Metody řazení ve vnitřní a vnější paměti. Algoritmy řazení výběrem, vkládáním a zaměňováním. Heapsort, Shell-sort, Radix-sort, Quick-sort. Řazení sekvenčních souborů. Řazení souborů s přímým přístupem. Operační a paměťová složitost algoritmů řazení. (A7B36ALG, A7B36DSA)

Metody řazení se dělí:

1. vnitřní řazení - metody vycházejí z předpokladu, že všechna data se vejdou do paměti. V paměti jsou data uložena ve formě datového pole. Rychlost algoritmu je tak závislá na počtu porovnání a záměn.

2. Vnější řazení - používáme tehdy, když objem dat je tak veliký, že nestačí vnitřní paměť a musíme tedy využít i paměť vnější. Rychlost algoritmu je pak ovlivněna počtem přístupů k vnější paměti.

Stabilní řazení - je stabilní tehdy, jestliže po seřazení zachovává vzájemné pořadí prvků se stejným klíčem.

Stabilní algoritmus je takový, který po seřazení zachová pořadí stejných prvků. Například pokud máme seznam studentů seřazený abecedně a seřadíme ho stabilním algoritmem podle čísel kruhů, zůstanou lidé ze stejného kruhu abecedně seřazení. Nestabilní algoritmus by mohl studenty ve stejném kruhu zpřeházet. Nestabilní algoritmy bývají rychlejší!

Příklad: Máme množinu prvků M. Pro každé dva prvky R a S o stejném klíči z této množiny platí, že pokud byl prvek R v neseřazené množině před prvkem S, pak je i v seřazené posloupnosti prvek R před prvkem S. Pokud tato vlastnost platí pro všechny možné množiny M, pak je algoritmus stabilní.

Přirozené řazení – Algoritmus se chová přirozeně, je-li doba seřazení již částečně uspořádané posloupnosti menší, než doba seřazení posloupnosti neuspořádané.

☐ 1, 2, 3, 4, 5, 6, 7, 8, 9 uspořádaná posloupnost

☐ 1, 3, 6, 2, 7, 4, 9, 5, 8 neuspořádaná posloupnost

☐ 1, 3, 2, 4, 5, 6, 8, 9, 7 částečné uspořádaná posloupnost

☐ **přirozený algoritmus seřadí prvky 1,3,2,8,9,7 a prvky 4,5,6 nechává**

Algoritmy řazení s kvadratickou složitost

Selection sort – (zkráceně Selectsort) je jednoduchý algoritmus uspořádávání s časovou složitostí $O(N^2)$. Pro svou jednoduchou implementaci bývá často používán pro uspořádávání malých množství dat.

1. Najdeme prvek s nejmenší hodnotou v posloupnosti dat
2. Zaměníme ho s prvkem na první pozici
3. Na první pozici se nyní nachází správný prvek, zbytek posloupnosti se uspořádá opakováním těchto kroků pro zbylých n-1 prvků, dokud je n > 1

Bubble sort - Řazení záměnou

Algoritmus opakovaně prochází seznam, přičemž porovnává každé dva sousedící prvky, a pokud nejsou ve správném pořadí, prohodí je. Porovnávání prvků běží do té doby, dokud není seznam seřazený. Pro praktické účely je neefektivní, využívá se hlavně pro výukové účely či v nenáročných aplikacích.
Algoritmus je univerzální (pracuje na základě porovnávání dvojic prvků), pracuje lokálně (nevyžaduje pomocnou paměť), je stabilní (prvkům se stejným klíčem nemění vzájemnou polohu), patří mezi přirozené řadicí algoritmy (částečně seřazený seznam zpracuje rychleji než neseřazený).
Název vyjadřuje průběh zpracování, při kterém prvky s vyšší hodnotou „probublávají“ na konec seznamu.

Pseudocode implementation

The algorithm can be expressed as:

```
procedure bubbleSort( A : list of sortable items )
  repeat
    swapped = false
    for i = 1 to length(A) - 1 inclusive do:
      if A[i-1] > A[i] then
        swap( A[i-1], A[i] )
        swapped = true
      end if
    end for
  until not swapped
end procedure
```

Insertion sort je jednoduchý řadicí algoritmus založený na porovnávání. Algoritmus Insert Sort pracuje tak, že prochází prvky postupně a každý další nesetříděný prvek zařadí na správné místo do již setříděné posloupnosti. Je to jeden z nejrychlejších algoritmů s kvadratickou časovou složitostí.

