot 10^{n/2} + D) = AC \cdot 10^n + (AD + BC) \cdot 10^{n/2} + BD.$$

[2] Po pozornějším zamyšlení si čtenář může uvědomit, že se jedná vlastně pouze o jiný pohled na stejný důkaz jako rozepisování rekurentního vzorce.

Nyní, jak vidíme, stačí spočítat součin čtyř $(n/2)$-ciferných čísel a pak výsledky spolu sečíst. Uvažme, jakou bude mít tento algoritmus časovou složitost:

$$T(n) = 4T(n/2) + cn.$$

Toto platí pro nějakou vhodnou konstantu c (výraz cn je režie na sčítání). Jednotky si zvolme tak, aby platilo:

$$T(1) = 1.$$

Jak takovou rekurenci vyřešíme? Máme opět dvě možnosti:

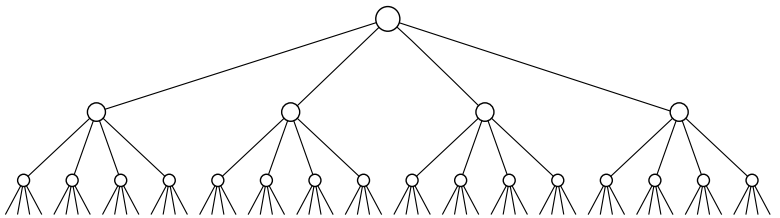
1. způsob: Řešení rozepsáním rekurentního vztahu:

$$\begin{aligned} T(n) &= 4T(n/2) + cn = \\ &= 4 \cdot (4T(n/4) + cn/2) + cn = 4^2T(n/4) + 2cn + cn = 4^2T(n/4) + 3cn = \\ &= 4^2 \cdot (2T(n/8) + cn/4) + 3cn = 4^3T(n/8) + 4cn + 3cn = 4^3T(n/8) + 7cn = \\ &\dots \end{aligned}$$

Odtud snadno vypozorujeme, že jednotlivé vztahy se vyvíjejí podle vzorce $T(n) = 4^kT(n/2^k) + (2^k - 1)cn$. Pro $k = \lceil \log_2 n \rceil$ je ovšem $2^k \leq 1$, takže $T(n/2^k) = \Theta(1)$ a dostaneme (horní celou část zanedbáme, ta ovlivní jen konstanty):

$$T(n) = 4^{\log_2 n}\Theta(1) + (2^{\log_2 n} - 1)cn = n^2\Theta(1) + (n-1)cn = \Theta(n^2).$$

2. způsob: Úvaha o stromu: Nakreslíme si strom rekurzivních volání našeho algoritmu:



Na i-té hladině stromu leží 4^i vrcholů, v nich jsou vstupy velikosti $n/2^i$, takže na celé hladině trávíme čas celkem $\Theta(4^i \cdot n/2^i) = \Theta(2^in)$. Velikosti vstupů klesají exponenciálně, takže celý strom je hluboký $k = \log_2 n$ (opět si dovolíme zapomenout na horní celou část). Celkem tedy spotřebujeme čas $\sum_{i=0}^{k} \Theta(2^in) = \Theta(n \cdot \sum_{i=0}^{k} 2^i) = \Theta(n^2)$.

Oba způsoby analýzy se tedy shodují na tom, že náš algoritmus má kvadratickou časovou složitost a že jsme si oproti klasickému algoritmu nikterak nepomohli.

