STŘEDNÍ PRŮMYSLOVÁ ŠKOLA STROJNÍ A ELEKTORTECHNICKÁ
ČESKÉ BUDĚJOVICE, DUKELSKÁ 13

Odborná maturitní práce

Řadicí algoritmy – výukový program

Autor: Radim Janda

Rok 2012-2013

Vedoucí práce: Bc. Vilém Janda

Oponent: Ing. Jan Janoud

Dokument obsahuje: Podrobnou dokumentaci celého projektu včetně důkladného rozebrání všech použitých řadicích algoritmů, výpisů programu a ukázek kódu.

Prohlášení

Prohlašuji, že jsem následující maturitní práci vypracoval sám pod vedením **Bc. Viléma Jandy**.

Všechna použitá literatura je uvedena na konci dokumentu.

- **Radim Janda ………………..**

Poděkování

Chtěl bych poděkovat panu **Ing. Janu Janoudovi** a **vedení školy**, že mi dovolili na následující maturitní práci pracovat.

Dále bych chtěl samozřejmě poděkovat vedoucímu mé práce **Bc. Vilému Jandovi**.

Obsah

1. Zadání práce

Téma maturitní práce: Řadicí algoritmy

Požadované dílčí body práce

1. Návrh implementace vybraných známých řadicích algoritmů
2. Vlastní implementace algoritmů v jazyku C#
3. Vytvoření grafického rozhraní zobrazující průběh výpočtu řadicích algoritmů
4. Vytvoření dokumentace a zpracování prezentace

Cíl hodnocení odborné práce

Vytvořit výukovou pomůcku, která vizuálně demonstruje principy výpočtu vybraných řadících algoritmů. K jednotlivým algoritmům budou také vypisovány některé důležité informace (název, slovní popis, rychlost, …)

Rozsah práce

Minimálně 15 stran textu + vývojové algoritmy + výpisy programů

Kritéria hodnocení práce

Původnost, technická úroveň návrhu a funkčnost, formální úroveň zprávy, úroveň ústní prezentace pří obhajobě práce.

Autor práce: Radim Janda

Vedoucí práce: Bc. Vilém Janda

Oponent: Ing. Jan Janoud

2. Úvod

2.1 Obecně o programu:

Program byl vytvořen v jazyce **C#** v prostředí **Microsoft Visual Studio 2012**. Visual Studio vytváří ke každému programu i spustitelný soubor čili není potřeba k jeho spuštění. Je však potřeba mít nainstalovaný Framework 4.5

Hlavní funkcí programu je grafické a teoretické znázornění sedmi nejznámějších řadících algoritmů: **Insertion Sort** (řazení vkládáním), **Bubble Sort** (bublinkové řazení), **Selection Sort** (výběrové řazení), **Shell Sort** (Shellovo řazení), **Heap Sort** (řazení haldou), **Merge Sort** (řazení slučováním) a **Quick Sort** (rychlé řazení). K co nejlepšímu pochopení principů si může uživatel sám zadat hodnoty, které bude daný algoritmus používat a nastavit rychlost tak, aby vše stíhal vnímat.

Vysvětlování principů algoritmů však není vše, co program dokáže. Další z funkcí je například zjišťování a vypisování důležitých informací ke každému algoritmu, které je následně možno uložit do **statistik**. Jedná se o informace jako je **čas seřazení**, **počet přesunů hodnot** a **počet porovnání hodnot**. **Statistika,** kam se informace ukládají, je v podstatě textový soubor, který se při otevření v programu zobrazí jako tabulka přehledně zobrazující průměrné hodnoty daných informací ke každému algoritmu. Ukládání do statistik je možno vypnout či zapnout a samotné statistiky je možno ukládat, načítat čí zcela vynulovat.

Jedním z cílů mé práce bylo **programovat na co nejlepší úrovni podle mých znalostí**. Snažil jsem se tedy napsat kód v co možná nejvyšší kvalitě:

- Celkový kód programu je přehledně rozdělen do tříd, následně do metod, které se používají i v jiných třídách.
- Proměnné jsou deklarovány pouze tam, kde se s nimi doopravdy pracuje a následně se předávají jako parametry.
- Kód je také psán tak, aby neobsahoval zbytečné příkazy a celkově zabíral co nejméně místa.
- Pomocí příkazů *try* a *catch* je předcházeno velkému množství chyb, které mohou nastat, což zamezuje pád programu.
- Také jsou zde uváděny komentáře pro celkové zlepšení přehlednosti kódu.

2.2 Mé důvody k tvorbě právě tohoto projektu:

K tvorbě tohoto projektu jsem měl několik důvodů. **Hlavní důvod** byl to, že jsem si chtěl procvičit až **zlepšit své znalosti jak v jazyce C#, tak v programování a logickém myšlení obecně**. Zkušenosti, které jsem získal při pracování na tomto projektu, jsou pro mě opravdu cenné, jelikož chci pokračovat ve studiu se zaměřením na programování a v budoucnu se třeba i ucházet o práci programátora.

Na řadící algoritmy jsem se zaměřil právě proto, že se o různé algoritmy zajímám již delší dobu a baví mě. Také jsem nevěděl o žádné jiné příručce, která by dané řadící algoritmy vysvětlovala tak kvalitně a podrobně jako můj program. Proto jsem usoudil, že jeho tvorba má i jistý smysl a třeba se pro něj jednou najde i nějaké reálné využití.

Dalším pozitivem je i to, že **vysvětlované algoritmy se budou probírat na většině vysokých škol, kam se hlásím já i moji spolužáci**. Takže jsem se již dopředu naučil jednu látku a věřím, že můj program pomůže pochopit dané algoritmy i některým mým spolužákům.

Jelikož se o programování zajímám již řadu let, měl jsem už nějaké zkušenosti jak s jazykem C# tak i s nižšími programovacími jazyky, se kterými jsem pracoval už i na prvním stupni základní školy, proto jsem si s realizací projektu věřil.

Oporou mi byl také můj starší bratr **Bc. Vilém Janda**, který mě kdysi učil programovat a dokonce se mi nabídl i jako vedoucí projektu, což jsem samozřejmě s radostí přijal.

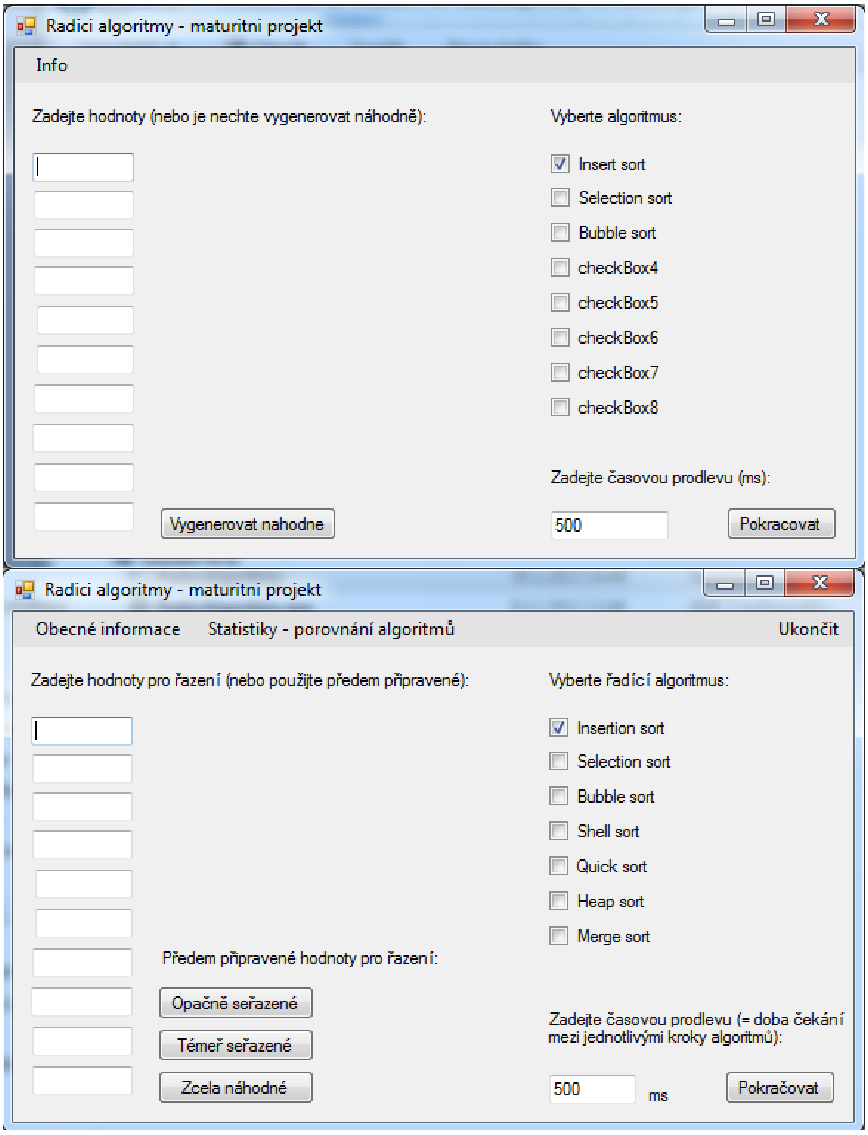
Jednalo se o nejrozsáhlejší projekt, do kterého jsem se kdy měl pustit a nebýt těchto faktorů, tak bych si na něj nejspíše netroufl.