- jednoduchá implementace
- efektivní na malých množinách
- efektivní na částečně seřazených množinách (běží v čase O(N + d), kde d je počet transpozic prvků množiny)
- efektivnější než většina ostatních O(N2) algoritmů (selection sort, bubble sort), průměrný čas je N2/4 a v nejlepším případě je dokonce lineární

Algoritmus

```
insertionSort(array A)
    for i = 1 to length[A] do
        value = A[i]
        j = i-1
        while j >= 0 and A[j] > value do
            A[j + 1] = A[j]
            j = j-1
        A[j+1] = value
```

Algoritmy řazení s logaritmickou složitostí.

Řadící algoritmy, které mají tzv. logaritmickou složitost, jsou již velmi dobře použitelné v praxi při řazení velkého množství dat

Algoritmy Quick Sort a Merge Sort využívá při řešení metodu „rozděl a panuj“. Tato metoda spočívá v tom, že řešený problém rozdělíme na dva nebo i více menších problémů, které jsou podobné původnímu problému. Dílčí problémy pak řešíme nezávisle na sobě a z jejich výsledků složíme výsledek celkový. Dělení původního problému na dílčí problémy musí někdy skončit – dělení končí ve chvíli, kdy k vyřešení zbývají malé snadno řešitelné problémy, které již zpracujeme přímo.

Tato metoda obvykle vede k použití rekurze.

Quick sort - řazení opakovaným tříděním je nejrychlejší univerzální řadící algoritmus. Nevýhodou je, že se jedná o rekurzívní algoritmus, a proto má veliké nároky na paměť. Pro větší objemy dat je třeba s touto skutečností počítat.

Tento algoritmus je typickým představitelem algoritmu založeném na principu „rozděl a panuj“.

Základní myšlenka:
Rozdělíme pole na dvě části – hodnotu dělícího prvku pole nazveme X (pivot) a prvky pole přerovnáme v poli tak, aby v levé části pole byly pouze prvky menší nebo rovné X a v pravé části prvky větší nebo rovné X. Po tomto rozdělení platí, že prvky ležící v levé části pole tam zůstanou i po úplném setřídění celého pole. Totéž platí i pravou část pole. Pak stačí samostatně setřídit levý a pravý úsek pole, jedná se dvě dílčí menší úlohy naprosto stejného typu, jako byla úloha původní - řešíme je naprosto stejným způsobem. Pole postupně dělíme tak dlouho, dokud nezískáme úseky délky 1- ty jsou samy o sobě již seřazené a nemusíme s nimi nic dělat

Volba hodnoty X (pivota):
Na vhodné volbě X závisí rychlost algoritmu. Pokud bychom za X zvolili pokaždé nejmenší (největší) prvek zpracovávaného úseku, rozdělí se pole v prvním kroku na úseky délky 1 a N-1, ve druhém kroku by se větší z nich opět dělil na úseky délek 1 a N-2 atd. Časová složitost by v tomto případě byla $O(N^2)$.
Nejlepší by bylo zvolit za číslo X pokaždé tzv. medián právě zpracovávaného úseku. Medián je číslo s prostřední hodnotou ze všech čísel úseku, např. medián z dvaceti čísel je desáté největší číslo z nich. Přesné vyhledávání mediánu je dosti pracné a v Quick Sortu se nepoužívá!
V Quick Sortu je X stanoveno jako prostřední prvek úseku: (L + R) div 2

Základní kroky algoritmu:

- volba pivotu
- přesun prvků menších než pivot před něj
- přesun prvků větších nebo rovných pivotu za něj
- spuštění algoritmu na část pole obsahující menší prvky
- spuštění algoritmu na část pole obsahující větší prvky (prvky stejné jako pivot je možné přeskočit)

Simple version

In simple pseudocode, the algorithm might be expressed as this:

```
function quicksort('array')
    create empty lists 'less' and 'greater'
    if length('array') ≤ 1
        return 'array'  // an array of zero or one elements is already sorted
    select and remove a pivot value 'pivot' from 'array'
    for each 'x' in 'array'
        if 'x' ≤ 'pivot' then append 'x' to 'less'
        else append 'x' to 'greater'
    return concatenate(quicksort('less'), 'pivot', quicksort('greater'))
```

Merge sort je řadící algoritmus, jehož průměrná i nejhorší možná časová složitost je (*O*(N log N)). Algoritmus je velmi dobrým příkladem programátorské metody rozděl a panuj.

