Exekučná efektívnosť použitia v jazyku integrovaných dopytov v aplikácii vytvorenej v jazyku C#

Execution Efficiency of the Use of Language-Integrated Queries in a C# Application

Igor Košťál[1]

Abstrakt
V mnohých aplikáciách musí programátor implementovať rôzne vyhľadávacie a usporiadavacie algoritmy. Častokrát tieto algoritmy vykonávajú zložitejšie vyhľadávanie a usporiadavanie, napr. vyhľadávajú študentov v ich zozname podľa rozsahu bodov na ubytovanie, ktoré dosiahli a v týchto skupinách ich usporiadavajú podľa týchto bodov alebo podľa priezviska. Takéto vyhľadávanie a usporiadavanie je možné vykonať pomocou usporiadavacích a vyhľadávacích metód aplikácie, bez použitia v jazyku integrovaných dopytov, alebo pomocou týchto dopytov. Zaujímalo nás, ktorý z týchto spôsobov vyhľadávania a usporiadavania je exekučne efektívnejší a vhodnejší pre uvedené vyhľadávanie a usporiadavanie najmä vo väčších, napr. v 2000-položkových súboroch dát študentov, v aplikácii vytvorenej v jazyku C#. Výsledky experimentu, v ktorom skúmame exekučnú efektívnosť usporiadavania a skupinového vyhľadávania pomocou vyhľadávacích a usporiadavacích metód a zdrojového kódu aplikácie, bez použitia v jazyku integrovaných dopytov, a pomocou týchto dopytov, sú uvedené v článku.

Kľúčové slová
v jazyku integrovaný dopyt, skupinové vyhľadávanie dát, usporiadavanie dát, programovací jazyk C#

Abstract
In many applications, the programmer has to implement various searching and sorting algorithms. These algorithms often perform more complex searching and sorting, e.g. they search for students in their list according to the range of points for an accommodation that they have achieved. In these groups, these algorithms sort students according to these points or the last name. Such searching and sorting can be done using the sorting and searching methods of the application, without using language-integrated queries, or by using these queries. We were interested in, which of these ways of searching and sorting is more execution efficient and more suitable for mentioned searching and sorting, especially in bigger data sets, e.g. in 2000-item students datasets, in an application created in C#. The results of the experiment, in which we examine the execution efficiency of sorting and group search using searching and sorting methods and the source code of the application without using language-integrated queries, and by using these queries, are presented in the paper.

Key words
language-integrated query, group searching for data, sorting data, the C# programming language

JEL classification
C88

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

1 Úvod

Ako sme uviedli vyššie, programátor musí v aplikáciách častokrát implementovať algoritmy, ktoré vykonávajú zložitejšie vyhľadávanie a usporiadavanie, napr. vyhľadávajú študentov v ich zozname podľa rozsahu bodov na ubytovanie, ktoré dosiahli, tzv. intervalové vyhľadávanie, a v týchto skupinách ich usporiadavajú podľa týchto bodov alebo podľa priezviska. Programátor má vždy záujem implementovať uvedené algoritmy čo najefektívnejšie, čiže exekučne alebo pamäťovo najefektívnejšie. Uvedené vyhľadávanie a usporiadavanie je možné vykonať pomocou usporiadavacích a vyhľadávacích metód a zdrojového kódu aplikácie, bez použitia v jazyku C# integrovaných dopytov, alebo pomocou týchto dopytov. Zaujímalo nás, ktorý z týchto spôsobov vyhľadávania a usporiadavania je exekučne efektívnejší a vhodnejší pre uvedené vyhľadávanie a usporiadavanie najmä vo väčších, napr. v 2000-položkových súboroch dát študentov, v aplikácii vytvorenej v jazyku C#. Predpokladáme, že vyhľadávanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku C# integrovaného skupinového dopytu je exekučne efektívnejšie ako vykonanie takéhoto vyhľadávania pomocou zdrojového kódu, ktorý nepoužíva v jazyku C# integrovaný skupinový dopyt. Za účelom overenia tejto hypotézy sme pomocou našej C# aplikácie, ktorá dokáže vyhľadávať študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku C# integrovaného skupinového dopytu a pomocou zdrojového kódu, ktorý nepoužíva v jazyku C# integrovaný skupinový dopyt a zároveň dokáže zmerať exekučné časy, vykonali experiment. Vyhodnotenie experimentu potvrdí alebo vyvráti túto našu hypotézu.

V nasledujúcich kapitolách sa krátko zaoberáme v jazyku C# integrovanými dopytmi, triednou architektúrou a kľúčovými časťami zdrojového kódu našej C# .NET aplikácie a vyššie spomenutým experimentom.

2 V jazyku C# integrovaný dopyt, dopytovacie výrazy

V jazyku integrovaný dopyt (anglicky: Language-Integrated Query (LINQ)) je názov alebo pomenovanie pre sadu technológií integrujúcich dopytovacie schopnosti priamo do C# jazyka. Dopyty sa zapisujú ako jednoduché reťazce bez uvádzania dátových typov. Dopytovacie výrazy sa zapisujú v deklaratívnej *dopytovacej syntaxe*. Pomocou dopytovacej syntaxe môžeme vykonať filtrovacie, usporiadavacie, spájacie, zoskupovacie a projektovacie operácie na dátových zdrojoch s minimálnym množstvom zdrojového kódu. Pomocou rovnakého dopytovacieho výrazu môžeme vykonať dopyt a transformáciu dát z nejakého dátového typu dátového zdroja na iný dátový typ. (Microsoft, 2023)

Vlastnosti dopytovacích výrazov (Microsoft, 2023):

- dopytovacie výrazy vykonávajú dopyty a transformujú dáta z nejakého pre LINQ známeho dátového zdroja na iný dátový typ. Napr. jeden dopyt môže získať dáta z SQL databázy a vyprodukovať XML dáta ako výstup.
- dopytovacie výrazy používajú veľa známych syntaktických konštrukcií jazyka C#, ktoré ich robia ľahšie čitateľnými.
- premenné v dopytovacích výrazoch sú všetky prísne typové.
- dopyt nie je vykonaný, pokiaľ neiterujeme cez dopytovú premennú, napr. vo *foreach* príkaze.
- kompilátor jazyka C# konvertuje dopytovacie výrazy na volania štandardných dopytovacích operátorových metód podľa pravidiel definovaných v špecifikácii jazyka C#. Dopyt môže byť vyjadrený pomocou dopytovacej syntaxe alebo môže byť vyjadrený pomocou metódovej syntaxe. V niektorých prípadoch je dopytovacia syntax čitateľnejšia, výstižnejšia a stručnejšia, v iných prípadoch je takouto

metódová syntax. Medzi týmito dvomi formami zápisu dopytu nie je žiaden sémantický alebo vykonávací rozdiel.

- niektoré dopytovacie operácie, napr. *Count* alebo *Max*, nemajú ekvivalentnú dopytovaciu klauzulu, preto musia byť vyjadrené vo forme volania metódy. Metódová syntax môže byť kombinovaná s dopytovacou syntaxou rôznym spôsobom.
- dopytovací výraz môže byť kompilovaný do výrazového stromu alebo do delegáta, závisí to od toho, na aký dátový typ bol dopyt aplikovaný. Dopyty na dátový typ *IEnumerable<T>* sú kompilované do delegátov, dopyty na dátové typy *IQueryable* a *IQueryable<T>* sú kompilované do výrazových stromov.

Všetky LINQ dopytové operácie pozostávajú z troch odlišných akcií (Microsoft, 2024a):

- získanie dátového zdroja,
- vytvorenie dopytu a
- vykonanie dopytu, napr. vo *foreach* príkaze cyklu.

