VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

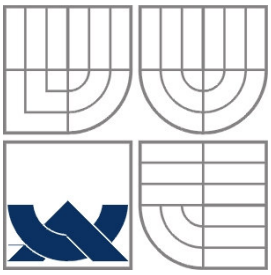
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DEMOSTRAČNÍ PROGRAM METOD ŘAZENÍ V JAZYCE C

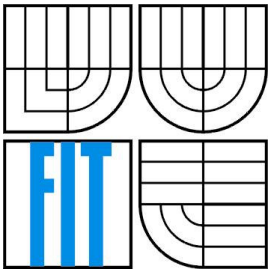
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Lukáš Pernica
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DEMOSTRAČNÍ PROGRAM METOD ŘAZENÍ V JAZYCE C

Demonstration Program of Sorting Methods in C Language

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Lukáš Pernica
AUTHOR

VEDOUCÍ PRÁCE — Prof. Ing. Jan Maxmilián Honzík, CSc.
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Pernica Lukáš**
Obor: Informační technologie
Téma: **Demonstrační program metod řazení v jazyce C**
Kategorie: Alg. a datové struktury

Pokyny:

1. Seznamte se detailně textem kapitoly 5 (řazení) ve studijní opoře pro předmět IAL
2. Seznamte se s chováním dodaného (exe) programu pro demonstraci vybraných metod řazení s grafickou i zvukovou reprezentací jejich funkce
3. Vytvořte v jazyce C demonstrační program s identickým chováním, rozšířený o metody dle zadání vedoucího BP.
4. Porovnejte výsledky dosažené v programu zapsaném v jazyce C s výsledky původního programu zapsaného v jazyce Basic.
5. Navrhněte příklady vhodné pro formulářově orientovanou písemnou zkoušku ze znalostí tohoto okruhu.

Literatura:

- Funkční demonstrační program (dodá vedoucí BP)
- Honzík,J.M.: Algoritmy. Studijní opora pro předmět Algoritmy, Elektronický text, FIT VUT v Brně

Při obhajobě semestrální části projektu je požadováno:

1. Funkční demonstrační program pro 4 zvolené metody.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Honzík Jan M., prof. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Lukáš Pernica**
Id studenta: 84203
Bytem: Kanice 158, 664 01 Bílovice nad Svitavou
Narozen: 24. 01. 1985, Brno
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Demonstrační program metod řazení v jazyce C
Vedoucí/školitel VŠKP: Honzík Jan M., prof. Ing., CSc.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnožením.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☒ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

V oboru informačních technologií je jednou ze základních dovedností každého programátora zvládnutí problematiky řadicích algoritmů. Řadicí algoritmy jsou využívány ve velmi širokém rozmezí a i když se v každém programovacím jazyce zapisují odlišně, jejich princip zůstává stejný.

V této práci se budu zabývat problematikou řadicích algoritmů a popisem programu, který je součástí této bakalářské práce. Program má jednoduchou formou demonstrovat principy nejpoužívanějších řadicích algoritmů a slouží tak jako pomůcka pro snazší pochopení metod řazení.

Klíčová slova

Řadicí algoritmus, demonstrační program, Binary Insert sort, Bubble sort, Select sort, Heap sort, Shell sort, Quick sort, Merge sort, rozděl a panuj, WIN API, grafické prostředí

Abstract

The one of the basic skill of every computer programer in the subject of Information technology is to manage the problems of the sorting algorithm. Sorting algorithms are used in the very wide range and though they are writed differently in a different programming language, their principal remains same.

At this work I'm going to deal with the problems of sorting algorithm and description of the program, which is part of this bachelor's thesis. The program have to demonstrate by the simple way the principles of the most used sorting algorithms and give us the clue to easier understanding of the sorting methods.

Keywords

Sorting algorithm, demonstrational program, Binary Insert sort, Bubble sort, Select sort, Heap sort, Shell sort, Quick sort, Merge sort, divide and conquer, WIN API, graphics environment

Citace

Pernica, Lukáš: Demonstrační program metod řazení v jazyce C, bakalářská práce, Brno, FIT VUT v Brně, 2007

Demonstrační program metod řazení v jazyce C

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Jana Maxmiliána Honzíka, CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Pernica
2.5.2007

Poděkování

Děkuji tímto vedoucímu své bakalářské práce, panu Prof. Ing. Janu Maxmiliánu Honzíkovi, CSc., nejenom za vedení a cenné rady, ale také za jeho připomínky a podnětné návrhy na různá rozšíření a zlepšení jak v průběhu návrhu a psaní programu, tak i při tvorbě tohoto textu.

Obsah

Úvod

Řadicí algoritmy jsou nedílnou významnou součástí tvorby algoritmů a programování již od samotných jejich počátků. To se nezměnilo ani s postupem času v procedurálním a funkcionálním programování až do dnešních dob, kdy se klade velký důraz na objektově orientované programování. Jediné co se mění je zápis jednotlivých algoritmů v jednotlivých programovacích jazycích. Princip zůstává pořád stejný. Řadicí algoritmy nalézáme i v oblastech, kde by se jejich použití mohlo zdát méně zřejmé, jako například při tvorbě webových stránek apod.

Funkcí řadicích algoritmů, jak již vyplývá z názvu, je seřazení dané posloupnosti prvků, nejčastěji do rostoucí či klesající posloupnosti. Nejjednodušší postupy řazení používají lidé stále pro vytvoření uspořádání nejrůznějších dokumentů či věcí, kde výsledkem je větší přehlednost, orientace, nebo jen estetický dojem. Pro řazení malého počtu prvků není potřeba znát žádné složité postupy. Někdy není zapotřebí vůbec žádný předem daný postup, protože na malém počtu prvků postupujeme intuitivně. S rozvojem počítačů bylo zapotřebí postupů řazení, které by byly vhodné pro zpracování strojem, tedy přesně daný postup, jak se má daná skupina prvků seřadit. Pro relativně malý počet prvků se dá použít jakýkoliv řadicí algoritmus, který má velkou časovou náročnost. Ve většině případů se však jedná o kvanta, kde záleží na časové složitosti daného algoritmu. Proto jsou vyvíjeny algoritmy, které jsou velice rychlé. Řadicí algoritmy se používají v široké programátorské oblasti, a proto by měl každý programátor znát princip alespoň těch nejjednodušších metod.

Problematikou algoritmů se zabývá mnoho knih a publikací, kde jsou vysvětlovány i kapitoly s řazením. Jako pomůcky k pochopení mohou sloužit i různé grafy, obrázky a také demonstrační programy. Jedním takovým demonstračním programem řadicích algoritmů se zabývá i tato práce.

Cílem této práce je vytvoření výukového programu, který na jednoduchých a názorných příkladech ukazuje funkci řadicích algoritmů při jejich práci. Program vzniká podle předlohy starého programu napsaného ještě v jazyce Basic, ke kterému se nedochovala již žádná dokumentace ani zdrojové kódy. Nově vyvíjený program, kterým se zabývá tato práce, je prioritně určen, jako učební pomůcka do předmětu IAL – „Algoritmy“. Předmět „Algoritmy“ je vyučován ve druhém ročníku bakalářského studia na fakultě informačních technologií VUT

v Brně. Protože výuka se ubírá směrem, kdy se jako základní jazyk vyučuje C, je i tento program psán v jazyce C, aby studentům poskytl nejen svou demonstrační grafickou část, ale i praktické ukázky jednotlivých úseků kódů, jak jsou zapsány ve zdrojovém kódu programu. Další zajímavou (a dalo by se říci netradiční) funkcí je zvuková kulisa. Je založena na přehrávání různě vysokých tónů při průběhu řazení demonstračního pole tímto programem. Pozorovatel tedy dostává nejen vizuální, ale i zvukový pohled na princip řadicí metody.

Dalším důvodem vzniku programu je také zmodernizování některých částí, které byly dříve, ve „vzorovém" programu napsaném v jazyce Basic, nevhodně zvoleny. Dále se jedná o doplnění dalších řadicích algoritmů, které nebyly implementovány, ale v předmětu „Algoritmy" jsou vyučovány.

Program se zabývá demonstrací principu řazení, ale nebere v úvahu přesnou časovou složitost jednotlivých algoritmů. Je to jednak způsobeno časovými zpožděními, které jsou vkládány z důvodu možnosti zpomalování a zrychlování metody. Změna rychlosti řadicí metody je vložena proto, aby bylo možné pozorovat činnost přeskupování prvků v poli. Dalším důvodem nepřesného měření časové složitosti je, že demonstraci je vhodné ukazovat na malém množství prvků.

Pro zjišťování časových složitostí algoritmů je vyvíjeno také velké množství programů. Ty pracují s velkými počty prvků seřazovanými v poli, kde se pohybujeme řádově ve stech tisících a milionech prvků. Na velkém množství jsou totiž dobře znát výhody složitějších algoritmů a také se ztratí časové zpoždění přípravných a závěrečných části některých algoritmů, kdy se musí řazené prvky upravovat. Dalším faktem je přesnější měření, kdy se na delším časovém úseku redukuje chyba měření.

Tato práce se dále zabývá problematikou řadicích algoritmů. V druhé kapitole této práce je obecné přiblížení pojmů algoritmů a řazení. Další kapitola popisuje princip jednotlivých řadicích algoritmů, jejich výhody a nevýhody. Ve čtvrté kapitole je návrh programu. První část čtvrté kapitoly je o studiu předloženého programu a druhá část se zabývá návrhem nového programu. V kapitole číslo 5 je popsána kompletní realizace programu. Jsou zde popsány jak jsou implementovány řadicí algoritmy a popis funkčnosti celého programu. V kapitole 6 jsou popsána rozšíření, která jsem navíc implementoval a která v původním programu napsaném v jazyce Basic nebyla. Následující kapitola Popisuje možnosti ovládání programu. V kapitole 8 je srovnání dosažených výsledků nového programu v C v porovnání s programem v jazyce Basic. Poslední kapitolou je zhodnocení a závěr, kde jsou shrnuty poznatky a výsledky této práce.

1 Algoritmy

Již v úvodu jsem několikrát zmínil pojem „Řadicí algoritmus“. Bylo by tedy vhodné alespoň krátce zmínit, co to algoritmy jsou.

1.1 Co je to algoritmus

Definice pojmu je převzatá viz [1].

„Algoritmus je konečná posloupnost/uspořádání postupů aplikovaných na konečný počet dat, jež dovoluje řešit přibližně stejné třídy problémů.“

Algoritmy se nepoužívají pouze v oblasti informačních technologií, ale obecně lze napsat algoritmus na jakoukoliv činnost v našem běžném životě. Může tak popsat např. kácení stromu v lese, kde nám algoritmus dává přesný popis, jak by se mělo postupovat. Lidé často používají různé algoritmy, aniž by si to sami uvědomovali, když vykonávají činnost podle nějakých přesných návodů.

Při programování jsou algoritmy hlavním stavebním prvkem kódu a daly by se charakterizovat v následujících bodech:

- mají specifikovaný vstup
- mají specifikovaný výstup
- mají jednoznačnou strukturu
- jsou konečné
- mají přesně daný sled kroků (stavů), ve kterých pracují

1.2 Řadicí algoritmy – obecně

Řadicí algoritmy jsou významnou částí programování a jejich znalost je velice důležitá. Snad v každém větším programu se vyskytuje nějaký řadicí algoritmus, ať už jednoduchý s vysokou časovou náročností, tak i složitější, ale vyznačující se svou rychlostí.

Řazení můžeme rozdělit do různých kategorií podle toho, jak pracují s prvky, které se řadí.

1 metoda vložení, kdy se vybere prvek a zařadí se na své správné místo v posloupnosti. Zástupcem může být např. „insert sort“

2 metoda výběru, kdy se hledá největší (nejmenší) prvek a zařadí se nakonec seřazené posloupnosti. Zástupcem může být např. „select sort“

3 metoda rozdělování, kdy se pole rozdělí na dva celky a v jedné části jsou pak prvky menší a ve druhé prvky větší. Zástupcem může být např. „quick sort“

1.3 Řazení versus třídění

V literatuře se uvádí pro stejné operace dvou pojmenování - třídění a řazení. Je nutné mezi těmito pojmy udělat rozdíl. Třídění je rozdělování položek či prvků podle jejich obecné vlastnosti – podle jejich třídy. Můžeme například třídit ovoce podle druhu nebo kuličky podle barvy. Pokud však chceme něčemu dát posloupnost, aby na jedné straně byl nejmenší prvek a na straně druhé největší, jedná se o řazení. Seřadíme např. žáky podle abecedy nebo různé předměty podle velikosti. Řazení podle velikosti je velice časté, právě v oblasti programování a na jednoduchosti číselného pole, jsou vysvětlovány i jednotlivé řadicí metody. Stejně bude v následujících kapitolách vysvětleno řazení i v této práci, tedy na poli čísel. Pojem třídění se používá jako překlad anglického slova sort. Vznik tohoto označení se váže do historie a popisuje ho prof. Ing. Honzík ve skriptech [2].

„Termín "třídění" v oblasti zpracování údajů vznikl v předpočítačové éře, kdy se mechanizované řazení na děrných štítcích provádělo postupným tříděním na mechanických třídicích strojích. Tyto stroje roztřídily (rozdělily) soubor štítků na 10 podsouborů podle hodnoty 0-9 vyděrované v zadaném sloupci. Operátor seřadil ručně tyto podsoubory za sebe do jednoho souboru a třídil je podle dalšího sloupce. Třídil-li se takto soubor štítků postupně podle sloupce 10,9,8,7,6 byl nakonec soubor seřazen podle hodnot vyděrovaných ve sloupcích 6-10.

Tento princip zachovává metoda "radix-sort" ("řazení tříděním"). Algoritmy používané později pro řazení na počítačích nevyužívaly princip třídění (sorting), ale pojem "sort" - "třídění" jim v názvu zůstal. V anglických názvech se kmen "sort" zachovává a v anglické terminologii se s pojmem "sorting" pro řazení budeme setkávat. V české terminologii se přidržíme termínu "řazení". Ostatně, ani v tělocviku neříkáme "setřiďte se podle velikosti.." a pokud něco třídíme, tak např. spíše ovoce podle druhu nebo velikosti nebo auta podle barvy. Řazení pro nás zůstane zvláštním případem třídění.“

2 Řadicí algoritmy

2.1 Binary Insert sort

Algoritmus pracuje na principu vkládání prvků na jejich místo v seřazené posloupnosti. Na začátku algoritmu je např. vlevo první prvek pole, který patří do seřazené části a zbytek prvků směrem doprava patří do části neseřazené. Vezme se první prvek neseřazené posloupnosti a hledá se místo v seřazené posloupnosti, kam prvek patří. Ke hledání správného místa v seřazené posloupnosti využívá algoritmus metody binárního vyhledávání.

