

Department of Computer Science  
Faculty of Science  
Palacký University Olomouc

# BACHELOR THESIS

Visualization of Sorting Algorithms



2019

Mykhailo Klunko

Supervisor: Mgr. Tomáš Kühn,  
Ph.D.

Study field: Computer Science, full-  
time form

## **Bibliografické údaje**

Autor: Mykhailo Klunko  
Název práce: Vizualizace třídících algoritmů  
Typ práce: bakalářská práce  
Pracoviště: Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci  
Rok obhajoby: 2019  
Studijní obor: Informatika, prezenční forma  
Vedoucí práce: Mgr. Tomáš Kühn, Ph.D.  
Počet stran: 14  
Přílohy: 1 CD/DVD  
Jazyk práce: anglický

## **Bibliographic info**

Author: Mykhailo Klunko  
Title: Visualization of Sorting Algorithms  
Thesis type: bachelor thesis  
Department: Department of Computer Science, Faculty of Science, Palacký University Olomouc  
Year of defense: 2019  
Study field: Computer Science, full-time form  
Supervisor: Mgr. Tomáš Kühn, Ph.D.  
Page count: 14  
Supplements: 1 CD/DVD  
Thesis language: English

## Anotace

*Cílem práce bylo vytvořit software pro podporu výuky třídících algoritmů pomocí vizualizace průběhu třídění nejznámějšími algoritmy a jejich variantami. Program byl vytvořen s podporou názorné vizualizaci vybraných algoritmů na zadaném či vygenerovaném vstupním poli a krokování průběhu výpočtu se souběžným zobrazením pseudokódu použitého algoritmu a aktuálních hodnot použitých proměnných.*

## Synopsis

*The main goal of the thesis was to create a learning support software with visualization of the most known sorting algorithms and their variations. The application has to support a graphic visualization of selected algorithms on randomly generated or manually created array, step-by-step execution possibility, pseudocode and current state of variables.*

**Klíčová slova:** třídící algoritmus; třídění; vizualizace; program

**Keywords:** sorting algorithm; sorting; visualization; software

Děkuji, děkuji, děkuji.

*I hereby declare that I have completed this thesis including its appendices on my own and used solely the sources cited in the text and included in the bibliography list.*

date of thesis submission

author's signature

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Usage of sorting algorithms? . . . . .	1
1.2	Visual solution . . . . .	1
1.3	Přepínače . . . . .	1
1.4	Geometrie stránky . . . . .	1
<b>2</b>	<b>Algorithms</b>	<b>1</b>
2.1	Insertion Sort . . . . .	1
2.2	Selection Sort . . . . .	2
2.3	Bubble Sort . . . . .	2
2.4	Cocktail Sort . . . . .	3
2.5	Quick Sort . . . . .	3
2.6	Merge Sort . . . . .	4
2.7	Heap Sort . . . . .	5
2.8	Counting Sort . . . . .	7
2.9	Radix Sort . . . . .	7
2.10	Bucket Sort . . . . .	8
<b>3</b>	<b>Documentation</b>	<b>8</b>
<b>4</b>	<b>User Guide</b>	<b>8</b>
	<b>Conclusions</b>	<b>9</b>
	<b>Conclusions</b>	<b>10</b>
<b>A</b>	<b>První příloha</b>	<b>11</b>
<b>B</b>	<b>Druhá příloha</b>	<b>11</b>
<b>C</b>	<b>Obsah přiloženého CD/DVD</b>	<b>11</b>
	<b>Acronyms</b>	<b>13</b>
	<b>References</b>	<b>14</b>

## List of Figures

1	Insertion Sort at start: orange - sorted, blue - unsorted . . . . .	2
2	Quick Sort principle . . . . .	4
3	Merge Sort principle . . . . .	5
4	Max-Heap . . . . .	6
5	Children positions in heap . . . . .	6
6	Radix Sort example . . . . .	7
7	Radix Sort . . . . .	8

## List of Tables

## List of theorems

## List of source codes

# 1 Introduction

Nowadays sorting algorithms are widely used in software. For example if you open file explorer on your PC, you may see files sorted in different ways. Students of computer science start learning different algorithms in the first year of studies and sorting algorithms are among them.

