Gymnázium, Praha 6, Arabská 14

Ročníkový Projekt

Alexandr Belčenko

Řadící Algoritmy

Duben 2020

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne _____ Alexandr Belčenko _____

Anotace

Vizualizace Algoritmů je projekt, jehož účelem je poskytnout grafickou reprezentaci řazení jakéhokoliv pole obsahující čísla. Animace jsou tvořeny pomocí grafické knihovny JavaFX jazyku Java. Může být případně použit jako pomůcka sloužící k lepšímu a rychlejšímu učení se o řadících algoritmech. V projektu je celkem zastoupeno 6 algoritmů, 3 z nichž jsou lehčí a 3 těžší.

Vizualizace Algoritmů is a project, the goal of which is to provide a graphical representation of sorting any array that contains numbers. The Animation is created though the graphical library JavaFX from Java. I can perhaps be used as a learning tool for better and faster learning about sorting algorithms. There are in total 6 algorithms represented in this project, 3 of which are easier and 3 harder.

Zadání

Vytvořím pomůcku k porovnání algoritmů pro uspořádání pole. Ke každému algoritmu také udělám krátké vysvětlení, kde se s ním bude moci člověk seznámit.

Uživatel bude moci zadat vlastní pole, které pak bude setřízeno vybraným algoritmem. Také tam budou předvolby pro nejlehčí a nejtěžší situace pro daný algoritmus. Algoritmy budou doprovázeny animací pole.

Obsah

1. Úvod

Vizualizace Algoritmů je program schopný graficky znázornit proces třídění v poli. Používá k tomu různých animací a změn barev.

Tento projekt jsem si vybral hlavně proto, že všechny tyto řadící algoritmy i programovací jazyk jsme se učili v předešlých letech, tedy jsem měl dostatečné znalosti a stačilo se jen doučit JavaFX do dostatečné úrovně. Jedním z dalších důvodů bylo, že v době výběru jsem si nebyl jistý, zda bude povinné dávat práci do SOČ, tak jsem tedy radši použil téma, které by případně spadalo pod jednu z kategorií této soutěže.

Cílem tohoto projektu bylo vyrobit interaktivní pomůcku, která by pomáhala s učením algoritmů s pomocí animace. Následující dokumentace obsahuje i seznam všech těchto algoritmů včetně jejich výhod a nevýhod.

2. Nástroje

Při tvorbě projektu byly použity tyto nástroje:

Java je objektově orientovaný programovací jazyk, vyvinutý firmou Sun Microsystems v roce 1995, kterou poté koupil Oracle. Je to jeden z nejstarších jazyků, který je stále velmi široce užívaný. Do roku 2020 to byl nejpopulárnějším programovacím jazykem na světě[1], poté ho však předstihli Python a C. Mezi hlavní rysy Javy patří vzájemná nezávislost, což znamená, že třídy jsou navzájem co nejméně závislé, co ulehčuje práci. Další výhodou je její přenosnost. Java kód napsaný na jakémkoliv počítači může být spuštěn na jakémkoliv jiném počítači který má nainstalovaný interpreter.

Knihovna **JavaFX** je knihovna na bázi Javy, která slouží k tvorbě grafického uživatelského rozhraní. Je to následník zastaralé knihovny Swing, která sloužila k podobným účelům. Bohužel ale oficiálně zůstávají ve standardní edici Javy pouze Swing a AWT, přestože je měla JavaFX plně nahradit.

IntelliJ IDEA je vývojové prostředí (IDE) pro jazyk Java. Bylo vyrobeno českou společností JetBrains, která vlastní mnoho dalších vývojových prostředí pro jiné jazyky. Společnost byla založena roku 2000 v Praze, ale od té doby rozšířili své působení a mají kanceláře ve čtyřech dalších zemích.

CSS je stylistický jazyk používaný k popisu způsobu zobrazení elementů na internetových stránkách. Funguje však i v JavaFX. Byl navržen organizací W3C (World Wide Web Consortium), která obvykle vyvíjí webové standarty jako například HTTP nebo HTML. Je to v této době jediný takový jazyk, který nemá v podstatě žádnou alternativu, možná jenom o mnoho více komplikovanější XSL.