Jakmile se najde místo, kam prvek patří, je postupně celá část seřazené posloupnosti od nalezeného místa posunuta o jednu pozici vpravo. Uvolní se tak místo pro prvek, který zařazujeme do seřazené posloupnosti. Po začlenění prvku do seřazené posloupnosti je délka seřazené posloupnosti zvýšena o 1 a délka neseřazené posloupnosti snížena o 1.

Algoritmus Binary Insert sort

Opakuj celý postup, dokud není velikost neseřazené posloupnosti 0.
Do „temp“ dej první prvek neseřazené posloupnosti.
Binárním vyhledáním najdi v seřazené posloupnosti místo pro prvek „temp“.
Posuň všechny prvky seřazené posloupnosti od nalezeného místa o jednu pozici vpravo.
Na uvolněné místo vlož prvek „temp“.
Zvětši velikost seřazené posloupnosti o 1 a neseřazené sniž o 1.

Obr. 2.1

Tento algoritmus je jeden ze základních a nejjednodušších a jeho časová složitost je kvadratická.

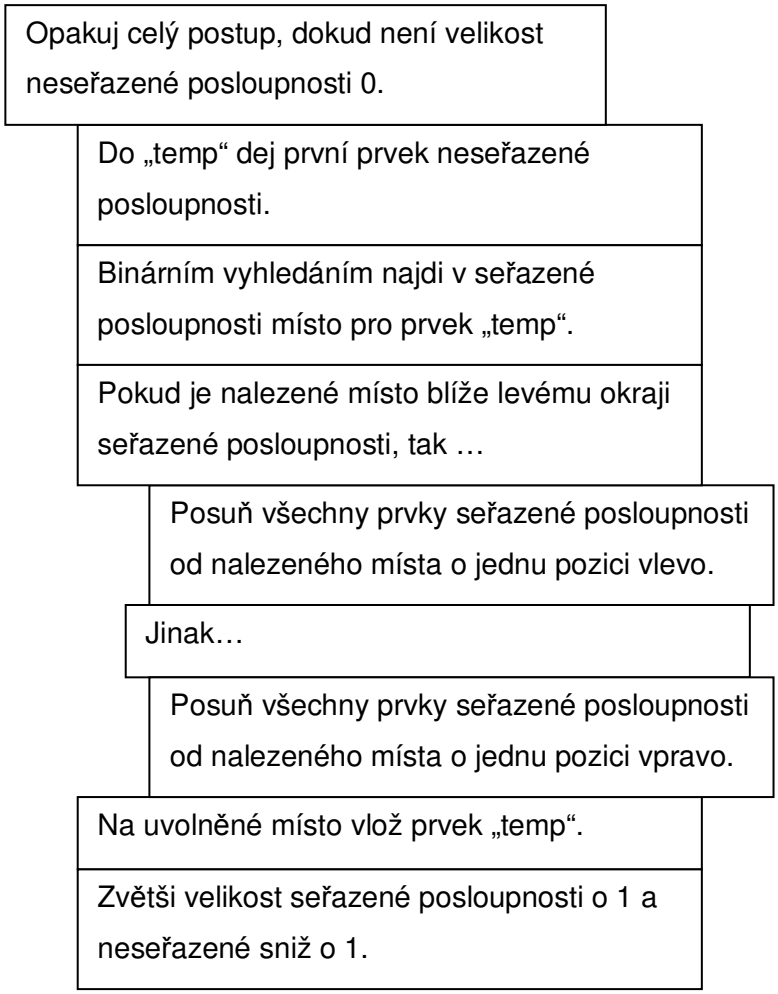
2.2 Binary Insert sort – modifikovaný

Modifikovaná verze binárního Insert sortu spočívá v tom, že se pracuje s dvojnásobně velkým polem. Zdrojové pole se umístí do pravé části dvojnásobného pole.

Nyní je postup stejný jako u základní verze. Vezme se první prvek z neseřazené posloupnosti a pomocí binárního vyhledávání se pro něj najde jeho správné místo v seřazené posloupnosti.

V této fázi nastává změna. Zjistí se, zda je nalezená pozice, kam se má prvek začlenit, blíže pravému nebo levému konci seřazené posloupnosti. Pokud je blíže levému konci, posunou se prvky seřazené posloupnosti od nalezené pozice směrem vlevo. Pokud je nalezené místo blíže pravému kraji seř. posloupnosti, posunou se prvky od nalezené pozice směrem vpravo. Velikost seřazené posloupnosti se poté zvýší o 1 a neseřazené se zmenší o 1. Řazení končí, jakmile je velikost neseřazené posloupnosti rovna 0. Výsledné pole je někde v uprostřed dvojnásobného pole. Pro identifikaci levého a pravého konce seřazené posloupnosti se používá levý a pravý ukazatel.

Algoritmus Binary Insert sort – mod.



Obr. 2.2

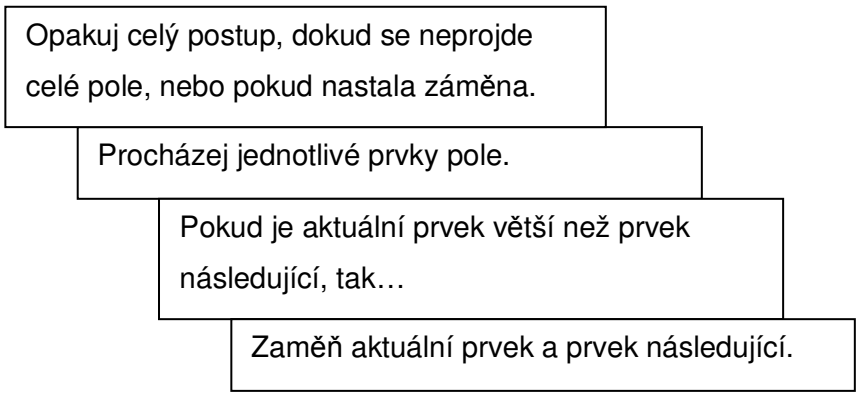
Modifikací algoritmu bylo dosaženo jistého zrychlení, protože se přesouvá méně prvků. K nejvýraznějšímu zrychlení dochází u opačně seřazené posloupnosti. Nevýhodou však zůstává fakt, že se muselo použít dvojnásobně velké pole. Paměťové nároky jsou tedy vyšší.

2.3 Bubble sort

Metoda dostala název podle svého chování, kdy jednotlivé prvky „probublávají“ polem. Ve své nejjednodušší variantě se jedná o 2 vnořené cykly, z nichž každý projde celé pole od začátku až do konce. Ve vnitřním cyklu se porovnávají 2 sousední prvky a jsou-li v opačném pořadí, než je výsledné požadované pořadí, tak se zamění navzájem mezi sebou. Řadíme-li pole vzestupně, prvek s největší hodnotou „probublá“ až na konec.

Drobnou modifikací tohoto algoritmu je zavedení pomocné booleovské proměnné, která hlídá zda v průchodu polem došlo alespoň k jedné záměně. Pokud k žádné záměně dvou prvků nedošlo, tak je pole seřazené a algoritmus se může ukončit.

Algoritmus Bubble sort



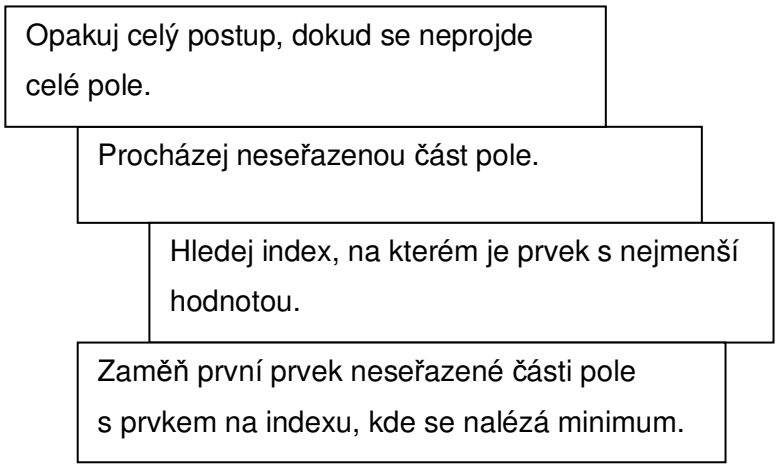
Obr. 2.3

Algoritmus má kvadratickou časovou složitost. Modifikací uvedenou v předchozím odstavci se dosáhne zlepšení při řazení uspořádané posloupnosti.

2.4 Select sort

Jedná se o nejzákladnější algoritmus, který projde celé neseřazené pole a vybere nejmenší prvek. Toto minimum zamění s prvním prvkem neseřazené posloupnosti, pokud řadíme vzestupně zleva doprava a máme vlevo posloupnost seřazenou a vpravo neseřazenou. Po záměně prvků se sníží velikost neseřazené části o 1 a velikost seřazené části se o 1 zvětší.

Algoritmus Select sort



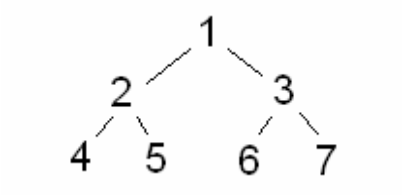
Obr. 2.4

Algoritmus je velice jednoduchý, ale má kvadratickou časovou náročnost.

2.5 Heap sort

Heap sort patří mezi algoritmy pracující na principu výběru. Ke své činnosti využívá vlastnosti binárního stromu, přesněji vztahů mezi rodiči a potomky.

Princip je založen na vlastnostech hromady – heap, která je reprezentována binárním stromem. Prvky z pole jsou umístěny do binárního stromu tak, že se berou prvky pole zleva doprava a staví se z nich strom od kořene.

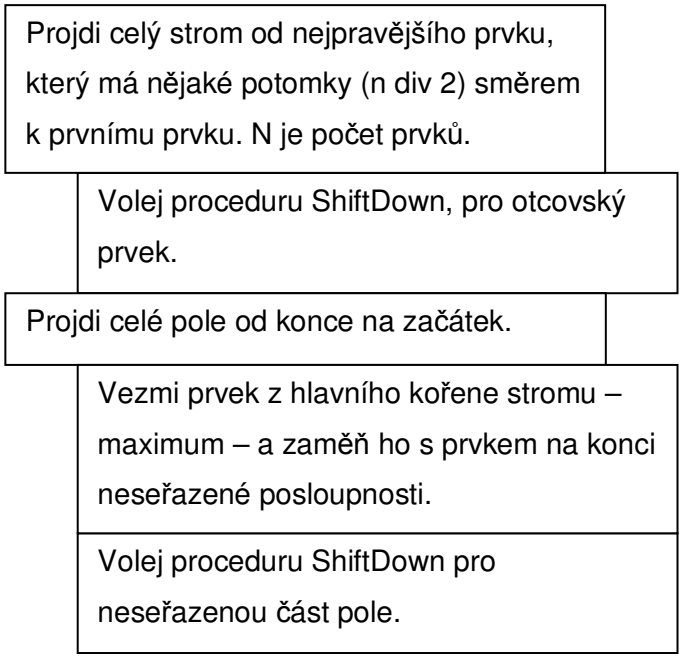


Obr. 2.5

Pro rodiče platí, že pokud je na indexu „i“, jeho levý potomek je na indexu „2i“ a jeho pravý potomek je na indexu „2i+1“. Algoritmus pracuje tak, že znovu ustavuje hromadu, která se poruší. Porušení hromady nastává v okamžiku, kdy odeberme prvek z hlavního kořene a zařadíme ho do výsledné posloupnosti. Poté se znovu ustaví hromada, aby se do hlavního kořene stromu dostal opět největší prvek. Prvek se dostává nahoru postupnou výměnou s ostatními prvky a prvky s nižší hodnotu se dostávají níže. Na začátku algoritmu je nutné provést prvotní ustavení hromady, kde se nejprve celý strom rozdělí na malé úseky – stromy o 2-3 prvcích. Tyto malé stromy se ustaví tak, aby byl v kořeni vždy největší prvek. Následně se spojí vždy dva malé stromy do jednoho většího a strom se opět znovuustaví. Malé stromy se takhle spojují, dokud není postaven původní strom. Následně se již volá procedura znovuustavení v cyklu pro celý strom. Prvky z kořene se přesouvají do výsledné posloupnosti, která se staví směrem od konce pole. Nové nalezené maximum se zaměňuje s posledním prvkem neseřazené části pole a neseřazená část pole se snižuje o 1.

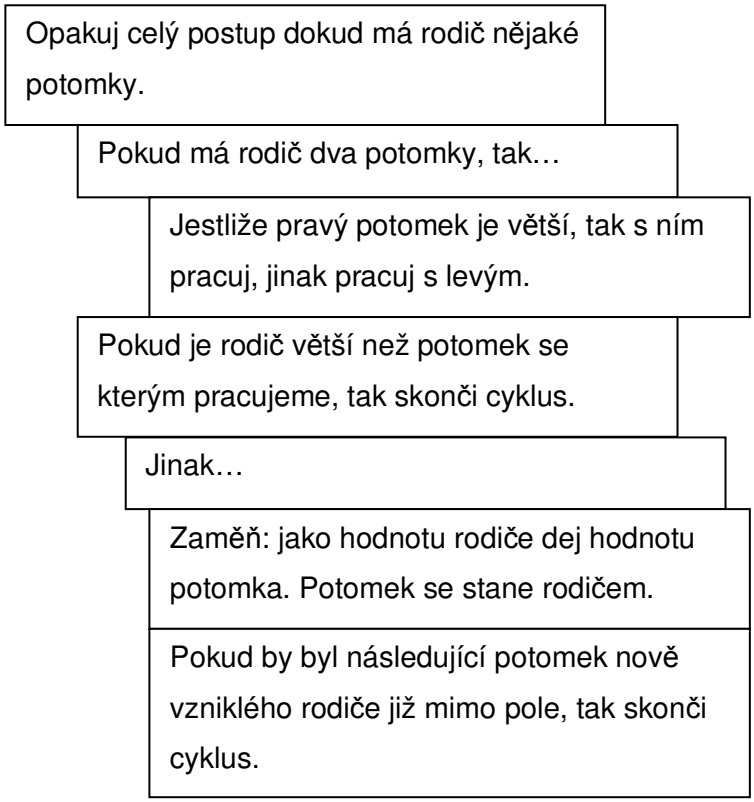
Procedura pro znovuustavení hromady se obvykle nazývá „ShiftDown“.

