JavaFX GUI aplikacija za testiranje algoritama sortiranja

Tkalčec, Dominik

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:493771

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Stručni studij

JAVAFX GUI APLIKACIJA ZA TESTIRANJE ALGORITAMA SORTIRANJA

Završni rad

Dominik Tkalčec

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Osijek, 04.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu završnog rada na preddiplomskom stručnom studiju

Ime i prezime studenta:	Dominik Tkalčec
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4583, 20.09.2018.
OIB studenta:	67963110638
Mentor:	Robert Šojo
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Časlav Livada
Član Povjerenstva:	Izv. prof. dr. sc. Alfonzo Baumgartner
Naslov završnog rada:	JavaFX GUI aplikacija za testiranje algoritama sortiranja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Kreirati aplikaciju za testiranje različitih algoritama za sortiranje koja omogućava prikaz rezultata testiranja stupčastim i linijskim dijagramom. Potrebno je kreirati GUI koristeći JavaFX, kojim se korisniku omogućava lakša interakcija sa samom aplikacijom. Opisati tehnologiju koja se koristi za kreiranje GUI-a, proces kreiranja GUI-a, te potrebne korake koji se trebaju izvršiti za ispravno testiranje algoritama i prikaz rezultata.
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	04.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 20.09.2019.

Ime i prezime studenta:	Dominik Tkalčec
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI4583, 20.09.2018.
Ephorus podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: JavaFX GUI aplikacija za testiranje algoritama sortiranja**

izrađen pod vodstvom mentora Robert Šojo

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
1.1. Zadatak završnog rada 1
2. KORIŠTENE TEHNOLOGIJE 2
2.1. Eclipse IDE 2
2.2. JavaFX 3
2.3. Scene Builder 3
3. ODABRANI ALGORITMI SORTIRANJA 5
3.1. Selection sort 6
3.2. Bubble sort 8
3.3. Insertion sort 10
3.4. Quick sort 12
3.5. Merge sort 14
3.6. Heap sort 16
4. APLIKACIJA 19
4.1. Implementacija 19
4.1.1. UML dijagram 25
4.2. Rad s aplikacijom 26
4.2.1. Odabir algoritama sortiranja 26
4.2.2. Generiranje testnih uzoraka i testiranje algoritama sortiranja 26
4.2.3. Spremanje/učitavanje rezultata testiranja 30
4.3. Rezultati testiranja 34
5. ZAKLJUČAK 38
LITERATURA 39
SAŽETAK 40
ABSTRACT 41
ŽIVOTOPIS 42
PRILOZI 43

1. UVOD

Sortiranje je osnovni postupak obrade podataka ukoliko ih želimo učiniti što preglednijim i jednostavnijim za korištenjem te samim time osigurati jednostavnije pretraživanje. Zbog sve veće primjene računala za pohranu raznih vrsta podataka javlja se potreba za računalno sortiranje podataka. Time dolazimo do pojma „*algoritam*“ koji možemo definirati kao niz naredbi za efikasno rješavanje određenog problema. U današnje vrijeme postoji velik broj algoritama za sortiranje podataka što predstavlja izazov koji algoritam sortiranja odabrati, a da nudi najbrže sortiranje.

Tema ovog završnog rada je izraditi aplikaciju za testiranje algoritama koristeći Java programski jezik te *JavaFX* programski paket za izradu desktop aplikacija. Za testiranje algoritama za sortiranje izabrani su sljedeći algoritmi: *Selection sort*, *Bubble sort*, *Insertion sort, Quick sort, Merge sort* i *Heap sort*. Aplikacija je podijeljena na dva osnovna dijela, odabir algoritama sortiranja te generiranje testnih uzoraka i testiranje algoritama sortiranja nad testnim uzorcima. U prvom dijelu odabiremo algoritme sortiranja na kojima želimo izvršiti testiranje tj. usporediti koji algoritam radi bolje pod kojim uvjetima. U drugom dijelu osim što možemo podesit granice unutar kojih želimo generiranje slučajnih brojeva imamo i mogućnost dodjele, izmijene i brisanja generiranih uzoraka. Klikom na gumb „*Generiraj*“ generiraju se testni uzorci prema unesenim informacijama za generiranje testnih uzoraka. Klikom na gumb „*Testiraj*“ dobivaju se rezultati testiranja u obliku stupčastog i linijskog grafa.

Ovaj završni rad razrađen je u 3 cjeline. U prvoj cjelini pod nazivom „*Korištene tehnologije*“ ukratko su razrađene tehnologije korištene za izradu aplikacije. U drugoj „*Odabrani algoritmi sortiranja*“ ukratko su opisani i vizualno prikazani principi izvršavanja algoritama, a u trećoj, zadnjoj cjelini „*Aplikacija*“ opisana je ideja na temelju koje je razvijena aplikacija te su prikazane mogućnosti korištenja aplikacije. Osim toga posljednja cjelina prikazuje i uspoređuje rezultate dobivene testiranjem algoritama sortiranja u nekoliko slučajeva.

1.1. Zadatak završnog rada

Zadatak završnog rada je izraditi aplikaciju za testiranje algoritama sortiranja koja omogućuje prikaz rezultata testiranja linijskim i stupčastim grafom. Potrebno je izraditi GUI koristeći JavaFX, čime se korisniku omogućuje jednostavnija komunikacija s aplikacijom. Također je potrebno opisati tehnologiju koja se koriste za izradu GUI-a, proces izrade GUI-a, te korake koje je potrebno izvršiti kako bi se dobio prikaz rezultata testiranja.

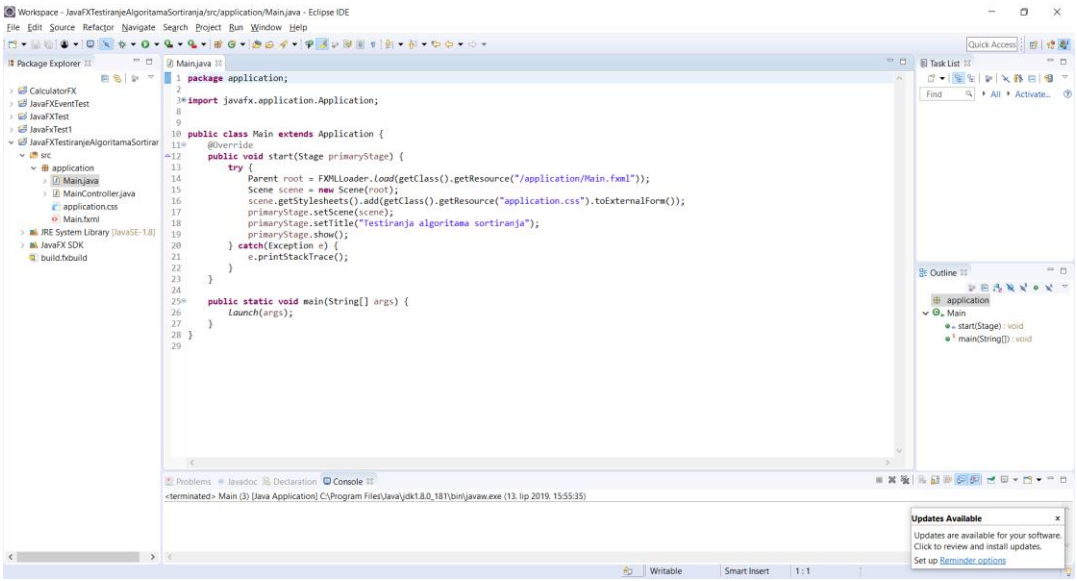
2. KORIŠTENE TEHNOLOGIJE

Aplikacija za testiranje algoritama sortiranja razvijena je koristeći *Eclipse* razvojno okruženje, Java programski jezik, *JavaFX* grafičko sučelje te *Scene Builder* programskog paketa.

2.1. Eclipse IDE

Eclipse je „*open-source*“ integrirano razvojno okruženje popularno za razvoj Java aplikacija, ali podržava i pakete za C/C++, PHP, JavaScript i Web programere te alate za modeliranje, izvješćivanje i testiranje.

Slika 2.1. prikazuje programsko okruženje *Eclipse* s aktivnom Java perspektivom koja je posvećena pisanju i uređivanju kôda te navigaciji kroz projekte. Perspektive predstavljaju skup prozora koji se koriste za neku svrhu. Lista na lijevoj strani prikazuje „*Package explorer*“ koji sadrži sve datoteke nekog projekta u trenutnom radnom prostoru. Radni prostor je mapa na računalu u koju se spremaju naši projekti. Najveći prostor u sredini se koristi za prikaz i uređivanje kôda, dok se na dnu prozora nalaze „*Javadoc*“ (Java dokumentacija), „*Problems*“ (prikaz mogućih grešaka ili problema) i „*Console*“ (ulazno-izlazna tabla) prikazi. Liste na desnoj strani prikazuju programerima kreirane metode, funkcije i varijable zajedno s mogućnosti kreiranja liste zadataka radi upravljanja projektom. Alatna traka sadrži prečace najkorištenijih funkcija. Osim Java perspektive postoji i *Debug* perspektiva koja omogućuje rad s pogreškama, njihovo ispravljanje te testiranje programa.



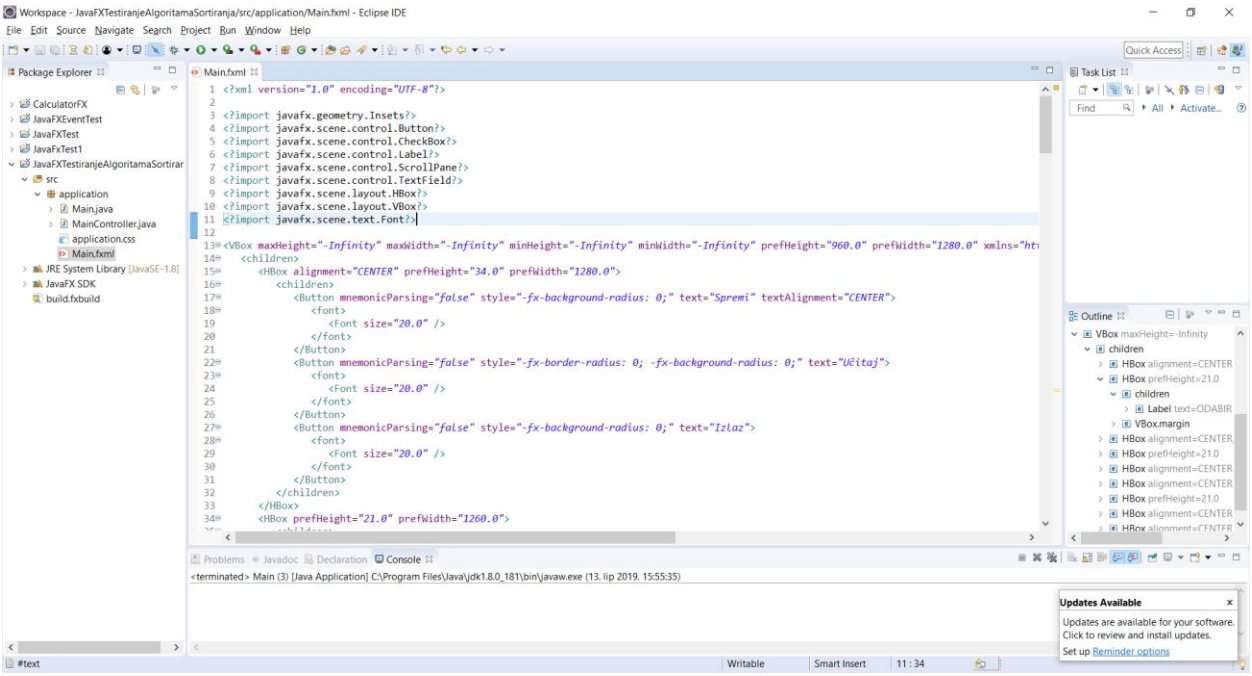
Slika 2.1. *Eclipse IDE programski paket za Java EE programere.*