Nakonec jsem však opravdu velmi rád, že jsem si projekt vybral, jelikož jsem získal opravdu velké množství zkušeností, které se mi budou v budoucnu hodit

3. Postup sestavování projektu

3.1 Návrh:

Nejprve jsem si navrhl, jak zhruba bude celý program vypadat. Původní návrh byl **sestrojit hlavní formu, ve které bude k výběru několik algoritmů a možnost zapsat vlastní hodnoty, se kterými bude algoritmus pracovat**. Z této formy se pak měla spouštět všechna další zobrazení algoritmů a různé další formy s informacemi. **Tohoto původního návrhu jsem se držel po celou dobu sestavování programu**, ale i přesto, že se původní návrh víceméně nezměnil, **byl program neustále obohacován** o další dodatky, které mě v průběhu tvorby napadli (**např. statistiky**) k tomu ale až později.



Zadejte hodnoty (nebo je nechte vygenerovat náhodně):

Vyberte algoritmus:

☑ Insert sort
☐ Selection sort
☐ Bubble sort
☐ checkBox4
☐ checkBox5
☐ checkBox6
☐ checkBox7
☐ checkBox8

Zadejte časovou prodlevu (ms): 500

Zadejte hodnoty pro řazení (nebo použijte předem připravené):

Vyberte řadící algoritmus:

☑ Insertion sort
☐ Selection sort
☐ Bubble sort
☐ Shell sort
☐ Quick sort
☐ Heap sort
☐ Merge sort

Předem připravené hodnoty pro řazení:

Zadejte časovou prodlevu (= doba čekání mezi jednotlivými kroky algoritmů): 500 ms

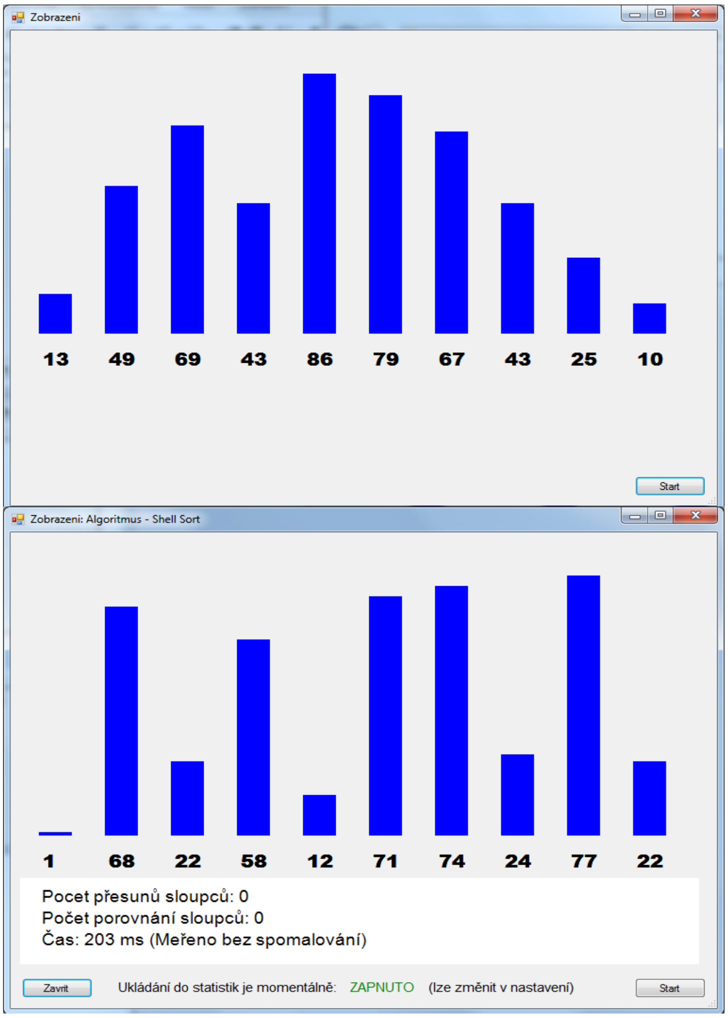
Na obrázku můžeme vidět, jak se vyvíjela hlavní forma. (starší verze – nahoře, novější verze – dole). Lze vidět, že v hlavní formě jsou změny pouze nepatrné, ale celkově vývoj programu značně pokročil.

3.2 Grafické zobrazení:

Jako první věc, která byla v programu opravdu funkční, bylo vykreslení sloupců ze zadaných hodnot. Sloupce znázorňují velikost hodnot a můžou být vykreslovány ze širokého spektra přirozených čísel. **Vykreslují se tak, že se vypočítá konstanta K = (největší číslo / maximální výška sloupce)**, podle které se pak bude odvíjet výška všech sloupců. **Je tedy zřejmé, že největší číslo bude mít nejvyšší sloupec a číslo, které je třeba o polovinu menší, bude mít rovněž o polovinu nižší i sloupec**.

Vykreslení sloupců bylo nastaveno jako událost *Paint*, která proběhne vždy při zavolání metody *Refresh()* a v několika dalších případech. Metodu pro vykreslení pak z vlastní třídy vyvolávají algoritmy po každém kroku. Hodnoty a barvy sloupce jsou obsaženy v polích.

Později bylo grafické zobrazení značně vylepšeno (přidání šipky, informace, atd) a následně i teoretické informace před spuštěním samotného grafického zobrazení.



Pocet přesunů sloupců: 0
Počet porovnání sloupců: 0
Čas: 203 ms (Meřeno bez spomalování)

Ukládání do statistik je momentálně: ZAPNUTO (lze změnit v nastavení)

Na obrázku můžeme vidět grafické zobrazení ještě před spuštěním algoritmu. (starší verze – nahoře, novější verze – dole)

3.3 Programování algoritmů

Když už bylo hotové vše, co bylo potřeba k zobrazení algoritmů, vrhnul jsem se na samotné algoritmy. Každý algoritmus je ve vlastní třídě a odtud mění obsahy polí, které jsou následně vykreslovány. Algoritmy jsem programoval vlastním způsobem, věděl jsem jak fungují pouze teoreticky.

Jako první jsem naprogramoval algoritmus **Insertion Sort**; tento algoritmus je poměrně jednoduchý a naprogramovat ho nebylo nijak zvlášť těžké. Hned po několika prvních pokusech už dělal víceméně to, co jsem zamýšlel, což mě mile překvapilo.

Problém nebyl ani s algoritmy **Bubble Sort** a **Selection Sort**, které byly také opravdu velmi jednoduché

Tak snadné už to však nebylo s dalšími algoritmy. Ty jsou totiž poněkud složitější a dalo mi velkou práci pochopit, jak přesně fungují, abych je mohl sestavit co nejlépe.

Ačkoli co nejlepší pochopení algoritmů **Shell Sort** a **Qucik Sort** mi zabralo poměrně dost času, jejich naprogramování už těžké nebylo.

Větší problém nastal až s algoritmem **Heap Sort,** kde se hodnoty řadí v haldě. Tuto haldu jsem každopádně musel někde znázornit, ale nechtěl jsem, aby se zobrazení tohoto algoritmu nějak výrazně odlišovalo od zobrazení ostatních algoritmů. Rozhodl jsem se tedy danou haldu znázornit v teoretickém rozboru algoritmu (před spuštěním grafického zobrazení) a grafické zobrazení ponechat ve standardním stylu.

Stejně tak to bylo i s algoritmem **Merge Sort**, který se od ostatních algoritmů liší tím, že hodnoty nepřesouvá, ale ukládá do pomocného pole, ze kterého je pak vrátí do pole hlavního. V tomto případě jsem grafické znázornění vyřešil úpravou šipky, která místo toho aby ukazovala na prohazované sloupce, ukazuje na sloupec, který je právě dáván do pomocného pole a vypíše k němu pořadí v tomto pomocném poli.

