EXTERNÍ TŘÍDĚNÍ

Třídění na páskách (50. a 60. léta):

sekvenční přístup k datům

algoritmy na principu Mergesortu (většinou)

Třídění na discích:

přímý přístup k blokům dat

Míra časové složitosti:

vstupní a výstupní operace (čtení a zápis dat)

operace v interní paměti jsou zanedbatelné

TŘÍDĚNÍ NA PÁSKÁCH - MERGESORT

Přímý MERGESORT - **dvoufázové mergování**

používá 3 pásky T1, T2, T3

nenáročný na vnitřní paměť (stačí 2 prvky)

Příklad:

0) T1: 44, 55, 12, 42, 94, 18, 6, 67 vstup
   T2:
   T3:

1a) T1: 44, 55, 12, 42 vstup se rozdělí
   T2: 94, 18, 6, 67
   T3:

1b) T1:
   T2: merguji se dvojice
   T3: 44, 94 | 18, 55 | 6, 12 | 42, 67

2a) T1: 44, 94 | 18, 55
   T2: 6, 12 | 42, 67 rozdělí se
   T3:

2b): T1: merguji se čtveřice
   T2:
   T3: 6, 12, 44, 94 | 18, 42, 55, 67

3a) T1: 6, 12, 44, 94

rozdělí se

T2: 18, 42, 55, 67

T3:

3b) T1:

T2:

T3: 6, 12, 18, 42, 44, 55, 67, 94

Výpočet má 3 stupně (obecně $\log n$)

V každém stupni 2 fáze: a) rozdělení (kopírování)

b) mergování

V každé fázi pro každý prvek 1 čtení

1 zápis

Složitost: $O(n \log n)$

Fáze a) je z hlediska třídění neproduktivní

(nepřeuspořádává prvky)

spotřebovává polovinu operací

Jednofázové mergování

používá 4 pásky

výsledek mergování se hned rozděluje na 2 pásky

vyvážené mergování - na každé pásce stejný počet běhů

Příklad:

0a) T1: 44, 55, 12, 42, 94, 18, 6, 67

T2:

T3:

T4:

0b): T1: 44, 55, 12, 42

T2: 94, 18, 6, 67

T3:

T4:

rozdělení

1) T1:

T2:

T3: 44, 94 | 6, 12

T4: 18, 55 | 42, 67

mergování + rozdělení

2) T1: 18, 44, 55, 94

T2: 6, 12, 42, 67

T3:

T4:

3) T1:

T2:

T3: 6, 12, 18, 42, 44, 55, 67, 94

T4:

Přirozený MERGESORT

pro soubory, které obsahují delší uspořádané úseky (běhy)

merguji se dvojice běhů

rozdělování se provádí podle počtu běhů, ne podle počtu prvků

Příklad:

0a) T1: 44, 55, 12, 42, 94, 18, 6, 67

T2:

T3:

T4:

0b) T1: 44, 55 | 18

T2: 12, 42, 94 | 6, 67

T3:

T4:

rozdělení

1) T1:

mergování + rozdělení

T2:

T3: 12, 42, 44, 55, 94

T4: 6, 18, 67

2) T1: 6, 12, 18, 42, 44, 55, 67, 94

T2:

T3:

T4:

Složitost: $O(n \log r)$

kde r je počet běhů ve vstupu

Mergesort s využitím interní paměti:

m = kapacita interní paměti

1) vytvoření běhů délky m:

ze vstupní pásky se načte m prvků, setřídí se, vypíšou se na pásku T1

načte se dalších m prvků, setřídí se, vypíšou na T2

— " — — " — — " — T1

2) mergování běhů

Složitost: krok 1) $O(n)$

krok 2) - $\log \frac{n}{m}$ průchodů

celkem $O(n \log \frac{n}{m})$

Technické vylepšení:

když se dá páska číst oběma směry, střídá se vzestupné a sestupné mergování

ušetří se převíjení pásek na začátek po každém průchodu

Příklad:

mám T1: 44, 94 | 6, 12

T2: 18, 55 | 42, 67

T3:

T4:

jsem na konci pásek T1, T2

jdu odzadu

merguju sestupně

na T3 a T4 zapisuji odpředu

T1:

T2:

T3: 67, 42, 12, 6

T4: 94, 55, 44, 18

jsem na konci T3, T4

jdu odzadu

merguji vzestupně

na T1 a T2 zapisuji odpředu

T1: 6, 12, 18, 42, 44, 55, 67, 94

T2:

T3:

T4:

Poznámka – může se stát, že výsledek bude setříděný obráceně

(pak se ještě jednou překopíruje)

Zobecnění - <u>vícecestné mergování</u>:

máme $2N$ pásek, $r = \frac{n}{m}$ běhů

běhy se rozdělí na N pásek

merguje se současně N běhů ze vstupních pásek

zmergované běhy se rozdělují na N výstupních pásek

<u>vyvážené</u> mergování - na každé pásce stejný počet běhů

<u>Složitost</u>: $O(n \log_N (\frac{n}{m}))$

nevýhoda - velké množství pásek

<u>Vícefázové mergování</u>

máme N vstupních pásek, 1 výstupní

běhy se na pásky rozdělí <u>nevyváženě</u>

<u>Příklad</u>:

1) T1: 13 běhů

T2: 8 běhů   zmergují se T1, T2 na T3

T3: 0

2) T1: 5 běhů
T2: 0 — zmergují se T1, T3 na T2
T3: 8 běhů

3) T1: 0
T2: 5 běhů — zmergují se T2, T3 na T1
T3: 3 běhy

4) T1: 3 běhy
T2: 2 běhy — atd.
T3: 0

5) T1: 1 běh
T2: 0
T3: 2 běhy

6) T1: 0
T2: 1 běh
T3: 1 běh

7) T1: 1 běh
T2: 0 — konec
T3: 0

Příklad:

T1	T2	T3	T4	T5	T6	pásky
16	15	14	12	8	0	počty běhů
8	7	6	4	0	8	
4	3	2	0	4	4	
2	1	0	2	2	2	
1	0	1	1	1	1	
0	1	0	0	0	0	konec

Podstatné je počáteční rozdělení běhů na pásky

1. příklad – jsou to dvě po sobě jdoucí Fibonacciho čísla

obecně – Fibonacciho čísla odpovídajícího řádu:

$$F_{i+1} = F_i + F_{i-1} + \dots + F_{i-k}$$

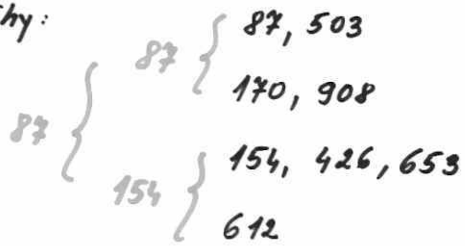
Když se počáteční počet běhů nedá takto rozdělit, doplní se hypotetickými prázdnými běhy.

Současné mergování více běhů

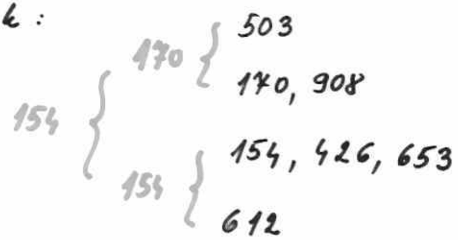
výběr minima probíhá v interní paměti:
použije turnajový strom nebo halda

Turnajový strom:

mám běhy:

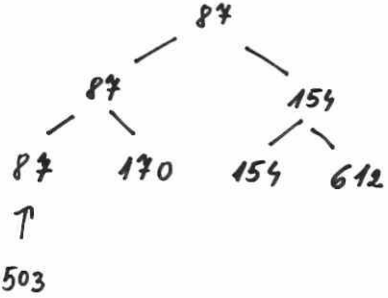


další krok:



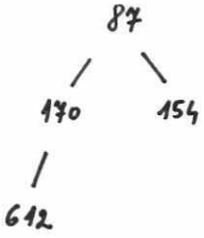
atd.

neboli:



atd.

Min-halda :



provedu DELETEMIN (87)
INSERT (503)
DELETEMIN
atd.

