Porovnanie exekučnej efektívnosti referenčného a presúvacieho usporiadavania štruktúrovaných dát v metódach C# aplikácie

Igor Košťál[1]

Abstrakt
Každá metóda s implementovaným ľubovoľným usporiadavacím algoritmom počas realizácie usporiadavania dát uložených napr. v 2-rozmernom neusporiadanom poli musí manipulovať s týmito neusporiadanými dátami. Metóda môže počas usporiadavania presúvať riadky takéhoto poľa, čím realizuje presúvacie usporiadavanie, alebo môže presúvať len referencie na tieto usporiadavané riadky, čím realizuje referenčné usporiadavanie, kedy metóda nepresúva riadky poľa, vôbec nimi nemanipuluje. Spôsob narábania s usporiadavanými dátami môže podstatne ovplyvniť exekučnú a pamäťovú efektívnosť usporiadavacej metódy. Pomocou metód nami vytvorenej C# aplikácie sme skúmali, ktorý z uvedených spôsobov narábania s uporiadavanými dátami je exekučne efektívnejší. Článok obsahuje porovnania exekučnej efektívnosti metód našej C# aplikácie s implementovaným presúvacím a referenčným usporiadavaním, vyhodnotenie výsledkov týchto porovnaní a tiež krátky opis dôležitých usporiadavacích metód tejto C# aplikácie.

Kľúčové slová
presúvacie usporiadavanie, referenčné usporiadavanie, algoritmus Quick Sort, C# aplikácia, rekurzívna metóda

Abstract
Each method with an implemented arbitrary sorting algorithm during the realization of sorting of data stored, e.g., in a 2-dimensional unordered array must manipulate with this unordered data. During sorting, the method can move the rows of such array, it carries out a move sorting, or it can only move references to these rows, it carries out a reference sorting, when the method does not move the rows of the array, it does not manipulate with them at all. The way of dealing with data that are sorting can significantly affect the execution and memory efficiency of the sorting method. Using the methods of the C# application created by us, we have examined which of the mentioned ways of dealing with sorting data is more execution efficient. The paper contains comparisons of the execution efficiency of the methods of our C# application with the implemented move and reference sorting, the evaluation of the results of these comparisons, and also a brief description of the important sorting methods of this C# application.

Key words
move sorting, reference sorting, quicksort algorithm, C# application, recursive method

JEL classification
C88

1 Úvod

Ako sme uviedli vyššie, z pohľadu exekučnej efektívnosti je veľmi dôležité, ako usporiadavacia metóda C# aplikácie s implementovaným ľubovoľným usporiadavacím

[1] Ekonomická univerzita v Bratislave, Fakulta hospodárskej informatiky, Katedra aplikovanej informatiky, Dolnozemská cesta 1/b, 852 35 Bratislava, e-mail: igor.kostal@euba.sk.

algoritmom narába s usporiadavanými dátami uloženými napr. v 2-rozmernom poli. Metóda môže počas usporiadavania presúvať alebo vymieňať celé riadky tohto poľa, ktoré môžu obsahovať aj rozsiahle štruktúrované dáta, vtedy realizuje presúvacie usporiadavanie. Alebo sa metóda môže vyhnúť presúvaniu alebo vymieňaniu často rozsiahlych riadkov takéhoto poľa. Vtedy bude usporiadavať len referencie na tieto riadky, ktorými nebude vôbec hýbať. Vtedy metóda realizuje referenčné usporiadavanie. Alebo metóda môže usporiadavať len indexy riadkov tohto poľa, čím realizuje indexové usporiadavanie, ktoré taktiež nehýbe týmito riadkami. Alebo môže metóda realizovať uspriadavanie na mieste, ktoré minimalizuje manipulovanie s celými riadkami usporiadavaného poľa.

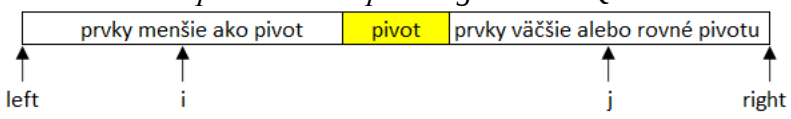
Nás zaujímalo, ktorý z vybraných spôsobov manipulácie usporiadavacej metódy s usporiadavanými dátami je exekučne efektívnejší, či je to presúvacie alebo referenčné usporiadavanie. Predpokladáme, že exekučne efektívnejšie by malo byť referenčné usporiadavanie. Pre účely tohto porovnania sme vytvorili C# aplikáciu, ktorá obsahuje metódy s implementovaným usporiadavacím algoritmom Quick Sort (rýchle usporiadavanie), pričom jedna z týchto metód usporiadavané dáta uložené v riadkoch 2-rozmerného poľa presúva alebo vymieňa, čím realizuje presúvacie usporiadavanie, iná metóda usporiadava len referencie ukazujúce na tieto riadky, čím realizuje referenčné usporiadavanie. C# aplikácia dokáže zmerať exekučné časy vykonávania každej z týchto metód. Pomocou ich porovnania sme skúmali, ktorý zo spôsobov narábania s usporiadavanými dátami je exekučne efektívnejší a ako ovplyvňuje množstvo usporiadavaných dát túto efektívnosť. Porovnaním týchto exekučných časov a jeho vyhodnotením sa zaoberáme v kapitole Experiment a jeho výsledky. V nasledujúcich kapitolách sa krátko zaoberáme usporiadavacím algoritmom Quick Sort, presúvacím a referenčným usporiadavaním, ktoré implementujú usporiadavacie metódy našej C# aplikácie a samotnými týmito metódami a ich dátovými štruktúrami, ktoré usporiadavajú. V Závere je zhrnutie dosiahnutých výsledkov nášho skúmania.

2 Usporiadavací algoritmus Quick Sort, jeho krátky opis

Na usporidavanie dát je možné použiť viacero usporiadavacích algoritmov, ako sú napr. výberové uspriadavanie (Selection Sort), vkladacie usporiadavanie (Insertion Sort), bublinkové usporiadavanie (Bubble Sort) a ďalšie. Naša C# aplikácia používa v svojich usporiadavacích metódach usporiadavací algoritmus rýchle usporiadavanie (Quick Sort), preto ho krátko opíšeme.