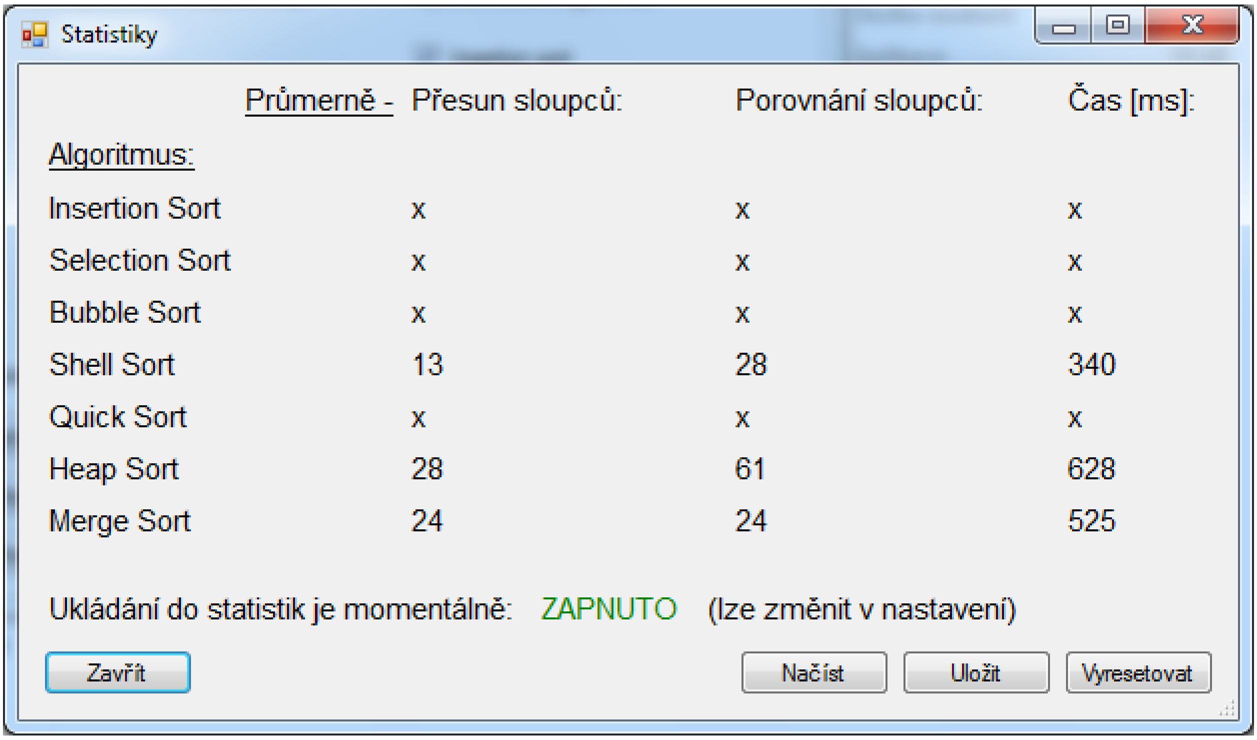
Podrobnější informace ke každému algoritmu a o tom, jak přesně je naprogramovaný, naleznete na stránkách 12 – 24.

3.4 Statistiky

Statistiky jsou jedno z posledních vylepšení, které jsem do programu přidal. Jejich účelem je podrobné zobrazení zobrazovaných informací ke každému algoritmu a tím pádem i možnost porovnání algoritmů.

Při tvorbě statistik jsem se snažil být opravdu důkladný. Nejen, že jsou ukládány do textového souboru a tím pádem jsou zachovány i po vypnutí programu, ale statistiky se mohou ukládat, načítat a další.

Celkově jsou statistiky zcela nová forma, která osahuje několik labelů, kde se vypisují informace a několik tlačítek. (viz přiložený obrázek)



Průmerně -	Přesun sloupců:	Porovnání sloupců:	Čas [ms]:
Algoritmus:			
Insertion Sort	x	x	x
Selection Sort	x	x	x
Bubble Sort	x	x	x
Shell Sort	13	28	340
Quick Sort	x	x	x
Heap Sort	28	61	628
Merge Sort	24	24	525

Ukládání do statistik je momentálně: ZAPNUTO (lze změnit v nastavení)

Fungují však poměrně jednoduše. Všechno ukládání/načítání statistik do textového souboru je uděláno pomocí stream writerů/leaderů. Každý textový soubor se skládá z 1+7*3 řádků s tím, že první řádek je úvodní a další tři sedmice řádků uchovávají všechny 3 informace ke každému algoritmu. Prvních 7 řádků = přesun sloupců, druhých 7 řádků = porovnání sloupců a třetích 7 řádků = čas.

Při každém projetí algoritmu se v případě, že je zapnuto ukládání statistik, tyto informace připíšou do příslušného řádku a následně jsou v zobrazení zprůměrovány.

Čtení z textového souboru probíhá po jednotlivém znaku. Každá hodnota je oddělena středníkem. Je také ošetřeno, pokud by v textovém souboru byla i nějaká chyba. V takovémto případě jste na počet chyb v textovém souboru upozorněni a je vám nabídnuto vyresetování statistik (vynulování).

Ukládání statistik v podstatě zkopíruje současně načtené statistiky do zvolené oblastí. Načtení zase přepíše současně načtené statistiky statistikami ze zvolené oblasti. V případě, že nechceme současně načtené statistiky přepsat, je nutné si je nejprve uložit.

Při programování statistik jsem se snažil postupovat co nejprofesionálněji. Můžete si tedy všimnout, že pokud chcete současné statistiky přepsat nebo vyresetovat, jste nejprve upozorněni na to, že o současné statistiky přijdete a musíte celou akci potvrdit. Také, jak už jsem zmínil, jste i upozorňováni na chyby v textovém souboru.

3.5 Použité prostředí

Program jsem vytvořil v prostředí **Visual Studio Express 2012**. Jedná se o verzi, které není placená, je pro ni ale nutná registrace. Pro můj projekt však plně stačila.

Toto prostředí je poměrně hodně přátelské k uživateli a jelikož s ním dělám již delší dobu, tak se mi s ním pracovalo zcela bez problémů.

Dá se zde i poměrně dobře zorientovat. Na začátku je nutné vytvořit si projekt a to kliknutím na nabídku New Project. V této nabídce si vybereme jazyk, ve kterém chceme programovat; v mém případě (C#) a dále si vybereme typ – např: Windows Form Application (v mém případě), nebo Console Application atd.

Windows Form Application jsem si vybral, protože jsem potřeboval sestavit již zmiňovanou hlavní formu a v tomto režimu je to velice snadné. Dále jsem do programu přidal několik tříd – každý algoritmus má vlastní třídu, ve které je obsažen jeho kód a třídu getinfo, která obsahuje veškeré teoretické informace. Třídy jsou zde zejména kvůli co nejpřehlednějšímu a nejefektivnějšímu sestavení kódu.

Kromě tříd jsem přidal ještě 4 další formy a to pro zobrazení, statistiku, nastavení statistik a úvodní zprávu.

K vytvoření dokumentace jsem použil **Microsoft World 2010**.

4. Použité řadící algoritmy

Ke všem použitým řadícím algoritmům jsem sestavil **obecný a podrobný popis jejich principu**. Tyto popisy jsou také zobrazeny v programu vždy před spuštěním grafického zobrazené.

U prvních třech jednoduchých algoritmům jsem také **sestavil vývojové diagramy**. Tyto diagramy jsem však oproti svému kódu nepatrně **zjednodušil, aby byly lépe pochopitelné.** K dalším algoritmům jsem vývojové diagramy již nesestavoval, jelikož by byly zbytečně složité a nepřehledné.

V následujícím podrobném popisu všech algoritmů uvidíme také **mou realizace algoritmů v jazyce C#**. Každý kód algoritmu v sobě má také příkazy, které slouží k zobrazení či počítání statistik. Tyto příkazy sice nepatrně zpomalují chod samotného algoritmu, ale jelikož nám jde hlavně o srovnání algoritmů a zpomalení je opravdu minimální, lze tyto příkazy zanedbat.
Z ukázkového kódu jsem všechny tyto příkazy odstranil a to hlavně kvůli zlepšení přehlednosti.