2.2. JavaFX

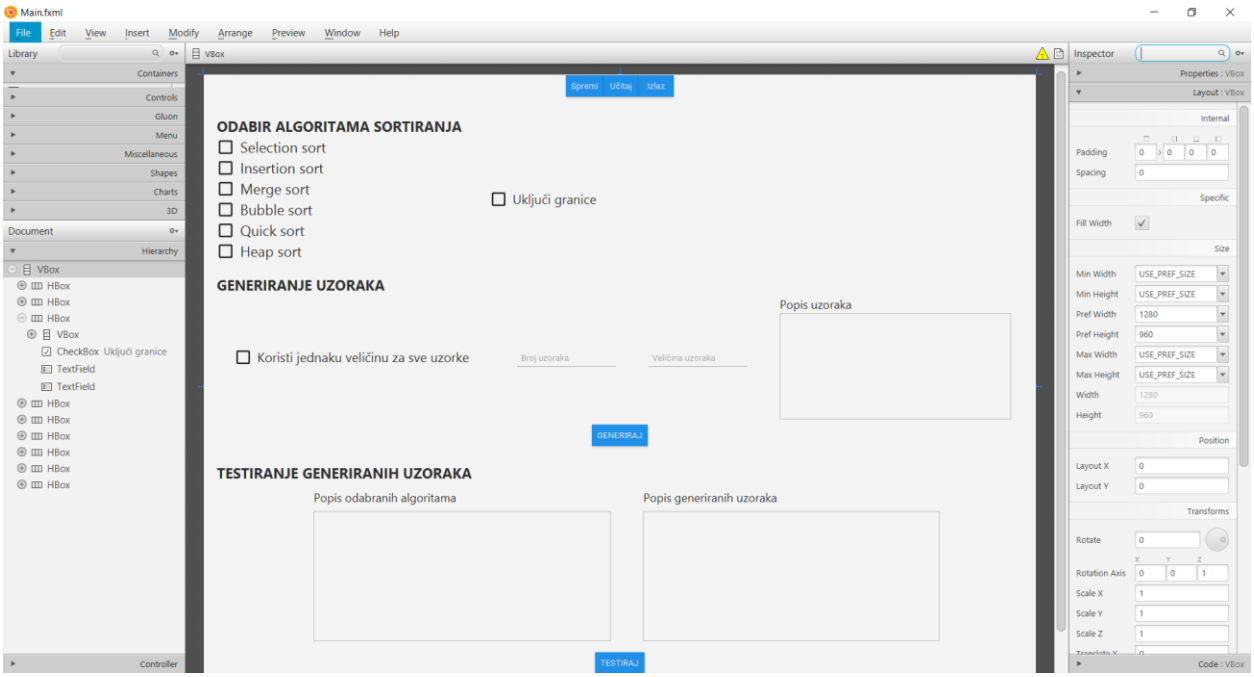
JavaFX sastoji se od medijskih i grafičkih paketa koji služe programerima za kreiranje, testiranje, dizajniranje, ispravljanje te razvijanje aplikacija koje se mogu pokretati na različitim platformama. Prema [1] dolazi kao zamjena za *Swing* GUI (engl. *Graphic User Interface*) te sadrži čišći i pregledniji kôd. GUI služi za interakciju između korisnika i aplikacije. Naglasak kod *JavaFX*-a je na odvajanju korisničkog sučelja (UI) od *backend* logike. UI (engl. *User Interface*) je podijeljen na Oracle-ov FXML skriptni jezik koji je baziran na XML-u, a koristi se za izradu dizajna *JavaFX* aplikacije te na CSS (engl. *Cascading Style Sheets*) stilski jezik koji se koristi za dodatno prilagođavanje izgleda komponenti kreiranih FXML jezikom. Za pisanje *backend* logike same aplikacije koristimo Java programski jezik. Java je najkorišteniji objektno orijentirani programski jezik koji pomoću JVM (engl. *Java Virtual Machine*) ima mogućnost pokretanja Java programa na bilo kojoj platformi. Korisničko sučelje možemo pisati ručno korištenjem FXML skriptnog jezika unutar .fxml datoteke ili pomoću *Scene Buildera*. Također korisničko sučelje možemo pisati i Java programskim jezikom, no takav način se ne preporuča budući da time gubimo na preglednosti i čitljivosti kôda te je time narušen integritet kôda.

2.3. Scene Builder

JavaFX Scene Builder je programski alat koji omogućuje jednostavno i brzo dizajniranje korisničkog sučelja aplikacije. Program u pozadini automatski generira FXML kôd pomicanjem grafički prikazanih UI komponenata na radni prostor. *Scene Builder* se prema [2] može koristiti zasebno kao samostalni alat za dizajniranje, ali također se može koristiti i zajedno s nekim Java IDE programskim paketom pri čemu se FXML kôd automatski generira u povezanu kontrolnu datoteku (Slika 2.2.) što omogućuje pokretanje te brzo povezivanje korisničkog sučelja s našom aplikacijom. Izgled *Scene Builder* programskog alata prikazan je na Slici 2.3. U ovom završnom radu *Scene Builder* je povezan s *Eclipse* IDE programskim paketom.



Slika 2.2. *Prikaz FXML kôda generiranog od strane Scene Buildera.*



Slika 2.3. *Prikaz korisničkog sučelja aplikacije u Scene Builderu.*

3. ODABRANI ALGORITMI SORTIRANJA

Postoji mnogo algoritama za sortiranje koji se razlikuju po nekim osobinama kao npr. broju operacija i vremenu potrebnom za izvršavanje. Za testiranje tih osobina koristit ćemo cjelobrojna polja radi jednostavnijeg prikaza rezultata. Cjelobrojna sortiranja spadaju u brojevna što znači da se mogu sortirati od najmanjeg prema najvećem (uzlazno) i od najvećeg prema najmanjem (silazno).

Prema [3] algoritmi sortiranja se mogu podijeliti na 4 vrste:

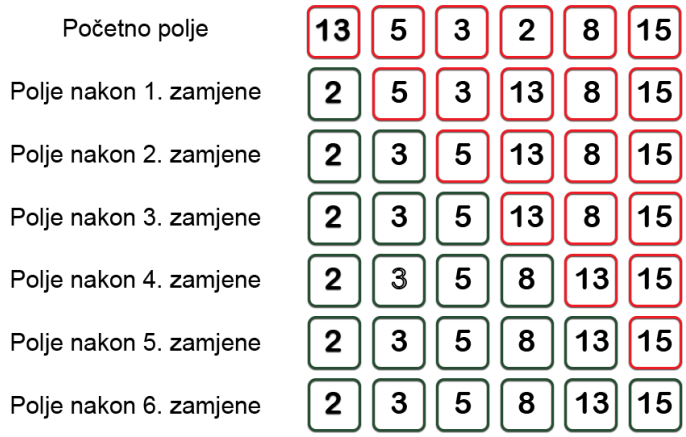
- Sortiranje zamjenom elemenata – sortiranje kod kojeg se pozicije mijenjaju ovisno o uvjetu usporedbe
- Rekurzivni algoritmi za sortiranje – algoritmi koji pozivaju sami sebe u svrhu sortiranja
- Sortiranje pomoću binarnih stabla – ubacuju elemente nesortiranog polja u stablo, a iz stabla ih vade u sortiranom redoslijedu
- Sortiranje umetanjem – uzima elemente iz nesortiranog dijela polja te ih ubacuje u sortirani dio

Algoritme sortiranja možemo analizirati prema složenosti. Složenost algoritama može biti memorijska (količina memorije potrebna da se algoritam izvrši) i vremenska (vrijeme potrebno da se algoritam izvrši). Vremenska složenost ovisi o broju ulaznih podataka (n), a definira se kao funkcija *T(n)*.

U ovom završnom radu za analiziranje složenosti algoritma koristit će se *O*-notacija koja daje složenost neovisnu o brzini računala već složenost ovisi o broju operacija.

3.1. Selection sort

Selection sort je primjer algoritma za sortiranje izborom najmanjeg elementa. Algoritam radi tako da prolazi kroz sve elemente polja za jedan puta manje od ukupnog broja elemenata u polju te svakim novim prolaskom sprema indeks početnog elementa. Uspoređuje vrijednost elementa na spremljenom indeksu sa svim preostalim elementima polja, te ukoliko je vrijednost s kojom uspoređuje manja od vrijednosti elementa na spremljenom indeksu sprema indeks toga elementa. Nakon usporedbe sa svim elementima dolazi do zamjene najmanje pronađene vrijednosti s vrijednosti elementa na spremljenom indeksu čime se dobije polje nakon 1. zamjene (Slika 3.1.). Postupak se ponavlja sve dok ne dođe do kraja polja.



