MERGESORT

Goldstine, von Neumann (1948)

algoritmus je ale starší, používal se již na
mechanických sorterech

v minulosti se nepoužíval pro interní třídění -
prostorová složitost je $O(n)$
(potřebuje pomocné pole)

vhodný pro externí třídění sekvenčních souborů
(pásky)

verze: přímý Mergesort
přirozený Mergesort - pro skoro setříděné
soubory (dlouhé běhy)

Přímý Mergesort

```
procedure Mergesort (A, B, l, r)
 if l<r then
    c := (l+r) div 2
    Mergesort (A, B, l, c)
    Mergesort (A, B, c+1, r)
    for i:=l to r do B(i) := A(i) enddo
    Merge (A, B, l, c, r)
endif
```

A... tříděné pole

B... pomocné pole

předosazení se dá vynechat, když se jejich role budou střídat

```
procedure Merge (A, B, l, c, r)
  i := l, j := c+1, k := l
  repeat
    if i > c then
        A(k) := B(j), j := j+1
    else if j > r then
        A(k) := B(i), i := i+1
    else if B(i) > B(j) then
        A(k) := B(j), j := j+1
    else
        A(k) := B(i), i := i+1
    endif
    k := k+1
  until k > r
```

Složitost:

Merge .... $O(n)$

Mergesort $(A, B, 1, n)$ ... $T(n) = 2T(\frac{n}{2}) + O(n)$

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

Přirozený Mergesort

běh (run) $a_i, a_{i+1}, \dots, a_j$

platí $a_{i-1} > a_i$, $a_i \le a_{i+1} \le \dots \le a_j$, $a_j > a_{j+1}$

algoritmus:

1) projde posloupnost a rozdělí ji na běhy
2) merguje běhy

problém: běhy mohou mít nestejnou délku

řešení:

a) použít klasické lineární mergování
   najít optimální pořadí běhů pro mergování

b) zachovat pořadí běhů
   použít jiný způsob mergování

Pořadí mergování nestejně dlouhých běhů

Slučovací strom: binární strom

listy jsou běhy
vnitřní vrcholy vznikají mergováním svých synů
kořen je výsledná setříděná posloupnost

používá se klasické lineární mergování

cena stromu $C(T) = \sum_i d_i w_i$

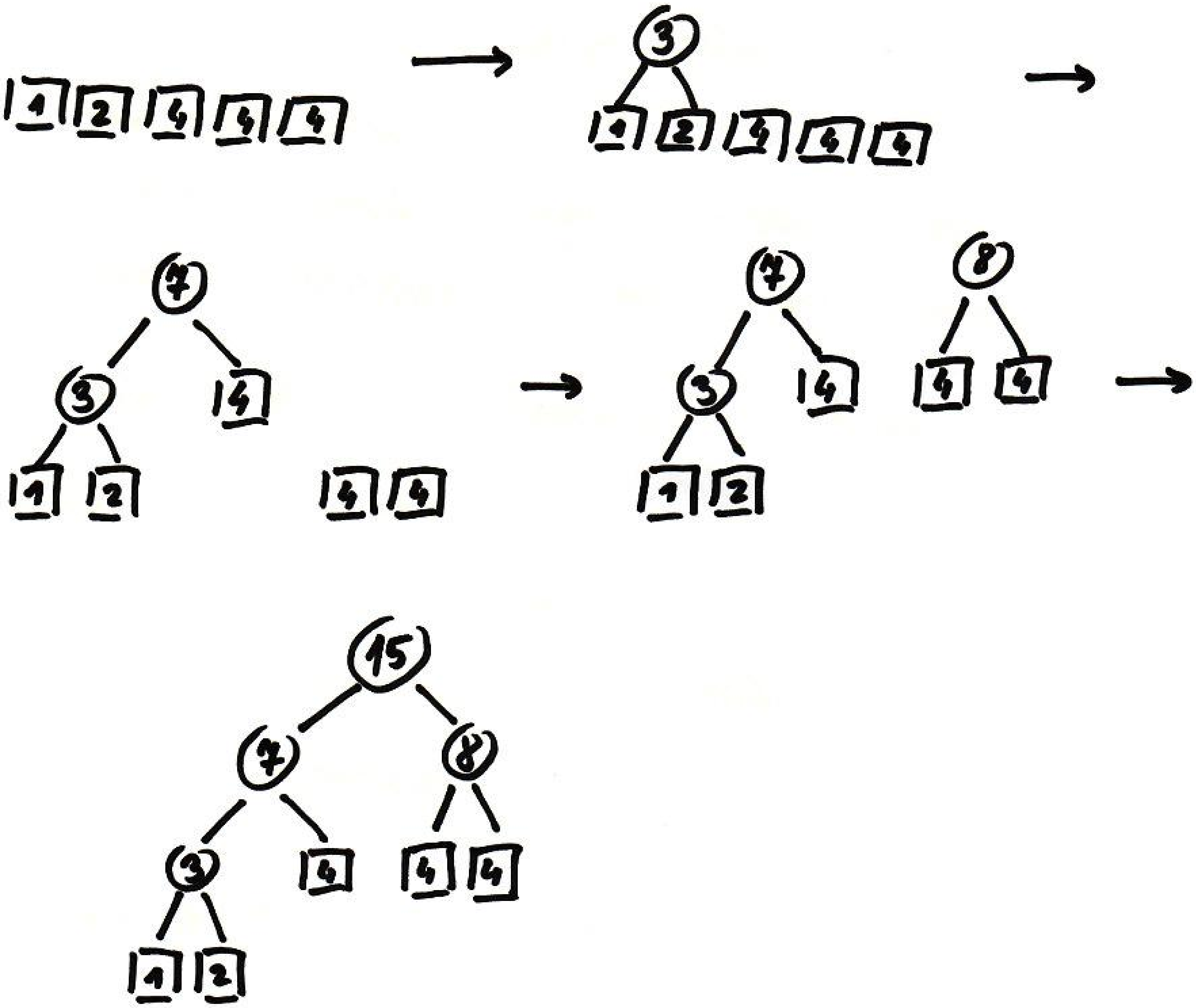
w_i ... délka běhu A_i (listu)
d_i ... hloubka — " —