Algoritmus Heap sort



Obr. 2.6

Procedura ShiftDown



Obr. 2.7

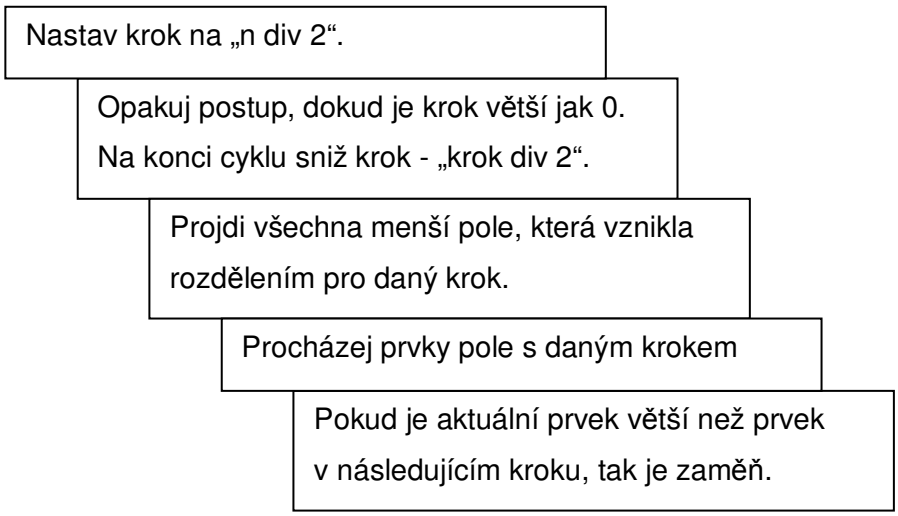
Algoritmus má logaritmickou časovou složitost.

2.6 Shell sort

Tento algoritmus pracuje podobně jako Bubble sort. Neprochází se však postupně prvek po prvku, ale porovnávané prvky se prochází s určitým krokem. Na začátku je krok „n div 2“ a stále se snižuje pomocí „div 2“. Pole se tedy rozdělí na menší úseky důsledkem zvoleného kroku. Pokud je např. krok 5 a pole má 20 prvků, budou v prvním malém poli prvky s indexem 1, 6, 11, 16 v druhém 2, 7, 12, 17 atd.

Prvky se tedy rychleji dostanou na svou přibližnou pozici. Následným snížením kroku je řazení jemnější a pole se rozdělí na méně částí. Posledním průchodem pole je klasický bubble sort, kdy krok dosáhl hodnoty 1. Poté algoritmus končí.

Algoritmus Shell sort



Obr. 2.8

Metoda je velice rychlá a její algoritmus je snadný a jednoduše pochopitelný. Její rychlost se pohybuje někde mezi Heap sortem a Quick sortem.

2.7 Quick sort

Metoda je rekurzivní a pracuje na principu „rozděl a panuj“.

2.7.1 Rozděl a panuj

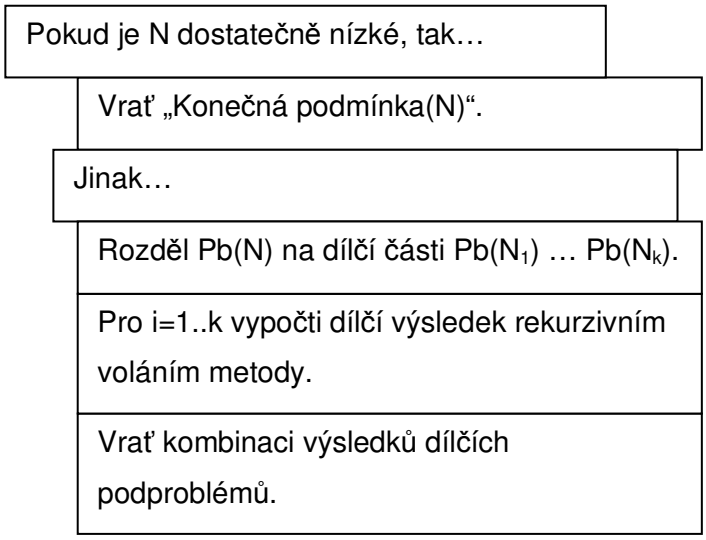
Programování typu „rozděl a panuj“ využívá základní vlastnost rekurze: rozdělení problému na konečný počet podproblémů stejného typu a spojení dílčích řešení pro nalezení řešení obecného. Pokud označíme řešený problém jako Pb a rozměr dat jako N, potom můžeme uvedený zákrok zapsat ve tvaru:

$$Pb(N) \rightarrow Pb(N_1) + Pb(N_2) + \ldots + Pb(N_k)$$

Problém řádu N byl rozdělen na k podproblémů. Znak „+“ znázorňuje, že výsledný problém dostaneme kombinací dílčích podproblémů.

Rozdělení problému neprobíhá jen z estetických důvodů, ale kvůli zvýšení efektivity programu. Jinak řečeno, jedná se o urychlení algoritmu. Technika „rozděl a panuj“ dovoluje v mnoha případech snížit časovou složitost algoritmu např. z $O(n)$ na $O(\log_2 n)$. Na druhé straně existuje skupina úloh, u kterých použití této metody časový průběh algoritmu neovlivní.

Formální zápis metody



Obr. 2.9

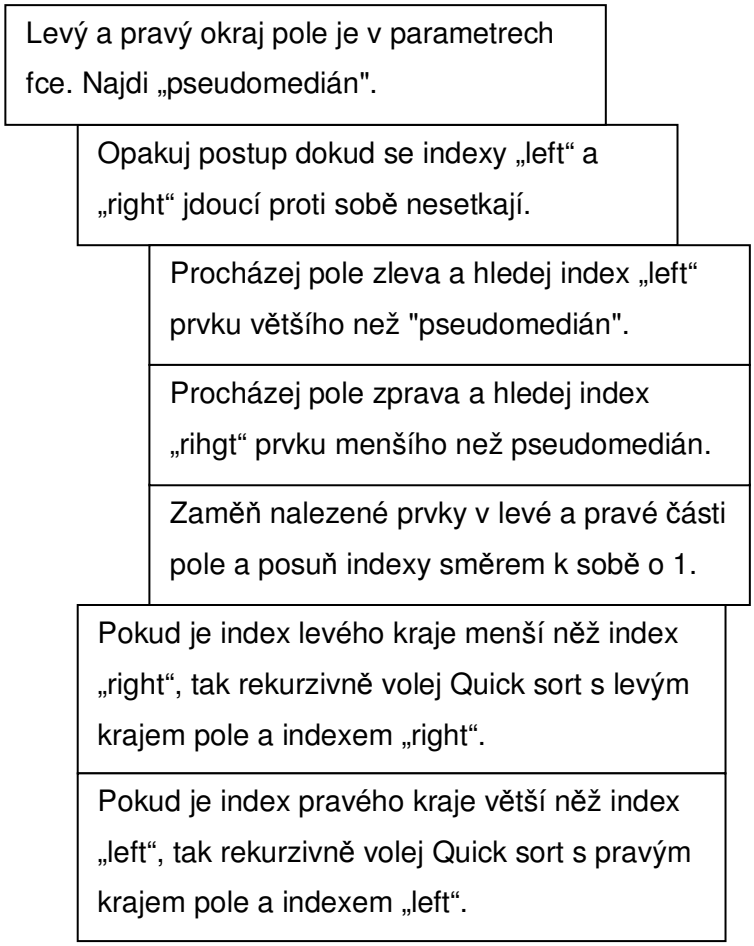
2.7.2 Princip metody Quick sort

Její princip spočívá v určení prvku zvaného „medián“. Medián je prvek, který má tu vlastnost, že polovina prvků je menší a druhá polovina prvků je větší. Potom se jednoduchým algoritmem prvky seřadí.

Hledá se první prvek vlevo od mediánu, který je větší než medián. Ten se zamění s prvním pravým prvkem (bráno odzadu), který je menší než medián. Takto se pole prochází obousměrně, dokud se nepotká procházení uprostřed. Stanovení mediánu je náročná operace, proto se používá jako medián prvek, který je uprostřed. Tento prvek se nazývá „pseudomedián“. Jedná se tedy o střed pole. Indexy, které se setkají a ukonči cyklus porovnávání prvků, jsou novými indexy menších částí pole, pro které se volá rekurzivně algoritmus Quick sort. Zavolá se nejprve pro levou část a pak pro pravou. V nově zavolaném Quick sortu se pracuje již s menším polem. Rekurze končí v okamžiku, jakmile meze předávaného pole nejsou kladné. Tedy pokud předávané pole nemá žádný prvek.

Quick sort se musí na začátku zavolat s počátečními hodnotami indexů pole, což je index prvního a posledního prvku pole.

Algoritmus Quick sort



Obr. 2.10

Metoda Quick sort je jednou z nejrychlejších metod. Časová náročnost je logaritmická.

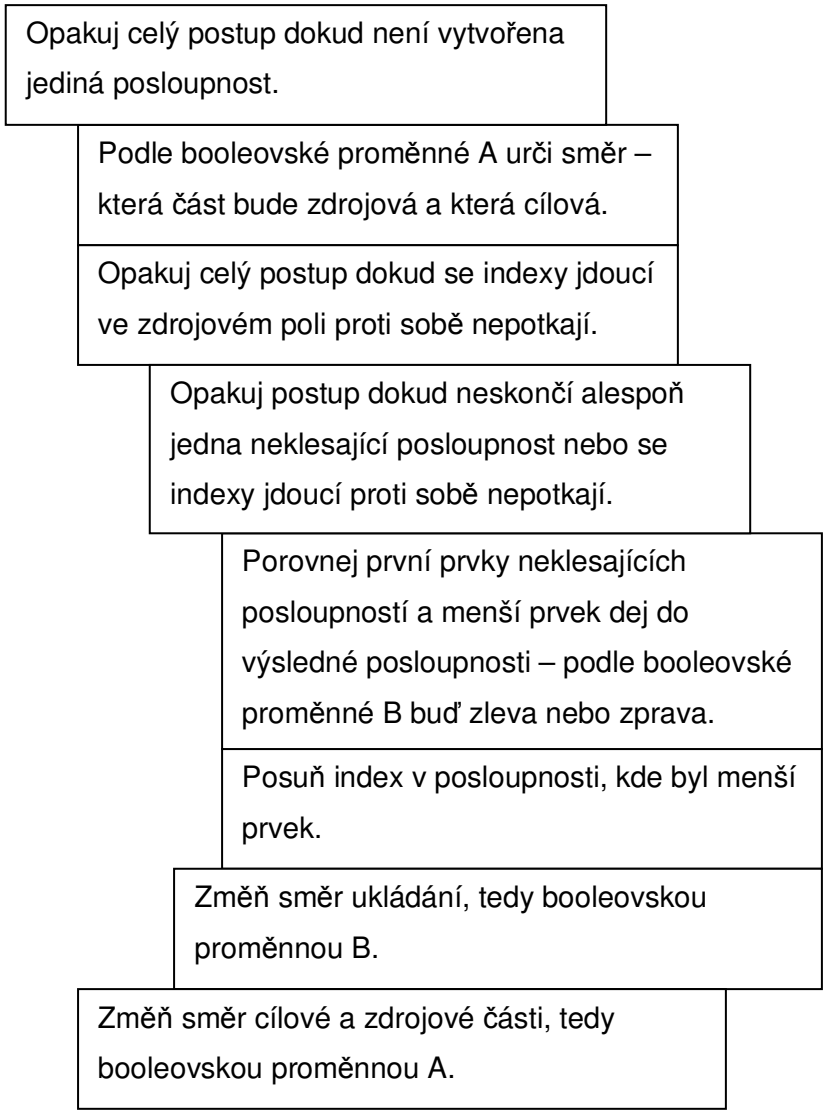
2.8 Merge sort

Algoritmus se vyznačuje tím, že pracuje ve dvojnásobném poli a využívá houpačkový mechanismus. Pracuje na principu spojení dvou neklesajících posloupností. Postupuje polem zároveň zleva i zprava proti sobě a do druhé poloviny dvojnásobného pole dává novou posloupnost vytvořenou ze dvou neklesajících posloupností. Jakmile ve zdrojovém poli skončí alespoň jedna neklesající posloupnost, je zbytek druhé připojen do cílové posloupnosti, která vzniká. První výsledná posloupnost se ukládá do cílového pole zleva.

V dalším kroku se ze zdrojového pole vezmou další dvě neklesající posloupnosti a spojí se do výsledné. Výsledná posloupnost se ukládá do cílového pole tentokrát zprava. Takhle se ukládání střídá, dokud jsou ve zdrojovém poli nějaké posloupnosti, tedy dokud se indexy jdoucí ve zdrojovém poli nepotkají. V cílovém poli vznikl poloviční počet neklesajících posloupností než byl ve zdrojovém poli. Nyní se z cílového pole stává zdrojové, ze zdrojového cílové a celý postup se opakuje.

Algoritmus končí, jakmile vznikne jen jedna neklesající posloupnost. Pokud poslední posloupnost vznikla ve druhé části dvojnásobného pole, překopíruje se do první části, tedy do části původní, kde bylo pole při startu algoritmu.

Algoritmus Merge sort



Obr. 2.11

Algoritmus patří mezi rychlé, jeho časová složitost je logaritmická, ale ke své činnosti potřebuje dvojnásobné pole.

3 Návrh programu

3.1 Studium dodaného programu

Celá praktická část práce vychází z programu, který byl vytvořen někdy v 80. letech minulého století. Jeho funkcí bylo demonstrovat řadicí algoritmy při jejich činnosti a sloužit tak jako názorná pomůcka při výuce. Program byl napsán v jazyce Basic.