Podívejme se ještě jednou na to, jak se náš algoritmus větví:

hloubka	počet úloh	velikost podúlohy
0	4^0	$n/2^0$
1	4^1	$n/2^1$
2	4^2	$n/2^2$
3	4^3	$n/2^3$
$\vdots$	$\vdots$	$\vdots$
k	4^k	$n/2^k$

Naskýtá se otázka, jestli bychom nemohli časovou složitost zlepšit. Toho bychom mohli dosáhnout například zlepšením členu cn v naší rekurenci, čili zefektivněním spojování podúloh. To ovšem není příliš nadějné (pokud čtenář nevěří, může si to dokázat), takže místo toho využijeme druhou šanci a to omezení větvení ze čtyř větví na tři. Připomeňme si, že potřebujeme spočítat:

$$XY = AC \cdot 10^n + (AD + BC) \cdot 10^{n/2} + BD.$$

Přitom ale nepotřebujeme znát součiny AD ani BC samostatně, nám stačí zjistit hodnotu celého výrazu $AD + BC$. Když budeme znát hodnotu výrazů: AC, BD a $(A + B)(C + D)$ (k tomu nám stačí 3 násobení a 2 sčítání), tak můžeme výraz $AD + BC$ získat následovně:

$$(A + B)(C + D) - AC - BD = AC + AD + BC + BD - AC - BD = AD + BC$$

Nyní nám již stačí jen tři násobení, ale potřebujeme dvě sčítání a jedno odčítání navíc. Nicméně tato komplikace je zanedbatelná oproti práci ušetřené menším větvením. (Nová dvě sčítání a jedno odčítání se V časové složitosti schová do cn.) Podívejme se opět na tabulku:

hloubka	počet úloh	velikost podúlohy
0	3^0	$n/2^0$
1	3^1	$n/2^1$
2	3^2	$n/2^2$
3	3^3	$n/2^3$
$\vdots$	$\vdots$	$\vdots$
k	3^k	$n/2^k$

Náš rekurentní vztah po zbavení se jednoho násobení vypadá následovně:

$$T(n) = 3T(n/2) + cn.$$

(L a P), takže už spolu porovnány nikdy ani nebudou. Pravděpodobnost, že y_i nebo y_j se staly pivotem pro posloupnost $y_i, \ldots, y_j$ je rovna $\frac{2}{j-i+1}$ (neboť pivota vybíráme rovnoměrně náhodně). Proto můžeme konečně vyjádřit:

$$\mathbb{E}[C] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = \sum_{d=1}^{n-1} \sum_{n=1}^{n-d} \frac{2}{d+1} \le \sum_{d=1}^{n-1} \frac{2n}{d+1} = 2n \sum_{d=2}^{n} \frac{1}{d} = \Theta(n \cdot \log n).$$

♡

Algoritmus: $QS(x = x_1, \ldots, x_n)$**:**

1. Pokud $n \leq 1 \Rightarrow$ vrátíme x.
2. Vybereme pivota.
3. Rozdělíme posloupnost na podposloupnosti L, S, P.
4. $L \leftarrow$ *QS(L)*, $P \leftarrow$ *QS(P)*
5. Vrátíme $L + S + P$.

Pozorování:

Když bude pivot vždy medián: $T(n) = \Theta(n) + 2T(n/2) \Rightarrow T(n) = \Theta(n \cdot \log n)$

Když bude pivot vždy minimum: $T(n) = \Theta(n) + T(n-1) \Rightarrow T(n) = \Theta(n^2)$

Poznámka: Hledání mediánu lineárně je sice hezké a asymptoticky lineární, ale má tak ošklivé multiplikativní konstanty, že se V praxi nepoužívá. Dokažme si tedy, že Quicksort s náhodnou volbou pivota má průměrnou časovou složitost také $\Theta(n \cdot \log n)$.

Věta: Quicksort s náhodnou volbou pivota provede ve střední hodnotě $\Theta(n \cdot \log n)$ porovnání.

Důsledek: Quicksort má průměrnou časovou složitost $\Theta(n \cdot \log n)$.

Důkaz: Nechť C značí počet porovnání, $y_1, \ldots, y_n$ je vstup V setříděném pořadí a C_{ij} značí počet porovnání prvků y_i s y_j pro $1 \leq i < j \leq n$. Uvědomme si, že dva prvky porovnává algoritmus nejvýše jednou, a to pouze tehdy, když je jeden z nich pivot. Proto C_{ij} nabývá hodnot 0 nebo 1. C_{ij} je tedy indikátor jevu, že byly prvky y_i a y_j porovnány.