Slika 3.1. *Postupak izvođenja Selection sort algoritma izborom najmanjeg elementa.*

Na Slici 3.1. prikazan je princip rada *Selection sort* algoritma objašnjenog u prethodnom odlomku. Crveno obrubljeni kvadratići predstavljaju elemente polja na kojima algoritam nastavlja daljnje sortiranje, dok zeleno obrubljeni kvadratići predstavljaju elemente polja na kojima algoritam više ne izvršava sortiranje.

```
public void selectionSort(int[] array){
    for (int i = 0; i < array.length - 1; i++) {
        int index = i;
        for (int j = i + 1; j < array.length; j++) {
            if (array[j] < array[index]) {
                index = j;
            }
        }
         int temp = array[index];
         array[index] = array[i];
         array[i] = temp;
    }
}
```

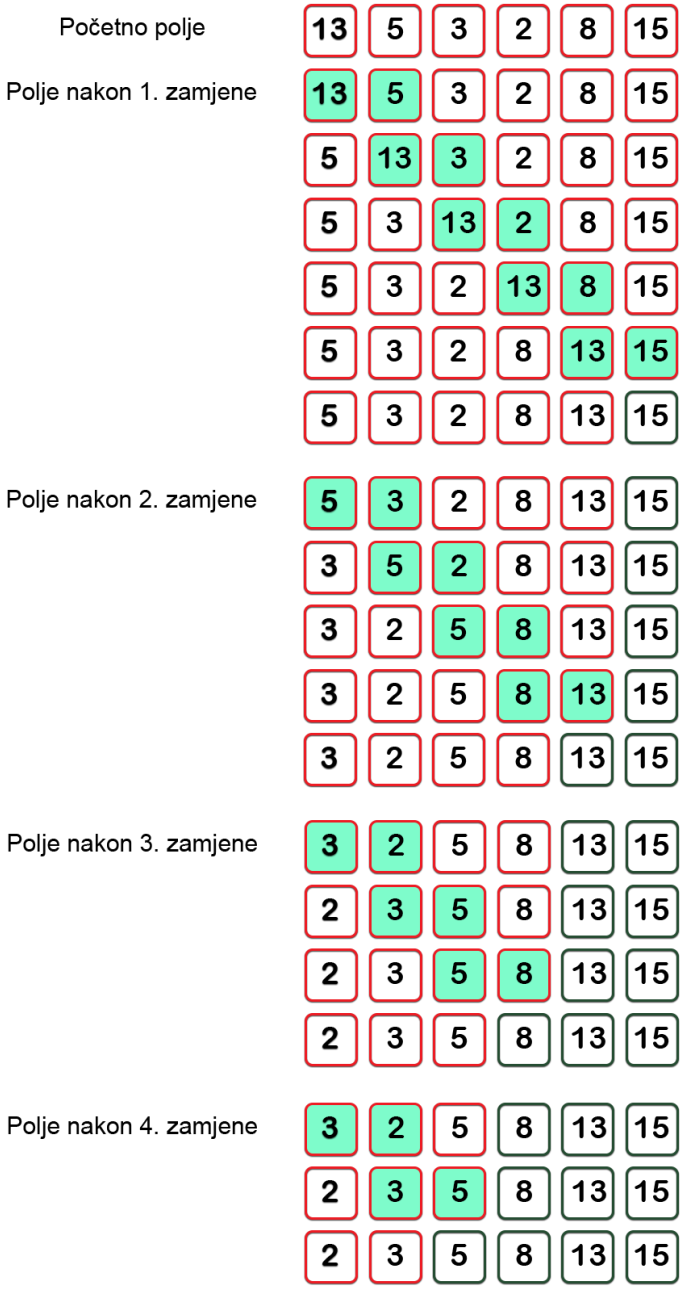
Slika 3.2. *Selection sort algoritam pisan u Java programskom jeziku prema [4].*

Metoda „*selectionSort()*“ prima samo cjelobrojno polje, a ne vraća ništa. Metoda „*length*“ vraća broj elemenata u *array* cjelobrojnom polju. Kod ovog algoritma vrši se i zamjena elemenata koji zadovoljavaju uvjet odnosno vrijednost prvog broja sprema na vrijednost drugog broja, a vrijednost drugog broja sprema na vrijednost prvog broja.

Vremenska složenost *Selection sort* algoritma je $O(n^2)$ budući da je uvijek potrebno n · n operacija za izvršavanje algoritma.

3.2. Bubble sort

Bubble sort je primjer algoritma za sortiranje zamjenom susjednih elemenata. Algoritam radi tako da prolazi poljem s lijeva na desno i uspoređuje dva susjedna elementa. Ukoliko je prethodni element veći od sljedećeg elementa vrši zamjenu njihovih vrijednosti. To radi sve dok sljedeći element ne dođe do kraja polja odnosno dok najveća vrijednost ne dođe na zadnji element polja. Nakon toga ponovo vrši usporedbu susjednih elemenata samo ovoga puta na polju bez zadnjeg elementa (bez elementa najveće vrijednosti). Postupak ponavlja sve dok u nekom prolazu ima bar jedne zamjene.



Slika 3.3. *Postupak izvođenja Bubble sort algoritma zamjenom susjednih elemenata.*

Na Slici 3.3. prikazan je princip rada *Bubble sort* algoritma objašnjenog u prethodnom odlomku. Kvadratići obrubljeni crvenom bojom označavaju elemente na kojima se vrši sortiranje, dok kvadratići obrubljeni zelenom bojom označavaju elemente koji sortirani te se na njima više ne provodi algoritam sortiranja. Zeleno osjenčani kvadratići označavaju susjedne elemente koji se međusobno uspoređuju.

```
public void bubbleSort(int array[]) {
      boolean swapped;
      for (int i = 0; i < array.length - 1; i++) {
          swapped = false;
          for (int j = 0; j < array.length - i - 1; j++) {
              if (array[j] > array[j + 1]) {
                  int temp = array[j];
                  array[j] = array[j + 1];
                  array[j + 1] = temp;
                  swapped = true;
              }
          }
          if (swapped == false)
              break;
      }
}
```

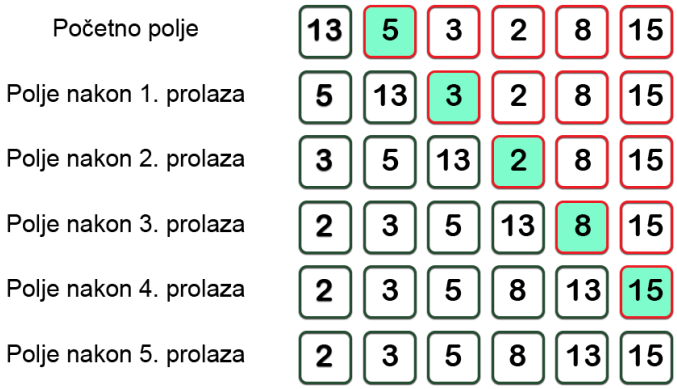
Slika 3.4. *Bubble sort algoritam pisan u Java programskom jeziku prema [4].*

Metoda *„bubbleSort()“* prima samo cjelobrojno polje, a ne vraća ništa. Također koristi metodu *length* koja vraća veličinu cjelobrojnog polja te se vrši zamjena elemenata koji zadovoljavaju uvjet. Ovaj *Bubble sort* algoritam se razlikuje od klasičnog po tome što koristi *boolean* varijablu koja ostaje *false* u slučaju da se u nekom koraku nije dogodila ni jedna zamjena odnosno da su svi elementi sortirani pri čemu se automatski završava izvođenje algoritma.

Vremenska složenost *Bubble sort* algoritma je $O(n^2)$ budući da je u najgorem slučaju prilikom svakog prolaska kroz polje potrebno zamijeniti pozicije susjednih elemenata.

3.3. Insertion sort

Insertion sort je primjer algoritma za sortiranje umetanjem elemenata. Algoritam prvim prolaskom kroz početno polje stavlja prvi element polja u sortirani dio polja, dok ostali dio polja stavlja u nesortirani dio. U nastavku algoritam pamti prvi element nesortiranog dijela polja te ga uspoređuje s elementima sortiranog dijela polja kako bi ga ubacio na pravo mjesto u sortirani dio. Ubacivanjem elemenata u sortirani dio brojevi u tome dijelu se slažu redoslijedom od najmanjeg prema najvećem. Algoritam u svakom sljedećem koraku povećava za jedan veličinu sortiranog dijela polja te smanjuje za jedan veličinu nesortirani dijela polja.



Slika 3.5. *Postupak izvođenja Insertion sort algoritma umetanjem elemenata.*

Na Slici 3.5. prikazan je princip rada *Selection sorta* objašnjenog u prethodnom odlomku. Elementi sortiranog dijela polja su na slici obrubljeni zelenom bojom, dok su elementi nesortiranog dijela polja obrubljeni crvenom bojom. Tirkizno plavom bojom su osjenčani oni elementi polja koje je u svakom koraku potrebno ubaciti u sortirani dio.

```java
void insertionSort(int array[])
{
    int n = array.length;
    for (int i = 1; i < n; ++i) {
        int key = array[i];
        int j = i - 1;
        while (j >= 0 && array[j] > key) {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
```

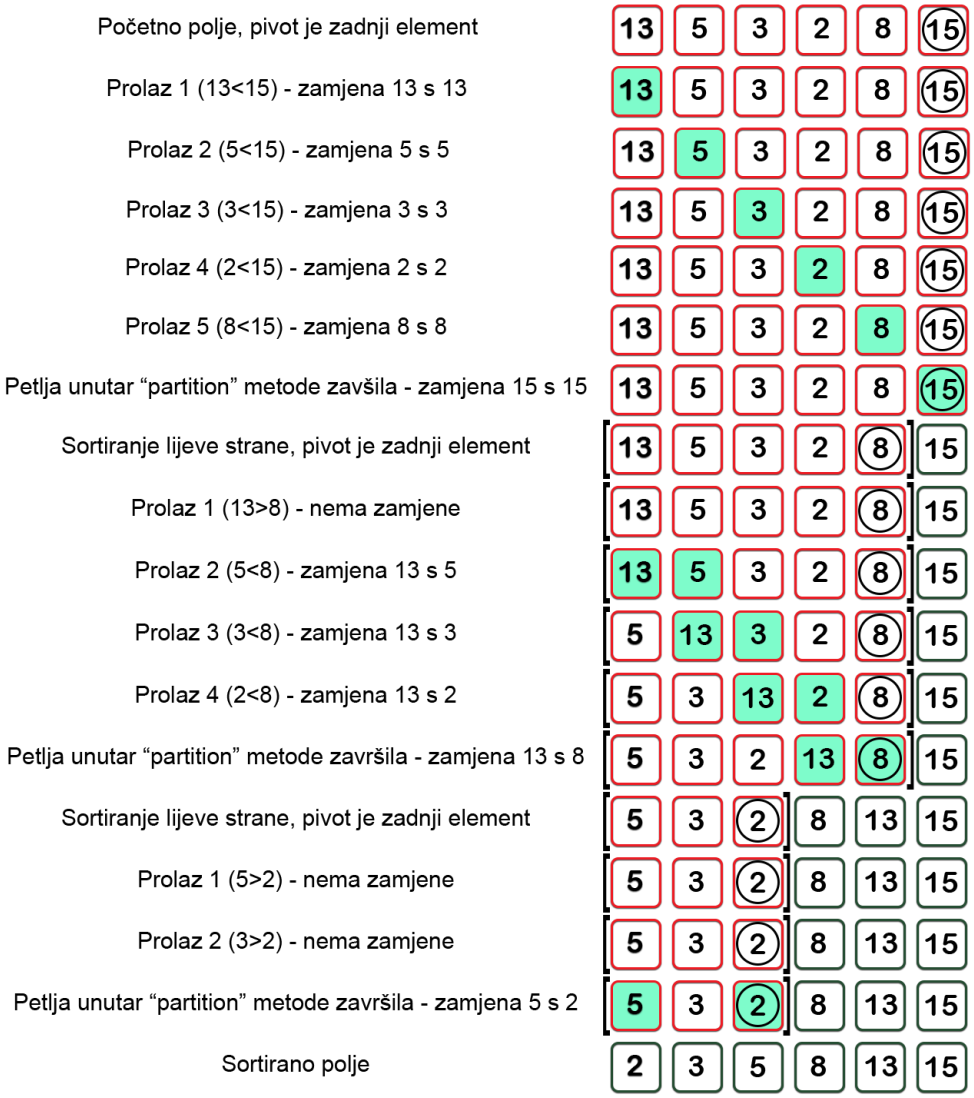
Slika 3.6. *Insertion sort algoritam pisan u Java programskom jeziku prema [4].*

Metoda „*insertionSort()*“ prima samo cjelobrojno polje, a ne vraća ništa. Također koristi metodu *length* koja vraća veličinu *array* cjelobrojnog polja. Petlja *for* izvršava se *n*-1 puta, a sadrži varijablu *key* koja predstavlja prvi element nesortiranog dijela polja te varijablu *j* koja predstavlja indeks posljednjeg element sortiranog dijela polja. Unutar for petlje nalazi se i *while* petlja koja prvi element iz nesortiranog dijela polja ubacuje u sortirani dio redoslijedom od najmanjeg prema najvećem.

