Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Vizualizace a porovnání algoritmů třídění tabulek

Václav Mareš

Bakalářská práce

2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2008/2009**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Václav MAREŠ**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Vizualizace a porovnání algoritmů třídění tabulek**

Zásady pro vypracování:

Teoretická část práce bude prvotně obsahovat úvod do datových struktur. Dále bude potřebné vytvořit přehled a popis algoritmů třídění tabulek. Praktická část bude primárně zaměřena na návrh a implementaci aplikace pro vizualizaci a porovnání algoritmů třídění tabulek. Testování aplikace se provede nad vybranými množinami dat.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. WRÓBLEWSKI, P.: Algoritmy, datové struktury a programovací techniky. ComputerPress, Brno, 2004. 2. WIRTH, N.: Algoritmy a štruktúry údajov, Alfa 1975. 3. LEWIS, H. R., DENENBERG, L.: Data structures and their algorithms. Berkley, Adison-Wesley, 1997.

Vedoucí bakalářské práce: **Ing. Jan Fikejz**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

Univerzita Pardubice -8- Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 14. 5. 2009

Václav Mareš

Poděkování

Rád bych touto cestou poděkoval vedoucímu mé bakalářské práce Ing. Janu Fikejzovi za cenné rady a připomínky, za jeho čas, trpělivost a ochotu při řešení problémů vzniklých při zpracovávání této práce.

Anotace

Tato práce se zabývá popisem základních třídících algoritmů. Cílem je vytvoření aplikace sloužící k porovnání a vizualizaci jednotlivých algoritmů.

Klíčová slova

Tabulka, třídící algoritmus, Insertinsort, Selectionsort, Bubblesort, Quicksort, Shellsort, Mergesort, Ripplesort, Shakersort, Heapsort

Title

Vizualization and comparing algorithms of sorting tables

Annotation

This bachelor work deals with the despription of basic algorithms of sorting. The aim is to create a specific application servant to comparing sorting algorithms and their visualization.

Keywords

Table, algorithms of sorting, Insertinsort, Selectionsort, Bubblesort, Quicksort, Shellsort, Mergesort, Ripplesort, Shakersort, Heapsort

OBSAH

Obsah 7
Seznam obrázků 9
Seznam tabulek 10
1 Úvod 12
2 Základní pojmy 13
2.1 Abstraktní datový typ 13
2.2 Abstraktní datová struktura 13
2.3 Problematika třídění 13
2.3.1 Třídící problém 13
2.3.2 Klasifikace algoritmů 14
2.3.3 Ověřování správnosti algoritmu 15
2.3.4 Dokazování konečnosti algoritmu 15
3 Abstraktní datový typ tabulka 16
3.1 Definice 16
3.2 Operace tabulky 16
3.3 Implementace na poli 16
3.4 Implementace na seznamu 17
3.5 Implementace na binárním vyhledávacím stromu 17
3.6 Implementace na implicitní kosočtvercové vyhledávací síti 19
3.7 Implementace na rozptýlených tabulkách 20
4 Třídící algoritmy 22
4.1 Úvod 22
4.2 Třídění přímým vkládáním 22
4.2.1 Insertion sort 22
4.2.2 Shell sort 25
4.3 Třídění přímým výběrem 28
4.3.1 Selectionsort 28
4.3.2 Heapsort 32
4.4 Třídění výměnou 36
4.4.1 Bubble sort 36
4.5 Quicksort 39

4.5.2 Shakersort ... 43
4.5.3 Shuttlesort ... 44
4.5.4 RippleSort ... 45
4.6 Třídění spojováním ... 45
4.6.1 Mergesort ... 45
5 Uživatelská část aplikace ... 49
5.1 Uživatelský popis aplikace ... 49
5.2 Ukázka animace ... 52
5.3 Ukázka třídění tabulky jedním algoritmem ... 53
5.4 Ukázka třídění tabulky více algoritmy ... 55
5.5 Ukázka grafu ... 56
6 Návrh a implementace aplikace ... 58
6.1 Úvod ... 58
6.2 Popis implementace aplikace ... 58
6.3 Diagram tříd ... 60
6.4 Stručný popis tříd a rozhranní ... 60
6.4.1 Třída Prvek ... 61
6.4.2 Třídy PrvekInt, PrvekDbl a PrvekStr ... 61
6.4.3 Rozhraní ITabulka ... 61
6.4.4 Třída Tabulka ... 61
6.4.5 Třída Scenar ... 62
6.4.6 Třída Vysledky ... 62
6.4.7 Třída Soubor ... 62
6.4.8 Statická třída RegVyrazy ... 62
6.4.9 Třída Grafika ... 62
6.4.10 Třída Generátor ... 63
6.4.11 Formulář AnimForm ... 63
6.4.12 Formulář PorovForm ... 63
6.4.13 Formulář GrafForm ... 63
6.4.14 Formulář ZobrazForm ... 64
6.4.15 Formulář ProgForm ... 64
6.4.16 Formulář HlavniForm ... 64
7 Porovnání algoritmů ... 65
7.1 Úvod ... 65

7.2 Testovací sestavy ... 66
7.3 Ověření rozptylu doby třídění ... 66
7.4 Naměřené výsledky prvního testu ... 67
7.5 Naměřené výsledky druhého testu ... 71
8 Závěr ... 74
9 Literatura ... 75
9.1 Seznam použité literatury ... 75
10 Seznam příloh ... 76

SEZNAM OBRÁZKŮ

Obrázek 1 - Datová struktura pole ... 17
Obrázek 2 - Datová struktura seznam ... 17
Obrázek 3 - Datová struktura strom ... 18
Obrázek 4 - Datová struktura BVS ... 19
Obrázek 5 - Datová struktura KVS ... 20
Obrázek 6 - Třídící algoritmus Insertionsort ... 23
Obrázek 7 - Graf výkonnosti Insertion sort ... 24
Obrázek 8 - Třídící algoritmus Shellsort ... 26
Obrázek 9 - Graf výkonnosti Shellsort ... 28
Obrázek 10 - Třídící algoritmus Selectionsort ... 30
Obrázek 11 - Graf výkonnosti Selectionsort ... 32
Obrázek 12 - Třídící algoritmus Heapsort ... 34
Obrázek 13 - Graf výkonnosti Heapsort ... 36
Obrázek 14 - Třídící algoritmus Bubblesort ... 37
Obrázek 15 - Graf výkonnosti Bubblesort ... 39
Obrázek 16 - Třídící algoritmus Quicksort ... 41
Obrázek 17 - Graf výkonnosti Quicksort ... 43
Obrázek 18 - Třídící algoritmus Mergesort ... 46

Obrázek 19 - Graf výkonnosti Mergesort 48
Obrázek 20 - Ukázka hlavního formuláře aplikace 50
Obrázek 21 - Ukázka formuláře zobrazujícího hodnoty prvků tabulky 51
Obrázek 22 - Ukázka formuláře pro animaci 52
Obrázek 23 - Ukázka prohození dvou prvků s hodnotami typu integer 53
Obrázek 24 - Ukázka prohození dvou prvků s hodnotami typu double 53
Obrázek 25 - Ukázka hlavního formuláře s výsledky 55
Obrázek 26 - Ukázka výsledků vybraných algoritmů 56
Obrázek 27 - Ukázka grafu 57
Obrázek 28 - Class diagram 60
Obrázek 29 - Zobrazení pomalejší skupiny třídících algoritmů do grafu. 69
Obrázek 30 - Zobrazení rychlejší skupiny třídících algoritmů do grafu. 69
Obrázek 31 - Zobrazení rychlejší skupiny algoritmů třídících 1.000.000 prvků 70
Obrázek 32 - Graf různých datových typů algoritmů Quicksort a Mergesort 72
Obrázek 33 - Graf různých datových typů algoritmů Insertionsort a Bubblesort 73

SEZNAM TABULEK

Tabulka 1 - První sestava s procesorem Intel Atom 66
Tabulka 2 - Druhá sestava s procesorem Intel Pentium IV 66
Tabulka 3 - Třetí sestava s procesorem AMD Athlon X2 66
Tabulka 4 - Test rozptylu naměřených hodnot 67
Tabulka 5 - Výsledky naměřené první sestavou. 67
Tabulka 6 - Výsledky naměřené druhou sestavou. 68
Tabulka 7 - Výsledky naměřené třetí sestavou 68
Tabulka 8 - Výsledky naměřené první sestavou. 70
Tabulka 9 - Výsledky naměřené druhou sestavou. 70

Tabulka 10 - Výsledky naměřené třetí sestavou. 70
Tabulka 11 - Naměřené výsledky různých datových typů algoritmem Quicksort 71
Tabulka 12 - Naměřené výsledky různých datových typů algoritmem Mergesort..... 71
Tabulka 13 - Naměřené výsledky různých datových typů algoritmem Insertionsort..... 72
Tabulka 14 - Naměřené výsledky různých datových typů algoritmem Bubblesort..... 72

1 ÚVOD

Cílem této práce je seznámit čtenáře s problematikou třídících algoritmů. Prakticky ukázat způsob třídění jednotlivých algoritmů. Možnost setřídění vlastních dat uložených v souborech a v poslední řadě porovnání doby třídění jednotlivých algoritmů.

Práce je rozdělena do dvou částí. První část seznamuje čtenáře se základními pojmy a problematikou třídění, dále vysvětlí pojem tabulka, včetně jejich pěti implementací. Poté následuje popis devíti základních třídících algoritmů, jejich vlastnosti, implementace v jazyce C# a graf výkonnosti.

Druhá, praktická část nám ukáže popis konkrétní aplikace, její návrh, implementaci v jazyce C# a nakonec porovnání deseti třídících algoritmů.

2 ZÁKLADNÍ POJMY

V této kapitole jsou vysvětleny některé pojmy z oblasti datových struktur a problematiky třídění. Vysvětlení pojmů vychází z použité literatury [3, 7].

2.1 Abstraktní datový typ

Abstraktní datový typ, zkráceně ADT, je výraz pro typy dat, které jsou nezávislé na vlastní implementaci. Primárním cílem je zjednodušit a zpřehlednit program, který provádí operace s daným datovým typem. ADT je sestávající ze dvou částí. Jednoho nebo více matematických prvků a jedné nebo více operací na prvcích těchto domén.

2.2 Abstraktní datová struktura

Abstraktní datová struktura je konkrétní realizace (instance) ADT. ADS implementuje algoritmy operací.

2.3 Problematika třídění

2.3.1 Třídící problém

Třídící problém se definuje následovně : je daná množina A = $\{a_1, a_2, ..., a_n\}$. Je zapotřebí najít permutaci Pi těchto n prvků, která zobrazuje danou posloupnost do neklesající posloupnosti $a_{\pi(1)}, a_{\pi(2)}, ..., a_{\pi(n)}$ tak, že $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$. Množina U, ze které vybíráme prvky třídění množiny se nazývá univerzum.

Třídící problém, takový jaký jsme ho definovali, je stále ještě jistou abstrakcí reálné situace, kdy obyčejně máme s každým prvkem z množiny U vázanou nějakou další informaci, která nemá na definici uspořádání žádny vliv. Prvky množiny U se nazývají klíče a informaci vázanou na klíč spolu s klíčem nazýváme záznam. Jestliže je velikost vázané informace příliš velká, je výhodnější setřídit jen klíče s odkazy na patřičné informace. Samotná informace se v tomto případě nepřesouvá.

Třídící metodu nazýváme **stabilní**, když zachovává relativní uspořádání záznamu se stejným klíčem nebo **přirozenou**, jestliže jeho složitost roste nebo klesá v závislosti na míře původní setříděnosti vstupní posloupnosti. Třídění se nazývá **insitu** (neboli na původním místě), jestliže algoritmus vyžaduje, kromě vlastního tříděného pole,

pomocnou paměť konstantního rozsahu. Jinými slovy algoritmus nepoužívá žádnou další paměť, jejíž velikost by byla závislá na rozsahu tříděných hodnot.

2.3.2 Klasifikace algoritmů

Další rozdělení třídících algoritmů je podle toho, jak třídící algoritmy pracují.

- **Algoritmy adresného třídění**. Tyto algoritmy využívají jednoznačného vztahu mezi absolutními hodnotami prvků z množiny U a jejich pozici v uspořádané množině. Pro výpočet tohoto vztahu můžeme použít libovolné operace kromě porovnání tříděných prvků.

- **Algoritmy asociativního třídění** jsou algoritmy, které používají pro určení pozice prvku v množině S jen relativní hodnoty prvků, které určují vzájemným porovnáváním těchto prvků.

- **Algoritmy hybridního třídění** jsou kombinací předcházejících algoritmů. Tyto algoritmy nejsou nijak omezovány při zjišťování pozice daného prvku.

Jestliže datový typ, nad kterým implementujeme třídící algoritmus vykonává **paralelní** informace, mluvíme o paralelním třídění, jinak mluvíme o třídění **sériovém**. Výběr vhodné datové struktury v podstatné míře ovlivňuje efektivitu algoritmu.

Vstupem pro třídící algoritmus je tříděná množina S, která je reprezentovaná posloupností. Tuto posloupnost můžeme v počítači reprezentovat polem nebo seznamem. Polem můžeme množinu S reprezentovat v případě, že máme k dispozici velkou paměť s přímým přístupem. Při převýšení kapacity paměti s přímým přístupem prvky musíme použít strukturu seznam.

Volba reprezentace už jednoznačně určuje, jakou datovou struktur můžeme při třídění použít. Tato volba vede k rozdělení třídících algoritmů do dvou odlišných tříd. První třída reprezentuje algoritmy vnitřního třídění a druhá algoritmy vnějšího třídění. Při vnitřním třídění máme větší volnost při výběru datových struktur a operací nad nimi. Naopak při vnějším třídění si musíme vystačit jen datovými struktury se sekvenčním přístupem.

2.3.3 Ověřování správnosti algoritmu

K ověření správnosti algoritmu nestačí vyzkoušet daný algoritmus na konečný počet vstupních dat. Není ale dokázáno, že se algoritmus při neočekávané kombinaci vstupních dat nezhroutí.

Pro ověřování správnosti algoritmu neexistuje metoda, která by byla univerzální. Algoritmus by měl být matematicky dokázán, sledem předem známých operací, které vedou pro všechna přípustná data ke správnému výsledku úlohy. Je pravděpodobné, že takový algoritmus je pak korektním řešením.

Algoritmus považujeme za korektní, pokud není opomenuta žádná z možností zpracování dat při průchodu algoritmem. Algoritmus je částečně správný, pokud platí, že pokud skončí vydá správný výsledek.

K dokázání částečné správnosti výpočtu můžeme použít tzv. invariant - tvrzení, které platí po celou dobu výpočtu. Také je nutné ověřit konečnost algoritmu, pro všechna přípustná data algoritmus po konečném počtu kroků končí.

2.3.4 Dokazování konečnosti algoritmu

Konečnost algoritmu je intuitivně zřejmá tím, že se pro vstupní data nemůže algoritmus zacyklit. Konečnost algoritmu dokazujeme takto: Pokud najdeme způsob, který každý stav výpočtu ohodnotí přirozeným číslem a ukážeme, že provedení jednoho kroku algoritmu se tato hodnota zmenší, je patrné, že algoritmus po konečném počtu kroku skončí, neboť interval počtu průchodů algoritmem je konečný.

3 ABSTRAKTNÍ DATOVÝ TYP TABULKA

Vysvětlení pojmu abstraktní datový typ tabulka vychází z použité literatury [3].

3.1 Definice

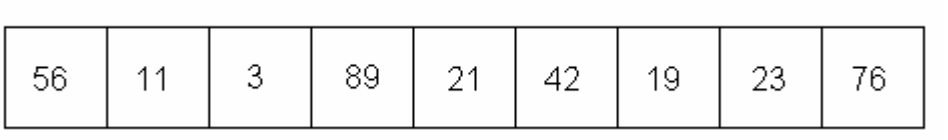
ADT Tabulka je množina s lineárním uspořádáním, přičemž uspořádání je určováno jednoznačnými klíčovými hodnotami (klíči) prvků.

3.2 Operace tabulky

- **Vytvoř** - Vytvoří prázdnou tabulku.
- **Zruš** - Zruší prázdnou tabulku.
- **Je Prázdná** - Tato operace prověří zda-li je tabulka prázdná. Vrací true nebo false.
- **Mohutnost** - Operace, která nám vrátí počet prvků.
- **Prohlídka** - U této operace předáme typ prohlídky a akce.
- **Vlož** - Operace vlož nám vloží prvek na konec tabulky.
- **Odeber** - Operace odeber nám odebere prvek s požadovaným klíčem.
- **Najdi** - Operace nám podle klíče najde a vrátí prvek.

3.3 Implementace na poli

Pole patří mezi jednoduché datové struktury. Přístup k prvkům pole je určen pomocí hodnoty indexu a není závislý na přístupu k jinému prvku. Pole je strukturou s přímým nebo náhodným přístupem. Počet prvků může být nastaven pevně (staticky), nebo se může měnit v době zpracování (dynamicky).

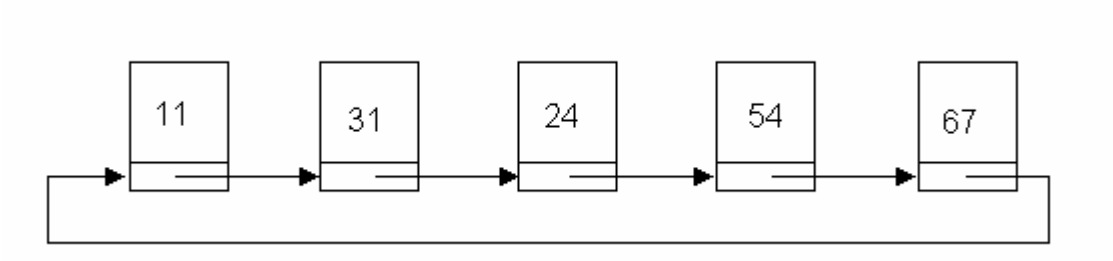


Obrázek 1 - Datová struktura pole[1]

Tabulku můžeme implementovat na utříděném nebo neutříděném poli. Implementace na utříděném poli má složitost operace vlož O(n), operace odeber O(n) a operace najdi O(log2n) (binární hledání). Na neutříděném poli má složitost operace vlož O(1), operace odeber O(1) bez operace najdi a O(n) s operací najdi.

3.4 Implementace na seznamu

Lineární seznam nebo také lineární spojový seznam je dynamická datová struktura, na kterou přistupujeme sekvenčně. Seznamy mohou být jednosměrné nebo obousměrné. V jednosměrném seznamu ukazuje každý prvek seznamu na následující prvek a v obousměrném seznamu ukazuje prvek na předchozí i následující prvky. Pokud poslední prvek ukazuje na prvek první, jedná se o kruhový seznam.



Obrázek 2 - Datová struktura seznam[2]

Tabulku můžeme implementovat na utříděný nebo neutříděný seznam. Složitosti jsou stejné jako při implementaci na pole.

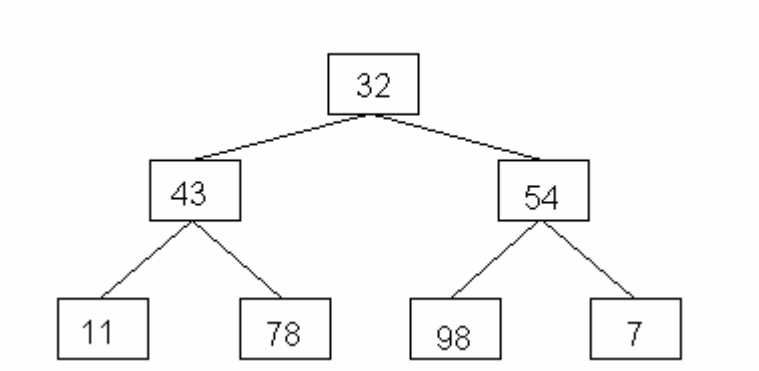
3.5 Implementace na binárním vyhledávacím stromu

Abstraktní datová struktura strom je hierarchicky uspořádaná množina dat. Tato dynamická datová struktura se nazývá podle svého grafického zobrazení. První prvek,

[1] Zdroj: Vlastní

[2] Zdroj: Vlastní

který do struktury stromu vložíme, se nazývá kořen. Podle počtu potomků rozdělujeme stromy na binární a k-cestné, přičemž binární strom není nic jiného než specifický případ k-cestného stromu. Binární strom je strom, u kterého každý prvek má pouze dva potomky. Zde rozlišujeme tři základní druhy rekurzivních prohlídek: **Preorder**, **Inorder** a **Postorder**. U k-cestného stromu má každý prvek právě *k* potomků. Dále potom máme stromy unární, kde každý prvek unárního stromu má přístup pouze k jednomu prvku (otci).