Počet všech porovnání je pak součet jednotlivých C_{ij}, čili

$$C = \sum_{i,j} C_{ij}.$$

Proto platí:

$$\mathbb{E}[C] = \mathbb{E}\left[\sum_{i,j} C_{ij}\right].$$

Díky linearitě střední hodnoty můžeme tvrdit, že:

$$\mathbb{E}[C] = \sum_{i,j} \mathbb{E}[C_{ij}].$$

Střední hodnota indikátoru jevu, že prvky y_i a y_j byly porovnány, je rovna pravděpodobnosti, že C_{ij} je 1.

$$\mathbb{E}[C_{ij}] = P[C_{ij} = 1].$$

Aby C_{ij} se rovnalo 1 pro prvky y_i a y_j, tak se musel pro podposloupnost $y_i, \ldots, y_j$ stát pivotem jako první jeden z y_i nebo y_j. Kdyby se tak nestalo, tak spolu při tomto rozdělování prvky y_i a y_j nebudou porovnány a dostanou se každý do jiné skupiny

Opět uvažme, kolik práce spotřebujeme v součtu přes všechny hladiny (hloubka stromu k je opět $\lceil \log_2 n \rceil$ a horní celou část zanedbáme):

$$\sum_{i=0}^{k} 3^i \cdot \frac{n}{2^i} = \sum_{i=0}^{k} \left(\frac{3}{2}\right)^i \cdot n = n \cdot \sum_{i=0}^{k} \left(\frac{3}{2}\right)^i = n \cdot \frac{\left(\frac{3}{2}\right)^{k+1} - 1}{\frac{3}{2} - 1} =$$

$$= n \cdot \frac{\left(\frac{3}{2}\right)^{k+1} - 1}{\frac{1}{2}} = 2 \cdot n \cdot \left[\left(\frac{3}{2}\right)^{k+1} - 1\right] = \Theta\left(n \cdot \left(\frac{3}{2}\right)^{\log_2 n}\right) =$$

$$= \Theta\left(n \cdot \frac{3^{\log_2 n}}{2^{\log_2 n}}\right) = \Theta\left(n \cdot \frac{3^{\log_2 n}}{n}\right) = \Theta\left(3^{\log_2 n}\right) = \Theta\left((2^{\log_2 3})^{\log_2 n}\right) =$$

$$= \Theta\left(2^{(\log_2 n) \cdot \log_2 3}\right) = \Theta\left((2^{\log_2 n})^{\log_2 3}\right) = \Theta\left(n^{\log_2 3}\right) = \Theta\left(n^{1.585}\right).$$

Upravený algoritmus má už tedy lepší časovou složitost, konkrétně $\Theta(n^{1.585})$. V praxi bychom samozřejmě činitele neštěpili až na jednociferná čísla, ale zastavili se u nějaké dostatečně malé délky (řekněme 50 cifer) a tam přepnuli na kvadratický algoritmus, který má menší režii.

(Mimochodem, asymptoticky tato složitost není nejlepší známá, pro násobení čísel existují efektívnější algoritmy, které dosahují časové složitosti $\Theta(n \log n)$, ale jednak mají vysoké multiplikativní konstanty a druhak jsou v nich už potřeba trochu pokročilejší techniky, jako je diskrétní Fourierova transformace, takže si je necháme na příští semestr.)

Kuchařková věta *(Master Theorem)*

Metody řešení rekurentních rovnic z předchozích dvou příkladů by jistě fungovaly i na jiné algoritmy, ale proč pokaždé zbytečně upravovat tolik výrazů? Raději si dokážeme obecnou větu, která půjde použít na většinu takovýchto rekurencí. Říká se jí *Master Theorem* nebo také (vzhledem k tomu, jak se používá) *Kuchařková věta*.