Vremenska složenost *Insertion sort* algoritma je $O(n^2)$ budući da postoji n-elemenata koje je potrebno ubaciti na određenu poziciju u sortirani dio polja, te isto tako postoji i *n*-elemenata koje je potrebno proći u nesortiranom dijelu polja.

3.4. Quick sort

Quick sort je primjer algoritma za sortiranje korištenjem rekurzije. Algoritam se zasniva na metodi „*podijeli pa vladaj*“ te na odabranom elementu liste tj. pivotu koji je u ovom primjeru zadnji element polja. Nakon toga vrši se usporedba preostalih elemenata polja s pivotom na temelju koje se radi zamjena elemenata ovisno o poziciji te uvjetu unutar petlje. Po završetku izvođenja for petlje unutar metode *„partition()“* pivot se postavlja na novu poziciju unutar polja. Na temelju pozicije pivota polje dijeli na dva dijela: dio koji sadrži sve elemente lijevo od pivota i dio koji sadrži sve elemente desno od pivota. Cijeli postupak kao i dijeljenje polja ponavlja zasebno za lijevi i zasebno za desni dio sve dok ne dobije sortirano polje.



Slika 3.7. *Postupak izvođenja Quick sort algoritma korištenjem rekurzije.*

Na Slici 3.7. prikazan je princip rada *Quick sort* algoritma objašnjenog u prethodnom odlomku. Elementi sortirani primjenom algoritma obrubljeni su zelenom bojom, dok su elementi koje je potrebno sortirati obrubljeni crvenom bojom. Kvadratići osjenčani tirkizno plavom bojom označavaju elemente polja na kojima se vrši zamjena. Uglate zagrade prikazuju dijelove polja odnosno dio polja koji sadrži elemente koji se nalaze lijevo od pivota i dio polja koji sadrži elemente koji se nalaze desno od pivota.

```
public int partition(int array[], int low, int high) {
    int pivot = array[high];
    int i = (low-1);
    for (int j=low; j<high; j++) {
        if (array[j] <= pivot) {
            i++;

            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }

    int temp = array[i+1];
    array[i+1] = array[high];
    array[high] = temp;

    return i+1;
}

public void quickSort(int array[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(array, low, pi-1);
        quickSort(array, pi+1, high);
    }
}
```

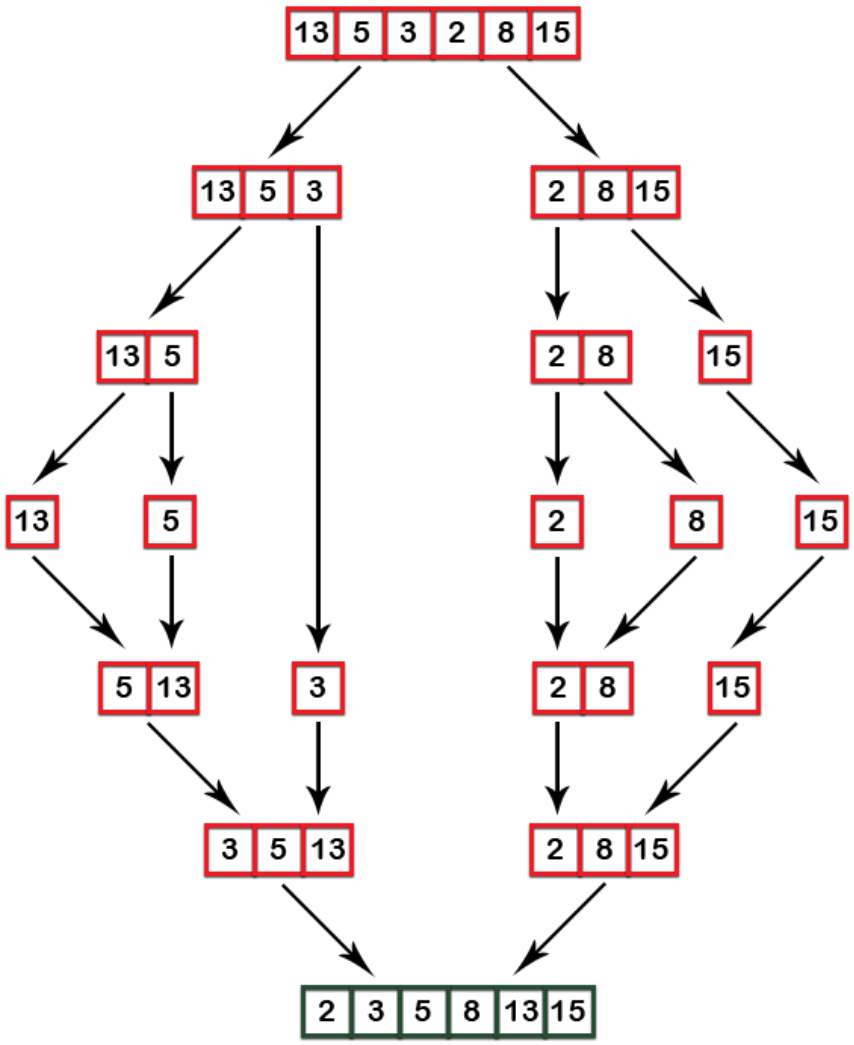
Slika 3.8. *Quick sort algoritam pisan u Java programskom jeziku prema [4].*

Metoda „*quickSort()*“ prima cjelobrojno polje, početni indeks te posljednji indeks. Ova metoda poziva „*partition()*“ metodu koja prima parametre kao i metoda „*quickSort()*“, a vraća cjelobrojnu vrijednost na temelju koje se vrši podjela polja na dio lijevo od pivota te dio desno od pivota.

Vremenska složenost *Quick sort* algoritma prema [4] je $O(n^2)$, a javlja se u najgorem slučaju ukoliko je polje već sortirano. Praktična primjena dokazuje da je *Quick sort* najbrži algoritam i da mu je prosječna vremenska složenost $O(n \cdot log n)$.

3.5. Merge sort

Merge sort je primjer algoritma za sortiranje korištenjem rekurzije. Ovaj algoritam se također zasniva na metodi „*podjeli pa vladaj*“ odnosno dijeli niz na približno dva jednaka podniza kao i *Quick sort*. Bitna razlika je u tome što *Merge sort* dijeli niz na podnizove sve dok ne dobije podniz od jednog elementa. Nakon što svaki podniz sadrži jedan element algoritam vrši spajanje tih elemenata određenim redoslijedom sve dok ne dobije konačni sortirani niz.



Slika 3.9. *Postupak izvođenja Merge sort algoritma korištenjem rekurzije.*

Na Slici 3.9. prikazan je princip rada *Merge sort* algoritma objašnjenog u prethodnom odlomku. Kvadratići sa zelenim obrubom prikazuju konačni sortirani niz, dok kvadratići s crvenim obrubom prikazuju nesortirani niz. Strelice označuju korake izvođenja algoritma.

```java
public void merge(int array[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 =  right - mid;

    int L[] = new int[n1], R[] = new int[n2];

    for (i = 0; i < n1; i++)
        L[i] = array[left + i];
    for (j = 0; j < n2; j++)
        R[j] = array[mid + 1+ j];

    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
             array[k] = L[i];
            i++;
        }
        else {
             array[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
       array[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
       array[k] = R[j];
        j++;
        k++;
    }
}

public void mergeSort(int array[], int left, int right)
{
    if (left < right) {

        int mid = left+(right-left)/2;

        mergeSort(array, left, mid);
        mergeSort(array, mid+1, right);

        merge(array, left, mid, right);
    }
}
```

Slika 3.10. *Merge sort algoritam pisan u Java programskom jeziku prema [4].*

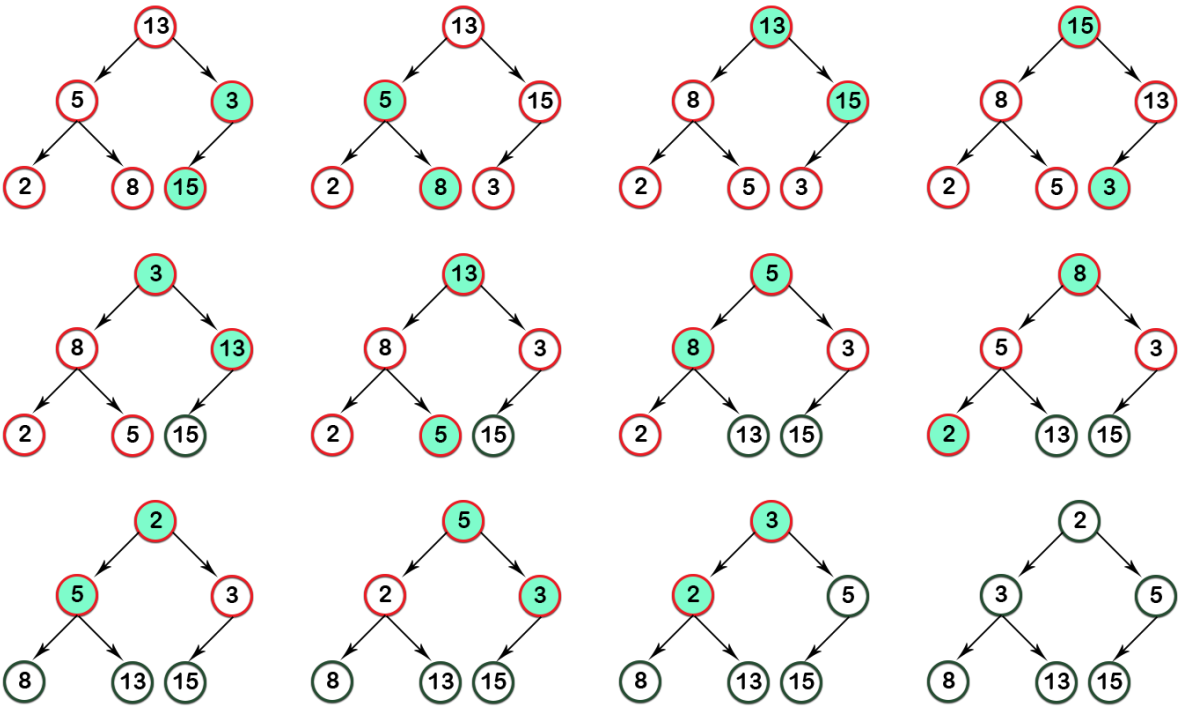
Metoda „*mergeSort()*“ rekurzivnim pozivanjem dijeli niz na podnizove sve dok ne dobije jedan element u svakom podnizu te poziva metodu „*merge()*“ koja vrši ponovo spajanje podnizova sortiranim redoslijedom u sortirani niz. Metoda „*mergeSort()*“ prima polje i raspon u kojem dijelu

niza se vrši sortiranje, dok metoda „*merge()*“ ima još i dodatan parametar *mid* koji predstavlja sredinu niza.