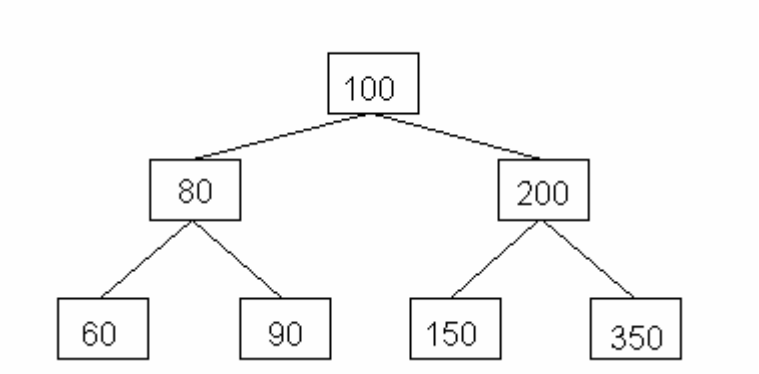


Obrázek 3 - Datová struktura strom[3]

Binární vyhledávací strom, zkráceně BVS je označení pro binární strom, pro jehož vrchol x, charakterizovaný klíčem Kx platí dvě pravidla.

- Je-li y vrchol z levého podstromu vrcholu x, pak Ky < Kx.
- Je-li y vrchol z pravého podstromu vrcholu x, pak Ky > Kx.

[3] Zdroj: Vlastní



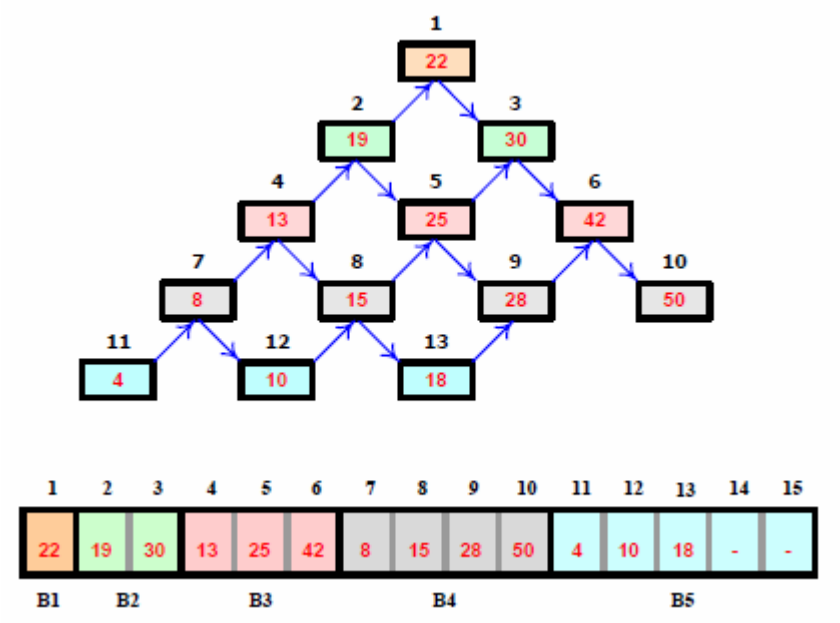
Obrázek 4 - Datová struktura BVS[4]

Zvolíme-li si jako implementační datovou strukturu BVS postupujeme při použití operace Najdi tak, že traverzujeme v BVS od kořene založeném na porovnání klíče hledaného prvku s klíčem aktuálního prvku a následným výběrem další vhodné (pravé nebo levé) větve pro další postup, pokud již prvek nebyl nalezen nebo lze jeho výskyt v BVS vyloučit. U operace vlož traverzujeme stejně jako u operace najdi, než najdeme místo pro nově vkládaný prvek, který je do BVS vložen jako list. Operace odeber nahradí prvek nejpravějším prvkem z jeho levého podstromu nebo nejlevějším prvkem z jeho pravého podstromu (pokud existuje). Operace prohlídku provede jednu ze tří základních prohlídek binárního stromu.

3.6 Implementace na implicitní kosočtvercové vyhledávací síti

Pro Kosočtvercovou vyhledávací síť dále KVS platí, že je utřídění ve vzestupném pořadí ve směru šipek (zdola šikmo vpravo nahoru a shora šikmo vpravo dolů). Dále pro každý prvek platí, že jeho následník vpravo dole má větší hodnotu klíče a předchůdce vlevo dole hodnotu menší; z toho vyplývá, že prvky na jedné úrovni (v bloku) jsou utříděny vzestupně zleva doprava. V i-tém bloku je pokaždé i prvků (kromě posledního bloku)

[4] Zdroj: Vlastní



Obrázek 5 - Datová struktura KVS[5]

- Následníci prvku indexu i (z bloku B) jsou i - B + 1, i + B + 1.
- Předchůdci prvku na indexu i (z bloku B) jsou i - B, i + B.
- Složitosti jednotlivých operaci jsou: Najdi O(n1/2), Vlož O(1/2) a Odeber O(1/2).

Operace vlož a odeber provádějí zatřídění prvku s klíčem K na jeho správné místo tak, že se porovná K s klíči jeho předchůdců a následníků, pokud existují. Je-li prvek s klíčem K menší než některý z jeho předchůdců, tak je vyměněn s větším z nich, pokud je větší než některý z jeho následníků, tak je vyměněn s menším z nich.

3.7 Implementace na rozptýlených tabulkách

Princip rozptýlených tabulek spočívá v určení adresy (indexu) hledaného prvku přímo z jeho klíče bez nutnosti uplatnění asociativního algoritmu založeného na porovnání hledaného klíče s klíči uložených prvků/záznamů. Cílem je dosáhnout asymptotické výpočetní složitosti O(1) pro základní operace. Transformace klíče je tedy nutná k tomu, aby byl omezen prostor možných hodnot klíče, který je daleko větší než prostor použitelných adres pro uložení záznamů.

[5] Zdroj: A. Kávička: Sylaby přednášek předmětu „Datové struktury“ v bakalářském studiu

Operace Vlož

- Nejprve výpočet primární adresy A = H(K) prvku (indexu prvku na poli)
- Je-li primární adresa volná a je na ni vkládaný prvek uložen
- Je-li primární adresa obsazena, je vkládaný prvek uložen na nové, vyalokované paměťové místo (se sekundární adresou) a je připojen k seznamu prvků této skupiny.

Operace Odeber

- Nejprve vypočet primární adresy prvku A = H(K)
- Asociativním vyhledáváním podle K je prohledán seznam kolizních prvků.
- Odebrán nalezený prvek

4 TŘÍDÍCÍ ALGORITMY

4.1 Úvod

V této kapitole se dostáváme ke konkrétnímu popisu třídících algoritmů. Každá podkapitola nám ukáže jeden třídící algoritmus. Je zde zvolen jednotný formát, kdy nejprve definujeme algoritmus, poté si tento algoritmus popíšeme a následně si ukážeme způsob třídění. Dále algoritmus v jazyce C#, nakonec parametry třídění a graf výkonnosti. V této kapitole jsem čerpal z použité literatury [2, 7, 8].

4.2 Třídění přímým vkládáním

4.2.1 Insertion sort

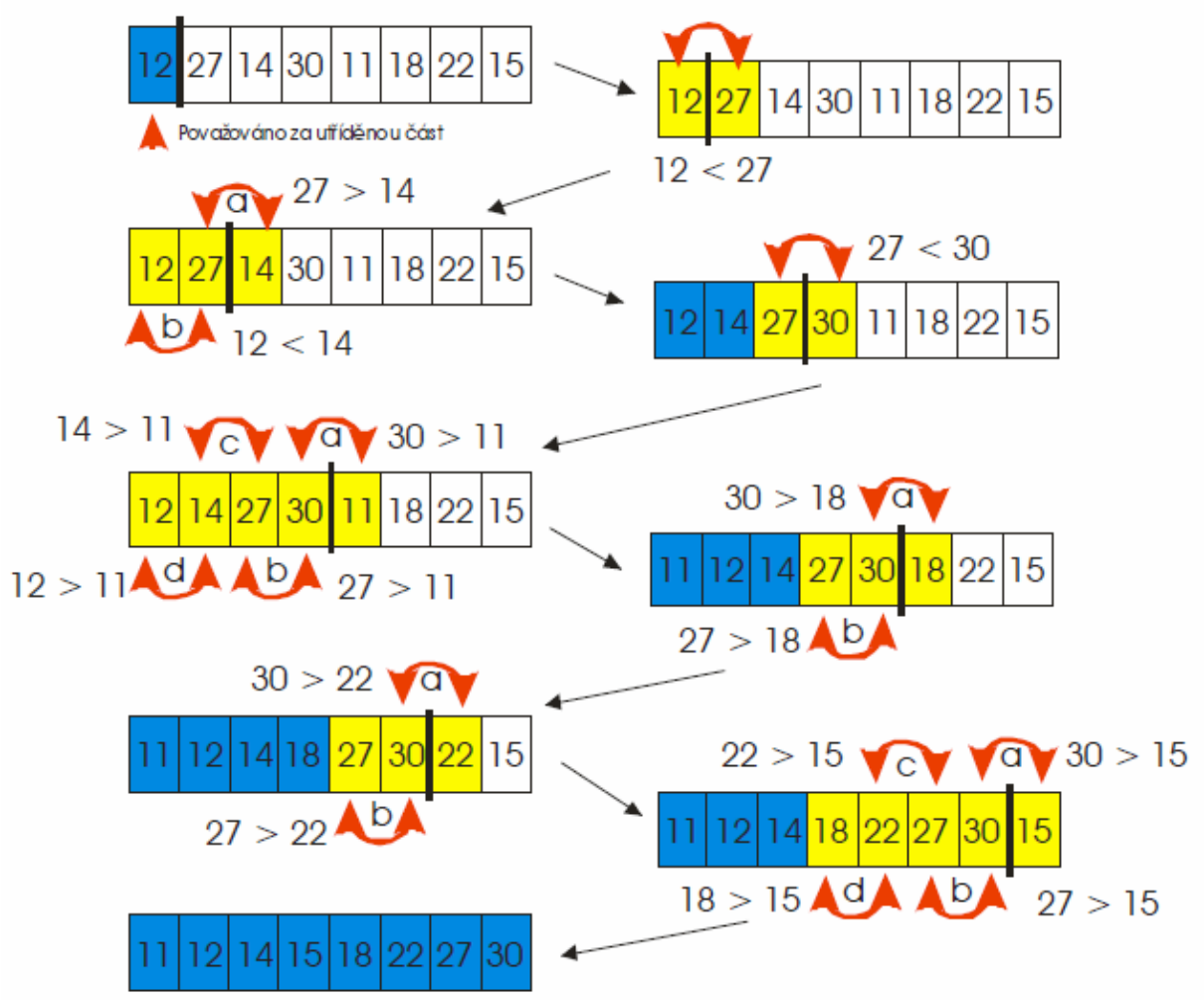
4.2.1.1 Úvodem o algoritmu

Algoritmus je založen na principu vkládání prvku na správné místo v posloupnosti. Je jednoduše implementovatelná a efektivní na malých množinách. Princip algoritmu je podobný algoritmu, jakou si karetní hráč seřazuje karty v ruce, když po rozdání karet přibírá karty ze stolu jednu po druhé a vkládá je mezi seřazené karty. Posloupnost prvků je rozdělena na utříděnou a neutříděnou část. Z nesetříděné části vybíráme prvky a vkládáme je do setříděné části tak, aby setříděná část zůstala setříděná. Tento postup opakujeme dokud není nesetříděná část prázdná.

4.2.1.2 Popis algoritmu

- První prvek necháme na svém místě a považujeme ho za utříděnou část
- Druhý prvek porovnáme s prvním, je-li menší, prohodíme ho s prvním prvkem, jinak ponecháme na svém místě.
- Vezmeme třetí prvek a porovnáme ho s prvním a druhým prvek. Je-li menší než některý z nich, zařadíme ho na příslušnou pozici a následující prvky podle potřeby posuneme. Jinak ponecháme na původním místě.
- Podobně postupujeme s dalšími prvky, dokud není nesetříděná část prázdná.

4.2.1.3 Ukázka třídění



Obrázek 6 - Třídící algoritmus Insertionsort[6]

4.2.1.4 Algoritmus

```
public static void InsertionSort(int[] array)
{
  int pom;
  for (int i = 1; i <= array.Lenght - 1; i++)
  {
     pom = array[i];
     for (int j = i - 1; j >= 0; j--)
     {
         if (array[j] > array[j + 1])
         {
            int tmp = array[j];
            array[j] = array[j + 1];
```

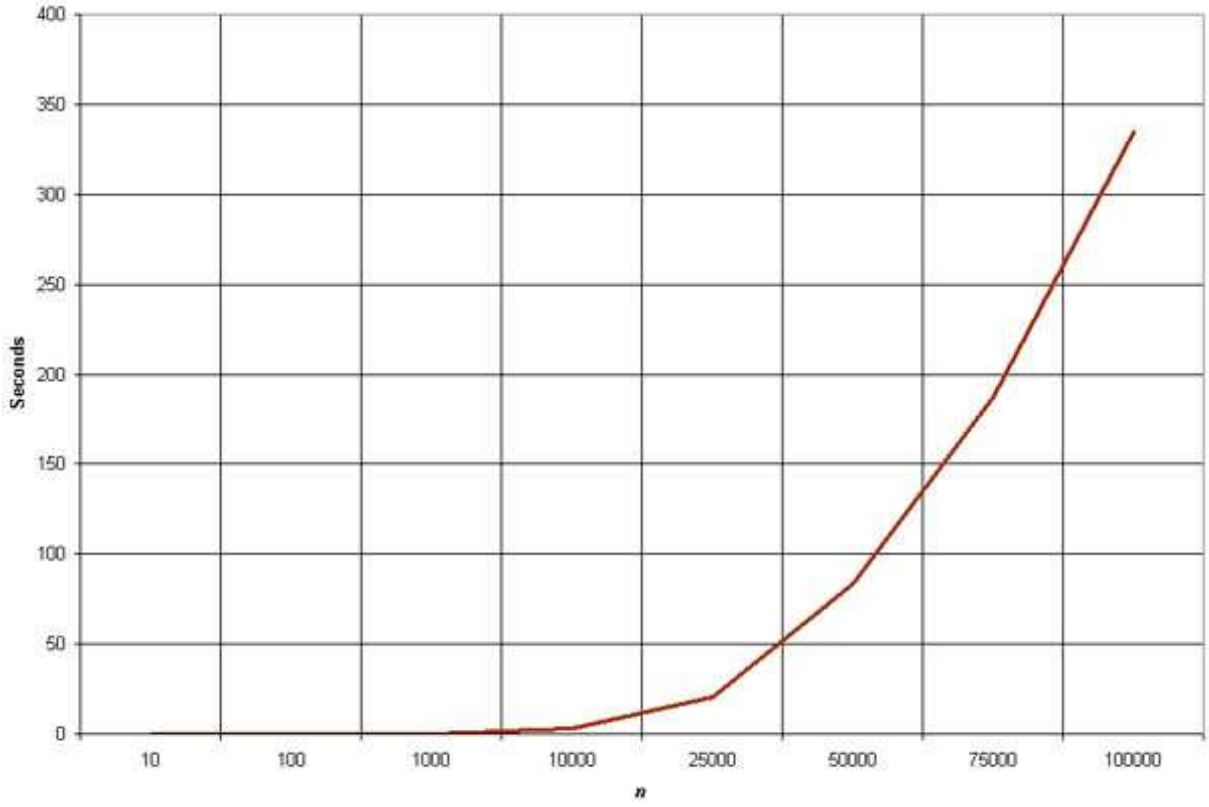
[6] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
                array[j + 1] = array[j];
            }
        }
    }
}
```

4.2.1.5 Parametry algoritmu

- Stabilní řazení : ANO
- Časová složitost : Θ (n^2)
- Potřeba dalších datových struktur : Θ(1)

4.2.1.6 Graf výkonnosti



Obrázek 7 - Graf výkonnosti Insertion sort[7]

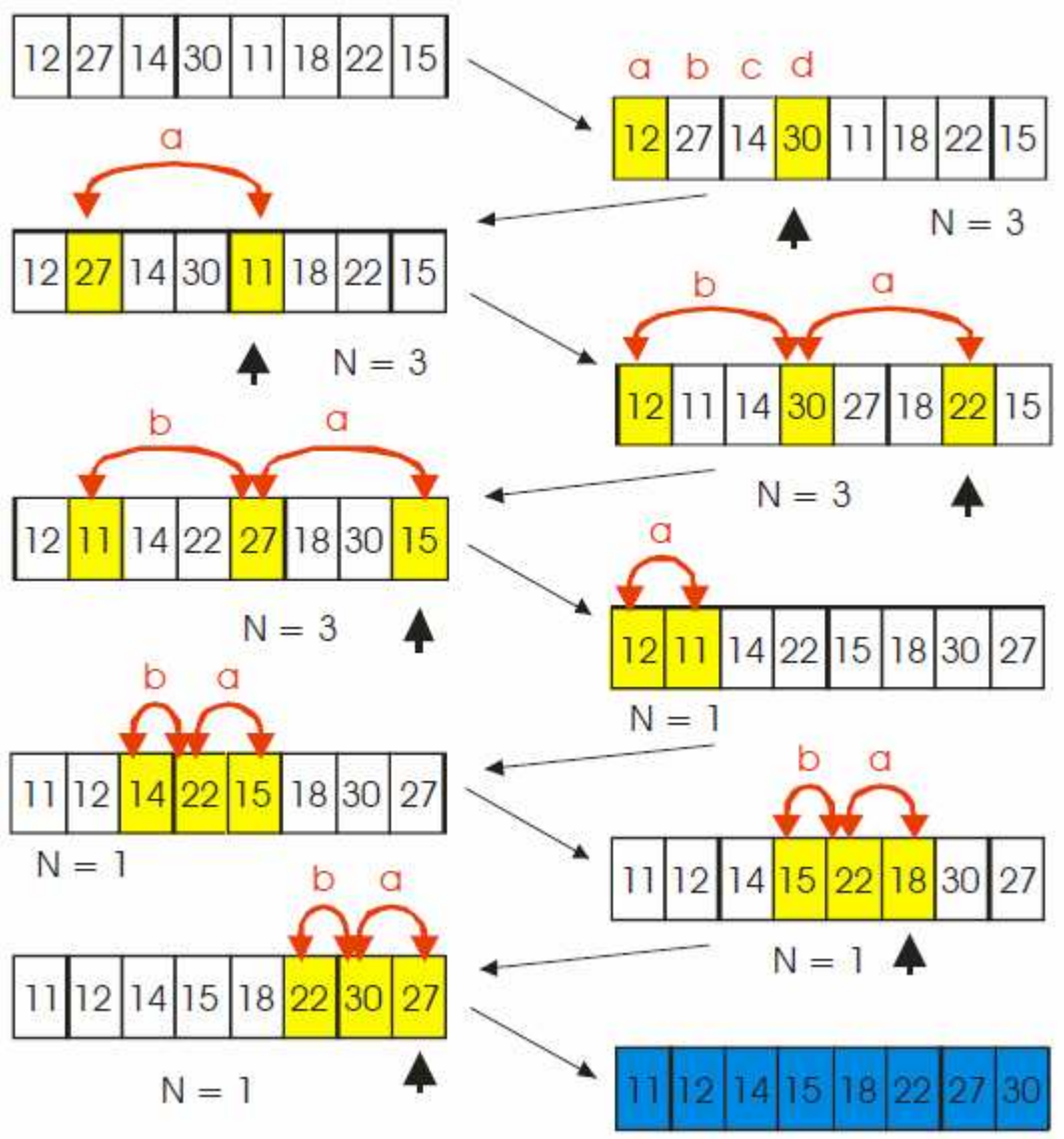
[7] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

4.2.2 Shell sort

4.2.2.1 Úvodem o algoritmu

Tuto metodu vytvořil pan Donald L. Shell, který ji poprvé publikoval v roce 1959. Metoda je velice podobná metodě BubbleSort, ale je podstatně vylepšena, a proto také rychlejší. Pan Shell při tvorbě této metody přišel na určité zákonitosti jako např., každý prvek se v posloupnosti přesune v průměrném případě o jednu třetinu celkové délky posloupnosti. Tohoto poznatku pan Shell využil a zmiňovanou metodu BubbleSort přepracoval ve smyslu, že měnil v každém průchodu posloupností vzdálenost dvojic prvků, které se měly uspořádat.

4.2.2.2 Ukázka třídění



Obrázek 8 - Třídící algoritmus Shellsort[8]

4.2.2.3 Algoritmus