Věta: *(Master Theorem)*

Předpokládejme, že $T(1) = \Theta(1)$ a $T(n) = a \cdot T(\lceil \frac{n}{b} \rceil) + \Theta(n^d)$, kde $a \geq 1$, $b > 1$, $d \geq 0$ a $a, b \in \mathbb{N}$. Potom $T(n)$ je:

$\Theta(n^d)$ když $a < b^d$,
$\Theta(n^d \cdot \log n)$ když $a = b^d$,
$\Theta(n^{\log_b a})$ když $a > b^d$.

Důkaz: Předpokládejme, že $n = b^k, k \in \mathbb{N}$, aby platilo $\lceil \frac{n}{b} \rceil = \frac{n}{b}$. Použijeme opět „důkaz stromem“. Strom rekurzivních volání se vždy větví na stejný počet větví,

konkrétně a, a velikosti vstupů klesají b-krát. Podívejme se na tabulku:

počet vrcholů na hladině	velikost vstupu	čas ve vrcholu	čas na hladině
1	n	$\Theta(n^d)$	$\Theta(n^d)$
a	n/b^1	$\Theta((n/b^1)^d)$	$\Theta(a^1 \cdot (n/b^1)^d)$
a^2	n/b^2	$\Theta((n/b^2)^d)$	$\Theta(a^2 \cdot (n/b^2)^d)$
a^3	n/b^3	$\Theta((n/b^3)^d)$	$\Theta(a^3 \cdot (n/b^3)^d)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
a^k	n/b^k	$\Theta((n/b^k)^d)$	$\Theta(a^k \cdot (n/b^k)^d)$

Celkový čas potřebný na vyřešení všech dílčích podúloh je následovný:

$$T(n) = \sum_{i=0}^{k} \Theta\left(a^i \left(\frac{n}{b^i}\right)^d\right) = \sum_{i=0}^{k} \Theta\left(n^d \left(\frac{a}{b^d}\right)^i\right) = \Theta\left(n^d \cdot \sum_{i=0}^{k} \left(\frac{a}{b^d}\right)^i\right).$$

Všimněme si sumy $\sum_{i=0}^{k} \left(\frac{a}{b^d}\right)^i$. Jedná se vlastně o geometrickou řadu s kvocientem $q = \frac{a}{b^d}$. Rozlišme následující případy:

1. $q < 1$: Práce na jednotlivých hladinách exponenciálně ubývá a součet řady (i kdyby byla nekonečná) se dá omezit nějakou konstantou, tedy $T(n) = \Theta(n^d)$.

2. $q = 1$: Práce na jednotlivých hladinách je stejně, to znamená, že suma je právě $1 + \log_b n$, a tedy $T(n) = \Theta(n^d \cdot \log n)$.

3. $q > 1$: Práce na jednotlivých hladinách přibývá, takže musíme geometrickou řadu sečíst poctivě. Víme, že součet geometrické řady od 0 do k s kvocientem q a prvním členem 1 je: $\frac{q^{k+1}-1}{q-1}$, což přibližne odpovídá q^k. Pak tedy platí:

$$T(n) = \Theta(n^d \cdot q^{\log_b n}).$$

Tento výraz vypadá poněkud ošklivě, ale ještě ho trochu (alespoň kosmeticky) upravíme:

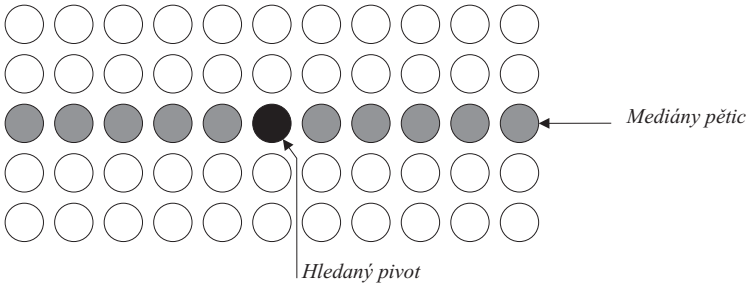
$$\Theta\left(n^d \cdot q^{\log_b n}\right) = \Theta\left(\frac{a^{\log_b n} \cdot n^d}{(b^d)^{\log_b n}}\right) = \Theta\left(\frac{\left(b^{\log_b a}\right)^{\log_b n} \cdot n^d}{(b^d)^{\log_b n}}\right) =$$

$$= \Theta\left(\frac{\left(b^{\log_b n}\right)^{\log_b a} \cdot n^d}{\left(b^{\log_b n}\right)^d}\right) = \Theta\left(\frac{n^{\log_b a} \cdot n^d}{n^d}\right) = \Theta\left(n^{\log_b a}\right).$$

Tyto tři případy přesně odpovídají rozdělení případů v tvrzení věty.