Hledá se optimální strom - s minimální cenou

(tj: minimální celkový počet operací při třídění)

mergování běhů délek x, y: $x+y-1$ porovnání
$x+y$ přiřazení

Příklad: běhy délek 1, 2, 4, 4, 4



cena: $3 \cdot 1 + 3 \cdot 2 + 2 \cdot 4 + 2 \cdot 4 + 2 \cdot 4 = 33$

Algoritmus

$V = \{v_1, \dots, v_m\}$ listy

$0 \le w_1 \le w_2 \le \dots \le w_m$ délky

I = množina vnitřních vrcholů (na začátku $I = \emptyset$)

k = počet —"— (—"— $k = 0$)

while $k < m-1$ **do**

vyber $x_1, x_2 \in I \cup V$ s dvěma nejmenšími hodnotami w

vytvoř nový vrchol x s hodnotou $w_x = w_{x_1} + w_{x_2}$

přidej x do I

$k := k+1$

odstraň x_1, x_2 z $I \cup V$

enddo

reprezentace: I, V fronty

(vrcholy v nich jsou setříděné podle w)

složitost: $O(m)$

Třídění

1) rozdělení na běhy, zjištění jejich délek

čas: $O(n)$

2) setřídění délek

čas: $O(m \log m)$

3) konstrukce slučovacího stromu

čas: $O(m)$

4) mergování

čas: $C(T) = \sum_i d_i w_i$

Binární mergování

1) nejjednodušší případ : $A = \{a_1 \leq a_2 \leq \dots < a_n\}$

$$B = \{b\}$$

tj. Insert b do uspořádaného pole A

logn porovnání

n dosazení do výsledného pole

2) $A = \{a_1 \leq a_2 \leq \dots \leq a_n\}$

$B = \{b_1 \leq b_2 \leq \dots \leq b_m\}$

$m < n$ (o hodně)

postupně : Insert b_1 do $\{a_1, \dots, a_n\}$

Insert b_2 do $\{b_1, \dots, a_n\}$

$\vdots$

Insert b_m do $\{b_{m-1}, \dots, a_n\}$

mlogn porovnání

n+m dosazení do výsledného pole

Hwang, Lin (1972)

rekurzivně:

1) $m = 0$ ... konec

2) $m \geq 1$, $m < n$

položíme $t = \lfloor \log \frac{n}{m} \rfloor$, porovnáme b_1 s a_{2^t}

a) $b_1 < a_{2^t}$

binárně se vyhledá pozice b_1 v $\{a_1, \dots, a_{2^t - 1}\}$

(t porovnání, výsledkem je pozice k)

překopírují se $a_1, \dots, a_{k-1}, b_1$ do výsledného pole,

odstraní se z původních posloupností

merguje se $\{a_k, \dots, a_n\}$ s $\{b_2, \dots, b_m\}$

b) $b_1 > a_{2^t}$

překopírují se $a_1, \dots, a_{2^t}$ do výsledného pole

mergují se $\{a_{2^t+1}, \dots, a_n\}$, $\{b_1, \dots, b_m\}$

přepočítáme $t' = \lfloor \log \frac{n'}{m'} \rfloor$ pro zredukované

posloupnosti

(je-li $m' > n'$, vymění se role A a B)

počet porovnání: $O\left(\left\lceil \log \binom{n+m}{m} \right\rceil + \min(m,n)\right)$

<u>Teoretický dolní odhad</u>

pro všechny mergovací metody založené na porovnávání prvků

$$\log \binom{n+m}{m} = \Theta\left(m \log \frac{n}{m}\right)$$

$\binom{n+m}{m}$ = počet možností, jak zatřídit m-prvkovou množinu do n-prvkové

<u>Manacher (1979)</u>

statická varianta předchozí metody

$t = \lfloor \log \frac{n}{m} \rfloor$ pro všechny iterace

složitost je řádově stejná

Pro $m=n$ je složitost $O(n)$.

