PREHĽAD TRIEDENIA V PASCALE

Záverečná práca, Fakulta prírodných vied, UCM Trnava

Ján Bokor

2000

P r e h l á s e n i e

Na tomto mieste záväzne prehlasujem, že táto práca bola napísaná len mnou. Program v Pascale na priloženej diskete je tiež výlučne mojím dielom a bol vytvorený bez pomoci iných osôb.

Autor

OBSAH

ÚVOD

Na triedenie sa spočiatku každý začínajúci programátor pozerá inak, ako keď už robí profesionálne programy. Najprv ako keby preň vôbec neexistovalo. Učí sa vypisovať na obrazovku texty ako „Hello world“ a podobné nezmysly, postupne sa naučí programy vetviť, učí sa pracovať s cyklami a... Kde je triedenie? Na čo vlastne takému programátorovi je? Doteraz som to predsa nepotreboval a už viem predsa všetko... alebo nie?

Aspoň ja som na to prišiel tak trochu sám od seba. Začali sme sa učiť polia. V nich už sa dalo uchovať údajov viac. A potom to prišlo ako blesk z jasného neba. Treba nejaké prvky v poli usporiadať. No to už bude problém. Nájsť najväčší, alebo najmenší prvok nie je ťažké, ale usporiadať? Mnohých napadne, že to bude mať aj čosi spoločné s tým najväčším alebo najmenším prvkom, len ho budeme hľadať viac krát. Ale to bol vtedy veľmi veľký problém. Predsa nebudem robiť také komplikované programy, predsa usporiadať pole nemôže byť taký zložitý problém...

Vlastne ma napadol ten najjednoduchší triediaci algoritmus aký vlastne existuje a ja som ho vtedy považoval za najkomplikovanejšiu úlohu, akú môže programátor riešiť.

Teraz sa doba zmenila (alebo skôr ja), triedenie predstavuje len zanedbateľnú časť programu, ktorý sa zaoberá riešením ďaleko komplikovanejšej úlohy a triedenie je už vecou samozrejmosti a jeho kód sa píše bez akéhokoľvek väčšieho zamyslenia.

A preto je úlohou tejto práce poskytnúť vám niekoľko triediacich metód, možno vás aj trošku motivovať a presvedčiť, že triedenie, vlastne nie je také hrozné ako to spočiatku môže vyzerať. Proste a jasne tu bude snaha o to, aby ste získali prehľad a potom niekedy v budúcnosti, keď budete pracovať na nejakom väčšom programe, aby ste si nerobili zbytočné vrásky z triedenia, aby ste to písali už viac menej automaticky a nemuseli prerušiť tok vašich myšlienok spracovávajúcich určite iný problém.

Ale pekne po poriadku (vlastne aj triedenie o poriadku je). Práca je rozdelená na tri hlavné časti. V tej prvej si definujeme pojem triedenia, povieme si, kde a načo sa asi v praxi využíva. Tu si ich ešte rozdelíme na dve také hlavné triedy, z toho o jednej sa len okrajovo zmienime.

Druhá časť bude pojednávať o samotných metódach tzv. vnútorného triedenia. Päť takých asi najpoužívanejších si rozoberieme pekne dopodrobna a potom si okrajovo povieme o nejakých ďalších.

No a tá posledná časť, to bude vyhodnotenie. Uvidíme na čo sa jednotlivé triedenia hodia, na čo nie, ktoré sú rýchlejšie a ktoré sa neoplatí používať. Nakoniec sú tu ešte prílohy. Jedna obsahuje procedúry s triediacimi algoritmami napísané v jazyku Pascal. Druhá príloha je elektronická, jedná sa o program Prehľad triedenia v Pascale, ktorý mi pomohol pri vyhodnocovaní jednotlivých triedení. V časti Vyhodnotenie som mu venoval jednu kapitolu, obsahujúcu jeho popis.

Pri tvorbe tejto práce nejaké žiadne väčšie problémy neboli, akurát som bol lenivý zaobstarať si viac materiálov z tejto oblasti, ale kniha od N. Wirtha, vlastne len časť o triedení bola tak rozsiahla a obšírna, že som z nej prebral veľa z toho, čo tu bude napísané. Tu taktiež prehlasujem, že všetky vzorce použité v tejto práci boli prebraté z knihy N. Wirtha, Algoritmy a štruktúry údajov. A na konci ma tak trochu tlačil čas, ale to už čitateľa nemusí zaujímať.

Na väčšie prekážky som narážal pri tvorbe programu. Tu som síce všetko relatívne stíhal, vyskytli sa však problémy iného razenia. Ako naprogramovať, aby mohlo bežať viac triediacich algoritmov naraz ? V tom mi narobil najväčšiu galibu Quicksort. Ten je totiž rekurzívny a nerekurzívna verzia je o dosť komplikovanejšia. Ale to som po čase vyriešil, len teda ešte nejaké tie chybičky estetického charakteru tam môžu ostať.

Tak hor sa do toho a dúfam, že vás to obohatí aspoň niekoľkými novými poznatkami.

1. TRIEDENIE

1. 1. Čo je triedenie

Triedenie je proces. Proces, ktorý zoraďuje údaje podľa nejakého kľúča. Ak by sme sa chceli vyjadrovať čo najexaktnejšie, tak pod triedením rozumieme proces preusporiadania danej množiny objektov v špecifickom poradí. Môžeme preusporiadavať číselné informácie vzostupne alebo zostupne, môžeme triediť textové reťazce podľa abecedy, záznamy ľudí podľa výšky ich príjmov a tak podobne. Iste sa s týmto procesom stretne každý začínajúci programátor. Skôr alebo neskôr (skôr to prvé), sa to preň stane nevyhnutnosťou. Je to totiž nevyhnutné pre neskoršie vyhľadávanie údajov.

Každý z nás je už od malička vedený k poriadku (aspoň teda dúfam). Heslo: „Poriadok je pre blbcov, inteligent musí zvládnuť aj chaos...“ je síce pekné, ale prečo si neuľahčiť prácu pri neskoršom vyhľadávaní, ktoré by bez triedenia trvalo omnoho dlhšie. A vyhľadávanie sa bude zrejme aj častejšie používať. Okrem toho na chaos v údajoch uložených v počítači, ktorých je často tisíce ba až milióny by zrejme naša inteligencia proste nestačila.

Čo vlastne triediacimi algoritmami triedime ? Ako som už spomenul, triedime údaje. Zväčša číselné a textové. A tieto číselné a textové údaje môžu byť len triediacimi kľúčmi nejakých zložitejších záznamov. Napríklad podľa “priezviska” roztriedime všetky údaje týkajúce sa “osoby”, aby sme tieto údaje mali k dispozícii ak nejakú konkrétnu osobu (podľa priezviska) v tom mori údajov hľadáme.

Tak sme sa dostali k možnostiam využívania triedenia. Každého už iste napadne, že sa tento proces vo veľkom využíva v každom systéme, ktorý pracuje s väčším množstvom dát. Informačné systémy obsahujúce veľké bázy dát sú asi tým najvhodnejším príkladom.

Vráťme sa však k triedeniu. Ako pracuje taký algoritmus triedenia? Tak iste tu musíme mať akésy pole neutriedených údajov, ktoré nazveme zdrojovou postupnosťou. Áno, je to postupnosť. Vieme vždy povedať, ktorý prvok je prvý, ktorý zaň nasleduje aj ktorý je posledný. Možno si niekto predstavuje, že tieto údaje ležia len v nejakej množine a teda nevieme ako za sebou nasledujú. Ale to nás ani nemusí zaujímať. Našou úlohou je predsa ich

usporiadať. A následnosť týchto údajov v zdrojovej postupnosti je daná ich časom vytvorenia. Totiž, dáta zvyčajne ukladáme do poľa, ktoré je predsa indexované, alebo do súboru, kde sa tiež zvyčajne zapisujú jeden za druhým, čím je určená ich vzájomná následnosť.

A teraz môžu nastať dve situácie. Po prvé, triedená postupnosť obsahuje "relatívne" málo prvkov. Vtedy ju načítame do nejakej premennej typu pole, ak v nej ešte nie je, a použijeme jednu z metód takzvaného vnútorného triedenia. Po druhé, triedená postupnosť je tak veľká, že sa nám nezmestí do operačnej pamäte. Tu nám už metódy vnútorného triedenia nepomôžu. Tu už ide o triedenie sekvenčných súborov, ale tými sa v tejto práci podrobne zaoberať nebudem.

Takže už vieme, čo to triedenie je. Už sme si tak trochu načrtli spôsoby jeho využitia, ale len načrtli. Preto sa pozrime bližšie na aplikácie triedenia v praxi. Aspoň uvidíme, že triedenie je v programovaní nevyhnutnosťou.

1. 2. Využitie v praxi

Mať poriadok v údajoch nie je len pre dobrý pocit, ale je to pre to, aby, keď tieto údaje potrebujeme, sme ich vedeli rýchlo nájsť. Ak máme už údaje roztriedené, stačí nám na ne aplikovať jeden z vyhľadávacích algoritmov, ktorý počíta s tým, že dané pole je už roztriedené!!! a hľadané je veľmi rýchlo nájdené.

V tejto práci sa máme zaoberať triedením. Preto sa o vyhľadávaní nejako ďalej zmieňovať nebudeme. Hádam, pretože to s triedením dosť úzko súvisí, len jeden klasický algoritmus vyhľadávania.

Binárne vyhľadávanie: Rozdelíme pole na dve rovnaké časti, alebo inak povedané, nájdeme údaj uprostred poľa. Porovnáme ho s hľadanou hodnotou. Ak ho nájdeme, tak sme skončili. Ak nie, vylúčime príslušnú časť poľa a pokračujeme v druhej. Nakoniec hľadaný údaj nájdeme a nemusíme pritom prezrieť pole celé.