Vraťme se nyní k možnosti, kdy n není mocnina b. Pro obecné $n \geq 1$ zvolme:

- n^+ ... nejbližší vyšší mocnina b
- n^- ... nejbližší nižší mocnina b



Pětice

Nyní už se tedy můžeme pustit do výpočtu časové složitosti. V každém kroku funkce zavolá sama sebe nejdříve na vstup velikosti $\frac{n}{5}$ a poté na vstup velikosti nejvýše $\frac{7}{10}n$. Ostatní operace zůstávají lineární. Časovou složitost V nejhorším případě tedy můžeme zapsat rekurentním vzorcem:

$$T(n) = \Theta(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right).$$

Tento rekurentní vzorec ale zatím neumíme obecně řešit. Mohli bychom ho postupně rozepsat, ale trvalo by dlouho, než bychom z toho něco vykoukali. Lepší bude rovnou dokázat, že tento rekurentní vzorec odpovídá lineární složitosti, zkusíme tedy dosadit $T(n) = cn$:

$$cn = n + \frac{1}{5}cn + \frac{7}{10}cn \quad \Longleftrightarrow \quad c = 10.$$

Tímto jsme tedy dokázali, že časová složitost tohoto algoritmu V nejhorším případě je $\Theta(n)$. Jelikož to rychleji určitě není možné, můžeme na závěr této kapitoly zformulovat tuto větu:

Věta: Hledání k-tého nejmenšího prvku V posloupnosti délky n má časovou složitost V nejhorším případě $\Theta(n)$. ♡

K zamyšlení: Proč jsme zvolili zrovna pětice? Jak by to dopadlo pro trojice? a jak pro sedmice? Fungoval by takový algoritmus? Byl by také lineární?

Quicksort

Již jsme se seznámili s třídícím algoritmem *Mergesort*. Podívejme se teď na algoritmus, který je také založen na metodě *rozděl a panuj*, ale chová se V podstatě opačně. Namísto slévání menších setříděných posloupností posloupnost vždy rozdělí na dvě pokud možno podobně velké části L a P, kdy V L jsou čísla menší než nějaký pivot a v P čísla větší. Pivot tvoří samostatnou část S. Obě části L i P rekurzivně setřídí a následně vrátí posloupnost, kde budou setříděné prvky z L, pak pivot z S a poté setříděné prvky z P. Využívá tedy podobnou myšlenku jako hledání k-tého nejmenšího prvku.

4. $\forall i : m_i \leftarrow$ medián(P_i).

5. $p \leftarrow$ *Select*$(m_1 \dots m\lceil n/5\rceil, \lceil n/10\rceil)$.

6. Rozdělíme X na L, S, P

7. Rekurzivně se zavoláme na jednu z L, S, P (tu, ve které se má vyskytovat hledaný prvek)

Náš deterministický algoritmus tedy rozšiřuje původní algoritmus tak, že vždy, když je potřeba zvolit pivota z posloupnosti X, tak si posloupnost rozdělí na pětice a v každé spočte medián. Následně potřebuje medián z těchto mediánů, což bude výsledný pivot – medián V původní posloupnosti. Ten získá tak, že se znovu zavolá na *Select* s parametry posloupnost mediánů a číslo $\lceil n/10\rceil$, neboť potřebuje právě prostřední prvek této posloupnosti, která má délku $\lceil n/5\rceil$. Když máme konečně výhodného pivota nalezeného, tak můžeme pokračovat, že si posloupnost opět rozdělíme na tři hromádky – prvky menší (L), stejné (S) a větší (P) než pivot. Následně jen vybereme hromádku, která odpovídá umístění k-tého nejmenšího prvku, a na tu se rekurzivně zavoláme.