Pro posloupnosti stejných délek není nic lepšího než klasické lineární mergování!

Příklad: zatřiďujeme 4 prvky do n-prvkové množiny

1) dynamická varianta

nejpravděpodobnější pozice pro 1. prvek –

v 1. čtvrtině větší množiny

dále zatřiďujeme 3 prvky do n'-prvkové množiny

nejpravd. pozice pro 1. prvek –

v 1. třetině větší množiny

atd.

2) statická varianta

nejpravd. pozice pro 1. prvek – v 1. čtvrtině větší množiny

— " — 2. — " — 2. — " —

atd.

Analýza očekávané složitosti:

Fernandez de la Vega, Frieze, Santha (1998)

Reprezentace mergovaných posloupností

a) polem

umožňuje binární vyhledání pozice

nedá se okamžitě uložit

b) lineárním seznamem

umožňuje jednoduché vkládání

nedá se binárně vyhledávat

c) stromem

Brown, Tarjan (1979)

použili výškově vyvážené stromy (AVL, (2,3)-stromy)

mergování postupnými Inserty prvků menšího stromu do většího

složitost : $O(m \log n)$

to je počet všech operací včetně vyvažování

dá se snížit na $O(m \log \frac{n}{m})$

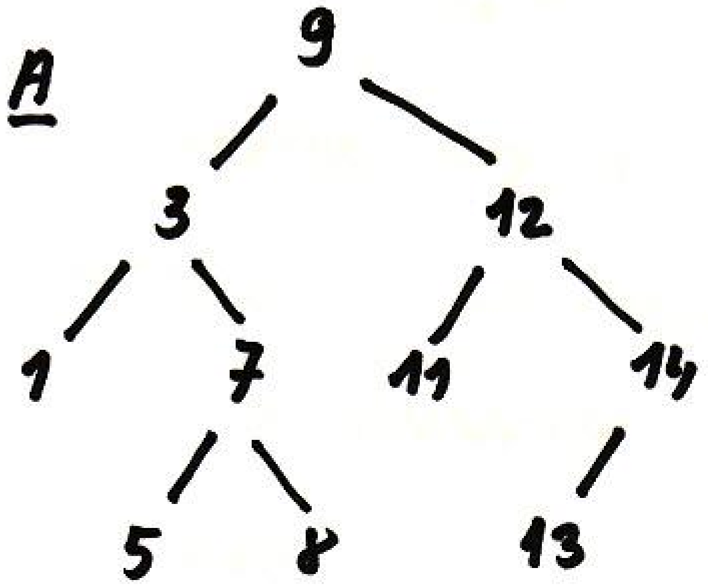
Formální algoritmy i výpočet složitosti jsou dost komplikované.