Vremenska složenost *Merge sort* algoritma prema [5] u najgorem slučaju je $O(n \cdot \log n)$ budući da ukupno ima $\log_2 n+1$ razina, a vremenska složenost za rekurzivne pozive unutar iste razine stabla je $O(n)$.

3.6. Heap sort

Heap sort je primjer algoritma za sortiranje pomoću hrpe. Postoje dvije vrste hrpa: *min* hrpa (koristi se za silazno sortiranje) i *max* hrpa (koristi se za uzlazno sortiranje). Algoritam radi tako da najprije ulazni cjelobrojni niz ubacuje u hrpu po razinama s lijeva na desno. Nakon toga vrši provjeru dali se najveći element čvorova koje uspoređujemo nalazi u korijenu te dali je svaki nadređeni čvor veći ili jednak čvorovima djece. Provjera se vrši od posljednjeg prema prvom čvoru. Kada algoritam dobije *max* hrpu (najveći element hrpe se nalazi u korijenu) vrši zamjenu korijena s posljednjim čvorom te se za taj čvor više ne vrši prethodno navedena provjera. Postupak se ponavlja sve ne ostane čvor s najmanjom vrijednosti.



Slika 3.11. *Postupak izvođenja Heap sort algoritma pomoću hrpe.*

Slika 3.11. prikazuje princip rada *Heap sort* algoritma objašnjenog u prethodnom odlomku. Kružići predstavljaju čvorove, a strelice povezuju nadređene čvorove s čvorovima djece. Kružići osjenčani tirkizno plavom bojom označuju čvorove u kojima dolazi do zamjene vrijednosti, dok kružići obrubljeni crvenom bojom označuju čvorove na kojima se vrši provjera dali se najveći element čvorova koje uspoređujemo nalazi u korijenu te dali je svaki nadređeni čvor veći ili jednak čvorovima djece. Postoje još i kružići obrubljeni zelenom bojom oni označuju čvorove hrpe na kojima se više ne vrši prethodno navedena provjera budući da su oni sortirani.

```
public void heapSort(int array[]) {
    int n = array.length;

    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(array, n, i);

    for (int i=n-1; i>=0; i--) {
        int temp = array[0];
        array[0] = array[i];
        array[i] = temp;

        heapify(array, i, 0);
    }
}
public void heapify(int array[], int n, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && array[l] > array[largest])
        largest = l;

    if (r < n && array[r] > array[largest])
        largest = r;

    if (largest != i) {
        int swap = array[i];
        array[i] = array[largest];
        array[largest] = swap;

        heapify(array, n, largest);
    }
}
```

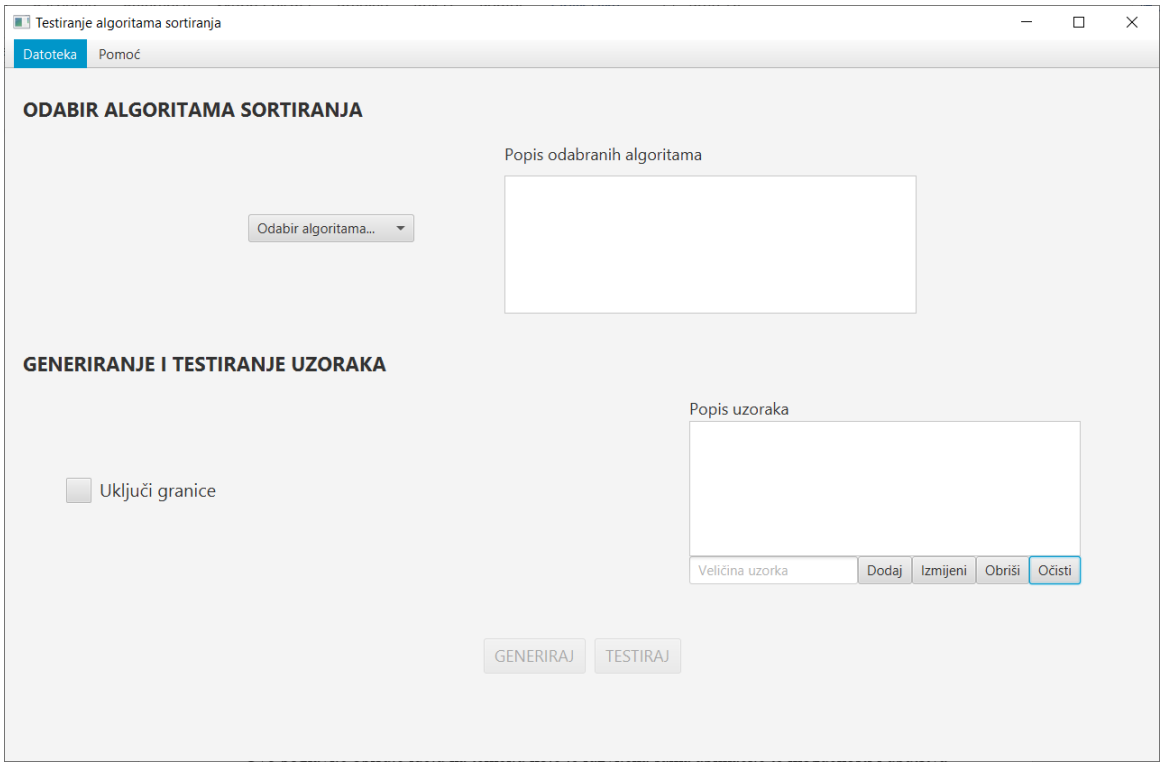
Slika 3.12. *Heap sort algoritam pisan u Java programskom jeziku prema [4].*

Metoda „*heapSort()“* prima početni niz te ga ubacuje u hrpu s lijeva na desno po razinama, a osim toga poziva još i metodu „*heapify()“* koja od postojeće hrpe radi *max* hrpu. Metoda „*heapify()“* prima polje, indeks do kojeg se vrši sortiranje te indeks korijena.

Vremenska složenost *Heap sort* algoritma prema [5] u najgorem slučaju je $O(n \cdot \log n)$ budući da je za svako ubacivanje u hrpu odnosno izbacivanje iz hrpe potrebna vremenska složenost od $O(\log n)$.

4. APLIKACIJA

JavaFX GUI aplikacija za testiranje algoritama sortiranja omogućava odabir željenih algoritama za koje se izvršava testiranje, dodavanje i brisanje testnih uzoraka, određivanje i izmjenjivanje veličine pojedinog uzorka, određivanje donje i gornje granice generiranja pseudo-slučajnih brojeva, generiranje testnih uzoraka pseudo-slučajnim cjelobrojnim brojevima te vizualni prikaz rezultata testiranja koristeći linijski i stupčasti graf. Osim toga aplikacija ima i mogućnost spremanje rezultata testiranja u JSON datoteku kao i učitavanje prethodno spremljenih rezultata iz JSON datoteke. Početni prozor nakon pokretanja aplikacije prikazan je na Slici 4.1.



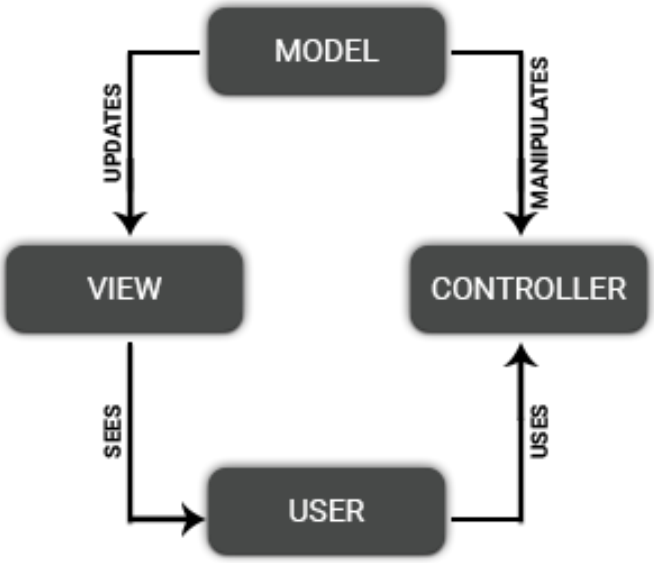
Slika 4.1. *Prikaz početnog prozora aplikacije.*

4.1. Implementacija

Arhitektura aplikacije organizirana je na MVC (engl. *Model-View-Controller*) principu podjele koji dijeli aplikaciju na tri osnovna dijela:

- *Model* – sadrži klase koje predstavljaju biznis logiku
- *View* – sadrži datoteke vezane samo za izgled korisničkog sučelja aplikacije (.fxml, .css)
- *Controller* – sadrži klase za povezivanje korisničkog sučelja s modelom podataka

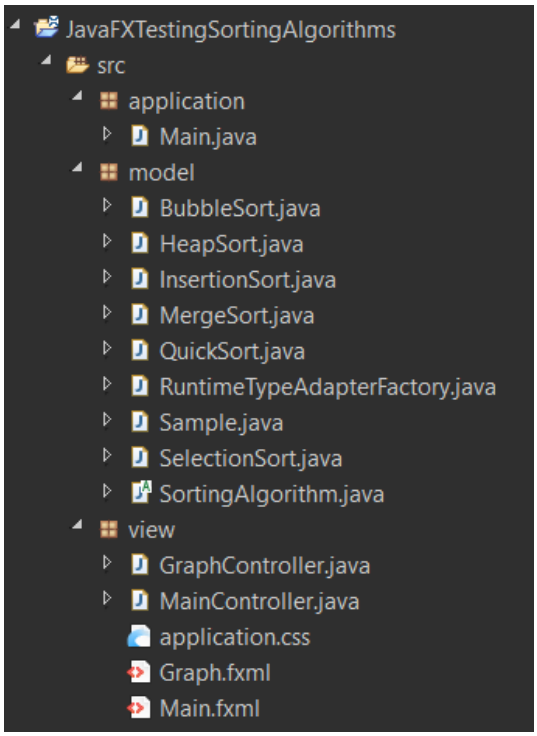
Primjer MVC arhitekture kao i odnos između svakog dijela prikazan je na Slici 4.2.