```
 public static void ShellSort(int[] a)
 {
     for (int in = a.length / 2; in > 0; in = (in == 2 ? 1 :
(int) Math.round(in / 2.2)))
     {
         for (int i = in; i < a.length; i++)
         {
             int temp = a[i];
```

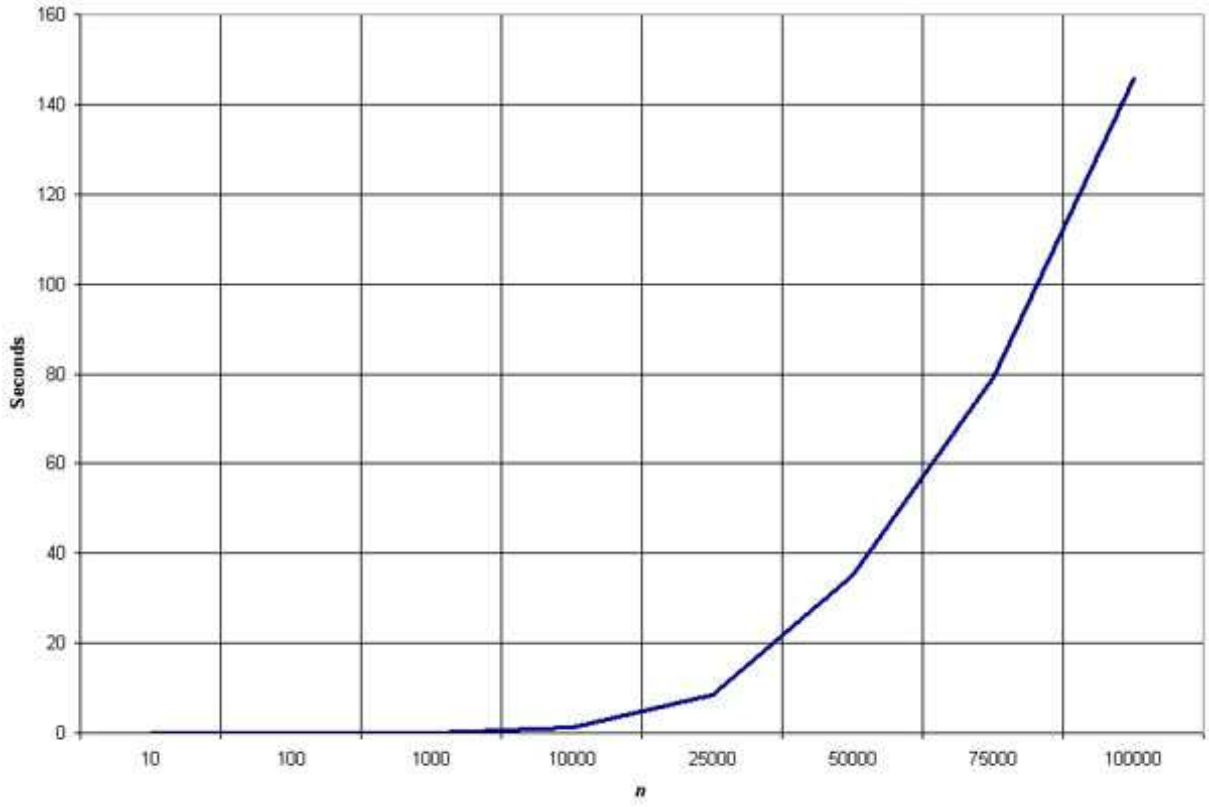
[8] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
            for (int j = i; j >= in && a[j - in] > temp; j
-= in)
            {
                a[j] = a[j - in];
                a[j - in] = temp;
            }
        }
    }
 }
```

4.2.2.4 Parametry algoritmu

- Stabilní řazení : NE
- Časová složitost : $\Theta (n^2)$
- Potřeba dalších datových struktur : $\Theta(1)$

4.2.2.5 Graf výkonnosti



Obrázek 9 - Graf výkonnosti Shellsort[9]

4.3 Třídění přímým výběrem

4.3.1 Selectionsort

4.3.1.1 Úvodem o algoritmu

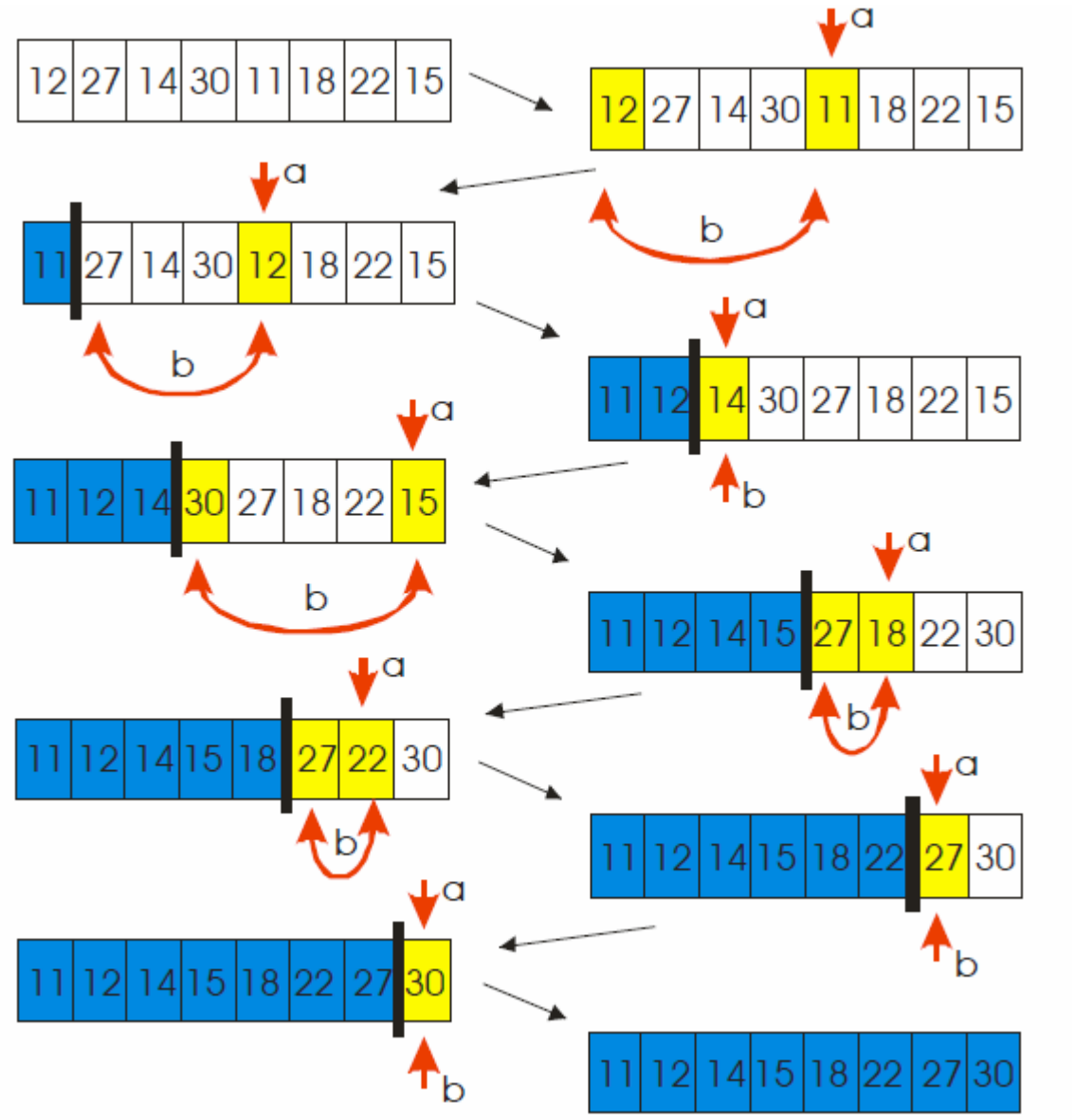
Selectionsort, selectsort nebo také minsort pracuje na principu nalezení minimálního prvku v nesetříděné části posloupností a jeho zařazení na konec již setříděné posloupnosti. Metoda prochází celou nesetříděnou část a hledá nejmenší prvek. Až jej nalezne, vrátí se na konec setříděné části posloupnosti a tento nejmenší prvek zde uloží. Tyto dvě činnosti vykonává do té doby, dokud neprojde celou posloupnost prvků a nesetřídí ji.

[9] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

4.3.1.2 Popis algoritmu

- V poli prvků najdeme nejmenší prvek a prohodíme s prvním prvkem. Tím dostaneme setříděnou a nesetříděnou část pole.
- V nesetříděném poli najdeme nejmenší prvek a vyměníme ho s prvním prvek v nesetříděné části pole.
- Opakujeme bod dvě dokud není setříděno.

4.3.1.3 Ukázka třídění



Obrázek 10 - Třídící algoritmus Selectionsort[10]

4.3.1.4 Algoritmus

```
public static void SelectionSort(int[] array)
{
   int min;
   for (int i = 0; i <= array.Length - 2; i++)
   {
       min = i;
       for (int j = i + 1; j <= array.Length - 1; j++)
```

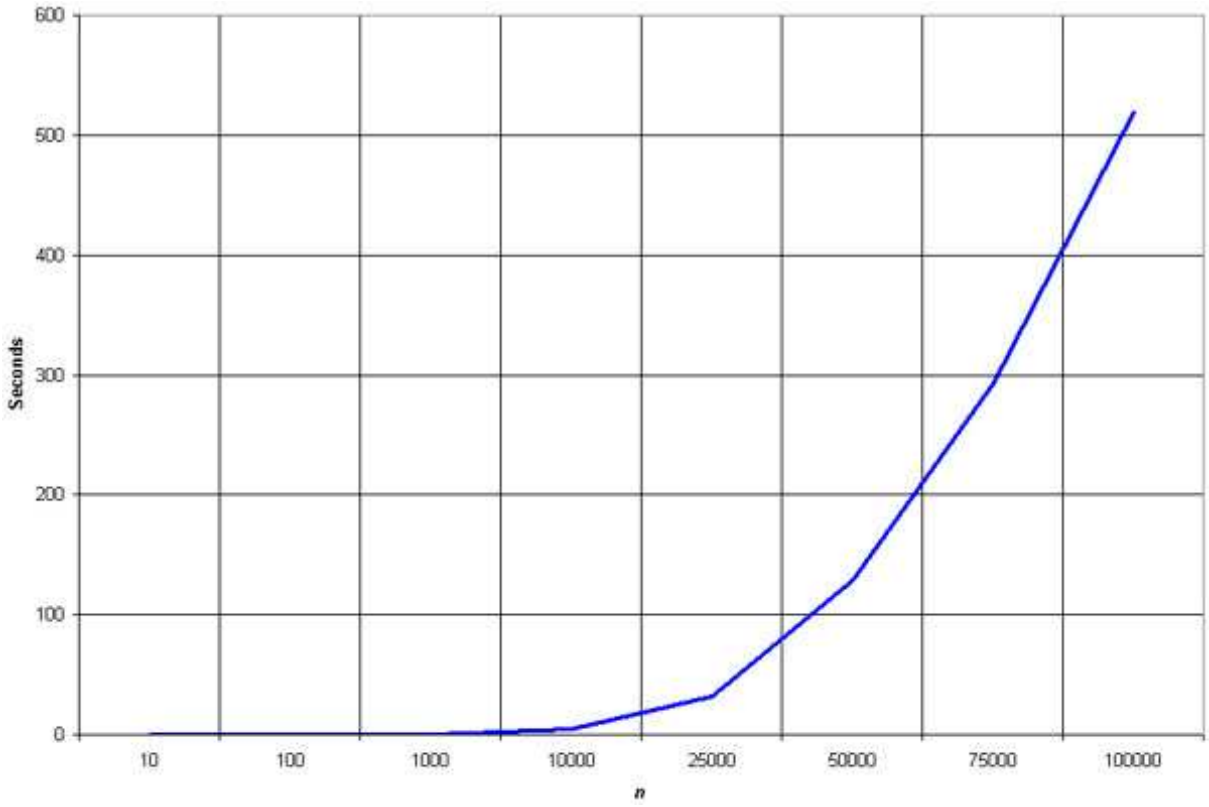
[10]Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
        {
            if (array[j] < array[min])
            {
               min = j;
            }
        }
            if (array[min] < array[i])
            {
                int tmp = array[min];
                array[min] = array[i];
                array[i] = tmp;
            }
    }
}
```

4.3.1.5 Parametry algoritmu

- Stabilní řazení : NE
- Časová složitost : $\Theta(n^2)$
- Potřeba dalších datových struktur : $\Theta(1)$

4.3.1.6 Graf výkonnosti



Obrázek 11 - Graf výkonnosti Selectionsort[11]

4.3.2 Heapsort

4.3.2.1 Úvodem o algoritmu

Heapsort nebo také řazení haldou patří mezi nejlepší obecné algoritmy řazení, založené na porovnání prvků. Nicméně je v průměru pomalejší než třídící algoritmus Quicksort. Základ pro Heapsort je datová struktura označovaná jako halda (angl. heap). Tato struktura umí efektivně provést operace vložení prvku a výběr největšího prvku. V praxi lze haldu vystavět přímo na poli způsobem, že jsou následovníci prvku n uložení do prvků 2n a 2n+1 a následné vybírání prvků lze provádět přeuspořádáváním dat v poli.

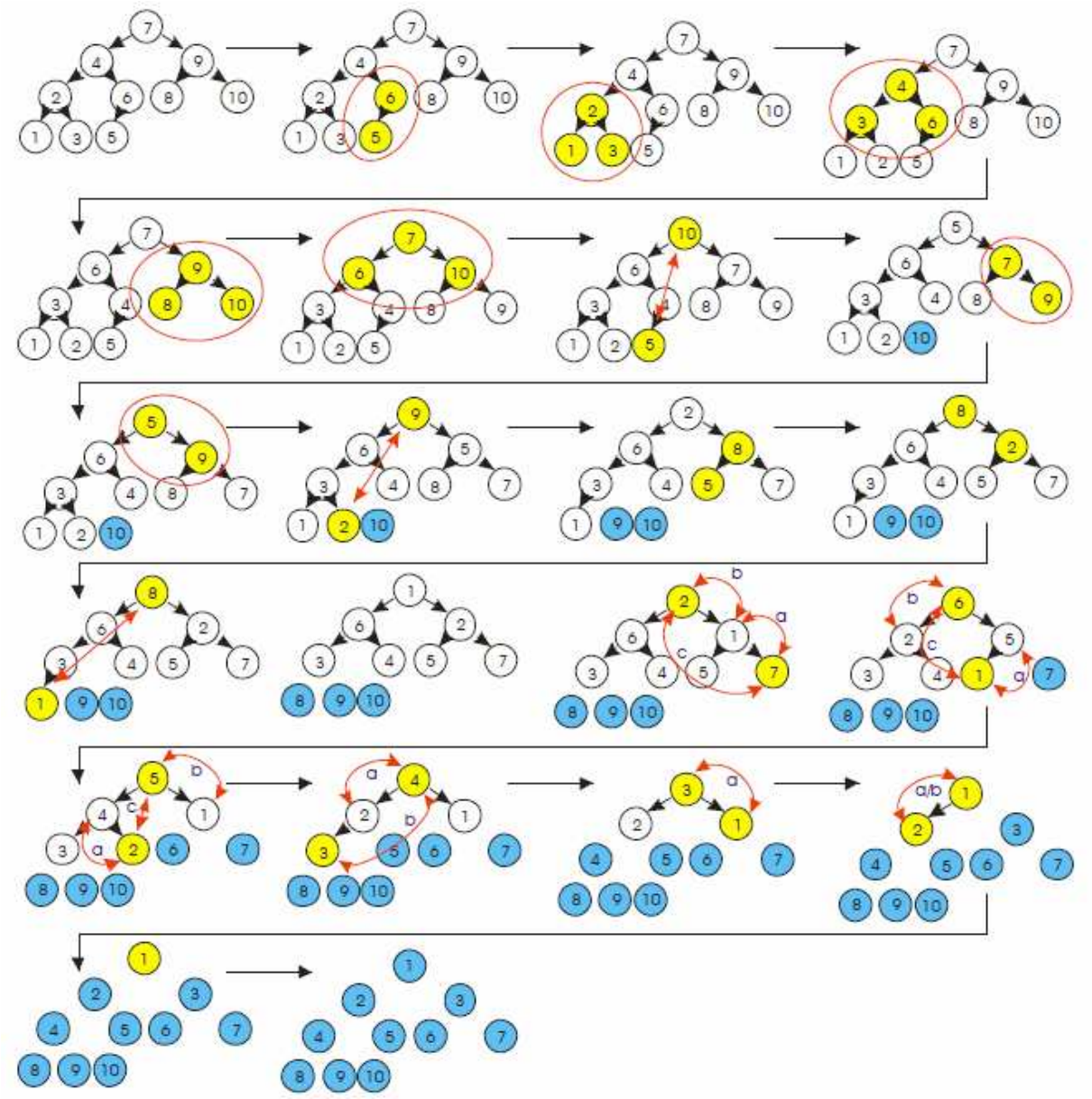
4.3.2.2 Popis algoritmu

- Nejprve vybudujeme haldu nad vybranou tabulkou dat.

[11] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

- Vezmeme vrchol haldy (Prvek s nejvyšší prioritou) a prohodíme s posledním prvkem haldy.
- Haldu zkrátíme o jeden prvek. Poslední prvek (bývalý vrchol haldy) už nemusíme počítat.
- Prvek n, který je nyní na vrcholu haldy, prohodíme s prvkem n+1 nebo 2n+1 podle toho, který má vyšší prioritu.
- Dále potom stejný prvek n prohazujeme s prvkem n+1 nebo 2n+1 podle toho, který má vyšší prioritu. Pokud prvky n+1 a 2n+1 mají vyšší prioritu než prvek n nebo tabulka další prvky neobsahuje, postupujeme na další bod.
- Dokud má halda prvky, cyklus opakujeme od bodu 2.

4.3.2.3 Ukázka třídění



Obrázek 12 - Třídící algoritmus Heapsort[12]

4.3.2.4 Algoritmus