Příklad: $A = \{1, 3, 5, 7, 8, 9, 11, 12, 13, 14\}$

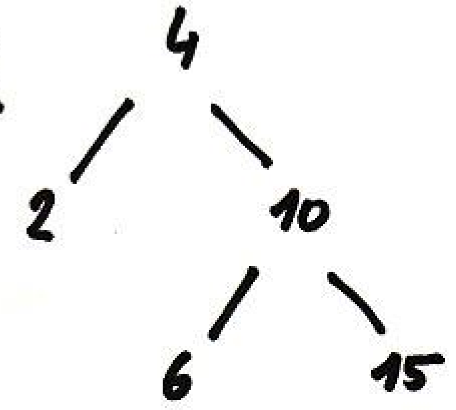
$B = \{2, 4, 6, 10, 15\}$

AVL-stromy

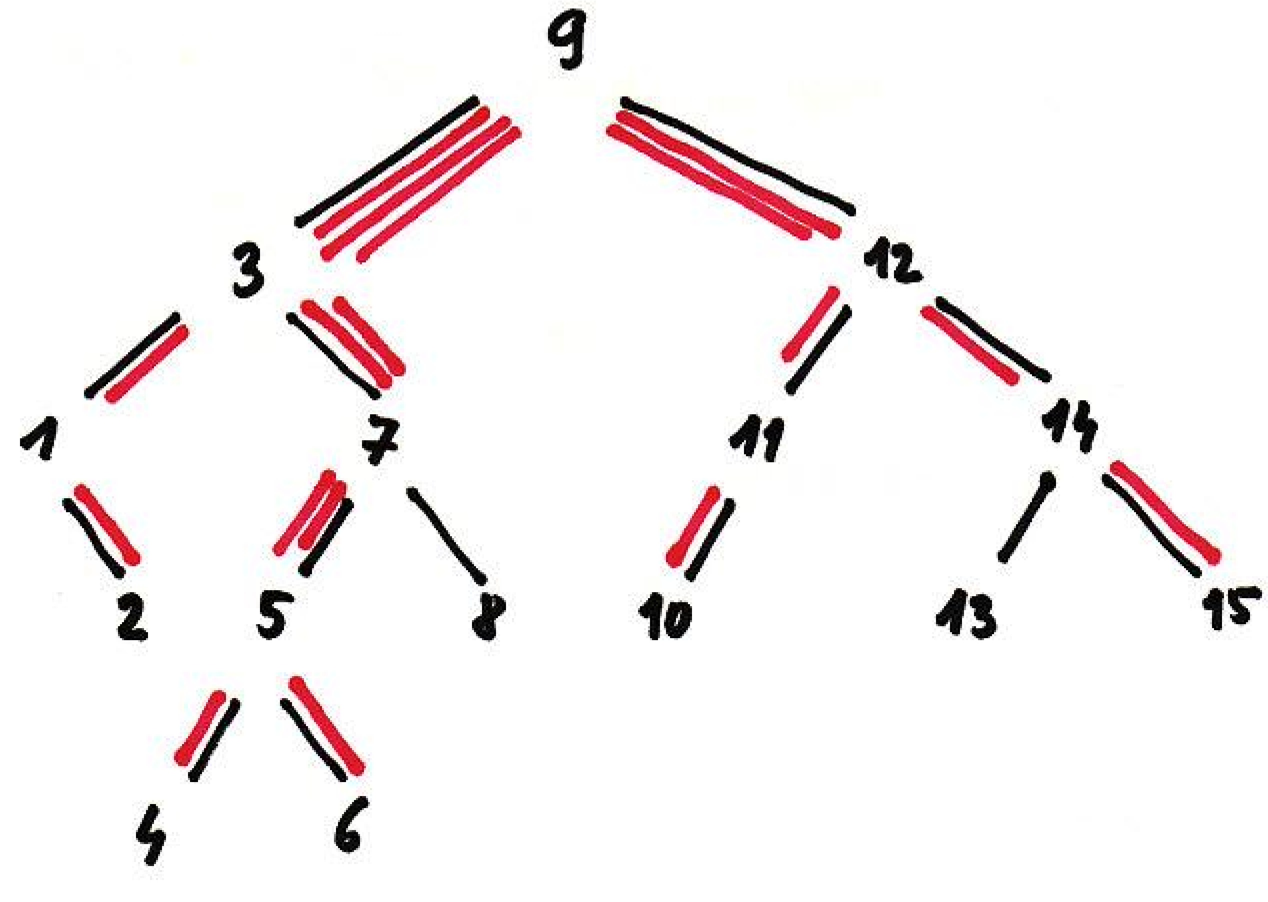
A



B



výsledek



Některé cesty se zbytečně procházejí opakovaně - dá se to zredukovat.

<u>Př:</u> po vložení 4 vím, že další vkládaný prvek bude větší, tím spíš větší než 3 porovnání s 3 můžu vynechat

Postup: cestou z kořene do 4 si pamatuji uzly s prvky > 4 (tj. 9, 7, 5)

při vkládání 6 je procházím zpátky, až narazím na první > 6 (tj: 7)

odtud začnu vkládat 6

Tím se dá ušetřit $O(m \log m)$ operací.

$\log m$ je hloubka, od které jsou cesty vkládaných prvků disjunktní

Další možnosti:

<u>Guha (1991)</u> – interpolační mergování

princip stejný jako Hwang, Lin

binární vyhledávání nahrazuje lineárním

<u>Fernandez de la Vega, Kannan, Santha (1993)</u> –

pravděpodobnostní mergování

na základě hodnot n, m se vypočte pravd. p

na základě p se rozhodne, s kterým prvkem

se nově zatřiďovaný začne porovnávat

obecně – pro různé hodnoty $\frac{n}{m}$ mohou být

vhodné různé algoritmy

<u>k-cestné mergování</u>

současně se mergují více než 2 běhy

<u>mergování na místě</u>

nepoužívá pomocné pole (tj. prostorová složitost $O(1)$)

zachovává složitost $O(n)$

Mergování na místě

Kronrod (1969)

jako pomocné pole se použije přímo některý blok
    tříděného pole (tzv. interní buffer)
bloky se musí během výpočtu přeuspořádovat
algoritmus komplikovaný
    v praxi neefektivní

Huang, Langston (1988)

stejný princip
algoritmus prakticky použitelný
    asi 1,5 - 3,5 krát pomalejší než klasické mergování
původně nestabilní
1989 - stabilní mergování na místě

Princip původní verze

máme pole délky n

obsahuje 2 setříděné části (A a B)

1) vybereme $\sqrt{n}$ největších prvků
   dáme je na levý kraj - to bude interní buffer

2) zbytek polí A a B rozdělíme na bloky velikosti $\sqrt{n}$
   bloky přeuspořádáme tak, aby jejich nejpravější prvky tvořily neklesající posloupnost
   bloky očíslujeme

3) mergujeme bloky 1 a 2 pomocí bufferu
   (prvky se postupně vyměňují s prvky bufferu)
   končí se, když se nejpravější prvek bloku 1 dostane na svou pozici

   výsledek: $\underbrace{\text{- - - - -}}_{\text{setříděná část}}\ \underbrace{\text{- - - - -}}_{\text{buffer}}\ \underbrace{\text{- - - - -}}_{\text{zbylé bloky}}$

4) vezmou se další 2 bloky tak, že poslední prvek levého bloku bude větší než první prvek pravého

5) atd.