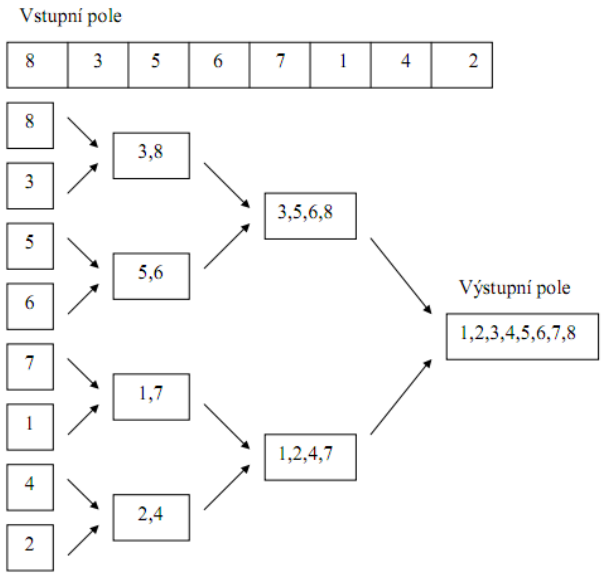
Základní princip:

- Rozdělíme řazené pole na dvě poloviny
- Seřadíme každou polovinu zvlášť
- Obě seřazené poloviny sloučíme do jednoho seřazeného pole

Algoritmus slučování dvou seřazených posloupností S1 a S2 je jednoduchý:
Oba slučované úseky se pomocí 2 pracovních indexů procházejí a prvky se porovnávají. Vždy menší z nich se přesune do cílové posloupnosti. Nakonec zkopírujeme do cílové posloupnosti zbytek S1 nebo S2.
Při slučování se vytvářejí sloučením dvou prvků nejprve uspořádané dvojice, sloučením dvojic uspořádané čtveřice, atd.

Implementace v pseudokódu

```
function mergesort(m)
    var list left, right
    if length(m) ≤ 1
        return m
    else
        middle = length(m) / 2
        for each x in m up to middle
            add x to left
        for each x in m after middle
            add x to right
        left = mergesort(left)
        right = mergesort(right)
        result = merge(left, right)
        return result
```



Velkou nevýhodou oproti algoritmům stejné rychlostní třídy (např. heapsort) je, že Mergesort pro svou práci potřebuje navíc pole o velikosti N.

Řazení sekvenčních souborů

Používá se Merge-sort. U ostatních algoritmů se předpokládal přímý přístup => přístup k poli. Disperze souboru (Soubor A rozdělen na soubory B a C). Potřebujeme tedy 3 soubory – označováno také jako 3-souborové slučování.

Heapsort - neboli **řazení haldou (Řazení výběrem z binárního stromu)** je jeden z nejlepších obecných algoritmů řazení založených na porovnávání prvků. Byť je v průměru o něco pomalejší než quicksort, je jeho zaručená časová náročnost O(N log N) a dokáže řadit data na původním místě (má pouze konstantní nároky na paměť). Heapsort není stabilní řadící algoritmus.

Binární strom - jedná se o datovou strukturu, pro kterou platí tato **pravidla:**
- binární strom se skládá z uzlů, které obsahují nějakou hodnotu a mohou mít nanejvýše dva následníky rozlišené jako levý a pravý
- uzel, který není následníkem žádného uzlu, se nazývá kořen stromu
- uzel, který nemá následníky, se nazývá list
- uzly tvoří hladiny

Halda je binární strom, ve kterém musí být v každém okamžiku splněny následující podmínky:

- pro každý uzel platí, že hodnota v něm uložená je menší než hodnota uložená v jeho libovolném následníkovi
- v každé hladině od první až do předposlední je maximální možný počet uzlů, tzn. v k-té hladině je 2k-1 uzlů.
- v poslední hladině jsou všechny uzly umístěny co možná nejvíce "vlevo", tzn. procházíme-li uzly předposlední hladiny zleva doprava, nejprve má několik z nich (popř. žádný) dva následníky, pak může být (ale nemusí) jeden uzel s jedním následníkem a zbývající uzly předposlední hladiny následníky nemají.

Pokud binární strom splňuje tyto tři uvedené vlastnosti, říkáme že je to **regulérní binární strom**. Pokud uděláme nějakou změnu v stromě, musíme následně zajistit regulérnost binárního stromu.