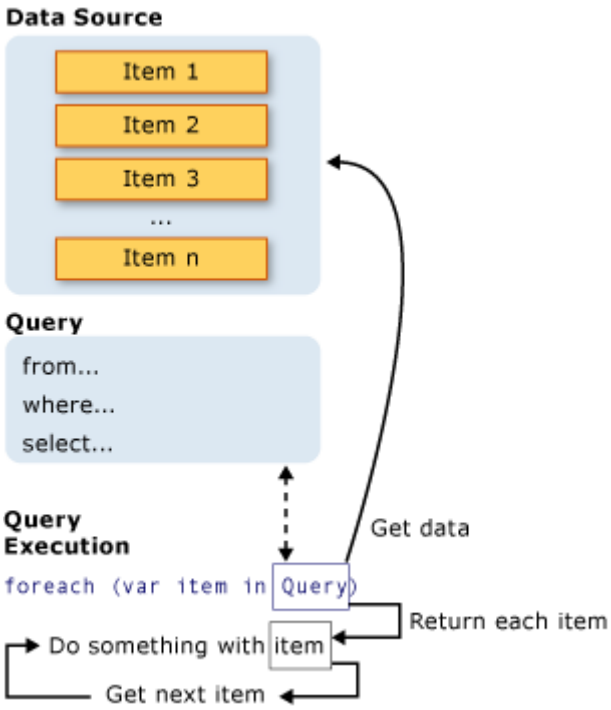
Príklad zdrojového kódu dopytovej operácie, ktorá získa dátový zdroj, zoznam študentov uložený v objekte *students* triedy *List*, vytvorí dopyt, v ktorom zoskupí a usporiada študentov podľa bodov na ubytovanie a vo *foreach* príkaze vykoná dopyt, zobrazí a zapíše takto získaných a usporiadaných študentov na konzolu a do diskového súboru, je na obr. 1.

Obr. 1: Zdrojový kód ukážkového dopytu

```
List<Student> students = new List<Student>();  // datovy zdroj, objekt 'students' triedy 'List'
List<Student> studentsSorted = new List<Student>();  // zoznam s buducimi usporiadanymi studentmi

var groupByPointsToAccomodationQuery =     // vytvorenie dopytu
        from student in students
        group student by student.Points_acc into newGroupAcc
        orderby newGroupAcc.Key descending
        select newGroupAcc;

foreach (var nameGroupAcc in groupByPointsToAccomodationQuery)  // vykonanie dopytu
{
        foreach (var item in nameGroupAcc)  {
            // vkladanie usporiadanych objektov s datami studentov podla bodov na ubytovanie
            // do noveho zoznamu, objektu 'studentsSorted' triedy 'List'
            studentsSorted.Add(item);

            string year_str = "";
            if (item.Aver_grade.ToString().Count() < 4)
                year_str = " " + item.Birthday.y.ToString();
            else
                year_str = item.Birthday.y.ToString();
                                                    . . .

            k++;
        }
}
```

Zdroj: Vlastné spracovanie

Obr. 2: Fungovanie LINQ dopytu



Zdroj: Microsoft, 2024a

Dopyt je sada inštrukcií, ktorá opisuje aké dáta chceme získať z daného dátového zdroja, alebo zdrojov, a aký tvar a organizáciu by tieto vrátené dáta mali mať (Microsoft, 2024b). Pre rôzne dátové zdroje sa používajú rozdielne natívne dopytovacie jazyky, napr. SQL sa používa pre relačné databázy, XQuery pre XML. Vývojári sa musia naučiť nový dopytovací jazyk pre každý typ dátového zdroja alebo dátového formátu. LINQ zjednodušuje túto situáciu ponúkajúc konzistentný C# jazykový model pre rôzne druhy dátových zdrojov a formátov. V LINQ dopyte vždy pracujeme so C# objektmi. Používame rovnaké základné kódové vzory pre dopytovanie a transformovanie dát v XML dokumentoch, SQL databázach, v .NET kolekciách a iných formátoch, pre ktoré je LINQ poskytovateľ prístupný. (Microsoft, 2024a)

Dopytovací výraz je dopyt vyjadrený v dopytovacej syntaxe. Kompilátorom jazyka C# je chápaný rovnako ako iný výraz a môže byť použitý v ľubovoľnom kontexte, v ktorom je možné použiť C# výraz. Dopytovací výraz sa skladá zo sady klauzúl napísaných v deklaratívnej syntaxe. Každá klauzula obsahuje jeden alebo viacero C# výrazov a tieto výrazy samotné môžu byť dopytovacími výrazmi alebo môžu obsahovať dopytovacie výrazy.

Dopytovací výraz musí začínať s klauzulou *from*, ktorá špecifikuje dátový zdroj a musí končiť klauzulou *select* alebo *group*, ktoré špecifikujú dátový typ vrátených elementov. Medzi prvou klauzulou *from* a poslednou klauzulou *select* alebo *group* môže dopytovací výraz obsahovať jednu alebo viacero voliteľných klauzúl *where*, *orderby*, *join*, *let* a dokonca aj ďalšie *from* klauzuly. Môžeme tiež použiť kľúčové slovo *into* pre získanie výsledkov z klauzúl *join* alebo *group*, ktoré slúžia ako zdroj pre viaceré dopytovacie klauzuly v rovnakom dopytovacom výraze. (Microsoft, 2024b)

Na obdržanom dátovom zdroji môžeme v dopytovacom výraze vykonať rôzne operácie (Microsoft, 2024c):

- filtrovanie dát pomocou klauzuly *where*,
- usporiadavanie dát pomocou klauzuly *orderby* a voliteľného kľúčového slova *descending*,

- zoskupovanie dát pomocou klauzuly *group* a voliteľného kľúčového slova *into*,
- spájanie dát pomocou klauzuly *join*,
- projektovanie (premietanie) dát pomocou klauzuly *select*.

Väčšinu z uvedených operácií a klauzúl sme použili v dopytovacích výrazoch v zdrojovom kóde našej C# .NET aplikácie.

3 C# .NET aplikácia hľadajúca študentov v ich zozname podľa rozsahu bodov na ubytovanie bez použitia a s použitím v jazyku integrovaných dopytov

Naša konzolová C# .NET aplikácia bola vytvorená v programovacom jazyku C# vo vývojom prostredí Microsoft Visual Studio Enterprise 2019. Aplikácia dokáže podľa požiadavky používateľa načítať dáta študentov z vybraného diskového súboru, tvoriaceho dátový vstup aplikácie, do zoznamu objektov. S týmto zoznamom študentov, s objektom knižničnej triedy *List*, pracujú metódy aplikácie. Pomocou svojho zdrojového kódu a svojich metód aplikácia dokáže:

- zobraziť neusporiadaný zoznam všetkých študentov na konzolu,
- vykonať základné usporiadanie študentov podľa používateľom vybraného kritéria, čiže podľa priezvisk alebo podľa bodov na ubytovanie študentov,
- ďalej aplikácia dokáže pomocou v jazyku integrovaného skupinového dopytu vyhľadať študentov podľa intervalov ich bodov na ubytovanie,
- pomocou svojho zdrojového kódu, bez použitia v jazyku integrovaného skupinového dopytu, vyhľadať študentov podľa intervalov ich bodov na ubytovanie,
- zmerať exekučné časy všetkých vyhľadávaní a ich výsledky spolu s exekučnými časmi jednotlivých vyhľadávaní zobraziť na konzolu (obr. 11 a 12) a zároveň ich zapísať do logovacieho súboru *LogFile.txt*.