6) setřídí se buffer

Příklad:

$$\overbrace{2, 3, 6, 7, 11, 12, 15, \underbrace{16, 17, 18}_{C}, \underbrace{21, 25}_{A'}}^{A} \;\vdots\; \overbrace{1, 4, 5, 8, 9, 10, 13, 14, 19, 20, \underbrace{22, 23, 24}_{B'}}^{B}$$

prohledáním obou částí odzadu vyberu $\sqrt{n} = 5$ největších prvků – A', B'

bloky A', B' dám dohromady: vytvořím C – obsahuje $|B'|$ prvků před A

prohodím B' a C (prvek po prvku)

složitost: $O(\sqrt{n})$

mám:

$$\underbrace{2, 3}_{F}, \underbrace{6, 7, 11, 12, 15}_{\sqrt{n}}, \underbrace{22, 23, 24, 21, 25}_{\text{buffer } \sqrt{n}} \;\vdots\; \underbrace{1, 4, 5, 8, 9}_{G\ \sqrt{n}}, \underbrace{10, 13, 14, 19, 20}_{\sqrt{n}}, \underbrace{16, 17, 18}_{C}$$

bude se mergovat 1. blok levé části (F) s 1. blokem pravé části (G)

pomocí |F| posledních prvků bufferu

2, 3, 6, ..., 22, 23, 24, 21, 25,
1, 4, 5, 8, 9, 10, ...

1, 25, 21, 4, ...

25 2

21 3

výsledek:

$\underbrace{25, 21}_{B_1}, \underbrace{6, 7, 11, 12, 15}_{\sqrt{n}}, \underbrace{22, 23, 24}_{B_2}, \underbrace{1, 2}_{H}, \underbrace{3, 4, 5, 8, 9}_{I}, \underbrace{10, 13, 14, 19, 20}_{\sqrt{n}}, \underbrace{16, 17, 18}_{C}$

prohodím H a B_1

H se dostane na svou konečnou pozici

buffer bude pohromadě

složitost : $O(\sqrt{n})$

$$\underbrace{1, 2}_{\text{konec}},\ \underbrace{6, 7, 11, 12, 15}_{J},\ \underbrace{22, 23, 24, 25, 21}_{\text{buffer}},\ \Big|\ \underbrace{3, 4, 5, 8, 9}_{\sqrt{n}},\ \underbrace{10, 13, 14, 19, 20}_{\sqrt{n}},\ \underbrace{16, 17, 18}_{c}$$

prohodím buffer s blokem J – složitost: $O(\sqrt{n})$

$$\underbrace{1, 2}_{\text{konec}},\ \underbrace{22, 23, 24, 25, 21}_{\text{buffer}},\ \underbrace{6, 7, 11, 12, 15}_{\sqrt{n}},\ \underbrace{3, 4, 5, 8, 9}_{\sqrt{n}},\ \underbrace{10, 13, 14, 19, 20}_{\sqrt{n}},\ \underbrace{16, 17, 18}$$

přeuspořádám bloky podle nejpravějších prvků:

$$\underbrace{1, 2}_{\text{konec}},\ \underbrace{22, 23, 24, 25, 21}_{\text{buffer}},\ \underbrace{3, 4, 5, 8, 9}_{1},\ \underbrace{6, 7, 11, 12, 15}_{2},\ \underbrace{16, 17, 18},\ \underbrace{10, 13, 14, 19, 20}$$