3.1.1 Vizuální část

Po spuštění programu se uživateli zobrazí pole prvků, graficky zobrazené pomocí různě dlouhých čar. Jednotlivé čáry mají odlišné barvy. Nebylo by vhodné použít pro každý prvek pole, tedy pro každou čáru, jinou barvu z důvodu názornosti. Technicky to sice vyloučené není, protože můžeme použít 16b. paletu. Jednotlivé odstíny by však nebyly lehce rozeznatelné a ztrácely by jednoduchou názornost. Je tedy využito pouze 15 základních odstínů barev, které se po určité periodě opakují. Hlavní je, aby v seřazené posloupnosti nebyly blízko sebe stejné odstíny barev a nerušilo to celkový dojem uspořádanosti. Výsledný efekt prvků různé hodnoty je tedy vytvořen délkou čar a jejich barvou.

Ovládání programu je textové, tzn., že se ovládá pomocí písmen, kterými se aktivují jednotlivé funkce. V pravém horním rohu je tabulka vypisující všechny použitelné klávesy.

Obr. 3.1

3.1.2 Ovládání programu

Stisk písmene - znaku

E – algoritmus „Einfüge“ (Insert sort)

B – algoritmus „Bubble“ (Bubble sort)

H – algoritmus „Heap“ (Heap sort)

A – algoritmus „Austausch“ (Select sort)

S – algoritmus „Shell“ (Shell sort)

Q – algoritmus „Quick“ (Quick sort)

M – ovládání zvuku – vypnuto/zapnuto

< – snížení rychlosti běhu algoritmu

> – zvýšení rychlosti běhu algoritmu

ESC – ukončení programu

3.1.3 Funkční část

3.1.3.1 Vizuální efekt

Vizuální efekty jsou tvořeny pomocí vykreslování čar, které znázorňují pole integrovaných hodnot. Každá integrovaná hodnota má svou délku čáry.

Znovu vykreslování pole se děje na základě změny, tedy pokud se v poli zamění nějaké prvky, je pole překresleno v nové podobě. Je tedy vidět, jak se jednotlivé prvky v poli přemísťují a dochází k jejich seřazování.

Při běhu řazení se měří čas, jak dlouho algoritmus probíhá. Čas je zobrazován v horní tabulce u příslušného algoritmu. Je to údaj, který odpovídá skutečnému času, po který uživateli programu běží na obrazovce algoritmus. Nejedná se tedy o reálný čas, za který by příslušný algoritmus seřadil pole o 50-ti prvcích se svou příslušnou efektivitou. Takový čas by nebyl pro diváka pozorovatelný, protože by se jednalo řádově o mikrosekundy. Pokud se tedy sníží příp. zvýší rychlost, jedná se pouze o vizuální efekt, kdy dochází k většímu zpoždění běhu algoritmu a naměřený čas tedy bude větší.

3.1.3.2 Zvukový efekt

Běh programu je také doprovázen zvukovým efektem. Každý index v poli má jinou výšku tónu. Nejhlubší tón má nejnižší index, tedy na začátku pole. Výška tónu roste s indexem a index s nejvyšší hodnotou má nejvyšší tón.

Zvuk je přehráván, když se zamění v poli nějaké dva prvky, nebo je prvek zařazen na místo v poli. Např. u algoritmu bubble sort, kde se zaměňují vždy sousední prvky, je znát, že tóny jsou velice blízko sebe a postupným průchodem pole od začátku ke konci je slyšet, jak se výška tónu stále zvedá. U Quick sortu je slyšet, jak jsou tóny vždy z určitého úseku (části pole), tedy jak se pole rozdělí na části metodou „Rozděl a panuj“ a tyto části jsou postupně seřazovány. U Shell sortu je výrazný velký krok ze začátku algoritmu, který se postupně snižuje. V ostatních algoritmech je také slyšet jejich typické chování, pro každou řadicí metodu.

Výstup zvuku je velice jednoduchý, a proto může být realizován pomocí PC speakeru. Zvuk je tedy možné přehrát i na počítačích, které nejsou vybaveny reproduktory, případně zvukovou kartou.

3.2 Návrh nového programu

3.2.1 Výběr prostředí

Výběr prostředí, ve kterém bude budoucí program pracovat a od kterého se budou odvíjet další etapy práce, je velice důležitou částí a nesmí se podcenit. Především mám na mysli grafické prostředí, které bude zapouzdřovat funkce vykreslování. Nemalou část hrají také funkce knihoven, které jsou v daném grafickém prostředí k dispozici. Vývojové prostředí je většinou volbou každého programátora především podle jeho zvyků. Mezi nejčastěji používanými vývojovými prostředími mohu jmenovat DevC++, C++Builder, Microsoft Visual Studio a další.

Při výběru grafického prostředí jsem se zaměřil především na jednoduchost používání a volání funkcí pro vykreslování, protože program má sloužit studentům jako pomůcka. Proto je vhodné, aby i při nahlédnutí do kódu byli schopni bez dalšího studování složitých konstrukcí pochopit, o co se jedná a jak daný problém funguje.

O každém grafickém prostředí bylo nutné nastudovat základní informace a zvážit jejich výhody a nevýhody. Prvním limitujícím faktorem, který vyplývá ze zadání, bylo použití programovacího jazyka C. Většina novějších grafických prostředí je již objektově zaměřena do C++. Dalším cílem bylo, aby na vývoj či úpravu kódu a také jeho překlad nebylo nutné používat dalších „nadbytečných“ knihoven či prostředí. Po prostudování a posouzení různých grafických prostředí mi vyšli 3 „kandidáti“.

Byla to grafická knihovna „Allegro“, potom velice oblíbená GTK+ a grafické prostředí WIN API. Tato prostředí bylo nutné nastudovat o něco podrobněji, abych mohl vybrat jednu konečnou a „nejlepší“ variantu.

Knihovna Allegro je velice zajímavou knihovnou a její používání je jednoduché. Je však zaměřena spíše do oblasti programování starších her, především pro OS DOS. Dokáže zpracovávat grafiku, zvuk, vstupy klávesnice i myši, různé časovače apod. U tohoto projektu by se mělo jednat spíše o příjemné uživatelské rozhraní, jednoduché ovládání a moderní vzhled, na

který jsou všichni navyklí, aby studenti nemuseli složitě studovat ovládání programu. Takový program v dnešní době vidím na principu oken a menu. Tyto požadavky jdou samozřejmě v knihovně Allegro splnit, ale vytváření oken, menu a tlačítek by bylo zbytečně složité a mohlo by znepřehlednit kód pro člověka neznalého speciálních funkcí.

Knihovna GTK+ je velmi známou a oblíbenou knihovnou, vzniklou v OS Unix. Její zkratka GTK+ znamená Gimp Toolkit, podle unixového kreslícího programu Gimp. Současná implementace podporuje platformy Unix i Windows. Knihovny GTK+ jsou šířeny pod licencí LGPL, a proto mohou být použity pro vývoj opensource, freeware, ale i placených komerčních aplikací. Pro Windows i Linux je nutné nainstalovat několik balíčků. Je zde jednoduchý kód pro vytvoření oken, různých tlačítek i menu.

Grafické prostředí WIN API je záležitostí operačního systému Windows. Je to velice jednoduché prostředí, které je intuitivní a snadné na pochopení. K práci a programování ve WIN API není potřeba doinstalovávat žádné další knihovny, pokud se používá některé z vývojových prostředí. Velkou výhodou WIN API je, že má velmi dobré podklady jak na internetu, tak i v knižní podobě a také existuje mnoho návodů, manuálů i diskusí v češtině, což může některým studentům velice pomoci. WIN API samozřejmě podporuje okna, různá tlačítka, menu a mnoho dalších. Vše je velmi jednoduché a intuitivní.

Z těchto tří prostředí pro programování grafiky, které jsem popsal opravdu velice stručně, jsem se rozhodoval mezi GTK+ a WIN API. Allegro nebylo pro své vlastnosti nejvhodnější pro program, který by měl sloužit i jako zdrojový kód studentům začínajícím s programováním. GTK+ má velkou výhodu v tom, že je multiplatformní a také má velice dobré zdroje informací jak na internetu, tak i v nejrůznějších publikacích.

Já jsem se nakonec rozhodl pro prostředí WIN API. Jednak z důvodu dobré dostupnosti návodů článků a publikací jak v angličtině, tak i v češtině, ale také se shodou okolností vyučuje WIN API ve třetím ročníku Fakulty Informačních Technologií na VUT v Brně v předmětu ITU – Tvorba uživatelských rozhraní. A jestliže má tento program jednou sloužit studentům FIT jako pomůcka pochopení řadicích algoritmů, mohou potom zdrojové texty posloužit jako názorný příklad některých konstrukcí právě pro předmět ITU. Lze říci, že jsem tímto krokem nepatrně rozšířil i výslednou použitelnost programu ve smyslu učební pomůcky. Primární cíl však zůstává stejný, tedy vytvořit program, který ukáže principy řadicích algoritmů.

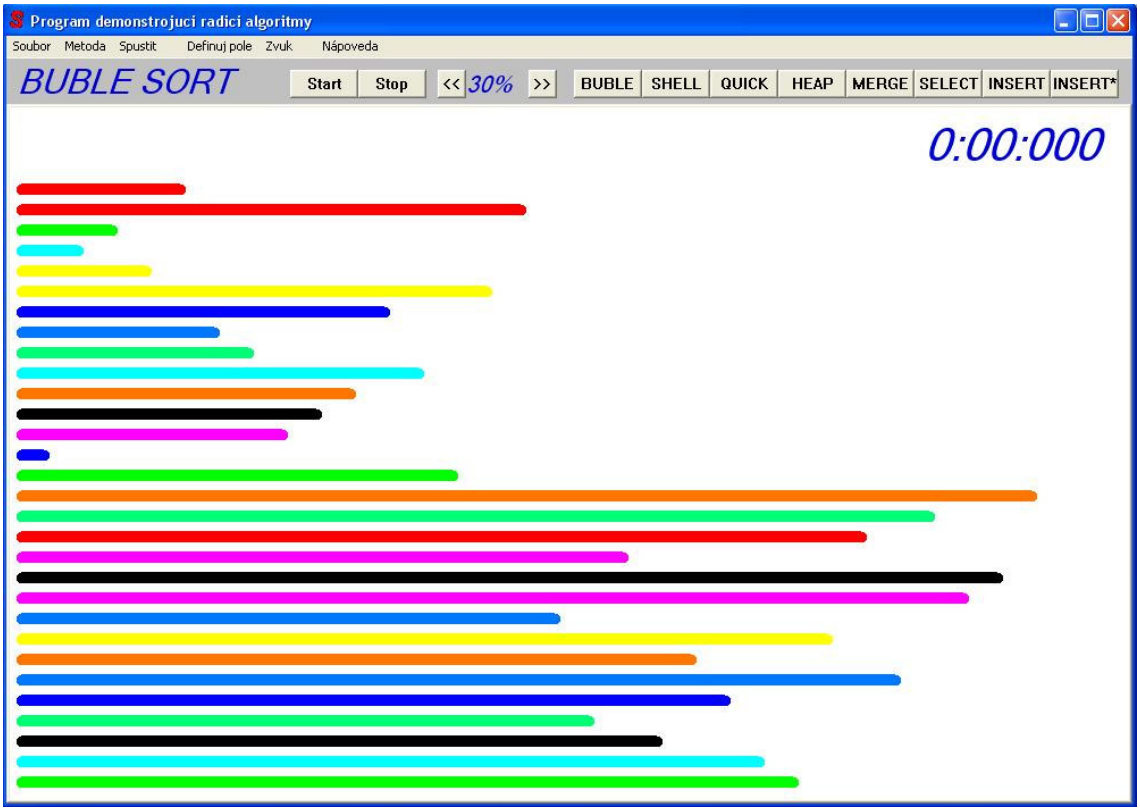
Jako vývojové prostředí jsem zvolil Microsoft Visual Studio 2003. Samozřejmě lze použít i jiná vhodná prostředí, jako Microsoft Visual Studio 2005, DevC++ nebo C++Builder.

3.2.2 Návrh vzhledu

Návrh vzhledu je další důležitou součástí návrhu celého programu. Není to tak zásadní otázka jako výběr programovacího jazyka nebo výběr grafického prostředí, protože s okny, prvky, tlačítky atd. se dá i v průběhu práce pohybovat, přesouvat a upravovat tak vzhled.

Hlavní je, aby program nepůsobil zmateně, aby byl přehledný a každý hned na první pohled poznal, kde jsou funkční prvky, jako je ovládání, kde se zobrazuje výstup apod. Nejčastější je v dnešní době okno, které má nahoře menu a případně další funkční tlačítka. Nebylo by proto vhodné toto zavedené schéma měnit, aby nedocházelo k nepřehlednosti. Barvy programu musí být voleny tak, aby vynikla ta část, která je podstatná a která má nést nějakou hodnotu či informaci .

Program jsem navrhl klasickým stylem jak sem již zmiňoval. Hlavním prvkem je tedy okno, které má ve své horní části menu. Pomocí menu se dají vyvolat všechny funkce implementované v programu. Dále je hlavní okno rozděleno na 2 části. V horním pruhu jsou tlačítka pro rychlou volbu nejpoužívanějších funkcí. Vlevo je název metody, která se právě používá pro řazení prvků. Horní část má celá šedou nevýraznou barvu, aby neupoutávala pozornost. Ve spodní části, která tvoří celý zbytek okna, je vykreslené pole hodnot. Pozadí je bílé, aby bylo výraznější než zbytek okna. V něm je hlavní výstup programu, který demonstruje zvolenou řadicí metodu. Vpravo je potom zobrazen čas, který měří délku běhu algoritmu.



Obr. 3.2

Velikost okna je přizpůsobena pro minimální rozlišení 1024x768, které je na všech počítačích běžně podporované. Velikost se dá zvětšovat normálním způsobem jak je v OS Windows obvyklé.

4 Vlastní realizace

4.1 Implementované algoritmy

Řadicí algoritmy implementované v programu jsou stejné jako v původním „Basic“ programu, který slouží jako předloha. Tyto základní algoritmy jsem po domluvě s vedoucím bakalářské práce rozšířil o další dva, které jsou svou funkcí velice zajímavé a jsou vyučovány v předmětu IAL. Jednotlivé algoritmy byly teoreticky popsány v dřívější kapitole.