C# .NET aplikácia obsahuje 3 triedy s nasledovnými členmi (uvedené sú len dôležité členy tried):

- *Date* - obsahuje celočíselné členy *d*, *m*, *y*. Objekty tejto triedy, vytvorené ako vnorené v triede *Student*, slúžia na ukladanie dňa, mesiaca a roku narodenia študenta.
- *Student* - obsahuje členské metódy pre načítanie a zmenu inštančných premenných obsahujúcich meno, priezvisko, bydlisko, body na ubytovanie, príjem, vzdialenosť bydliska od univerzity, priemernú známku a číslo ISIC karty študenta a tiež vnorený objekt *Birthday* triedy *Date*, ktorý obsahuje deň, mesiac a rok narodenia študenta, ktorého dáta sú uložené v objekte triedy *Student*. Ďalej táto trieda obsahuje nasledujúce kľúčové členské metódy:
  - *QuickS_movingSort* - verejná statická rekurzívna členská metóda (obr. 4), ktorá usporiada objekty zoznamu *students* (objekt triedy *List*) obsahujúce dáta študentov pomocou implementovaného algoritmu Quick Sort. Táto metóda používa presúvacie usporiadavanie, tzn. počas usporiadavania presúva objekty zoznamu *students*.
  - *Compare_sts* - privátna statická členská metóda (obr. 3), ktorá porovnáva dvoch študentov *a* a *b* podľa kritéria *criterX*. Táto metóda dokáže porovnať týchto dvoch študentov podľa priezvisk (*criterX == 41*), bodov na ubytovanie (*criterX == 1*), podľa vzdialeností ich bydlísk od univerzity (*criterX == 2*), ich príjmov (*criterX == 3*), priemerných známok (*criterX == 4*), ich dátumov narodenia a podľa ich čísiel ISIC kariet. Metóda *Compare_sts* je volaná metódou *QuickS_movingSort*.

Obr. 3: Časť zdrojového kódu členskej metódy 'Compare_sts' triedy 'Student'

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

  if (criterX == 41)  // studenti sa budu usporiadavat najskor podla PRIEZVISKA,
  { // najskor sa porovnaju podla priezviska, ak maju aj priezvisko rovnake,
    porovnanie = String.Compare(a.LastName.ToLower(), b.LastName.ToLower());
    if (porovnanie < 0)   return 1;  else if (porovnanie > 0)   return -1;
    // tak potom sa porovnaju 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 podla bodov na ubytovanie od NAJVACSIEHO poctu
    // bodov po najmensi, 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 sa porovnaju podla ich priemernej znamky za doterajsie studium od NAJMENSEJ po
    // najvacsiu 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 NAJMENSIEHO prijmu po najvacsi, 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 skoly od NAJVACSEJ vzdialenosti po najmensiu
    if (a.Distance_resid > b.Distance_resid) return 1;
    if (a.Distance_resid < b.Distance_resid) return -1;
  }
  if (criterX == 1)  {  // studenti sa budu usporiadavat najskor podla BODOV na UBYTOVANIE,
    // cize najskor sa porovnaju podla bodov na ubytovanie od NAJVACSIEHO poctu bodov po
    // najmensi, 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 sa porovnaju podla priezviska, ak maju aj priezvisko rovnake,
    porovnanie = String.Compare(a.LastName.ToLower(), b.LastName.ToLower());
    if (porovnanie < 0)   return 1;  else if (porovnanie > 0)   return -1;

    // tak potom sa porovnaju 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 podla ich priemernej znamky za doterajsie studium od NAJMENSEJ po najvacsiu
    // 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 NAJMENSIEHO prijmu po najvacsi, 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 skoly od NAJVACSEJ vzdialenosti po najmensiu
    if (a.Distance_resid > b.Distance_resid) return 1;
    if (a.Distance_resid < b.Distance_resid) return -1;
  }
          . . .
}
```

Zdroj: Vlastné spracovanie

Obr. 4: Zdrojový kód členskej metódy 'QuickS_movingSort' triedy 'Student'

```
public static void QuickS_movingSort(List<Student> data, int left, int right, int criterx)
{
  if (left < right)
  {
        int i = left, j = right;
        Student p = data[(left + right) / 2];
        do
        {
            while (Compare_sts(data[i], p, criterx) > 0) i++;
            while (Compare_sts(p, data[j], criterx) > 0) j--;

            if (i >= j)
              break;
            Student t = new Student();
            t = data[i];
            data[i] = data[j];
            data[j] = t;
        } while (true);
        QuickS_movingSort(data, left, j, criterx);
        QuickS_movingSort(data, j + 1, right, criterx);
  }
}
```

Zdroj: Vlastné spracovanie

- *GetPercentile* - verejná statická členská metóda (obr. 5), ktorá vráti vypočítaný percentil študenta *s* z jeho bodov na ubytovanie. Napr., ak má študent 51, 52, 53, ... , 59 bodov na ubytovanie, tak táto metóda vypočíta a vráti percentil 5. Túto metódu volajú v jazyku integrované skupinové dopyty v statickej metóde *Main* triedy *Program*.

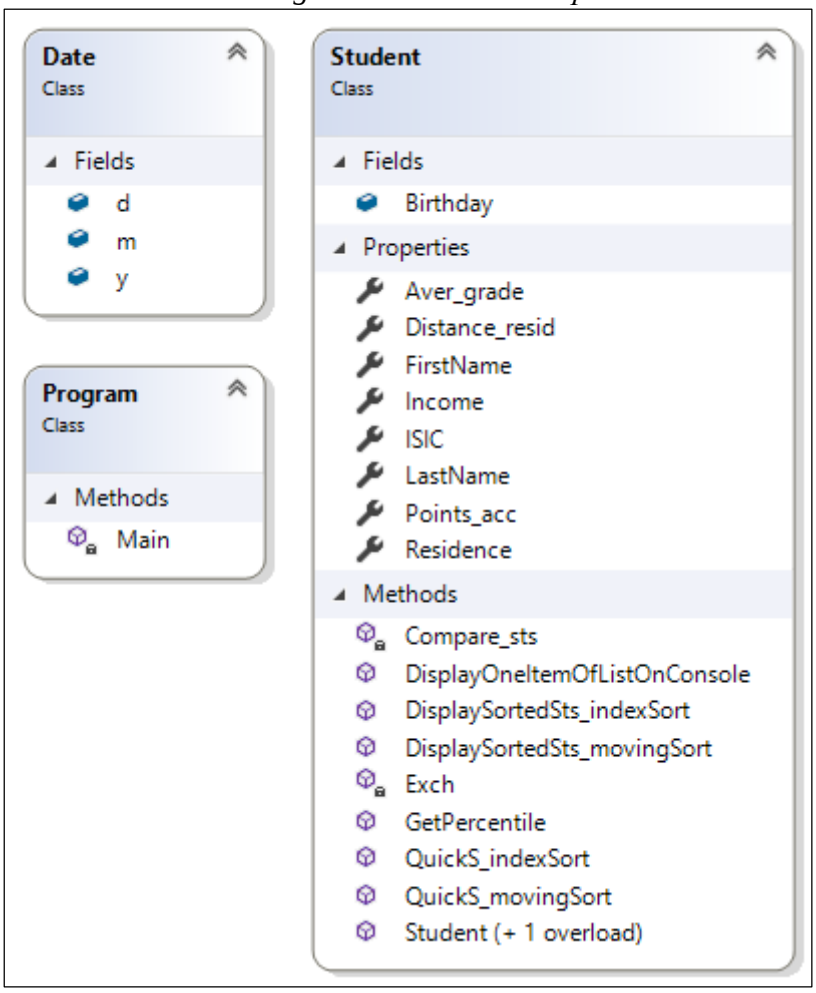
Obr. 5: Zdrojový kód členskej metódy 'GetPercentile' triedy 'Student'