Slika 4.2. *Prikaz MVC arhitekture.*

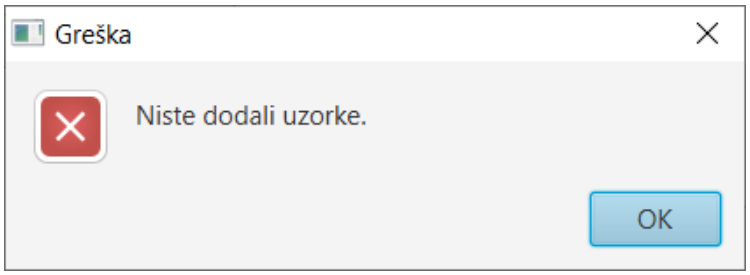
Prema MVC arhitekturi slijedi podjela aplikacije na sljedeće pakete (Slika 4.3.):

- *Application*
- *Model*
- *View*



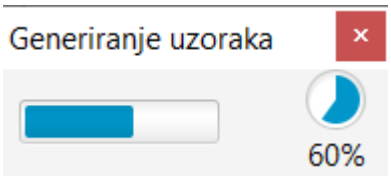
Slika 4.3. *Arhitektura klasa i paketa unutar aplikacije.*

Program kreće od „*Main*“ klase koja osim što pokreće proces pokretanja aplikacije stvara i početni prozor aplikacije utemeljen na „*Main.fxml*“ i application.css datotekama. „*Main.fxml*“ datoteka sadrži Java XML kôd koji se koristi za stvaranje korisničkog sučelja odnosno dodavanje korisničkih komponenti, dok application.css opisuje njihov izgled. Izgled komponenti smještenih unutar „*Main.fxml*“ datoteke prikazan je na Slici 4.1. Glavna klasa koja obavlja komunikaciju između korisničkog sučelja i *backend-a* je „*MainController*“. Ona također obavlja i biznis logiku te vrši generiranje testnih uzoraka i testiranje algoritama sortiranja nad tim uzorcima. Zadatak biznis logike unutar aplikacije je zabraniti korisniku unos podataka koji nisu u dozvoljenom formatu ili nisu unutar dozvoljenog raspona. Osim toga „*MainController*“ klasa hvata iznimke te ih pretvara u tekst razumljiv korisniku i prikazuje u obliku dijaloškog okvira upozorenja (Slika 4.4.).



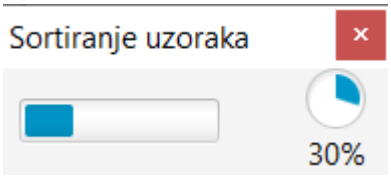
Slika 4.4. *Prikaz dijaloškog okvira upozorenja.*

Klikom na „*Generiranje*“ izvršava se poziv metode „*generateSamples()*“ koja generira testne uzorke odnosno polje ili polja ovisno o broju i veličini uzoraka te rasponu donje i gornje granice generiranja pseudo-slučajnih brojeva. Za vrijeme generiranja poziva se statična klasa „*ProgressForm*“ koja unutar dijaloškog okvira prikazuje postotak izvršenosti zadatka (Slika 4.5.).



Slika 4.5. *Prikaz postotka izvršenosti generiranja uzoraka.*

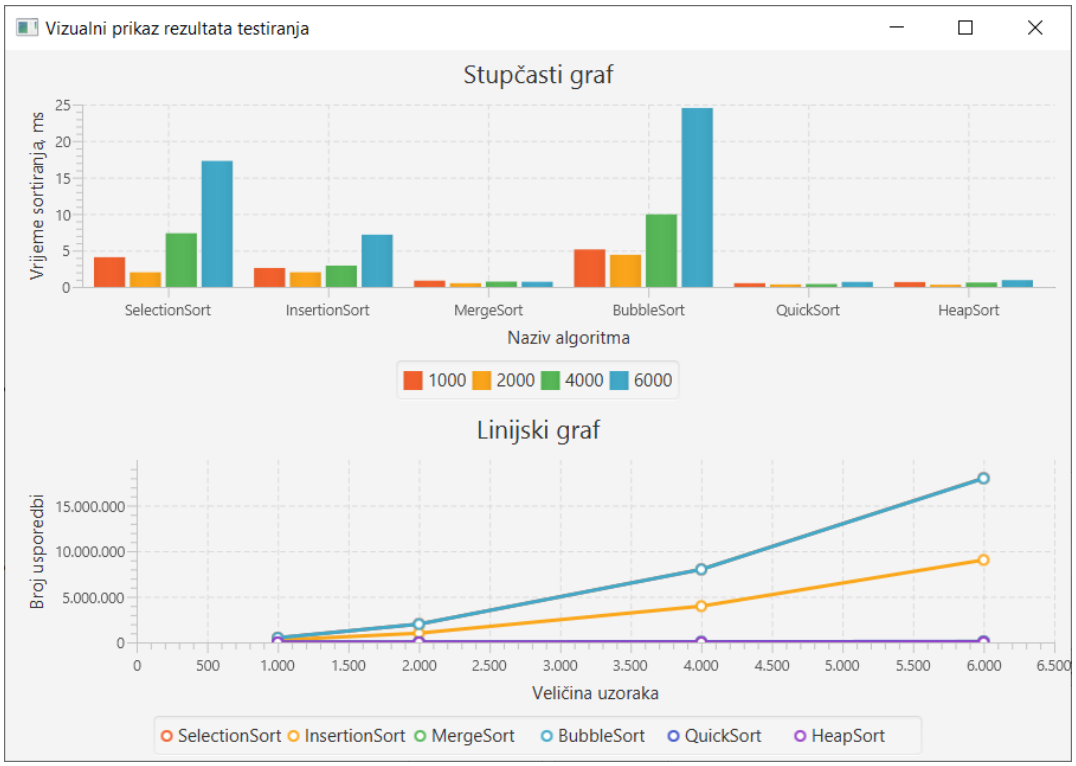
Klikom na „*Testiranje*“ izvršava se metoda „*testGeneratedSamples()*“ koja izvršava sortiranje generiranih testnih uzoraka pozivanjem klasa istog imena kao ime odabranih algoritama. Postotak izvršenosti sortiranja se također prikazuje unutar dijaloškog okvira pozivanjem statične klase „*ProgressForm*“ (Slika 4.6.).



Slika 4.6. *Prikaz postotka izvršenosti testiranja.*

Nakon završetka sortiranja informacije o vremenu potrebnom za sortiranje kao i informacije o broju usporedbi tijekom sortiranja spremaju se u objekte tipa ovisnog o nazivu algoritma, npr. navedene informacije za *Selection sort* algoritam spremaju se u objekt tipa „*SelectionSort*“. Na kraju svi kreirani objekti odabranih algoritama spremaju se u polje objekata tipa „*SortingAlgorithm*“ koje predstavlja roditeljsku klasu svih klasa za sortiranje. Tako popunjeno polje objekata kao i polje koje sadrži veličine svih uzoraka dodjeljuju se metodi unutar klase „*GraphController*“.

„*GraphController*“ klasa obavlja prikaz linijskog i stupčastog grafa kreiranih „*Graph.fxml*“ datotekom u ovisnosti o dobivenim rezultatima testiranja (Slika 4.7.).



Slika 4.7. *Vizualni prikaz rezultata testiranja.*

Kod prikazivanja rezultata ključna je metoda „*initData()*“ koja se izvršava automatski prilikom pozivanja „*GraphController*“ klase.

```
void initData(SortingAlgorithm[] sortingAlgorithm, int[]
sizeOfGeneratedSamples) {

	for(int i = 0; i < sizeOfGeneratedSamples.length; i++) {
		XYChart.Series<String, Double> timeSeries = new
XYChart.Series<>();

	timeSeries.setName(Integer.toString(sizeOfGeneratedSamples[i]));

	for(int j = 0; j < sortingAlgorithm.length; j++) {
		timeSeries.getData().add(new
XYChart.Data<>(sortingAlgorithm[j].getName(),
sortingAlgorithm[j].getTotalTime(i)));
	}

	bcTimeChart.getData().add(timeSeries);
	}



	for(int i = 0; i < sortingAlgorithm.length; i++) {
		XYChart.Series<Number, Number> sampleSeries = new
XYChart.Series<>();
		sampleSeries.setName(sortingAlgorithm[i].getName());

		for(int j = 0; j < sizeOfGeneratedSamples.length; j++) {
			sampleSeries.getData().add(new
XYChart.Data<>(sizeOfGeneratedSamples[j],
sortingAlgorithm[i].getListOfNumberComparisons(j)));
		}

		lcCompareChart.getData().add(sampleSeries);
	}
}
```

Programski kod 4.1. *Prikaz initData() metode.*

Metoda „*initData()*“ primljene rezultate testiranja prikazuje stupčastim i linijskim grafom. Stupčasti graf prikazuje ovisnost veličine testnih uzoraka o vremenu potrebnom za sortiranje generiranih testnih uzoraka, dok linijski graf prikazuje ovisnost veličine uzoraka o broju usporedbi koji je potreban za sortiranje generiranih testnih uzoraka za svaki pojedini algoritam.

Za spremanje i učitavanje rezultata testiranja koriste se metode „*saveSortingResults()*“ i „*loadSortingResults()*“. Metodu za spremanje „*saveSortingResults()*“ moguće je pozvati samo u

slučaju da postoje rezultati testiranja odnosno da je izvršeno testiranje algoritama sortiranja. Ova metoda poziva standardnu Java klasu „*FileChooser*“ koja otvara prozor za odabir direktorija u koji želimo spremiti rezultate testiranja. Rezultati se spremaju u obliku JSON datoteke koja sadrži granice za generiranje pseudo-slučajnih brojeva, listu dodanih uzoraka te listu tipa „*SortingAlgorithm*“ koja predstavlja roditeljsku klasu svih korištenih algoritama odnosno lista tipa „*SortingAlgorithm*“ sadrži podatke o vremenu potrebnom za sortiranje i broju usporedbi tijekom sortiranja za svaki korišteni algoritam. Izgled spremljenih rezultata u JSON datoteci prikazan je na Slici 4.8.