4.1 Insertion Sort (Řazení vkládáním)

Řazení vkládáním je jeden z nejjednodušších algoritmů pracujících na principu porovnávání. Je také velmi rychlý, zejména při řazení již téměř seřazených řad.

Princip algoritmu:

Algoritmus postupně projíždí celou řadu hodnot a každou hodnotu bude přesunovat do leva tak dlouho, dokud nenarazí na hodnotu nižší, než je právě přesunovaná hodnota.
Následně se vrátí na místo poslední neseřazené hodnoty; děj se opakuje, dokud se neseřadí všechny hodnoty.
V tomto případě budou tedy hodnoty nakonec seřazeny zleva (od nejmenší) doprava (do největší).

Moje realizace algoritmu v jazyku C#:

```
int[] cislo = new int[10];
int sloupec = 0;
int sloupecp = 0;
int sloupecs = 0;
while (sloupec < 10)
    // Pohyb hlavniho ukazatele
{
    sloupecp = sloupec;
    sloupecs = sloupec;
    while (sloupecp > 0)
        // Presun cisla do leva tak dlouho dokud je před nim větší číslo
```

```
          // popř. ukončení pokud je číslo úplně na začátku
        {
          sloupecp = (sloupecp - 1);
          if (cislo[sloupecp] > cislo[sloupecs])
          {   // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot
(cislo[a], cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
            form.Swap(ref cislo[sloupecp], ref cislo[sloupecs], sloupecp,
sloupecs);
            sloupecs = (sloupecp);
          }
          else
          {
            sloupecp = 0;
          }
        }
        sloupec = (sloupec + 1);
      }
```

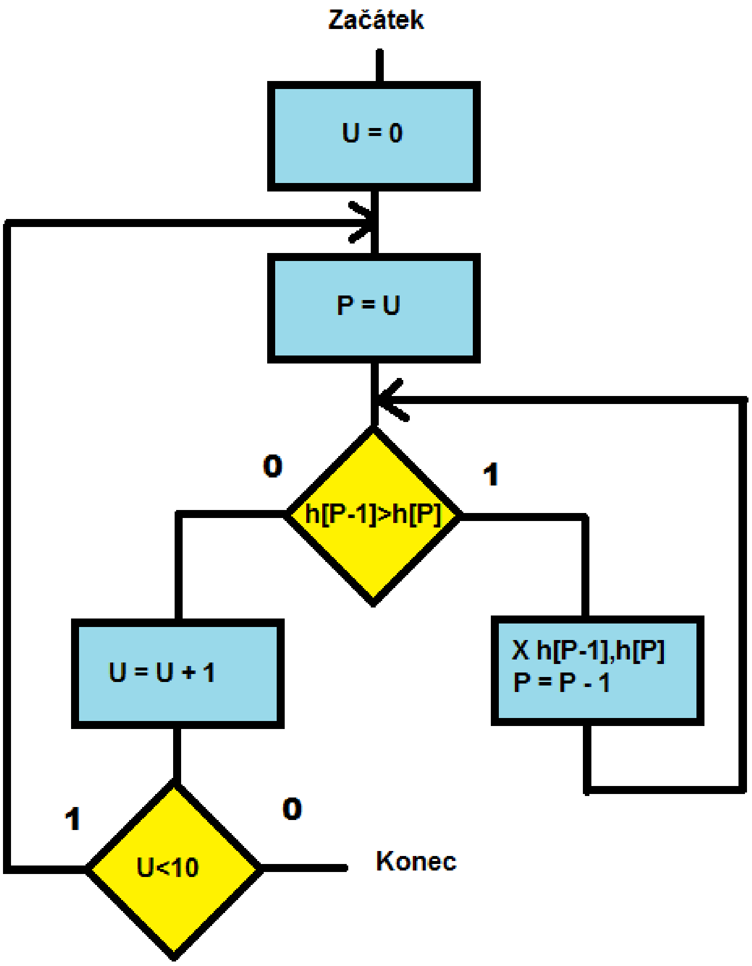
Vývojový diagram (pro hodnoty značené 0 až 9):

U = **Hlavní ukazatel,** od kterého bude přesun probíhat.

P = **Ukazatel** (nebo v případě pole index) **přesunu** čili **h[P] bude hodnota, na kterou ukazuje**

X h[P-1],h[P] - znamená **prohození** hodnot **h[P-1]** a **h[P].**

Vývojový diagram jsem zde lehce zjednodušil, protože v tomto případě nesmí být záporný index pole, tedy hodnota h[-1] bude nulová.



4.2 Selection Sort (Řazení výběrem)

Řazení výběrem je velmi jednoduchý řadící algoritmus, pracující na principu projetí celé řady hodnot a vybrání nejnižší hodnoty.

Jeho nejvíce efektivní použití je u zcela neseřazených hodnot malého počtu.

Princip algoritmu:

Algoritmus postupně projíždí celou řadu hodnot vždy od první neseřazené hodnoty (zleva) a hledá jinou nejvhodnější hodnotu (v tomto případě co nejmenší).

Pakliže není nalezena menší hodnota, tak se počáteční hodnota stane seřazenou.

Pakliže je nalezeno více menších hodnot, vybere se ta nejmenší a prohodí se s počáteční hodnotou.

S takto seřazenými hodnotami se již dále nijak nepracuje a algoritmus opět pokračuje od další první neseřazené hodnoty, dokud nejsou všechny hodnoty seřazeny.

Moje realizace algoritmu v jazyku C#:

```
            int[] cislo = new int[10];
            int sloupecmin = 0;
                // Do teto promene se zapisuje hodnota minimálního sloupce
            int sloupecminnm = 0;
                // Do teto promene se zapisuje cislo minimalniho sloupce

            for (int i = 0; i < 10; i++)
            {
                sloupecmin = cislo[i];
                sloupecminnm = i;
                for (int ik = i+1; ik < 10; ik++)
                {
                    if (cislo[ik] < sloupecmin)
                        // Zápis nejnižší hodnoty
                    {
                        sloupecmin = cislo[ik];
                        sloupecminnm = ik;
                    }
                } // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot (cislo[a],
cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
                form.Swap(ref cislo[i], ref cislo[sloupecminnm], i, sloupecminnm);
            }
```

Vývojový diagram (pro hodnoty značené 0 až 9):

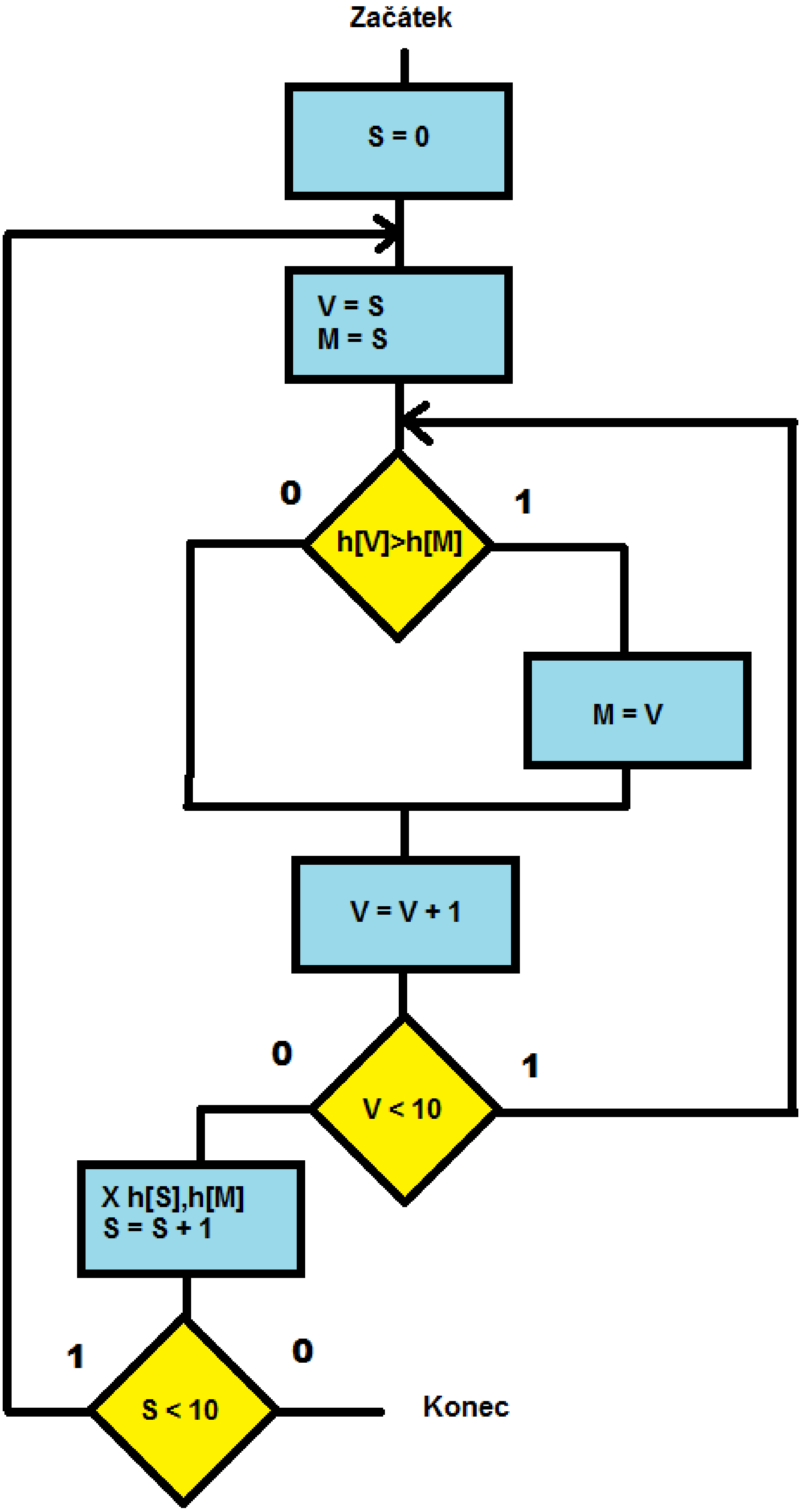
S = **Ukazatel** na **poslední seřazenou hodnotu**.