The main goal of the thesis was to create a program which would serve as a tool for understanding how most known sorting algorithms work. Since I faced the problems of sorting during the course of algorithm design, now there is understanding that the visual representation is a vital part of studying process.

Text of the thesis describes principles of some basic sorting algorithms.

## 1.1 Usage of sorting algorithms?

## 1.2 Visual solution

## 1.3 Přepínače

Styl kidiplom je z hlediska uživatele zastoupen ekvivalentně nazvanou třídou, kterou je třeba volat na začátku dokumentu:

Následuje přehled přepínačů, je vždy uvedeno jméno přepínač, včetně výchozí hodnoty. Přepínače uvádí tabulka ??.

## 1.4 Geometrie stránky

Tento styl používá list velikosti A4. Pro sazbu prací je třeba použít jednostrannou sazbu. Levý okraj je rozšířen s ohledem na vazbu výsledné knižní podoby práce.

# 2 Algorithms

This section describes algorithms represented in the program.

## 2.1 Insertion Sort

Insertion sorting algorithm has a simple idea. Assume an array with items to be sorted. We divide the array into two parts: sorted one and unsorted one. At the beginning sorted part consists of the first element. Then, for the each next step we take first item from the unsorted part of the array and insert it into the right place among the sorted items.

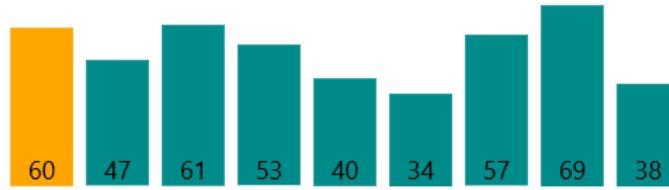


Figure 1: Insertion Sort at start: orange - sorted, blue - unsorted

In order to insert element into the right place in the sorted part, we compare the item from the unsorted part with each item from the sorted part in the direction from left to right. Comparing continues until smaller or equal element is found or no elements to compare left. After each comparison, if current item in the sorted part is greater, we move that current item one position right.

Complexity of Insertion Sort is  $\Theta(n^2)$ .

## 2.2 Selection Sort

Selection Sort algorithm is based on the repeated selection. As well as in the [Insertion Sort](#), here we consider using sorted part and unsorted part.

Assume an array of items to sort. At the beginning of the sorting process unsorted part is represented by the whole array. Then, the first item of the unsorted part is set as the smallest key and is compared with the follow-up elements. When smaller item is found, it is set as a new smallest key. After the end of the array is reached the smallest item is swapped with the first element of the unsorted part and it becomes the sorted part of the array. This step is repeated till the array is sorted.

Complexity of this sorting algorithm is  $\Theta(n^2)$ .

## 2.3 Bubble Sort

Bubble Sort is based on the idea of exchanging two adjacent elements if they have wrong order. Algorithm works stepping through the all elements from left to right so the largest elements tend to move or "bubble" to the right. That is why the algorithm is called Bubble Sort.

Now we are going to the details. Let us have an unsorted array. The algorithm does iterations through the unsorted part which is the whole array at the beginning. And with each iteration through the array the range of inspected items gets smaller by one till only two elements left. After this two elements are compared, the array is considered as sorted.

Bubble Sort complexity is  $\Theta(n^2)$ .



## 2.4 Cocktail Sort

Cocktail Sort or also known as Bidirectional Sort. This algorithm similarly to the Bubble Sort uses the idea of exchanging unordered adjacent items of array.

Assume array that needs to be sorted in ascending order. Above we described Bubble Sort and this algorithm has a significant problem. It iterates through array only in one direction. This way, smaller items which are closer to the end of array reach its right positions slowly.