Až teraz sa dostávame k praktickému využitiu triediacich algoritmov. Načo si budeme uchovávať údaje, ktoré nikdy nebudeme potrebovať? A keď ich budeme potrebovať, tak prečo si nespríjemniť prácu ich vyhľadávaním? Pri menších množstvách dát je to vlastne

možno jedno, ale väčšina dôležitých programov (ak nie všetky) ich spracúvajú stovky, tisíce, ba až milióny. A tu je už ten časový faktor dôležitý.

Prvé, čo by nás mohlo napadnúť, kde sa triedenie údajov vo veľkom využíva, sú databázy. A databázy sú všade. V podnikoch, závodoch, administratívnych ustanovizniach, v rôznych organizáciách, ktoré môžu mať tie najrôznejšie ciele. Hromadia sa v nich ohromné množstvá údajov, s ktorými sa ďalej pracuje. Iste by operátor nebol nadšený, keby mu vyhľadávanie nejakej položky trvalo povedzme viac ako päť minút. A databázy sú implementované aj v každej väčšej aplikácii. Dokonca aj v hrách.

1. 3. Triedenie polí

Pojem triedenia polí sa zvykne označovať aj ako vnútorné triedenie. Je to zrejme preto, lebo proces triedenia počíta hneď s celým poľom a teda celé pole je vo vnútri. Je celé obsiahnuté v procese a s tým sa aj počíta.

Takže metódami vnútorného triedenia si môžeme dovoliť triediť len prvky, ktoré si môžeme dovoliť načítať do poľa. V niektorých prípadoch môže byť limitujúca veľkosť operačnej pamäte, ale napríklad v Borland Pascale si nemôžete dovoliť, aby vaše deklarované premenné nepresahovali 64 kB. To znamená, že metódami vnútorného triedenia sa napríklad nedá v Pascale normálne utriediť pole zložené z prvkov typu integer, ktorý má zvyčajne 2 B, ak tých prvkov je viac, ako 32 765. Okrem toho sa vám do tých 64 kB musia zmestiť aj iné premenné používané v cykloch, premenná použitá pri výmene dvoch prvkov a potrebná bude zrejme aj premenná obsahujúca počet prvkov. A všetky tieto premenné musia mať po 2 B.

A to bol samozrejme len ilustračný príklad. V praxi sa zväčša netriedia obyčajné čísla, triedia sa celé záznamy k tým číslam prislúchajúce a tie majú určite viac ako po 2 B. Tým sme sa dostali k tomu, na čo vlastne metódy vnútorného triedenia používame. Na triedenie relatívne málo početnej množiny prvkov. Nuž, relatívne preto, lebo aj takých zopár tisíc údajov by sme mohli považovať za dosť.

A aby sme sa ešte viac neobmedzovali na veľkosti spracovaných údajov, budú nás ďalej zaujímať len triediace algoritmy pracujúce s jedným poľom. Všetky operácie budú vykonávané len tam. Použiť dve polia, kde jedno by bola zdrojová postupnosť, druhé cieľová,

je príliš jednoduché, neefektívne a pri trochu väčších poliach využívajúce takmer dvojnásobné množstvo pamäti. Hlavne toto bude ešte spomenuté nižšie.

Keďže práve tieto triedenia sú hlavným predmetom našich ďalších úvah, bližšie sa im teraz venovať nebudeme. Až v kapitole nazvanej Analýza si podrobnejšie popíšeme princípy aspoň niektorých z nich. Konkrétne to budú Triedenie priamym výberom, Triedenie priamym vkladaním, Bublinové triedenie, Dobosziewiczovo triedenie a v súčastnosti jeden z najrýchlejších algoritmov s priliehavým názvom Quicksort. Potom si povieme ešte pár slov o niektorých ďalších triedeniach, ale fakt len málo. S týmito triedeniami sa stretneme aj v ďalšej kapitole Vyhodnotenie. Takže by bolo slušné sa zmieniť aj o nejakých algoritmoch vonkajšieho triedenia alebo triedenia sekvenčných súborov, keď už sme ich spomenuli.

1. 4. Triedenie sekvenčných súborov

Triedenie sekvenčných súborov alebo vonkajšie triedenie nepracuje s celou množinou triediacich prvkov naraz. To sa ani nedá. Pretože touto skupinou algoritmov triedime gigantické množstvá údajov, ktoré sa určite nezmestia čo i len do operačnej pamäte a dokonca hovoriť o tom 64 kB obmedzení v Pascale je v tomto prípade, musíte uznať, že smiešne.

A vymyslieť metódy, ktoré to mali zvládnuť sa veľmi rýchlo stalo nevyhnutnosťou. Ako ľudstvo pomaly (pomaly?) vstupovalo do veku informácií, zrazu zisťovalo, že tých informácií potrebuje čoraz viac a viac. Podniky si potrebujú urobiť evidenciu zamestnancov, evidenciu účtovníctva, prípadne aj výrobkov, služieb, ktoré poskytujú. Len jedna položka môže obsahovať tisíce prvkov. Každý prvok môže mať aj desiatky atribútov. Proste týchto dát je vždy viac ako sa môže vmestiť do operačnej pamäte. Dalo by sa to kľudne zaradiť do Murphyho zákonov. A tak prišli na svet vonkajšie metódy triedenia. Tu je aspoň niekoľko z nich.

Prv ako si ich krátko popíšeme, mali by sme vedieť v čom sa hlavne líšia tieto metódy. No, to vlastne vieme, ale ako v praxi nás to obmedzuje? Pri triedení poľa sme mali k dispozícii pole celé, čiže ľahko sa dalo dostať k akémukoľvek prvku. Stačilo len zadať index. V tomto prípade máme prístup len k jednému!!! prvku.

Sekvenčný súbor si môžeme predstaviť ako pásku, na ktorej sú sekvenčne, jeden za druhým, porozhadzované naše údaje. Tie môžeme snímať prostredníctvom nejakej hlavičky, takže vždy máme prístup k len jednému prvku. A to je ten zásadný rozdiel medzi metódami vonkajšieho a vnútorného triedenia.

Triedenie priamym zlučovaním:

Patrí medzi najzaužívanejšie a najdôležitejšie metódy. Spočíva v spájaní dvoch usporiadaných postupností do jednej.

Postup vyzerá asi nasledovne: Zdrojová postupnosť sa rozdelí na dve rovnaké časti. Spájaním prislúchajúcich prvkov oboch častí dosiahneme novú postupnosť, ktorá je tvorená usporiadanými dvojicami. Potom opäť celú postupnosť rozdelíme a rovnako ako minule spojíme. Výsledkom budú usporiadané štvorice. Tento postup aplikujeme dovtedy, pokiaľ nemáme prvky usporiadané.

Triedenie prirodzeným zlučovaním:

Vylepšená metóda triedenia priamym zlučovaním. Počíta s tým, že sa v pôvodnej postupnosti už objavujú aj usporiadané časti, nazývané reťazce alebo behy, ktoré by sme mohli zlučovať. Ak rozdelíme postupnosť na dve a v každej sa nachádza n behov, aj po ich zlúčení bude v postupnosti n behov. Potom sa opäť postupnosť rozdelí a tak sa počet behov zredukuje na n/2. Princíp by mal byť už jasný.

Triedenie vyváženým viaccestným zlučovaním:

Minulé metódy používali tzv. dvojfázové zlučovanie. Tenoraz pôjde o viacfázové zlučovanie, ktoré ale budeme brať ako jednofázové vyvážené triedenie zlučovaním. To znamená, že v každom mieste je rovnaký počet vstupných a výstupných súborov, na ktoré sa po sebe nasledujúce behy striedavo umiestňujú.

To by hádam stačilo. A komu nie tak aspoň zmienka, že existuje polyfázové triedenie, nejaké kombinácie s vnutornými triediacimi algoritmami a iste aj iné. Kto sa chce dozvedieť viac, nech sa obráti na knihu N. Wirtha, Algoritmy a štruktúry údajov, z ktorej bol robený aj tento veľmi krátky výťah.

2. ANALÝZA

2. 1. Obsah triedenia

Prv než sa pozrieme na samotné triediace algoritmy, by sme si mali ujasniť zopár faktov. Triedenie je proces správajúci sa podľa nejakého algoritmu. K naprogramovaniu tohto algoritmu nám stačí ovládať priraďovanie, porovnávanie, prácu s poliami a cyklami. Vlastne sa budeme zaoberať len jedným poľom. Bolo by síce jednoduchšie mať tie polia dve, v jednom by bola zdrojová postupnosť a v druhom by sme to len nejako utriedené naskladali, ale už programátorská etika by nám nemala dovoliť takto márnotratne pracovať s pamäťou.

Budeme sa teda zaoberať len jedným poľom. Teda nám nič iné neostane, len výmenami jeho prvkov dosiahnuť jeho usporiadanosť. Iste poznáte túto postupnosť príkazov:

```
pp:=a;
a:=b;
b:=pp;
```

Je to výmena hodnoty a s hodnotou b za použitia pomocnej premennej pp. A to je v podstate jadro každého triedenia, i keď sa v niektorých prípadoch práve v takomto tvare nevyskytuje. A túto výmenu používa proces triedenia mnoho krát. Snaha získať efektívnejšie algoritmy viedla k znižovaniu počtu týchto výmen.