4 3 2 1

[1] TIOBE - The Software Quality Company, *TIOBE Index* [online], TIOBE software BV, Dostupné online

3. Jak funguje

V této kapitole popisuji obecné techniky použity v projektu.

Projekt se skládá z několika částí.

Main

Main je obvykle nejdůležitější třída, tady však ale slouží k inicializaci stage (jeviště), které v sobě obsahuje veškerý viditelný obsah

Controller

Controller je třída odpovídající za grafickou stránku aplikace a také všechna tlačítka, která spouštějí vybrané algoritmy. Obsahuje v sobě také proměnné označující velikost okna, počet prvků, mezery mezi prvky, atd., podle kterých se pak ovládá velikost jednotlivých prvků uvnitř.

Node

Node je zde základním stavebním blokem všech animací. Reprezentuje prvek v poli a má v sobě uloženou určitou hodnotu, podle které se pak řadí. Obsahuje také základní metodu potřebnou pro pohyb move, která vrací Transition k posunutí Node na pozici x.

Randomize

Randomize odpovídá za generaci náhodného pole o určité velikosti.

sortParent

SortParent je abstraktní rodič všech tříd označujících Algoritmy pro řazení pole. Obsahuje 3 metody.

- StartSort, což je abstraktní metoda. Potřebujeme ji, abychom mohli spustit řazení bez ohledu na vybraný Algoritmus.
- WorstCase, je další abstraktní metoda. V potomcích obsahuje pro daný rozměr a algoritmus nejkomplikovanější pole.
- Changeclr je metoda k zabarvení jednotlivých Node, používaná k indikaci operací.

- Swap je metoda, která používá metodu z Node move, k prohození dvou objektů v poli.

Algoritmy

V projektu je celkem 6 řadících algoritmů. Všechny fungují na podobném principu. Dostanou neseřazené pole a poté s využitím jejich konkrétního algoritmu jej setřídí. Každý obsahuje SequentialTransition transitions, což je typ animace, který přehraje uložené animace podle pořadí přidání. Metoda startsort pak slouží k spuštění jiných metod, které po dostání pole ho zpracovávají a pro každou viditelnou operaci přidávají novou animaci. Je použito celkem 4 typů animací:

- Už zmíněný SequentialTransition,
- ParallelTransition, která přehraje všechny uložené animace zároveň,
- TranslateTransition je animace použita k posunu prvků z jedné pozice do druhé,
- Filltransition je použita k zabarvení prvků pro ukázání procesu.

Když pak máme pole setříděné, pošleme finální SequentialTransition do Controlleru, který vyplní zaznamenané animace. Před dalším použitím stejného Algoritmu musíme animace vyprázdnit.

4. Algoritmy

Bubble sort

Bubble sort je jedním z nejlehčích řadících algoritmů. Jeho princip také není komplikovaný. Algoritmus opakovaně prochází seznam, přičemž porovnává každé dva sousedící prvky, a pokud nejsou ve správném pořadí, prohodí je. Pro většinu pokročilejších programů je tento způsob velmi neefektivní.

Dá se vylepšit několika způsoby. Můžeme po každému procházení polem jej zmenšit o jeden prvek, neboť víme, že poslední prvek je na správném místě. Dále můžeme počítat celkový počet prohození, čímž zjistíme, kdy je pole již seřazené.

Časová složitost:

- Nejlepší $O(n)$
- Nejhorší $O(n^2)$
- Průměrná $O(n^2)$

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat
        swapped := false
        for i := 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap(A[i-1], A[i])
                swapped := true
            end if
        end for
    until not swapped
end procedure
```

5

Selection sort

Selection sort, nebo také řazení výběrem, je také jeden z lehčích řadících algoritmů.

Algoritmus opakovaně prochází seznam, přičemž hledá vždy nejmenší prvek v seznamu, který následně prohodí s prvním možným neseřazeným prvkem. Na jedné straně seznamu nám tak roste řada seřazených prvků, do které dodáváme další, až máme nakonec seřazenou řadu.

Na rozdíl od Bubble sortu je tento algoritmus nestabilní, což znamená, že i když by byl seznam částečně seřazen, nebylo by zaručeno, že by se čas řazení zmenšil.

Tento algoritmus se používá hlavně u malých seznamů, kde až tak nezáleží na rychlosti řazení, protože tam není viditelný rozdíl. Selection sort má tedy výhodu lehčí implementace než jeho rychlejší soupeři.

Časová složitost:

- Nejlepší $O(n^2)$
- Nejhorší $O(n^2)$
- Průměrná $O(n^2)$