Počáteční vytvoření běhů

1) načíst do interní paměti maximální množství prvků
   interně je setřídit
   vznikají běhy stejné délky m

2) metoda Replacement selection
   mohou vznikat běhy delší než m, v průměru 2m
   princip: jakmile zapíšu první prvek z interní paměti na pásku, uvolní se tím místo a může se hned načíst další

Postup:

načtu m prvků do paměti, vytvořím min-haldu

provedu DELETEMIN

minimum zapíšu na výstupní pásku

načtu další prvek ze vstupní pásky -

- je-li větší než odebrané minimum, přijde do téže haldy (a téhož běhu)
- je-li menší, dám ho stranou

provedu další DELETEMIN

atd.

až vyprázdním celou haldu, postavím novou z prvků, které jsem odložila (můžu ji stavět průběžně)

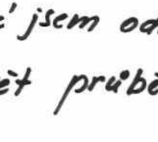
Příklad:

třídím 11, 94, 81, 96, 12, 35, 17, 99, 28, 58, 41, 75

$m = 3$

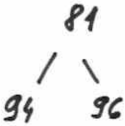
1) načtu 11, 94, 81

vytvořím haldu



2) vypíšu 11, načtu 96

vytvořím haldu



3) vypíšu 81, načtu 12

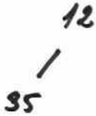
halda:

12 tam nepatří

4) vypíšu 94, načtu 35

1. halda: 96

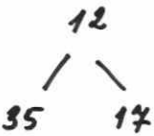
2. halda:



5) vypíšu 96, načtu 17

hotový běh: 11, 81, 94, 96

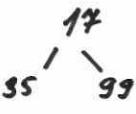
nová halda:



6) vypíšu 12 – začátek nového běhu

načtu 99

halda:



atd.

výsledek – běhy: 11, 81, 94, 96

12, 17, 28, 35, 41, 58, 99

75

PŘIHRÁDKOVÉ TŘÍDĚNÍ NA PÁSKÁCH

Příklad:

třídím 0, 1, 2, 3, 7, 6, 5, 4

tj. v binárním zápise:

000, 001, 010, 011, 111, 110, 101, 100

použijí se 4 pásky – 2 pro vstup
2 pro výstup

0) T1: 000, 001, 010, 011, 111, 110, 101, 100 vstup
T2:
T3:
T4:

1) T1:
T2: rozdělení podle poslední cifry
T3: 000, 010, 110, 100
T4: 001, 011, 111, 101

2) T1: 000, 100, 001, 101
T2: 010, 110, 011, 111
T3:
T4: rozdělení podle prostřední cifry v pořadí T3, T4

3) T1:

rozdělení podle 1. cifry

T2:

T3: 000, 001, 010, 011

T4: 100, 101, 110, 111

4) T1: 000, 001, 010, 011, 100, 101, 110, 111

T2:

spojení T3, T4

T3:

T4:

Složitost: k průchodů pro čísla v rozsahu

$0, \dots, 2^k - 1$

V každém průchodu se každé číslo 1x kopíruje

celkem: $O(k \cdot n)$

QUICKSORT NA 2 PÁSKÁCH

mám pásky 0 a 1 - fungují jako zásobníky
jsou obousměrné

vstupní soubor je na pásce 0

(vejde-li se do vnitřní paměti, setřídí se interně a vrátí na pásku)

pivot = poslední prvek souboru = k

páska 0 se čte odzadu -
prvky > k se kopírují na pásku 1

páska 0 se čte odpředu -
prvky = k se kopírují na pásku 1

páska 0 se čte odzadu -
prvky < k se kopírují na pásku 1

výsledek: T1: prvky > k, prvky = k, prvky < 1
T0: prázdná

rekurze:
zpracují se prvky < k a setříděné se zapíší na T0
prvky = k se překopírují na T0
zpracují se prvky > k a setříděné se zapíší na T0

Poznámka: Při rozdělování se střídají pásky, ale taky nerovnosti –

tj. když rozdělují soubor (část) z T1 na T0, zapisuji ho na T0 v pořadí:

prvky < k, prvky = k, prvky > k

Složitost: nejhorší $O(n^2)$

očekávaná $O(n \log n)$

Příklad:

T0: 11, 94, 81, 96, 12, 35, 17, 99, 28, 58, 41, 75

T1:

rozdělím podle 75

T0:

T1: $\underbrace{99, 96, 81, 94}_{>75}$, 75, $\underbrace{41, 58, 28, 17, 35, 12, 11}_{<75}$

zpracuji prvky < 75, rozdělím podle 11

T0: 11, $\overbrace{12, 35, 17, 28, 58, 41}^{>11}$

T1: $\underbrace{99, 96, 81, 94}$, 75

zpracuji prvky > 11, rozdělím podle 41

T0: 11

T1: $\underbrace{99, 96, 81, 94}, 75, \underbrace{58}_{>41}, 41, \underbrace{28, 17, 35, 12}_{<41}$

prvky < 41 rozdělím podle 12

T0: $11, 12, \overbrace{35, 17, 28}^{>12}$

T1: $\underbrace{99, 96, 81, 94}, 75, \underbrace{58}, 41$

prvky > 12 rozdělím podle 28

T0: 11, 12

T1: $\underbrace{99, 96, 81}, 94, 75, \underbrace{58}, 41, \underbrace{35}_{>28}, 28, \underbrace{17}_{<28}$

zkopíruji 12 na T1

zpracuji prvky < 12 (žádné nejsou)

zkopíruji 11 na T1

zpracuji prvky < 11 (žádné nejsou)

T0:

T1: $\underbrace{99, 96, 81, 94}_{>75}, 75, \underbrace{58, 41, 35, 28, 17, 12, 11}_{<75}$

kopíruji na T0: prvky < 11

= 11

< 12

= 12

< 28

= 28

< 41

= 41

< 75

= 75

T0: 11, 12, 17, 28, 35, 41, 58, 75

T1: 99, 96, 81, 94

dotřídím prvky > 75, rozdělím podle 94

T0: $11, \dots, 75, \underbrace{81}_{<94}, 94, \underbrace{96, 99}_{>94}$

T1:

zpracuji prvky > 94, rozdělím podle 99

T0: 11, ..., 75, 81, 94

T1: $99, \underbrace{96}_{<99}$

zkopíruji 94 na T1

zpracuji prvky < 94 (tj. 81)

T0: 11, ..., 75

T1: 99, 96, 94, 81

kopíruji na T0: prvky < 94

= 94

< 99

= 99

> 99 (žádné nejsou)

<u>výsledek:</u>

T0: 11, 12, 17, 28, 35, 41, 58, 75, 81, 94, 96, 99

T1:

TŘÍDĚNÍ NA DISCÍCH

na páskách: prvky se přenášejí po jednom

na disku: — " — po blocích

<u>značení</u>: N počet prvků v souboru

M počet bloků, které se vejdou do interní paměti

B počet prvků v jednom bloku

P počet bloků, které se dají přenášet současně (pro nás $P=1$)

platí: $1 \leq B \leq M < N$, $1 \leq P \leq \lfloor \frac{M}{B} \rfloor$

<u>složitost</u> (počet I/O operací):

$$\Omega\left(\frac{N}{PB} \cdot \frac{\log(1+\frac{N}{B})}{\log(1+\frac{M}{B})}\right)$$

v nejhorším i průměrném případě

<u>Aggarwal</u>, <u>Vitter</u> (1988)

MERGESORT

$\frac{N}{B}$ bloků se postupně načte do interní paměti

(vejde se jich $\frac{M}{B}$)

obsah interní paměti se setřídí

vytvoří se $\frac{N}{M}$ běhů

běhy se po blocích uloží zpět na disk

složitost: $\frac{N}{B}$ čtení, $\frac{N}{B}$ zápisů

další fáze: vždy $\frac{M}{B}$ běhů se merguje do jednoho

(z každého běhu se načte 1 blok, když se při mergování vyčerpá, načte se další blok z téhož běhu)

počet fází: $\log_{\frac{M}{B}} \frac{N}{B}$

v každé se přečte a zapíše $\frac{N}{B}$ bloků

celkem: $2\frac{N}{B} + 2\frac{N}{B}\log_{\frac{M}{B}}\frac{N}{B} = O\left(\frac{N}{B}\cdot\frac{\log\frac{N}{B}}{\log\frac{M}{B}}\right)$

vylepšení použitím Replacement selection

nevýhoda - netřídí na místě

EXQUISIT (Six, Wegner 1984)

externí varianta Quicksortu

stačí $M = 2$

Algoritmus:

1) načte se první a poslední blok vstupního souboru F

   z jejich prvků se vypočte pivot H

2) F se rozdělí na část F_1 (prvky menší než H) a F_2 (prvky větší než H)

   pointer L jde od 1 do B v levém bloku

   -"- R -"- B do 1 v pravém -"-

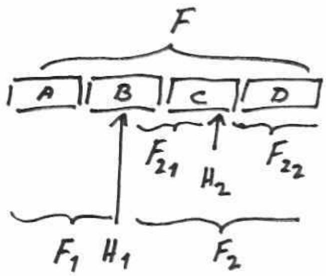
   prvky se porovnávají s H a případně se prohodí

   když pointer dosáhne hranice bloku, blok se zapíše na disk a načte se sousední

3) když se pointery setkají, třídí se rekurzivně F_1 a F_2

Příklad:

disk:



interní paměť:

$\boxed{A}\boxed{D}$ začátek

$\boxed{B}\boxed{D}$ L přešel hranici mezi A a B

$\boxed{B}\boxed{C}$ R —"— D a C

$\boxed{B}\boxed{A}$ L a R se setkaly v B

dotřiďuje se F_1 (dokončí se A)

$\boxed{B}\boxed{D}$ začíná se třídit F_2

$\boxed{C}\boxed{D}$ L přešel z B do C

$\boxed{C}\boxed{B}$ L a R se setkaly v C

dotřiďuje se F_{21} (dokončí se B)

$\boxed{C}\boxed{D}$ dotřiďuje se F_{22} (dokončí se C, D)

Složitost

průměrný případ - řeší se rekurze

$$T(N) = \left\lfloor \frac{N}{B} \right\rfloor + 1 + \frac{1}{N-1} \sum_{k=1}^{N-1} T(k) + T(N-k) \;, \quad N > B+1$$

$$T(N) = 0 \text{ pro } N \leq B+1$$

řešení:

$$T(N) \leq 2 \frac{N}{B} \log\left(\frac{N}{B+1}\right) + \frac{3N}{B} + 1 - \frac{N}{B(B+1)}$$

nejhorší případ:

není $O\left(\left(\frac{N}{B}\right)^2\right)$, ale $O\left(\frac{N^2}{B}\right)$

tj. když se při každém rozdělení oddělí 1 prvek

volba pivota

je ještě významnější než v interním Quicksortu

možností je méně (počítá se jen ze 2 bloků)

doporučení: $H = \left\lfloor \frac{F(L) + F(R)}{2} \right\rfloor$ fiktivní pivot

výhoda - není třeba ho po rozdělení umístit na správnou pozici

Poznámky:

1) Blok se neukládá na disk, dokud není nutné uvolnit místo v interní paměti

2) Blok se nepřepisuje (na disku), pokud v něm neproběhla žádná výměna

(tj. např. u setříděného souboru není žádný zápis na disk, pouze čtení)

3) Pořadí volání rekurze – třídí se dřív levá část

4) Rekurze se zastaví, když se příslušná část souboru vejde do interní paměti

5) Je-li $M > 2$, může se pracovat s logickými bloky místo fyzických

6) Jiný způsob rozdělování v interní paměti – pomocí dvoukoncové haldy

(operace MIN a MAX v konstantním čase, ostatní v logaritmickém)

HEAPSORT

Wiedermann (1983)

Wiedermann, Šíma, Neruda (1997)

používá se <u>k-regulární halda bloků</u>

a) každý uzel v obsahuje M záznamů

(logický blok S_v velikosti M)

b) $k \approx \frac{M}{B}$ (tj. počet fyzických bloků, které se vejdou do interní paměti)

c) haldová podmínka:

když v je synem u v haldě T, pak pro všechna $x \in S_v$, $y \in S_u$ platí $x \leq y$

záznamy v uzlech jsou vzestupně setříděné

Dá se reprezentovat polem.