```
public static int GetPercentile(Student s)
{
    return s.Points_acc > 0 ? (int)s.Points_acc / 10 : 0;
}
```

Zdroj: Vlastné spracovanie

Okrem týchto kľúčových metód trieda *Student* obsahuje dva konštruktory, jeden bezparametrický a druhý parametrický, verejnú statickú rekurzívnu členskú metódu *QuickS_indexSort*, ktorá usporiada objekty zoznamu *students* (objekt triedy *List*) obsahujúce dáta študentov pomocou implementovaného algoritmu Quick Sort indexovým usporiadavaním, bez ich premiestňovania v tomto zozname, privátnu statickú členskú metódu *Exch*, ktorá vymieňa len indexy v poli indexov bez premiestňovania objektov zoznamu *data* a ktorá je volaná členskou metódou *QuickS_indexSort*, ďalej verejnú statickú členskú metódu *DisplaySortedSts_ indexSort*, ktorá poskladá a vráti reťazec s dátami usporiadaných študentov členskou metódou *QuickS_indexSort*, verejnú statickú členskú metódu *DisplaySortedSts_movingSort*, ktorá poskladá, zobrazí na konzolu a vráti reťazec s dátami usporiadaných študentov členskou metódou *QuickS_movingSort* a verejnú statickú členskú metódu *DisplayOneItemOfListOnConsole*, ktorá poskladá, zobrazí na konzolu a vráti reťazec s dátami študenta uloženého v objekte *itemX* triedy *Student*.

Obr. 6: Diagram tried C# .NET aplikácie



Zdroj: Vlastné spracovanie

- *Program* - v jej statickej metóde *Main* sú vytvorené dva objekty *students* a *studentsSorted* triedy *List*. Po používateľovom výbere vstupného diskového súboru tvoriaceho dátový vstup aplikácie, je v zdrojovom kóde metódy *Main* vytvorený dátový prúd *sr*, ktorý je objektom triedy *StreamReader*, a do tohto dátového prúdu spojeného s vybratým vstupným diskovým súborom sú pomocou konštruktora triedy *StreamReader* načítané všetky dáta z pripojeného ASCII diskového súboru. V ďalšom zdrojovom kóde metódy *Main* sú po jednotlivých riadkoch čítané dáta z dátového prúdu *sr*, čo sú vždy dáta jedného študenta a tieto dáta sú zapísané do inštančných premenných objektu triedy *Student*, ktorý je následne pridaný na koniec zoznamu *students*. Ďalej metóda *Main* vypíše dáta študentov z neusporiadaného zoznamu *students* na konzolu, následne usporiada tento neusporiadaný zoznam študentov podľa ich priezvisk pomocou v jazyku integrovaného skupinového dopytu (obr. 7), usporiadaný zoznam zapíše do objektu *studentsSorted* triedy *List*, vypíše ho na konzolu (obr. 11 a 12) a zapíše ho do logovacieho súboru *LogFile.txt*.
  V ďalšom zdrojovom kóde metódy *Main* sú pomocou v jazyku integrovaného skupinového dopytu vyhľadaní študenti podľa intervalov ich bodov na ubytovanie (obr. 8), je zmeraný exekučný čas tohto vyhľadávania a následne sú títo vyhľadaní študenti vypísaní na konzolu (obr. 11 a 12) a do logovacieho súboru *LogFile.txt*.

Obr. 7: Usporiadanie študentov podľa ich priezvisk pomocou v jazyku integrovaného skupinového dopytu, ich zobrazenie na konzolu a ich zapísanie do premennej 'str_for_sw', ktorá bude zapísaná do logovacieho súboru 'LogFile.txt', v zdrojovom kóde metódy 'Main'

```
List<Student> students = new List<Student>();  // datovy zdroj, objekt 'students' triedy 'List'
List<Student> studentsSorted = new List<Student>();  // zoznam s buducimi usporiadanymi studentmi
                                                    . . .
var groupByLastNamesQuery =
from student in students
group student by student.LastName into newGroup
orderby newGroup.Key
select newGroup;
                                                    . . .
flagConsole = 0;
k = 1;
foreach (var nameGroup in groupByLastNamesQuery)
{
      foreach (var item in nameGroup)
      {   // vkladanie usporiadanych objektov s datami studentov podla 'Lastname' do zoznamu
          // 'studentsSorted'
          studentsSorted.Add(item);

          string year_str = "";
          if (item.Aver_grade.ToString().Count() < 4)
                year_str = " " + item.Birthday.y.ToString();
          else
                year_str = item.Birthday.y.ToString();
                                                    . . .
          if (k < 21)  // ak je poradove cislo studenta nacitaneho zo zoznamu studentov 'k < 21', tak ho
          {             // zobrazime na konzolu
            Console.WriteLine($"{rank_cons}  {item.ISIC} {item.LastName}{gapLastFirstNameCons}
                        {item.FirstName}\t({item.Points_acc}) " +
                        $"{item.Income} {item.Aver_grade} {year_str}-{month_str}-{day_str}\t  " +
                        $"{item.Residence}\t({item.Distance_resid})");
          }
          else if (flagConsole == 0)
          {
            Console.WriteLine($"\nThe list of students is too extensive, greater than 20 items." +
                        $"\nFull list of students is written into the 'LogFile.txt' file");
            flagConsole++;
          }

      str_for_sw += $"{rank}  {item.ISIC} {item.LastName}{gapLastFirstName_sw}{item.FirstName}" +
                  $"{gapFirstNamePoints_acc_sw}({item.Points_acc}) " +
                  $"{item.Income} {item.Aver_grade} {year_str}-{month_str}-{day_str}\t  " +
                  $"{item.Residence}\t({item.Distance_resid})" + Environment.NewLine;

          k++;
       }
}
```

Zdroj: Vlastné spracovanie

V ďalšom zdrojovom kóde metódy *Main* sú pomocou volania statickej členskej metódy *QuickS_movingSort* študenti uložení v objektoch triedy *Student*, ktoré sú uložené v zozname *students*, usporiadaní podľa ich priezvisk. Táto metóda používa presúvacie usporiadavanie, tzn. počas usporiadavania presúva objekty v zozname *students*. Následne sú títo usporiadaní študenti pomocou volania statickej členskej metódy *DisplaySortedSts_movingSort* vypísaní na konzolu (obr. 11 a 12). Ďalší zdrojový kód metódy *Main* vyhľadá študentov podľa intervalov ich bodov na

ubytovanie, zmeria exekučný čas tohto vyhľadávania a vyhľadaných študentov spolu so zmeraným exekučným časom vypíše na konzolu (obr. 9) a do logovacieho súboru *LogFile.txt*. Tento zdrojový kód nepoužíva v jazyku integrovaný skupinový dopyt.

Obr. 8: Vyhľadanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu, ich zobrazenie na konzolu a ich zapísanie do premennej 'str_for_sw', ktorá bude zapísaná do logovacieho súboru 'LogFile.txt', v zdrojovom kóde metódy 'Main'

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Reset(); // zastavi meranie casu a resetuje uplynuty cas na 0
stopWatch.Start(); // spusti alebo obnovi meranie uplynuteho casu

var groupByPercentileQuery =
from student in studentsSorted
let percentile = Student.GetPercentile(student) // do skupiny vyberame studentov, ktori maju percentil
group new                    // napr. 5, napr. studenti s bodmi na ubytovanie 51, 53, 57 maju percentil 5
{
  student.FirstName,
  student.LastName,
  student.Points_acc,
  student.Income,
  student.Aver_grade,
  student.Birthday,
  student.Residence,
  student.Distance_resid
} by percentile into percentGroup
orderby percentGroup.Key
select percentGroup;

stopWatch.Stop(); // zastavi meranie uplynuteho casu v intervale
TimeSpan ts_IS = stopWatch.Elapsed;  // ziska uplynuty casa ako hodnotu 'TimeSpan'.
string strIntervalStopWatch_IS = ts_IS.ToString();
double msIntervalStopWatch_IS = ts_IS.TotalMilliseconds;
                                                    . . .
foreach (var studentGroup in groupByPercentileQuery) {
Console.WriteLine($"{studentGroup.Key * 10} - {studentGroup.Key * 10 + 9} [{studentGroup.Count()}]");
  str_for_sw += $"{studentGroup.Key * 10} - {studentGroup.Key * 10 + 9} [{studentGroup.Count()}]\n";
  flagConsole = 0;  k = 1;
  foreach (var item in studentGroup) {
      string year_str = "";
      if (item.Aver_grade.ToString().Count() < 4)  year_str = " " + item.Birthday.y.ToString();
      else  year_str = item.Birthday.y.ToString();
                                                    . . .
      else {   // ak je poradove cislo studenta nacitaneho zo zoznamu studentov 'k < 21', tak ho
        if (k < 21)  // zobrazime na konzolu
          Console.WriteLine($"\t{item.LastName}\t     {item.FirstName}\t({item.Points_acc})" +
                            $" {item.Income}  {item.Aver_grade}  {year_str}-{month_str}-{day_str}\t  " +
                            $"{item.Residence}\t({item.Distance_resid})");
        else if (flagConsole == 0)  {
          Console.WriteLine($"\nThe list of students is too extensive, greater than 20 items.\n" +
                            $"Full list of students is written into the 'LogFile.txt' file.\n");   flagConsole++;  }

        str_for_sw += $"\t{item.LastName}\t     {item.FirstName}\t({item.Points_acc})" +
                            $" {item.Income}  {item.Aver_grade}  {year_str}-{month_str}-{day_str}\t  " +
                            $"{item.Residence}\t({item.Distance_resid})\n";  }
      k++;  } }
```