V každém algoritmu je volána funkce pro zpoždění, z důvodu vizuálního efektu funkce. Toto zpoždění se řídí podle toho, jak je nastaveno do uživatele na stupnici. Ve zpožďovací funkci se volá překreslení obrazovky, aby bylo vidět novou změnu prvků v řazeném poli. Také se volá funkce pro zvukový efekt, kde se předává hodnota indexu, na kterém se prvek nachází a také hodnota prvku. Podle zvoleného režimu přehrávání zvuků, se přehrává zvuk buďto podle indexu, a nebo podle hodnoty prvku.

Zdrojový kód všech implementovaných řadicích algoritmů je v příloze.

4.1.1 Bubble sort

Jeden z nejjednodušších algoritmů pro seřazování prvků. Algoritmus je implementován pomocí dvou cyklů. Vnější cyklus je „while“ a končí, jakmile nedojde k žádné záměně dvou prvků v poli. Jedná se o zlepšení základní verze algoritmu, protože pokud je posloupnost seřazena dřív než za n^2 průchodů, cyklus skončí. V krajním případě u seřazené posloupnosti proběhne cyklus pouze jednou.

Vnitřní cyklus je vytvořen pomocí cyklu „for“, který prochází postupně všechny prvky pole od začátku ke konci a porovnává vždy sousední prvky. Pokud je prvek na nižším indexu větší než prvek na vyšším indexu, zamění je. Po zaměnění prvků je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším.

4.1.2 Shell sort

Algoritmus pracující na velice jednoduchém a důmyslném principu. Prochází pole podobně jako „Bubble sort“, avšak s tím rozdílem, že v průběhu řazení mění svůj krok. Skládá se ze dvou cyklů. Jednoho vnitřního a jednoho vnějšího.

Vnější cyklus je „while“, který prochází pole tak dlouho, dokud je krok průchodu větší jak 1, nebo pokud je stále co řadit, tedy pokud nastala v minulém průchodu nějaká záměna. Jinými slovy, cyklus skončí jakmile nedošlo k záměně a krok je menší nebo roven 1. Při každém novém průchodu cyklem se krok dělí dvěma, tedy se stále snižuje.

Vnitřní cyklus je „for“, který prochází pole prvek po prvku, ale prvky jsou porovnávané s krokem, který je nastaven ve vnějším cyklu. Začíná krokem „n div 2“. Srovná se tedy velikost prvního prvku a prvku, který je od prvního vzdálen o přednastavený krok. Výsledný efekt je stejný, jako při průchodu všech částí pole s daným krokem.

Pokud je prvek na nižším indexu větší než prvek na vyšším indexu, tak se tyto prvky zamění. Po zaměnění prvků, je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším.

4.1.3 Quick sort

Algoritmus „Quick sort“ je jedním z nejrychlejších řadicích algoritmů. Pracuje na principu „Rozděl a panuj“. Algoritmus se volá rekurzivně, vždy pro jinou část pole.

V první řadě je nutné stanovit „pseudomedián", který se vypočítá jako střed pole, mezi levým a pravým okrajem. „Levý“ a „pravý“ okraj je předáván jako parametr funkce a ke každému rekurzivnímu volání jsou hranice pole jiné.

Hlavní cyklus „while“ s podmínkou na konci běží tak dlouho, dokud se indexy jdoucí proti sobě nesetkají. Indexy jdoucí proti sobě jdou v poli ohraničeném „levým“ a „pravým“ okrajem. Dále se hledá první prvek směrem zleva doprava, který je větší jak „pseudomedián". Poté se hledá první prvek směrem zprava doleva, který je menší jak „pseudomedián“. Pokud se indexy nepřekřížily, jsou tyto dva nalezené prvky zaměněny.

Po zaměnění prvků je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším.

Po skončení hlavního cyklu se rekurzivně zavolá „Quick sort“ s novými indexy pole.

4.1.4 Heap sort

Algoritmus patří mezi rychlé řadicí metody a pracuje pomocí vlastností v hromadě. Hromada je reprezentovaná pomocí binárního stromu. Hlavní funkcí v tomto algoritmu je znovuustavení hromady. K tomuto účelu slouží funkce „ShiftDown“.

V hlavní části algoritmu jsou 2 cykly „for“. První cyklus je přípravný, kde se musí prvotně ustanovit celá hromada. Z vlastností binárního stromu vyplývá, že pokud prvek na pozici „x“ je rodič, tak jeden jeho potomek je na pozici „2*x“ a druhý na pozici „(x*2)+1“. Prvotní ustavení hromady se tedy volá pro jednotlivé prvky postupně, jak se prochází pole. Polem se prochází od hodnoty „(n div 2)-1“ (v jazyku C se pole počítá od 0 do n-1) a postupuje se směrem k nule. Pro každý prvek se volá funkce „ShiftDown“.

Druhý cyklus prochází pole od posledního prvku k prvku prvnímu. Po každém ustavení hromady je v poli první prvek největší. Tento prvek se vezme a vloží se odzadu do seřazené posloupnosti. Následné volání „ShiftDown“ je již s polem o jeden prvek menší. Pole se zkracuje odzadu. Jakmile se zaměňuje první prvek s prvkem posledním v neseřazené posloupnosti, je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším.

Funkce „ShiftDown“ prochází neseřazenou část pole a znovu ustaví strom. To znamená, že do hlavního kořene, tedy na začátek pole, se dostane prvek s největší hodnotou. Funkce má jeden cyklus „while“. Prochází se tak dlouho, dokud se procházení neocitne mimo strom, nebo pokud není hromada znovu ustavena. V těle cyklu se zjistí, zda aktuálně testovaný rodič má jednoho či 2 potomky. Pokud má dva potomky, vybere se větší z nich a testuje se, zda je větší než jeho rodič. Pokud ano, zamění se rodič se svým potomkem. Potom je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším.

4.1.5 Merge sort

Algoritmus „Merge sort“ využívá ve své činnosti tzv. houpačkový mechanismus a pracuje ve dvojnásobném poli. Rozšíření pole je zde pro demonstrační účely vytvořeno stylem, že původní pole se zmenšilo na polovinu a pak se pracuje s prostorem původního pole. Proto je nejprve volána přípravná funkce pro „Merge sort“, která pole zmenší a z ní je teprve zavolána funkce pro řazení.

Myšlenkou metody je seřazení dvou neklesajících posloupností do jedné. Ve zdrojovém poli se postupuje proti sobě a čtou se dvě proti sobě jdoucí neklesající posloupnosti. Do druhé části pole se vkládá výsledná spojená posloupnost. Jakmile se přečtou ve zdrojovém poli dvě neklesající posloupnosti, pokračuje se dalšími dvěmi, ale výsledná spojená se ukládá do cílového pole z druhé strany. Takto se počet posloupností snižuje, až vznikne jedna jediná výsledná. Algoritmus je sestaven pomocí tří vnořených cyklů „while“, které mají podmínku na konci.

Vnější cyklus probíhá tak dlouho, dokud vzniká více jak jedna posloupnost. V každém průběhu se mění zdrojové pole na cílové a opačně. Uvnitř tohoto cyklu je další, který prochází zdrojové pole z obou stran. Pracuje tak dlouho, dokud se indexy jdoucí proti sobě nepřekříží. Uvnitř tohoto cyklu se v každém průchodu změní směr ukládání posloupnosti do cílového pole. Ukládá se střídavě zleva doprava a zprava doleva.

Nejvnitřnější cyklus prochází neklesající posloupnosti a řadí je do výsledné. Při své činnosti hlídá, zda se jedná stále o neklesající posloupnost a testuje, který prvek z posloupnosti je menší. Menší prvek je dán do výsledné posloupnosti. Při zařazení prvku do výsledné posloupnosti je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem řazeného prvku a následně s indexem, kam se prvek zařazuje do výsledné posloupnosti.

Výsledné seřazené pole může skončit v rozšířené části původního pole. V takovém případě je přesunuto na původní startovní místo.

4.1.6 Select sort

Je to nejjednodušší algoritmus. Pracuje na principu výběru, kdy se hledá nejmenší prvek v poli a zařadí se do výsledné posloupnosti. Tento algoritmus patří mezi časově náročné, ale jeho princip je jedním z nejzákladnějších. Je implementován pomocí dvou vnořených cyklů „for“. Na začátku

je celé pole neseřazené a v průběhu řazení se seřazená posloupnost tvoří zleva doprava a neseřazená posloupnost se tedy zleva zmenšuje.

Vnější cyklus prochází postupně celé pole. V každém běhu vezme jeden prvek a dá ho do pomocné proměnné. Tento prvek je vždy prvním prvkem neseřazené části pole. Ve vnitřním cyklu se prochází celá část neseřazeného pole, tedy od prvku, který je v pomocné proměnné dále doprava. Hledá se prvek, který je nejmenší. Jakmile se projede celé pole, je v pomocné proměnné prvek s nejmenší hodnotou. Také je nutné si uložit index, na kterém se tento prvek nachází. Poté se zamění prvek, který je na začátku neseřazené posloupnosti s prvkem, který se našel jako nejmenší. Následně je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem nižším z obou zaměňovaných prvků a následně s indexem vyšším. V následujícím průchodu se opět zmenší neseřazená část pole o jedna. Prvek, který jsme zařadili na začátek neseřazené části pole se stane prvkem na konci seřazené posloupnosti.

4.1.7 Binary Insert sort

Tento algoritmus patří mezi jednoduché, ale časově náročné. Pracuje na principu vkládání, kdy se vezme první prvek neseřazené posloupnosti a v seřazené posloupnosti se hledá místo, na které patří. Jakmile se najde místo kam se má prvek zařadit, musí se pro něj uvolnit místo. Místo se uvolňuje posunutím ostatních prvků o jednu pozici. Posouvání části pole je časově náročná operace. Algoritmus je implementovaný pomocí tří cyklů.

Vnější cyklus je „for“ a prochází celé pole. Seřazená posloupnost se tvoří zleva doprava a na začátku je v ní jeden prvek. V každém průchodu cyklem se seřazená část zvětší o jedna. Vždy se vezme první prvek z neseřazené posloupnosti a dá se do pomocné proměnné. Ve vnitřním cyklu se pomocí binárního vyhledávání hledá v seřazené části místo, kam patří prvek v pomocné proměnné. Druhý cyklus, který následuje po binárním vyhledávání, uvolní místo pro hledaný prvek podle nalezeného indexu. Tuto funkci zastává cyklus „for“. Přesune všechny prvky od nalezené pozice směrem doprava. Po posunu každého prvku je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem prvku, který se posouvá a následně s indexem, kam se prvek posouvá.

Na uvolněné místo je vložen prvek z pomocné proměnné. Je opět zavolána funkce pro zpoždění a vykreslení pole a také pro zvukový efekt.

4.1.8 Binary insert sort – modifikovaný

Jedná se o upravenou verzi binárního „Insert sortu“. Pracuje ve dvojnásobně velkém poli. Z důvodů demonstrace chodu algoritmu je nutné před samotným voláním algoritmu upravit řazené pole. Pole se zmenší na polovinu a algoritmus pak bude využívat celý prostor původního pole. Bude tedy k dispozici dvojnásobný prostor pro řazené pole. Výsledná posloupnost je umístěna někde uprostřed celého pole. Vizuálně začíná prvním prvkem zleva.

Úprava algoritmu spočívá v části kódu, kde se uvolňuje místo pro prvek, který má být začleněn do výsledné posloupnosti. Seřazené pole se posouvá doleva, pokud nalezené místo je blíže levému okraji seřazené posloupnosti, anebo doprava, pokud je blíže pravému okraji. Tato úprava efektivně zvýší rychlost řazení u opačně seřazené posloupnosti.

Algoritmus je tvořen jedním vnějším cyklem „while“. V každém průchodu se hledá místo pro jeden prvek. Vnitřní cyklus pro binární vyhledání je „while“. Následně se musí zjistit, jestli je místo, na které se má prvek zařadit, blíže levému nebo pravému konci seřazené posloupnosti. Podle toho se upraví velikost seřazené části. Pokud se bude posouvat doleva, musí se posunout o jednu pozici vlevo i místo, kam se má začlenit prvek. Je to z toho důvodu, že binární vyhledávání by zařadilo prvek směrem vpravo. Ostatní proměnné se nastaví tak, aby mohlo proběhnout příslušné posunutí. Podle připravených kroků se posune část pole buď vlevo, nebo vpravo. Toto posunutí je vytvořeno cyklem „while“. Při posunutí každého prvku je volána funkce pro zpoždění a vykreslení pole. Pak je 2x zavolána funkce pro zvukový výstup. Jednou s indexem prvku, který se posouvá a následně s indexem, kam se prvek posouvá. Na uvolněné místo je vložen prvek z pomocné proměnné. Je opět zavolána funkce pro zpoždění a vykreslení pole a také pro zvukový efekt.

4.2 Implementace programu

Program ve WIN API funguje pomocí zpráv. Pro jeho činnost je důležitá hlavní funkce a procedura na odchytávání a zpracování zpráv. Pokud okno dostane zprávu, ať už od systému nebo od uživatele, je zachycena a pošle se do funkce zpracovávání zpráv. Pokud je pro danou zprávu nastavena nějaká akce, tak se provede, jinak se zpracuje defaultním způsobem.

4.2.1 Hlavní funkce

V Hlavní funkci WIN API jsou nastaveny vlastnosti hlavního okna. Okno je vytvořeno voláním funkce „CreateWindow()“. Následně jsou pak nastaveny vlastnosti podokna, které bude součástí hlavního okna. Následuje smyčka odchytávání zpráv, ve které se čtou všechny zprávy, které jsou oknu doručeny. Zachycená zpráva je buďto přeložena a poslána ke zpracování, a nebo je smyčka ukončena a celý program končí.

4.2.2 Procedura hlavního okna

Zde se vyhodnocují zprávy, které náleží hlavnímu oknu. Zprávy se rozdělují pomocí přepínače „switch“. Při vytvoření hlavního okna, je přijata zpráva WM_CREATE. Je vytvořeno podokno uvnitř hlavního okna. Podokno bude sloužit jako hlavní plocha pro zobrazování řazeného pole. Dále jsou na zprávu WM_CREATE vytvořena všechna ovládací tlačítka. Jsou inicializovány základní vlastnosti, tedy nahrání předdefinované posloupnosti do pole a nastavení zvuku.

Další zprávou, která se zpracovává, je WM_PAINT. Tuto zprávu dostává okno pokaždé, když je potřeba překreslit hlavní okno. Se zprávou pro hlavní okno se automaticky generuje zpráva pro překreslení podokna. Podokno zachytává a zpracovává svoje zprávy. Jedinou akcí je zde vykreslení textu do horního pruhu.