Usporiadavací algoritmus Quick Sort usporiadava dáta metódou rozdeľuj a panuj (divide and conquer) (Sedgewick, 1998). Kľúčovou činnosťou tohto algoritmu je rozdelenie usporiadavaného poľa na dve časti a preusporiadanie prvkov tohto poľa tak, že vľavo od pivota (vodiaci prvok rozdeľovania a usporiadavania prvkov poľa) budú prvky menšie ako pivot a vpravo od pivota budú prvky väčšie alebo rovné pivotu. Keď algoritmus takéto preusporiadanie vykoná, potom je rekurzívne aplikovaný na ľavú a pravú časť poľa (Sedgewick, 2011). Po dokončení týchto rekurzívnych volaní usporiadavacích metód s implementovaným algoritmom Quick Sort a v nich vnorených prípadných ďalších rekurzívnych volaní týchto metód, je usporiadavanie dokončené.

Obr. 1: Usporiadavané pole algoritmom Quick Sort



Zdroj: Vlastné spracovanie

3 Presúvacie a referenčné usporiadavanie, ich krátky opis

Usporiadavacie metódy našej C# aplikácie používajú dva spôsoby manipulovania s usporiadavanými dátami, presúvacie a referenčné usporiadavanie, preto ich krátko opíšeme.

Presúvacie usporiadavanie, ako vyplýva z jeho názvu, skutočne presúva alebo vymieňa usporiadavané dáta. Metódy našej C# aplikácie usporiadavajú dáta uložené v 2-rozmernom poli objektov, ktoré obsahujú štruktúrované dáta v ich inštančných premenných. Každý z týchto objektov je uložený v jednom riadku tohto 2-rozmerného poľa a obsahuje relatívne rozsiahle štruktúrované dáta (obr. 7). Preto aj exekučná réžia presúvania alebo výmeny týchto riadkov nie je zanedbateľná a ovplyvňuje, hlavne pri veľkom množstve usporiadavaných riadkov poľa, napr. 1000 a viac, exekučný čas samotného usporiadavania.

Referenčné usporiadavanie usporiadava referencie na jednotlivé riadky tohto 2-rozmerného poľa objektov. Samotnými riadkami poľa nehýbe. Usporiadavacia metóda našej C# aplikácie preskupí referencie tak, že zobrazovacia metóda pri sekvenčnom prístupe k týmto už usporiadaným referenciám zobrazí riadky poľa usporiadané podľa požadovaného kritéria. Usporiadavacia metóda počas usporiadavania tohto 2- rozmerného poľa objektov presúva alebo vymieňa len referencie v poli referencií (obr. 7), samotnými riadkami poľa nehýbe. Aj presúvanie alebo vymieňanie referencií v ich poli má nejakú exekučnú réžiu. Otázkou je exekučná réžia ktorého spôsobu manipulácie s dátami, presúvacieho alebo referenčného usporiadavania, je menšia. V kapitole Experiment a jeho výsledky sa snažíme na túto otázku odpovedať.

4 C# aplikácia s implementovaným presúvacím a referenčným usporiadavaním v jej metódach používajúcich algoritmus Quick Sort

Naša C# aplikácia, používajúca presúvacie a referenčné usporiadavanie v metódach implementujúcich algoritmus Quick Sort pri usporiadavaní 2-rozmerných polí objektov, obsahujúcich vo svojich inštančných premenných štruktúrované dáta študentov, bola vyvinutá v jazyku C# ako konzolová aplikácia vo vývojovom prostredí Microsoft Visual Studio 2019. Jej zdrojový kód uložený v zdrojom súbore *Program.cs* obsahuje nasledujúce triedy:

- *Student* - objekt tejto triedy obsahuje vo svojich inštančných premenných *firstname, surname, birthday* (objekt triedy *Date*), *residence, isic, points_acc, income, aver_grade* a *distance_resid* dáta jedného študenta, jeho meno, priezvisko, dátum narodenia, bydlisko, číslo ISIC karty, body na ubytovanie, príjem v dolároch, priemernú známku za doterajšie štúdium a vzdialenosť bydliska od školy. Jeden objekt triedy *Student* je uložený v jednom riadku dvoch 2-rozmerných polí objektov triedy *Student* (obr. 7). Tieto polia inštančné metódy C# aplikácie usporiadavajú.
- *Date* - objekt tejto triedy obsahuje vo svojich inštančných premenných *d*, *m* a *y* deň, mesiac a rok narodenia jedného študenta. Objekt *birthday* tejto triedy je vložený ako inštančná premenná do objektu triedy *Student* (obr. 7), ktorý obsahuje dáta jedného študenta.
- *IndexS_QuickS* - objekt tejto triedy obsahuje verejné inštančné premenné *refArrSts* a *ref2dArrSts*, čo sú dve 2-rozmerné polia objektov triedy *Student* (obr. 7) obsahujúce dáta študentov, ktoré usporiadavacie inštančné metódy C# aplikácie usporiadavajú a privátnu inštančnú premennú *p2d*, čo je 2-rozmerné pole rozmeru 1 x 1 (1 jednoprvkový riadok a 1 jednoprvkový stĺpec) slúžiace na ukladanie pivota v metódach *QuickS_movingSort2d* a *QuickS_referSort2d*. Členmi tejto triedy sú usporiadavacie metódy *QuickS_movingSort* (metóda usporiadava 2-rozmerné pole objektov triedy *Student refArrSts* (obr. 7) obsahujúce 1-rozmerné pole referencií), *QuickS_movingSort2d* a *QuickS_referSort2d* (metódy usporiadavajú 2-rozmerné pole objektov triedy *Student*

ref2dArrSt (obr. 7) obsahujúce 2-rozmerné pole referencií) s implementovaným algoritmom Quick Sort. Trieda tiež obsahuje koštruktor, členské metódy *CreateStudentsArray*, *DisplaySortedSts_movingSort*, *Display_ref2dArrSts*, slúžiace na načítanie dát študentov z dátového prúdu spojeného so vstupným diskovým súborom do 2 polí *refArrSts* a *ref2dArrSts* a zobrazenie aktuálneho stavu týchto polí a ďalšie metódy.