Zdroj: Vlastné spracovanie

Obr. 9: Vyhľadanie študentov podľa intervalov ich bodov na ubytovanie bez použitia v jazyku integrovaného skupinového dopytu, ich zobrazenie na konzolu a ich zapísanie do premennej 'str_for_sw', ktorá bude zapísaná do logovacieho súboru 'LogFile.txt', v zdrojovom kóde metódy 'Main'

```
stopWatch.Reset(); // zastavi meranie casu a resetuje uplynuty cas na 0
stopWatch.Start(); // spusti alebo obnovi meranie uplynuteho casu

int flagConsWITHOUT_LINQ = 0;
Console.WriteLine($"{30} - {39}");
str_for_sw += $"\n{30} - {39}\n";
// hladanie studentov v skupine s bodmi na ubytovanie v rozsahu '30 - 39'
foreach (var item in students)  {
  if (item.Points_acc >= 30 && item.Points_acc < 40)  {
      str_for_sw += Student.DisplayOneItemOfListOnConsole(item, countInGroup + 1,
                                                                  ref flagConsWITHOUT_LINQ);
      countInGroup++;
  }
}
if (countInGroup == 0)  {
  Console.WriteLine($"\tNONE students");
  str_for_sw += $"\tNONE students\n";
}
else  {
  Console.WriteLine($"\t[{countInGroup}]\n");
  str_for_sw += $"\t[{countInGroup}]\n";
}
                                                  . . .

flagConsWITHOUT_LINQ = 0;
countInGroup = 0;
Console.WriteLine($"{90} - {100}");
str_for_sw += $"{90} - {100}\n";
// hladanie studentov v skupine s bodmi na ubytovanie v rozsahu '90 - 100'
foreach (var item in students)  {
  if (item.Points_acc >= 90 && item.Points_acc <= 100)  {
      str_for_sw += Student.DisplayOneItemOfListOnConsole(item, countInGroup + 1,
                                                                  ref flagConsWITHOUT_LINQ);
      countInGroup++;
  }
}
if (countInGroup == 0)  {
  Console.WriteLine($"\tNONE students");
  str_for_sw += $"\tNONE students\n";
}
else  {
  Console.WriteLine($"\t[{countInGroup}]\n");
  str_for_sw += $"\t[{countInGroup}]\n";
}

stopWatch.Stop(); // zastavi meranie uplynuteho casu v intervale
                  // ziska uplynuty casa ako hodnotu 'TimeSpan'
TimeSpan ts_IS1 = stopWatch.Elapsed;
string strIntervalStopWatch_IS1 = ts_IS1.ToString();
double msIntervalStopWatch_IS1 = ts_IS1.TotalMilliseconds;
```

Zdroj: Vlastné spracovanie

Nasledujúci zdrojový kód metódy *Main* vykoná podobnú činnosť, s tým rozdielom, že študentov z neusporiadaného zoznamu *students* **usporiada podľa ich bodov**

na ubytovanie pomocou v jazyku integrovaného skupinového dopytu (obr. 1), usporiadaný zoznam zapíše do objektu *studentsSorted* triedy *List* a vypíše ho na konzolu (obr. 11 a 12). Ostatná časť zdrojového kódu vykoná rovnakú činnosť, ako je uvedené v dvoch odstavcoch vyššie, s tým rozdielom, že študenti budú v ďalšom usporiadanom zozname, ktorý je usporiadaný pomocou volania statickej členskej metódy *QuickS_movingSort*, a v dvoch vyhľadávaniach podľa intervalov ich bodov na ubytovanie, prvé je vykonané pomocou v jazyku integrovaného skupinového dopytu (obr. 10), druhé je vykonané bez takéhoto dopytu (zdrojový kód pre takéto vyhľadávanie je zhodný so zdrojovým kódom na obr. 9), vždy usporiadaní podľa ich bodov na ubytovanie (obr. 11 a 12).

Výsledky všetkých vyhľadávaní spolu s exekučnými časmi jednotlivých vyhľadávaní metóda *Main* zobrazí na konzolu (obr. 11 a 12) a zároveň ich zapíše do logovacieho súboru *LogFile.txt*.

Obr. 10: Vyhľadanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu, ich zobrazenie na konzolu a ich zapísanie do premennej 'str_for_sw', ktorá bude zapísaná do logovacieho súboru 'LogFile.txt', v zdrojovom kóde metódy 'Main'

```
Stopwatch stopWatch = new Stopwatch();
stopWatch.Reset(); // zastavi meranie casu a resetuje uplynuty cas na 0
stopWatch.Start(); // spusti alebo obnovi meranie uplynuteho casu

var groupByPercentilePointsToAccomQuery =
from student in studentsSorted
let percentile = Student.GetPercentile(student)  // do skupiny vyberame studentov, ktori maju percentil
group new  {                  // napr. 5, napr. studenti s bodmi na ubytovanie 51, 53, 57 maju percentil 5
  student.FirstName,
  student.LastName,
  student.Points_acc,
  student.Income,
  student.Aver_grade,
  student.Birthday,
  student.Residence,
  student.Distance_resid
} by percentile into percentGroupAcc
orderby percentGroupAcc.Key
select percentGroupAcc;
// zastavi meranie uplynuteho casu v intervale
stopWatch.Stop();                  // ziska uplynuty casa ako hodnotu 'TimeSpan'
TimeSpan ts_IS = stopWatch.Elapsed;  string strIntervalStopWatch_IS = ts_IS.ToString();
double msIntervalStopWatch_IS = ts_IS.TotalMilliseconds;
                                                    . . .
foreach (var studentGroupAcc in groupByPercentilePointsToAccomQuery)  {
  Console.WriteLine($"{studentGroupAcc.Key * 10} - {studentGroupAcc.Key * 10 + 9}
                          [{studentGroupAcc.Count()}]");
  str_for_sw += $"{studentGroupAcc.Key * 10} - {studentGroupAcc.Key * 10 + 9}
                    [{studentGroupAcc.Count()}]\n";

flagConsole = 0;  k = 1;
  foreach (var item in studentGroupAcc)  {
      string year_str = "";
      if (item.Aver_grade.ToString().Count() < 4)  year_str = " " + item.Birthday.y.ToString();
      else  year_str = item.Birthday.y.ToString();
                                                . . . } }
```