```
public void orderAsc(int vector[]) {
   int i, j, min, x;
   for (i = 0; i < vector.length-1; i++) {
      min=i;
      for (j = i+1; j < vector.length; j++)
         if (vector[j] < vector[min])
            min = j;

      x = vector[i];
      vector[i] = vector[min];
      vector[min] = x;
   }
}
```

6

Insertion sort

Insertion sort, neboli řazení vkládáním, je pravděpodobně nelepší algoritmus se složitosti $O(n^2)$.

Je stabilní, a stejně jako předešlé algoritmy lehký k implementaci. Jako Selection sort se používá k seřazení malých seznamů, ale je také efektivní k částečnému seřazení. Na rozdíl od předchozích je ale schopen řadit pole současně se vstupem.

Prochází seznam pouze jednou a postupně přidává prvky do seřazeného seznamu. Po každém kroku najde místo dalšímu prvku a přidá ho. Je tedy jedno, jestli v seznamu již byly prvky, nebo jestli je tam postupně přidáváme.

Časová složitost:

- Nejlepší $O(n)$
- Nejhorší $O(n^2)$
- Průměrná $O(n^2)$

```
i ← 1
while i < length(A)
    x ← A[i]
    j ← i - 1
    while j >= 0 and A[j] > x
        A[j+1] ← A[j]
        j ← j - 1
    end while
    A[j+1] ← x[3]
    i ← i + 1
end while
```

7

Merge sort

Merge sort, nebo také řazení slučováním, je první z více komplikovaných algoritmů. Vytvořil ho Americký matematik John von Neumann v roce 1945. Užívá se v něm rekurze, což znamená, že algoritmus volá sám sebe.

Funkce je nejčastěji rozdělena na dvě části: Metoda merge a samotný mergesort.

Merge je metoda, která potřebuje 2 pole, které pak následně sloučí do jednoho. Porovná vždy první prvek každého pole, a uloží jej do setříděného pole, poté zbývající první prvek s dalším prvkem druhého pole, a tak pokračuje dále, dokud v jednom z polí nedojdou prvky, načež z druhého pole doplní zbývající prvky.

Samotný mergesort obsahuje tři řádky kódu. Zaprvé zavolá sám sebe na nejdříve první, a pak druhou polovinu řady a zadruhé zavolá metodu merge, aby tyto 2 poloviny spojil. Za pomoci rekurze se pole rozděluje na 2 části tak dlouho, dokud nám nezůstanou jednoprvková pole, které pak postupně sloučíme zpět do jedné řady.

Výhoda Merge sortu je, že čas potřebný k řazení je téměř nezávislý na počátečním seřazení posloupnosti, je tedy dobrý k úkolům, kde potřebujeme pole seřadit vždy za určitý čas a bez výkyvů v čase. Hlavní nevýhoda Merge sortu je jeho velká spotřeba paměti. Merge sort na rozdíl od všech zde zmíněných algoritmů potřebuje navíc pole o stejné velikosti jako původní, což zabírá místo.

Časová složitost:

- Nejlepší $O(n \cdot \log(n))$
- Nejhorší $O(n \cdot \log(n))$
- Průměrná $O(n \cdot \log(n))$

```
function merge_sort(list m) is
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)
```

```
function merge(left, right) is
    var result := empty list

    while left is not empty and right is not empty do
        if first(left) ≤ first(right) then
            append first(left) to result
            left := rest(left)
        else
            append first(right) to result
            right := rest(right)