Kľúčové členské metódy triedy *IndexS_QuickS* s implementovaným algoritmom Quick Sort a používajúce presúvacie a referenčné usporiadavanie sú:

- ***QuickS_movingSort*** - rekurzívna metóda usporiadava 2-rozmerné pole objektov triedy *Student* a používa **presúvacie usporiadavanie**. Metóda je zavolaná ako inštančná metóda objektu *sort_obj* triedy *IndexS_QuickS* v statickej metóde *Main* našej C# aplikácie nasledujúcim príkazom

sort_obj.QuickS_movingSort(sort_obj.refArrSts, 0, sort_obj.students_count - 1, criter);

V tomto volaní inštančná metóda *QuickS_movingSort* usporiadava 2-rozmerné pole *sort_obj.refArrSts* objektov triedy *Student* obsahujúce ***1-rozmerné pole referencií*** (obr. 7).

- ***QuickS_movingSort2d*** - rekurzívna metóda usporiadava 2-rozmerné pole objektov triedy *Student* a používa **presúvacie usporiadavanie**. Metóda je zavolaná ako inštančná metóda objektu *sort_obj* triedy *IndexS_QuickS* v statickej metóde *Main* našej C# aplikácie nasledujúcim príkazom

sort_obj.QuickS_movingSort2d(sort_obj.ref2dArrSts, 0, sort_obj.students_count - 1, criter);

V tomto volaní inštančná metóda *QuickS_movingSort2d* usporiadava 2-rozmerné pole *sort_obj.ref2dArrSts* objektov triedy *Student* obsahujúce ***2-rozmerné pole referencií*** (obr. 7).

- ***QuickS_referSort2d*** - rekurzívna metóda usporiadava 2-rozmerné pole objektov triedy *Student* a používa **referenčné usporiadavanie**. Metóda je zavolaná ako inštančná metóda objektu *sort_obj* triedy *IndexS_QuickS* v statickej metóde *Main* našej C# aplikácie nasledujúcim príkazom

sort_obj.QuickS_referSort2d(sort_obj.ref2dArrSts, 0, sort_obj.students_count - 1, criter);

V tomto volaní inštančná metóda *QuickS_movingSort2d* usporiadava 2-rozmerné pole *sort_obj.ref2dArrSts* objektov triedy *Student* obsahujúce ***2-rozmerné pole referencií*** (obr. 7).

Inštančná metóda *QuickS_referSort2d* vykonávajúca referenčné usporiadavanie vymieňa len referencie v ľavom stĺpci 2-rozmerného poľa referencií *ref2dArrSt* vždy spolu s ich príslušnými referenciami *ref2dArrSts[i][0]* z nultého stĺpca tohto 2-rozmerného poľa referencií (obr. 7), na ktoré vymieňané referencie odkazujú. Táto metóda nehýbe 2-rozmerným poľom objektov triedy *Student*.

Jazyk C# neumožňuje syntakticky oddeliť referenčnú premennú od nereferenčnej premennej neobsahujúcej referenciu. Okrem toho, referencie uložené v referenčných premenných sú v korektných príkazoch jazyka C# automaticky dereferencované na entitu, na ktorú referencia odkazuje. Z tohto dôvodu sme museli vytvoriť nultý stĺpec *ref2dArrSts[i][0]* 2-rozmerného poľa referencií (obr. 7), aby metóda *QuickS_referSort2d* vykonávala len usporiadavanie referencií, tzv. referenčné usporiadavanie, čiže aby pri výmene referencií v zavolanej privátnej metóde *Exch2d*

```
private void Exch2d(ref Student[][] dataX, int ix, int jx) {
    // metoda vymiena LEN referencie v 1-rozmernom poli referencii 'dataX[]', ktore odkazuju na
    // referencie na objekty triedy 'Student' ulozene v 0-tych prvkoch dalsieho pola referencii 'dataX[i][0]'
    Student[] t;  t = new Student[1];
    t[0] = new Student();  t = dataX[ix];
    dataX[ix] = dataX[jx];
    dataX[jx] = t;  }
```

vykonala skutočne len výmenu referencií, nie výmenu objektov v 2-rozmernom poli objektov triedy *Student*. Ak by neexistoval nultý stĺpec *ref2dArrSts[i][0]* 2-rozmerného poľa referencií, tak by vzniklo jednorozmerné pole referencií, ktoré, ak by použila metóda *Exch2d* ako svoj prvý skutočný parameter, vtedy by jej príkazy používajúce prvky poľa *dataX[ix]* a *dataX[jx]* spôsobili dereferencovanie referencií uložených v týchto prvkoch poľa na objekty triedy *Student* a tie by metóda *Exch2d* vymenila v poli týchto objektov, čo by zodpovedalo operácii presúvacieho usporiadavania, ale nezodpovedalo by to operácii referenčného usporiadavania.

Metóda *QuickS_movingSort* realizuje presúvacie usporiadavanie vykonaním výmeny objektov v 2-rozmernom poli objektov triedy *Student*, pretože referencie *data[i]* a *data[j]* v 1-rozmernom poli referencií *refArrSts* (obr. 7) sa v jej príkazoch

```
Student t = new Student();
t = data[i];
data[i] = data[j];
data[j] = t;
```

dereferencujú na objekty triedy *Student* a tie sa vymenia v tomto 2-rozmernom poli objektov triedy *Student*. Takáto operácia zodpovedá presúvaciemu usporiadavaniu.

Fungovanie metód *QuickS_movingSort* (obr. 2), *QuickS_movingSort2d* (obr. 3) a *QuickS_referSort2d* (obr. 4) je tiež zrejmé z ich zdrojových kódov a komentárov v nich.