Zdroj: Vlastné spracovanie

Obr. 11: Príklad vstupu (tučným písmom) a výstupu C# .NET aplikácie, ktorá vyhľadala študentov podľa intervalov ich bodov na ubytovanie (v nich ich usporiadala podľa priezviska) pomocou v jazyku integrovaného skupinového dopytu a bez jeho použitia vo vstupnom dátovom súbore 'students10c2en.txt' s 10 študentmi

```
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.

    The number of the students: 10
---- Unsorted students:

 1  35988187656170751 Gagg        Hersch    (35) 483 2,27 1999-05-21 Stamford     (169)
 2  35352737044886535 Dougill     Percival  (53) 237 2,77 2000-02-10 Joliet       (383)
 3  35150528961405232 Cashen      Shayla    (57) 566 2,74 2002-03-22 Jamaica      (164)
 4  35978397704389789 Mannin      Cchaddie  (66) 340 2,05 2001-04-17 Jacksonville (32)
 5  32630895575785021 Mancer      Willey    (61) 160 2,92 2001-04-06 Augusta      (783)
 6  32322011686544786 Gerardot    Edithe    (57) 383 1,2  2001-01-22 Topeka       (703)
 7  33931482295090410 L'Anglois   Jeana     (51) 421 2,8  2000-08-06 Boynton-Beach (289)
 8  35350197848168233 Danilovitch Roxine    (72) 627 4,66 2003-03-01 Chicago      (326)
 9  33928014847747866 Gabbidon    Guilbert  (73) 381 2,05 1999-07-23 Bloomington  (572)
10  35201185262631838 Coopper     Roslyn    (62) 570 4,68 2003-11-24 Grand-Rapids (177)


Choose sorting criterion of students:
sorting according to:
- their last names (insert: 1),
- points for an accommodation (insert: 2),
1
                The students will be sorted according to their 'LAST NAMES'.

---- The students sorted according to LAST NAMES by a LINQ query:

 1  35150528961405232 Cashen      Shayla    (57) 566 2,74 2002-03-22 Jamaica      (164)
 2  35201185262631838 Coopper     Roslyn    (62) 570 4,68 2003-11-24 Grand-Rapids (177)
 3  35350197848168233 Danilovitch Roxine    (72) 627 4,66 2003-03-01 Chicago      (326)
 4  35352737044886535 Dougill     Percival  (53) 237 2,77 2000-02-10 Joliet       (383)
 5  33928014847747866 Gabbidon    Guilbert  (73) 381 2,05 1999-07-23 Bloomington  (572)
 6  35988187656170751 Gagg        Hersch    (35) 483 2,27 1999-05-21 Stamford     (169)
 7  32322011686544786 Gerardot    Edithe    (57) 383 1,2  2001-01-22 Topeka       (703)
 8  33931482295090410 L'Anglois   Jeana     (51) 421 2,8  2000-08-06 Boynton-Beach (289)
 9  32630895575785021 Mancer      Willey    (61) 160 2,92 2001-04-06 Augusta      (783)
10  35978397704389789 Mannin      Cchaddie  (66) 340 2,05 2001-04-17 Jacksonville (32)

---- The students sorted according to RANGES of POINTS for an ACCOMODATION
---- by a LINQ group query
---- (students in groups are sorted according to last names, then according to first names)
---- Execution time: 00:00:00.0007345, 0.7345 ms:

30 - 39 [1]
      Gagg        Hersch      (35) 483  2,27  1999-05-21      Stamford        (169)
```

Zdroj: Vlastné spracovanie

Obr. 11: (Pokračovanie)

```
50 - 59 [4]
      Cashen      Shayla      (57) 566  2,74  2002-03-22      Jamaica         (164)
      Dougill     Percival    (53) 237  2,77  2000-02-10      Joliet          (383)
      Gerardot    Edithe      (57) 383  1,2   2001-01-22      Topeka          (703)
      L'Anglois   Jeana       (51) 421  2,8   2000-08-06      Boynton-Beach   (289)
60 - 69 [3]
      Coopper     Roslyn      (62) 570  4,68  2003-11-24      Grand-Rapids    (177)
      Mancer      Willey      (61) 160  2,92  2001-04-06      Augusta         (783)
      Mannin      Cchaddie    (66) 340  2,05  2001-04-17      Jacksonville    (32)
70 - 79 [2]
      Danilovitch Roxine      (72) 627  4,66  2003-03-01      Chicago         (326)
      Gabbidon    Guilbert    (73) 381  2,05  1999-07-23      Bloomington     (572)


---- The students sorted according to LAST NAMES (then according to first names)
     by the 'QuickS_movingSort' static method (moving sort in Quick Sort):

 1  35150528961405232 Cashen      Shayla      (57) 566 2,74 2002-03-22 Jamaica       (164)
 2  35201185262631838 Coopper     Roslyn      (62) 570 4,68 2003-11-24 Grand-Rapids  (177)
 3  35350197848168233 Danilovitch Roxine      (72) 627 4,66 2003-03-01 Chicago       (326)
 4  35352737044886535 Dougill     Percival    (53) 237 2,77 2000-02-10 Joliet        (383)
 5  33928014847747866 Gabbidon    Guilbert    (73) 381 2,05 1999-07-23 Bloomington   (572)
 6  35988187656170751 Gagg        Hersch      (35) 483 2,27 1999-05-21 Stamford      (169)
 7  32322011686544786 Gerardot    Edithe      (57) 383 1,2  2001-01-22 Topeka        (703)
 8  33931482295090410 L'Anglois   Jeana       (51) 421 2,8  2000-08-06 Boynton-Beach (289)
 9  32630895575785021 Mancer      Willey      (61) 160 2,92 2001-04-06 Augusta       (783)
10  35978397704389789 Mannin      Cchaddie    (66) 340 2,05 2001-04-17 Jacksonville  (32)


---- The students sorted according to RANGES of POINTS for an ACCOMODATION
---- WITHOUT using a LINQ query
---- (students in groups are sorted according to last names, then according to first names):

30 - 39
      Gagg        Hersch      (35) 483  2,27  1999-05-21      Stamford        (169)
      [1]
40 - 49
      NONE students
50 - 59
      Cashen      Shayla      (57) 566  2,74  2002-03-22      Jamaica         (164)
      Dougill     Percival    (53) 237  2,77  2000-02-10      Joliet          (383)
      Gerardot    Edithe      (57) 383  1,2   2001-01-22      Topeka          (703)
      L'Anglois   Jeana       (51) 421  2,8   2000-08-06      Boynton-Beach   (289)
      [4]
60 - 69
      Coopper     Roslyn      (62) 570  4,68  2003-11-24      Grand-Rapids    (177)
      Mancer      Willey      (61) 160  2,92  2001-04-06      Augusta         (783)
      Mannin      Cchaddie    (66) 340  2,05  2001-04-17      Jacksonville    (32)
      [3]
70 - 79
      Danilovitch Roxine      (72) 627  4,66  2003-03-01      Chicago         (326)
      Gabbidon    Guilbert    (73) 381  2,05  1999-07-23      Bloomington     (572)
      [2]
80 - 89
      NONE students
90 - 100
      NONE students
---- Execution time: 00:00:00.0135332, 13.5332 ms
```

Zdroj: Vlastné spracovanie

Obr. 12: Príklad vstupu (tučným písmom) a výstupu C# .NET aplikácie, ktorá vyhľadala študentov podľa intervalov ich bodov na ubytovanie (v nich ich usporiadala podľa ich bodov na ubytovanie) pomocou v jazyku integrovaného skupinového dopytu a bez jeho použitia vo vstupnom dátovom súbore 'students10c2en.txt' s 10 študentmi

```
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.

    The number of the students: 10
---- Unsorted students:

 1  35988187656170751 Gagg          Hersch     (35) 483 2,27 1999-05-21 Stamford     (169)
 2  35352737044886535 Dougill       Percival   (53) 237 2,77 2000-02-10 Joliet       (383)
 3  35150528961405232 Cashen        Shayla     (57) 566 2,74 2002-03-22 Jamaica      (164)
 4  35978397704389789 Mannin        Cchaddie   (66) 340 2,05 2001-04-17 Jacksonville (32)
 5  32630895575785021 Mancer        Willey     (61) 160 2,92 2001-04-06 Augusta      (783)
 6  32322011686544786 Gerardot      Edithe     (57) 383 1,2  2001-01-22 Topeka       (703)
 7  33931482295090410 L'Anglois     Jeana      (51) 421 2,8  2000-08-06 Boynton-Beach (289)
 8  35350197848168233 Danilovitch   Roxine     (72) 627 4,66 2003-03-01 Chicago      (326)
 9  33928014847747866 Gabbidon      Guilbert   (73) 381 2,05 1999-07-23 Bloomington  (572)
10  35201185262631838 Coopper       Roslyn     (62) 570 4,68 2003-11-24 Grand-Rapids (177)


Choose sorting criterion of students:
sorting according to:
- their last names (insert: 1),
- points for an accommodation (insert: 2),
2
                The students will be sorted according to 'POINTS FOR an ACCOMMODATION'.

---- The students sorted according to POINTS for an ACCOMODATION by a LINQ query:

 1  33928014847747866 Gabbidon      Guilbert   (73) 381 2,05 1999-07-23 Bloomington  (572)
 2  35350197848168233 Danilovitch   Roxine     (72) 627 4,66 2003-03-01 Chicago      (326)
 3  35978397704389789 Mannin        Cchaddie   (66) 340 2,05 2001-04-17 Jacksonville (32)
 4  35201185262631838 Coopper       Roslyn     (62) 570 4,68 2003-11-24 Grand-Rapids (177)
 5  32630895575785021 Mancer        Willey     (61) 160 2,92 2001-04-06 Augusta      (783)
 6  35150528961405232 Cashen        Shayla     (57) 566 2,74 2002-03-22 Jamaica      (164)
 7  32322011686544786 Gerardot      Edithe     (57) 383 1,2  2001-01-22 Topeka       (703)
 8  35352737044886535 Dougill       Percival   (53) 237 2,77 2000-02-10 Joliet       (383)
 9  33931482295090410 L'Anglois     Jeana      (51) 421 2,8  2000-08-06 Boynton-Beach (289)
10  35988187656170751 Gagg          Hersch     (35) 483 2,27 1999-05-21 Stamford     (169)


---- The students sorted according to RANGES of POINTS for ACCOMODATION
---- by a LINQ group query
---- (students in groups are sorted according to points for an accomodation, then according to last
      names):
---- Execution time: 00:00:00.0000998, 0.0998 ms

30 - 39 [1]
      Gagg          Hersch     (35) 483  2,27  1999-05-21       Stamford       (169)
```