Přehled běžných univerzálních algoritmů řazení

Název		Časová složitost			Dodatečná paměť	Stabilní	Přirozená	Metoda
Anglicky	Česky	Minimum	Průměrně	Maximum				
Bubble sort	Bublinkové řazení	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	ano	ano	záměna
Heapsort	Řazení haldou	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	ne	ne	halda, záměna
Insertion sort	Řazení vkládáním	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	ano	ano	vkládání
Merge sort	Řazení slučováním	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(\log n)$	ano	ano	slučování
Quicksort	Rychlé řazení	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	ne	ne	záměna
Selection sort	Řazení výběrem	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	zprav. ne	ne	výběr
Shell sort	Shellovo řazení	$O(n^{1+\frac{c}{\sqrt{m}}})$[1]		$O(n \log^2 n)$[1]	$O(1)$	ne	ano	vkládání

Algoritmy řazení s lineární složitostí O(N)

RADIX SORT - Přihrádkové řazení

Tento algoritmus není univerzální, jeho omezení spočívá v tom, že:

- Je použitelný pro řazení pouze celých čísel (resp. záznamů, kde třídícím klíčem je celé číslo)
- Hodnoty, které budeme řadit musí být z předem známého, ne příliš velkého rozmezí. K práci použijeme pole indexované přímo tříděnými čísly. Maximální přípustný rozsah tříděných hodnot proto závisí na tom, jak velké pole si můžeme dovolit vymezit v paměti počítače pro tento účel.

Základní princip:

- Předpoklad: D a H jsou konstanty takové, že všechna tříděná čísla jsou z intervalu <D,H>. Připravíme si H-D+1 "přihrádek". Postupně budeme zpracovávat tříděné záznamy a každý z nich zařadíme vždy do té přihrádky, jejíž označení se shoduje s klíčem záznamu. Záznamy zařazené do téže přihrádky mají stejný klíč a na jejich vzájemné pořadí v principu nezáleží. Jestliže však budeme přihrádky realizovat jako seznamy a budeme v nich zachovávat pořadí jednotlivých záznamů, docílíme toho, že algoritmus bude tzv. stabilní. Po rozmístění všech záznamů do přihrádek pak už jenom stačí obsah přihrádek vypsat jednu po druhé.

Víceprůchodové přihrádkové třídění (řazení podle řádů).
Přihrádkové řazení lze použít i v situaci, jestliže je rozsah hodnot klíčů příliš velký. Je-li například : klíčem až šesticiferné číslo, nebudeme moci deklarovat pole C o milionu prvcích. Použijeme pak tzv. víceprůchodové přihrádkové třídění (řazení podle řádů).

Základní princip:
Nejprve utřídíme posloupnost čísel podle nejnižšího řádu, potom celou posloupnost najednou utřídíme podle vyšších řádů atd. až k řádům nejvyšším.
Na závěr posledního průchodu budou hodnoty seřazeny.

Vstup: 125, 082, 625, 324, 715, 304, 065, 706, 102

Postup řazení:

	0	1	2	3	4	5	6	7	8	9
1.průchod			082 102		324 304	125 625 715 065	706			
2.průchod	102 304 706	715	324 125 625				065		082	
3.průchod	065 082	102 125		304 324			625	706 715		

Výstup: 065, 082, 102, 125, 304, 324, 625, 706, 715

Po skončení třetího průchodu bychom pak vybrali obsah jednotlivých přihrádek (počínaje přihrádkou 0) a dostali bychom seřazenou posloupnost čísel.

Počet průchodů závisí na řádu řazených hodnot. V našem příkladě jsme měli nejvíce čísla třímístná, proto řazení končí po třetím průchodu