Obr. 2: Zdrojový kód členskej metódy 'QuickS_movingSort'

```
public void QuickS_movingSort(Student[] data, int left, int right, int criterx) {
 if (left < right) {
  int i = left, j = right;
  Student p = data[(left + right) / 2];  // urcenie pivota, riadku pola (vodiaceho prvku usporiadavania)
  do {
      while (Compare_sts(data[i], p, criterx) > 0) i++; // hladanie indexu 'i' prvku v lavej casti pola 'data',
      while (Compare_sts(p, data[j], criterx) > 0) j--; // ktory bude vymeneny s prvkom s indexom 'j',
                                                        // najdenym v tomto cykle, v pravej casti tohto pola
      if (i >= j) break;
      // nasleduj. 4 prikazy vymenia 2 riadky pola objektov triedy 'Student' 'data', cize presuvaju data
      Student t = new Student();
      t = data[i];  data[i] = data[j];  data[j] = t;
  } while (true);
  QuickS_movingSort(data, left, j, criterx); // rekurzivne volanie metody pre lavu cast pola 'data'
  QuickS_movingSort(data, j + 1, right, criterx); // rekurzivne volanie metody pre pravu cast pola 'data'
 } }
```

Zdroj: Vlastné spracovanie

Obr. 3: Zdrojový kód členskej metódy 'QuickS_movingSort2d'

```
public void QuickS_movingSort2d(Student[][] data, int left, int right, int criterx) {
 if (left < right) {
  int i = left, j = right;

  p2d[0][0] = data[(left + right) / 2][0];  // urcenie pivota, riadku pola (vodiaceho prvku usporiadavania)

  do {
      while (Compare_sts(data[i][0], p2d[0][0], criterx) > 0) i++; // hladanie indexu 'i' prvku v lavej casti
                                                                   // pola 'data',
      while (Compare_sts(p2d[0][0], data[j][0], criterx) > 0) j--; // ktory bude vymeneny s prvkom
                                                // s  indexom 'j', najdenym v tomto cykle, v pravej casti tohto pola
      if (i >= j) break;
      // zavolana metoda vymeni 2 riadky pola objektov triedy 'Student' 'data', cize presuva data
      Exch2d_moving(ref data, i, j);
  } while (true);
  QuickS_movingSort2d(data, left, j, criterx); // rekurzivne volanie metody pre lavu cast pola 'data'
  QuickS_movingSort2d(data, j + 1, right, criterx); // rekurzivne volan. metody pre pravu cast pola 'data'
 }
}
```

Zdroj: Vlastné spracovanie

Obr. 4: Zdrojový kód členskej metódy 'QuickS_referSort2d'

```
public void QuickS_referSort2d(Student[][] data, int left, int right, int criterx) {
 if (left < right) {
  int i = left, j = right;

  p2d[0][0] = data[(left + right) / 2][0]; // urcenie pivota, referencie (vodiaceho prvku usporiadavania)
  do {
      while (Compare_sts(data[i][0], p2d[0][0], criterx) > 0) i++; // hladanie indexu 'i' prvku v lavej casti
                                                                   // pola 'data',
      while (Compare_sts(p2d[0][0], data[j][0], criterx) > 0) j--; // ktory bude vymeneny s prvkom
                                                // s  indexom 'j', najdenym v tomto cykle, v pravej casti tohto pola
      if (i >= j) break;
      // zavolana metoda vymeni 2 referencie v poli referencii na objekty triedy 'Student' 'data', cize
      // vymiena referencie, data nepresuva
      Exch2d(ref data, i, j);
  } while (true);
  QuickS_referSort2d(data, left, j, criterx); // rekurzivne volanie metody pre lavu cast pola 'data'
  QuickS_referSort2d(data, j + 1, right, criterx); // rekurzivne volanie metody pre pravu cast pola 'data'
 }
}
```

Zdroj: Vlastné spracovanie

Všetky tieto členské metódy volajú pre potreby v nich implementovaného algoritmu Quick Sort porovnávaciu privátnu členskú metódu *Compare_sts*, ktorá porovnáva dva prvky 2-rozmerného poľa objektov triedy *Student* podľa kritéria ktoré dostane do tretieho parametra *criterX*. Na základe výsledkov porovnania metóda určí poradie v akom budú títo 2 študenti usporiadaní volajúcou usporiadavacou metódou.

Metóda *Compare_sts* dokáže porovnávať a určovať poradie študentov podľa viacerých kritérií, podľa *bodov na ubytovanie*, *vzdialenosti ich bydliska od univerzity*, *príjmu študenta* a *ich priemernej známky za doterajšie štúdium*.

Ak metóda dostane do tretieho parametra *criterX* napr. hodnotu 1, tak vtedy bude metóda porovnávať a tým určovať poradie dvoch prvkov 2-rozmerného poľa objektov triedy *Student*, čiže 2 študentov, podľa bodov na ubytovanie od väčšieho po menší počet bodov. Ak majú títo 2 študenti rovnaké body na ubytovanie, tak potom sa budú usporiadavať podľa ich priemernej známky za doterajšie štúdium od menšej po väčšiu známku. Ak majú aj priemernú známku

rovnakú, tak potom sa budú usporiadavať podľa ich príjmu od menšieho po väčší príjem. Ak majú aj príjem rovnaký, tak potom sa budú títo 2 študenti usporiadavať podľa vzdialenosti ich bydliska od univerzity od väčšej vzdialenosti po menšiu. Ak majú porovnávaní študenti všetky uvedené údaje rovnaké, tak potom sa budú usporiadavať podľa ich dátumu narodenia od mladšieho po staršieho študenta. Ak majú aj dátum narodenia rovnaký, tak potom sa budú usporiadavať abecedne podľa ich priezviska. Ak majú aj priezvisko rovnaké, tak potom budú usporiadaní abecedne podľa ich mena. Ak majú aj meno rovnaké, tak potom budú usporiadaní podľa veľkosti čísla ich ISIC karty, od menšieho čísla po väčšie.

Podobnú situáciu pri usporiadavaní študentov s niektorými rovnakými údajmi ilustruje príklad výstupu programu na obr. 5.