Abychom dokázali, že tento algoritmus bude mít opravdu lineární časovou složitost, musíme si nejdříve dokázat následující lemma:

Lemma: V každém kroku vypadne alespoň $3/10 \cdot n$ prvků.

Důkaz: V důkazu budeme trochu čtenáře šidit tím, že budeme předpokládat celočíselné výsledky (např. že počet prvků posloupnosti, na kterou se zavoláme, je vždy dělitelný pěti). Čtenář si může jako cvičení dokázat, že kdyby výsledky nevycházely celočíselně, tak to nevadí.

Představme si vybrané pětice seřazené podle velikosti od největšího prvku a zakresleme je do sloupců. Jejich mediány tedy vyplňují prostřední řadu. Tyto pětice pak seřaďme podle velikosti jejich mediánů (nejmenší vlevo)[3]. Hledaný pivot se tedy nachází (pokud předpokládáme pro jednoduchost lichý počet pětic) přesně uprostřed. o prvcích nad pivotem a napravo od něj můžeme určitě říct, že jsou větší nebo rovny pivotu, prvky pod ním a nalevo od něj jsou zase určitě menší nebo rovny pivotu.

Podle konstrukce algoritmu tedy zaručeně vypadne jedna nebo druhá skupina prvků. Obě tyto skupiny přitom obsahují, jak je vidět z obrázku, alespoň $3/10 \cdot n$ prvků. ♡

[3] Algoritmus ovšem nikde takto pětice neřadí! Jen nám to pomůže v důkazu správnosti.

Platí tedy, že $\frac{n}{b} \leq n^- \leq n \leq n^+ \leq n \cdot b$. Zajisté i platí $T(n^-) \leq T(n) \leq T(n^+)$. Ale $T(n^-) = \Theta((n^-)^c) = \Theta(n^c)$ a $T(n^+) = \Theta((n^+)^c) = \Theta(n^c)$. Pak tedy i $T(n) = \Theta(n^c)$. Věta tedy platí i v případě, že n není mocnina b. ♡

Příklad: Porovnejme si některé známé algoritmy a jejich časovou složitost pomocí *Master Theoremu*:

algoritmus	a	b	d	časová složitost
Mergesort	2	2	1	$\Theta(n \cdot \log n)$
Násobení I.	4	2	1	$\Theta(n^2)$
Násobení II.	3	2	1	$\Theta(n^{\log_2 3})$
Binární vyhledávání	1	2	0	$\Theta(\log n)$

Hledání k-tého nejmenšího prvku (mediánu)

V tomto oddílu se budeme zabývat tím, jak co nejrychleji najít V jakékoli posloupnosti n čísel k-tý nejmenší prvek, popřípadě medián. Pro ty, kdo medián neznají, tu máme definici:

Definice: *Medián* posloupnosti $a_1, a_2, \dots, a_n$ je takové a_i, že nejvýše $n/2$ prvků je menších než a_i a nejvýše $n/2$ prvků je větších než a_i. Platí tedy, že medián je buď $\lfloor n/2\rfloor$-tý, nebo $\lceil n/2\rceil$-tý nejmenší prvek posloupnosti.

Nejjednodušším řešením by určitě bylo celou posloupnost nejdříve setřídit a pak už jednoduše vybrat požadovaný prvek. To bychom dokázali V celkem slušném čase $\Theta(n \log n)$, ale už teď můžeme prozradit, že to jde V čase $\Theta(n)$. Jak?

Použijme metodu *rozděl a panuj*. Nějakým způsobem si zvolíme jeden prvek posloupnosti a nazveme ho *pivot*. Poté rozdělíme zadanou posloupnost na tři disjunktní množiny. do první dáme všechny prvky menší než pivot, do druhé rovné pivotu jako pivot a do třetí větší než pivot. Tímto máme zajištěno, že prvky z první množiny jsou určitě menší než prvky z druhé a ty než prvky z třetí. O tom, jak jsou prvky uspořádány uvnitř těchto množin, ale nic nevíme.