Avšak pozor! Výmena nie je všetko, čo sa v procese koná. Musia tu byť ešte aj porovnania. Efektivitu algoritmu ovplyvňuje aj ich počet. A je zrejmé, že tieto počty výmen a porovnaní sú vlastne funkciami jednej premennej a to počtu prvkov v poli. Túto hodnotu budeme označovať písmenom n.

Ale vzhľadom na to, že v triedeniach sa používa výmena aj v iných podobách a môžu sa objaviť aj iné priradenia pracujúce s prvkami poľa, skôr sa používa namiesto počtu výmen, počet presunov, čo je vlastne počet priradení. Okrem toho aj výmena pozostáva až z troch príkazov, kým porovnanie iba z jedného. Bolo potrebné si toto ujasniť, lebo mnohí by mohli chápať počet presunov ako počet výmen a ďalej uvedené vzťahy by mu nesedeli.

Čo sa týka označenia, budem ďalej používať pre počet presunov označenie M a pre počet porovnaní C. Celkový počet krokov algoritmu teda bude

$$M + C$$

a bude plne závisieť od počtu prvkov v poli a základného rozloženia prvkov.

Ďalej si budeme konkrétnejšie všímať prípady, kedy je počet krokov maximálny, spravidla ak sú triedené prvky usporiadané opačne, a minimálny, ak už je pole vlastne utriedené. Budú samozrejme uvedené aj priemerné hodnoty. Všetky uvedené vzorce pochádzajú z knihy N. Wirtha, Algoritmy a štruktúry údajov. Takže vrhnime sa na to...

2. 2. Triedenie priamym výberom

Triedenie priamym výberom, patrí medzi tie triedenia, ktorými programátori triediť údaje zväčša začínajú. Ide totiž o ten najjednoduchší algoritmus, ktorí na triedenie vôbec existuje. Jeho princíp spočíva v tom, že prehľadáme celú zdrojovú postupnosť a pri tom nájdeme ten najväčší, resp. najmenší prvok. Ten vymeníme za prvok prvý. Potom prehľadávame opäť celú postupnosť, už samozrejme bez prvého prvku a hľadaný údaj vymeníme s druhým v postupnosti. Celú procedúru takto opakujeme až máme pole roztriedené.

Čiže celý proces triedenia nám rozdelil postupnosť triedených údajov na postupnosti dve. Tá prvá (na začiatku neobsahuje ani prvý prvok) je, alebo bude už utriedená a ďalej sa nemení. Iba postupne sa na jej koniec pridáva ďalší a ďalší prvok až má táto postupnosť zhodný počet prvkov ako zdrojová a vtedy sa proces skončí.

Podrobnejšie si to rozoberieme na príklade. Majme zdrojovú postupnosť prvkov:

10 15 5 25 20

Triedime ju od najväčšieho prvku po najmenší. Čiže treba v tejto postupnosti nájsť prvok s najväčšou hodnotou. To, myslím, by nemal byť veľký problém. V jednej premennej

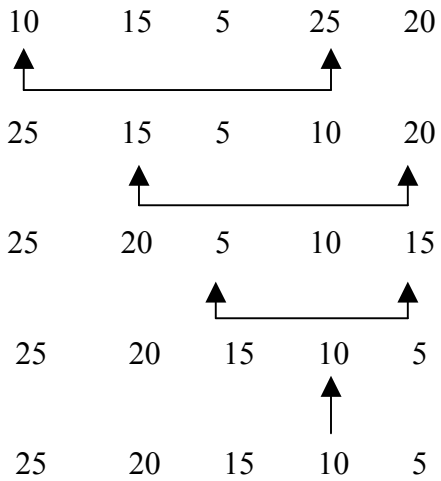
uchovávame najväčšiu, doposiaľ nájdenú hodnotu. Na začiatku do nej vložíme prvý prvok. Je to 10. Porovnáme s ďalším. Je to 15. Keďže je väčšia, priradíme ju našej premennej. Ďalšie číslo je 5. To väčšie nie je, tak ho necháme tak. Takto sa dostaneme k číslu 25, ktoré je väčšie ako 15. Naša premenná uchovávajúca najväčšiu hodnotu sa zväčší na 25. Posledná dvadsiatka už na 25 nemá a tak máme v našej premennej najväčšie číslo nachádzajúce sa v neutriedenom poli.

Pravda, v nejakej ďalšej premennej si aj musíme pamätať, kde sa tá najväčšia hodnota nachádza. Nasleduje totiž výmena najväčšieho prvku poľa (jeho neutriedenej časti!!!) s aktuálnym, v našom prípade prvým prvkom. Postupnosť po tomto prvom prechode vyzerá takto:

25 15 5 10 20

V ďalšom prechode už prvý prvok ignorujeme a pracujeme iba s neutriedenou časťou. Totiž, ak by sme brali do úvahy aj prvý prvok, či neskôr aj prvky patriace do už usporiadanej časti, nikdy by sme pole neusporiadali. Jednoducho kvôli tomu, že by sme vždy našli len ten jediný najväčší prvok postupnosti.

Celý postup usporiadania tohto poľa triedením priameho výberu by prebiehalo takto:



V takomto jednoduchom prípade sa aj zdá, že táto metóda je dosť optimálna. Len nezdá sa vám, že je tu trochu veľa porovnaní ? Teraz sa pozrieme, ako to vyzerá v najoptimálnejšom prípade, v najhoršom prípade a potom si uvedieme priemerné hodnoty počtu presunov a porovnaní.

Najoptimálnejší prípad:

Najoptimálnejší prípad nastane vtedy, ak máme usporiadať pole už usporiadané. Optimálne situácie sú taktiež tie, v ktorých je v poli povymieňaných len zanedbateľné množstvo prvkov. Povymieňaných dvojíc! Ak by bolo pole skoro usporiadané, napr: 5, 25, 20, 15, 10, nie je to až tak optimálna situácia, ako by sa na prvý pohľad mohlo zdať.

Ak ale máme postupnosť už usporiadanú, tak počet presunov sa dá ľahko určiť. Do premennej, kde sa uchováva najväčšia hodnota, sa načíta len prvý prvok, ktorý sa na konci prechodu vlastne vymení len sám so sebou. A ako vieme, výmena pozostáva až z troch presunov. Počet prechodov je n – 1. Čo sa týka počtu porovnaní, po prvom prechode poľom je to n – 1, potom n – 2, až po 1. Čiže tie vzťahy potom vyzerajú nasledovne:

$$M_{min} = 3(n - 1)$$
$$C_{min} = (n^2 - n) / 2$$

Najhorší prípad:

Zrejme už tušíte, že najhorší prípad nastane, ak prvky postupnosti budú usporiadané opačne. Čo sa týka porovnaní, pri každom prechode sa prehľadáva a porovnáva rovnaké množstvo prvkov ako v hoci najoptimálnejšom, teda ich počet je rovnaký, ale počet presunov je väčší. A to o počet ďalších priradení do našej premennej obsahujúcej najvyššiu hodnotu.

$$M_{max} = \text{trunc}(n^2/4) + 3(n - 1)$$
$$C_{max} = (n^2 - n) / 2$$

Priemer:

Počet porovnaní samozrejme ostáva nemenný. Problémy tu vznikajú s počtom presunov. Napriek jednoduchosti algoritmu sa táto hodnota dá ťažko určiť. Preto bude uvedená len približná.

$$M_{priem} \cong n(\ln n + \gamma)$$
$$C_{priem} = (n^2 - n) / 2$$

Kde γ je Eulerova konštanta rovná 0.577216...

Čo napísať na záver. Asi, že tento algoritmus nie je až taký zlý ako by sme mohli očakávať od jeho jednoduchosti. Len počet porovnaní, ak by sa dal nejako zredukovať...

2. 3. Triedenie priamym vkladaním

Triedenie priamym vkladaním vlastne predstavuje opačný prístup, ako triedenie priamym výberom. Opäť si postupnosť rozdelíme na dve časti, z čoho tá prvá bude usporiadaná a druhá nie, ale tentoraz ostane nemenná časť druhá. Teda okrem toho, že z nej bude prvok po prvku ubúdať až celkom zanikne. Čo je pravda pochopiteľné, triedime predsa celú postupnosť.

Na rozdiel od predošlého algoritmu budeme zasahovať tentoraz do prvej, utriedenej časti a to tým, že budeme vkladať na príslušné miesto prvý prvok vybraný z druhej časti postupnosti. Utriedené prvky nachádzajúce sa za týmto miestom vloženia, vrátane toho miesta sa posunú o jedno miesto ďalej, čím sa utriedená časť obohatí o ďalší údaj. Potom podobne zaradíme ďalší, opäť prvý údaj, neutriedenej časti a týmto spôsobom pokračujeme dovtedy, kým nezaradíme posledný prvok pôvodnej postupnosti na svoje miesto.

Opäť si to ukážeme na názornom príklade:

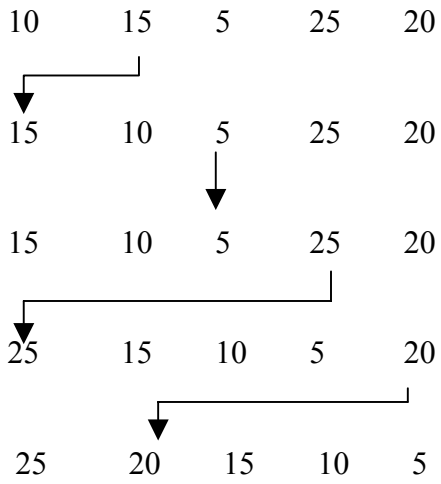
10 15 5 25 20

Toto je naše neutriedené pole. Utriedenú časť v tomto okamihu predstavuje len prvý prvok, teda 10. Opäť triedime od najväčšieho prvku k najmenšiemu. Takže vezmeme druhý prvok, 15 a uložíme ho do poľa pred prvý prvok nášho poľa. To nám poslúži ako akýsi nárazník, čo uvidíme neskôr. Porovnávame s predchádzajúcim prvkom. Je to 10. Vzhľadom na to, že $10 < 15$, desiatku uložíme na miesto 15 a porovnávame s ďalším prvkom. V našom prípade ide o nultý prvok, kam sme vložili našu 15. Tu už neplatí, že $15 < 15$ (to je tá funkcia nárazníka), takže s triedením druhého prvku sme skončili.