Obr. 5: Príklad výstupu C# aplikácie (vstupy vložené používateľom sú zobrazené tučným písmom)

```
  Choose input file: students11c2sk.txt (insert: 1), students10c2en.txt (2), students100c2en.txt (3),
students200c2en.txt (4), students300c2en.txt (5), students400c2en.txt (6), students500c2en.txt (7),
students600c2en.txt (8), students700c2en.txt (9), students800c2en.txt (10), students900c2en.txt (11),
students1000c2en.txt (12), students1100c2en.txt (13), students1200c2en.txt (14), students1300c2en.txt
(15), students1400c2en.txt (16), students1500c2en.txt (17), students1600c2en.txt (18),
students1700c2en.txt (19), students1800c2en.txt (20), students1900c2en.txt (21), students2000c2en.txt
(22)
  2
  The student's data loaded from the 'students10c2en.txt' file.

  Choose sorting criterion of students:
  sorting according to:
  -points for accommodation (insert: 1),
  -distance of their residence from the university (insert: 2),
  -income of students (insert: 3),
  -average grade (insert: 4)
  1

  The students will be sorted according to 'points for accommodation'.

        *** students stored in the 'Student[] refArrSts' array ***

  students sorted by the 'QuickS_movingSort' method
  (moving sort in Quick Sort, Execution time: 00:00:00.0007735, 0,7735 ms)

   1  34122788215722749  Rodenborch Royce     (94) 403  1,4  2000-11-14  South-Bend  (410)
   2  34162078930108726  Wilcox Ab            (84) 628  1,41  2002-09-12  Kirkton  (475)
   3  32572965109821616  Casiero Jenny        (78) 511  1,54  2000-12-04  Waterbury  (216)
   4  32411560480235319  Casiero Ezra         (78) 511  1,54  2000-12-03  Monticello  (84)
   5  32521773578114203  Garton Magda         (78) 431  2,11  2001-12-19  Washington  (462)
   6  34235504039842896  Porch Lorry          (72) 164  4,2  2002-08-13  Eaton  (577)
   7  33577388665139502  Hair Hortense        (67) 311  4,41  1999-09-04  Orlando  (327)
   8  34350655375829554  Solloway Tabbatha    (63) 406  3,56  2000-10-13  Boulder  (382)
   9  32857203688836212  Planque Ellswerth    (59) 230  2,43  2002-09-19  Adelaide-Mail-Centre  (44)
  10  36103100162715669  Jerdein Aurie        (37) 319  4,59  2003-12-06  Sacramento  (204)

  ------------------------------------------------------------------------------
        *** students stored in the 'Student[][] ref2dArrSts' array ***

  students sorted by the 'QuickS_movingSort2d' method
  (moving sort in Quick Sort, Execution time: 00:00:00.0005476, 0,5476 ms)
   1  34122788215722749  Rodenborch Royce     (94) 403  1,4  2000-11-14  South-Bend  (410)
   2  34162078930108726  Wilcox Ab            (84) 628  1,41  2002-09-12  Kirkton  (475)
   3  32572965109821616  Casiero Jenny        (78) 511  1,54  2000-12-04  Waterbury  (216)
   4  32411560480235319  Casiero Ezra         (78) 511  1,54  2000-12-03  Monticello  (84)
   5  32521773578114203  Garton Magda         (78) 431  2,11  2001-12-19  Washington  (462)
```

```
  6  34235504039842896  Porch Lorry          (72) 164  4,2  2002-08-13  Eaton  (577)
  7  33577388665139502  Hair Hortense        (67) 311  4,41  1999-09-04  Orlando  (327)
  8  34350655375829554  Solloway Tabbatha    (63) 406  3,56  2000-10-13  Boulder  (382)
  9  32857203688836212  Planque Ellswerth    (59) 230  2,43  2002-09-19  Adelaide-Mail-Centre  (44)
 10  36103100162715669  Jerdein Aurie        (37) 319  4,59  2003-12-06  Sacramento  (204)

 students sorted by the 'QuickS_referSort2d' method
 (reference sort in Quick Sort, Execution time: 00:00:00.0004304, 0,4304 ms)

  1  34122788215722749  Rodenborch Royce     (94) 403  1,4  2000-11-14  South-Bend  (410)
  2  34162078930108726  Wilcox Ab            (84) 628  1,41  2002-09-12  Kirkton  (475)
  3  32572965109821616  Casiero Jenny        (78) 511  1,54  2000-12-04  Waterbury  (216)
  4  32411560480235319  Casiero Ezra         (78) 511  1,54  2000-12-03  Monticello  (84)
  5  32521773578114203  Garton Magda         (78) 431  2,11  2001-12-19  Washington  (462)
  6  34235504039842896  Porch Lorry          (72) 164  4,2  2002-08-13  Eaton  (577)
  7  33577388665139502  Hair Hortense        (67) 311  4,41  1999-09-04  Orlando  (327)
  8  34350655375829554  Solloway Tabbatha    (63) 406  3,56  2000-10-13  Boulder  (382)
  9  32857203688836212  Planque Ellswerth    (59) 230  2,43  2002-09-19  Adelaide-Mail-Centre  (44)
 10  36103100162715669  Jerdein Aurie        (37) 319  4,59  2003-12-06  Sacramento  (204)
```

Zdroj: Vlastné spracovanie

Fungovanie metódy *Compare_sts* je zrejmé z jej zdrojového kódu a komentárov v ňom (obr. 6).

Obr. 6: Časť zdrojového kódu členskej metódy 'Compare_sts' súvisiaca s porovnávacím kritériom body na ubytovanie