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
        append first(left) to result
        left := rest(left)
    while right is not empty do
        append first(right) to result
        right := rest(right)
    return result
```

8

Quick sort

Quick sort, neboli rychlé řazení, je jeden z nejrychlejších běžných algoritmů řazení založených na porovnávání prvků. Jeho tvůrcem je britský počítačový vědec Charles Antony Richard Hoare. Byl vytvořen v roce 1961.

Na rozdíl od Merge sortu není stabilní, což ho může zpomalovat, ale obecně je pro menší řady rychlejší. Jeho hlavní výhodou je malá spotřeba paměti, nepotřebuje žádné externí nástroje a všechny změny může provádět uvnitř pole, má však horší výkon při většímu počtu prvků.

Quick sort vybere nejdřív náhodný či nenáhodný prvek a zvolí jej jako takzvaný pivot. Poté přesune všechny prvky větší než pivot na jednu stranu a prvky menší než pivot na druhou. Nakonec rekurzivně zavolá sám sebe s parametry pro levou a pravou část (menší a větší pivotu), což zase vybere pivot v každé z částí a podle něj pole seřadí. Toto pokračuje, dokud není pole seřazeno.

Hlavní problematikou je zde volba pivotu, dá se volit jako vždy první nebo poslední prvek v poli, nebo se dá volit nějakým jiným procesem, ale čím komplikovanější ten proces je, tím více zpomaluje algoritmus, jehož hlavní výhodou je rychlost.

Dobře implementovaný Quick sort je na reálných datech dokazatelně rychlejší než Merge sort či Heap sort, má ale stálý risk, že narazí na nejhorší možný případ a zpomalí celý proces.

Časová složitost:

- Nejlepší $O(n \cdot \log(n))$
- Nejhorší $O(n^2)$
- Průměrná $O(n \cdot \log(n))$

Heap sort

Heap sort, či také řazení haldou, používá takzvané haldy, což je typ binárního stromu stromu, kde potomci jsou vždy menší či větší než rodiče. Heap sort byl vymyšlen Kanaďanem J. W. J. Williamsem v roce 1964, který také vymyslel koncept Haldy jako datové struktury.

Heap sort musí pro své fungování nejdříve vytvořit haldu, což ho velmi zpomaluje. Často se užívá pomocné metody heapify, která dostane index prvku a zvolí jeho potomky, načež zkontroluje, jestli je indexovaný prvek větší než jeho potomci, jinak vybere větší a vymění ho s indexovaným. Samotný heapsort pak pouze odebere první prvek, který je zaručeně nejmenší/největší prvek prvek v poli a na zbytek zase zavolá heapify aby se pole zase setřídilo.

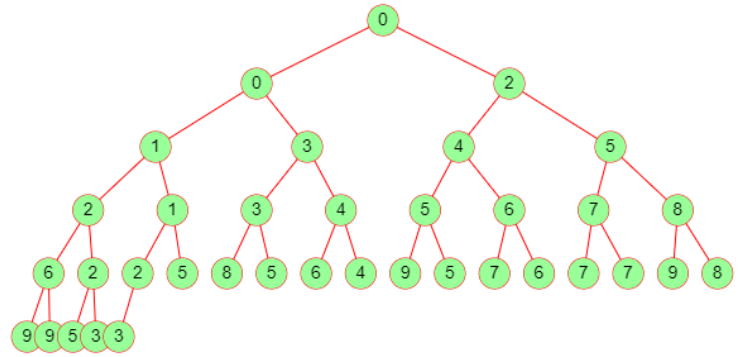
Je to nejpomalejší algoritmus z těchto 3. O mnoho pomalejší než dobrý Quick sort a o trochu pomalejší než Mergesort. Na rozdíl od nich má ale hodně výhod. Jako Merge sort má zaručenou časovou složitost, čímž se vyhýbá špatným případům, ale také se provádí celý v jednom poli, což hodně šetří na paměti.

Časová složitost:

- Nejlepší $O(n \cdot \log(n))$
- Nejhorší $O(n \cdot \log(n))$
- Průměrná $O(n \cdot \log(n))$

```
Heapsort(A) {
   BuildHeap(A)
   for i <- length(A) downto 2 {
      exchange A[1] <-> A[i]
      heapsize <- heapsize -1
      Heapify(A, 1)
}
```

```
BuildHeap(A) {
   heapsize <- length(A)
   for i <- floor( length/2 ) downto 1
      Heapify(A, i)
}
```



10