```
public static void HeapSort(int[] array)
{
    for(int i = array.Length/2 - 1; i >=0; i--)
    {
        repairTop(array, array.Length - 1, i);
    }
```

[12] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

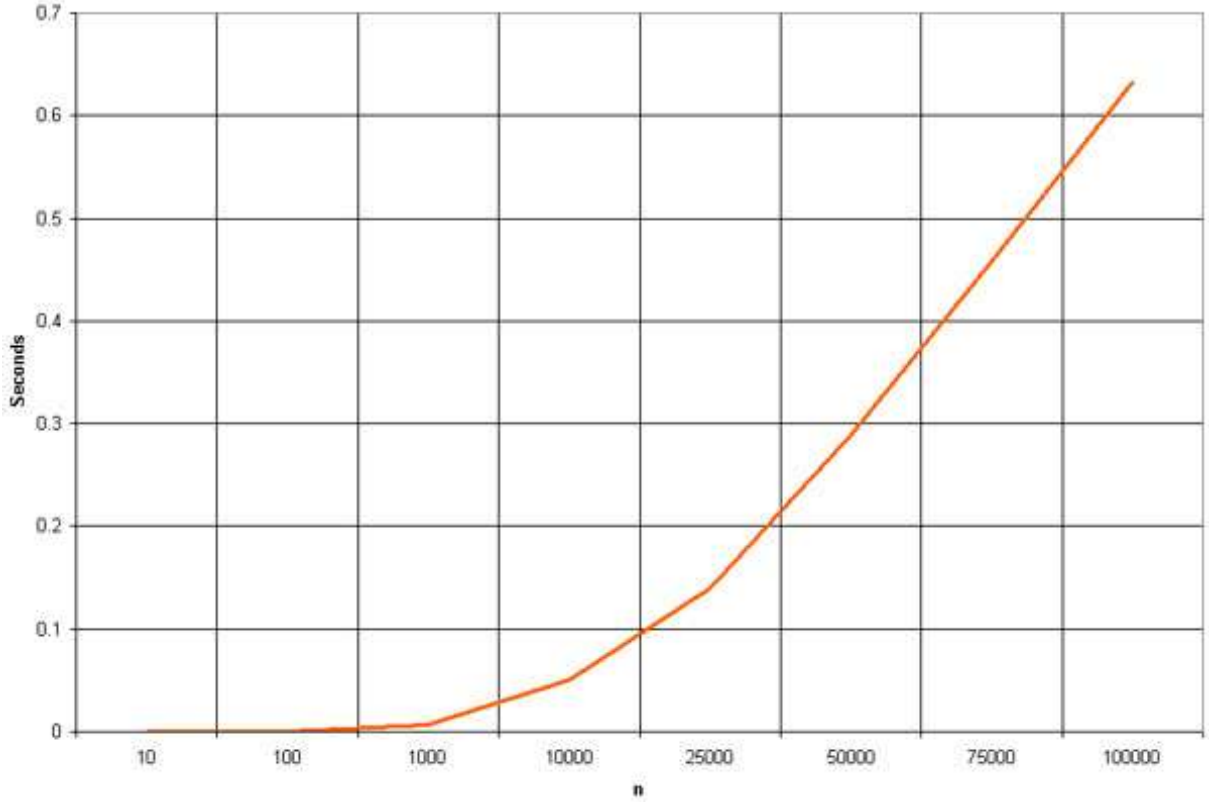
```
     for(int i = array.Length - 1; i > 0; i--)
     {
          swap(array, 0, i);
          repairTop(array, i-1, 0);
     }
 }
 private static void repairTop(int[] array, int bottom, int
topIndex)
 {
     int tmp = array[topIndex];
     int succ = topIndex*2 + 1;
     if(succ  <  bottom  &&  array[succ]  >  array[succ+1])
succ++;

     while(succ <= bottom && tmp > array[succ])
     {
         array[topIndex] = array[succ];
         topIndex = succ;
         succ = succ*2+1;
         if(succ  <  bottom  &&  array[succ]  >  array[succ+1])
succ++;
     }
     array[topIndex] = tmp;
 }
 private static void swap(int[] array, int left, int right)
 {
     int tmp = array[right];
     array[right] = array[left];
     array[left] = tmp;
 }
```

4.3.2.5 Parametry algoritmu

- Stabilní řazení : NE
- Časová složitost : Θ(n*log(n))
- Potřeba dalších datových struktur : Θ(1)

4.3.2.6 Graf výkonnosti



Obrázek 13 - Graf výkonnosti Heapsort[13]

4.4 Třídění výměnou

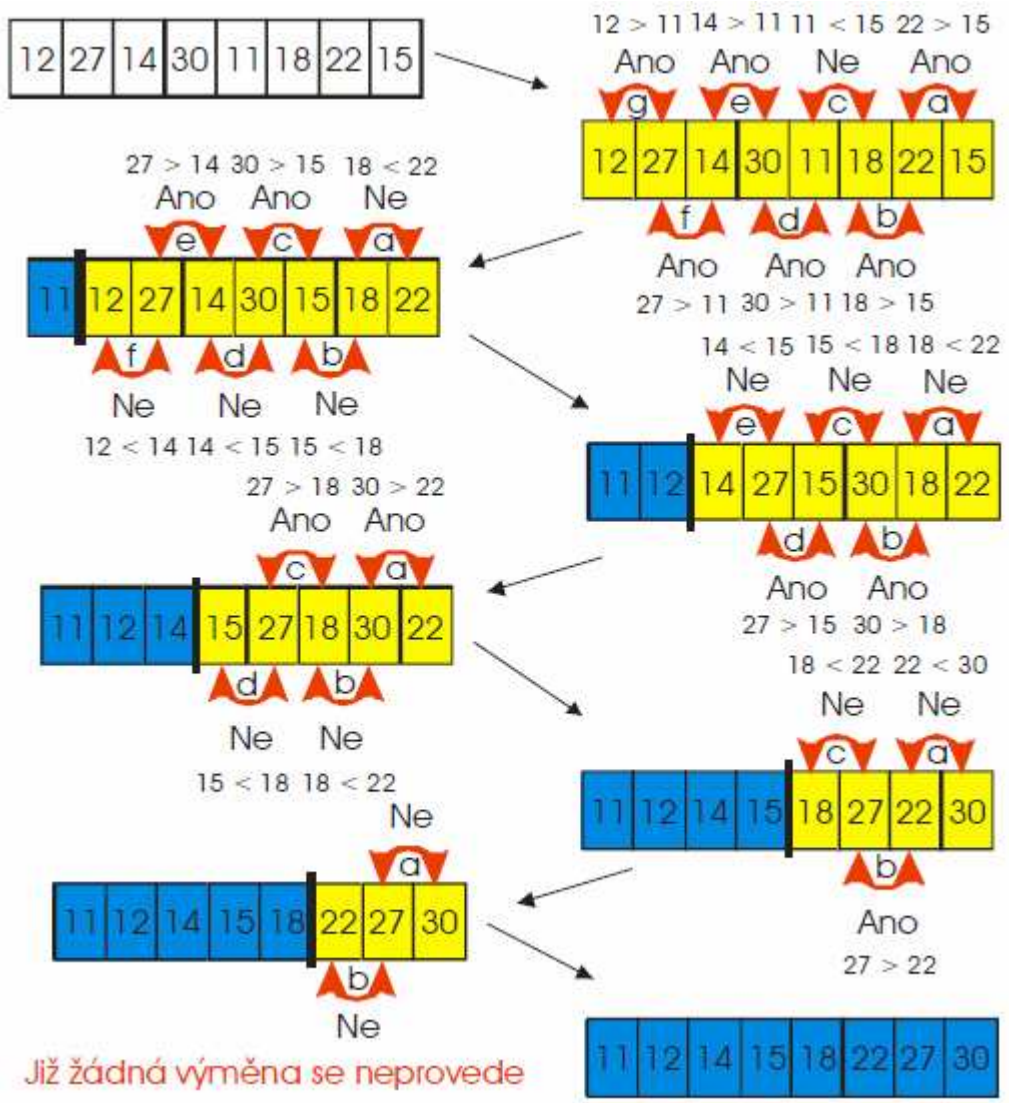
4.4.1 Bubble sort

4.4.1.1 Úvodem o algoritmu

Tento algoritmus je ze všech algoritmů nejjednodušší, ale také nejpomalejší. Spočívá v jednoduché záměně dvou sousedních prvků, pokud nevyhovují dané podmínce. Algoritmus obsahuje dva do sebe vnořené cykly, vnější zajišťuje zmenšování počtu prvků a vnitřní cyklus zajišťuje vlastní výměny dvou prvků. Existuje druhá, rychlejší metoda algoritmu Bubble sort, která délku vnitřního cyklu zmenší o jeden cyklus. Bubblesort má varianty Shakersort, Ripplesort a Shuttlesort.

[13] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

4.4.1.2 Ukázka třídění



Obrázek 14 - Třídící algoritmus Bubblesort[14]

4.4.1.3 Algoritmus

```
public static void BubbleSort(int[] array)
{
    for (int i = 0; i <= array.Length - 2; i++)
    {
       for (int j = 0; j <= array.Lenght - 2; j++)
       {
          if (array[j] < array[j + 1])
          {
              int tmp = array[j];
```

[14] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
                array[j] = array[j + 1];
                array[j + 1] = tmp;
            }
        }
    }
}
```

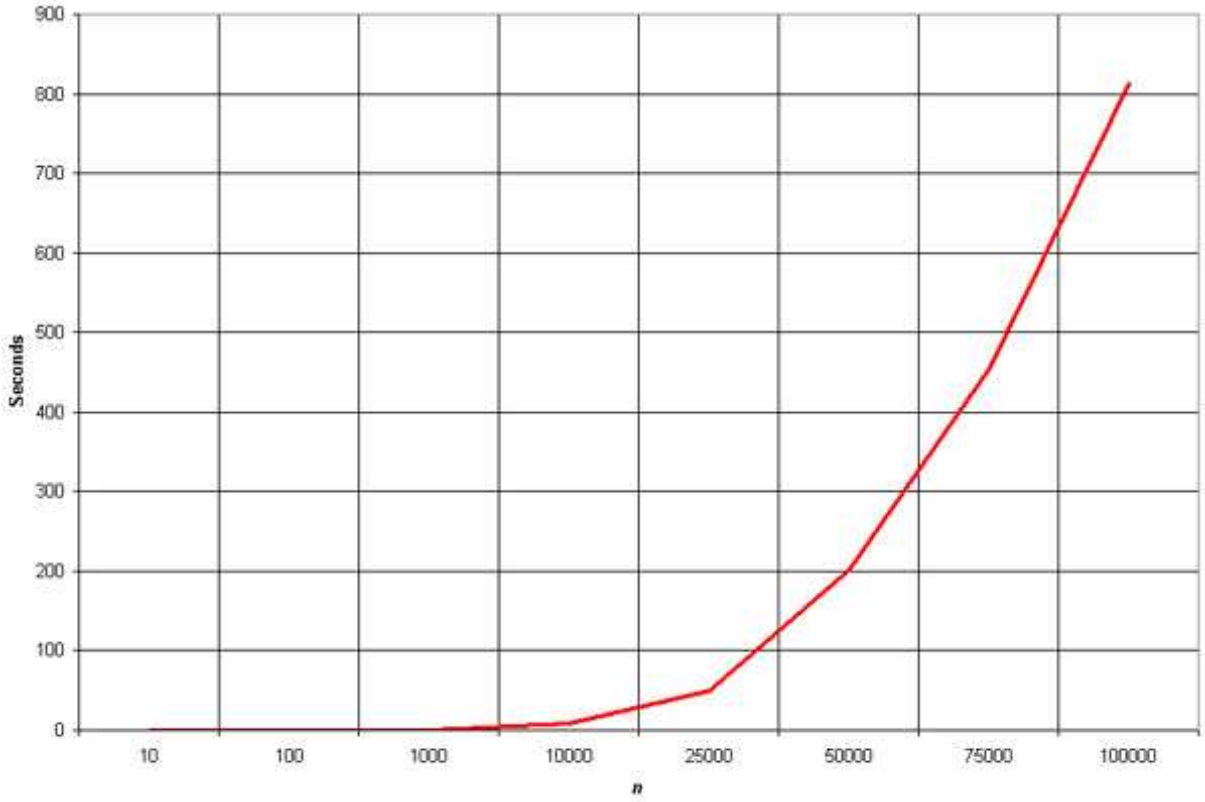
4.4.1.4 Algoritmus metody s klesajícím krokem

```
public static void BubbleSort(int[] array)
{
    for (int i = 0; i <= array.Length - 2; i++)
    {
       for (int j = 0; j <= array.Lenght - 2 - i; j++)
       {
          if (array[j] < array[j + 1])
          {
              int tmp = array[j];
              array[j] = array[j + 1];
              array[j + 1] = tmp;
          }
       }
    }
}
```

4.4.1.5 Parametry algoritmu

- Stabilní řazení : ANO
- Časová složitost : Θ(n2)
- Potřeba dalších datových struktur : Θ(1)

4.4.1.6 Graf výkonnosti



Obrázek 15 - Graf výkonnosti Bubblesort[15]

4.5 Quicksort

4.5.1.1 Úvodem o algoritmu

Tento rychlý, rekurzivní algoritmus je původní a není odvozený z jiného algoritmu, třídí na principu rozděl a panuj. Algoritmus vymyslel jistý pan Sir Charles Antony Richard Hoare v roce 1962. Základní myšlenkou Quicksortu je rozdělit posloupnosti prvků na dvě přibližně stejně velké části. U tohoto algoritmu volíme tzv. Pivota, to jest prvek, který leží uprostřed posloupnosti.

Princip tohoto algoritmu spočívá v rozdělení původní posloupnosti do N podposloupností, pro které platí, že všechny prvky první podposloupnosti jsou menší než všechny prvky druhé podposloupnosti, to samé platí s dalšími podposloupnostmi. Poté se

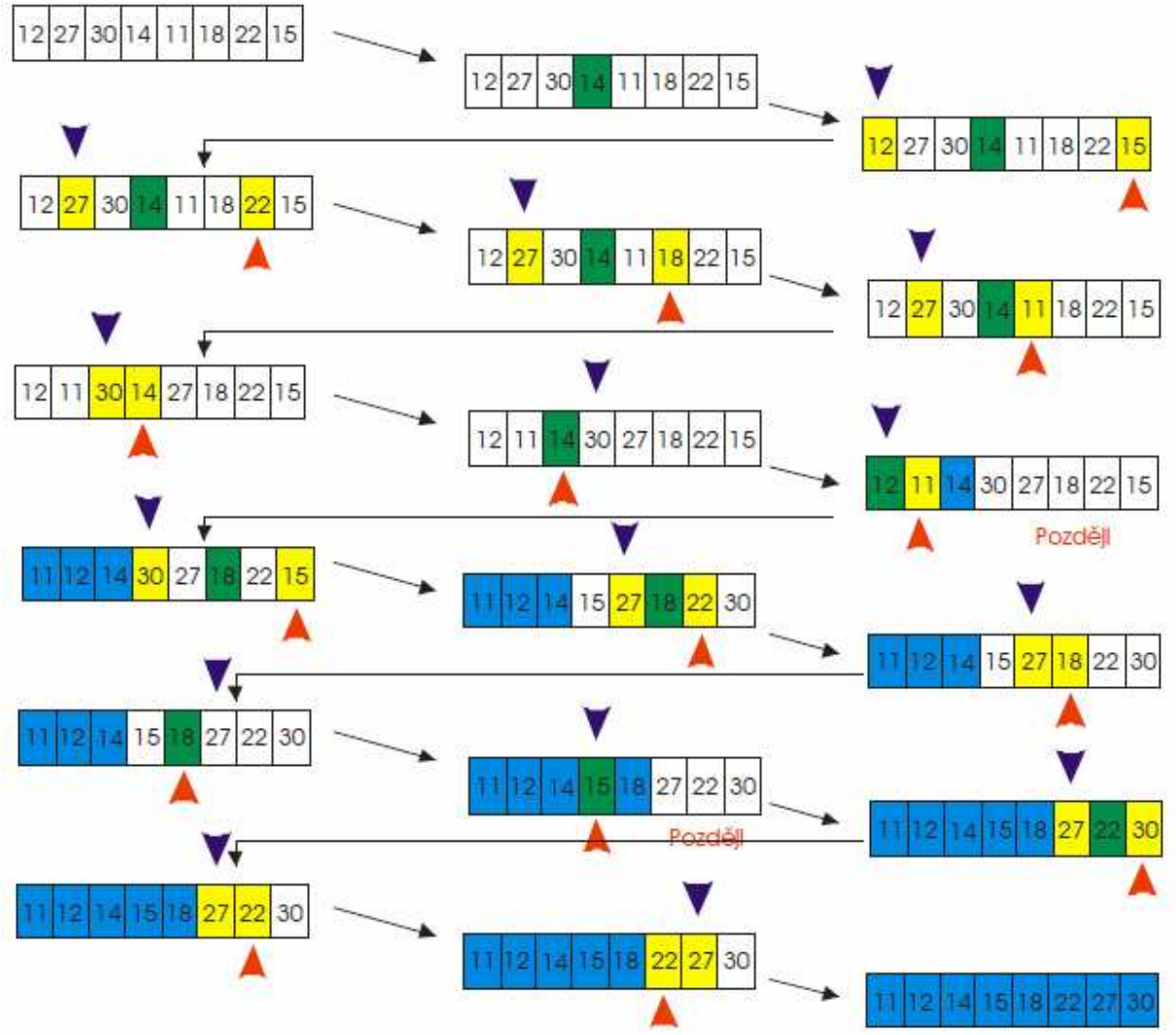
[15] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

každá posloupnost rozdělí na N' podpodposloupností, pro které také platí podmínka rostoucí nerovnosti, atd. Rozdělování skončí, až jsou podposloupnosti prázdné.

4.5.1.2 Popis algoritmu

- Na začátku si zvolíme prvek tzv. Pivot, který je uprostřed posloupnosti prvků.
- Poté procházíme posloupnost od začátku i od konce a porovnáváme prvky s pivotem. V levé části posloupnosti musí být menší prvky jak pivot a v pravé části větší.
- Jakmile narazíme na dvojici prvků, jeden z levé a druhý z pravé strany, které podmínku nesplňují, dané prvky prohodíme.
- Takto pokračujeme dokud index levého prvku není větší nebo roven indexu pravého prvku. Vzniknou nám dvě podposloupnosti.
- Nakonec daný algoritmus opakujeme na každou podposloupnost zvlášť.
- Třídění je ukončeno ve chvíli, kdy délka posloupnosti je 1.