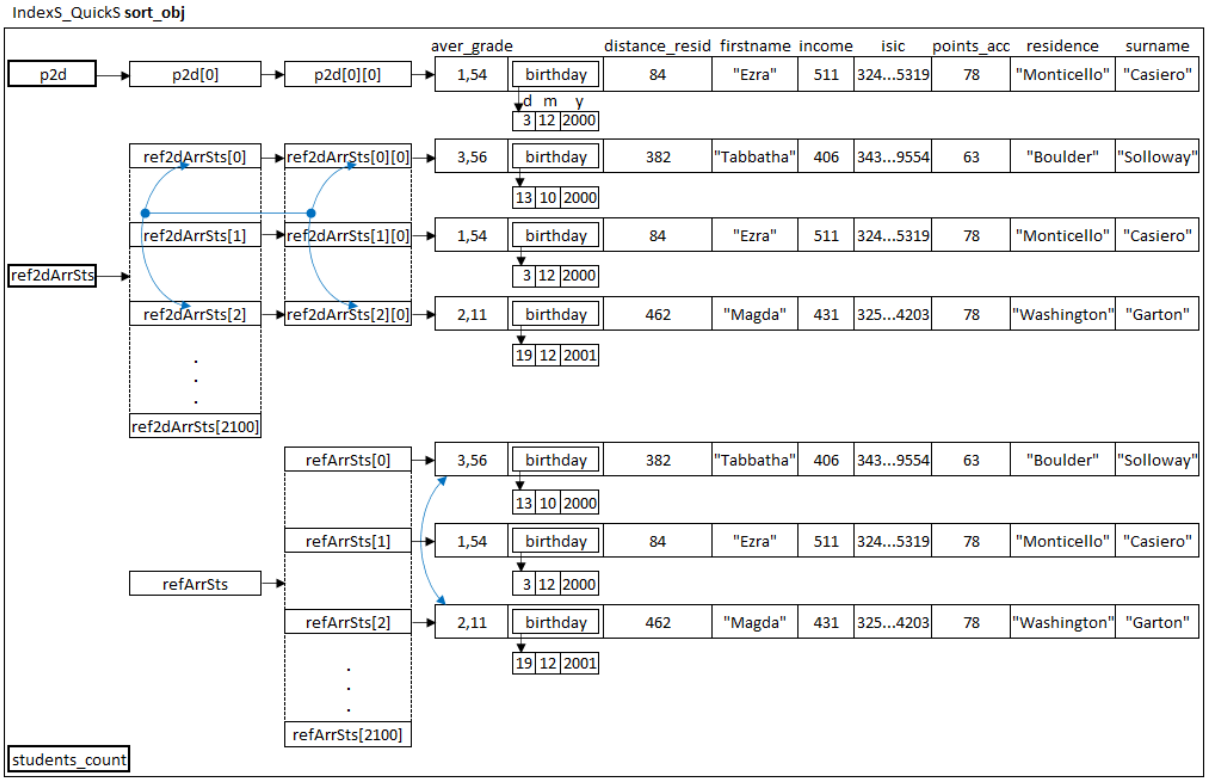
```
private int Compare_sts(Student a, Student b, int criterX) {
  int porovnanie;

  if (criterX == 1) // studenti sa budu porovnavat a usporiadavat najskor podla BODOV na UBYTOVAN.,
  {
      // cize najskor sa porovnaju a vo volajucej metode usporiadaju podla bodov na ubytovanie od
      // VACSIEHO poctu bodov po mensi, ak maju body na ubytovanie rovnake,
      if (a.points_acc > b.points_acc) return 1;
      if (a.points_acc < b.points_acc) return -1;

      // tak potom podla ich priemernej znamky za doterajsie studium od MENSEJ po vacsiu
      // priemernu znamku, ak maju aj priemernu znamku rovnaku,
      if (a.aver_grade < b.aver_grade) return 1;
      if (a.aver_grade > b.aver_grade) return -1;

      // tak potom podla prijmu studenta od MENSIEHO prijmu po vacsi, ak maju aj prijem rovnaky,
      if (a.income < b.income) return 1;
      if (a.income > b.income) return -1;

      // tak potom podla vzdialenosti ich bydliska od univerzity od VACSEJ vzdialenosti po
      // mensiu, ak maju aj vzdialenost bydliska rovnaku,
      if (a.distance_resid > b.distance_resid) return 1;
      if (a.distance_resid < b.distance_resid) return -1;
  }
                                          . . .
      long da, db;
      da = a.birthday.d + a.birthday.m * 100 + a.birthday.y * 10000;
      db = b.birthday.d + b.birthday.m * 100 + b.birthday.y * 10000;

      // tak potom sa budu porovnavat a vo volajucej metode usporiadavat podla ich DATUMU
      // NARODENIA od mladsieho po starsieho, ak maju aj datum narodenia rovnaky,
      if (da < db) return -1;
      if (da > db) return 1;
```

```
// tak potom sa porovnaju a vo vol. metode usporiad. podla priezviska, ak maju aj priezv. rovnake,
porovnanie = String.Compare(a.surname.ToLower(), b.surname.ToLower());
if (porovnanie < 0) return 1;
else if (porovnanie > 0) return -1;

// tak potom sa porovnaju a vo volaj. metode usporiadaju podla mena, ak maju aj meno rovnake,
porovnanie = String.Compare(a.firstname.ToLower(), b.firstname.ToLower());
if (porovnanie < 0) return 1;
else if (porovnanie > 0) return -1;

// tak potom sa porovnaju a vo volaj. met. usporiadaju podla velkosti ISIC, od mensieho po vacsie
if (a.isic < b.isic) return 1;
else if (a.isic > b.isic) return -1;
return 0;  }
```

Zdroj: Vlastné spracovanie

Všetky uvedené kľúčové usporiadavacie inštančné metódy *QuickS_movingSort*, *QuickS_movingSort2d* a *QuickS_referSort2d* po zavolaní v statickej metóde *Main* C# aplikácie pracujú s inštančnými premennými objektu *sort_obj*, pretože sú zavolané ako jeho inštančné metódy. Vnútornú štruktúru objektu *sort_obj* zobrazuje nasledujúci obr. 7.