Vnější řazení

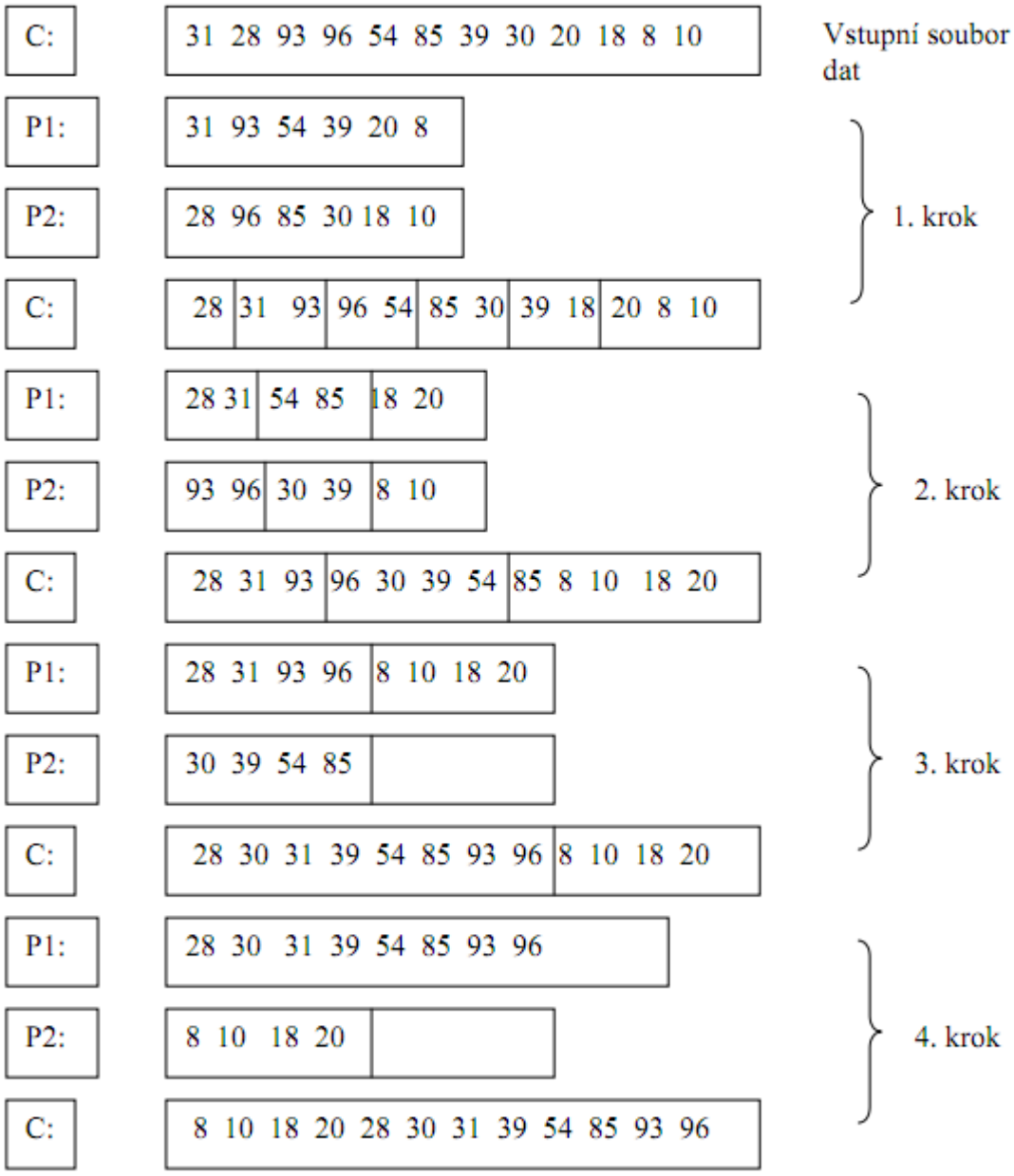
Algoritmy vnějšího řazení slouží k uspořádání rozsáhlých souborů dat. Řeší situaci, kdy se všechna data nevejdou najednou do operační paměti počítače. Data jsou pak umístěna na vnějších médiích. Při takovém uložení dat je velmi nevhodné řadit data některou z metod vnitřního řazení.
Podstatou vnějšího řazení je přesouvání řazených údajů mezi sekvenčními soubory dat uloženými na vnějších paměťových mediích. Při tomto přesouvání se data řadí po částech, postupně se z nich vytvářejí delší a delší seřazené úseky, až budou nakonec všechna data seřazená v jednom souboru.
Základní operací, kterou provádíme se seřazenými úseky dat, je tzv. slučování. Jde v principu o stejný postup jako u vnitřního třídění slučováním . Rozdíl spočívá v tom, že slučované úseky nyní nejsou uloženy v poli, nýbrž jsou v souborech.
Existuje řada různých algoritmů vnějšího řazení. Vůbec nejjednodušší postup se nazývá **přímé slučování.**