Solution is to make Bubble Sort iterate left-to-right and right-to-left. Cocktail Sort uses two cycles inside a big one:

1. Iterate from  $a$  to  $b$ , compare adjacent elements and swap if they are not ordered.
2. Iterate from  $b$  to  $a + 1$  same way as in the step 1
3. Repeat steps 1. and 2. but with a bit different range from  $a = a + 1$  to  $b = b - 1$

## 2.5 Quick Sort

Quick Sort works on the principle "divide and conquer". It recursively applies itself on smaller parts of array until it is not sorted.

Algorithm takes one item at unsorted array or its part, usually it is the leftmost or the rightmost element of array. Then this item also known as pivot is moved to its final position in the array that it should occupy. While determining pivot's position, other elements of array are rearranged the way that no bigger elements are on the right and no smaller elements are on the left.

This way, it is enough to apply Quick Sort on each part of array not including pivot until array is not sorted.

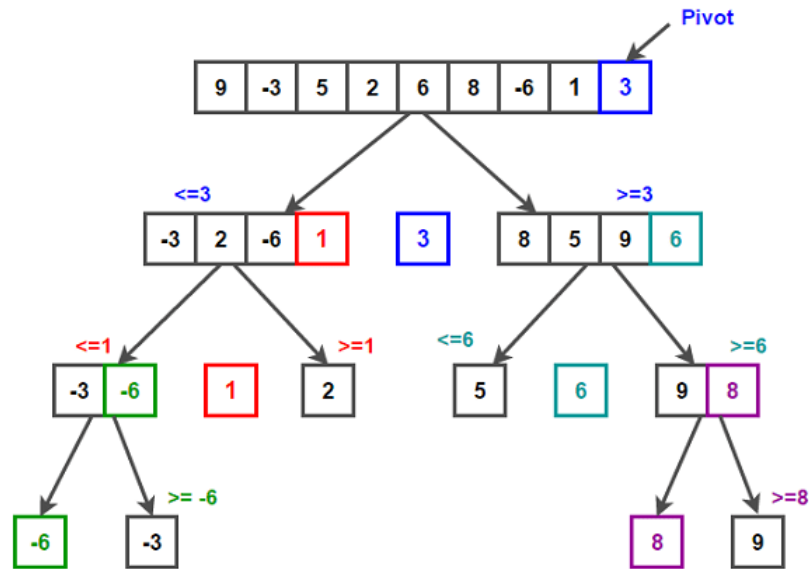


Figure 2: Quick Sort principle

There are several methods of partitioning of array into two parts, here I want to describe one that is demonstrated in the software part of this work.

Firstly, a pivot and index item are selected on the unsorted array or its part. Assume pivot is the last item and index is the first. Next each item of array except pivot is compared with the pivot. If current item is lesser or equal to pivot, it is swapped with the index item, next in order item becomes index. Finally, index and pivot are swapped and this way pivot is on its final position.

## 2.6 Merge Sort

Merge Sort as well as Quick Sort is an algorithm of type "divide and conquer". The logic of it is simple: recursively divide data into two parts till it becomes indivisible then merge parts back.

Merge procedure itself takes items from each previously divided part one by one, compares them and moves the smallest to the output, repeats previous step.