Budování haldy

soubor se rozdělí na logické bloky velikosti M

každý se načte do interní paměti

tam se setřídí

zapíše zpátky na disk

<u>úprava haldy</u> - podle Floyda (shora dolů):

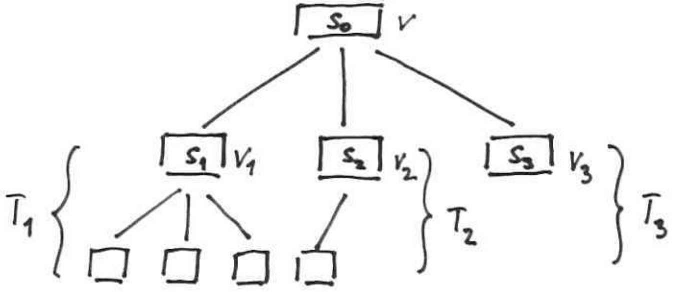
q = počet vrcholů v haldě

$i := \frac{q-1}{k}$

while $i \geq 1$ **do** restore(i, k); $i := i-1$ **enddo**

restore(i,k) – obnoví haldovou podmínku v haldě s kořenem i

<u>Příklad:</u> máme T



T_1, T_2, T_3 už jsou správně vytvořené 3-regulární haldy

Není-li splněna haldová podmínka pro kořen v, pak:

1) pro $i = 0, 1, \dots, k$ označím $s_i = \min\{S_i\}$
   posloupnost $s_0, s_1, \dots, s_k$ setřídím vzestupně
   tj. najdu permutaci π, že $s_{\pi(0)} \le s_{\pi(1)} \le \dots \le s_{\pi(k)}$
   (tomu odpovídá pořadí hald $T_{\pi(0)}, T_{\pi(1)}, \dots, T_{\pi(k)}$)
2) vytvořím $U = S_0 \cup S_1 \cup \dots \cup S_k$
   setřídím U vzestupně pomocí $(k+1)$-cestného mergování – výsledek bude Σ
3) Σ rozdělím na úseky $\sigma_0, \sigma_1, \dots, \sigma_k$ délky M

4) protože $s_{\pi(i)} \le t_i = \min\{\sigma_i\}$ pro všechna i, můžu přiřadit: σ_i kořeni $T_{\pi(i)}$ pro $i = 1, \dots, k$

σ_0 kořeni T

haldová podmínka bude splněna v $T_1, \dots, T_k$

5) není-li haldová podmínka splněna v kořeni T, vyměním σ_0 a σ_k (je v kořeni $T_{\pi(k)}$) rekurzivně obnovím haldovou podmínku v $T_{\pi(k)}$

Složitost:

při vhodné implementaci $O\left(\frac{N}{B}\right)$ pro každé $k \geq 2$

Heapsort

a) vybudovat haldu

b) opakovat DELETEMAX

Složitost Heapsortu:

$$O\left(k \cdot \frac{N}{B} \cdot \frac{\log \frac{N}{B}}{\log k}\right)$$ pro každé $k \geq 2$

Pro $k > 2$ se fáze b) provede jinak a tím zmizí multiplikativní konstanta k.

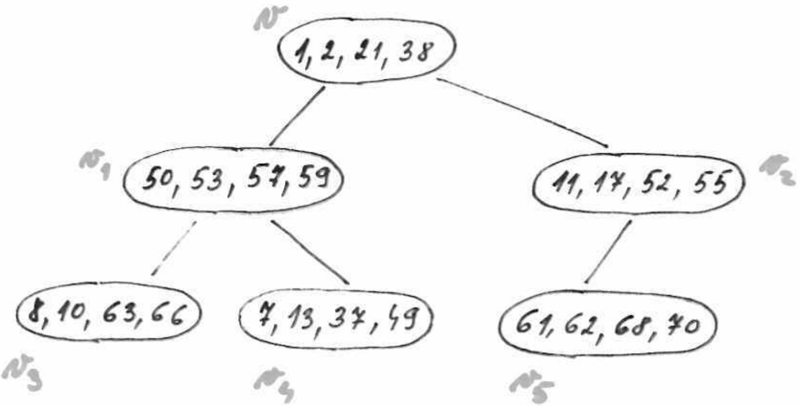
Příklad: třídím posloupnost

21, 1, 2, 38, 53, 50, 57, 59, 55, 52, 11, 17, 8, 10, 63, 66, 7, 37, 13, 49, 61, 62, 70, 68