Obr. 7: Vnútorná štruktúru objektu 'sort_obj' C# aplikácie. V poli 'ref2dArrSts' je 2 modrými šípkami znázornené referenčné usporiadavanie, ktoré realizuje napr. inštančná metóda 'QuickS_referSort2d' a v poli 'refArrSts' je jednou modrou šípkou znázornené presúvacie usporiadavanie, ktoré realizuje napr. inštančná metóda 'QuickS_movingSort'



Zdroj: Vlastné spracovanie

5 Experiment a jeho výsledky

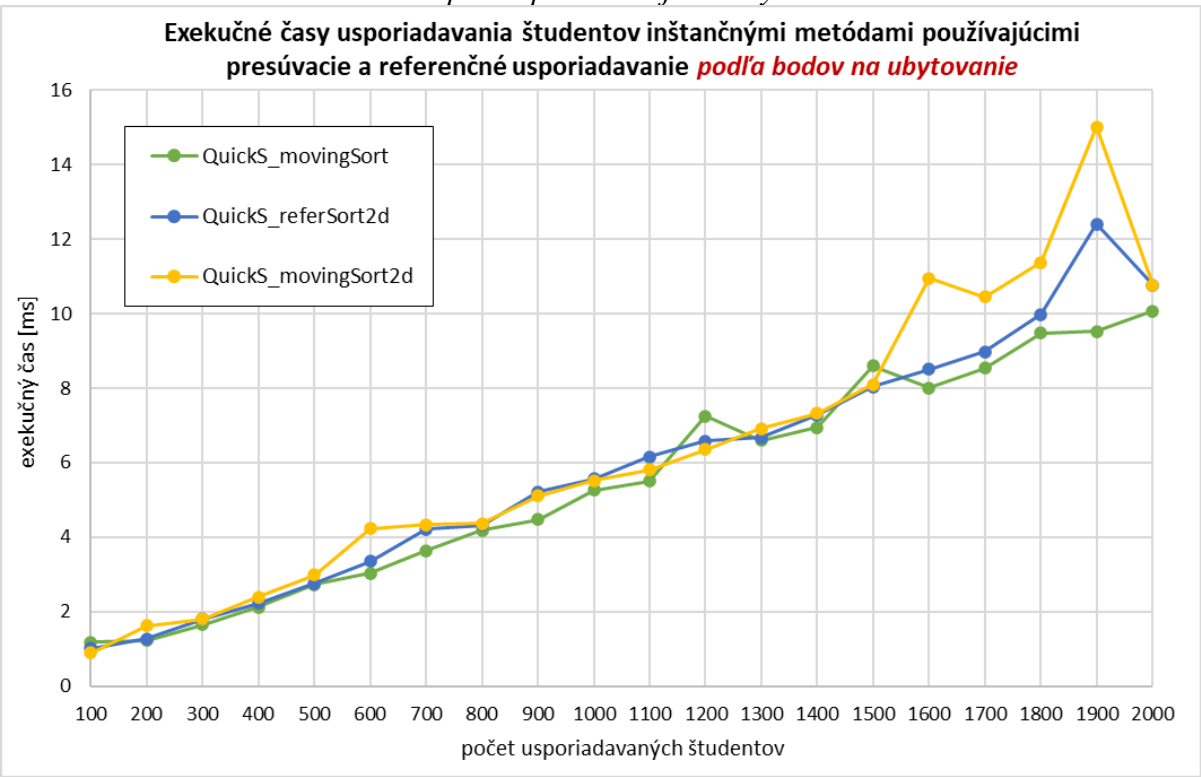
V experimente sme chceli overiť, či je pravdivý náš predpoklad, že referenčné usporiadavanie je exekučne efektívnejšie ako presúvacie usporiadavanie. Na potvrdenie alebo vyvrátenie tohto predpokladu sme použili našu C# aplikáciu. Jej 3 inštančné metódy *QuickS_movingSort*, *QuickS_movingSort2d* a *QuickS_referSort2d*, všetky s implementovaným algoritmom Quick Sort, dokážu vykonávať presúvacie a referenčné usporiadavanie. Inštančná metóda *QuickS_movingSort* usporiadava pole objektov triedy *Student refArrSts* s 1-rozmerným poľom referencií a používa presúvacie usporiadavanie. Inštančná metóda *QuickS_movingSort2d* usporiadava pole objektov triedy *Student ref2dArrSts* s 2-rozmerným poľom referencií a používa presúvacie usporiadavanie a inštančná metóda *QuickS_referSort2d* usporiadava rovnaké pole objektov triedy *Student* s rovnakým 2-rozmerným poľom referencií, ale používa referenčné usporiadavanie. Keďže obe metódy *QuickS_movingSort2d* a *QuickS_referSort2d* usporiadavajú úplne rovnaké pole, tým majú úplne totožné usporiadavané dátové štruktúry aj s infraštruktúrou tvorenou 2-rozmerným poľom referencií, čo objektivizuje porovnávanie exekučných časov oboch týchto metód. Z tohto dôvodu porovnanie exekučných časov oboch týchto metód poskytuje úplne korektné výsledky. Inštančná metóda *QuickS_movingSort* usporiadava pole objektov triedy Student refArrSts s rovnakými hodnotami usporiadavaných štruktúrovaných dát, avšak s jednoduchšou infraštruktúrou, len s 1-rozmerným poľom referencií, čo jej dáva určitú exekučnú výhodu pri usporiadavaní rovnakých štruktúrovaných dát oproti metódam *QuickS_movingSort2d* a *QuickS_referSort2*.

Exekučné časy usporiadavaní všetkých týchto 3 metód boli zmerané statickou metódou *Main* C# aplikácie a spolu s výsledkami usporiadavaní študentov týmito metódami boli touto C# aplikáciou zapísané do logovacieho diskového súboru *LogFile.txt*.