V posledním kroku našeho algoritmu se pak rozhodneme, na kterou množinu svůj algoritmus rekurzivně zavoláme. Pokud je k menší nebo rovno než velikost první množiny, pokračujeme V první množině, pokud je k menší nebo rovno než součet velikostí první a druhé množiny, pak hledaným prvkem je právě vybraný pivot a algoritmus skončí, a nakonec pokud ani jedna podmínka splněna nebyla, pustíme se do hledání ve třetí množině, ovšem už nehledáme k-tý nejmenší prvek, ale l-tý, kde l se rovná k minus velikost prvních dvou množin. Pro větší názornost zapíšeme tento algoritmus formálněji:

Select(k, X)**:** (Hledání k-tého nejmenšího prvku V množine X)

1. Jestliže $|X| \leq 1$, vyřešíme triviálně.
2. Zvolíme pivota $p \in X$.
3. Rozdělíme množinu X na tři podmnožiny: $L = \{x \in X; x < p\}$, $S = \{x \in X; x = p\}, P = \{x \in X; x > p\}$.
4. Jestliže $k \leq |L|$, vrátíme výsledek funkce *Select*(k, L).

5. Jestliže $|L| < k \leq |L| + |S|$, vrátíme pivota p.
6. Jestliže $|L| + |S| < k$, vrátíme výsledek funkce $Select(k - |L| - |S|, P)$.

Na první pohled je vidět, že se algoritmus zastaví (vstup se vždy zmenší alespoň o 1) a že vydá vždy správný výsledek. Jak je to ovšem s časovou složitostí? Rozdělení do množin a podmínky V druhém a třetím kroku mají lineární složitost, čemuž se nevyhneme. Při nešťastné volbě pivota se nám může stát, že počet rekurentních volání může být až n, tedy celková složitost V nejhorším případě je $\Theta(n^2)$, čímž jsme si oproti prostému setřídění ještě pohoršili. Co s tím? Jak je vidět, velmi důležitá je volba pivota. Tu můžeme provést několika způsoby:

a) Pivot by se V setříděné posloupnosti vyskytoval uprostřed, vstup by se tedy stále půlil. Časovou složitost vypočteme z rekurentního zápisu:

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n) = \Theta\left(n + \frac{n}{2} + \frac{n}{4} + \ldots\right) = \Theta(n).$$

To by bylo sice skvělé, ale nalezení takového pivota je vlastně vyřešení úlohy hledání mediánu, o což se snažíme. Tedy jsme si vůbec nepomohli.

b) Pivot by se V setříděné posloupnosti náchazel V prostředních dvou čtvrtinách (nazvěme tento prvek „*lžimedián*“). Tím bychom V každém kroku určitě odstranili množinu velikosti alespoň čtvrtiny vstupu. Časová složitost tohoto řešení by byla:

$$T(n) = T\left(\frac{3}{4}n\right) + \Theta(n) = \Theta\left(n + \frac{3}{4}n + \frac{9}{16}n + \ldots\right) = \Theta(n).$$

Tímto bychom tedy také dosáhli lineární časové složitosti. Ale jak vybrat pivota tak, aby se nacházel V prostředních dvou čtvrtinách a aby nám nám to nepokazilo lineární složitost?

Zkusme vybrat pivota náhodně. Pravděpodobnost, že vytáhneme zrovna lžimedián, je alespoň 1/2. (Pokud by se prvky nemohly opakovat a byl jich sudý počet, byla by to přesně 1/2.) Tuto pravděpodobnost si označme p.

Volba lžimediánu

1. Vybereme rovnoměrně náhodně pivota z množiny X.
2. Otestujeme, zda je pivot lžimedián.
3. Pokud není $\Rightarrow$ pokračujeme znovu od začátku, jinak konec.

Pivota vybíráme rovnoměrně náhodně, tedy každý prvek posloupnosti má stejnou pravděpodobnost, že bude vybrán.