V = **Ukazatel** (nebo v případě pole index) **výběru** čili **h[V] bude hodnota, na kterou ukazuje**

M = **Ukazatel** (nebo v případě pole index) **maximální hodnoty** čili **h[M] bude hodnota, na kterou ukazuje**

X h[S],h[M] znamená **prohození** hodnot h[S] a h[M].

Vývojový diagram jsem zde lehce oproti svému kódu zjednodušil, v mém programu mám ukazatele i jeho hodnotu rozdělené zvlášť do dvou proměnných.



Vývojový diagram – Selection Sort

4.3 Bubble Sort (Bublinkové řazení)

Řazení výběrem je dle mého názoru též jednoduchý řadící algoritmus, pracující podobně jako Řazení výběrem.

Algoritmus projede celou řadu hodnot tolikrát, kolik je zde použitých hodnot a při každém projetí přesouvá nejmenší hodnoty na začátek.

Jeho nejvíce efektivní použití je u zcela neseřazených hodnot malého počtu.

Princip algoritmu:

Algoritmus postupně projíždí celou řadu hodnot od konce (vpravo) až do poslední neseřazené hodnoty (vlevo). Při projíždění přesouvá nejnižší hodnotu, na kterou narazil (v případě, že narazí na nižší hodnotu, tak ponechá tuto současnou a bude přesouvat hodnotu nejnižší).

Jakmile se dostane na konec či k již seřazené hodnotě, seřadí přesouvanou hodnotu a jede opět od konce.

Je tedy zřejmé, že algoritmus celou řadu projede tolikrát, kolik je zde hodnot.

Moje realizace algoritmu v jazyku C#:

```
            int[] cislo = new int[10];
            for (int ik = 0; ik < 10; ik++)     // Posun zeleneho ukazatele
            {
                for (int i = 9; i > ik; i--)    // Posun cerveneho ukazatele
                {
                    if (cislo[i] < cislo[i - 1])    // Pripadne prohozeni sloupcu
                    {   // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot
(cislo[a], cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
                        form.Swap(ref cislo[i], ref cislo[i - 1], i, i - 1);
                    }

                }
            }
```

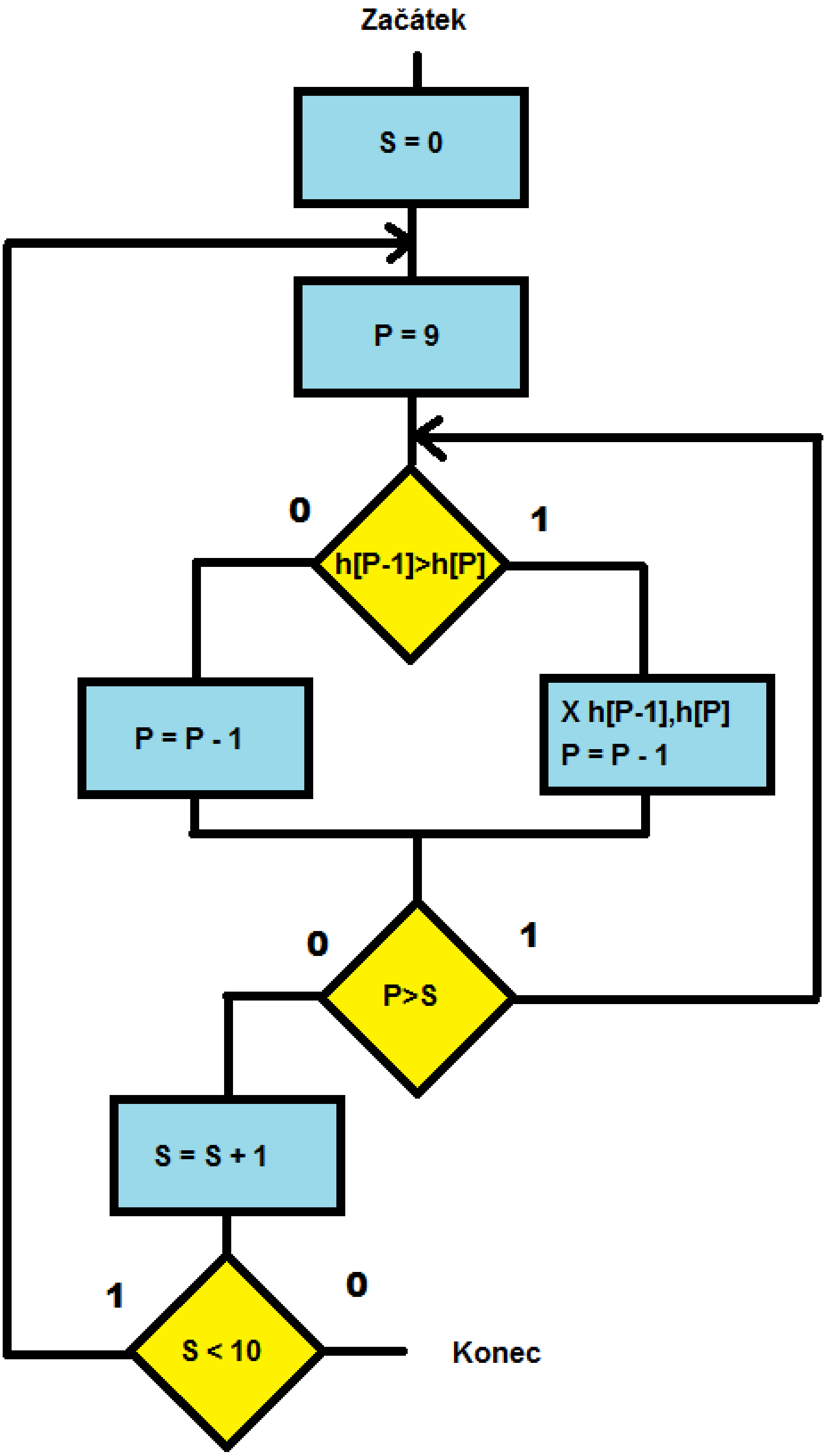
Vývojový diagram (pro hodnoty značené 0 až 9):

U = **Hlavní ukazate,l** od kterého bude přesun probíhat.

P = **Ukazatel** (nebo v případě pole index) **přesunu** čili **h[P] bude hodnota, na kterou ukazuje**

X h[P-1],h[P] - znamená **prohození** hodnot h[P-1] a h[P].

Zde jsem nemusel vývojový diagram nijak zvlášť zjednodušovat, jelikož samotný kód algoritmu se mi podařilo udělat velice krátce a efektivně.



Vývojový diagram – Bubble Sort

4.4 Shell Sort (Shellovo řazení)

Shellovo řazení je známý algoritmus, který byl pojmenován po svém vynálezci. Základem algoritmu jsou dva procházející ukazatele, které si mezi sebou prohazují hodnoty a posouvají se společně s určitým rozestupem (inkrement), jenž se v průběhu zmenšuje.