složitost: najít minimum z nejpravějších prvků – $O(\sqrt{n})$

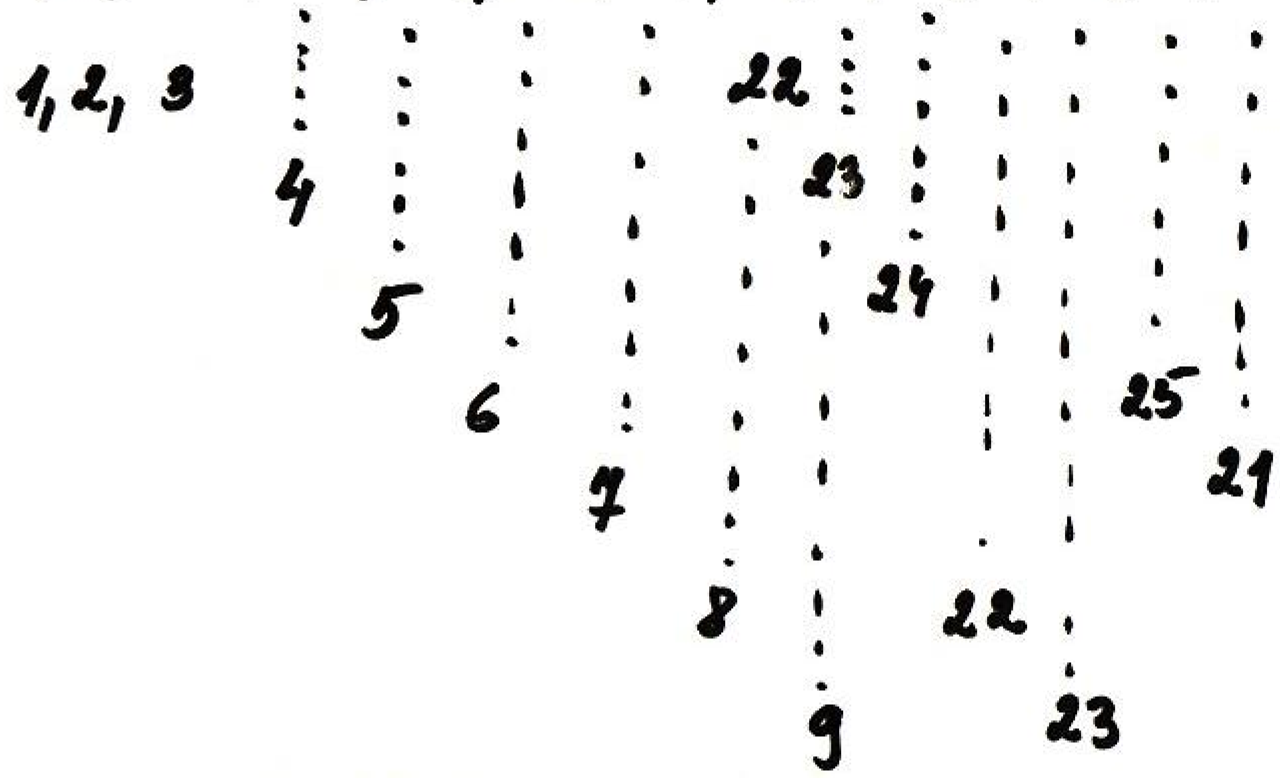
přemístit blok na začátek – $O(\sqrt{n})$

opakuje se $\sqrt{n}$-krát

celkem: $O(n)$

budou se mergovat bloky 1 a 2 pomocí bufferu

1, 2, 22, 23, 24, 25, 21, 3, 4, 5, 8, 9, 6, 7, 11, 12, 15, 16, ...



konec - nejpravější prvek bloku 1 se dostal na svou pozici

složitost: $O(\sqrt{n})$

<u>výsledek</u>:

$\underbrace{1, 2, 3, 4, 5, 6, 7, 8, 9}_{\text{konec}},\ \underbrace{24, 22, 23, 25, 21}_{\text{buffer}},\ \underbrace{11, 12, 15, 16, 17, 18}_{1},\ \underbrace{10, 13, 14, 19, 20}_{2}$

konec bloku 1 : poslední prvek je větší než první prvek bloku 2

blok 2 má velikost $\sqrt{n}$

vše se opakuje nejvýše $\sqrt{n}$-krát

složitost: $O(n)$

nakonec se setřídí buffer

složitost : $O(n)$

<u>složitost celkem</u> : $O(n)$

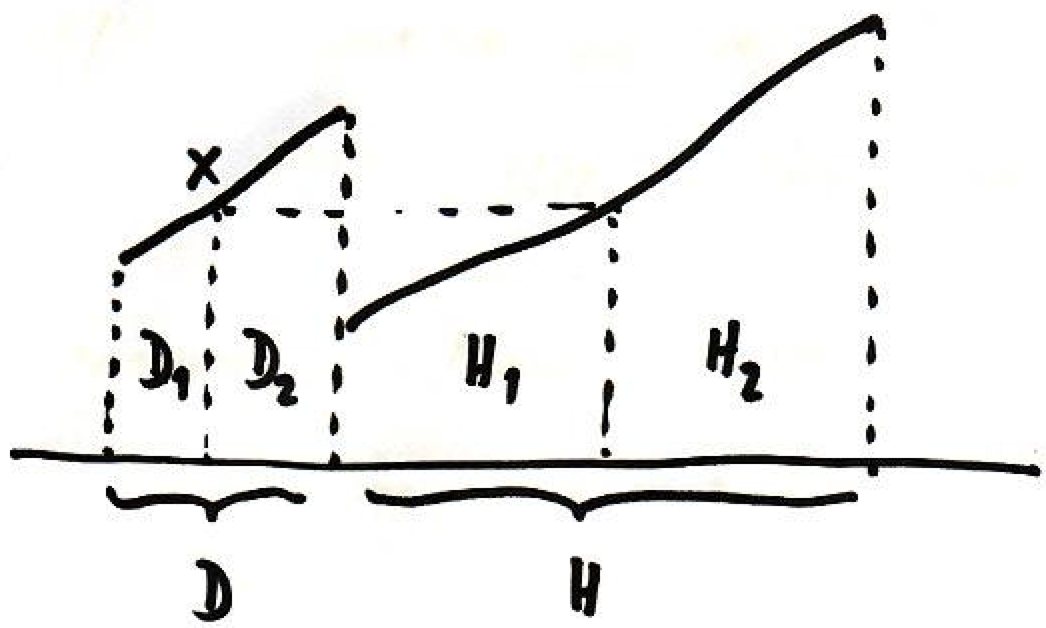
Dvořák, Ďurian (1988)