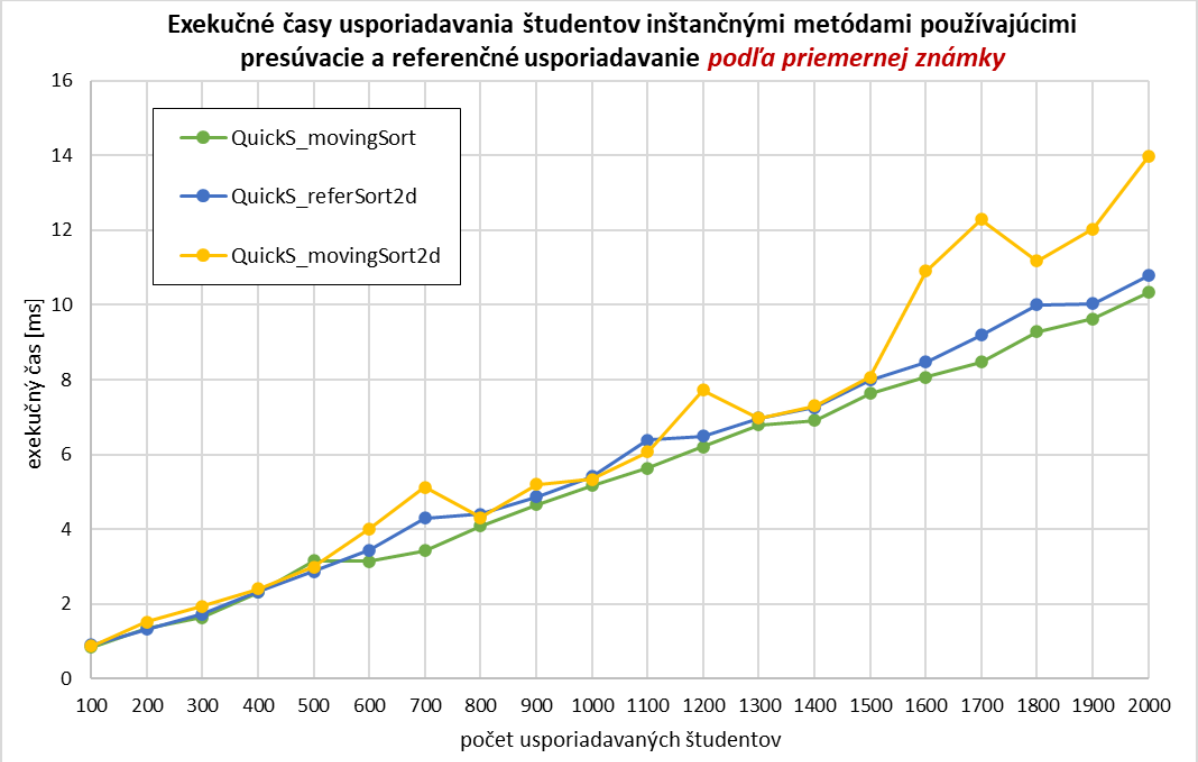
Počas experimentu všetky tieto 3 inštančné metódy usporiadavali rovnako početné obsahovo rovnaké vzorky študentov uložené v oboch týchto poliach *refArrSts* a *ref2dArrSts*. Merania exekučných časov všetkých 3 inštančných metód sa uskutočnili s 20 vzorkami štruktúrovaných dát študentov uložených v oboch poliach, ktoré obsahovali nasledovné počty študentov, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 a 2000.

Exekučné časy usporiadavania všetkých 20 vzoriek študentov všetkými 3 inštančnými metódami s implementovaným usporiadavacím algoritmom Quick Sort a používajúcimi presúvacie a referenčné usporiadavanie pre 2 usporiadavacie kritériá, *usporiadavanie podľa bodov na ubytovanie* a *usporiadavanie podľa priemernej známky*, sú zobrazené v nasledujúcich grafoch (obr. 8).

Obr. 8: Exekučné časy usporiadavania študentov inštančnými metódami C# aplikácie používajúcimi presúvacie a referenčné usporiadavanie podľa bodov na ubytovanie a podľa priemernej známky



Zdroj: Vlastné spracovanie



Zdroj: Vlastné spracovanie

Krátka analýza výsledkov experimentu.

Vykonaný experiment potvrdil náš predpoklad. Referenčné usporiadavanie je exekučne efektívnejšie oproti presúvaciemu usporiadaniu pri usporiadavaní vyššie uvedených vzoriek študentov. Rozdiely v exekučnej efektívnosti v prospech referenčného usporiadavania sú zreteľnejšie pri vyšších počtoch usporiadavaných študentov. V našich meraniach sa tento rozdiel stával výraznejším pri usporiadavaní 1500 a viac študentov. Z výsledkov vykonaného experimentu vyplýva, že hlavne pri väčších množstvách štruktúrovaných položiek uložených v 2-rozmernom poli takýchto položiek s 2-rozmerným poľom referencií je efektívnejšie referenčné usporiadavanie a dáva zmysel implementovať ho do usporiadavacích metód profesionálnych C# aplikácií. Ak tieto metódy nepotrebujú pracovať v takomto 2-rozmernom poli položiek s 2-rozmerným poľom referencií, ale pracujú v ňom len s 1-rozmerným poľom referencií, vtedy je exekučne efektívnejšie presúvacie usporiadavanie. Takéto pole *refArrSts* s 1-rozmerným poľom referencií usporadúvala metóda *QuickS_movingSort*, ktorá používala presúvacie usporiadavanie a bola zo všetkých 3 usporiadavacích metód, síce s veľmi malým rozdielom, ale exekučne najefektívnejšia.

6 Záver

Ako vyplýva z krátkej analýzy výsledkov experimentu, jeho výsledky potvrdili náš predpoklad. Referenčné usporiadavanie je exekučne efektívnejšie oproti presúvaciemu usporiadaniu pri usporiadavaní rozsiahlejších štruktúrovaných dát uložených v 2-rozmernom poli objektov obsahujúcich tieto štruktúrované dáta. Rozdiely v exekučnej efektívnosti v prospech referenčného usporiadavania sú zreteľnejšie pri vyšších počtoch usporiadavaných štruktúrovaných dát, v našom prípade štruktúrovaných dát študentov. V našich meraniach sa tento rozdiel stával výraznejším pri usporiadavaní 1500 a viac študentov. Z tohto vyplýva, že má zmysel implementovať referenčné usporiadavanie do usporiadavacích metód profesionálnych C# aplikácií, aj keď je nutné do usporiadavaného poľa štruktúrovaných dát vložiť 2-rozmerné pole referencií.

Literatúra

[1] Microsoft Corp. (2022). *Microsoft documentation*. Retrieved October 12, 2022, from https://docs.microsoft.com

[2] Sedgewick, R. (1998). *Algorithms in C parts 1-4. Fundamentals, data structures, sorting, searching*. Addison-Wesley Publishing Company, Inc.

[3] Sedgewick, R., Wayne, K. (2011). *Algorithms fourth edition*. Pearson Education, Inc.

[4] Sedgewick, R. (2018). *Algorithms, 4th Edition*. Retrieved October 12, 2022, from https://algs4.cs.princeton.edu/home/