Princip algoritmu:

Celou řadu hodnot prochází zleva doprava dva ukazatelé s tím, že jeden ukazatel je posunut doprava o jistý inkrement (=počet sloupců / 2,2).

Ukazatelé si mezi sebou neustále porovnávají sloupce a přesunují se doprava. Pokud je hodnota v pravém ukazateli menší než hodnota v levém ukazateli, hodnoty se prohodí a je-li to možné, pravý ukazatel se přesune o inkrement doleva k levému ukazateli, kde opět porovnává hodnoty s pravým sloupcem. Takto se bude postupovat, dokud se hodnota nepřesune co nejvíce doleva, poté se ukazatelé vrátí na místa, kde byla hodnota poprvé přesunuta.

Následně se ukazatelé opět přesunují doprava, dokud se pravý ukazatel nepřesune na konec. Jakmile se tak stane, je inkrement opět vydělen číslem 2,2 a celá situace se opakuje s nižším inkrementem, dokud není inkrement nulový.

Moje realizace algoritmu v jazyku C#:

```
            int pocets = 10; int ic = 0; int hop = 0; int last = 0;
            int[] cislo = new int[10];
            ic = Convert.ToInt32(pocets / 2.2);
                // Vypočtení inkrementu (počet sloupců)/2.2, převod do Int
            while (ic > 0)
                // Program bude postupovat tak dlouho dokud není inkrement 0
            {
                for (int i = ic; i < pocets; i ++)
                    // Pohyb pravého ukazatele
                {
                    hop = 1;    // Určuje počet skoků vlevo (násobí se s inkrementem)
                    last = i;   // Pomocná proměná k dočasnému uchování porovnávané hodnoty
                    while ( (i-(ic*hop)) >= 0) // Levý ukazatel, porovnávání
                                            {
                        if (cislo[i - (ic * hop)] > cislo[last])
                        {   // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot
(cislo[a], cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
                            form.Swap(ref cislo[i - (ic * hop)], ref cislo[last], i - (ic *
hop), last);
                            last = (i - (ic * hop));
                        }
                        else
                        {
                            hop = i;    // Ukončení cyklu
                        }
                        hop = hop + 1;
                    }
```

```
                }
                ic = Convert.ToInt32(ic / 2.2);
}
         // Hodnoty inkrementů  budou 5 -> int(5/2.2) = 2 -> int(2/2.2) = 1 -> int(1/2,2) = 0
```

4.5 Quick Sort (Rychlé řazení)

Rychlé řazení je jeden z nejrychlejších řadicích algoritmů. Základem jsou dva přibližující se ukazatele, které si mezi sebou prohazují hodnoty a tam, kde se střetnou, bude hodnota zcela jistě seřazena.

Princip algoritmu:

Celou řadu hodnot prochází dva ukazatelé. Levý ukazatel začíná co nejvíce vlevo (buď na okraji nebo zprava od seřazené hodnoty) a pravý ukazatel začíná zleva od nejblíže seřazené hodnoty (tam kde je to možné), případně na pravém okraji.

Děj začíná tak, že se pravý ukazatel přibližuje k levému a porovnávají si mezi sebou hodnoty. Jakmile je hodnota v pravém ukazateli nižší než ta v levém, hodnoty jsou prohozeny a místo pravého ukazatele se bude přibližovat ten levý.

Po dalším prohození hodnot se zase přibližuje opačný ukazatel a takto algoritmus postupuje, dokud se ukazatelé nestřetnou.

Tam, kde se ukazatele střetnou, bude seřazená hodnota. Dále ukazatele postupují stejně jako na počátku, dokud nebudou seřazeny všechny hodnoty.

Moje realizace algoritmu v jazyku C#:

```
int ready = 0;
int[] cislo = new int[10];
int a;  // urcuje levy index razeni
int b;  // urcuje pravy index razeni
int i;  // pomocna promena pri zjistovani indexu
private int[] cislordy = new int[10];

while (ready == 0)
{
    // Nastaveni leveho indexu razeni
    a = -1;
    i = 0;
    while (a == -1 && i < 10 && ready == 0)
    {
        if (cislordy[i] == 0)
            a = i ;
        else
            i = i + 1;

    }

    // Nastaveni praveho indexu razeni
    b = -1;
    while (b == -1 && i < 11 && ready == 0 && i >= 0)
```

```
                {
                    i = i + 1;
                    if (i > 9)
                    {
                        b = 9;
                    }
                    else
                    {
                        if (cislordy[i] == 1)
                            b = i - 1;
                    }
                }

                // Skontroluje spravnost indexu a spusti razeni
                if (a != -1 && b != -1)
                {
                    // Spusti razeni s danymi indexi a zapise vysledek
                    int x = sort(a, b);     // sort(levy index, pravy index)
                    cislordy[x] = 1;
                }
                else
                    ready = 1;
            }




        private int sort(int lindex, int rindex)
        {
            while (lindex != rindex)
            {
                // Pohyb praveho praveho ukazatele

                while (cislo[lindex] <= cislo[rindex] && lindex != rindex)
                {
                    rindex = rindex - 1;
                }
                // Pripadne prohozeni sloupcu
                if (lindex != rindex)
                   // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot
(cislo[a], cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
                    form.Swap(ref cislo[lindex], ref cislo[rindex], lindex, rindex);

                // Pohyb leveho ukazatele

                while (cislo[lindex] <= cislo[rindex] && lindex != rindex)
                {
                    lindex = lindex + 1;
                }
                // Pripadne prohozeni sloupcu
                if (lindex != rindex)
                    form.Swap(ref cislo[lindex], ref cislo[rindex], lindex, rindex);

            }
            return (lindex);
        }
```

Nastavení ukončení (ready = 1) a další práce s polem cislordy[] se také provádí ve funkci, která má na starost vybarvování sloupců.

4.6 Heap Sort (Řazení haldou)

Řazení haldou je známý řadící algoritmus, který si dané hodnoty roztřídí do jakési haldy, ve které pak probíhá řazení.

Tvar haldy:

Pro pochopení algoritmu je nutné si nejprve představit danou haldu.
Zkratka S.číslo představuje hodnoty (sloupce) podle toho, jak jsou seřazeny (zleva doprava).

```
              (S.1)
              /     \
         (S.2)      (S.3)
          /    \       \     \
    (S.4)  (S.5)  (S.6) (S.7)
     /     \       \
(S.8)  (S.9)   (S.10)
```

Princip algoritmu:

Když se podíváme na haldu, vidíme, že čísla sloupců přiřazená k vyššímu sloupci, lze spočítat velmi snadno:
- Sloupec vlevo dole = 2*čislo vyššího sloupce
- Sloupec vpravo dole = 2*čislo vyššího sloupce + 1

Samotné řazení pak probíhá v těchto spojích tak, že se hodnoty porovnávají a nejvyšší se přesouvají co nejvíce nahoru. Jakmile se v této haldě uspořádají všechny hodnoty, prohodí se poslední sloupec s prvním. Sloupec, který byl dán na poslední místo (= sloupec s největší hodnotou), je pak seřazen a z celé haldy mizí.
Takto řazení postupuje i nadále, dokud nebudou seřazeny všechny hodnoty.

Moje realizace algoritmu v jazyku C#:

```
            int lastrdy = 0;
            int[] cislo = new int[10];
            int b;          // Pomocna promena k zapisovani vysledku
            int ik = 5;     // Promena urcujici posledni sloupec ktery k sobe ma prirazeny 
alespon 1 dalsi sloupec
            int k = 5;      // Pomocna promena urcujici ke kteremu sloupci pripada jen 1 
sloupec

            while (lastrdy > 0)
            {
                for (int i = 0; i < ik; i++)
                {
                    b = cmp3(i,k);
                    while (b != -1)
                    {
                        int c = b;  // Pomocna promena k zapisovani vysledku
                        b = cmp3(c,k);
                    }

                }
                if (ik == k)
                {
                    ik = ik - 1;
                }
                else
```

```
                {
                    k = k - 1;
                }
                form.Swap(ref cislo[0], ref cislo[lastrdy - 1], 0, lastrdy - 1);

                lastrdy = lastrdy - 1;
            }




        private int cmp3(int a, int b)
            // Tato metoda porovna a popripadne prohodi prirazene sloupce vzhedem k sloupci 
vybranemu (int a)
            // Sloupce musi byt cislovany od 1
            // V takovemto pripade budou prirazene sloupce   1 -> 2*a     2 -> 2*a+1
        {
            a = a + 1;
            if (a < 1)
                return (-1);

            if (a == b)     // V pripade ze ke sloupci A pripada jen jeden sloupec
            {

                if (cislo[a - 1] < cislo[a + a - 1])
                {   // form.Swap(cislo[a], cislo[b], a, b) = prohození daných hodnot 
(cislo[a], cislo[b]), nikoliv však indexů, které jsou jako parametry z grafických důvodů
                    form.Swap(ref cislo[a - 1], ref cislo[a + a - 1], a - 1, a + a - 1);
                    return ((a/2)-1);
                }
            }
            else            // V pripade ze ke sloupci A pripadaji 2 sloupce
            {
                if (cislo [a + a - 1] < cislo[a + a])
                {
                    if (cislo[a + a] > cislo[a - 1])
                    {
                        form.Swap(ref cislo[a + a], ref cislo[a - 1], a + a, a - 1);
                        return((a/2)-1);
                    }
                }
                else
                {
                    if (cislo[a + a - 1] > cislo[a - 1])
                    {
                        form.Swap(ref cislo[a + a - 1], ref cislo[a - 1], a + a - 1, a - 1);
                        return((a/2)-1);
                    }
                }

            }
            return (-1);
        }
```