Zpráva WM_DESTROY je přijata před zavřením okna. Zde se pouze pošle ukončovací zpráva.

Zpráva WM_SIZE je přijata při změně velikosti okna. Změní se velikosti hlavního okna a podle nich se přepočítají rozměry podokna.

Pro zpracování kliknutí na tlačítko, nebo výběru položky z menu, slouží zpráva WM_COMMAND. Zde jsou popsány všechny akce na jednotlivé podněty od uživatele. Při stisknutí tlačítka výběru metody nebo vybráním ekvivalentní položky v menu, je nastavena metoda do proměnné „g_method“. Proměnná je globální, protože k ní přistupují různé funkce v průběhu programu a předáváním hodnoty jako parametr by se celá situace příliš komplikovala. Po té je zavoláno překreslení okna, aby se zobrazil text zvolené metody.

Při stisku tlačítka „START“ na spuštění řazení se nastaví parametry do struktury vlákna, ve kterém řazení poběží. Zkontroluje se, jestli již není vlákno vytvořeno, což by znamenalo, že řazení již probíhá. Je to kontrola, aby se nespustilo řazení vícekrát najednou. Pokud žádné vlákno neběží, vytvoří se vlákno nové a zavolá se funkce vlákna a zároveň se nastaví příznak toho, že bylo vytvořeno vlákno a běží.

Při stisku tlačítka na „STOP“ pro zastavení řazení, je zkontrolováno jestli vlákno existuje a je tedy co zastavovat. Pokud ano, tak je vlákno zrušeno a běh algoritmu ukončen. Ve struktuře, podle které se řídí vlákno, je nastaven příznak toho, že žádné vlákno není vytvořeno.

Při zvolení povelu na definování pole se volají funkce, které do pole prvků nastaví hodnoty. Podle zvoleného povelu je to předdefinovaná posloupnost, která je pokaždé stejná, aby bylo možné si na stejné posloupnosti spustit všechny algoritmy. Dále je to opačná posloupnost, na které se projevují zajímavé vlastnosti algoritmů. Jako poslední možnost je to generování náhodné posloupnosti.

Dále je možné vyvolat nápovědu tvořenou pouze jednoduchým způsobem pomocí „MessageBoxu“. Stejně tak i položka „O programu“.

Při zvolení možnosti „Konec“ pro ukončení programu, je poslána ukončovací zpráva, kterou přijme již zmíněné zpracování WM_DESTROY.

Nastavování časového zpoždění je možné od 10% po kroku 10% až do 100%. Podle předem vypočítaných hodnot se do proměnné času přičítá nebo odečítá předdefinovaná konstanta. Hlídají se mezní hodnoty, tedy maximální a minimální hodnota zpoždění. Po každém zvýšení zpoždění se volá funkce pro překreslení okna, aby se vypsala nová hodnota rychlosti řazení.

Poslední z těchto povelů je nastavení zvukového efektu. Zde se pouze do příznaku zvuku uloží jedna ze tří hodnot. Bez zvuku, zvuk podle indexu v poli a zvuk podle hodnoty prvku. Příznak zvuku je součástí struktury pro vlákno, aby mezi sebou mohlo komunikovat běžící vlákno a původní okno.

4.2.3 Procedura podokna

Podokno, které je součástí hlavního okna, má také svoje zpracovávání zpráv. Podokno je vytvořeno pouze pro účel vykreslení řazeného pole a času. Jedinou zprávou, kterou procedura podokna zpracovává, je tedy WM_PAINT pro překreslení podokna. Při této zprávě se volá funkce, která vykreslí řazené pole a vypíše čas, jak dlouho řazení běží.

4.2.4 Spuštění řadicího algoritmu

Při spuštění zvoleného algoritmu se zavolá funkce vlákna. Slouží k tomu, aby hlavní okno mohlo normálně reagovat na zprávy a obsluhovat je a přitom mohlo probíhat řazení. Všechny informace o čase zvuku atd., jsou ve struktuře předávané při vytvoření vlákna. Před spuštěním vybrané metody se musí zvolit, která metoda se má spustit. O to se postará „switch“, který podle proměnné „g_method“ zavolá požadovanou metodu. Jednotlivé řadicí metody a jejich implementace byly popsány dříve.

4.2.5 Vykreslování ve WIN API

Pro vykreslování ve WIN API se používá „Divice context“. Je to abstraktní kreslící plocha, která má své rozměry a atributy. Před vypsáním textu nebo vykreslením nějaké grafiky je nutné zavolat funkci zahajující kreslení a získat tak „Divice context“. Po skončení kreslení je nutné „Divice context“ uvolnit.

4.2.6 Funkce vykreslení textu

Funkce pro vykreslování textu vypisuje do hlavního okna název řadicí metody a rychlost jakou bude řadicí algoritmus běžet. Rychlost je udávaná v procentech .Nejprve je nutné nastavit požadované vlastnosti fontu jako je velikost písma, barva apod. Text metody je vypsán podle

hodnoty nastavené v proměnné „g_method“. Rychlost v procentech je nejdříve převedena na procenta. Poté je text vypsán. Text se vypisuje funkcí „TextOut“ na požadované souřadnice.

4.2.7 Funkce vykreslení pole

Funkce pro vykreslování pole vykresluje do podokna pole prvků a do horního rohu vypisuje čas. Čas se spočítá podle hodnoty uložené při startu programu. Formát času je nutné upravit na hodnotu, která se vypíše. Je tedy nutné přidat nulu u času, který je zobrazen jednou číslicí. To je nutné udělat u setin, které se vypisují na tři platná místa a u vteřin, které se vypisují na dvě platná místa. Čas dostane tedy formát např. 3:05:085. První jsou minuty, následují vteřiny a posledním údajem jsou setiny. Pro vypsání času je nejprve nastaven font, velikost písma, kurzíva a barva. Čas je vypsán do pravého horního rohu podokna.

Pro vykreslení pole se nastaví nejprve pero, kterým se bude kreslit. U pera se nastaví tloušťka a barva. Generování barvy je závislé na tom, jakou hodnotu má vykreslovaná čára. Jinými slovy, prvek s jakou hodnotou se bude vykreslovat. Barev je předdefinovaných 10. Prvky od hodnoty 1 až 10 mají tedy barvy předdefinované. Následující desítka se opakuje. Opakovaní barev je zvoleno tak, aby v seřazené posloupnosti nebyly vedle sebe dvě stejné barvy nebo stejné odstíny.

Čára reprezentující jeden prvek v poli je vykreslena na pozici podle indexu v poli. První poloha je nastavena jako konstanta a další prvky se vykreslují s konstantním krokem přičítajícím se k y-lonové souřadnici. Délka čáry se řídí podle velikosti prvku v poli. Je určena výchozí délka a ta je vynásobena hodnotou prvku v poli, který je právě vykreslován. Tak je dosaženo toho, že prvky s nízkou hodnotou jsou reprezentovány krátkými čarami a prvky s vysokou hodnotou čarami dlouhými.

4.2.8 Zvukový efekt

Zvukový efekt programu je možné nastavit na 3 různé hodnoty. Zvuk trvá vždy konstantní dobu. Při volání funkce se jako parametry předávají: index prvku v poli a hodnota prvku, pro který se má zvuk přehrát. První možností je vypnutý zvuk. Při této volbě se žádný zvuk nepřehraje, ale

počká se pouze dobu, po kterou by byl případný zvuk přehráván, aby se rychlost řazení pole neměnila při vypnutém a zapnutém zvuku.

Další možností zvukového efektu je řízení frekvence tónu podle indexu, na kterém se prvek v poli nachází. Je zvolena základní frekvence zvuku a ta je vynásobena příslušným indexem prvku. Prvky na nízkém indexu mají hluboké tóny, tedy tóny s nízkou frekvencí a prvky s vysokým indexem v poli mají vysoký tón, neboli vysokou frekvenci zvuku.

Třetím typem zvuku je řízení výšky tónu podle hodnoty prvku. Princip je shodný jako u zvuku řízení indexem, jen s tím rozdílem, že základní frekvence je násobena velikostí prvku. Prvky s malou hodnotou přehrávají hluboké tóny, prvky s velkou hodnotou přehrávají tóny vysoké.

5 Rozšíření

5.1 Metody

Program jsem oproti původnímu zadání rozšířil o vhodné prvky a úpravy. Prvním rozšířením je přidání dvou řadicích algoritmů, které jsou svou funkcí velice zajímavé. Prvním algoritmem je metoda „Merge sort“. Tato metoda ukazuje houpačkový mechanismus využitý při řazení pole. Druhou metodou je modifikovaná verze „Binárního insert sortu“. Tento algoritmus ukazuje zrychlení, kterého lze docílit zdvojením pole a posunováním částí pole, která je menší a tedy posun je méně náročný. Oba algoritmy jsou popsány jak v teoretické části této práce, tak i v části popisující implementaci.

5.2 Zvuk

V původním programu byla možnost volby zvuku buďto vypnutého, nebo zvuku, který se přehrává na základě indexu v poli. Já jsem přidal ještě třetí možnost a to přehrávání zvuku podle hodnoty prvku v poli. Tato možnost v některých algoritmech pěkně ukazuje, že např. prvky s malou hodnotou jsou postupně přidávány do výsledné posloupnosti a algoritmus s nimi dále již nepracuje.

5.3 Ovládání

Program dále umožňuje zastavovat běh algoritmu a znovu ho spustit od polohy, kde byl zastaven. Uživatel si tak může v jakékoli fázi program pozastavit a sledovat tak podrobněji změny v poli, pokud by mu nestačil nejpomalejší mód rychlosti řazení.

5.4 Přednastavení

Před spuštěním řazení se dá zvolit posloupnost, jaká bude řazena. V původním programu tato volba nebyla a jednalo se pouze o náhodnou posloupnost. V novém programu je možnost zvolit předdefinovanou posloupnost, která je vždy stejná a mohou se všechny algoritmy demonstrovat na stejné posloupnosti. To ukáže rozdíly mezi jednotlivými metodami. Další možností je opačně seřazená posloupnost. Zde se ukáže efektivita či neefektivita některých algoritmů. Na některé metody to zas na druhou stranu nemá vůbec vliv. Poslední možností je vygenerování úplně náhodné posloupnosti prvků. Pokud se nechá posloupnost seřadit a spustí se na ni nějaká metoda znovu, demonstruje to příklad řazení již seřazené posloupnosti. Je vidět, že některé algoritmy se snaží chvíli testovat a přeskládat prvky a některé metody poznají, že se jedná o seřazenou posloupnost a skončí.

6 Ovládání

Program je možné ovládat pomocí menu a tlačítek rychlé volby v horním pruhu okna.
V menu jsou následující položky: Soubor, Metoda, Spustit, Definuj pole, Zvuk, Nápověda.

Položka „Soubor“ obsahuje pouze volbu „Konec“, která ukonči program. Program je samozřejmě také možné ukončit obvyklým křížkem.

Položka „Metoda“ umožňuje výběr řadicí metody. Po vybrání jedné metody se její název vypíše v horním pruhu. Následně spuštěné řazení bude prováděno právě touto zvolenou metodou. Na výběr má uživatel z těchto metod: Bubble sort, Shell sort, Quick sort, Heap sort, Merge sort, Select sort, binární Insert sort a modifikovaná verze binárního Insert sortu.

Položka „Spustit“ obsahuje možnosti spustit řazení metodou, která je vybraná a nebo zastavit spuštěné řazení. Je zde také k dispozici volba pro zvýšení a snížení rychlosti řazení. Po upravení rychlosti se zobrazí procentuelní rychlost v horním pruhu okna.

Položka „Definuj pole“ má na výběr tři volby. První z nich je nastavení předdefinované posloupnosti do pole, které se bude řadit. Následující volbou je nahrání opačné posloupnosti do pole. Jako poslední je vygenerování náhodné posloupnosti.

Položka „Zvuk“ umožňuje výběr zvukového efektu, který provází řazení prvků. Zvuk je možné vypnout a tudíž nebude řazení provázeno žádným zvukovým efektem. Další volbou je zvuk „Podle hodnoty prvku“. Tato volba nastaví přehrávání zvukového efektu, který řídí svou výšku tónu podle hodnoty prvku v řazeném poli. Prvky s malou hodnotou reprezentuje hluboký tón, prvky s velkou hodnotou reprezentuje vysoký tón. Poslední volbou je zvuk „Podle indexu v poli“. Výška tónu se řídí podle indexu, na kterém se prvek v poli nachází. Prvky na nízkém indexu mají hluboký tón a prvky na vysokém indexu mají vysoký tón.

Položka „Nápověda“ umožňuje zobrazit krátký popis nazvaný „O programu“. Je zde velice stručně napsané o jaký program se jedná. Možnost „Nápověda“ zobrazí stručný popis programu, co je výstupem programu a na jakou oblast řadicích algoritmů se program nezaměřuje. Stručně je zde popsáno ovládání programu.

7 Srovnání

Při srovnání obou programů, jak starého napsaného v jazyce „Basic“, tak nového napsaného v jazyce „C“, jsou výsledky velice podobné. Čas měřený v obou programech u jednotlivých metod je přibližně stejný. Některé metody jsou ve starém programu rychlejší, ale to záleží na zapsání řadicího algoritmu a jeho případných modifikacích.

Vizuální efekt dosažený u nového programu je srovnatelný se starým programem. Jedná se především o to, jak se přeskládávají prvky. Nejsou dodrženy přesně barvy jednotlivých čar, reprezentujících prvky pole. Není však účelem vytvořit přesnou kopii, ale aby výsledný efekt byl stejný. Barvy tedy dodržují pravidlo, aby nebyly dva stejné odstíny vedle sebe.

Zvukový efekt nového programu je trochu odlišný. Původní přehrávání zvuků bylo u většiny algoritmů pouze jednou při záměně prvků. Zde se mi zdálo vhodnější a i po konzultaci s vedoucím bakalářské práce, jsem zvolil přehrávání zvuků při záměně prvků, pro oba zaměňované prvky. Pokud se zaměňují v poli dva prvky mezi sebou, tak je přehrán zvuk pro oba z nich. Je tedy slyšet zvuk obou zaměňovaných prvků.