Označme si T náhodnou veličinu značící dobu běhu algoritmu. Potom střední hodnota této náhodné veličiny $\mathbb{E}[T] = \Theta(n) \cdot \mathbb{E}[\textit{počet průchodů cyklem}]$.

Teď si dokažme, že budeme-li náhodně vybírat pivota tak dlouho, až se strefíme do lžimediánu, tak „v průměru“ budeme muset tahat jen $1/p$-krát.

Lemma: *(O džbánu a vodě)* Čekání na náhodnou událost, která nastává s pravděpodobností p, trvá V průměru $1/p$.

Důkaz: Označme N počet pokusů. Potom střední hodnota počtu pokusů je:

$$\begin{aligned}
\mathbb{E}[N] &= p \cdot 1 + (1-p) \cdot (1 + \mathbb{E}[N]) \\
\mathbb{E}[N] &= p + 1 + \mathbb{E}[N] - p - p \cdot \mathbb{E}[N] \\
\mathbb{E}[N] \cdot (1 - 1 + p) &= 1 \\
\mathbb{E}[N] &= 1/p
\end{aligned}$$

S pravděpodobností p událost nastane a s odvrácenou pravděpodobností $(1-p)$ jsme jeden pokus promarnili a musíme celý proces opakovat. Jednoduchými úpravami nám vyjde, že střední hodnota počtu pokusů je $1/p$.

„V průměru se tedy chodí $1/p$-krát se džbánem pro vodu, než se ucho utrhne…“ ♡

Z lemmatu tedy plyne, že V našem případě $\mathbb{E}[\textit{počet průchodů cyklem}] \leq 2$. V průměru tedy na druhý pokus vytáhneme lžimedián. Ten pak použijeme jako pivot. Tímto tudíž dosáhneme průměrné časové složitosti $\Theta(n)$.

Věta: Pravděpodobnostním algoritmem lze najít k-tý nejmenší prvek z n prvků V průměrném čase $\Theta(n)$.

Poznámka: Když budeme volit pivota náhodně a nebudeme se starat o to, zda se jedná o lžimedián, či ne, tak dostaneme také průměrný čas $\Theta(n)$.

K volbě lžimediánu jsme volili vždy náhodného pivota, jednalo se tedy o randomizovaný algoritmus. Stejný výsledek nám ale vyjde i při deterministickém algoritmu, kdy budeme volit pivota vždy stejně (např. na první pozici ve vstupní posloupnosti), ale budeme mít nějak zaručeno, že vstupy budou dostatečně náhodné. U randomizovaného algoritmu se jednalo o průměr přes náhodná čísla, u deterministického algoritmu o průměr přes náhodné vstupy.

Už tedy víme, že když budeme volit pivota hodně špatně, tak se můžeme dostat až na časovou složitost $\Theta(n^2)$. Když budeme volit o něco lépe, tak se můžeme dosáhnout průměrné časové složitosti $\Theta(n)$. Existuje ale i algoritmus, který pracuje vždy (nejen V průměrném případě) V čase $\Theta(n)$. Podívejme se, jak bude vypadat …

Volba pivota:

1. Rozdělíme vstup na pětice … $\Theta(n)$
2. Spočteme medián každé pětice … $\Theta(n)$
3. Spočteme medián mediánů pětic tak, že rekurzivně zavoláme tentýž algoritmus $Select(n/10, \text{mediány pětic})$. Tím získáme pivota.

Pro úplnost se podívejme, jak bude vypadat celý algoritmus na hledání k-tého nejmenšího prvku posloupnosti za použití naší nové volby pivota:

Deterministický lineární algoritmus na výběr mediánu lineárně

1. *Vstup:* $X = x_1 \ldots x_n$, k $(1 \leq k \leq n)$
2. Pokud $n \leq 5 \Rightarrow$ vyřešíme hrubou silou.
3. Vstup rozdělíme na pětice $P_1 \ldots P_{\lceil n/5 \rceil}$.