Zdroj: Vlastné spracovanie

Obr. 12: (Pokračovanie)

```
50 - 59 [4]
      Cashen      Shayla      (57) 566  2,74  2002-03-22    Jamaica          (164)
      Gerardot    Edithe      (57) 383  1,2   2001-01-22    Topeka           (703)
      Dougill     Percival    (53) 237  2,77  2000-02-10    Joliet           (383)
      L'Anglois   Jeana       (51) 421  2,8   2000-08-06    Boynton-Beach    (289)
60 - 69 [3]
      Mannin      Cchaddie    (66) 340  2,05  2001-04-17    Jacksonville     (32)
      Coopper     Roslyn      (62) 570  4,68  2003-11-24    Grand-Rapids     (177)
      Mancer      Willey      (61) 160  2,92  2001-04-06    Augusta          (783)
70 - 79 [2]
      Gabbidon    Guilbert    (73) 381  2,05  1999-07-23    Bloomington      (572)
      Danilovitch Roxine      (72) 627  4,66  2003-03-01    Chicago          (326)


---- The students sorted according to POINTS for an ACCOMODATION (then according to last names)
    by the 'QuickS_movingSort' static method (moving sort in Quick Sort):

 1  33928014847747866 Gabbidon   Guilbert   (73) 381 2,05 1999-07-23  Bloomington    (572)
 2  35350197848168233 Danilovitch Roxine    (72) 627 4,66 2003-03-01  Chicago        (326)
 3  35978397704389789 Mannin     Cchaddie   (66) 340 2,05 2001-04-17  Jacksonville   (32)
 4  35201185262631838 Coopper    Roslyn     (62) 570 4,68 2003-11-24  Grand-Rapids   (177)
 5  32630895575785021 Mancer     Willey     (61) 160 2,92 2001-04-06  Augusta        (783)
 6  35150528961405232 Cashen     Shayla     (57) 566 2,74 2002-03-22  Jamaica        (164)
 7  32322011686544786 Gerardot   Edithe     (57) 383 1,2  2001-01-22  Topeka         (703)
 8  35352737044886535 Dougill    Percival   (53) 237 2,77 2000-02-10  Joliet         (383)
 9  33931482295090410 L'Anglois  Jeana      (51) 421 2,8  2000-08-06  Boynton-Beach  (289)
10  35988187656170751 Gagg       Hersch     (35) 483 2,27 1999-05-21  Stamford       (169)

---- The students sorted according to RANGES of POINTS for an ACCOMODATION
---- WITHOUT using a LINQ query
---- (students in groups are sorted according to points for an accomodation, then according to last
      names):

30 - 39
      Gagg        Hersch      (35) 483  2,27  1999-05-21    Stamford         (169)
      [1]
40 - 49
      NONE students
50 - 59
      Cashen      Shayla      (57) 566  2,74  2002-03-22    Jamaica          (164)
      Gerardot    Edithe      (57) 383  1,2   2001-01-22    Topeka           (703)
      Dougill     Percival    (53) 237  2,77  2000-02-10    Joliet           (383)
      L'Anglois   Jeana       (51) 421  2,8   2000-08-06    Boynton-Beach    (289)
      [4]
60 - 69
      Mannin      Cchaddie    (66) 340  2,05  2001-04-17    Jacksonville     (32)
      Coopper     Roslyn      (62) 570  4,68  2003-11-24    Grand-Rapids     (177)
      Mancer      Willey      (61) 160  2,92  2001-04-06    Augusta          (783)
      [3]
70 - 79
      Gabbidon    Guilbert    (73) 381  2,05  1999-07-23    Bloomington      (572)
      Danilovitch Roxine      (72) 627  4,66  2003-03-01    Chicago          (326)
      [2]
80 - 89
      NONE students
90 - 100
      NONE students
---- Execution time: 00:00:00.0113859, 11.3859 ms
```

Zdroj: Vlastné spracovanie

4 Experiment, jeho výsledky, ich krátka analýza

Účelom experimentu je potvrdiť alebo vyvrátiť našu hypotézu: predpokladáme, že vyhľadávanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku C# integrovaného skupinového dopytu je exekučne efektívnejšie ako vykonanie takéhoto vyhľadávania pomocou zdrojového kódu, ktorý nepoužíva v jazyku C# integrovaný skupinový dopyt.

Experiment sme vykonali pomocou našej C# .NET aplikácie, ktorá na vstupe načítala vstupné dáta postupne z 20 diskových súborov: *students100c2en.txt*, *students200c2en.txt*, *students300c2en.txt*, *students400c2en.txt*, *students500c2en.txt*, *students600c2en.txt*, *students700c2en.txt*, *students800c2en.txt*, *students900c2en.txt*, *students1000c2en.txt*, *students1100c2en.txt*, *students1200c2en.txt*, *students1300c2en.txt*, *students1400c2en.txt*, *students1500c2en.txt*, *students1600c2en.txt*, *students1700c2en.txt*, *students1800c2en.txt*, *students1900c2en.txt* a *students2000c2en.txt*. Tieto vstupné súbory, ako je zrejmé z ich názvov, obsahovali dáta 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900 a 2000 študentov.