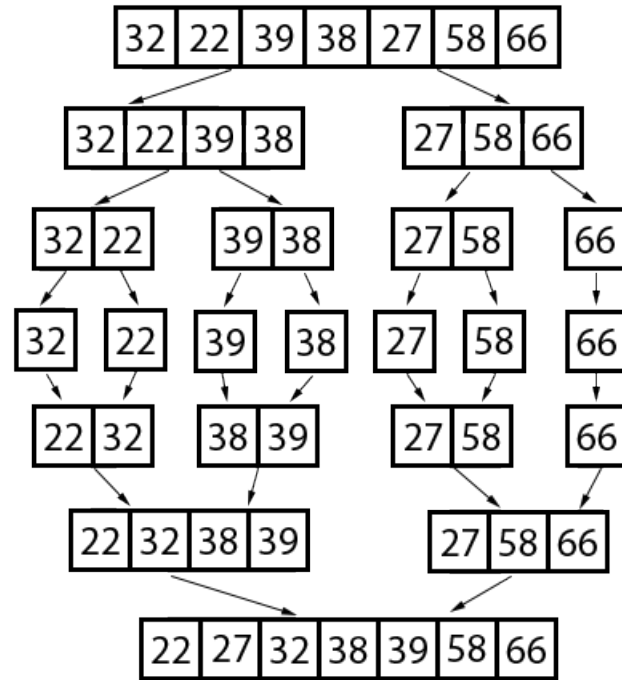


Figure 3: Merge Sort principle

## 2.7 Heap Sort

Heap Sort is a selection based algorithm and it offers another interesting approach to sorting. In comparison with the [Selection Sort](#) it has optimized selection by using binary heap data structure.

Binary heap is a complete binary tree; it means that all levels of tree, except the last one, must be completely filled with nodes. Also, this data structure satisfies the *heap condition*: each node key is greater than or equal to its child keys (this heap type is called *max-heap*).

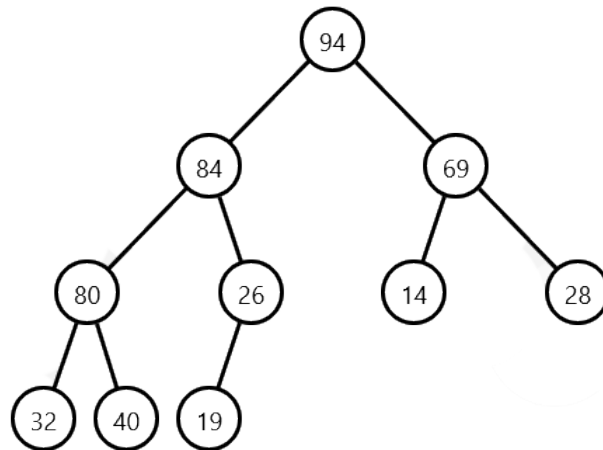


Figure 4: Max-Heap

Binary heap may be implemented by simple array. Item at position zero is a root node, items at position one and two are respectively left and right children of the root. From that representation it is easy to find children of each node (if they exist). Assume a node at position  $k$  then its left child is at  $2k + 1$  and its right child is at  $2k + 2$ .



Figure 5: Children positions in heap

**Heap Sort itself works as follows:**

1. Build max-heap
2. Swap root and the last node, reduce size of heap by one
3. Build max-heap without the node on a reduced position
4. Repeat steps 2 and 3 until the range of array is one

To build *max-heap* from current node we need to assure that right and left child comprise max-heaps. This way, in the first step procedure for building max-heap is recursively applied for each node that has at least one child from bottom to top.

After each swap of the root node and the node at last considered position last node takes its final place. This way it joins the sorted part of array.

Worst and average case complexity of Heap Sort are both  $\Theta(n \log(n))$ .

## 2.8 Counting Sort

Counting Sort is usually used for sorting keys in some range. Algorithm is based on counting keys of distinct values. Final positions of keys are calculated from the previous computations.

For better understanding let us have an example. Assume we have integers in range from 0 to  $k$  and empty output array. For this range we create helping array that will keep counters for each number from the range. Then all counters from the helping array are set to 0. To obtain final positions for items to be sorted we need to make further computations.

Firstly, for each element from the initial array we increase the respective counter which is determined from the element value. That means value of the element is its counter position in the helping array. Secondly, we need to sum each counter with previous. Finally, for each item from the initial array the respective counter is decreased by one and the value of counter now is the final position, we move item to the output array.

Worst case complexity for Counting Sort is  $\Theta(k + n)$ , where  $k$  is number of items in range and  $n$  is size of array to be sorted.

## 2.9 Radix Sort

The Radix sorting algorithm is based on the idea of using separate digits to determine final number positions. It works with the help of some stable sorting algorithm, it can be for example [Counting Sort](#), taking digits one by one from least significant to most significant.