4.5.1.3 Ukázka třídění



Obrázek 16 - Třídící algoritmus Quicksort[16]

4.5.1.4 Algoritmus

```
public static void Quicksort(int[] array)
{
    int levy = 1;
    int pravy = array.Lenght;
    Quick(levy, pravy, array);
}
private static void Quick(int levy, int pravy, int[] array)
{
    int pivot;
    int i, j = 0;
```

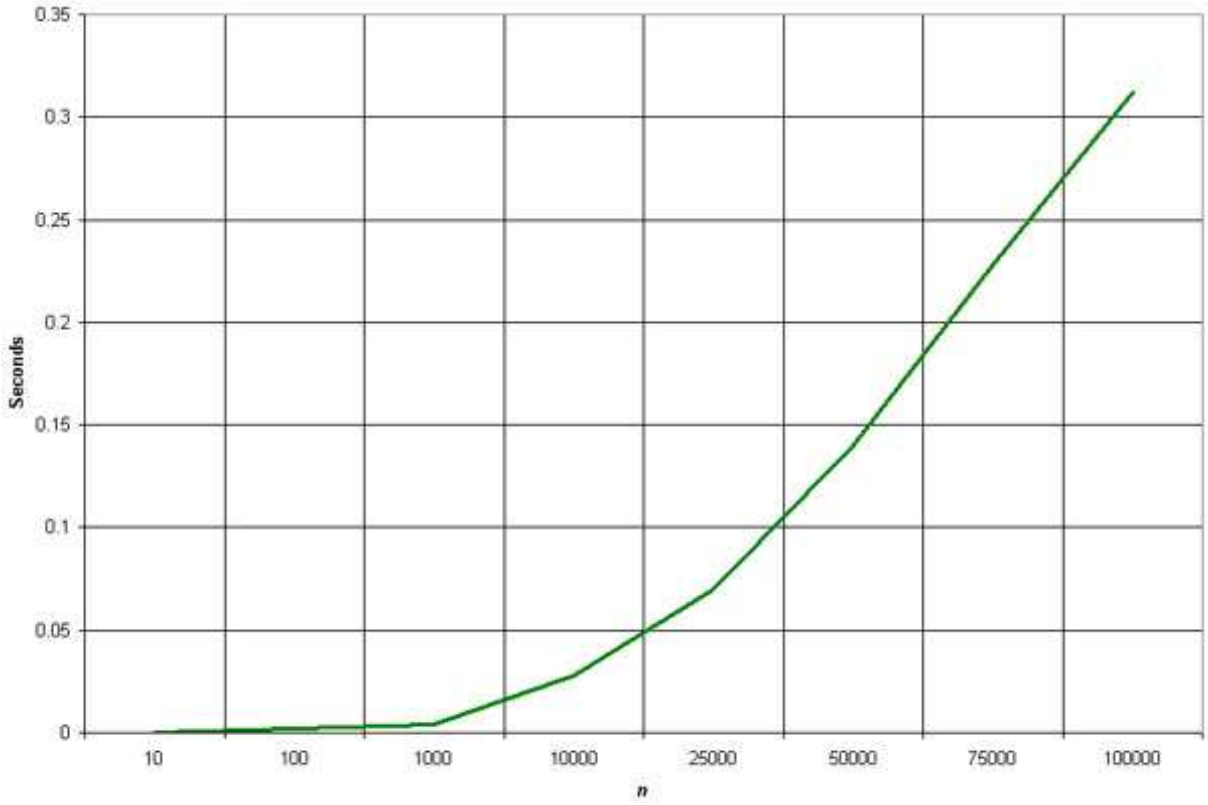
[16] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
    i = levy;
    j = pravy;
    pivot = array.Length / 2;
    do
    {
        while (array[i - 1] < pivot)
        {
            i++;
        }
        while (Pivot < array[j - 1])
        {
            j--;
        }
        if (i <= j)
        {
            int tmp = array[i - 1];
            array[i - 1] = array[j - 1];
            array[j - 1] = tmp;
            i++;
            j--;
        }
    } while (!(i > j));
    if (levy < j)
    {
        Quick(levy, j, array);
    }
    if (i < pravy)
    {
        Quick(i, pravy, array);
    }
}
```

4.5.1.5 Parametry algoritmu

- Stabilní řazení : NE
- Časová složitost : Θ(n2)
- Očekávaná časová složitost : Θ(n*log(n))
- Potřeba dalších datových struktur : Θ(1)

4.5.1.6 Graf výkonnosti



Obrázek 17 - Graf výkonnosti Quicksort[17]

4.5.2 Shakersort

4.5.2.1 Úvodem o algoritmu

Shakersort nebo taktéž Shakesort varianta Bubblesortu prochází pole střídavě zleva doprava a zprava doleva. Seřazené části pole jsou v průběhu řazení na obou koncích pole a při ukončení řazení se spojí. Tento postup připomíná práci barmana se shakerem, kterým pohybuje nahoru a dolů, proto algoritmus dostal název shakersort.

4.5.2.2 Algoritmus

```
public static void ShakerSort(int[] array)
{
    for (int i = 0; i < array.Length/2; i++)
    {
        bool swapped = false;
        for (int j = i; j < array.Length - i - 1; j++)
```

[17] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

```
        {
            if(array[j] > array[j + 1])
            {
                int tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
                swapped = true;
            }
        }
        for (int j = array.Length - 2 - i; j > i; j--)
        {
            if(array[j] < array[j - 1])
            {
                int tmp = array[j];
                array[j] = array[j - 1];
                array[j - 1] = tmp;
                swapped = true;
            }
        }
        if(!swapped) break;
    }
}
```

4.5.2.3 Parametry algoritmu

- Stabilní řazení : ANO
- Časová složitost : $\Theta(n^2)$
- Potřeba dalších datových struktur : $\Theta(1)$

4.5.3 Shuttlesort

4.5.3.1 Úvodem o algoritmu

Další varianta Bubblesortu zvaná Shuttlesort pracuje tak, že dojde-li u dvojice k výměně (např. při průchodu zleva doprava), vrací se metoda s prvkem, který se posunuje doleva tak dlouho, dokud dochází k výměně. Pak se vrací do pozice u níž ukončila posun doprava a pokračuje směrem vpravo. Metoda končí, porovná-li úspěšně poslední dvojici prvků.

4.5.4 RippleSort

4.5.4.1 Úvodem o algoritmu

Poslední varianta Bubblesortu, která si pamatuje pozici první dvojice u které došlo k výměně. V dalším cyklu začíná porovnávat až od předcházející dvojice.

4.5.4.2 Algoritmus

```
public static void RippleSort(int[] array)
{
    bool vymena = true;
    for (int i = 0; i <= array.Length - 2; i++)
    {
        vymena = true;
        for (int j = 0; j <= array.Lenght - 2; j++)
        {
            if (array[j] < array[j + 1])
            {
                vymena = false;
                int tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
            }
        }
        if (vymena == true)
        {
            break;
        }
    }
}
```

4.6 Třídění spojováním

4.6.1 Mergesort

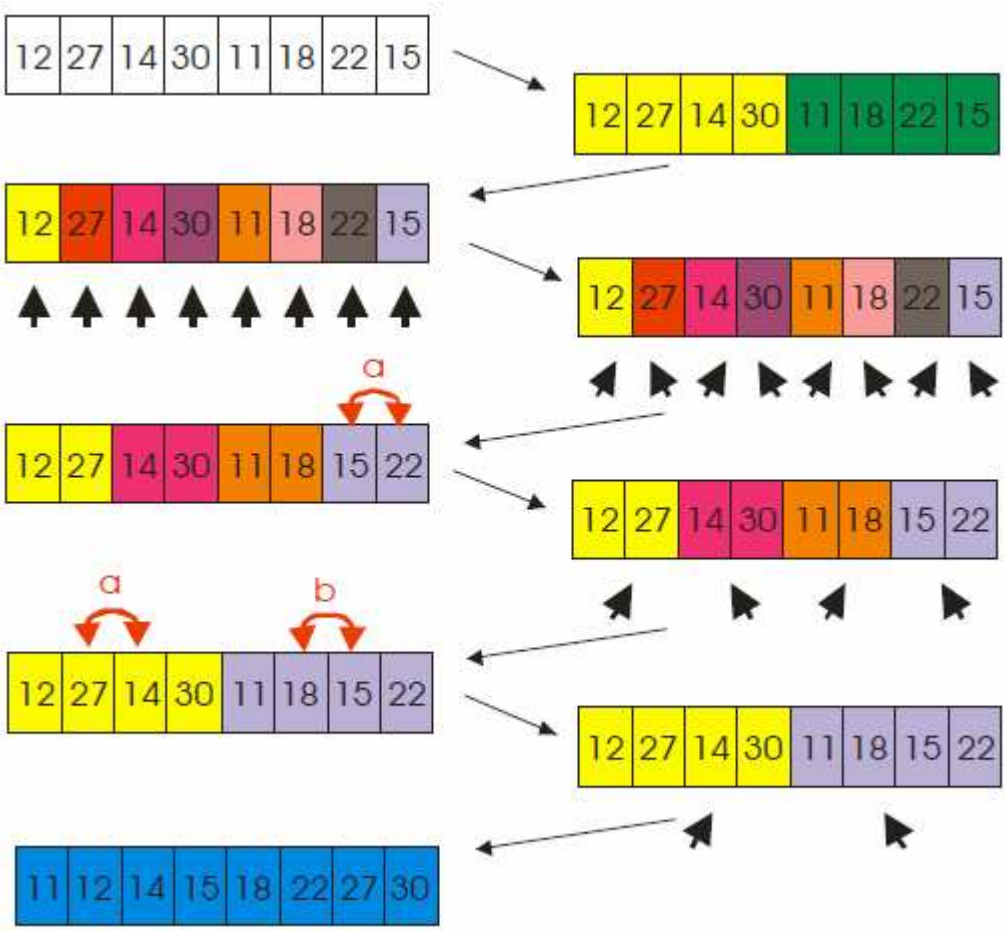
4.6.1.1 Úvodem o algoritmu

Tento algoritmus vymyslel pan John Von Neumann v roce 1945. Mergesort patří mezi algoritmy, které třídí na principu rozděl a panuj. Algoritmus pracuje na principu sekvenčního setřídění dvou seřazených posloupností prvků do jedné. Algoritmus se většinou uplatňuje při třídění na externích paměťových médiích.

4.6.1.2 Popis algoritmu

- Nejprve rozdělíme nesetříděnou posloupnost na dvě přibližně stejné posloupnosti.
- Poté rekurzivně setřídíme prvky v obou posloupnostech s využitím MergeSort.
- Nakonec spojíme dvě utříděné posloupnosti do jedné.

4.6.1.3 Ukázka třídění



Obrázek 18 - Třídící algoritmus Mergesort[18]

4.6.1.4 Algoritmus

```
 public static void MergeSort(int[] array, int[] aux, int
left, int right)
 {
     if(left == right) return;
```

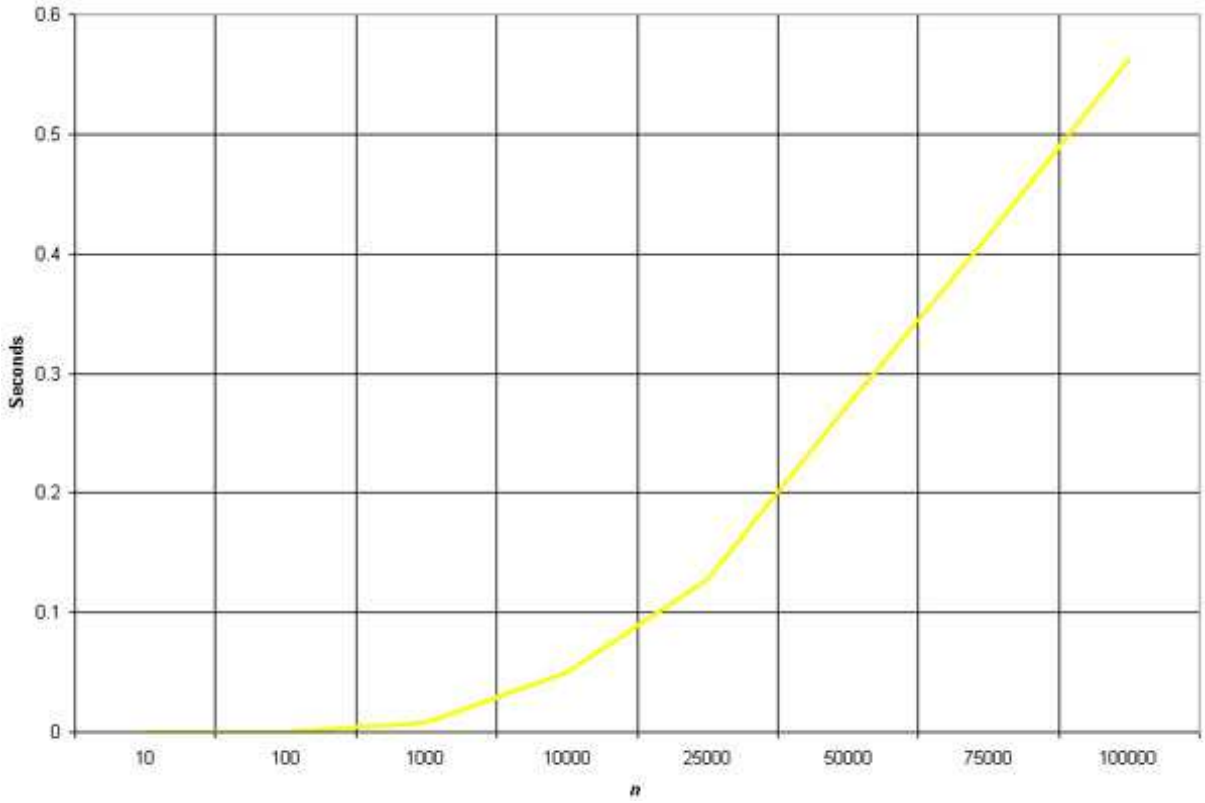
[18] Zdroj: PACLT, Radek. Porovnání a prezentace algoritmů třídění tabulek, Bakalářská práce.