Postupnosť teraz vyzerá takto:

15 10 5 25 20

Usporiadanú časť tvoria už dva prvky. Preto vezmeme tretí a podobne ho vložíme na svoje miesto. Celý postup je zobrazený v nasledujúcej tabuľke.



Táto metóda je síce podobná ako metóda triedenia priamym výberom, zdá sa dokonca, že i počet výmen je väčší, ale teraz si ukážeme, že zdanie môže klamať. Veľmi klamať.

Najoptimálnejšia situácia:

Je to vtedy, ak sú prvky v triedenej postupnosti už roztriedené. Dochádza k minimálnemu počtu presunov a teda aj porovnaní. Ale v každom prechode poľom musíme porovnať prvé dva prvky, aby sme zistili, že ich už nemusíme vymieňať. Čo sa týka presunov vždy na začiatku dávame aktuálny prvok na nultú pozíciu a potom na konci ho berieme späť. Čiže vzťahy vyzerajú nasledovne:

$$C_{min} = n - 1$$
$$M_{min} = 2\,(n - 1)$$

Najhoršia situácia:

Nastane vtedy, ak triedime opačne usporiadané pole. Vtedy sa nám počet krokov zhruba n – znásobí:

$$C_{max} = 1/2\,(n^2 + n) - 1$$
$$M_{max} = 1/2\,(n^2 + 3n - 4)$$

Spočíva to najmä v tom, že pri každom prechode musíme zachádzať ako s porovnaniami tak i s presunmi až na začiatok usporiadanej časti postupnosti, kde bude proces zastavený až tým nárazníkom, čo bol predtým spomenutý.

Priemer:

Tu dúfam, postačí ak uvediem len vzťahy.

$$C_{priem} = 1/4\ (n^2 + n - 2)$$
$$M_{priem} = 1/4\ (n^2 + 9n - 10)$$

Vylepšenie metódy:

Ako sa už spomínalo, behom procesu je postupnosť rozdelená na usporiadanú a neusporiadanú časť. Pričom do tej usporiadanej časti sa postupne pridávajú prvky. Ale keďže tá je už usporiadaná prečo nevyužiť nejaký vyhľadávajúci algoritmus, ktorý rýchlejšie určí miesto vloženia nášho prvku ? Niekto to zrejme urobil a bolo tu triedenie binárnym vkladaním. Binárnym preto, lebo bolo použité binárne vyhľadávanie.

Toto triedenie je efektívnejšie pri väčšom množstve triedených údajov a ak sú tieto údaje dosť „divoko“ poprehadzované. Naopak, správa sa ešte horšie v prípade, ak sú prvky už usporiadané.

Čo sa týka vzťahov vyjadrujúcich počty porovnaní a presunov, tak počet presunov je stále rádu n^2 a výraznejšie sa nemení. Takže len priemerný počet porovnaní:

$$C = n\ (\log n - \log e \pm 0.5)$$

Takže to bola ukážka ďalšieho triedenia. Ale... nie je tam nejako veľa presunov ? Keď sa musí od miesta vloženia posunúť každý prvok o miesto ďalej. Uvidíme, čo nám prinesú nasledujúce metódy.

2. 4. Bublinové triedenie

Tu bola zrejme snaha skombinovať obe predchádzajúce triedenia a tým vytvoriť nové. To sa v podstate aj podarilo, lenže sme tým vynašli asi najmenej efektívny algoritmus vnútorného triedenia, aký sa dá. Ale zas na zahodenie nie je. Jeho modifikácie jeho efektivitu môžu zvýšiť.

Toto triedenie sa pôvodne nazývalo triedením priamou výmenou, ale charakteristické „vybublávanie“ prvkov smerom nahor (t. j. na svoje miesto) ho poznačilo i na názve.

Prečo behom jediného prezerania zdrojovej postupnosti máme pracovať len s jedným jej prvkom ? Zoberieme si posledný prvok a porovnáme ho s predposledným. Ak spĺňa podmienku výmeny, tak ho vymeníme a porovnáme s ďalším prvkom. Takto pokračujeme až nenarazíme na koniec neusporiadanej časti poľa alebo na prvok, ktorý túto podmienku výmeny nespĺňa. V tom prípade si vezmeme do parády práve ten prvok a pokračujeme s ním.

Iste si všimnete, že po prvom prechode poľa sa na prvé miesto dostane ten správny prvok. Čiže ďalej môžeme pracovať už len s poľom o jeden prvok menším. A opäť máme postupnosť rozdelenú na dve časti.

Príklad. Vezmeme si zdrojovú postupnosť takú, akú sme používali v predošlých prípadoch.

10 15 5 25 20

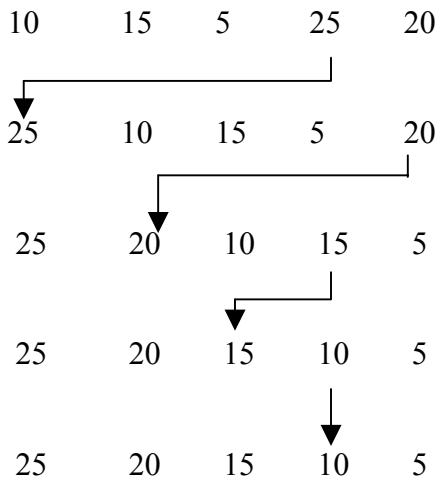
Tentoraz začíname od konca (ale vlastne sa dá aj od začiatku). Posledné je číslo 20. Je väčšie ako 25? Nie. Takže 20-kou sa prestaneme zapodievať a vrhneme sa na 25. Tá je väčšia ako 5. Aj ako 15 a aj ako 10. Postupným vymieňaním sa dostane na prvé miesto, kým ostatné prvky o jedno posunie.

Dostaneme postupnosť:

25 10 15 5 20

Tentoraz je posledná 20. Túto nič nezastaví a tak sa ňou prebubleme až na druhé miesto. Ale s 25-kou sme ju však už neporovnávali. Po prvom prechode sme totiž na prvú pozíciu dosadili najväčší prvok, takže druhý prechod sa pri druhom prvku už zastaví.

Celý proces bude vyzerať nasledovne:



Najoptimálnejší prípad:

Prvky sú už usporiadané. Keďže v tomto triedení z hľadiska presunov manipulujeme s prvkami len ich vymieňaním, je zrejmé, že keď už je pole usporiadané, nebude sa vymieňať nič a teda počet presunov bude nulový.

Počet porovnaní. Tak s tými musíme ísť pri každom prechode až do konca neusporiadanej časti poľa, či prvky máme usporiadané, či nie. Takže vzťahy majú nasledujúcu podobu:

$$C_{min} = 1/2\ (n^2 - n\ )$$
$$M_{min} = 0$$

Najhoršia situácia:

To sú opačne usporiadané prvky. Ako sme spomínali vyššie, počet porovnaní nezávisí od usporiadanosti prvkov v poli, bude teda rovnaký ako pri najoptimálnejšej situácii. Ale počet presunov sa rapídne zvýši. Keďže sú prvky na začiatku opačne usporiadané, musíme za každým porovnaním vymeniť prvky. Čiže počet presunov bude toľko, koľko počet porovnaní krát tri. Lebo výmena sa skladá z troch presunov.

Tu sú vzorce:

$$C_{max} = 1/2\ (n^2 - n\ )$$
$$M_{max} = 3/2\ (n^2 - n\ )$$

Priemer:

Priemerný počet porovnaní nie je ťažké určiť, vzhľadom na to, že pre obe krajné prípady je rovnaký. To isté to bude tu. A priemerný počet presunov sa rovná aritmetickému priemeru počtov presunov v oboch krajných prípadoch.

$$C_{priem} = 1/2\ (n^2 - n\ )$$
$$M_{priem} = 3/4\ (n^2 - n\ )$$

Modifikácie metódy:

Mnohokrát sa stáva, že postupnosť je už utriedená, ale proces ešte nekončí. Prebiehajú tam zbytočné porovnávania, každým prechodom sa porovnáva to isté. Tomu sa dá ľahko zamedziť, zavedením nejakej premennej, ktorá by nám dávala informáciu o tom, či v aktuálnom prechode poľom nenastala nejaká výmena. Po prechode poľom si to zistíme a keď nie, tak končíme.

Touto modifikáciou neznížime počet presunov. Ten ostáva rovnaký. Maximálny počet porovnaní tiež. Minimálne však pole musíme aspoň raz prejsť, aby sme sa uistili, že nemusíme naozaj nič vymieňať. Čiže vzťahy pre počet porovnaní pre toto vylepšenie budú nasledovné:

$$C_{min} = n - 1$$
$$C_{max} = 1/2\ (n^2 - n\ )$$
$$C_{priem} = 1/2\ [n^2 - n\ (k + \ln n)]$$

Ďalšie vylepšenie spočíva v odstránení asymetrie tohto triedenia. Majme pole 20 15 10 5 25. Bublinové triedenie usporiada prvky v podstate v priebehu prvého prechodu. Kým s poľom 5 25 20 15 10 musí algoritmus prebehnúť až do konca. Pričom v oboch prípadoch nám tam „robí šarapatu“ len jeden prvok. Čo keby sme po každom prechode poľom zmenili smer postupu?