```
Heapify(A, i) {
   le <- left(i)
   ri <- right(i)
   if (le<=heapsize) and (A[le]>A[i])
      largest <- le
   else
      largest <- i
   if (ri<=heapsize) and (A[ri]>A[largest])
      largest <- ri
   if (largest != i) {
      exchange A[i] <-> A[largest]
      Heapify(A, largest)
   }
```

9

5. Design

K designu jsem použil CSS. JavaFX má speciální formát CSS, který je sice plnohodnotný, ale má jinou syntaxi, takže je těžké najít specifické příkazy.

Pro pozadí jsem vybral linear gradient dvou odstínů zelené a pro tlačítka jiný odstín. Zdálo se mi, že žlutá na zelené dost dobře vyniká, a také mám rád skoro všechny odstíny zelené.

Barva prvků není ovládaná CSS, ale kódem. Při generaci nového pole má každý prvek nastavenou narvu, která se dá později změnit. Je tomu tak, protože potřebuji v průběhu animací měnit barvu k indikaci nějaké činnosti. Ze začátku jsou všechny žluté a podle potřeby se můžou měnit na červené, oranžové a fialové.

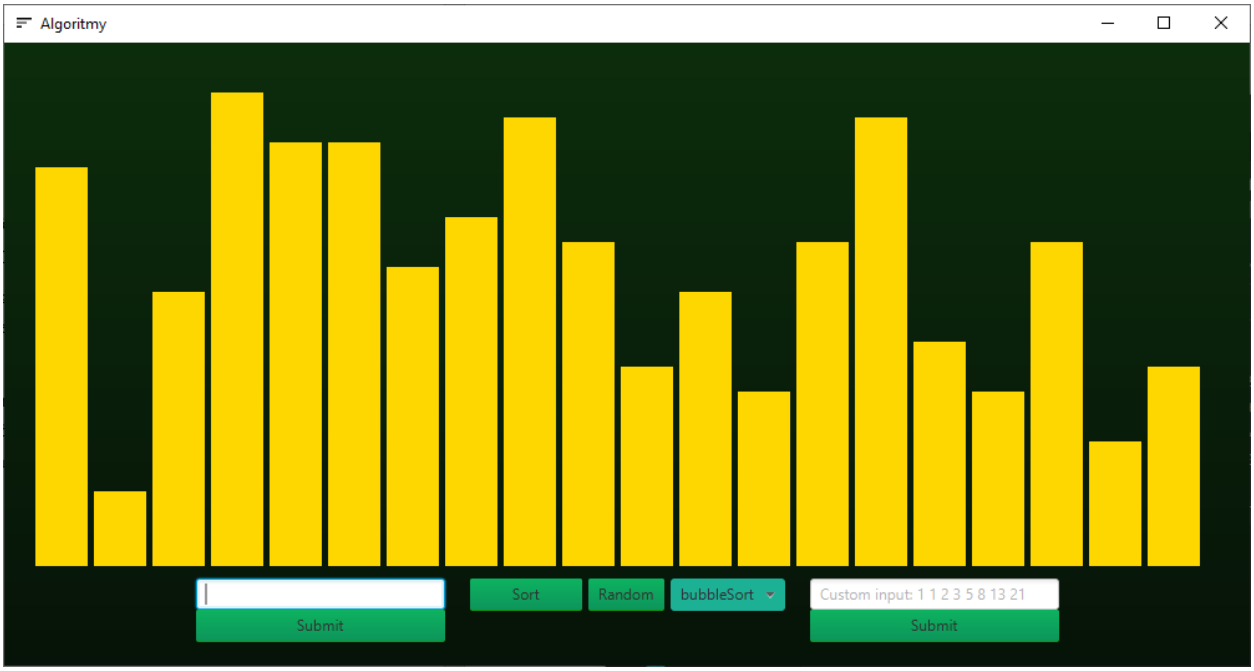
Ikonku, viditelnou v levém horním rohu, jsem stáhnul z velkého repositáře free to use ikonek na stránce material.io.

6. Návod

V této kapitole popíšu, jak používat aplikaci.

Pro spuštění aplikace stačí stáhnout a spustit .jar soubor, který se nachází v rootu github repozitáře.

Pokud by bylo potřeba otevřít aplikaci ve vývojovém prostředí stačí zkopírovat adresář sample, který se nachází v adresáři src.



11

Aplikace má velikost 1000x500, což se dá změnit pouze v kódu (třída Controller: a_window_width a window_height), všechny rozměry jsou derivovány relativně k těmto rozměrům. Také se dá změnit rychlost animací změnou movement_duration parametru v classu Node.

Uprostřed se nachází 2 tlačítka a výběrová lišta:

- Sort spustí libovolný právě vybraný algoritmus z výběrové lišty.
- Random vrátí náhodných 20 prvků.

Nalevo a napravo se nachází 2 textové vstupy:

- Levý vstup slouží k výběru počtu náhodných prvků.
- Pravý vstup slouží k určení vlastního pole, kde jde zadat řadu čísel oddělenými mezerou, která se pak ukazují na displayi.
- Tlačítko nejvíce vpravo slouží k nastavení nejtěžšího pole pro daný algoritmus, funguje pro všechny, kromě Heap sortu, který nejhorší případ nemá.

7. Budoucí plány

Jelikož tato práce zahrnuje pouze několik ze základních Algoritmů pro řazení, existuje určitě mnoho místa pro zlepšení a rozšíření.

Jiný typ algoritmů, které tento projekt neobsahuje jsou například Radix sorty, které fungují na základě procházení jednotlivých číslic hodnot v poli. Sice by bylo těžké nějak tento proces znázornit, mohlo by to být zajímavé porovnání mezi metodami

Jako další možné vylepšení by pak byla možnost vidět dva algoritmy vedle sebe, aby bylo možné porovnat jejich rychlost na jednotlivých polích.

Další věc, která se mi v této práci nepodařila, je dynamický resize okna, aby obsah měnil velikost podle aktuální velikosti okna.

A nakonec jako u většiny projektů se i zde jistě dá výrazně zlepšit rychlost, se kterou se provádějí operace. Sice na tom čím dál tím méně záleží, kvůli technologickému pokroku nejde často ani poznat rozdíl mezi účinnými a neúčinnými programy, ale vždy je lepší na to nespoléhat a snažit se dělat věci tak, aby nám naše počítače vydržely co nejdéle.