```
     int middleIndex = (left+right)/2;
     MergeSort(array, aux, left, middleIndex);
     MergeSort(array, aux, middleIndex + 1, right);
     merge(array, aux, left, right);
     for(int i = left; i <= right; i++)
     {
         array[i] = aux[i];
     }
 }
 private static void merge(int[] array, int[] aux, int left,
int right)
 {
     int middleIndex = (left+right)/2;
     int leftIndex = left;
     int rightIndex = middleIndex + 1;
     int auxIndex = left;
     while(leftIndex <= middleIndex && rightIndex <= right)
     {
         if(array[leftIndex] >= array[rightIndex])
         {
             aux[auxIndex] = array[leftIndex++];
         }
         else
         {
             aux[auxIndex] = array[rightIndex++];
         }
          auxIndex++;
      }
      while(leftIndex <= middleIndex)
      {
         aux[auxIndex] = array[leftIndex++];
         auxIndex++;
      }
      while(rightIndex <= right)
      {
          aux[auxIndex] = array[rightIndex++];
          auxIndex++;
      }
  }
```

4.6.1.5 Parametry algoritmu

- Stabilní řazení : ANO
- Časová složitost : Θ(n*log(n))
- Potřeba dalších datových struktur : Θ(n)

4.6.1.6 Graf výkonnosti



Obrázek 19 - Graf výkonnosti Mergesort[19]

[19] Zdroj: Sorting algorithms, linux.wku.edu/~lamonml/algor/sort

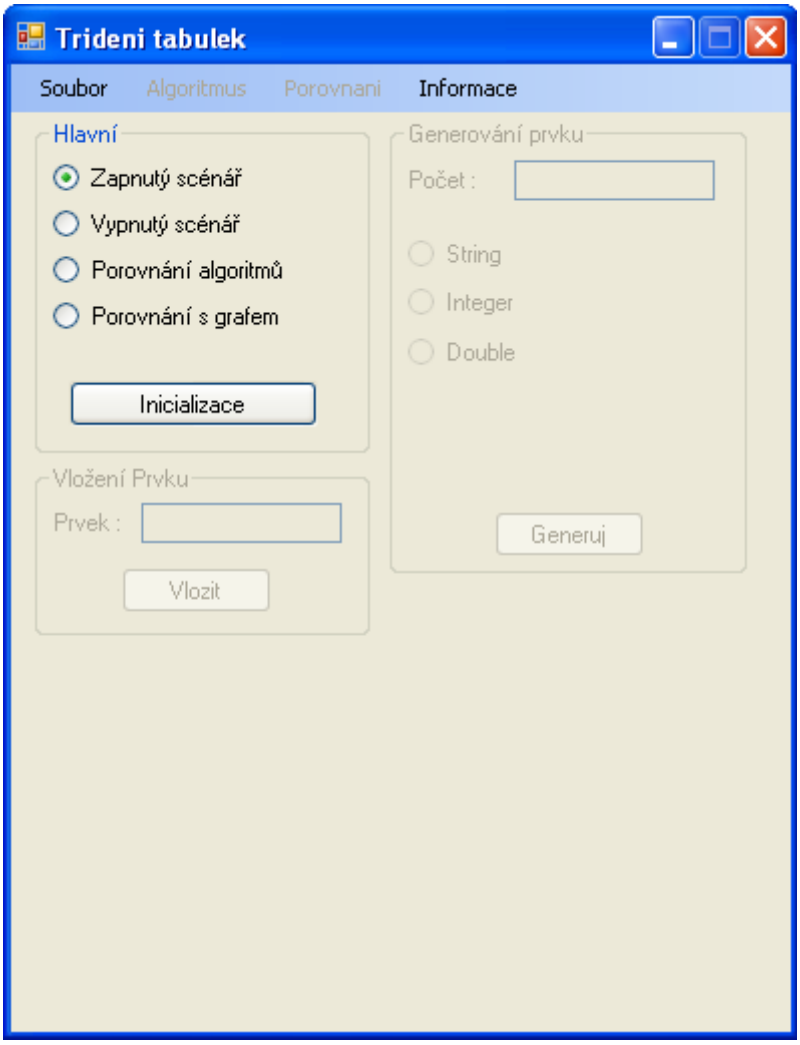
5 UŽIVATELSKÁ ČÁST APLIKACE

Aplikace sloužící k animaci a porovnávání algoritmů byla programována v jazyce C#, ve vývojovém nástroji Visual Studio 2008 s použitým .NET frameworkem 3.5 SP1. Aplikace podporuje třídění tabulek s prvky o třech datových typech. Na začátku automaticky zjistí o jaký datový typ se jedná a podle toho přizpůsobí tabulku. Tím nám nastává jeden vážný problém. Číslo může být datového typu `integer` a `double`, stejně tak jako datového typu `string`. Proto do datového typu `string` vkládám jenom znaky „a-z“ a „A-Z“ a píšu-li v této práci o datovém typu string, mám na mysli můj „upravený“ `string`.

5.1 Uživatelský popis aplikace

Daná aplikace se skládá ze čtyř funkčních částí:

- Grafická animace průběhu algoritmu.
- Zobrazení doby třídění, počtu porovnání a počtu prohození jednoho zvoleného algoritmu.
- Zobrazení doby třídění, počtu porovnání a počtu prohození na více zvolených algoritmech.
- Zobrazení doby třídění, počtu porovnání a počtu prohození na více zvolených algoritmech s možností zobrazení doby třídění do grafu.



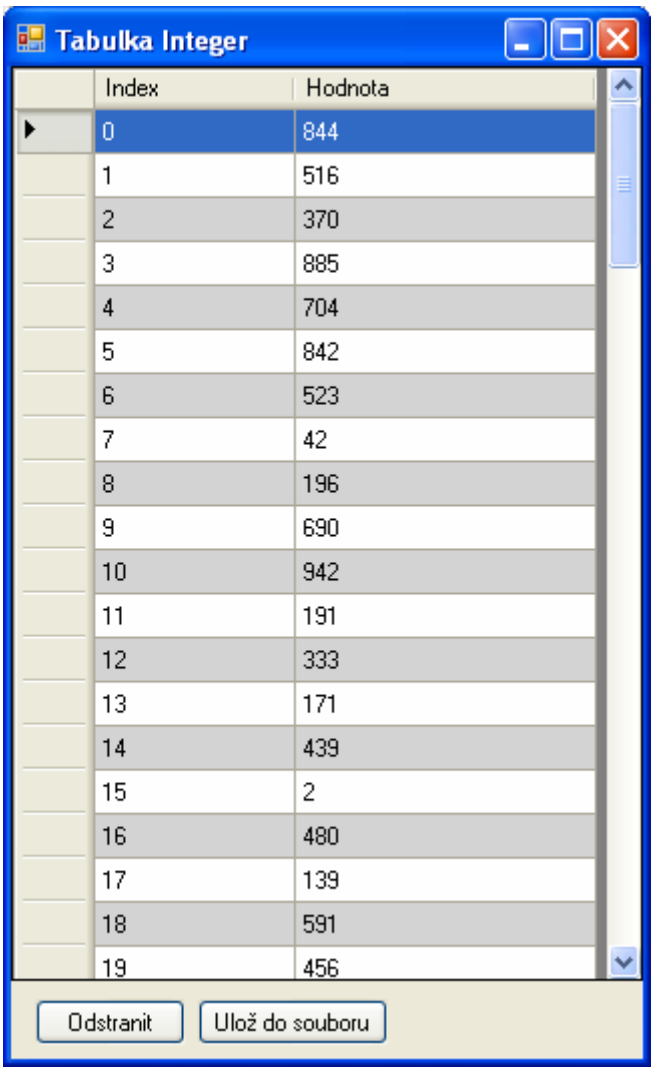
Obrázek 20 - Ukázka hlavního formuláře aplikace[20]

Daná funkční část se vybere zaškrtnutím radiobuttonu v Groupboxu se jménem: „Hlavní“. Poté se musí stisknout tlačítko inicializace.

Nyní je potřeba vytvořit tabulku a poté ji naplnit daty. Toto můžeme udělat několika způsoby. První způsob je zadat data přímo **z klávesnice**, toto se provádí v Groupboxu se jménem: „Vložení Prvku“. Vkládat můžeme řetězce, čísla i desetinná čísla. Jako řetězec se zde rozumí pouze množina písmen, jak velkých, tak malých. Toto opatření je z důvodu jednoznačnosti čísla a řetězce (jak bylo řečeno v úvodu). Jako desetinné číslo se rozumí číslice oddělené znakem „.“. Podle prvních vložených dat si aplikace sama zvolí jeden ze tří typů tabulky. Data, která nepatří do žádného typu, se automaticky „zahazují“. Další ze

[20] Zdroj: Vlastní

způsobů vložení dat je přes **generátor**, který najdeme v Groupboxu se jménem „Generování prvků“. Třetí možnost je vložení dat **ze souboru**, kliknutí v menu Soubor a Otevřít. Všechny způsoby lze mezi sebou kombinovat. Data se mohou prohlédnout nebo upravit kliknutím na položku s názvem: „Soubor” a poté na položku: “Zobraz hodnoty”. Data z tabulky můžeme v menu Soubor poté kliknutím na položku Uložit ukládat. Máme-li zvolenou funkci animace průběhu algoritmu, lze do tabulky vložit maximálně 20 prvků. Při vyšším počtu by se některé prvky zobrazili mimo obrazovku. U ostatních funkcí je maximum nastaven na 1.000.000.



Obrázek 21 - Ukázka formuláře zobrazujícího hodnoty prvků tabulky[21]

[21] Zdroj: Vlastní

5.2 Ukázka animace

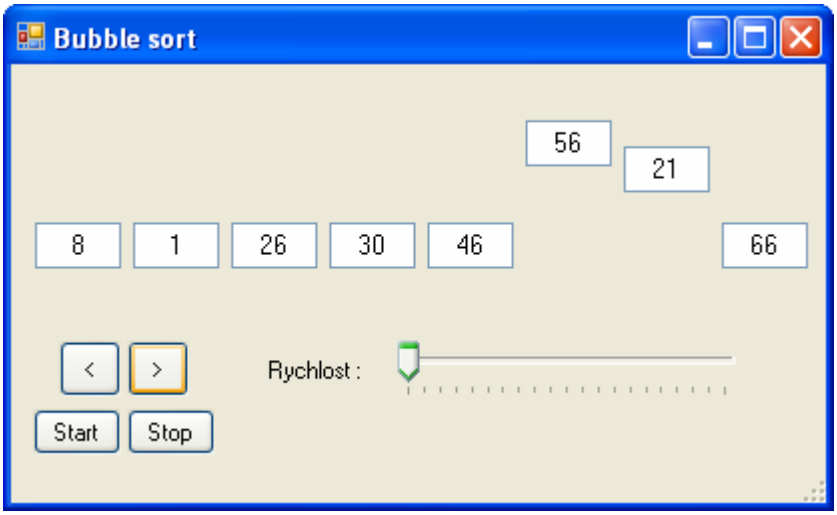
Po naplnění tabulky daty můžeme začít pracovat s třídícími algoritmy. Při zvolení funkce animace průběhu algoritmu zvolíme třídící algoritmus kliknutím v menu na položku s názvem „Algoritmus“, kde si můžeme vybrat jeden z daných třídících algoritmů, kromě algoritmu Mergesort. Tento algoritmus se nepodařilo animovat s důvodu složitosti algoritmu. Poté se nám nemodálně spustí animační formulář, kde si můžeme danou animaci prohlédnout.

Animace algoritmu může posloužit jako výukový materiál. Má jednoduché a přehledné ovládání. Princip animace funguje na bázi změny lokace Textboxu. Jako podklad pro samotné vykreslení animace slouží formulář, který obsahuje čtyři tlačítka, osm až dvacet Textboxů (podle počtu prvků v tabulce) a Trackbar na změnu rychlosti animace.

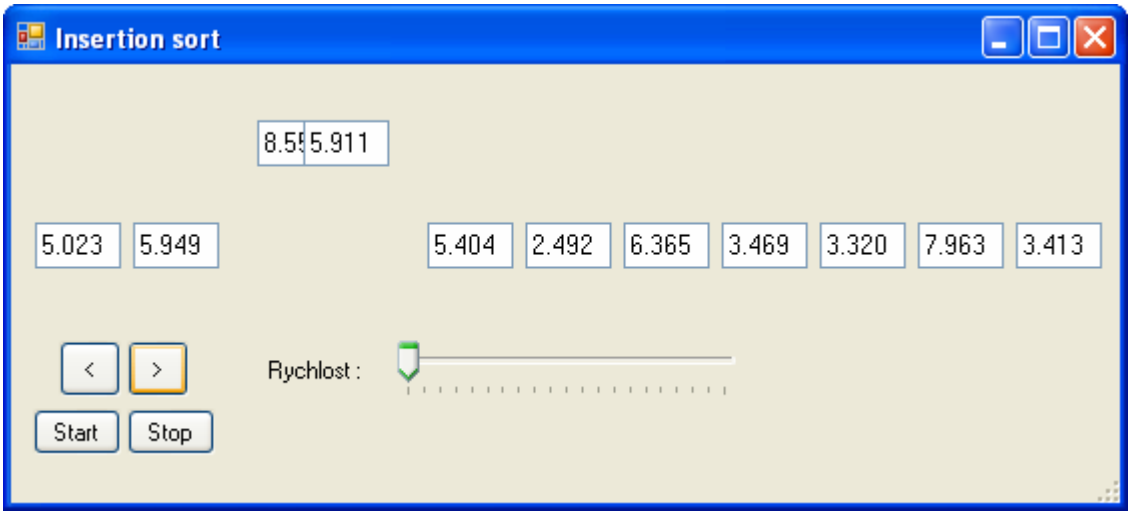
Obrázek 22 - Ukázka formuláře pro animaci[22]

Animovat můžeme prvky všech tří datový typů, ale délka Textboxu je pevně daná. Z toho plyne omezenost počtu znaků a velikosti čísla. Tlačítka Start a Stop fungují k spuštění respektive k zastavení animace. Tlačítka se šipkami potom k jednotlivým krokům. Rychlost animace se mění vždy po dokončení záměny dvojice prvků. Název algoritmu se zobrazí jako název formuláře.

[22] Zdroj: Vlastní



Obrázek 23 - Ukázka prohození dvou prvků s hodnotami typu integer[23]



Obrázek 24 - Ukázka prohození dvou prvků s hodnotami typu double[24]

5.3 Ukázka třídění tabulky jedním algoritmem

Jestliže máme zvolenou funkci Zobrazení výsledků jednoho zvoleného algoritmu, naplníme tabulku daty a zvolíme třídící algoritmus, kliknutím v menu na položku s názvem „Algoritmus“, kde si můžeme vybrat jeden z daných třídících algoritmů.

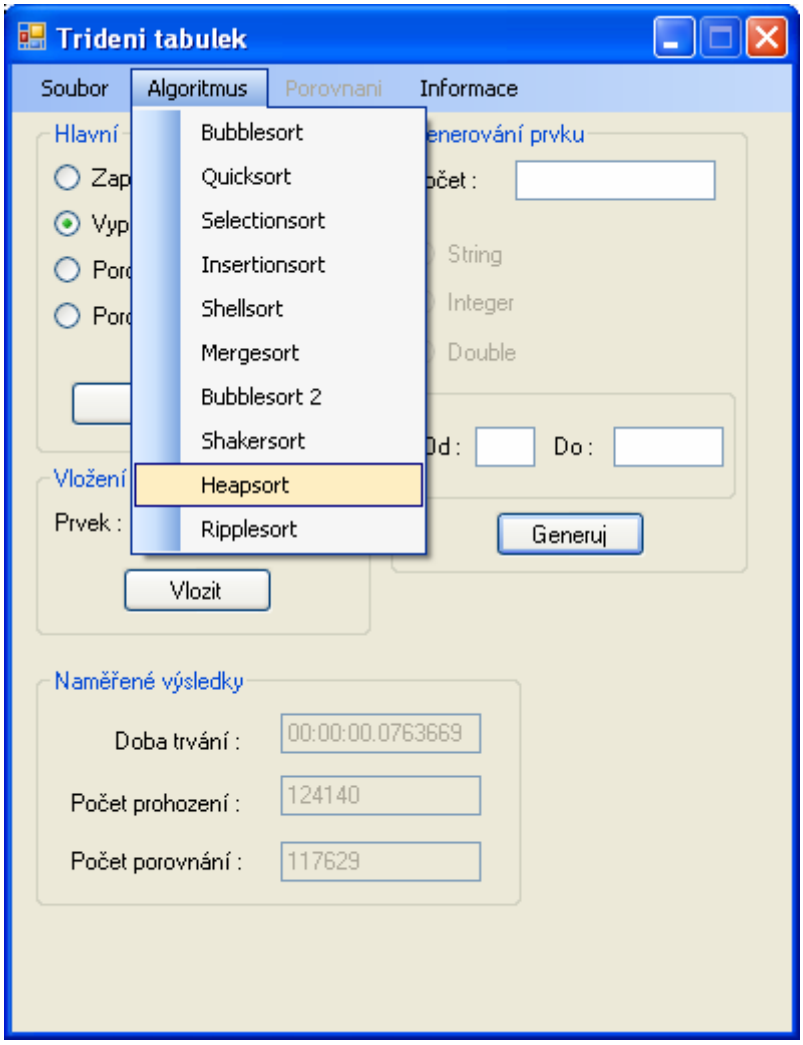
[23] Zdroj: Vlastní

[24] Zdroj: Vlastní

Třídění tabulky jedním algoritmem můžeme použít v případě, chceme-li utřídit vlastní množinu dat. Načteme si data ze souboru, provedeme třídění libovolným algoritmem a poté uložíme zpět do souboru. Nebo nás zajímají pouze výsledky jediného algoritmu. Tyto výsledky najdeme v Groupboxu s názvem „Naměřené výsledky“. Měříme veličiny:

- Dobu trvání algoritmu
- Počet prohození prvků dvou prvků
- Počet porovnání dvou prvků.

U této zvolené funkce aplikace nelze výsledky uložit do souboru.



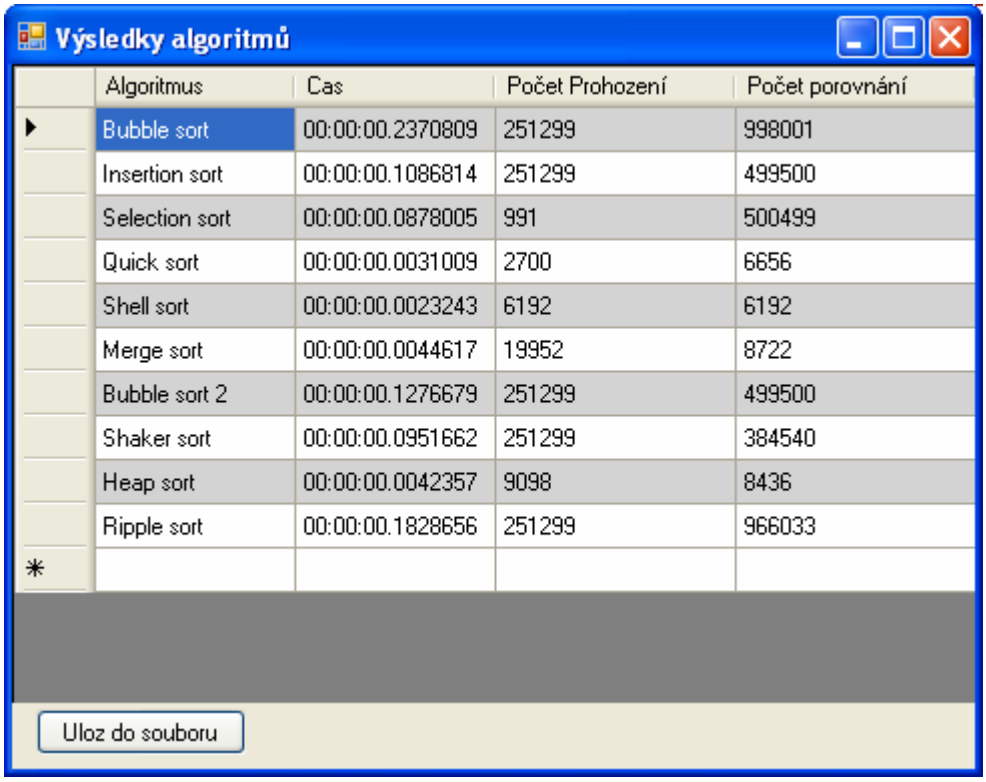
Obrázek 25 - Ukázka hlavního formuláře s výsledky[25]

5.4 Ukázka třídění tabulky více algoritmy

Tato funkční část se může použít při porovnání algoritmů podle různých, v našem případě opět třech, veličin.

Zvolíme-li porovnání výsledků více zvolených algoritmů, musíme po naplnění tabulky daty zvolit v Groupboxu s názvem: „Algoritmy“ algoritmy, které chceme nad tabulkou vykonat. Dále v menu klikneme na položku s názvem: „Porovnání“ a vybereme, zda zobrazit výsledky do formuláře nebo do souboru. Při zvolení zobrazení výsledků do formuláře můžeme srovnávat výsledky vzestupně i sestupně a po prohlédnutí dodatečně uložit do souboru.

[25] Zdroj: Vlastní



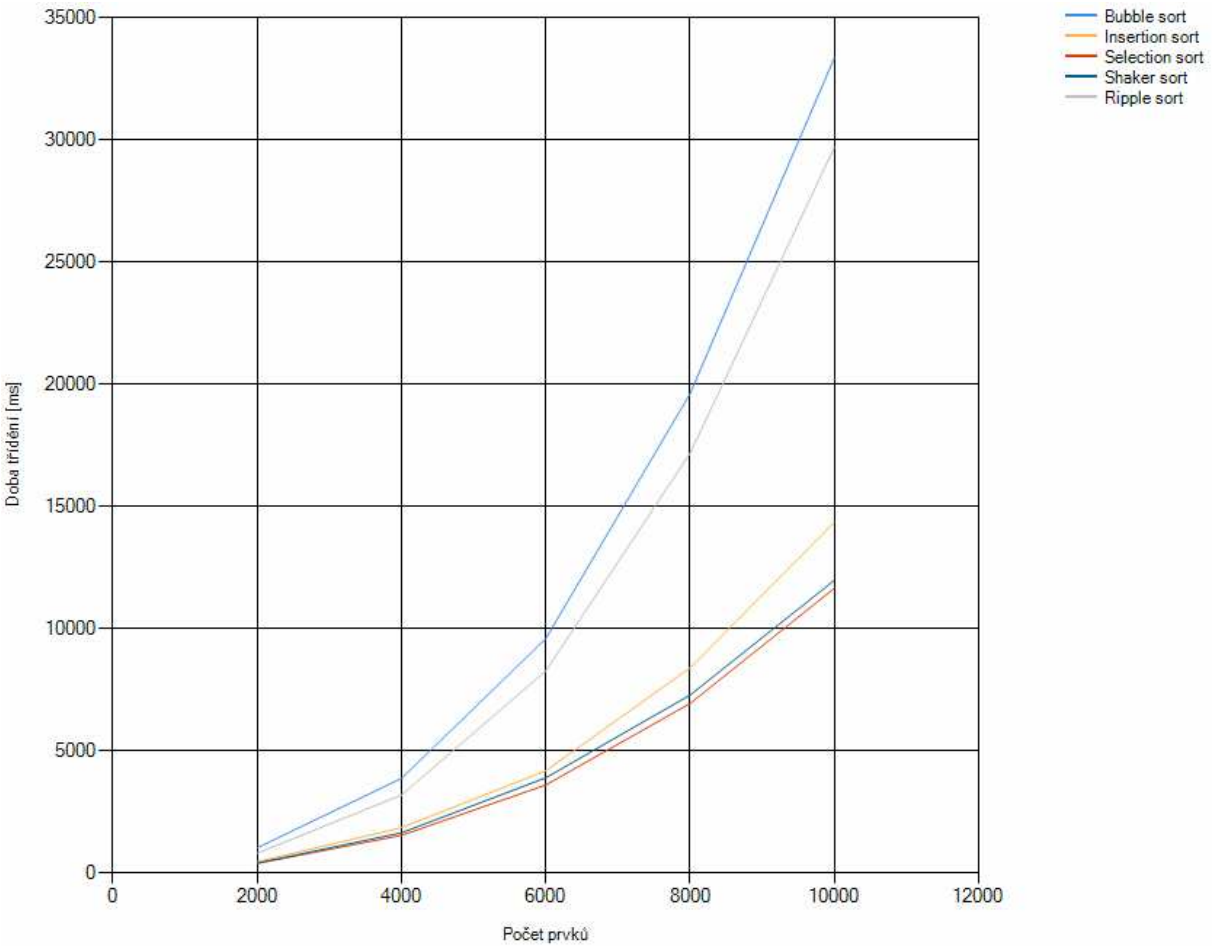
Obrázek 26 - Ukázka výsledků vybraných algoritmů[26]

5.5 Ukázka grafu

Čtvrtá funkce aplikace je podobná té předchozí s možností zobrazení doby trvání daných algoritmů do grafu. Toho dosáhneme při kliknutí ve formuláři výsledků na tlačítko Zobrazit graf.

Z grafu můžeme pak pouhým nahlédnutím zjistit, jaký algoritmus je rychlejší. Zde byla použita komponenta Chart, kterou jsem musel do Visual Studia a .NET Frameworku doinstalovat. Komponenta vyžadovala .NET Framework 3.5 SP1.

[26] Zdroj: Vlastní



Obrázek 27 - Ukázka grafu[27]

Na osách má graf počet prvků a dobu třídění v milisekundách. Zakreslí jednotlivé body do grafu a poté je spojí úsečkou. Barva úsečky se zvolí automaticky. Zobrazovat do grafu můžeme libovolný počet třídících algoritmů.

[27] Zdroj: Vlastní

6 NÁVRH A IMPLEMENTACE APLIKACE

6.1 Úvod

Na samotném začátku, než jsem začal přemýšlet o obecném návrhu struktury aplikace, jsem si musel zvolit programovací jazyk, vývojové prostředí a popřípadě nějaký framework. V úvahu připadaly jazyk Pascal a vývojové prostředí Delphi, jazyk Java a vývojové prostředí NetBeans nebo JDeveloper a jazyk C++ s vývojovým prostředím Visual Studio. Původně jsem si chtěl vybrat jazyk Java, ale zároveň jsem chtěl vytvářet program ve Visual Studiu. Nakonec jsem se dozvěděl o jazyce C# a frameworku .NET. Oboje mě velice nadchlo. Jazyk C# má s jazykem Java podobnou syntaxi a je zde možnost vytvářet aplikace ve Visual Studiu. Jako framework jsem si zvolil .NET framework verzi 3.5 SP1 kvůli komponentě Chart.

6.2 Popis implementace aplikace

Poté, co jsem si vybral programovací jazyk, vývojové prostředí a framework následovaly týdny učení jazyka C# a frameworku .NET. Po pochopení základních částí jazyka jsem vytvořil třídu `Tabulka` a abstraktní třídu `Prvek`, ze které dědily třídy `PrvekStr`, `PrvekInt` a `PrvekDbl`. Nejprve jsem chtěl naprogramovat tři třídy tabulka (každou pro jeden datový typ) a jednu třídu pro prvky, která byla generická. Od toho záměru jsem upustil z důvodu zbytečné složitosti. Následně na to jsem postupně přidával kódy jednotlivých třídících algoritmů.

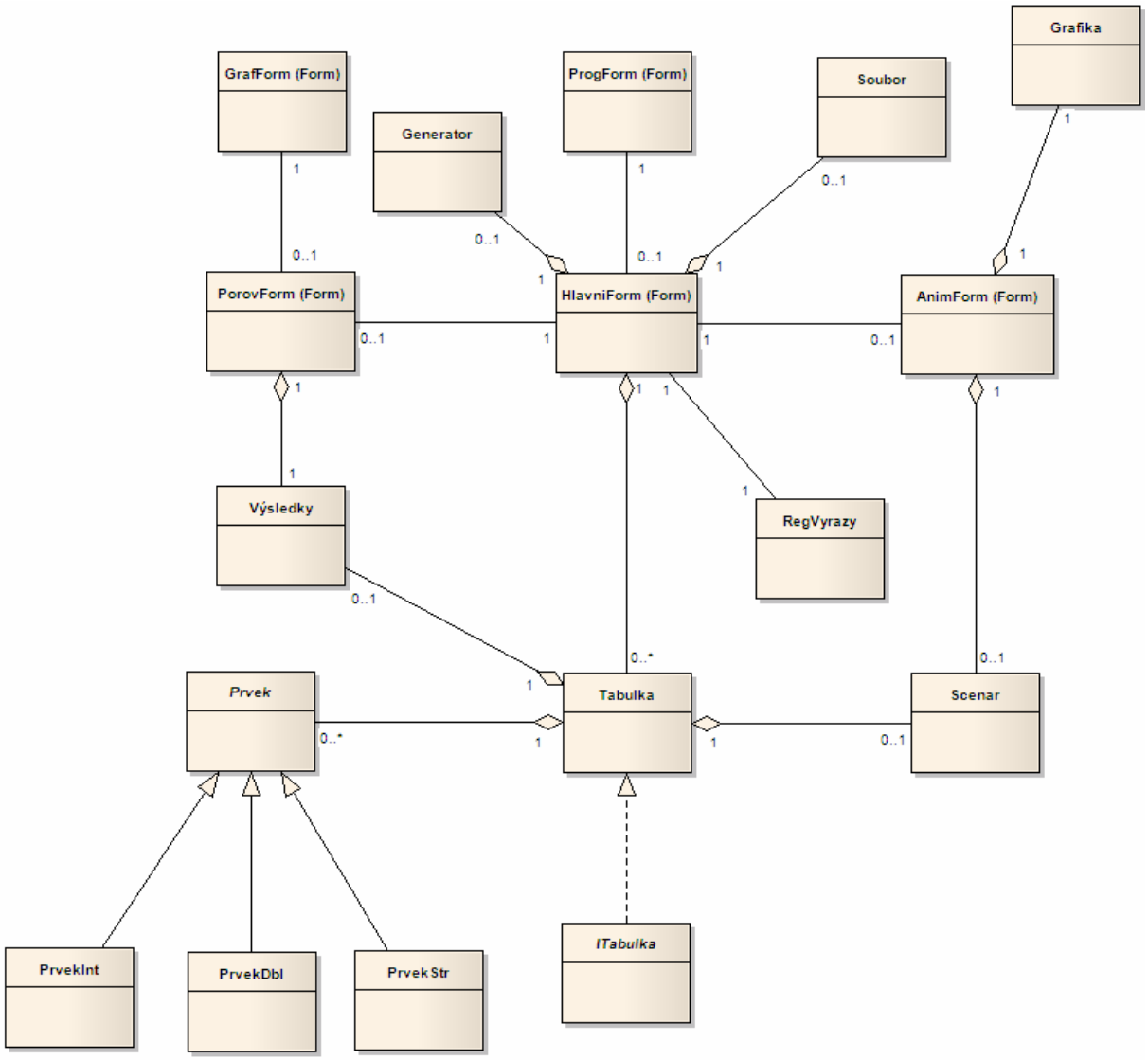
Po třídě `Tabulka` následovaly třídy pro generování prvků, nahrávání ze souboru a statická třída s regulárními výrazy, kterou jsem postupem času aktualizoval.

Implementované algoritmy jsem potřeboval odzkoušet, proto jsem se následně vytvořil hlavní formulář, ve kterém původně měla být i animace s neměnným počtem prvků. Dále jsem potom paralelně vytvářel kompletní animaci s nastavitelným počtem prvků a porovnání algoritmů. K animaci jsem potřeboval třídu `Scenar` na uchování scénáře a třídu `Vysledky`, která mi uchovávala výsledky měření třídění algoritmu. Původně jsem zamýšlel vykreslovat jednotlivé hodnoty prvků graficky přes třídu `Graphics`, ale po zjištění možnosti změny lokace u `Textboxu` (bude vysvětleno v další kapitole) jsem se rozhodl pro tuto jednodušší volbu. Animace mi i přes to zabrala největší část práce.

Při zkoušení aplikace jsem neviděl, co program právě provádí, proto jsem přidat další formulář s komponentou `Progressbar`, která mi ukazovala stav aplikace. Hodnoty prvků byly třeba zobrazit, editovat a popřípadě smazat, proto jsem vytvořil další formulář pro zobrazení hodnot prvků.

Nakonec jsem přidal poslední formulář k zobrazovaní grafu a snažil se celou aplikaci odladit. To se mi doufám alespoň z části povedlo.

6.3 Diagram tříd



Obrázek 28 - Class diagram[28]

6.4 Stručný popis tříd a rozhranní

Aplikace obsahuje celkem 13 tříd, z toho je jedna abstraktní, jedna statická a jedno rozhraní. Dále z 5-ti formulářů a statické třídy, která obsahuje `Main` metodu, ve které se program spouští.

[28] Zdroj: Vlastní

6.4.1 Třída Prvek