4.7 Merge Sort (Řazení slučováním)

Řazení slučováním je další známý řadicí algoritmus, který nefunguje na principu prohazování hodnot, ale využívá pomocného pole, do kterého jsou hodnoty seřazovány a následně vráceny do hlavního pole.

Princip algoritmu:

Algoritmus si nejprve hodnoty neustále rozděluje na poloviny, dokud není každá hodnota samostatně rozdělena. Následně začíná od hodnot, které byly rozděleny jako poslední.
V každé rozdělené částí pak mezi sebou porovná hodnoty a uloží je seřazené do pomocného pole.
Z tohoto pomocného pole je po seřazení vrátí do pole hlavního (vykreslí tedy danou část již seřazenou).
Takto postupuje do dalších rozdělených částí, dokud neseřadí všechny.

Moje realizace algoritmu v jazyku C#:

```
int[] cislo = new int[10];
int[] cislo2 = new int[10];

// Postupně spouští řazení algoritmu tak jak byli hodnoty rozděleny
sort(3, 4, 5);
sort(2, 3, 5);
sort(0, 1, 2);
sort(0, 2, 5);
sort(8, 9, 10);
sort(7, 8, 10);
sort(5, 6, 7);
sort(5, 7, 10);
sort(0, 5, 10);
```

Je vidět, že zde jsem použil jednoduchou variantu „ručního“ vyvolávání funkce sort(), která sice bude fungovat jen pro daný počet hodnot (10), ale pro můj projekt to postačí a funguje zcela bezproblémově.

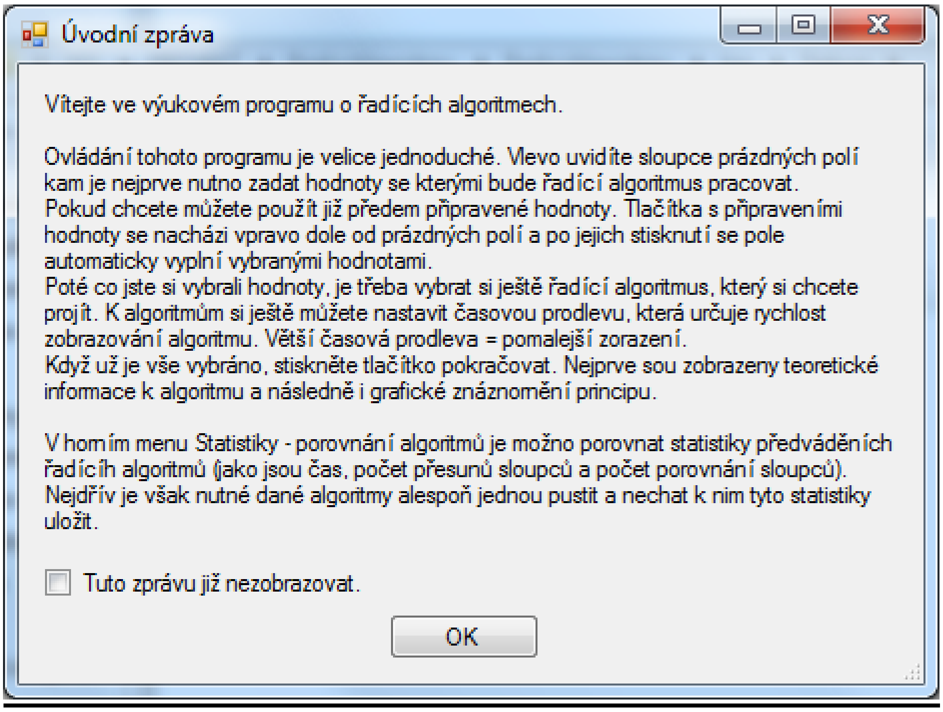
Funkci sort(), která je zde spouštěna, můžeme vidět na další straně.

```
private void sort(int a, int b, int end)
{
    int start = a;
    int enda = b;
    int move = a;

    while (a < enda | b < end)
    {
        if (a == enda | b == end)
        {
            if (a == enda)
            {
                while (b < end)
                {
                    cislo2[move] = cislo[b];
                    move = move + 1;
                    b = b + 1;
                }
            }
            else
            {
                while (a < enda)
                {
                    cislo2[move] = cislo[a];
                    move = move + 1;
                    a = a + 1;
                }
            }
        }
        else
        {
            if (cislo[a] > cislo[b])
            {
                cislo2[move] = cislo[b];
                if (move != b)
                    info[0] = info[0] + 1;
                cislosipka[b] = move + 1 - start;
                b = b + 1;
                move = move + 1;
            }
            else
            {
                cislo2[move] = cislo[a];
                a = a + 1;
                move = move + 1;
            }
        }


    }
    novysloupec(start, end);
}



private void novysloupec(int start, int end)
    // Vykresli novy sloupec z pomocneho pole
{
    form.ColorReset();
    for (int i = start; i < end; i++)
    {
        cislo[i] = cislo2[i];
    }
}
```

5. Funkce programu celkově

5.1 První spuštění

Program je k uživateli velice přátelský již od prvního spuštění. Po startu programu se otevře již zmiňovaná hlavní forma a uživatel je hned přivítán úvodní zprávou, která mu vysvětlí, jak se s programem pracuje.



Vítejte ve výukovém programu o řadících algoritmech.

Ovládání tohoto programu je velice jednoduché. Vlevo uvidíte sloupce prázdných polí kam je nejprve nutno zadat hodnoty se kterými bude řadící algoritmus pracovat.
Pokud chcete můžete použít již předem připravené hodnoty. Tlačítka s připravenými hodnoty se nachází vpravo dole od prázdných polí a po jejich stisknutí se pole automaticky vyplní vybranými hodnotami.
Poté co jste si vybrali hodnoty, je třeba vybrat si ještě řadící algoritmus, který si chcete projít. K algoritmům si ještě můžete nastavit časovou prodlevu, která určuje rychlost zobrazování algoritmu. Větší časová prodleva = pomalejší zorazení.
Když už je vše vybráno, stiskněte tlačítko pokračovat. Nejprve sou zobrazeny teoretické informace k algoritmu a následně i grafické znázornění principu.

V horním menu Statistiky - porovnání algoritmů je možno porovnat statistiky předváděních řadícíh algoritmů (jako jsou čas, počet přesunů sloupců a počet porovnání sloupců).
Nejdřív je však nutné dané algoritmy alespoň jednou pustit a nechat k nim tyto statistiky uložit.

☐ Tuto zprávu již nezobrazovat.