For better understanding let us go to the details. Assume the array of integers. Let  $d$  be the number of digits of the largest item from the array. If there are numbers from the array that have less digits than the  $d$  is, then insufficient digits are counted as digits with zero values.

This way, for each digit from least significant to most we sort the input array according to the current digit. Here we use [Counting Sort](#) or another stable algorithm.

112	112	102	097
937	102	112	102
648	962	535	112
102	535	937	535
535	937	648	648
097	097	962	937
962	648	097	962

Figure 6: Radix Sort example

## 2.10 Bucket Sort

Bucket Sort as well as [Counting Sort](#) requires to know range of sorted data in advance. It is suitable when data is uniformly distributed over the range.

At the beginning algorithm creates  $n$  intervals of same size. For each interval there is created a dynamic structure which will held items from the input. This dynamic structures are called buckets.

Firstly, Bucket Sort distributes data to buckets. Then, there is need to sort each non-empty bucket. This task can be done by using, for example, [Insertion Sort](#). Finally, items from each bucket one by one are moved to the output array.

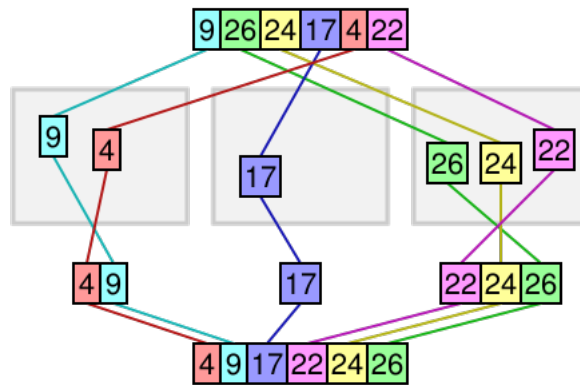


Figure 7: Radix Sort

## 3 Documentation

## 4 User Guide

## Conclusions

Závěr práce v “českém” jazyce.

## Conclusions

Thesis conclusions in “English”.



## A První příloha

Text první přílohy

## B Druhá příloha

Text druhé přílohy

## C Obsah přiloženého CD/DVD

Na samotném konci textu práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. jeho závazné adresářové struktury, důležitých souborů apod.

### **bin/**

Instalátor `INSTALATOR` programu, popř. program `PROGRAM`, spustitelné přímo z CD/DVD. / Kompletní adresářová struktura webové aplikace `WEBOVKA` (v ZIP archivu) pro zkopírování na webový server. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový běh instalátoru a programu z CD/DVD / pro bezproblémový provoz webové aplikace na webovém serveru.

### **doc/**

Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

### **src/**

Kompletní zdrojové texty programu `PROGRAM` / webové aplikace `WEBOVKA` se všemi potřebnými (příp. převzatými) zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu / adresářové struktury pro zkopírování na webový server.

### **readme.txt**

Instrukce pro instalaci a spuštění programu `PROGRAM`, včetně všech požadavků pro jeho bezproblémový provoz. / Instrukce pro nasazení webové aplikace `WEBOVKA` na webový server, včetně všech požadavků pro její bezproblémový provoz, a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Navíc CD/DVD obsahuje:

### **data/**

Ukázková a testovací data použitá v práci a pro potřeby testování práce při tvorbě posudků a obhajoby práce.

**install/**

Instalátory aplikací, runtime knihoven a jiných souborů potřebných pro provoz programu PROGRAM / webové aplikace WEBOVKA, které nejsou standardní součástí operačního systému určeného pro běh programu / provoz webové aplikace.

**literature/**

Vybrané položky bibliografie, příp. jiná užitečná literatura vztahující se k práci.

U veškerých cizích převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na CD/DVD, je uveden jejich zdroj (např. webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.



## References

- [1] KNUTH, Donald E. *The Art of Computer Programming, Volume 3, Sorting and Searching*. Second. 2005.
- [2] R., Sedgewik. *Algoritmy v C: základy, datové struktury, třídění, vyhledávání*. 2003.
- [3] Available from WWW: [en.wikipedia.org](http://en.wikipedia.org).