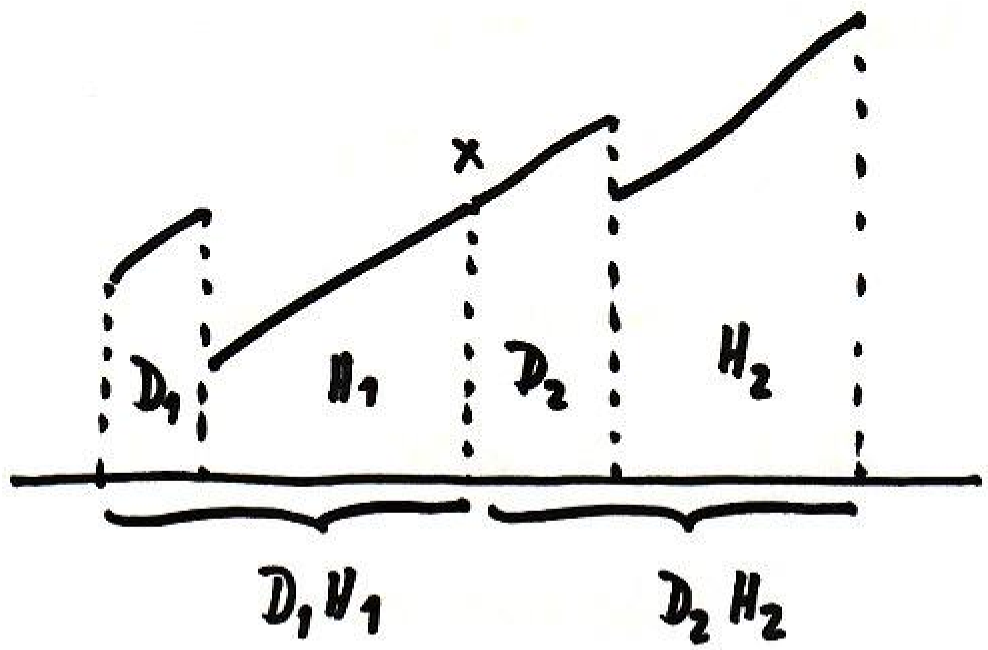
vylepšení metody DM autorů Dudzinski, Dydek (1981)
(která je zjednodušením dekompoziční metody Pratta)

procedure DecMerge (D, H)
  if $|D|>0$ **and** $|H|>0$ **then**   {nechť D je menší než H}
    $x :=$ medián D
    $D_1 := \{y \in D : y < x\}$, $D_2 := \{y \in D : y > x\}$
    $H_1 := \{y \in H : y < x\}$, $H_2 := \{y \in H : y > x\}$
    {máme uspořádání $D_1 \, x \, D_2 \, H_1 \, H_2$}
    prohoď úseky $x D_2$ a H_1
    {máme uspořádání $D_1 \, H_1 \, x \, D_2 \, H_2$}
    DecMerge (D_1, H_1)
    DecMerge (D_2, H_2)
  endif

Složitost: $O(m \log(\frac{n}{m}+1))$ porovnání
$O((m+n)\log m)$ výměn
zásobník pro rekurzi $O(m)$
(verze bez zásobníku $O(1)$)

grafic*ky*:





Výměna úseků xD_2 a H_1 - nejsou stejně velké:
rotace úseku xD_2, rotace H_1, rotace celého úseku

procedure Rot(a,b)
    i:= a, j:= b
    while i<j **do**
        prohoď A(i) s A(j)
        i:= i+1, j:=j-1
    enddo

Symvonis (1995)

stabilní mergování na místě v lineárním čase

Chen (1998) - Quadripartitesort

kombinace Quicksortu a Mergesortu

merguje se na místě pomocí interního bufferu

očekávaný počet porovnání menší než u Quicksortu

čas ?

QUADRIPARTITESORT

Princip (příklad):

pole se rozdělí na poloviny
pravá se setřídí (rekurzivním voláním téže procedury)

třídíme: $\underbrace{3\ 8\ 1\ 7}\ \underbrace{6\ 5\ 4\ 2}$

dostaneme: $\underbrace{3\ 8\ 1\ 7}\ \underbrace{2\ 4\ 5\ 6}$

pravá (setříděná) část se rozdělí na poloviny:

$\underbrace{3\ 8\ 1\ 7}\ \underbrace{2\ 4}_{X}\ \underbrace{5\ 6}_{Y}$

levá část se rozdělí na prvky menší a větší než poslední prvek X:

$\underbrace{3\ 1}_{S}\ \underbrace{8\ 7}_{L}\ \underbrace{2\ 4}_{X}\ \underbrace{5\ 6}_{Y}$

prohodí se bloky L a X:

$\underbrace{3\ 1}_{S}\ \underbrace{2\ 4}_{X}\ \underbrace{8\ 7}_{L}\ \underbrace{5\ 6}_{Y}$

rekurzivně se setřídí levá polovina, tj. SX
(X je setříděná):

dostaneme nejprve $\underbrace{1\ 3}_{S}\ \underbrace{2\ 4}_{X}\ \underbrace{8\ 7}_{L}\ \underbrace{5\ 6}_{Y}$

zmergují se setříděné bloky S a X pomocí L

L dám na začátek: 8 7 2 4 1 3 5 6

1 8

2 7

3 7

výsledek: $\underbrace{1\ 2\ 3\ 4}\ 8\ 7\ 5\ 6$

levá polovina je setříděná

rekurzivně se dotřídí pravá - jako interní buffer pro mergování se použije část levé poloviny

Experimental results are shown in Table 1. The first column, n, is the number of elements to be sorted. For each n, we prepared identical sets of 20 randomly arranged distinct values for each sorting algorithm. The second column, QPsort, is the average number of comparisons needed by QuadripartiteSort. The third column, Wsort, is the result presented for WeakHeapSort from [1]. The last column, Qsort, is the result of a version of QuickSort, which splits with the median of three randomly chosen values. The data in parentheses represent the expected number of comparisons of each instance. By our simulation, the expected value for QPsort is conjectured to be approximately $n \log n - n$. Indeed, the formula values in parentheses always exceed those observed for QPsort. The expected values for columns Wsort and Qsort are approximately $(n - 0.5)\log n - 0.431n$ [1] and $1.188(n + 1)\log(n - 1) - 2.255n + 2.507$ [10], respectively.

With respect to the number of comparisons, empirical results reveal that QPsort is better than Wsort and Qsort. Also, from our testing, we obtained results on the number of exchanges. The number of exchanges observed for Wsort is larger and approximately $0.5n \log n$. The number of exchanges observed for Qsort is almost the same as that observed for QPsort using 256-way merging strategy and approximately $0.25n \log n$.

4. REMARKS

We have described a simple and efficient sorting algorithm which requires at most $n \log n + 1.75n$ comparisons in the worst case. If using 256-way or more merging in Local_MergeSort, this algorithm becomes an in-place (using constant workspace) sorting algorithm whose comparison plus exchange total is optimal. Since an average case analysis is much more complicated, we cannot solve it in this paper. But our simulation results have shown that the average number of comparisons required by this

TABLE 1
The Average Number of Comparisons

n	QPsort	Wsort	Qsort
10	22(23)	26(27)	24(23)
50	225(232)	254(258)	230(231)
100	549(564)	614(619)	571(574)
500	3977(3982)	4247(4271)	4237(4211)
1000	8982(8965)	9497(9547)	9691(9598)
5000	55873(56438)	59222(59367)	61645(61731)
10000	121856(122877)	128465(128740)	135485(135326)
50000	728894(730482)	758835(759824)	820195(814483)

algorithm is approximately $n \log n - n$. This figure is almost equal to that required by MergeSort free to take advantage of $O(n)$ workspace.

ACKNOWLEDGMENTS

The author expresses appreciation to Dr. Yasushi Fuwa and Prof. Yatsuka Nakamura for encouragement and suggestions. I also thank an anonymous referee whose scholarly report helped to improve the readability of the paper.

REFERENCES

1. R. D. Dutton, Weak-heap sort, *BIT* **33** (1993), 372–381.
2. C. A. R. Hoare, Algorithms 63, 64 and 65, *Comm. ACM* **4** (7) (1961), 321–322.
3. E. C. Horvath, Stable sorting in asymptotically optimal time and extra space, *J. Assoc. Comput. Mach.* **25** (1978), 177–199.
4. B. C. Huang and M. A. Langston, Practical in-place merging, *Comm. ACM* **31** (1988), 348–352.
5. J. H. Kingston, "Algorithms and Data Structures: Design, Correctness, Analysis," pp. 175–194, Addison-Wesley, Reading, MA, 1990.
6. D. E. Knuth, "Sorting and Searching," "The Art of Computer Programming," Vol. 3, Addison-Wesley, Reading, MA, 1973.
7. M. A. Kronrod, An optimal ordering algorithm without a field of operation, *Dokl. Akad. Nauk SSSR*, **186** (1969), 1256–1258. (In Russian.)
8. H. Mannila and E. Ukkonen, A simple linear-time algorithm for in situ merging, *Informat. Process. Lett.* **18** (1984), 203–208.
9. L. T. Pardo, Stable sorting and merging with optimal space and time bounds, *SIAM J. Comput.* **6** (1977), 351–372.
10. I. Wegener, Bottom-up heap sort, a new variant of heap sort beating on average quicksort (if n is not very small), *in* "Proceedings of Mathematical Foundations of Computer Science, 1990, Banska Bystrica, Czechoslavakia," pp. 516–522.