Základním kamenem aplikace je abstraktní třída `Prvek`, která má dvě metody: `PorovnejMensi` a `PorovnejVetsi` obě metody mají jeden argument, a to abs. třídu Prvek, tzn. Prvek porovnává sebe s dalším prvkem. Metody vrací datový typ `bool`. Původně tato třída obsahovala pouze metodu `Porovnej`, ale potřeboval jsem odlišit tři stavy (znaménka <,>,=) aby se rovnající prvky neprohazovaly.

6.4.2 Třídy PrvekInt, PrvekDbl a PrvekStr

Třídy `PrvekInt`, `PrvekDbl` a `PrvekStr` dědí obě metody z abstraktní třídy Prvek, které definují. Dále mají datový typ a vlastnost s názvem `Hodnota` typu `Integer`, `Double` a `String`, podle druhu prvku. Do datového typu `Hodnota` jsou ukládána data pro třídění.

6.4.3 Rozhraní ITabulka

Rozhraní `ITabulka` obsahuje základní metody tabulky jako jsou: `Vlož`, `Odeber`, `Najdi`, `VratPocetPrvku` a `JePrazdna`. Zde by se dala použít místo rozhraní abstraktní třída.

6.4.4 Třída Tabulka

Tato třída implementuje metody rozhraní `ITabulka`, které zde definuje. Dále má metody na editaci hodnoty prvku, prohození místa dvou prvků a dvě metody sloužící k porovnání hodnot prvku. Tabulka také obsahuje metody všech třídících algoritmů a jejich pomocné metody. Ve třídících algoritmech jsou implementované metody tříd `Vysledky` a `Scenar` na spuštění měřícího času, zastavení měřícího času, měření počtu prohození a počtu porovnání dvou prvků. Samotné prvky ukládá do .NET Frameworkové kolekce `Arraylist`. Třídu `Scenar` využívá ve funkci Grafická animace algoritmu pro uložení indexů prohazovaných prvků. Do třídy `Vysledky` zapisuje naměřené hodnoty třídění jako jsou čas strávený samotným tříděním, počet porovnání a počet prohození. Kromě základního konstruktoru má další dva konstruktory. Kopírovací konstruktor sloužící k rozkopírování tabulky při třídění na více algoritmech a kopírovací konstruktor s nastavením počtu prvků sloužící na rozdělení tabulky při zobrazení do grafu.

6.4.5 Třída Scenar

Tato třída slouží k uchování dat určených pro animaci třídění. Obsahuje strukturu `Polozka`, do které se ukládají indexy prohazovaných prvků, cela struktura se potom ukládá do `Arraylistu`. Dále do `ArrayListu` ukládá původní pozice prvků. Třída obsahuje kromě základních metod na obsluhu struktury `Polozka,` metodu vracející počet prohození a metodu kontrolující dvojice, která projde všechny dvojice a ty se stejným číslem potom vymaže. V praxi to má za následek to, že se neprohazují prvky se stejnou hodnotou. Instance této třídy se po skončení třídícího algoritmu předá Animačnímu formuláři.

6.4.6 Třída Vysledky

Tato třída obsahuje veškerá naměřená data, která je třeba zobrazit. Byla zde použita třída `Stopwatch` na přesné změření doby trvání třídění. Třída má metody potřebné pro vrácení času jak v milisekundách pro graf, tak ve formátu: "hh:mm:ss".

6.4.7 Třída Soubor

Třída `Soubor` slouží k načítání a ukládání hodnot pro prvky a také k uložení naměřených výsledků do souboru. Načítání a ukládání do souboru je řešeno pomocí `OpenFileDialogu` a `SaveFileDialogu`. Hodnoty lze načíst a uložit pouze do textového souboru. Jako oddělovač jednotlivých hodnot je zde použit „enter“ nový řádek.

6.4.8 Statická třída RegVyrazy

K určení datového typu potřebujeme nezbytně regulární výrazy. Všechny potřebné regulární výrazy potřebné pro celou aplikaci jsou uloženy v této třídě. Kromě základních reg. výrazů na určení `Stringu`, `Integeru` a `Doublu` jsou zde regulární výrazy na ošetření všech vstupních prvků.

6.4.9 Třída Grafika

Tato třída slouží ke grafické animaci prohození dvou prvků. Celá animace je v podstatě postavena na změně lokace `Textboxů` ve kterých jsou napsány hodnotu tříděných prvků. Je zde použita metoda `DoEvents` třídy `Application`, která v mém případě posloužila k lepšímu průběhu překreslování jednotlivých `Textboxů`. Rychlost překreslení pozice je řešena pomocí uspání vlákna.

6.4.10 Třída Generátor

Poslední ze zmiňovaných tříd slouží ke generování hodnot všech tří typů. Pro samotné generování je použita třída `Random`.

6.4.11 Formulář AnimForm

První z uvedených formulářů slouží k vlastní animaci třídění. K tomu využívá třídu `Grafika` a třídu `Scenar`. Jsou zde tlačítka Play a Pause k zpuštění a zastavení animace a tlačítka dopředu a dozadu určené k jednomu kroku prohození. Dále je zde obsažen `TrackBar` pro nastavení rychlosti. Do konstruktoru formuláře se vloží jako parametr počet prvků a vyplněný scénář a kompletní tabulka.

6.4.12 Formulář PorovForm

Tento formulář slouží k zobrazení naměřených hodnot zvolených třídících algoritmů. Obsahuje komponentu `DataGridView`, která se stará o samotné zobrazení a komponentu panel, ve které jsou dvě tlačítka. První slouží k uložení naměřených hodnot do souboru a druhé na zobrazení grafu třídění. Graf lze zobrazit pouze tehdy, je-li při prvotní inicializaci zvolena čtvrtá možnost a to „Porovnání s grafem“. Do prvního konstruktoru se vloží jako parametr `ArrayList` výsledků a počet zvolených algoritmů. Druhý konstruktor, který se volá při funkci aplikace „Porovnání s grafem“, má navíc jako parametr hranici (Co je hranice bude vysvětleno později).

6.4.13 Formulář GrafForm

Další z formulářů nám zprostředkovává vykreslení samotného grafu. Obsahuje komponentu Chart, kterou jsem si musel dodatečně doinstalovat. Kvůli této komponentě je potřeba .NET Framework 3.5 SP1, bez této verze frameworku vám aplikace při kliknutí na vykreslení grafu vypíše chybovou hlášku. Parametry konstruktoru formuláře jsou počet algoritmů, `ArrayList` výsledků a hranice. Graf se skládá z pěti úseček. Počáteční body těchto úseček jsou spočítány vydělením počtu prvků pěti. Např. Vloží-li do tabulky 2000 prvků máme hranici 400. Aplikace rozkopíruje tabulku do pěti dalších tabulek. První tabulka bude mít 400 prvků, druhá 800, třetí 1200, atd. Nad každou tabulkou se provede daný třídící algoritmus a hodnoty se vykreslí do grafu.

6.4.14 Formulář ZobrazForm

Pro zobrazení, editování a odstraňování prvků slouží formulář `ZobrazForm`. K samotnému zobrazení hodnot prvků využívá komponentu `DataGridView`. Dále formulář obsahuje komponentu `Panel`, ve které jsou tlačítka na smazaní prvku a na uložení tabulky do souboru. Konstruktor formuláře má jako parametr tabulku.

6.4.15 Formulář ProgForm

`ProgForm` je formulář s prvkem `ProgressBar`, sloužící ke grafickému zobrazení náročnější činnosti aplikace, aby byl uživatel informován o průběhu činnosti aplikace. Dále také vypisuje do popisku činnost, kterou právě provádí.

6.4.16 Formulář HlavniForm

Poslední a zároveň nejdůležitější formulář je výchozím formulářem celé aplikace. Jeho inicializace se provádí ve statické třídě `program.cs` v metodě `Main`. Formulář se skládá z `Menu` a pěti hlavních `GroupBox`ů, sloužících ke generování prvků, vkládání hodnoty prvku, výběru funkce aplikace, zvolení více algoritmů pomocí `Checkbox`ů a výstup naměřených výsledků. V menu obsahuje položky pro uložení a načtení dat do souboru, výběr jednoho algoritmu pro třídění, zobrazení hodnot tabulky, zobrazení formuláře výsledků a informace o třídících algoritmech.

7 POROVNÁNÍ ALGORITMŮ

7.1 Úvod

Než se pustím do samotného porovnávání algoritmů, musím si stanovit počet prvků a třídících algoritmů. Nejprve budu testovat s 50.000 prvky, zde zahrnu všechny třídící algoritmy, které mám naprogramované, jsou to: Selectionsort, Insertionsort, Bubblesort, Bubblesort s klesajícím krokem (Označen jako Bubblesort 2), Quicksort, Heapsort, Ripplesort, Shakersort, Mergesort a Shellsort. Původně jsem chtěl testovat se 100.000 prvky, nicméně test by trval poměrně dlouhou dobu. Hodnotu prvku budu generovat v rozsahu od 1 do 100.000. Poté budu testovat se 1.000.000 prvky. 1.000.000 prvků je na třídící algoritmy vycházející z Bubblesortu příliš, proto je z tohoto testu vypustím, stejně tak vypustím algoritmy Insertionsort a Selectionsort. To znamená, že setřídění 1.000.000 prvků budu provádět algoritmy Quicksort, Mergesort, Heapsort, Shellsort, Insertionsort a Selectionsort. Tyto testy budu provádět na tří odlišných sestavách, abychom zjistili, zdali má doba třídění v mé aplikaci vliv na výpočetní výkon, popřípadě jaký.

Druhý test bude prováděn na sestavě s nejvýkonnějšími parametry. Zde budu testovat dobu třídění u prvků s různými typy hodnot (`integer`, `string` a `double`) a použiji třídící algoritmy: Quicksort, Mergesort, Insertionsort a Bubblesort. Nechám vygenerovat opět 50.000 prvků. U datového typu string nejprve nastavím počet znaků na 20, poté na 6 a nakonec jen na 1, u integeru a doublu budu prvky generovat s hodnotami v rozsahu od 1 do 1.000.000 a od 1 do 10. Zde zjistíme, zdali se citelně změní doba třídění u prvků s různými typy hodnot.

V obou testech také budeme sledovat samotné porovnání jednotlivých algoritmů z hlediska doby setřídění. Tabulky jsou setříděny podle doby třídění. V tabulkách je také uvedeno i počet prohození a počet porovnání. Doba třídění je ve formátu „hh:mm:ss“. Do grafů jsem vybíral náhodné algoritmy.

Před samotným testem si nejprve ověříme rozptyl naměřených hodnot doby třídění. Vygeneruji si 10.000 prvků s hodnotami v rozsahu 1 až 100.000, zvolím si algoritmus Selectionsort a nechám tímto algoritmem utřídit tabulku. Tento test provedu desetkrát. Z tohoto testu zjistíme odchylku měření.

7.2 Testovací sestavy

Pro porovnávání třídících algoritmů jsem zvolil tři pro mě dostupné, ale odlišné konfigurace. První využívá pomalý procesor Atom od společnosti Intel, druhá výkonnější Pentium IV a třetí nejvýkonnější dvoujádrový AMD Athlon X2. Veškeré testy budou prováděny pod operační systémem Windows XP a .NET Frameworkem 3.5 SP1.

Podrobný popis sestav:

Procesor	Intel Atom N270, 1.60GHz, 512KB L2 cache
FSB	533MHz
Paměť	1024MB
OS	Windows XP

Tabulka 1 - První sestava s procesorem Intel Atom

Procesor	Intel Pentium IV, 2,4GHz, 1MB L2 cache
FSB	533MHz
Paměť	512MB RAM
OS	Windows XP

Tabulka 2 - Druhá sestava s procesorem Intel Pentium IV

Procesor	AMD Athlon X2 5000+, 2,6GHz, 2MB L2 cache
HT	2GHz
Paměť	2048MB RAM
OS	Windows XP

Tabulka 3 - Třetí sestava s procesorem AMD Athlon X2

7.3 Ověření rozptylu doby třídění

Tímto testem a následným výpočtem jsme zjistili průměrnou odchylku naměřených hodnot v sekundách a v procentech. Odchylka od průměru nám vyšla pod 1% což je přijatelný výsledek.

Selection sort	Doba třídění
1. Test	00:00:08.2981317
2. Test	00:00:08.1983928
3. Test	00:00:08.5108766
4. Test	00:00:08.2180805
5. Test	00:00:08.2032328
6. Test	00:00:08.2193206
7. Test	00:00:08.1973979
8. Test	00:00:08.2162140
9. Test	00:00:08.1931454
10. Test	00:00:08.2259496
Průměrná odchylka [s]	0.063
Odchylka od průměru [%]	0.76

Tabulka 4 - Test rozptylu naměřených hodnot

7.4 Naměřené výsledky prvního testu