Prvý prechod by prebehol od konca na začiatok. Tým by sme dosadili na prvé miesto najväčší prvok. Druhý prechod realizujeme od druhého prvku na koniec. Na koniec tým dosadíme najmenší prvok. A tak pokračujeme ďalej, znova z konca na začiatok, zo začiatku na koniec (pričom vynechávame už usporiadané prvky na okrajoch poľa) až skončíme kdesi v strede poľa. Pri vyššie spomenutej vylepšenej verzii aj skôr.

Táto metóda dostala názov Triedenie pretriasaním. Ale jediné, čo nám priniesla bolo odstránenie asymetrie, ktorú sme si ukázali pred vysvetlením metódy. Takto budú oba spomenuté prípady utriedené za zhruba rovnaký počet prechodov. Konkrétne sa budú líšiť len o jeden prechod, bude to závisieť od toho, z ktorej strany začneme. Čo sa týka počtu porovnaní a počtu presunov, vzťahy sú totožné s bublinovým triedením a vylepšeným bublinovým triedením.

2. 5. Dobosziewiczovo triedenie

Iste ste si všimli pri predchádzajúcich postupoch, že sme prezerali celé polia, teda celú ich neusporiadanú časť. Museli sme pritom robiť mnoho porovnaní a v bublinovom triedení a triedení priamym vkladaním sa robili výmeny prvkov, ktoré boli vedľa seba. Nie je to síce zlé, ale keď si spomeniete na triedenie priamym výberom, kde sa robili výmeny na väčšie vzdialenosti, tam bol počet porovnaní rádovo nižší ako n^2. Ak by sa nám takto podarilo zredukovať aj porovnania, zrejme by sme dostali o dosť efektívnejšiu metódu.

Počet porovnaní úzko súvisí s prehliadaním poľa. Doteraz sme prehliadali pole z jedného konca na druhý. Čo keby sme teraz neprehliadali celé pole. Proste sa spýtame, či treba dajaké dva vybrané prvky vymeniť, alebo nie. A keďže sú efektívnejšie výmeny na väčšiu diaľku, zvolíme si nejakú konštantu k, ktorá nám bude udávať, od ktorého prvku začneme. Najväčší efekt dosiahneme pre k = 2/3 n. To k berieme samozrejme celočíselné. Potom už len porovnávame, prípadne uskutočníme výmenu. Porovnáme prvý prvok s k+1 –ím prvkom, druhý s k+2 –ím prvkom a tak ďalej až dôjde k porovnaniu s posledným prvkom. Potom k vypočítame ako 2/3 aktuálneho k. Opäť ho berieme len ako celé číslo. Ak nám vyjde 0, položíme k = 1.

V našom príklade je prvé k rovné trom. Ak máme teda našu starú známu postupnosť prvkov:

10 15 5 25 20

...porovnáme prvý prvok so štvrtým. Keďže 25 je viac ako 10, urobíme výmenu. Pokračujeme ďalším prvkom. Je 15 viac ako k+2. prvok ? k+2. prvok je piaty prvok, čo je 20, čiže prvky 15 a 20 vymeníme. Porovnávali sme však už aj posledný prvok a tak musíme urobiť nový výpočet čísla k. A s tým ideme opäť od začiatku. Pozor!!! Od začiatku!!! Tu už nie je pole rozdelené na dve časti ako v predošlých prípadoch.

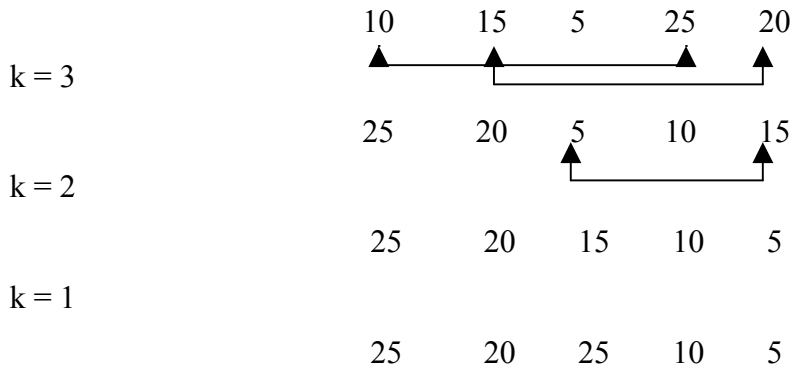
k = trunc((2/3) * k). Po dosadení vyjde číslo 2. S týmto k-čkom triedime ďalej.

25 20 5 10 15

Porovnávame 25 s 5, 20 s 10 a 5 s 15. V poslednom prípade dôjde na výmenu.

Kedy končiť? Keď sme takto prešli postupnosť s k rovným jednej a nenastala už žiadna výmena. Môže dôjsť k prípadu, keď k bude nula. Vyššie sme spomenuli, že v tom prípade položíme k rovné 1 a prechádzame pole s tým.

Takto sa bude správať naše pole počas Dobosziewiczovho triedenia:



Hneď po prvom prechode s k = 1 nebola uskutočnená ani jedna výmena takže môžeme skončiť a pole je utriedené.

Najoptimálnejší prípad:

Nastane samozrejme vtedy, ak prvky sú už utriedené. Počet presunov bude iste nulový a počet porovnaní bude tiež minimálny možný, lebo porovnávať musíme tak či tak a najskôr môžeme skončiť až keď prvý krát prebrázdime pole s k = 1. Bohužiaľ sa mi nepodarilo dostať k materiálom obsahujúcich presné vzťahy a bola aj snaha si nejaké odvodiť, ale dostal som sa len k približným výsledkom. Hlavné je, počet porovnaní je približne úmerný n ln n, čo už naozaj dokazuje veľkú efektivitu algoritmu.

Najhorší prípad:

No tu to už nie je až také jednoduché. Hlavne preto, že sa prvky porovnávajú na veľkú vzdialenosť a v prvých prechodoch na mnohé prvky vôbec nemusí dôjsť. Čiže prvý prechod by príliš malé, alebo veľké prvky mohol kľudne preskočiť a pri druhom prechode by sa na svoje posty presúvali pomalšie. Nemôžeme teda teraz tvrdiť, že najhorší prípad nastane pre opačne usporiadanú postupnosť.

Stačí uviesť jeden príklad. Majme štvorprvkovú postupnosť: 5 10 15 20. Je opačne usporiadaná. Aby ju Dobosziewiczov algoritmus utriedil, potrebuje 8 porovnaní a 4 výmeny. Avšak pre zdrojovú postupnosť: 5 15 10 20, potrebuje porovnaní až 11!!! a výmen 5. Tie tri porovnania naviac nabral vďaka tej poslednej výmene, ktorá sa uskutočnila v prechode, ktorý bol minule braný už ako posledný. Takže musel pole prezrieť ešte raz. To znamenalo plus tých n – 1 porovnaní.

Čo sa týka vzťahov, počet porovnaní bude taký istý ako v najlepšom prípade len teda plus niekoľko málo krát n – 1, čo tam nabehne pre opätovné prehliadanie poľa s k = 1. Pre veľké postupnosti stále platí, že počet porovnaní je približne rádu n ln n. Počet presunov, to je tri krát počet výmen. A počet výmen je celkom určite menší ako počet porovnaní. Minimálne o to n – 1, i keď aj tie predošlé porovnania sa niektoré obídu bez výmeny. Počet krokov by sme teda tiež mohli odhadnúť rádovo na n ln n.

Priemer:

Ak to celé zhrnieme, neostáva nám nič iné, než napísať, že Dobosziewiczovo triedenie je oveľa efektívnejšie ako predošlé uvedené triediace algoritmy. A ak by sme mali k dispozícii aj vzťahy, nenašli by sme tam člen n^2.

2. 6. Quicksort

Názov už môže o tomto triedení vypovedať mnoho. Je samozrejme opodstatnený. Je to totiž najrýchlejší algoritmus na triedenie aký zatiaľ poznáme. Jeho pôvodný názov je triedenie rozdeľovaním, lebo jeho základným princípom je rozdeliť pole na dve časti (tentoraz nie na

usporiadanú a neusporiadanú) a vymieňaním prvkov medzi nimi dosiahnuť usporiadanosť poľa. Rýchlym triedením ho však nazval už aj jeho autor C. A. Hoare a to sa ujalo.

Algoritmus vychádza z podobnej úvahy ako Dobosziewiczov triediaci algoritmus a síce z toho, že viac sú efektívne výmeny na väčšie vzdialenosti. To je však jediné, čo tieto algoritmy spája. Ako sme už naznačili tu sa pole hneď na začiatku rozdelí na dve zhruba rovnaké časti a to prvkom ležiacim v strede poľa. Potom sa vezme prvá polovica. Prvý prvok. Porovnáme ho s prvkom v strede. Ak je väčší, čiže spĺňa podmienku, že v prvej polovici chceme mať väčšie prvky, necháme ho tam kde je a ideme ďalej. Porovnáme druhý prvok s tým v strede a tak pokračujeme až nenájdeme nejaký prvok, ktorý je menší alebo sa aspoň rovná nášmu strednému prvku. Iste je každému zrejmé, že ďalej ako do stredu sa nedostaneme.