Přímé slučování

Předpokládáme, že na začátku práce jsou všechna tříděná data uložena v jednom souboru. Neznáme jejich počet a nevíme ani nic o jejich počátečním uspořádání. Ze všech řazených čísel vytvoříme nejdříve uspořádané dvojice, z nich pak slučováním vzniknou čtveřice, z nich pak osmice atd., až budou všechna čísla seřazena.
Každý krok výpočtu se skládá ze dvou fází. V první, tzv. **rozdělovací fázi** rozdělíme setříděné úseky vstupního souboru do dvou pomocných souborů P1 a P2. Do každého z nich překopírujeme polovinu úseků ze vstupního souboru. Protože předem nevíme, kolik úseků vstupní soubor obsahuje, provádíme rozdělování tak, že čteme vstupní soubor a jednotlivé uspořádané úseky zapisujeme střídavě do P1 a P2. Je-li počet všech úseků lichý, bude P2 obsahovat o jeden úsek méně než P1.
V druhé fázi, tzv. **slučovací fázi** pak ze souborů P1 a P2 vytváříme výsledný výstupní soubor. Nejprve sloučíme první úseky ze souborů P1 a P2 do jednoho uspořádaného úseku dvojnásobné délky a ten zapíšeme do výstupního souboru, potom totéž uděláme s druhými úseky souborů P1 a P2 atd. až do konce souborů. Pokud obsahuje P1 o jeden úsek více než P2, překopírujeme tento úsek do výstupního souboru.

Časová složitost:

U vnějšího řazení jsou z hlediska časových nároků nejvýznamnější vstupně výstupní operace, tj. operace čtení dat ze souboru a zápis dat do souboru. Doba porovnání dvou čísel je oproti nim zanedbatelná.


C:
31 28 93 96 54 85 39 30 20 18 8 10
Vstupní soubor dat
P1:
31 93 54 39 20 8
P2:
28 96 85 30 18 10
1. krok
C:
28 31 93 96 54 85 30 39 18 20 8 10
P1:
28 31 54 85 18 20
P2:
93 96 30 39 8 10
2. krok
C:
28 31 93 96 30 39 54 85 8 10 18 20
P1:
28 31 93 96 8 10 18 20
P2:
30 39 54 85
3. krok
C:
28 30 31 39 54 85 93 96 8 10 18 20
P1:
28 30 31 39 54 85 93 96
P2:
8 10 18 20
4. krok
C:
8 10 18 20 28 30 31 39 54 85 93 96

Stručný popis postupu řazení vnější metodou Merge Sort:

V 1. kroku se provede přepis do pomocných souborů tak, že jedno číslo ze vstupního souboru se uloží do P1, druhé do P2, třetí do P1 a tak až do konce souboru C
Pak se z pomocných souborů vytvoří soubor C tak, že se vezme první číslo v P1 a první číslo v P2, porovnají se a nejdříve se zapíše číslo menší, pak číslo větší. Totéž se udělá s dalšími čísly. V soboru C tak vzniknou uspořádané dvojice.

Ve 2. kroku se do pomocných souborů P1 a P2 přepisují ze vstupního souboru C vždy dvojice čísel.
Pak se z pomocných souborů vytvoří nový soubor C tak, že se vezmou první dvě čísla z P1 a první dvě čísla z P2, porovnávají se a do C se zapíší tak, aby byla ve správném pořadí. Pak se stejným způsobem zpracují další dvojice z P1 a P2. V souboru C tak vzniknou uspořádané čtveřice.

Ve 3. kroku se do pomocných souborů P1 a P2 přepisují ze vstupního souboru C vždy čtveřice čísel.
Pak se z pomocných souborů vytvoří nový soubor C tak, že se vezmou první čtyři čísla z P1 a první čtyři čísla z P2, porovnávají se a do C se zapíší tak, aby byla ve správném pořadí. Pak se stejným způsobem zpracují další čtveřice z P1 a P2. V souboru C tak vzniknou uspořádané osmice.

Ve 4. kroku se do pomocných souborů P1 a P2 přepisují ze vstupního souboru C vždy osmice čísel.
Pak se z pomocných souborů vytvoří nový soubor C tak, že se vezme prvních osm čísel z P1 a prvních osm čísel z P2, porovnávají se a do C se zapíší tak, aby byla ve správném pořadí. Pak se stejným způsobem zpracují další skupiny osmi čísel z P1 a P2. Soubor C v našem případě je již uspořádan

Podklady

http://informatikaou.wz.cz/1-algoritmy_a_datove_struktury_2-treterova-datove_struktury_a_spec_algoritmy.pdf

Wikipedie

Stránky předmětu