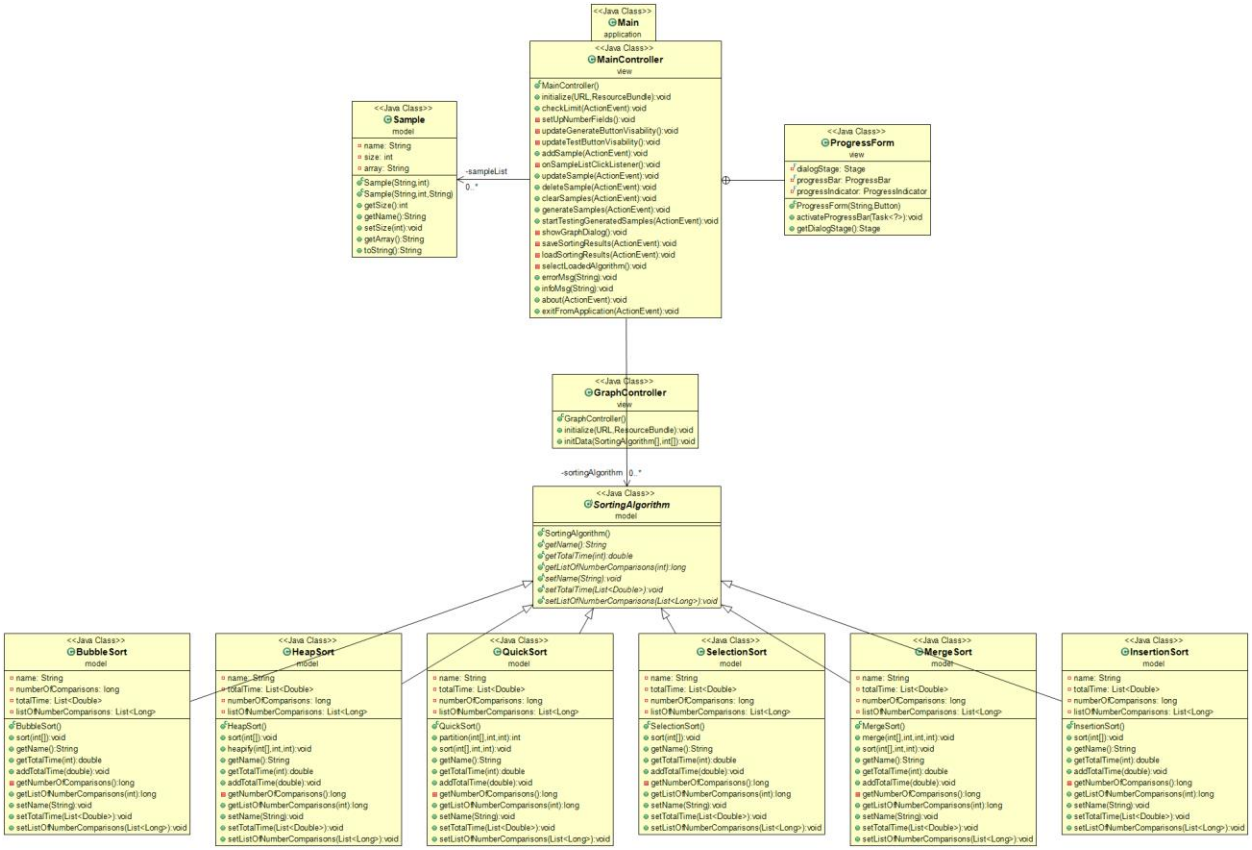

```
{
  "sizeOfSamples": [
    2000,
    4000
  ],
  "sortingAlgorithms": [
    {
      "name": "SelectionSort",
      "totalTime": [
        2.4772,
        7.903
      ],
      "numberOfComparisons": 0,
      "listOfNumberComparisons": [
        1999000,
        7998000
      ]
    }
  ],
  "upperLimit": 1000,
  "lowerLimit": -1000
}
```


Slika 4.8. *Izgled rezultata spremljenih u JSON datoteku.*

Metoda za učitavanje *„loadSortingResults()“* također poziva standardnu Java klasu *„FileChooser“* koja otvara prozor za odabir JSON datoteke. Nakon odabira datoteke JSON podaci se vraćaju u izvorne tipove podataka kako bi se mogla pozvati *„showGraphDialog()“* metoda koja prikazuje prozor za vizualni prikaz rezultata i prosljeđuje učitane rezultate testiranja do prethodno objašnjene *„initData()“* metode koja se nalazi unutar *„GraphController“* klase.

4.1.1. UML dijagram

Prema [6] UML (engl. *Unified Modeling Language*) se definira kao vrsta jezika za modeliranje koja sadrži skup pravila za grafički prikaz radi dizajniranja i opisivanja poslovnih procesa, softvera, organizacijskih struktura i ostalih sustava. Na Slici 4.9. prikazan je primjer UML dijagrama klasa koji prikazuje njihovu međusobnu vezu, vrstu veze te spisak atributa i metoda prikazanih klasa. UML dijagram prikazan na Slici 4.9. sadrži prikaz svih generiranih klasa unutar aplikacije kao i prikaz njihovih atributa i metoda. Crveni kvadratići predstavljaju privatne metode unutar klasa, dok zeleni kružići predstavljaju javne metode unutar klasa. Kvadratići s crvenim obrubom označavaju atribute unutar klasa.



Slika 4.9. *Prikaz UML dijagrama.*

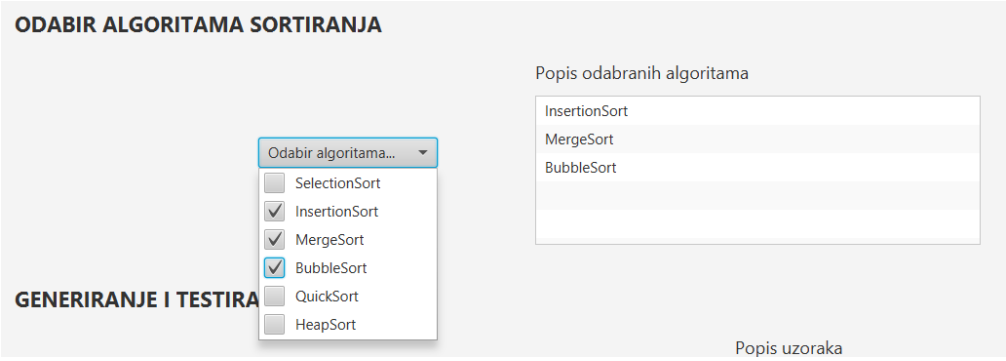
4.2. Rad s aplikacijom

JavaFX GUI aplikacija za testiranje algoritama sortiranja podijeljena je na tri osnovna dijela:

- Odabir algoritama sortiranja
- Generiranje testnih uzoraka i testiranje algoritama sortiranja
- Spremanje/učitavanje rezultata testiranja

4.2.1. Odabir algoritama sortiranja

Na Slici 4.10. prikazan je postupak odabira algoritama sortiranja za koje želi izvršiti testiranje.



Slika 4.10. *Prikaz postupka odabira algoritama sortiranja.*

Odabir algoritama vrši se klikom miša na padajući izbornik „*Odabir algoritama*“ te stavljanjem kvačice ispred željenih algoritma pri čemu on se automatski prikazuju na popisu odabranih algoritama.

4.2.2. Generiranje testnih uzoraka i testiranje algoritama sortiranja

Stavljanjem kvačice pored teksta „*Uključi granice*“ otvara se mogućnost unosa raspona unutar kojeg će program izvršiti generiranje pseudo-slučajnih cjelobrojnih brojeva (Slika 4.11.).

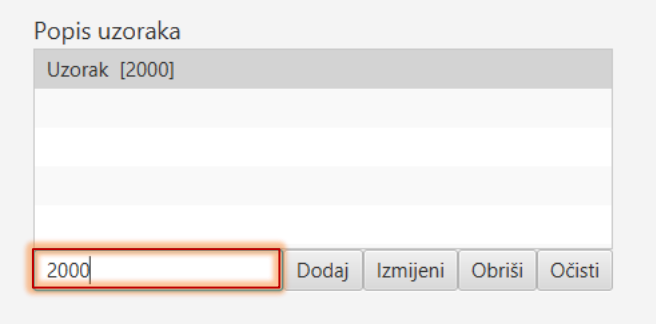
- Ograničenja:
    - Donja granica ne može biti manja od -65 536
    - Gornja granica ne može biti veća od 65 535
    - Donja granica ne može biti veća od gornje

- Gornja granica ne može biti negativna

- Ukoliko granice nisu uključene generiraju se slučajni brojevi od -65 536 do 65 535.

Slika 4.11. *Postupak uključivanja i podešavanja granica.*

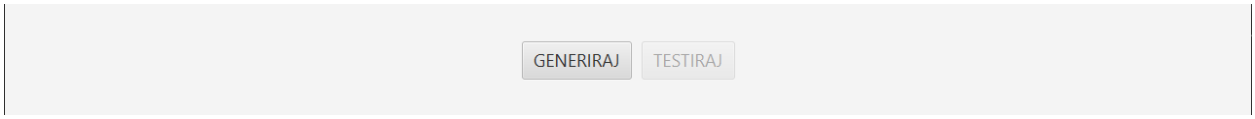
U prostor označen na Slici 4.12. unosi se veličina uzorka za koji se pritiskom na tipku „*Dodaj*“ dodaje novo polje veličine ovisne o unesenoj vrijednosti.



Slika 4.12. *Prikaz postupka dodavanja uzorka.*

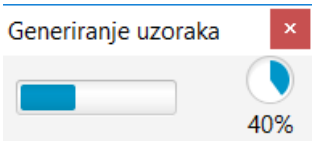
- Pojašnjenje gumbova za manipuliranjem uzorcima:
  - Dodaj – koristi se za dodavanje novog polja željene veličine na popis uzoraka
  - Izmijeni – koristi se za izmjenu veličine odabranog uzorka
  - Obriši – koristi se za brisanje odabranog uzorka
  - Očisti – koristi se za brisanje svih uzoraka s popisa
- Ograničenja:
  - Veličina uzorka ne može biti veća od 1 000 000
  - Broj uzoraka ne može biti više od 5
  - Veličina uzorka ne može biti negativna
  - Veličina uzorka ne može biti 0

Nakon što je odabran minimalno jedan algoritam za sortiranje i nakon što je dodan minimalno jedan uzorak otvara se mogućnost klika na gumb „*Generiraj*“ (Slika 4.13.).



Slika 4.13. *Gumb „Generiraj“ postaje aktivan.*

Klikom na „*Generiraj*“ generiraju se polja ovisno o broju dodanih uzoraka i veličina polja ovisno o unesenim veličinama uzoraka te broju odabranih algoritama. U polja se ubacuju pseudo-slučajno generirani cjelobrojni brojevi u rasponu od donje do gornje granice. Stvaraju se identične kopije polja za svaki odabrani algoritam odnosno svaki algoritam izvršava testiranje na polju jednake duljine, jednakih brojeva i jednakog redoslijeda pri čemu se ostvaruje ravnopravnost prilikom izvršavanja testiranja. Postotak izvršenosti generiranja prikazuje unutar novog prozora (Slika 4.14.).

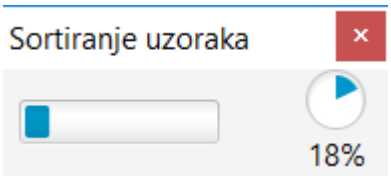


Slika 4.14. *Postotak izvršenosti generiranja uzoraka.*

Nakon uspješno završenog generiranja testnih uzoraka otvara se mogućnost klika na gumb „*Testiraj*“ (Slika 4.15.).