Na obrázku – Úvodní zpráva

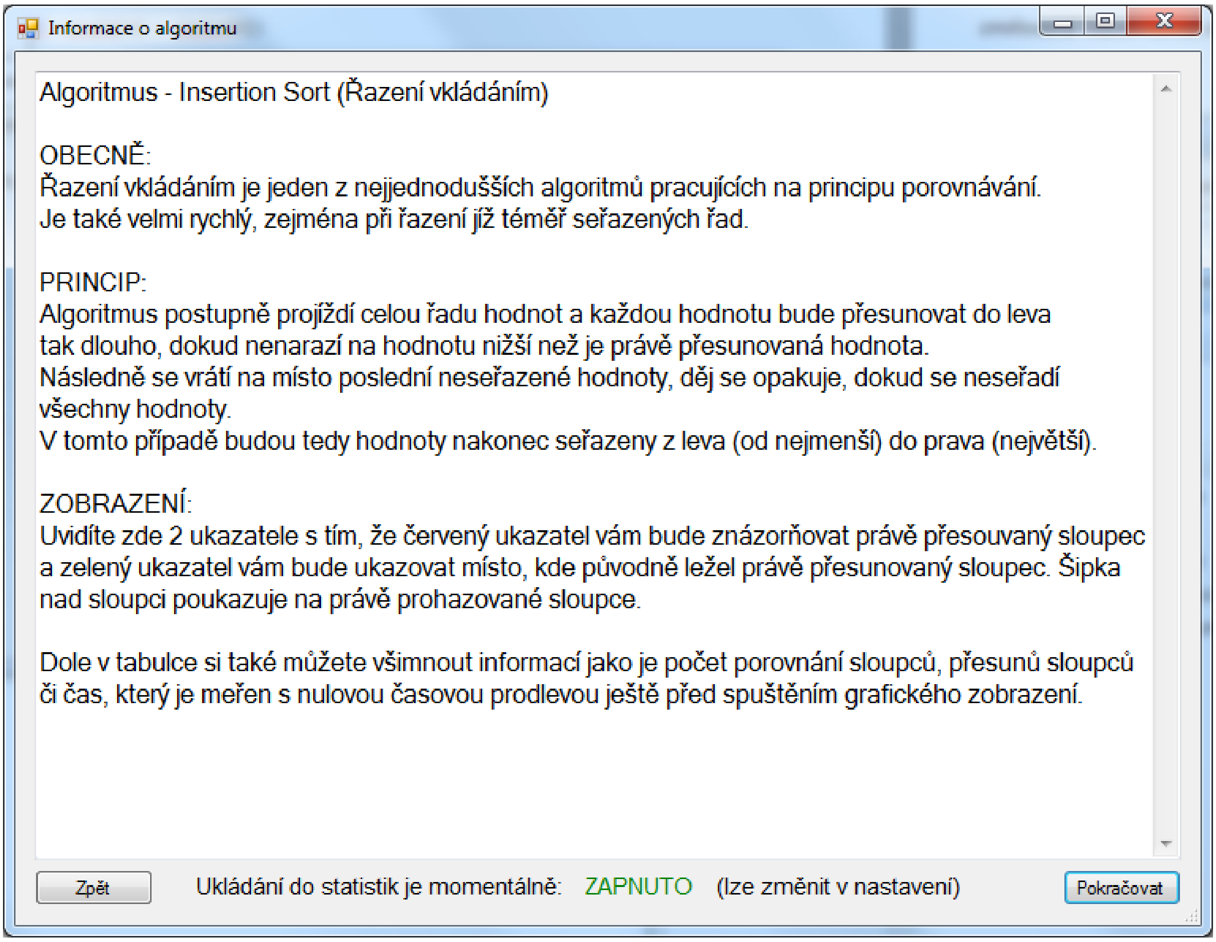
Po přečtení této zprávy by se již uživatel měl v hlavní formě plně vyznat. Zobrazování zprávy se tedy následně může vypnout.

5.2 Výukové možnosti

Jak již jistě víme, program slouží **hlavně k vysvětlování principu řadicích algoritmů**. Vysvětlování je zpracováno dvěma způsoby, a to **graficky** a **teoreticky**

Pokud se chceme naučit principy algoritmů, nastavíme si vlastní hodnoty, vybereme algoritmus, případně rychlost (časovou prodlevu) a stiskneme tlačítko pokračovat (dle návodu v úvodní zprávě).

Poté, co tak učiníme, se nám nejprve zobrazí teoretické informace k algoritmu:



Algoritmus - Insertion Sort (Řazení vkládáním)

OBECNĚ:
Řazení vkládáním je jeden z nejjednodušších algoritmů pracujících na principu porovnávání.
Je také velmi rychlý, zejména při řazení již téměř seřazených řad.

PRINCIP:
Algoritmus postupně projíždí celou řadu hodnot a každou hodnotu bude přesunovat do leva
tak dlouho, dokud nenarazí na hodnotu nižší než je právě přesunovaná hodnota.
Následně se vrátí na místo poslední neseřazené hodnoty, děj se opakuje, dokud se neseřadí
všechny hodnoty.
V tomto případě budou tedy hodnoty nakonec seřazeny z leva (od nejmenší) do prava (největší).

ZOBRAZENÍ:
Uvidíte zde 2 ukazatele s tím, že červený ukazatel vám bude znázorňovat právě přesouvaný sloupec
a zelený ukazatel vám bude ukazovat místo, kde původně ležel právě přesunovaný sloupec. Šipka
nad sloupci poukazuje na právě prohazované sloupce.

Dole v tabulce si také můžete všimnout informací jako je počet porovnání sloupců, přesunů sloupců
či čas, který je meřen s nulovou časovou prodlevou ještě před spuštěním grafického zobrazení.

Ukládání do statistik je momentálně: ZAPNUTO (lze změnit v nastavení)

Na obrázku – Teoretické informace pro algoritmus Insertion Sort.

Jakmile si přečteme teoretické informace, bychom měli pochopit princip řazení v hrubých obrysech..

To nejdůležitější je zde však grafické zobrazení, na které přejdeme stiskem pokračovat. Grafické zobrazení obsahuje sloupce vysoké dle zadaných hodnot. Po spuštění se sloupce začnou přesouvat a barevně znázorňovat: např. uvidíme, které sloupce se právě porovnávají a přesuvají.

Grafické zobrazení si můžeme podrobněji prohlédnout na stránce 8.

5.3 Porovnání vlastností algoritmů

Ke všem algoritmům jsou v průběhu grafického zobrazení vypisování informace: čas, počet přesunů sloupců a počet porovnání sloupců. Tyto informace lze ukládat do tzv. statistik, které jsou přehledně zobrazeny pro každý algoritmus zvlášť.

Statistiky mají více funkcí a lze se k nim dostat z hlavní formy nahoře v menu Statistiky - porovnání funkcí. Podrobnější popis statistik najdeme na stránkách 10 – 11.

5.4 Chyby v programu

Program je poměrně velmi dobře funkční, neobsahuje žádné zásadní chyby a nepadá. Dokonce jsou zde i vyřešeny případy, kdy je do kolonky pro hodnoty zadán neplatný znak, či pokud jsou textové soubory pro statistiky nějak poškozeny a tak podobně.

Jediná známá chyba v programu se vyskytuje velice výjimečně. Jedná se o případ, kdy grafické zobrazení chvíli po spuštění zamrzne a následně se po skončení algoritmu objeví až seřazené sloupce. Tato chyba se však objevuje cca u jednoho ze 30ti případů spuštění grafického zobrazení, při dalším spuštění zobrazení stejného algoritmu pro ty samé hodnoty již vše proběhne bez problémů.

Jelikož se tato chyba vyskytuje natolik výjimečně, není zcela možné vypátrat její příčinu, ale zároveň se nejedná o žádnou větší chybu, která by nějak zásadně obtěžovala uživatele.

6. Závěr

Program se mi povedlo sestavit bez větších potíží a myslím si, že je na dobré úrovni. Celkově vypadá zhruba tak, jak jsem si na začátku představoval.

Dokumentaci jsem se snažil zpracovat co nejlépe a nejpocrobněji. Doufám, že se mi podařilo srozumitelně popsat vše důležité a že jsem zmínil všechno, co bylo třeba.

Použitá literatura:

Teoretické informace o řadicích algoritmech jsem čerpal z těchto zdrojů:

MARTIN, David R. Sorting Algorithm Animations. [online]. [cit. 2013-04-05]. Dostupné z: http://www.sorting-algorithms.com/

DUDKA, Kamil. Algoritmy. [online]. [cit. 2013-04-05]. Dostupné z: http://dudka.cz/studyIAL

MIČKA, Pavel. Řadicí algoritmy. [online]. [cit. 2013-04-05]. Dostupné z: http://www.algoritmy.net/category/1019/Radici-algoritmy

Algoritmy a datove struktury. [online]. [cit. 2013-04-05]. Dostupné z:

http://eamos.pf.jcu.cz/amos/kat_inf/modules/low/kurz_obsah.php?kod_kurzu=kat_inf_30599

Podrobnější informace o příkazech jazyka C# jsem čerpal ze stránky:

http://msdn.microsoft.com/cs-cz/library/

Nebyla zkopírována žádná část kódu ani textu. Kód jsem psal svým způsobem a texty vlastními slovy.