8. Závěr

V průběhu posledních několika měsíců jsem v JavaFX vytvořil aplikaci schopnou animace pole podle několika řadících algoritmů, které jsou popsány v dokumentaci. Uživatel má možnost zadat do aplikace vlastní pole a seřadit jej jakýmkoliv algoritmem. Aplikaci je možno použít jako pomůcku pro znázornění různých způsobů seřazení pole.

Při práci jsem pokročil ve svých programovacích dovednostech, hlavně v použití knihovny JavaFX, kterou budu pravděpodobně používat i nadále při práci na podobných projektech.

9. Seznam Obrázků

1	https://en.wikipedia.org/wiki/CSS#/media/File:CSS3_logo_and_wordmark.svg
2	https://en.wikipedia.org/wiki/IntelliJ_IDEA#/media/File:IntelliJ_IDEA_Icon.svg
3	https://en.wikipedia.org/wiki/JavaFX#/media/File:JavaFX_Logo.png
4	https://en.wikipedia.org/wiki/Java_(programming_language)#/media/File:Java_programming_language_logo.svg
5	https://en.wikipedia.org/wiki/Bubble_sort
6	https://en.wikipedia.org/wiki/Selection_sort
7	https://en.wikipedia.org/wiki/Insertion_sort
8	https://en.wikipedia.org/wiki/Merge_sort
9	https://www.cc.gatech.edu/classes/cs3158_98_fall/heapsort.html
10	https://miro.medium.com/max/1200/0*xW5bNnEmKXznSy4n.png
11	

10. Zdroje

Stack Overflow. *Stackoverflow.com* [online]. Dec 20, 2015 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/34381863/javafx-write-the-input-rows-of-a-textarea-to-array

Set BorderPane components size. *Stackoverflow.com* [online]. Aug 21, 2013 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/18441895/set-borderpane-components-size

How to add hint text in a Textfield in JavaFX. *Stackoverflow.com* [online]. Apr 10, 2018 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/34069030/how-to-add-hint-text-in-a-textfield-in-javafx

Adding a arraylist to a choicebox in my javafx program. *Stackoverflow.com* [online]. Dec 8, 2019 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/59238115/adding-a-arraylist-to-a-choicebox-in-my-javafx-program

Center an object in BorderPane. *Stackoverflow.com* [online]. Nov 18, 2015 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/33773179/center-an-object-in-borderpane

How to set placeholder in JavaFX? *Stackoverflow.com* [online]. Oct 28, 2017 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/23363222/how-to-set-placeholder-in-javafx

What is the recommended way to make a numeric TextField in JavaFX? *Stackoverflow.com* [online]. Oct 14, 2017 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/7555564/what-is-the-recommended-way-to-make-a-numeric-textfield-in-javafx

How to iterate through an ArrayList of Objects of ArrayList of Objects? *Stackoverflow.com* [online]. Jul 24, 2014 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/24943663/how-to-iterate-through-an-arraylist-of-objects-of-arraylist-of-objects

Meaning of regular expressions like - \\d , \\D, ^ , $ etc [duplicate]. *Stackoverflow.com* [online]. May 2, 2016 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/24943663/how-to-iterate-through-an-arraylist-of-objects-of-arraylist-of-object