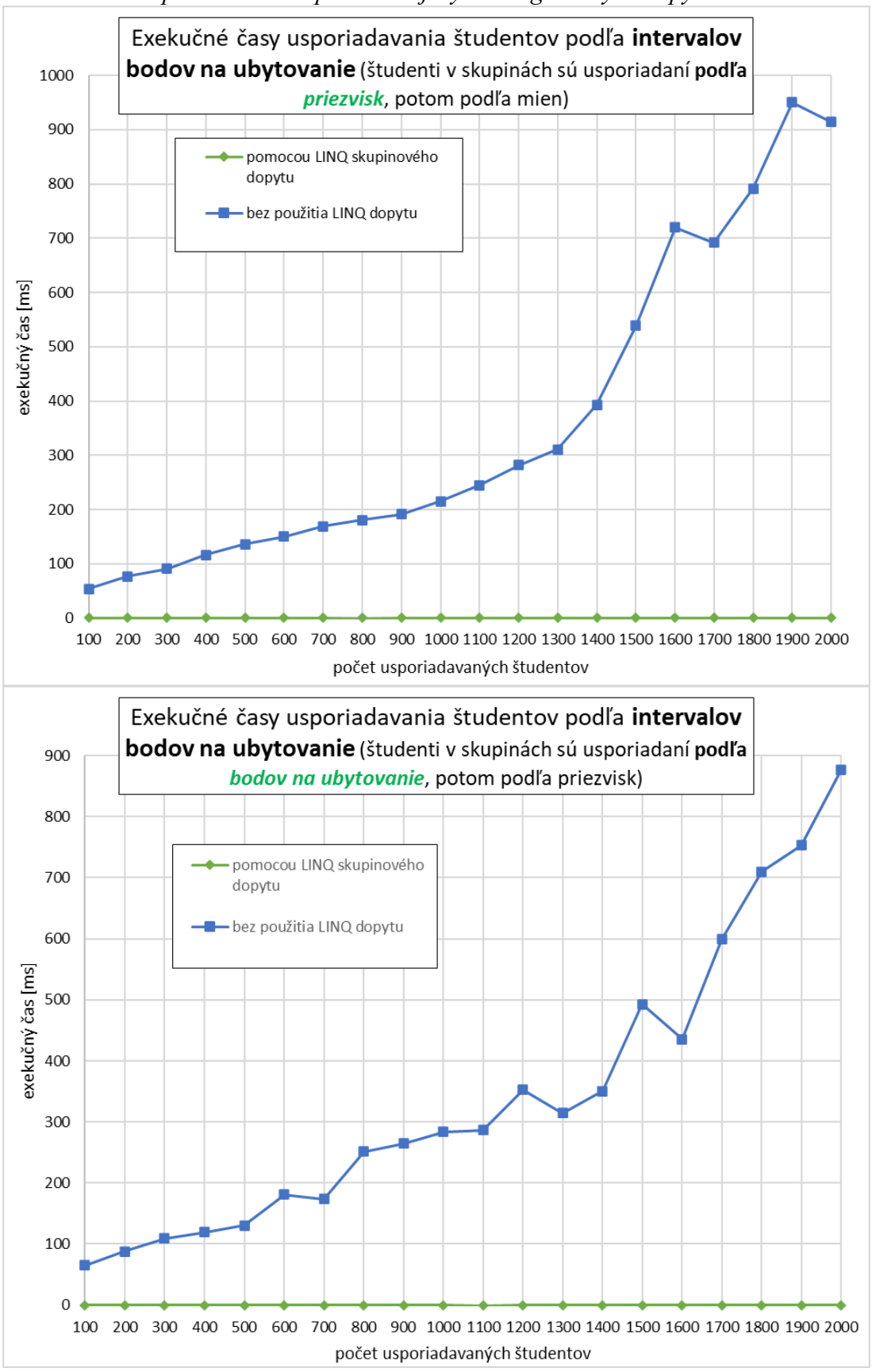
Po načítaní dát študentov z každého z uvedených vstupných diskových súborov do dátového prúdu spojeného s vybratým vstupným diskovým súborom aplikácia vykonala nasledujúce činnosti:

- zobrazila neusporiadaný zoznam všetkých študentov na konzolu,
- vykonala základné usporiadanie študentov podľa používateľom vybraného kritéria, čiže podľa priezvisk alebo podľa bodov na ubytovanie študentov,
- pomocou v jazyku integrovaného skupinového dopytu vyhľadala študentov podľa intervalov ich bodov na ubytovanie a zmerala exekučný čas tohto vyhľadávania. Študentov v jednotlivých skupinách usporiadala podľa ich priezvisk alebo podľa ich bodov na ubytovanie. Toto základné usporiadanie študentov v skupinách bolo vybraté pred vykonaním vyhľadávania používateľom.
- pomocou svojho zdrojového kódu, bez použitia v jazyku integrovaného skupinového dopytu, vyhľadala študentov podľa intervalov ich bodov na ubytovanie a zmerala exekučný čas tohto vyhľadávania. Študentov v jednotlivých skupinách usporiadala podľa ich priezvisk alebo podľa ich bodov na ubytovanie, čo bolo dané výberom používateľa pred vykonaním tohto vyhľadávania.
- výsledky všetkých vyhľadávaní spolu s exekučnými časmi jednotlivých vyhľadávaní zobrazila na konzolu (obr. 11 a 12) a zároveň ich zapísala do logovacieho súboru *LogFile.txt*.

Počas experimentu C# .NET aplikácia fungovala na počítači s nasledovnou základnou hardvérovou konfiguráciou: Intel Core i5-8250U Processor (6MB Cache, 1.60 GHz (Processor Base Frequency), 3.40 GHz (Max Turbo Frequency)), 4 GT/s (Bus Speed), 4 Cores, 8 Threads, RAM: 8 GB. Na tomto počítači bol nainštalovaný 64-bitový operačný systém Microsoft Windows 10 Home a Microsoft .NET Framework 4.

Exekučné časy všetkých uvedených vyhľadávaní sú uvedené v nasledujúcich grafoch. Exekučné časy všetkých vyhľadávaní študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaných skupinových dopytov, zobrazené v oboch grafoch, majú hodnoty menšie ako 1 ms.

Obr. 13: Exekučné časy usporiadavania študentov podľa intervalov bodov na ubytovanie s použitím a bez použitia v jazyku integrovaných dopytov



Zdroj: Vlastné spracovanie

Krátka analýza výsledkov experimentu

Z porovnania exekučných časov vyhľadávaní študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu a pomocou zdrojového kódu, ktorý nepoužíva v jazyku integrovaný skupinový dopyt, je zrejmé, že vyhľadávanie študentov pomocou v jazyku integrovaného skupinového dopytu je exekučne násobne efektívnejšie ako realizácia takéhoto vyhľadávania bez použitia v jazyku integrovaného skupinového dopytu. Dokonca, pri vyhľadávaní študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu v dátovej sade 2000 študentov je takéto vyhľadávanie 6223-krát rýchlejšie ako rovnaké vyhľadávanie v rovnakej dátovej sade, ale pomocou zdrojového kódu, ktorý nepoužíva v jazyku integrovaný skupinového dopyt. Takéto veľmi veľké zrýchlenie sme nepredpokladali. Uvedené skutočnosti potvrdzujú našu hypotézu: vyhľadávanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu je exekučne efektívnejšie ako vykonanie takéhoto vyhľadávania pomocou zdrojového kódu, ktorý nepoužíva v jazyku integrovaný skupinový dopyt.

5 Záver

Z uvedených výsledkov experimentu je zrejmé, že vyhľadávanie študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu je mnohonásobne, dokonca až 6223-násobne exekučne efektívnejšie ako vykonanie takéhoto vyhľadávania pomocou zdrojového kódu, ktorý nepoužíva v jazyku integrovaný skupinový dopyt. Vysokú exekučnú efektívnosť vyhľadávania študentov podľa intervalov ich bodov na ubytovanie pomocou v jazyku integrovaného skupinového dopytu zásadne neovplyvňuje základné usporiadanie týchto študentov pred samotným vyhľadávaním, tzn. táto vysoká exekučná efektívnosť takéhoto vyhľadávania študentov pomocou v jazyku integrovaného skupinového dopytu je nezávislá na prvotnom usporiadaní študentov.

Na základe výsledkov nášho experimentu môžeme jednoznačne odporučiť použitie v jazyku integrovaného skupinového dopytu pre skupinové vyhľadávanie študentov, napr. podľa intervalov ich bodov na ubytovanie.

Literatúra

1. Microsoft Corp. (2023). *Language Integrated Query (LINQ)*. https://learn.microsoft.com/sk-sk/dotnet/csharp/linq/.
2. Microsoft Corp. (2024a). *Introduction to LINQ Queries in C#*. https://learn.microsoft.com/sk-sk/dotnet/csharp/linq/get-started/introduction-to-linq-queries.
3. Microsoft Corp. (2024b). *Query expression basics*. https://learn.microsoft.com/sk-sk/dotnet/csharp/linq/get-started/query-expression-basics.
4. Microsoft Corp. (2024c). *Standard Query Operators Overview*. https://learn.microsoft.com/sk-sk/dotnet/csharp/linq/standard-query-operators/.