8 Závěr a zhodnocení

Cílem celé práce bylo vytvoření programu, který je svým chováním podobný původnímu poskytnutému programu, který sloužil jako předloha. Program sloužící jako předloha byl napsán v programovacím jazyce Basic někdy v 80-tých letech minulého století. Dochoval se jen spouštěcí soubor. Napsáním nového programu se získal nejenom program demonstrující princip řadicích metod, ale získaly se i zdrojové kódy, které mohou sloužit při studiu programu. Může být také dále modifikován a mohou být přidávány další doplňky.

Program je implementován v programovacím jazyce C s použitím grafického prostředí WIN API. To umožňuje použití moderních programovacích technik a vytváření uživatelských prostředí, která jsou v dnešní době obvyklá.

Program má také oproti původnímu několik podstatných rozšíření. V první řadě se jedná o nový vzhled a ovládání, které je v dnešní době typické. V původním programu byla možnost ovládání pouze textová, tedy ovládání pomocí předdefinovaných klávesových zkratek. Nový program svou konstrukcí a použitím grafického rozhraní WIN API vytváří klasickou aplikaci pomocí oken a samozřejmě ovládání pomocí myši. Dalšími rozšířeními jsou nové řadicí algoritmy, které jsou svou funkcí zajímavé a ukazují mechanismy využívané pro řazení prvků. Jedním z nich je například tzv. „houpačkový mechanismus“. Dále se jedná o rozšíření zvukových efektů a možnosti definování pole hodnot před zahájením řazení. Program je také oproti původní implementaci možné zastavovat a znovu spouštět uprostřed běhu řadicí metody.

Nový program dosahuje podobných časů při řazení jednotlivými metodami jako starý program.

Nejvíce času jsem při praktické části této práce, tedy implementaci programu, věnoval výběru grafického prostředí. Časově náročné na tom bylo seznámit se s možnými grafickými prostředími a následná volba toho „nejvhodnějšího“. Zde jsem se často dostal do slepé uličky, kdy jsem při podrobnějším studiu některého grafického prostředí narazil na problém, který vylučoval použití některého prostředí pro tuto práci. Další fází byla pak příprava a návrh programu. Musel jsem vytvořit návrh a zkusit sestavit hlavní komponenty a i to někdy vedlo k tomu, že návrh nebyl vhodný a musel jsem začít s jiným postupem. Po odladění základní kostry programu jsem již mohl přidávat ostatní prvky.

V této práci jsem se naučil řešit komplexnější problémy od začátku úplného návrhu a vybírání vhodných prostředí, přes samotnou implementaci i testování, až po napsání konečné zprávy o průběhu práce.

Jsem si vědom toho, že na práci by se daly udělat vylepšení a modifikace, které by jeho funkčnost i efektivitu zlepšily. Zpětně po implementování programu a napsání tohoto textu mohu říci, že by se mohlo vyřešit lépe překreslování obrazovky. Další dobrou modifikací, kterou bych udělal, kdybych tuto práci měl vylepšit, je ukázka času jednotlivých řadicích metod. V tomto programu stejně jako ve starém programu v jazyce Basic není čas tou primární komponentou. Časový efekt jednotlivých metod je ovlivněn především časovým zpožděním z důvodu vizuálního efektu. Bylo by tedy vhodné zamyslet se nad možnými úpravami, jak docílit efektu, aby každá řadicí metoda řadila podle předpokládané časové náročnosti.

Poslední částí zadání bylo vytvoření otázek pro písemnou zkoušku. Při návrhu těchto otázek jsem kladl důraz především na jednoznačnost zadání. Vytvořené testové otázky jsou v příloze této práce.

Literatura

[1] Wróblewski, Piotr. *Algoritmy : Datové struktrury a programovací techniky*. Brno : Computer Press, 2004. 351 s.

[2] Honzík, Jan. *Algoritmy : Studijní opora*, 27.2.2007. Dostupný na URL <https://www.fit.vutbr.cz/study/courses/IAL/private/Opora/Opora-IAL-2007-02-RP-verze-3.pdf> (květen 2007). Řazení, s. 140-171.

[3] Honzík, Jan, Hruška, Tomáš, Máčel, Michal. *Vybrané kapitoly z Programovacích technik*. 3. upr. vyd. Brno : Nakladatelství Vysokého učení technického v Brně, 1991. 218 s. Dostupný na URL: <https://www.fit.vutbr.cz/study/courses/IAL/private/Texty/Skripta/ads.pdf> (květen 2007)

[4] Horowitz, Ellis, Sahni, Sartaj. *Fundamentals of Data Structures* [CD-ROM]. 1983. [cit. 2007-04-20]. Dostupný na CD-ROM Algorithms and Data Structures, Dr. Dobb´s.

[5] Korsh, James, Garrett, Leonard. *Data Structure, Algorithms and Program Style Using C* [CD-ROM]. 1988. [cit. 2007-04-02]. Dostupný na CD-ROM Algorithms and Data Structures, Dr. Dobb´s.

[6] Cormen, Thomas, Leiserson, Charles, Rivest, Roland. Introduction to Algorithms [CD-ROM]. 1990. [cit. 2007-04-25]. Dostupný na CD-ROM Algorithms and Data Structures, Dr. Dobb´s.

[7] Chalupa, Radek. *Učíme se Win API* [online]. Builder 2002 [cit. 2007-03-15]. Dostupný na URL: <http://www.builder.cz/art/cpp/winapi1.html>. (květen 2007)

[9] Skála, Kamil. *Seriál Win 32 API* [online]. Programujte 2002 [cit. 2007-03-15]. Dostupný na URL: <http://www.programujte.com/search.php?rsvelikost=sab&rstext=all-phpRS-all&rstema=129&menu5f3be53332d2d4cb4e78b1d815034f83=84:129>. (květen 2007)

[10] *MSDN* [online]. 2000 , 2007 [cit. 2006-12-05]. Dostupný na URL: <http://msdn.microsoft.com>. (květen 2007)

[11] *Wikipedie* [online]. 2002 , 2007 [cit. 2007-03-05]. Dostupný na URL: <http://cs.wikipedia.org>. (květen 2007)

[12] *Wikipedie* [online]. 2001 , 2007 [cit. 2007-04-05]. Dostupný na URL: <http://en.wikipedia.org>. (květen 2007)

Seznam příloh

Příloha 1. - Bubble sort

```
void buble_sort(HWND hWnd)
{//bulbe sort

    int i,pom,change;
    change = 1;

    while(change)
    {   // pokud nenastala zmena v razenem poli je pole serazene
        change = 0;
        for(i=0; i<MAX-1; i++)
        { // vnitrni cyklus ktery projde vsechny prvky a porovna je mezi sebou
            if (array[i]>array[i+1])
            {   // pokud je prvek s nizsim indexem vetsi
                // nez jeho pravy soused, tak se prohodi tyto 2 prvky
                pom = array[i];             // prohozeni prvku
                array[i] = array[i+1];
                array[i+1] = pom;
                change = 1;     // pormena zmeny se nastavi
                                // na 1 tzn. ze pole jeste neni serazeno
                    Wait(hWnd);         // zpozdeni kvuli vizualnimu efektu
                    sound(i, array[i]);   // zvuk ma vzdy zpozdeni SOUND_DELAY
                                          // i kdyz je zvuk vypnut
                    sound(i+1, array[i+1]);  // zvuk ma vzdy zpozdeni SOUND_DELAY
                                             // i kdyz je zvuk vypnut
            }
        }
    }
}
```

Příloha 2. - Quick sort

```
void quick_sort(HWND hWnd, int left_tail, int right_tail)
{
   int left, right;
   int pivot, tmp;
   left = left_tail;            // leva strana casti useku se kterym se prave pracuje
   right = right_tail;                  // prava strana casti useku se kterym se prave pracuje
   pivot = array[(left + right) / 2];
                                 // vyber pivota - prostredniho prvku zpracovavane casti
   do { // cyklus trva tak dlouho, dokud se leva prava cast nepotka
         while ((array[left] < pivot) && (left < right_tail))
                  left++;  // hledam zleva prvek ktery je vetsi jak pivot
         while ((pivot < array[right]) && (right > left_tail))
                  right--; // hledam zprava prvek ktery je mensi nez pivod
         if (left <= right) {  // pokud se leva a prava cast jeste neprekrizila tak
                                  // prvky ktere jsem nalezl prohodim
               tmp = array[left];
               array[left] = array[right];
               array[right] = tmp;
               left++;       // posunu casti pole o jedna smerem k sobe
               right--;
                     Wait(hWnd);  //zpozdeni kvuli vizualnimu efektu
                     sound(left, array[left]);           //zvuk ma vzdy zpozdeni SOUND_DELAY
                                                // i kdyz je zvuk vypnut
                     sound(right, array[right]);  //zvuk ma vzdy zpozdeni SOUND_DELAY
                                                // i kdyz je zvuk vypnut
      }
   } while (left <= right);



   if (left_tail < right)      // dokud se nedojde k puvodnimu levemu kraji
                               // pravym prvkem tak znovu volam quick rekurzivne
      quick_sort(hWnd, left_tail, right);       // razeni leve casti
   if (left < right_tail)      // dokud se nedojde k puvodnimu pravemu kraji
                               // levym prvkem tak znovu volam quick rekurzivne
      quick_sort(hWnd, left, right_tail);       // razeni prave casti
}
```

Příloha 3. - Shell sort

```
void shell_sort(HWND hWnd)
{
    int flag = 1, step = MAX, i, tmp;
    while( flag || (step>1))  // boolean priznak, cyklus pokracuje pokud doslo ke zmene
    {
       flag = 0;   // vynulovani priznaku na 0 pro detekci budouciho prohozeni prvku
       step = (step+1) / 2;          // nastaveni kroku pro nasledujici
                                     // bublinovy pruchod s krokem "step"
       for (i = 0; i < (MAX - step); i++)       // bublinovy pruchod s krokem "step"
      {
            if (array[i + step] < array[i])       // porovnani aktualni prvkua prkdu
                                                  // nasledujiciho - po kroku step
           {
                tmp = array[i + step];                  // prohozeni prvku
                array[i + step] = array[i];
                array[i] = tmp;
                flag = 1;   // identifikace ze probehlo prohozeni prvku
                     Wait(hWnd);  // zpozdeni kvuli vizualnimu efektu
                     sound(i, array[i]);       // zvuk ma vzdy zpozdeni SOUND_DELAY
                                               // i kdyz je zvuk vypnut
                     sound(i+step, array[i+step]);
           }
      }
    }
    return;
}
```

Příloha 4. - Heap sort

```
void heap_sort(HWND hWnd)
{
  int i, tmp;
  for (i = (MAX / 2)-1; i >= 0; i--)
  siftDown(hWnd,i,MAX);                    // prvotni ustanoveni hromady - heap
  for (i = MAX-1; i >= 1; i--)
  {         // postupne prochazeni pole - stromu , kde se vezme prvni(korenovy)
            // prvek stromu a zatridi se na sravne misto
        tmp = array[0];                    // prohozeni prvku
        array[0] = array[i];
        array[i] = tmp;
              Wait(hWnd);                  // zpozdeni kvuli vizualnimu efektu
              sound(0, array[0]);          // zvuk ma vzdy zpozdeni SOUND_DELAY
                                           // i kdyz je zvuk vypnut
              sound(i, array[i]);
        siftDown(hWnd,0, i-1);             // znovu ustaveni hromady - stromu - nejvetsi
                                           // prvek ze zbytku pole se musi dostat nahoru
  }
}

void siftDown(HWND hWnd, int root, int bottom)
{  // algoritmus znovuustaveni hromady - stromu
  int done, maxChild, tmp;
  done = 0;       // priznak ukonceni znovu ustavovani
  while ((root*2 <= bottom) && (!done))  // cyklus pokracuje pokud je potomek
                                         // roota jeste ve stromu (stale parti do stromu)
  {
    if (root*2 == bottom)  // pokud je potomek roota roven hornimu
                           // konci stromu - poslednimu prvku stromu
      maxChild = root * 2;// je poslednim prvkem take prvek nejvyssi,
                          // protoze je to levy syn a praveho syna root nema
    else if (array[root * 2] > array[root * 2 + 1])     // root ma oba syny- potomky
      maxChild = root * 2;                              // vetsi je levy syn - potomek
    else
      maxChild = root * 2 + 1;                                  // vetsi je pravy syn - potomek
    if (array[root] < array[maxChild])       // test zda je prvek root mensi nez jeho syn - potomek
    {
      tmp = array[root];                     // prohozeni roota a syna - potomka
      array[root] = array[maxChild];
      array[maxChild] = tmp;
            Wait(hWnd);                      // zpozdeni kvuli vizualnimu efektu
            sound(root, array[root]);        // zvuk ma vzdy zpozdeni SOUND_DELAY
                                             // i kdyz je zvuk vypnut
          sound(maxChild, array[maxChild]);
      root = maxChild;// rootem se stava prvek ktery byl vetsi - root se tedy posunul smerem dolu
    }
    else
      done = 1;              // hromada - strom je ustaveny
  }
}
```