Podobne zapracujeme aj na druhej strane. Vezmeme posledný prvok, porovnáme ho s tým v strede, postup je zrejmý. Nakoniec nájdeme dva prvky, ktoré vymeníme. V tomto duchu pokračujeme dovtedy, pokiaľ sa nám prehľadávacie premenné (jedna ide zo začiatku dopredu, druhá od konca) neprekrížia. Výsledkom je to, že v prvej polovici poľa máme väčšie prvky a v druhej menšie. Čiže stačí stanoviť len nové hranice a rekurzívne zavolať ten istý postup. Pre väčšie pochopenie si to ukážeme na príklade.

Majme postupnosť čísel, na ktoré si už všetci iste zvykli. Chceme ju usporiadať od najväčšieho prvku k najmenšiemu.

10 15 5 25 20

Stredný prvok je 5. Pred ním chceme mať väčšie prvky a za ním menšie. Náhodou nám vyšlo, že práve prvok v strede je najmenší, ale to nevadí. Prvé číslo je 10. Je väčšie, ako 5? Je. Necháme ho tok jak je a vrhneme sa na druhý prvok. Ani o 15 však nemáme záujem a tak sa dostaneme až k strednému prvku. Ten je väčší alebo rovný sám sebe, tak sa uňho zastavíme.

Prezrieme pole zozadu. Hneď prvá 20 nám špiní rady, kde by mali byť malé hodnoty a tak vstupuje do operácie výmeny s číslom 5.

10 15 20 25 5

V nasledujúcom kroku sa prehliadajúce premenné stretnú na štvrtom prvku. Nastane nezmyselná výmena (lebo v programe je `i<=j`) a potom sa i a j prekrížia. Treba definovať

nové hranice. Tu sa postupnosť delí na dve časti, jedna (ľavá) s hranicami 1 až 3, druhá... V tomto prípade i = 5 takže druhá časť nebude. Na tú prvú aplikujeme presne ten istý postup. Stredný prvok je 15 na druhom mieste. Hneď krajné prvky vyhovujú výmene, takže ich vymeníme a postupujeme ďalej. Skončíme samozrejme keď si takto rekuretne rozdrobíme postupnosť na jednoprvkové postupnosti.

Analýza:

Trošku vybočíme zaužívaných častí ako sú najoptimálnejší prípad, najhorší prípad a priemer. Môžeme si aspoň povedať, pre ktoré prípady je toto triedenie naozaj rýchle. Samozrejme si dobre vedie aj v tých priemerných situáciách, ale nemôžem nespomenúť fakt, že prípad, ktorý prvé tri rozoberané triedenia brali ako najhorší, Quicksort zvládne behom prvého prechodu. Vlastne po vzore tohto príkladu bol Quicksort aj stavaný, ak sa nad tým zamyslíte.

Čo sa týka porovnaní, behom jedného prechodu poľom alebo jeho časťou je ich počet rovný počtu prvkov, ktoré danú časť postupnosti tvoria. Na začiatku to bude n. Celkový počet porovnaní by mal byť teda:

$$C_{priem} = n \log n$$

Výmeny však nenasledujú za každým porovnaním ani v tom najhoršom prípade. Priemerne je ich šesťnásobne menej takže:

$$V_{priem} = n/6 \log n$$

A presunov sa nezbavíte ani keď budete triediť už utriedené pole. Vždy sa totiž to pole bude deliť, zakaždým sa bude počítať stredný prvok...

No ešte sme nespomenuli ten najhorší prípad. Už sme spomenuli, že obrátené poradie patrí skôr medzi tie lepšie prípady, tak čo bude to najhoršie? Nuž, to nám už tak trochu ukázal náš príklad. Myslím, že mnohí mali z toho tak trochu pocit, že sa neuberáme k usporiadaniu tou najlepšou cestou a že je to trochu nemotorné. To robil ten prvok v strede, ktorý bol zhodou okolností najmenší. A najmenšie, či najväčšie prvky v stredoch intervalov, to je tá najväčšia slabina algoritmu qiucksort. V tom prípade sa naše pole vždy rozdelí len na jeden interval (alebo dva, z toho ten druhý má už len jeden prvok), presne ako v našom názornom

príklade a to skvelé log n v počte porovnaní sa razom priblíži n. Čiže z n log n sa stane n^2 a triedenie sa stane porovnateľným s triedením priamym výberom a podobných.

Modifikácia:

V našom prípade sme pole rozdeľovali podľa stredného prvku. V prípade, že sa tam nachádza nejaký extrémny prvok, je to však kameň úrazu tohoto triedenia. A z rýchleho triedenia sa stane triedenie pomalé. To nás zvádza k tomu, aby sa prvok, podľa ktorého pole rozdeľujeme, vybral iný ako stredový. Celkom dobre by mohol byť zvolený prvok prvý alebo posledný. To by zlepšovalo situáciu, ktorá bola ako najhoršia popísaná vyššie.

Tak, či tak, v skutočnosti dosiahneme najlepšie priemerné hodnoty pri použití stredného prvku. Autor triedenia Hoare ešte navrhuje, aby sa tento prvok bral ako medián, tj. stredná hodnota malej vzorky, napríklad troch prvkov náhodne vybratých z aktuálnej časti postupnosti. Priemerne to síce neovplyvní počet krokov, ale môže to značne zlepšiť výkon v najhoršom prípade.

A čo dodať nakoniec. Ďalšie triedenie, dokonca momentálne jedno z najlepší, ale taktiež má svoje nedostatky. Napríklad dokáže ignorovať postupnosť, ktorá je už zväčša, alebo jej časti sú už utriedené. Počkáme si na vyhodnotenie.

2. 7. Niektoré ďalšie triedenia

Predtým, než začneme vyhodnocovať uvedené a vyššie rozpísané triedenia, napíšem ešte zopár slov o niektorých ďalších, aby to nevyzeralo, že tie hore, sú jediné vnútorné triedenia, ktoré existujú. Určite majú všetky nejaké tie modifikácie, vylepšenia, niektoré sme si spomenuli, iste mnohé ani nie, ale keď človek pochopí princíp daného triedenia a trochu sa nad tým zamyslí, je dosť možné, že príde opäť na niečo nové. Tak tu máte ďalšie inšpirácie.

Shellovo triedenie:

Jedná sa vlastne o zlepšenie metódy triedenia priamym vkladaním. Jej druhý názov aj znie triedenie vkladaním so zmenšovaním kroku. V čomsi ale zas pripomína aj

Dobosziewiczovo triedenie. Tiež sa tam vyberá nejaký ten krok k, ktorý môžeme považovať za vzdialenosť, na ktorú budeme dva prvky navzájom porovnávať. A podobne, ako v Dobosziewiczovom triediacom algoritme, sa toto k bude zmenšovať.

Čo sa týka zas podobnosti s triedením priamym vkladaním, tam je zas rovnaký systém výmen. Aby bolo všetko všetkým jasné, tak je to vlastne triedenie priamym vkladaním, s tým, že berieme ako postupnosť prvý prvok, k + 1 –vý prvok, 2k + 1 –vý prvok až kam to pôjde, potom druhý prvok, k + 2 –hý prvok a tak ďalej, až cez k –ty prvok, 2k –ty prvok... A potom toto k zmenšíme (napríklad na polovicu) a môžeme si celý postup zopakovať. Skončíme pri prechode s k = 1.

Stromové triedenie:

Toto je zas založené na princípe triedenia priamym výberom. Totiž, prečo by sme si mali pri tomto triedení všímať iba náš hľadaný najväčší resp. najmenší prvok? Prečo by sme si behom prechodu poľom nemohli „dať bokom“ i nejaké tie informácie o zložení poľa? Napríklad prostredníctvom porovnania každého druhého prvku s so svojim ľavým susedom môžeme určiť, ktorý je väčší z danej dvojice. Ďalším prechodom prehliadajúcim už len polovice poľa, ktorá je určená tými nájdenými prvkami z prvého prechodu, nájdeme tak maximálne prvky každej štvorice prvkov. To speje k vytvoreniu akéhosi binárneho stromu, na vrchole ktorého bude tróniť náš hľadaný najväčší prvok. Tento najväčší prvok potom z celého stromu buď nahradíme prázdnou hodnotou alebo hodnotou ekvivalentnou s $-\infty$. Potom už len postupujeme od koreňa stromu k jednotlivým listom. Na druhej úrovni narazíme len na jednu použiteľnú hodnotu (ostatné, tie nekonečné nás nezaujímajú), ktorá je druhou najväčšou. Takto postupujeme a vymazávame hodnoty zo stromu, až jedinými prvkami na strome budú prázdne hodnoty alebo záporné nekonečná.

Toto triedenie je efektívnejšie ako prvé tri uvedené a dopodrobna preberané, dokonca je efektívnejšie než vyššie spomenuté Shellovo triedenie.

Triedenie haldou:

Je modifikácou stromového triedenia. Vlastne vylepšením stromového triedenia. Každého iste napadne, že pre stromové triedenia potrebujeme 2n – 1 pamäťových buniek, ktoré reprezentujú náš strom. Ďalej, nebolo by jednoduchšie z nášho stromu priamo odstraňovať prvky namiesto nahradzovať nejakými prázdnymi hodnotami? Toto vŕtalo hlavou

aj istému J. Williamsovi a ten potom vychádzajúc z týchto myšlienok vynašiel metódu triedenia haldou.

To by aj mohlo stačiť. Nie je našou úlohou rozplývať sa nad množstvom triedení, poďme sa radšej pozrieť na vyhodnotenie našich vytipovaných triedení...