První test dopadl dle očekávání. Ukázalo se, že na každé sestavě proběhne test za jinou dobu. Nejlépe si vedla třetí (nejvýkonnější) sestava, na druhém místě skončila druhá sestava a na poslední místě s velkým odstupem skončila třetí sestava v podobě malého 8,9” notebook Acer Aspire One. Výkonové rozdíly byly docela znatelné, zejména porovnání první a třetí sestavy. Následující tabulky nám ukáží podrobné srovnání jednotlivých algoritmů třídících 50.000 prvků.

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:00.3339628	198.039	607.801
Heapsort	00:00:00.4635114	736.951	704.993
Mergesort	00:00:00.5537692	1.568.928	718.385
Shellsort	00:00:00.6395634	642.916	642.916
Shakersort	00:06:52.3740676	622.665.140	934.229.115
Selectionsort	00:07:38.6578981	49.987	1.250.024.999
Insertionsort	00:10:10.8048106	622.665.140	1.249.975.000
Bubblesort 2	00:11:00.3740291	622.665.140	1.249.975.000
Ripplesort	00:13:46.7976787	622.665.140	2.491.050.178
Bubblesort	00:16:12.7679277	622.665.140	2.499.900.001

Tabulka 5 - Výsledky naměřené první sestavou.

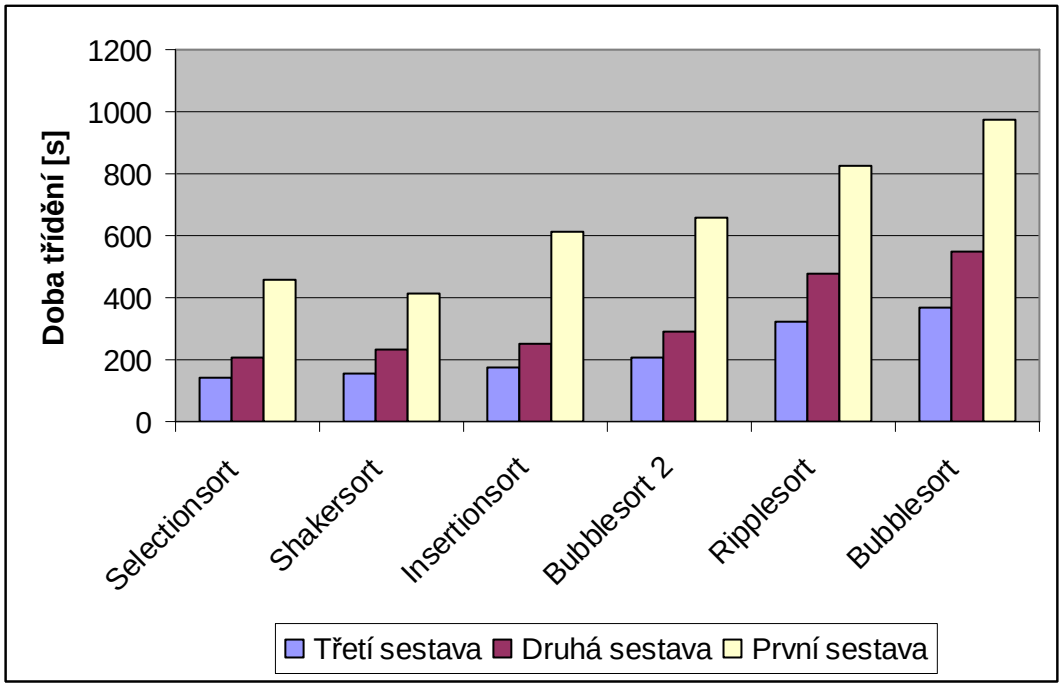
Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:00.1488300	198.039	607.801
Mergesort	00:00:00.2400388	1.568.928	718.385
Shellsort	00:00:00.2564658	642.916	642.916
Heapsort	00:00:00.2700427	736.951	704.993
Selectionsort	00:03:24.8206696	49.988	1.250.024.999
Shakersort	00:03:55.0920379	622.665.140	934.229.115
Insertionsort	00:04:09.0068436	622.665.140	1.249.975.000
Bubblesort 2	00:04:52.0032168	622.665.140	1.249.975.000
Ripplesort	00:07:57.1497969	622.665.140	2.491.050.178
Bubblesort	00:09:05.8342331	622.665.140	2.499.900.001

Tabulka 6 - Výsledky naměřené druhou sestavou.

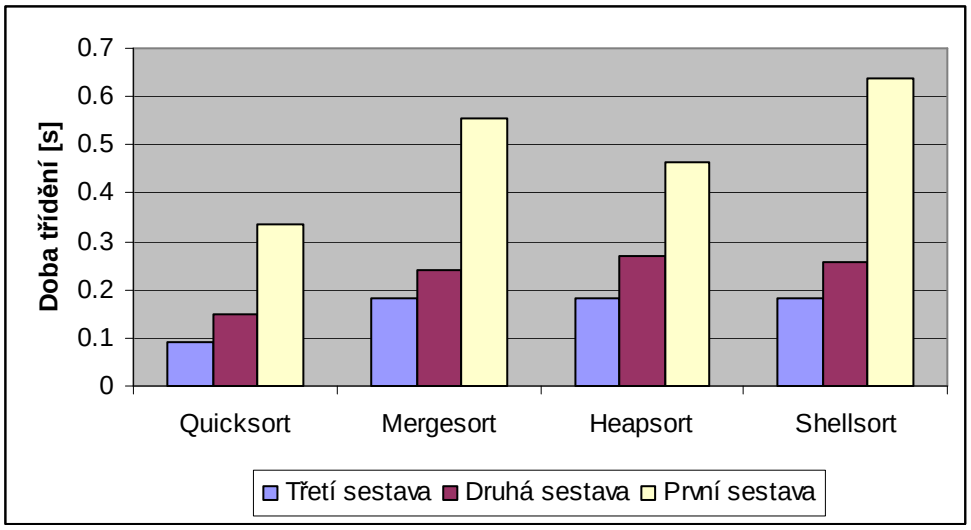
Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:00.0919537	198.039	607.801
Mergesort	00:00:00.1802775	1.568.928	718.385
Heapsort	00:00:00.1817074	736.951	704.993
Shellsort	00:00:00.1828540	642.916	642.916
Selectionsort	00:02:19.7390258	49.986	1.250.024.999
Shakersort	00:02:34.2389403	622.665.140	934.229.115
Insertionsort	00:02:56.1538297	622.665.140	1.249.975.000
Bubblesort 2	00:03:23.6985320	622.665.140	1.249.975.000
Ripplesort	00:05:21.7370538	622.665.140	2.491.050.178
Bubblesort	00:06:08.8684411	622.665.140	2.499.900.001

Tabulka 7 - Výsledky naměřené třetí sestavou

Na první pohled je zřejmé, že třídící algoritmy můžeme rozdělit podle doby třídění do dvou skupin. Proto jsem s 1.000.000 prvky druhou (pomalejší) skupinu vyřadil. Následující graf nám přehledně ukazuje rozdíly jednak mezi sestavami, tak i mezi jednotlivými třídícími algoritmy. Původně jsem chtěl do grafu zahrnout kompletně všechny třídící algoritmy, ale z důvodu velkého rozptylu doby třídění jsem je zahrnul do grafu následujícího.



Obrázek 29 - Zobrazení pomalejší skupiny třídících algoritmů do grafu.



Obrázek 30 - Zobrazení rychlejší skupiny třídících algoritmů do grafu.

Následující tabulky nám ukáží podrobné srovnání jednotlivých algoritmů třídících 1.000.000 prvků.

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:06.7214640	5.400.633	14.243.710
Mergesort	00:00:09.8441698	39.902.848	18.674.205
Shellsort	00:00:14.2418715	15.171.026	15.171.026
Heapsort	00:00:14.4695828	19.047.539	18.396.657

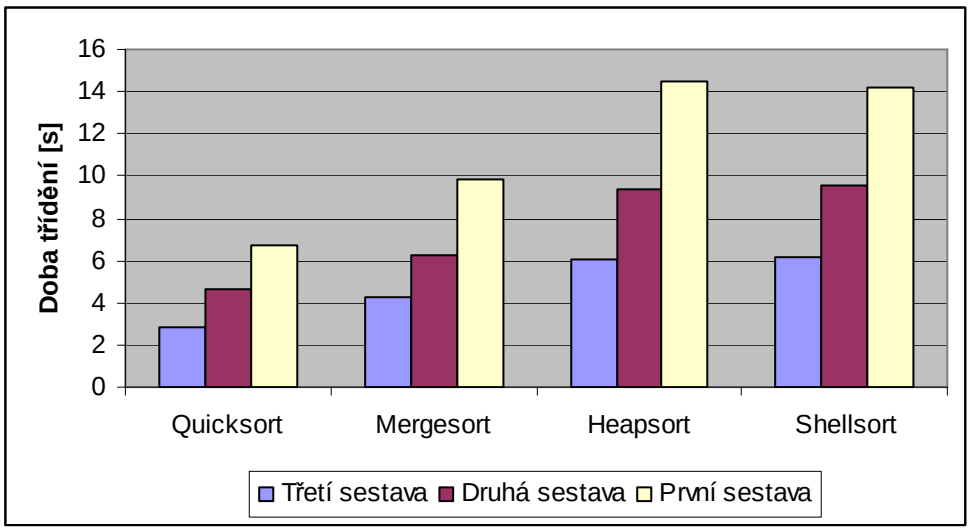
Tabulka 8 - Výsledky naměřené první sestavou.

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:04.6077931	5.400.633	14.243.710
Mergesort	00:00:06.2700893	39.902.848	18.674.205
Heapsort	00:00:09.3755337	19.047.539	18.396.657
Shellsort	00:00:09.6048916	15.171.026	15.171.026

Tabulka 9 - Výsledky naměřené druhou sestavou.

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání
Quicksort	00:00:02.8132819	5.400.633	14.243.710
Mergesort	00:00:04.2493329	39.902.848	18.674.205
Heapsort	00:00:06.0420436	19.047.539	18.396.657
Shellsort	00:00:06.1222211	15.171.026	15.171.026

Tabulka 10 - Výsledky naměřené třetí sestavou.



Obrázek 31 - Zobrazení rychlejší skupiny algoritmů třídících 1.000.000 prvků

Zde je patrné, jak veliké jsou rozdíly mezi těmito čtyřmi algoritmy a zbývajícími šesti algoritmy. Shrneme-li celý test, můžeme s jednoznačností říct, že nejrychleji má setříděné prvky algoritmus Quicksort, po něm na druhém místě algoritmus Mergesort a o třetí místo bojují algoritmy Heapsort a Shellsort, jejich časy jsou při testování 1.000.000 prvků téměř shodné. Podíváme-li se na ostatní parametry, zjistíme, že třídící algoritmy Bubblesort,

Insertionsort, Ripplesort, Shakersort a Bubblesort s klesajícím krokem mají stejný počet prohození, tj. dáno shodností struktury algoritmu. Počet porovnání pak má nejméně Quicksort.

7.5 Naměřené výsledky druhého testu

Druhý test dopadl také dle očekávání. Nejdelší dobu třídění měla tabulka s prvky o datovém typu `string`. To je způsobeno prvkem s datovým typem `string`, který využívá v metodách `PorovnejVetsi` a `PorovnejMensi` statickou třídu `String` a její metodu `CompareOrdinal`. Nejkratší zase tabulka s prvky o datovém typu integer. Zajímavě dopadl třídící algoritmus Mergesort, který měl rychleji setříděné prvky s hodnotou typu integer a double v rozsahu od 1 do 1000000 než od 1 do 10.

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání	Počet znaků / rozsah čísla	Datový typ
Quicksort	00:00:00.0869008	331.360	171.393	1 - 10	Integer
Quicksort	00:00:00.0904676	198.381	601.048	1 - 1.000.000	Integer
Quicksort	00:00:00.0953200	314.810	187.428	1	String
Quicksort	00:00:00.1046597	196.114	641.854	1 - 10	Double
Quicksort	00:00:00.1068002	196.173	663.003	1 - 1.000.000	Double
Quicksort	00:00:00.1190603	196.320	636.592	6	String
Quicksort	00:00:00.1354177	194.861	648.463	20	String

Tabulka 11 - Naměřené výsledky různých datových typů algoritmem Quicksort

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání	Počet znaků / rozsah čísla	Datový typ
Mergesort	00:00:00.1292882	1.568.928	718.078	1 - 1.000.000	Integer
Mergesort	00:00:00.1369241	1.568.928	688.786	1 - 10	Integer
Mergesort	00:00:00.1377548	1.568.928	718.223	1 - 1.000.000	Double
Mergesort	00:00:00.1496522	1.568.928	718.128	1	String
Mergesort	00:00:00.1529602	1.568.928	717.999	1 - 10	Double
Mergesort	00:00:00.1604942	1.568.928	709.043	6	String
Mergesort	00:00:00.2103509	1.568.928	718.474	20	String

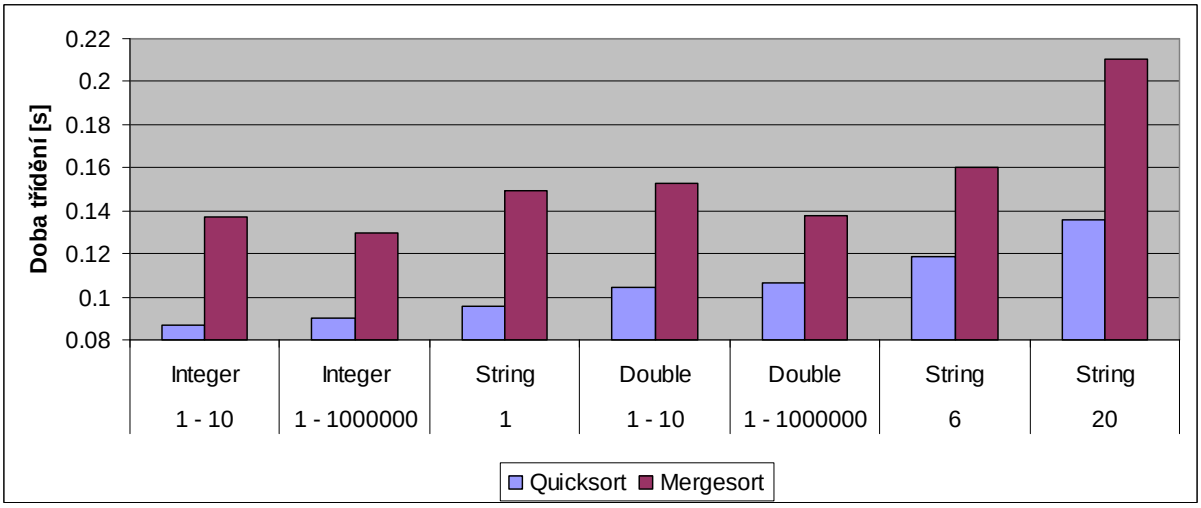
Tabulka 12 - Naměřené výsledky různých datových typů algoritmem Mergesort

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání	Počet znaků / rozsah čísla	Datový typ
Insertionsort	00:02:45.2280200	553.800.009	1.249.975.000	1 - 10	Integer
Insertionsort	00:03:02.8305520	623.362.478	1.249.975.000	1 - 1.000.000	Integer
Insertionsort	00:03:04.8393752	600.859.846	1.249.975.000	1	String
Insertionsort	00:03:09.2974909	624.096.721	1.249.975.000	1 - 10	Double
Insertionsort	00:03:15.4545396	624.554.734	1.249.975.000	1 - 1.000.000	Double
Insertionsort	00:04:11.9197723	627.821.964	1.249.975.000	6	String
Insertionsort	00:04:49.6242958	623.420.336	1.249.975.000	20	String

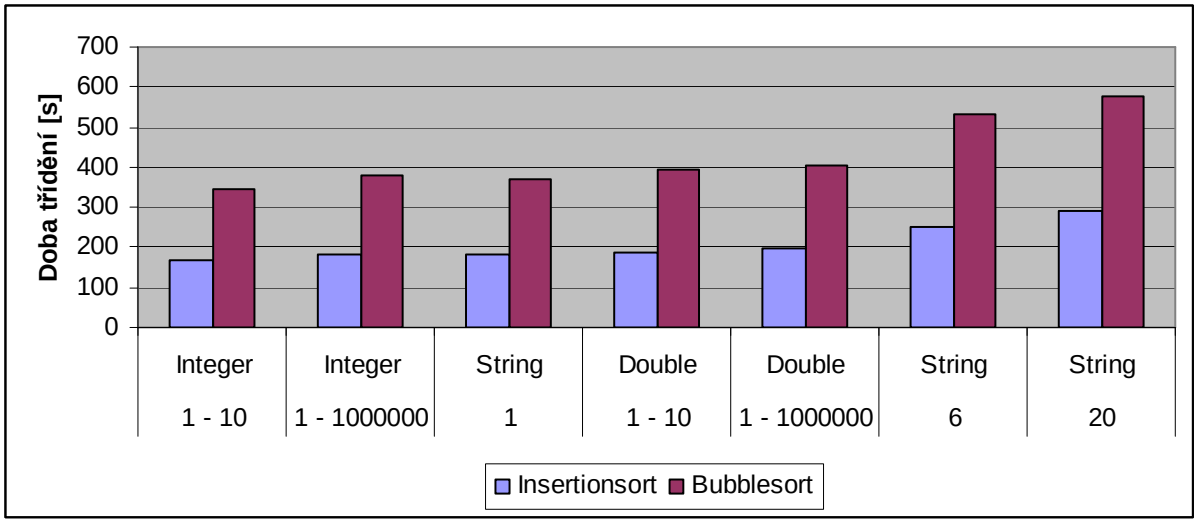
Tabulka 13 - Naměřené výsledky různých datových typů algoritmem Insertionsort

Název algoritmu	Doba třídění	Počet prohození	Počet porovnání	Počet znaků / rozsah čísla	Datový typ
Bubblesort	00:05:43.6153779	553.800.009	2.499.900.001	1 - 10	Integer
Bubblesort	00:06:11.6816971	600.859.846	2.499.900.001	1	String
Bubblesort	00:06:19.4999957	623.362.478	2.499.900.001	1 - 1.000.000	Integer
Bubblesort	00:06:36.1885935	624.096.721	2.499.900.001	1 - 10	Double
Bubblesort	00:06:42.0712011	624.554.734	2.499.900.001	1 - 1.000.000	Double
Bubblesort	00:08:51.4701767	627.821.964	2.499.900.001	6	String
Bubblesort	00:09:34.6324502	623.420.336	2.499.900.001	20	String

Tabulka 14 - Naměřené výsledky různých datových typů algoritmem Bubblesort



Obrázek 32 - Graf různých datových typů algoritmů Quicksort a Mergesort



Obrázek 33 - Graf různých datových typů algoritmů Insertionsort a Bubblesort

Z grafů je patrná shodnost doby třídění u algoritmů Insertionsort a Bubblesort u prvků s datovým typem integer a double. Dále je potom jasný odstup prvků s datovým typem string. To je způsobeno složitějším porovnávacím algoritmem.

8 ZÁVĚR

Cílem této práce bylo seznámit čtenáře s třídícími algoritmy po teoretické, ale i po praktické stránce. V teoretické části se podařilo provést ucelený přehled základních třídících algoritmů. Jednotlivé algoritmy jsou vždy popsány, uvedeny jejich vlastnosti a implementace v jazyce C# .Nebyly opomenuty ani základní pojmy z datových struktur a popis abstraktního datového typu tabulka.

Cílem praktické části bylo vytvořit aplikaci na vizualizaci a porovnání algoritmů. Aplikace byla naprogramována pomocí jazyka C#, frameworku .NET Framework 3.5 a vývojového nástroje Visual Studio 2008, byly zde použity komponenty DataGridView a Chart, která je součástí .NET frameworku 3.5 SP1. S tímto jazykem jsem se seznamoval od začátku, včetně používání Visual Studia 2008 a tříd .NET Frameworku. Design aplikace byl navržen a vytvořen s ohledem na přehlednost a přístupnost, tudíž je možné využívat animaci třídění jako pomůcku při studiu principu třídících algoritmů. Podle grafu, který je součástí aplikace, lze pouhým okem porovnat podle doby třídění vybrané algoritmy.

V budoucnu by se mohla ještě vylepšit animace o zabarvení utříděných, respektive neutříděných Textboxů, vyznačení Pivota u třídícího algoritmu Quicksort. Dále by mohla obsahovat další třídící algoritmy jako jsou Radixsort, Bucketsort a Countingsort. A v poslední řadě možnost načítání dat ze souborů csv.

Hlavní přínosy práce:

- Animace třídících algoritmů pro podporu výuky předmětu Datové Struktury.
- Možnost setřídění vlastní množiny dat.
- Porovnání třídících algoritmů dle vybraného kritéria se zobrazením do grafu.

Pro mě jako autora práce bylo toto téma zajímavé. Dozvěděl jsem se spoustu nových věcí, zvláště o vynikajícím jazyce C# a .NET Frameworku, v jehož učení a programování bych chtěl v budoucnu dál pokračovat.

9 LITERATURA

9.1 Seznam použité literatury

[1] WIRTH, Niklaus. *Algoritmy a Struktury údajov*. Alfa, Bratislava, 1989, ISBN 80-05-00153-3

[2] BERAN, Radek. *Algoritmy* [online]. Dostupný z www: <http://www.beranr.webzdarma.cz/algoritmy/algoritmy.html>

[3] PACLT, Radek. *Porovnání a prezentace algoritmů třídění tabulek*. Univerzita Pardubice, Fakulta elektrotechniky a informatiky, Bakalářská práce. 2004, 79 s

[4] SHARP, John. *Microsoft Visual C# 2008*. Brno : Computer Press, 2008. 592 s. ISBN 978-80-251-2027-9.

[5] SELLS, Chris. *C# a WinForms*. Brno : Zoner Press, 2005. 648 s. ISBN 80-86815-25-0.

[6] WRÓBLEWSKI, Piotr. *Algoritmy, datové struktury a programovací techniky*. Brno : Computer Press, 2004, 352 s. ISBN: 80-251-0343-9.

[7] KAVICKA, Antonín. *Datové struktury*. Elektronické sylaby přednášek předmětu Datové struktury. 2007.

[8] *Algoritmy.net* [online]. Dostupný z www: <http://algoritmy.net/index.html>

[9] LEWIS, H. R., DENENBERG, L.: *Data structures and their algorithms*. Berkley, Adison-Wesley, 1997.¨

10 SEZNAM PŘÍLOH

- CD obsahující zdrojové kódy aplikace.