Příloha 5. - Merge sort

```
void merge_sort_prep(HWND hWnd)
{// v totomto programu musi byt pro demosntracni ucely prispubeno pole.
// aby bylo mozne pracovat s dvojnasobne velkym polem, tak se puvodni pole zmensi
// na polovonu a razeni se bude demnstrovat s polovocnim poctem prvku
// cele puvodni pole bude potom vytvaret nove dvojnasobne pole pro algoritmus merge
    int n, i;
    n = MAX/2;                  // zmenseni polen a polovinu
    for (i=n; i<MAX; i++) array[i]=0;    // vynulovani druhe casti pole pro vetsi nazornost
    merge_sort(hWnd ,n);                 // zavolani samotneho algoritmu merge sort
}

void merge_sort(HWND hWnd, int n)
{
    int direct=1, m, i, j, k, p, h, end_l, end_m, end_r, tmp;
    // direct - smer kterym se pole presunuje - leva cilova, nebo prava cilova
    // i, j - hranice puvodniho pole
    // k, p - hranice druhe casti - rozsireni pole
    // h - priruste - kladny/zaporny podle toho ze ktere casti pole se presouva kam
    // end_x skoncilo se vlevo, vpravo, ve stredu
    do
    { // cyklus se provadi tak dlouho, dokud se presouva vice jak jedna posloupnost
        // nastaveni hodnot indexu v poli
        if (direct==1)
        { // leve pole je zdrojove, prave je cilove
            i=0; j=n-1; k=n; p=(2*n)-1;
        }else
        { // prave pole zdrojove a leve pole je cilove
            k=0; p=n-1; i=n; j=(2*n)-1;
        }
        m=0;    // pocitadlo posloupnosti vlozenych do ciloveho pole
        h=1;    // prvni prirustek bude kladny
        do
        {
            end_l = 0; end_r = 0; end_m = 0;    // nastaveni konce cyklu
            do
            {
                if (array[i] < array[j])
                {
                    array[k]=array[i];  // zkopirujeme mensi prvek do ciloveho pole
                    if (i==j)
                    {
                        end_m = 1;      // indexi se setkali, koncime cyklus
                    }else
                    {
                        i++;            // posun an dalsi prvek
                        k=k+h;          // posun na dalsi prvek
                        if (array[i] < array[i-1]) end_l = 1;// konci posloupnost zleva
                    }
                    Wait(hWnd);         // zpozdeni kvuli vizualnimu efektu
                    sound(i, array[i]); // zvuk ma vzdy zpozdeni SOUND_DELAY
                                        // i kdyz je zvuk vypnut
                    sound(k, array[k]);
                }else
```

```
                        {
                          array[k]=array[j];          // zkopirujeme mensi prvek do ciloveho pole
                          if (i==j)
                          {
                                  end_m = 1;          // indexi se setkali, koncime cyklus
                          }else
                          {
                                  j--;                // posun an dalsi prvek
                                  k=k+h;              // posun na dalsi prvek
                                  if (array[j] < array[j+1]) end_r = 1;  // konci posloupnost zprava
                          }

                          Wait(hWnd);                 // zpozdeni kvuli vizualnimu efektu
                          sound(j, array[j]);         // zvuk ma vzdy zpozdeni SOUND_DELAY
                                                      // i kdyz je zvuk vypnut
                          sound(k, array[k]);
                        }
                } while (end_l==0 && end_r==0 && end_m==0);

                tmp=k;  // do ciloveho pole se bude vkladat z druhe strany --> zamenim k a p
                k=p;
                p=tmp;
                m++;    // byla presunuta jedna posloupnost, tak ji prictu
                h=-h;   // bude se vkladat zas z druhe strany, proto se bude odecitat/pricitat
        }while (i != j);

        direct = !direct;

    }while (m!=1);

    if (!direct)
    {  // vysledna posloupnost je v prave casti dvojnasobneho pole
       // prekopiruje se serazene pole do leve - puvodni casti
        for (i=0; i<n; i++)
        {
                array[i] = array[i+n];
                Wait(hWnd);             // zpozdeni kvuli vizualnimu efektu
                sound(i, array[i]);     // zvuk ma vzdy zpozdeni SOUND_DELAY
                                        // i kdyz je zvuk vypnut
                sound(i+n, array[i+n]);
        }
    }

}
```

Příloha 6. - Select sort

```
void select_sort(HWND hWnd)
{
	int i, j, k, tmp;
	for (i=0; i<MAX-1; i++)
	{
		k = i;			// pozice k se nastavi na zacatek neserazene casti pole
		tmp = array[k];		// zapamatuju si prvni prvek neserazene casti pole
		for (j=i+1; j<MAX; j++)
		{	// hledam prvek ktery je nejmensi
			if (array[j] < tmp)
			{
				tmp = array[j];		// nejmensi si ulozim do tmp
				k = j;			// zapamatuju si index na kterem nejmensi lezi
				Wait(hWnd);		// zpozdeni kvuli vizualnimu efektu
				sound(j, array[j]);	// zvuk ma vzdy zpozdeni SOUND_DELAY
							// i kdyz je zvuk vypnut
			}
		}
		array[k] = array[i];  // prohodim prvni prvek neserazene casti s prvkem nejmensim
		array[i] = tmp;	// v pristim pruchodu bude neserazena cast opet o jeden prvek mensi

		Wait(hWnd);		// zpozdeni kvuli vizualnimu efektu
		sound(i, array[i]);	// zvuk ma vzdy zpozdeni SOUND_DELAY i kdyz je zvuk vypnut
		sound(k, array[k]);

	}
}
```

Příloha 7. - Bimary Insert sort

```
void binary_insert_sort(HWND hWnd)
{
    int i, j, left, right, middle, tmp;

    for (i=1; i<MAX; i++)
    {
        tmp = array[i];          // prvni prvek si ulozim pro budouci porovnavani
        left = 0;                // hranice neserazene casti
        right = i-1;
        while (left <= right)
        {                        // vyhledani mista v serazene cast, kam se ma zaclenit
                                 // prvek v promene pom se vyhleda binarne
            middle = (left + right) / 2;
            if (tmp < array[middle])
            {
                right = middle - 1;
            }else
            {
                left = middle + 1;
            }
        }
        for (j=i-1; j>=left; j--)
        {                        // musi se presunoout cel cast jiz serazeneho pole,
                                 // aby se uvolnila mezera pro prvek ktery tam patri.
            array[j+1] = array[j];

            Wait(hWnd);          // zpozdeni kvuli vizualnimu efektu
            sound(j, array[j]);  // zvuk ma vzdy zpozdeni SOUND_DELAY
                                 // i kdyz je zvuk vypnut
            sound(j+1, array[j+1]);
        }
        array[left] = tmp;       // na uvolnene misto se zacleni prvek

        Wait(hWnd);              // zpozdeni kvuli vizualnimu efektu
        sound(left, array[left]); // zvuk ma vzdy zpozdeni SOUND_DELAY i kdyz je zvuk vypnut

    }
}
```

Příloha 8. - Binary Insert sort - modifikovaný

```
void binary_insert_sort_m_prep(HWND hWnd)
{
    // v totomto programu musi byt pro demosntracni ucely prispubeno pole, aby bylo mozne
    // pracovat s dvojnasobne velkym polem, tak se puvodni pole zmensi na polovonu a razeni
    // se bude demnstrovat s polovocnim poctem prvku cele puvodni pole bude potom
    // vytvaret nove dvojnasobne pole pro modifikovany algoritmus binaryinsert sort.

    int n, i;

    n = MAX/2;          // zmenseni pole na polovinu

    for (i=n; i<MAX; i++)
    {
        array[i] = array[i-n];
        array[i-n] = 0;
    }

    binary_insert_sort_m(hWnd, n);
}

void binary_insert_sort_m(HWND hWnd, int n)
{   // modifikovana verxze insert sortu, ktera pracuje v dvojnasobnm poli
    // ale usetri presuny velky casti pole
    int i, j, left, right, middle, tmp, L, R, step;

    L = n;      // leva strana serazene casti
    R = n;      // prava strana serazene posloupnosti
                // na zacatku jsou obe na prvnim prvku pole, ktere radinme
    for (i=n+1; i<2*n; i++)
    {                           // vnejsi cyklus for, ktery prochazi neserazenou cast pole
        tmp = array[i];         // pomocny prvek na porovnavani
        left = L;               // leva zarazka pro binarni vyhledani
        right = R;              // prava zarazka pro binarni vyhledani
        while (left <= right)
        {                       // binarni vyhledani mista kam vlozit prvek tmp
            middle = (left + right) / 2;
            if (tmp < array[middle])
            {
                right = middle - 1;
            }else
            {
                left = middle + 1;
            }
        }
        middle = (L + R) / 2;   // index strudu serazene posloupnosti




        if (left <= middle)
        {                       // bude se posouvat smerem doleva, protoze prvek
```

```
                                        // tmp budeme davat blize k leve casti L
                L--;                    // rozsirim pole serazenych prvku doleva
                step = 1;               // krok je kladny
                j = L;                  // cyklus bude posupovat od leve strany
                left--;                 // místo, kam se zacleni prvekse musi posunout, protoze se
                                        // pocitalo pri binarnim vyhledani ze se bude posouvat v pravo
        }else
        {       // prvek se ma zaradit az do druhe poloviny serazene posloupnosti,
                // proto posuneme se serazenou casti do prava.
                R++;
                step = -1;
                j = R;
        }

        while (j != left)   // while cykli tak dlouho, dokud nedosahne mista,
                            // kam se ma zaradit novy prvek
        {       // posunuti pole, podle promene "step" se pricita nebo odecita
                // od zacatecni hodnoty "j", ktera byla nastavena
                // podele horni podminky bud na pravy nebo levy konec serazene posloupnosti.

                array[j] = array[j+step];

                Wait(hWnd);             // zpozdeni kvuli vizualnimu efektu
                sound(j, array[j]);     // zvuk ma vzdy zpozdeni SOUND_DELAY
                                        // i kdyz je zvuk vypnut
                sound(j+step, array[j+step]);


                j = j + step;
        }
        array[left] = tmp;

        Wait(hWnd);             // zpozdeni kvuli vizualnimu efektu
        sound(left, array[left]);  // zvuk ma vzdy zpozdeni SOUND_DELAY i kdyz je zvuk vypnut

    }
}
```

Příloha 9. - Otázky a příklady pro písemnou zkoušku z předmětu IAL

1.

```
void sort()
{
    int i, j, k, tmp;
    for (i=0; i<MAX-1; i++)
    {
        k = i;                      // pozice k se nastavi na zacatek neserazene casti pole
        tmp = array[k];             // zapamatuje se prvni prvek neserazene casti pole
        for (j=i+1; j<MAX; j++)
        {       // hleda se prvek ktery je nejmensi
                if (array[j] < tmp)
                {
                        tmp = array[j];         // nejmensi se ulozi do tmp
                        k = j;                  // zapamatuje se index, na kterem nejmensi lezi
                }
        }
        array[k] = array[i];  // zameni se prvni prvek neserazene casti s prvkem nejmensim
        array[i] = tmp;       // v pristim pruchodu bude neserazena cast opet o jeden prvek mensi
    }
}
```

Řadicí metoda pracuje na principu

a) vkládání

b) výběru

c) rozdělování

d) slučování

2. Metoda „rozděl a panuj“, kterou využívá Quick sort v rekurzivní formě, rozděluje řazené pole

a) na začátku běhu algoritmu a pak pracuje postupně s jednotlivými úsek.

b) v průběhu činnosti algoritmu, kdy se rekurzivní volání algoritmu provádí vždy s menší, tedy rozdělenou částí pole.

c) při rekurzivním volání, kdy se pole rozděluje vždy na poloviny a následující řazení probíhá tedy v polovičním poli než v předchozím běhu.

d) při rekurzivním volání algoritmu. Rozdělí se na tolik úseků, jaká je hodnota „pseudomediánu“.

3. Doplňte vhodný kus kódu do algoritmu, tak aby se jednalo funkční řadicí algoritmus.

```
void sort()
{
    int i,pom,change;
    change = 1;
    while(change)
    {
        ……..        // sem doplňte vynechaný kód
        for(i=0; i<MAX-1; i++)
        {
            if (array[i]>array[i+1])
            {
                pom = array[i];
                array[i] = array[i+1];
                array[i+1] = pom;
                change = 1;
            }
        }
    }
}
```

a) if (change = 1) break;

b) array[i] = pom;

c) change = 0;

d) change = change / 2;

4. Doplňte vhodný kus kódu do algoritmu, tak aby se jednalo funkční řadicí algoritmus.

```
void sort()
{
    int i, j, left, right, middle, tmp;

    for (i=1; i<MAX; i++)
    {
        ……..        // sem doplňte vynechaný kód
        while (left <= right)
        {
                middle = (left + right) / 2;
                if (tmp < array[middle])
                {
                        right = middle - 1;
                }else
                {
                        left = middle + 1;
                }
        }
        for (j=i-1; j>=left; j--)
        {
                array[j+1] = array[j];
        }
        array[left] = tmp;
    }
}
```

a) tmp = array[i];

left = 0;
right = i-1;

b) tmp = array[i];

left = i-1;
right = 0;

c) tmp = array[left]

left++;

d) tmp = array[right]

right++;

5. Jakou hodnotu bude mít proměnná „poc1“ po seřazení pole, tedy po skončení algoritmu. Řazené pole je ** 1 2 3 6 5 4 **

```
void sort()
{
    int i,pom,change,poc1,poc2;
    change = 1;
    poc1 = 0;
    poc2 = 0;
    while(change)
    {
        change = 0;
        for(i=0; i<MAX-1; i++)
        {
            poc1++;
            if (array[i]>array[i+1])
            {
                poc2++;
                pom = array[i];
                array[i] = array[i+1];
                array[i+1] = pom;
                change = 1;
            }
        }
    }
}
```

a) 10
b) 13
c) 15
d) 20
e) 25

6. Jakou hodnotu bude mít proměnná „poc2“ z předchozího algoritmu po seřazení pole, tedy po skončení algoritmu. Řazené pole je ** 1 2 3 6 5 4 **

a) 0
b) 3
c) 2
d) 1

7. Vlastnost „stabilita“ řadicí metody

a) respektuje pořadí položek se stejnými klíči.
b) rerespektuje pořadí položek se stejnými klíči.
c) zaručuje, že metoda svým chováním nezhroutí běh porgramu.
d) nevytváří v paměti neuvolněná místa – memory leak

8. Algoritmus Heap sort využívá pro svoji činnost vlastnosti

a) binárního stromu a využívá vlastnosti mezi otci a potomky.
b) binárního stromu a nevyužívá vlastnosti mezi otci a potomky.
c) metody „rozděl a panuj“ a vlastnosti hromady v paměti.
d) metody řazení bez přesunu položek.

9. 5adicí metada Shell sort je svou činností podobná metodě

a) Quick sort, kdy rozděluje pole na menší části, které se postupně řadí v rekurzivním volání.
b) Heap sort, ale při vytváření binárního stromu postupuje odzadu.
c) Bubble sort, ale využívá měnící se krok při procházení pole.
d) Insert sort, ale při posunu části pole (pro uvolnění místa prvku) využívá proměnlivý krok.

10. Algoritmus „List Merge sort“ – řazení pole setřiďováním seznamů –

a) se používá pouze na řazení seznamů.
b) v prvním kroku zřetězí neklesající posloupnosti s pomocí indexů v pomocném poli. Prvky v pomocném poli mají roli ukazatelů do řazeného pole.
c) vytvoří několik pomocných polí rozdělením hlavního pole podle „pseudomediánu“ metodou „rozděl a panuj“. Tyto pomocná pole se pak vloží do specializovaných seznamů.
d) jedná se o zavádějící název. Metoda neslouží k řazení.