3. VYHODNOTENIE

3. 1. Popis programu Prehľad triedenia v Pascale

Skôr než sa vrhneme do vyhodnotenia našich piatich vytipovaných triediacich algoritmov, napíšem zopár slov o elektronickej prílohe tejto práce. Program Prehľad triedenia v Pascale som potom využíval na porovnávanie vyššie zmienených algoritmov. Vzhľadom na to, že tento program dokáže spomalene spustiť súčastne až štyri rôzne algoritmy naraz, ktoré aplikuje na jednu a tú istú zdrojovú postupnosť, som mal tú možnosť tieto triedenia porovnať Hlavne z hľadiska rýchlosti.

Program pozostáva zo súborov: triedeni.exe, tscr.chr, egavga.bgi, default.dat, menu.txt, 1.txt, 2.txt, 3.txt, 4.txt, 5.txt, 6.txt. Bol programovaný v prostredí Borland Pascalu. Totiž ak to má byť Prehľad triedenia v Pascale, tak prečo to nespraviť v jeho tradičnom prostredí, v jeho tradičnej 16 –farebnej grafike a v jeho tradičnom rozlíšení 640 x 480. Ďalej tu bola aj tá vec, že autor bol na tomto prostredí odchovaný a tomu aj najlepšie rozumie (proste zarytý pascalista).

Čo sa týka prostredia, dnes by som ho vytvoril už iné. Rozžiari sa na vás bledomodrá obrazovka, na nej sa zjaví ukazovateľ myši, hore je čosi, ako roletové menu, dole je čosi ako tlačítko alebo okienko Štart. Tým sa spúšťa samotný proces triedenia. Ale musí byť aktívne minimálne jedno triedenie. To si môžete vybrať z horného menu.

Horné menu. Pozostáva zo štyroch hlavných položiek: Zdroj, Vyber, Popis, Koniec. Tak Zdroj, to súvisí so zdrojovou postupnosťou. Môžete sa kedykoľvek vrátiť k defaultnej postupnosti, ktorá je mimochodom uložená v súbore default.dat, môžete si dať vytvoriť úplne náhodné pole, môžete editovať toto pole v takom editovacom okne a voľba reset načíta do všetkých okien, kde prebieha triedenie súčasnú zdrojovú postupnosť. Totiž, proces triedenia môžete, cez spomenuté tlačítko, kedykoľvek pozastaviť a potom spustiť. Cez reset sa pri pozastavení môžete vrátiť k postupnosti pred štartom procesu, ak ste zdrojovú postupnosť už nezmenili. V tom prípade sa všetky triedenia resetujú do novej zdrojovej postupnosti.

Editovanie postupnosti je síce trošku primitívne, ale je. Postupnosť je tvorená celými číslami od 0 do 60 a je v rozsahu 2 až 60 prvkov. V tejto postupnosti sa môžete šípkami hore a dole presúvať, vložiť prvok, vymazať prvok, ďalej máte možnosť prevrátiť postupnosť, utriediť postupnosť (aby ste napríklad videli chovanie algoritmov pre usporiadané pole a opačne usporiadané pole) a náhodne premiešať prvky v postupnosti.

Ďalšia hlavná položka v menu je Vyber. V nej môžete vybrať najviac 4 triedenia z piatich. Totiž, štyri sa mi tak akurát vmestili na obrazovku. Sú dostupné Triedenie priamym výberom, priamym vkladaním, Bublinové, Dobosziewiczovo a Quicksort. Ale to je možné meniť len pred procesom triedenia alebo ak je už proces triedenia skončený. Skončený!!! nie pozastavený. Nemôžete predsa pridať nový algoritmus v čase, keď ostatné triedenia sú už v rôznom štádiu usporiadanosti. Jedine ak prerušiť proces príkazom reset.

Položka Popis. Po výbere jednej jej zložky sa objaví zelené okno a v ňom opis toho, čo ste si vybrali. Sú tu stručné popisy jednotlivých algoritmov a môžete si prečítať aj pár slov o programe.

Položku Koniec myslím nemusím rozpisovať.

K celému hornému menu sa môžete dostať behom procesu len ak proces pozastavíte.

Ako prebieha samotné triedenie? Samotné triedenie prebieha v modrých oknách, ktoré samozrejme nesú označenie triedenia, aby človek vedel o aký algoritmus ide. Čísla sú reprezentované zelenými paličkami rôznej dĺžky, ich výmena je robená aj nakreslením šípky, aby mal pozorovateľ prehľad o tom, ktoré prvky sa posledne vymieňali. Každý algoritmus je špeciálne rozdelený na jednotlivé kroky, pričom jeden krok je buď porovnanie, v ktorom je aspoň jeden porovnávaný prvok prvkom poľa, a priradenie, kde sa aspoň raz objaví prvok poľa. A prvé tri triedenia počítajú kroky tak, že to presne zodpovedá s v minulej kapitole uvedenými vzťahmi. Dobosziewiczovo triedenie a Quicksort boli programované tým istým spôsobom ako tie tri, takže, hoci výsledné hodnoty nemôžem porovnať so žiadnymi vzťahmi, ich pokladám za presné a autentické.

Tieto hodnoty, ktoré nás po celý čas zaujímali, ako počet porovnaní a počet presunov sa vám vypíšu po každom skončenom triedení. Okrem toho tam bude celkový počet krokov, a počet prvkov zdrojovej postupnosti.

Problémy, s ktorými som zápasil, boli spočiatku vymyslieť, ako rozdeliť algoritmus triedenia na jednotlivé kroky (i klasická výmena je záležitosťou troch krokov). Vyriešil som to prostredníctvom stavov. Každé triedenie si pamätá svoj stav a po vykonaní kroku sa dostanú do nejakého nového stavu. Každý stav reprezentoval jeden krok. Potom som s tým už

nemal problémy. Akurát Quicksort, mi narobil nejaké tie vrásky, hlavne preto, lebo som bol napokon nútený použiť nerekurzívnu verziu priamo z Wirthovej knihy (taktiež som ju musel samozrejme upraviť, aby vyhovovala tej multitaskingovej povahe tejto práce), čím mi pribudlo na poli globálnych premenných mnoho nových premenných. Ale funguje to a to je hlavné. Potom tu boli samozrejme problémy estetického razenia, hlavne so zobrazovaním myši, čo som riešil už viac menej vo voľnom čase. Ale chybička sa možno ešte nájde...

3. 2. Samotné vyhodnotenie

Konečne je tu to, čo sme si už dlho sľúbili. Vyhodnotenie našich, trúfam si povedať dobre popísaných triedení. Tak najprv si ukážeme tabuľku, ktorá znázorňuje počínanie jednotlivých triedení na jednoduchom poli zloženom z 13 prvkov. Pôvodná postupnosť bola použitá takáto:

0	60	5	55	10	50	15	45	20	40	25	35	30

Triedili sme to od najväčšieho prvku po najmenší. Z tohto si iste viete predstaviť usporiadanú postupnosť ako aj obrátene usporiadanú, takže tie vypisovať nebudem. Na získanie v tabuľke uvedených hodnôt som použil v predchádzajúcej kapitole popisovaný program. Ak ešte vždy niekto neverí jeho výsledkom, pre prvé tri triedenia si sami môžeme overiť hodnoty pre tie extrémne prípady podľa už uvedených vzťahov. Aby boli po ruke, uvedieme si ich ešte raz:

Triedenie priamym výberom:

$$M_{min} = 3 (n - 1)$$
$$C_{min} = 1/2 (n^2 - n)$$
$$M_{max} = trunc(n^2 /4) + 3 (n - 1)$$
$$C_{max} = 1/2 (n^2 - n)$$

Triedenie priamym vkladaním:

$$M_{min} = 2 (n - 1)$$

$$C_{min} = n - 1$$

$$M_{max} = 1/2 (n^2 + 3 n - 4)$$

$$C_{max} = 1/2 (n^2 + n) - 1$$

Bublinové triedenie:

$$M_{min} = 0$$

$$C_{min} = 1/2 (n^2 - n)$$

$$M_{max} = 3/2 (n^2 - n)$$

$$C_{max} = 1/2 (n^2 - n)$$

Pre tých dôverčivejších len uvádzam, že sám som to prepočítal a presne to sedí. Čiže aj ostatné výsledky získané týmto programom sú presné. A tu je už tabuľka:

Postupnosť	Usporiadaná	Pôvodná	Opačne usp.
Priamy výber			
Porovnaní:	36	59	78
Presunov:	78	78	78
Celkovo krokov:	**114**	**137**	**156**
Priame vkladanie			
Porovnaní:	24	66	102
Presunov:	12	54	90
Celkovo krokov:	**36**	**120**	**192**
Bublinové			
Porovnaní:	0	126	234
Presunov:	78	78	78
Celkovo krokov:	**78**	**204**	**312**
Dobosziewiczovo			
Porovnaní:	0	54	42
Presunov:	46	58	58
Celkovo krokov:	**46**	**112**	**100**
Quicksort			
Porovnaní:	28	55	46
Presunov:	42	60	42
Celkovo krokov:	**70**	**115**	**88**

Pohľad na tabuľku nám už môže prezradiť o daných triedeniach rôzne veci. Napriek tomu si to ešte zhrňme.

Triedenie priamym výberom:

Veľmi používané v amatérskejších programoch, pre jeho prehľadnosť a jednoduchosť. Z tabuľky vidíme, že na rozdiel od mnohých iných triedení nerobí až taký časový rozdiel medzi extrémnymi prípadmi. Hodí sa preto na triedenia polí, o ktorých usporiadanosti nemáme ani najmenšiu predstavu.

Triedenie priamym vkladaním:

V tomto triedení už na usporiadanosti veľmi záleží. Ak sa pozriete na počet krokov pre usporiadanú zdrojovú postupnosť, ten sa od predchádzajúceho triedenia líši. A nie o málo. Preto toto triedenie aj najlepšie využijete, ak ho použijete na triedenie poľa, ktoré už je takmer utriedené.