$N = 24$

nechť $B = 1$, $M = 4$, použiji binární haldu

1) setřídím úseky délky 4
2) naházím je do haldy



3) upravím haldu

a) prohodím uzly v_2 a v_5

b) uzel v_1 nelze prohodit s žádným z jeho synů

najdu $s_1 = \min\{v_1\} = 50$

$s_2 = \min\{v_3\} = 8$

$s_3 = \min\{v_4\} = 7$

platí $s_3 < s_2 < s_1$

vytvořím $U = \{v_1\} \cup \{v_3\} \cup \{v_4\}$

setřídím U pomocí 3-cestného mergování:

$$\underbrace{7, 8, 10, 13}_{\sigma_0},\ \underbrace{37, 49, 50, 53}_{\sigma_1},\ \underbrace{57, 59, 63, 66}_{\sigma_2}$$

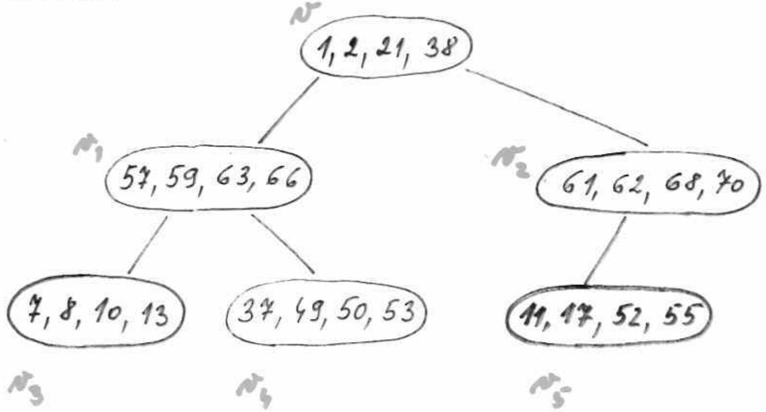
σ_0 přijde do v_1

σ_1 —"— v_4

σ_2 —"— v_3

prohodím uzly v_1 a v_3

dostanu:



c) v nelze prohodit s v_1 ani s v_3

najdu $s_1 = \min\{v\} = 1$

$s_2 = \min\{v_1\} = 57$

$s_3 = \min\{v_2\} = 61$

platí $s_1 < s_2 < s_3$

vytvořím $U = \{v\} \cup \{v_1\} \cup \{v_2\}$, setřídím:

$$\underbrace{1, 2, 21, 38}_{\sigma_0}, \underbrace{57, 59, 61, 62}_{\sigma_1}, \underbrace{63, 66, 68, 70}_{\sigma_2}$$

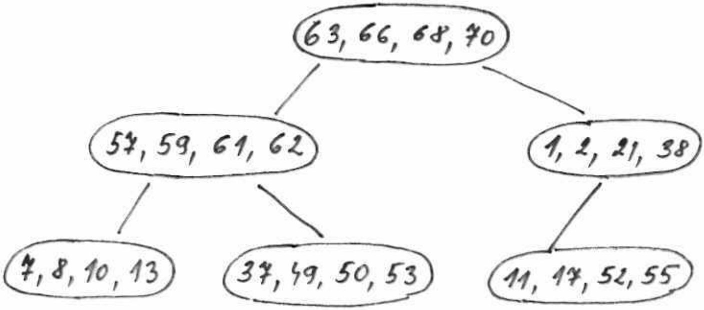
σ_0 přijde do v

σ_1 —"— v_1

σ_2 —"— v_2

prohodím v a v_2

dostanu:



v_2

v_5

d) upraví se halda s kořenem v_2

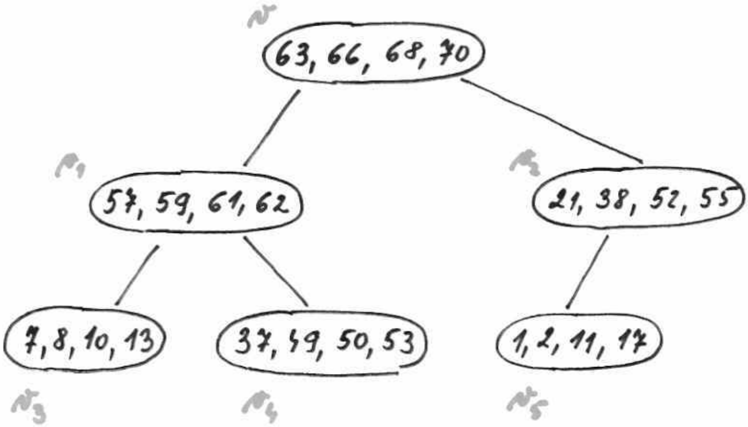
setřídím $U = \{v_2\} \cup \{v_5\}$

rozdělím na 2 části

menší prvky přijdou do v_2, větší do v_5

prohodí se v_2 a v_5

Výsledná halda:



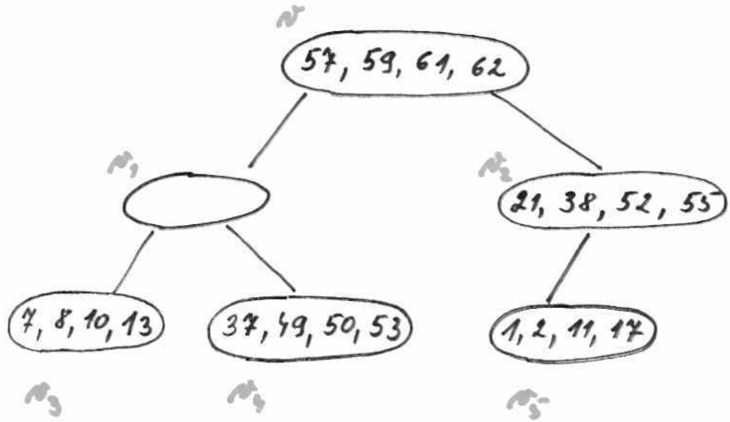
Třídění:

a) odeberu v_1, nahradím ho v_5, upravím haldu atd.

hodí se pro binární haldy, ne pro více regulární

b) odeberu N

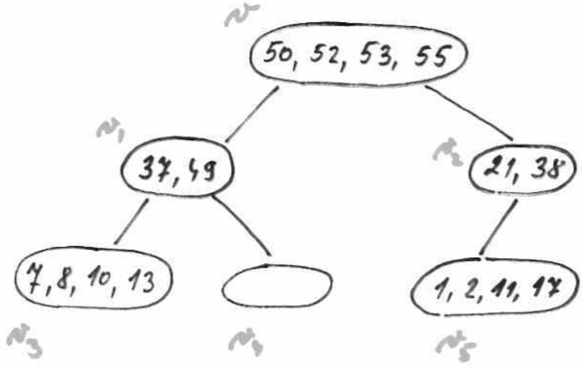
do kořene namerguji 4 největší prvky z N_1 a N_2



do N_1 namerguji 4 největší prvky z N_3 a N_4 (tj. N_4)

odeberu kořen

namerguji do něj 4 největší prvky z N_1 a N_2



doplním uzly v_1 a v_2

odeberu kořen

atd.

<u>Pozor</u>! Když se vyprázdní nějaký uzel a chci ho doplnit z jeho synů, musím nejdřív vyčistit interní paměť (mohlo v ní něco zůstat z předchozího mergování - to se musí vrátit na disk).

Prostorová složitost

Algoritmus se dá implementovat tak, aby třídil na místě, ale je to technicky složité.

Poznámky:

1) Existují externí verze i dalších interních třídicích algoritmů

2) Prvky externího třídění se uplatňují i v interních algoritmech, pokud vnitřní paměť je hierarchická!

Míra složitosti : počet operací (porovnání, výměny, ...)
+ počet výpadků z cache