Set Font globally in JavaFX. *Stackoverflow.com* [online]. Feb 3, 2014 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/18408884/set-font-globally-in-javafx

How to set radial-gradient with css in javafx. *Stackoverflow.com* [online]. May 4, 2016 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/37026738/how-to-set-radial-gradient-with-css-in-javafx

JavaFX Application Icon. *Stackoverflow.com* [online]. Aug 30, 2017 [cit. 2021-03-31]. Dostupné z: https://stackoverflow.com/questions/10121991/javafx-application-icon

Sequential Transition. *Tutorialspoint.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.tutorialspoint.com/javafx/javafx_sequential_parallel.htm

Working with code problems in IntelliJ IDEA. *Blog.jetbrains.com* [online]. [cit. 2021-03-31]. Dostupné z: https://blog.jetbrains.com/idea/2020/08/working-with-code-problems-in-intellij-idea/

Creating a TextField for integer only input when not using a data source. *Vaadin.com* [online]. [cit. 2021-03-31]. Dostupné z: https://vaadin.com/docs/v7/framework/articles/CreatingATextFieldForIntegerOnlyInputWhenNotUsingADataSource

EricCanull. FX Sort Animation Demo. *Github.com* [online]. Apr 27, 2018 [cit. 2021-03-31]. Dostupné z: https://github.com/EricCanull/fxsortinganimation

chriszq. Visual Sorting Algorithms. *Github.com* [online]. Jul 18, 2018 [cit. 2021-03-31]. Dostupné z: https://github.com/chriszq/VisualSortingAlgorithms

How to make Numeric | Decimal TextField in JavaFX Example Tutorial. *Tutorialsface.com* [online]. Dec 9, 2016 [cit. 2021-03-31]. Dostupné z: http://www.tutorialsface.com/2016/12/how-to-make-numeric-decimal-textfield-in-javafx-example-tutorial/

JavaFX Button. *Tutorials.jenkov.com* [online]. 12 Sep, 2020 [cit. 2021-03-31]. Dostupné z: http://tutorials.jenkov.com/javafx/button.html#:~:text=Button%20Size,-The%20JavaFX%20Button&text=The%20methods%20setMinWidth()%20and,width%2C%20JavaFX%20will%20do%20so.

Quick Sort. *Log2base2.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.log2base2.com/algorithms/sorting/quick-sort.html

Split() String method in Java with examples. *Geeksforgeeks.org* [online]. Dec 4, 2018 [cit. 2021-03-31]. Dostupné z: https://www.geeksforgeeks.org/split-string-java-examples/

JavaFX CSS Reference Guide. *Docs.oracle.com* [online]. [cit. 2021-03-31]. Dostupné z: https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html

Sort icon. *Material.io* [online]. [cit. 2021-03-31]. Dostupné z: https://material.io/resources/icons/?icon=sort&style=baseline

Compile and build applications with IntelliJ IDEA. *Jetbrains.com* [online]. Mar 8, 2021 [cit. 2021-03-31]. Dostupné z: https://www.jetbrains.com/help/idea/compiling-applications.html#run_packaged_jar

Insertion Sort. *Javatpoint.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.javatpoint.com/insertion-sort

Heap Sort. *Javatpoint.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.javatpoint.com/heap-sort

Merge Sort. *Javatpoint.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.javatpoint.com/merge-sort

Quick Sort. *Javatpoint.com* [online]. [cit. 2021-03-31]. Dostupné z: https://www.javatpoint.com/quick-sort