Napríklad, máte pole, do ktorého ukladáte ďalšie údaje. Triediť pole po každom pridaní jediného prvku, môže zaberať aj veľa času a ak sa to nevyžaduje a chceme vkladať do poľa viac údajov, tie môžeme vkladať iba odhadom na miesta kam patria a pole utriedime až po určitom množstve zadaných údajov alebo na konci zadávania. Tu sa dá použiť triedenia priamym vkladaním, ktorá je v tomto prípade dosť rýchla (lebo pole je takmer usporiadané a ak požijeme dobrý odhad pri vsúvaní ďalších prvkov, aj také ostane) a pole budeme mať vždy (aj behom zadávania tých údajov) veľmi dobre usporiadané.

Bublinové triedenie:

Ruky preč od neho. Je to najpomalšie triedenie, aké sme si spomenuli. Okrem príťažlivého názvu a zaujímavému princípu „vybublávania“ prvkov na svoje miesta, nám naozaj nemôže viac toho ponúknuť. Je pravda, že aj tu platí, že si triedenie vedie lepšie v takmer usporiadanej postupnosti, ale keď máme k dispozícii triedenie priamym vkladaním, ktoré má ďaleko lepšie charakteristiky, prečo nepoužiť práve to.

Možno by sa našlo uplatnenie pre vylepšené bublinové triedenie, alebo triedenie pretriasaním, ale to by muselo byť to pole už veľmi dobre usporiadané. A ak nie, tak prečo si komplikovať život písaním zložitejšieho programu, keď triedenie priamym vkladaním je jednoduchšie. Ruky preč!!!

Dobosziewiczovo triedenie:

Dobosziewiczovo triedenie už patrí tak trochu do inej triedy týchto algoritmov, hlavne preto, že najhorší prípad sa nedá presne určiť. Ako vidíme, pre obrátenú postupnosť potreboval menej krokov ako pre pôvodnú (ktorá by mohla reprezentovať nejaké priemerne rozhádzané pole). Na druhej strane, najoptimálnejší prípad má rovnaký ako ostatné triedenia. Okrem iného už neprehľadáva pole lineárne, tak sa dostalo toto triedenie do skupiny tých, síce trochu komplikovanejších, ale radikálne rýchlejších algoritmov. A ak ho porovnáme s algoritmom quicksort, zistíme, že v tomto prípade bolo dokonca lepšie (okrem obrátenej postupnosti).

Čo ešte dodať? Pre veľké polia dát, a prostredia nedokážuce spracúvať údaje rekurzívne je to, myslím, správna voľba. Kľudne ho môžeme využívať aj v profesionálnych aplikáciách.

Quicksort:

Naše známe a populárne rýchle triedenie má aj svoje slabiny. Samozrejme je priemerne najrýchlejším algoritmom. A úplne ignoruje usporiadanosť prvkov v zdrojovej postupnosti. Jeho algoritmus je však trochu zložitejší a tak sa by bolo dobré ho využívať skôr pri práci s poliami o veľkom množstve údajov. Veď, ako sa dá všimnúť, rozdiel medzi týmito rýchlymi triedeniami (tam spadá aj Dobosziewiczovo) a „pomalými“ je v tomto prípade s výnimkou opačného usporiadania dosť malý.

Každé, alebo aspoň takmer každé triedenie sa dá napísať rekurzívnou formou. V prípade quicksortu je to však jediný rozumnejší spôsob zápisu algoritmu. Ak však rekurziu nemáte príliš v láske (ani ja ju v programovaní akosi nepoužívam), kľudne sa vráťte k Dobosziewiczovi. Quicksort sa dá síce napísať aj nerekurzívnou formou, ale algoritmus sa tým o dosť skomplikuje, pribudnú aj ďalšie premenné, jedna dokonca typu pole a pokiaľ sa jedná o veľké množstvo údajov, tak to je tiež nezanedbateľne rozsiahle.

Quicksort alebo Dobosziewicz? Ktorý je efektívnejší? Ak by som mal subjektívne vyjadriť svoj názor, po všetkých tých príkladoch, ktoré som v programe Prehľad triedenia v Pascale robil, myslím že Rýchle triedenie. Keď už sa tak volá. Ale ten Dobosziewicz nie je trochu viac sympatický?...

ZÁVER

Uff... Dúfam, že sa mi podarilo vnútiť začínajúcim programátorom to, že bez triedenia sa asi nezaobídu, ostatní sa možno poučili, ak už všetky triediace algoritmy nepoznali a aspoň niektorým táto práca vnukla nejaký ten nápad a vymyslia si vlastný triediaci algoritmus. Možností je veľa, špeciálnych prípadov, ktoré nastávajú v praxi života tiež, len v nich treba nájsť nejaké skryté princípy a nové, špecializované super rýchle triedenie môže byť na svete.

A koho takto neosvieti, nech nezúfa. Iste sa dá z tejto práce vybrať jedno z triedení, ktoré sa práve hodí. Tie triedenia sú dobré. Na mnohé by ste iste prišli sami, mnohé sú trošku zložitejšie, o tých sa zasa môžete dozvedieť z rôznych zdrojov, napr. z tohto tu, čo držíte práve v ruke. Ale zapamätajte si jedno. Bez triedenia to nejde.

Kdesi za týmto by mala byť ešte príloha, v ktorej nájdete jednotlivé procedúry triedení, ktoré sme si dôkladnejšie popísali. Samozrejme v jazyku Pascal. Tie procedúry triedia pole od najväčšej hodnoty po najnižšiu. Komu to tak nevyhovuje, v niektorých podmienkach treba zmeniť znamienka je väčšie za je menšie alebo naopak. Ak neviete ktoré, ešte raz si prečítajte kapitolu o príslušnom triedení a ak ani to nepomôže, naučte sa Pascal.

Mnohé informácie, ako aj samotné programy (len trochu som ich pozmenil) som odčerpal z knihy pána Wirtha, Algoritmy a štruktúry údajov. Čo sa týka Dobosziewiczovho triedenia, ktoré v zmienenej publikácii nebolo, musím sa poďakovať pani RNDr. Marte Miklášovej, za jej ochotu mi to aj dva krát vysvetliť.

Myslím, že viac netreba, čo dodať, stačí už len vybrať ten správny triediaci algoritmus a ideme na to...

Zoznam použitej literatúry:

KOSTOLANSKÝ I., II.: Spracovanie údajov počítačom (Základný kurz), VEDA, vydavateľstvo SAV, Bratislava, 1999, 162 str.

MIKLÁŠOVÁ M., VADKERTI P., ŠRUBÁROVÁ R.: Základy počítačov a programovania, VEDA, vydavateľstvo SAV, Bratislava, 1999, 96 str.

MOLNÁR Ľ.: Programovanie pre 3. ročník gymnázia. Jazyk Pascal, Slovenské pedagogické nakladateľstvo, Bratislava, 1986, 146 str.

WIRTH N.: Algoritmy a štruktúra údajov, vydavateľstvo Alfa, Bratislava, 1988, 488 str.

PRÍLOHA 1: Triedenia v pascale

Uvedené procedúry triedia číselné pole `a[index]` (nech je typu real) od najväčšej hodnoty po najnižšiu. Ďalšie premenné `i, j` budú celočíselné a budú sa používať v cykloch. `n` obsahuje celkový počet prvkov v poli a indexovanie poľa prebieha od `1` po `n`.

Triedenie priamym výberom:

```
Procedure priamyvyber;
var i,j,k:integer;
    x:real;
begin
  for i:=1 to n-1 do begin
    k:=i;x:=a[i];
    for j:=i+1 do
     if a[j]>x then begin
       k:=j;x:=a[j];
     end;
    a[k]:=a[i];a[i]:=x;
  end;
end;
```

Triedenie priamym vkladaním:

```
Procedure priamevkladanie;
var i,j:integer;
    x:real;
begin
  for i:=2 to n do begin
    x:=a[i];a[0]:=x;j:=i-1;
    while x>a[j] do begin
      a[j+1]:=a[j];j:=j-1;
    end;
    a[j+1]:=x;
  end;
end;
```

Bublinové triedenie:

```
Procedure bublinovetriedenie;
var si,j:integer;
    x:real;
```

```
begin
  for i:=2 to n do begin
    for j:=n downto i do
     if a[j-1]<a[j] then begin
       x:=a[j-1];
       a[j-1]:=a[j];
       a[j]:=x;
     end;
  end;
end;
```

Dobosziewiczovo triedenie:

```
Procedure Dobosziewicz;
var i,k:integer;
    x:real;
    vymena:boolean;
begin
  k:=n;
  repeat
    vymena:=false;
    k:=trunc(k*2/3);
    if k=0 then k:=1;
    for i:=1 to n-k+1 do
     if a[i]<a[i+k] then begin
       x:=a[i+k];
       a[i+k]:=a[i];
       a[i]:=x;
       vymena:=true;
     end;
  until (k=1)and(not vymena);
end;
```

Quicksort:

```
procedure QuickSort;

procedure Sort(l, r: Integer);
var
  i,j,x,y: integer;
begin
  i:=l;j:=r;x:=a[(l+r) div 2];
  repeat
    while a[i]>x do i:=i+1;
    while x>a[j] do j:=j-1;
    if i<=j then
    begin
```

```
      y:=a[i];a[i]:=a[j];a[j]:=y;
      i:=i+1;j:=j-1;
    end;
  until i>j;
  if l<j then Sort(l,j);
  if i<r then Sort(i,r);
end;

begin {QuickSort};
  Sort(1,n);
end;
```

PRÍLOHA 2: Disketa