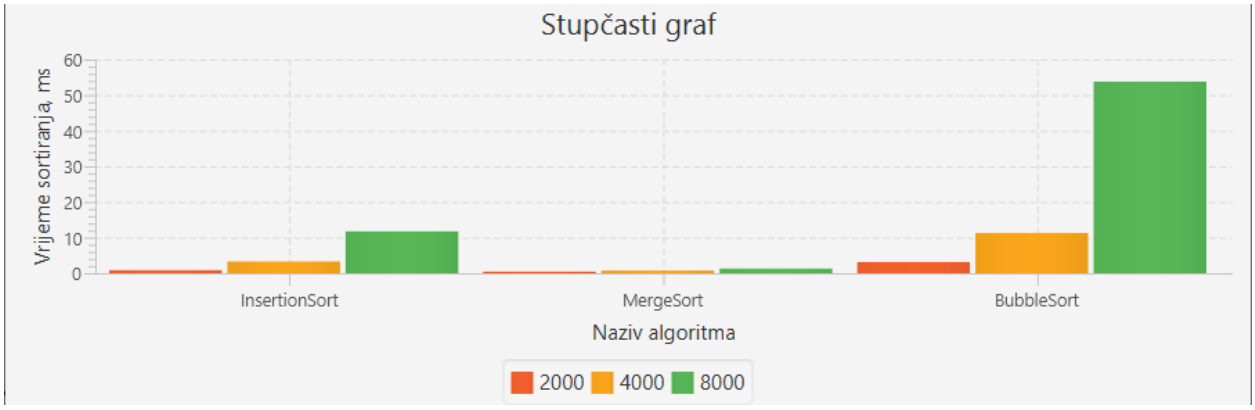
Slika 4.15. *Gumb „Testiraj“ postaje aktivan.*

Klikom na „*Testiraj*“ izvršava se sortiranje generiranih testnih uzoraka pomoću odabranih algoritama za sortiranje. Proces testiranja mjeri vrijeme koje je potrebno svakom algoritmu kako bi sortirao dobiveno polje te bilježi broj usporedbi koje su se dogodile tijekom sortiranja. Postotak izvršenosti sortiranja prikazuje se unutar novog prozora (Slika 4.16.).

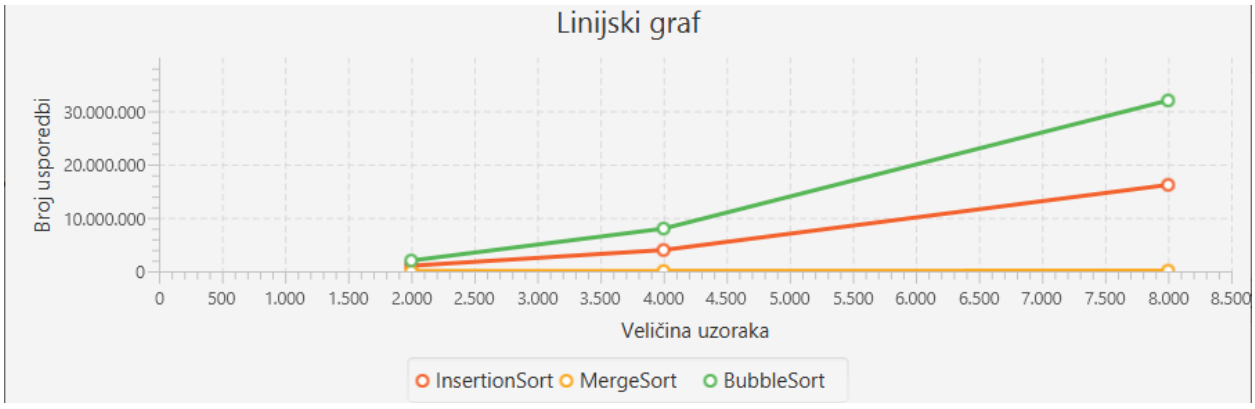


Slika 4.16. *Postotak izvršenosti sortiranja uzoraka.*

Nakon završetka sortiranja otvara se novi prozor koji sadrži stupčasti graf koji prikazuje ovisnosti potrebnog vremena o veličini testnih uzorka za odabrane algoritme, te linijski graf koji prikazuje ovisnost broja usporedbi o veličini testnih uzoraka za odabrane algoritme. Na slici 4.17. nalazi se prikaz stupčastog, dok se na Slici 4.18. nalazi prikaz linijskog grafa za tri uzorka veličine: 2000, 4000 i 8000 koji su sortiranih pomoću *Bubble*, *Insertion* i *Merge sort* algoritama.



Slika 4.17. *Prikaz stupčastog grafa (X-os prikazuje naziv algoritma, Y-os prikazuje vrijeme trajanja sortiranja u ms).*

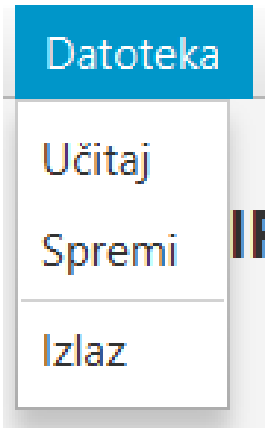


Slika 4.18. *Prikaz linijskog grafa (X-os prikazuje veličinu testnih uzorka, Y-os prikazuje broj usporedbi).*

Po završetku testiranja gumb „*Testiraj*“ ponovo postaje neaktivan sve dok se ne ispune uvjeti za novo testiranje odnosno dok se polje ponovo ne generira pseudo-slučajnim cjelobrojnim vrijednostima.

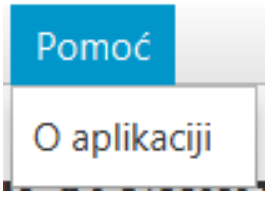
4.2.3. Spremanje/učitavanje rezultata testiranja

Aplikacija sadrži glavni izbornik koji se sastoji od dvije stavke: „*Datoteka*“ i „*Pomoć*“. Klikom na „*Datoteka*“ u glavnom izborniku otvara se padajući izbornik sa stavkama „*Učitaj*“, „*Spremi*“ i „*Izlaz*“ (Slika 4.19.).



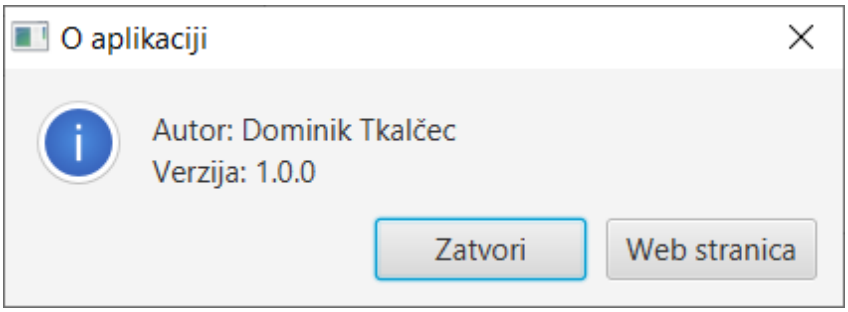
Slika 4.19. *Prikaz padajućeg izbornike stavke „Datoteka“.*

Klikom na „*Pomoć*“ u glavnom izborniku otvara se padajući izbornik sa stavkom „*O aplikaciji*“ (Slika 4.20.)



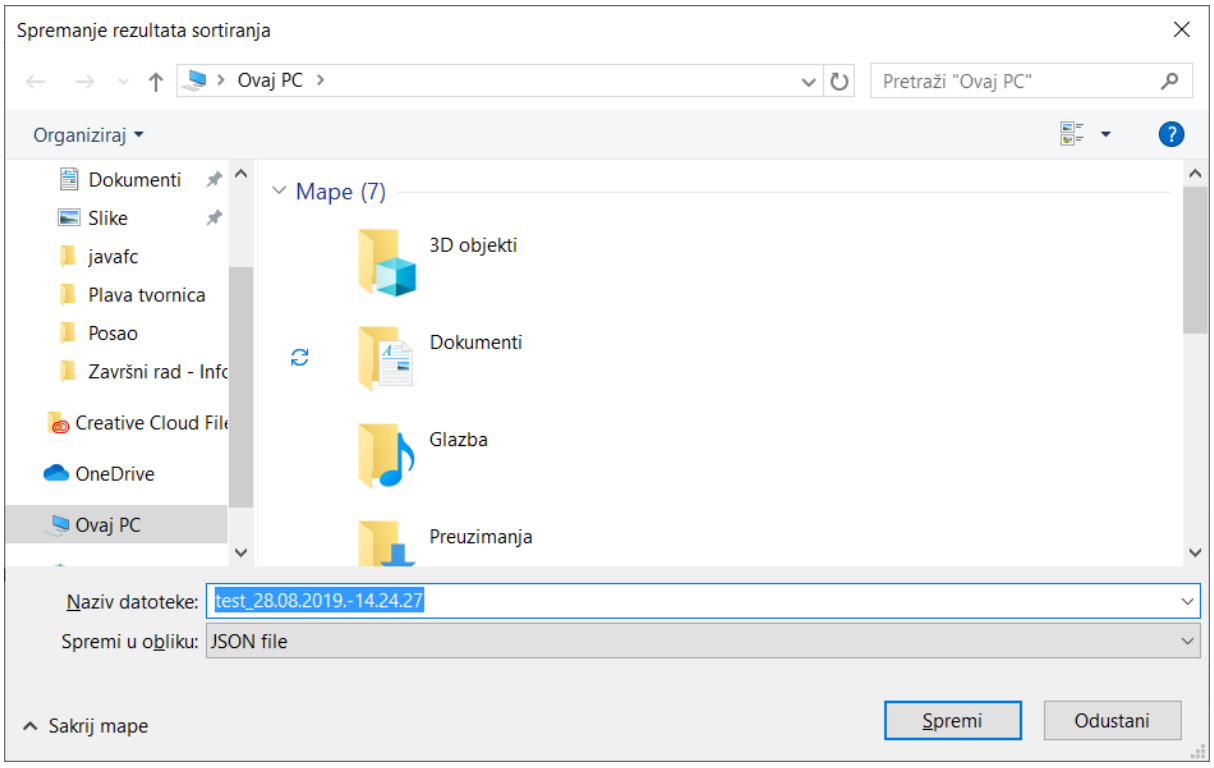
Slika 4.20. *Padajući izbornik stavke „Pomoć“.*

Klikom na stavku „*O aplikaciji*“ u padajućem izborniku otvara se novi prozor s informacijama o aplikaciji kao i gumbom „*Web stranica*“ koja vodi na web stranicu autora aplikacije (Slika 4.21.).



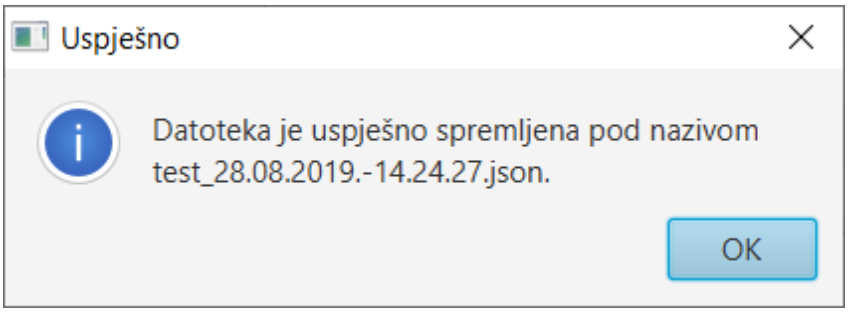
Slika 4.21. *Prozor s informacijama o aplikaciji.*

Rezultati dobiveni testiranjem spremaju se u željeni direktorij klikom na stavku „*Spremi*“ pri čemu se automatski otvara prozor za odabir direktorija (Slika 4.22.). Rezultati se spremaju u JSON datoteku naziva „*test_datum-vrijeme*“. Kreirana JSON datoteka sadrži donju i gornju granicu generiranja pseudo-slučajnih brojeva, listu koja sadrži veličine testnih uzoraka te vrijeme sortiranja i broj usporedbi sortiranja za svaki korišteni algoritam.



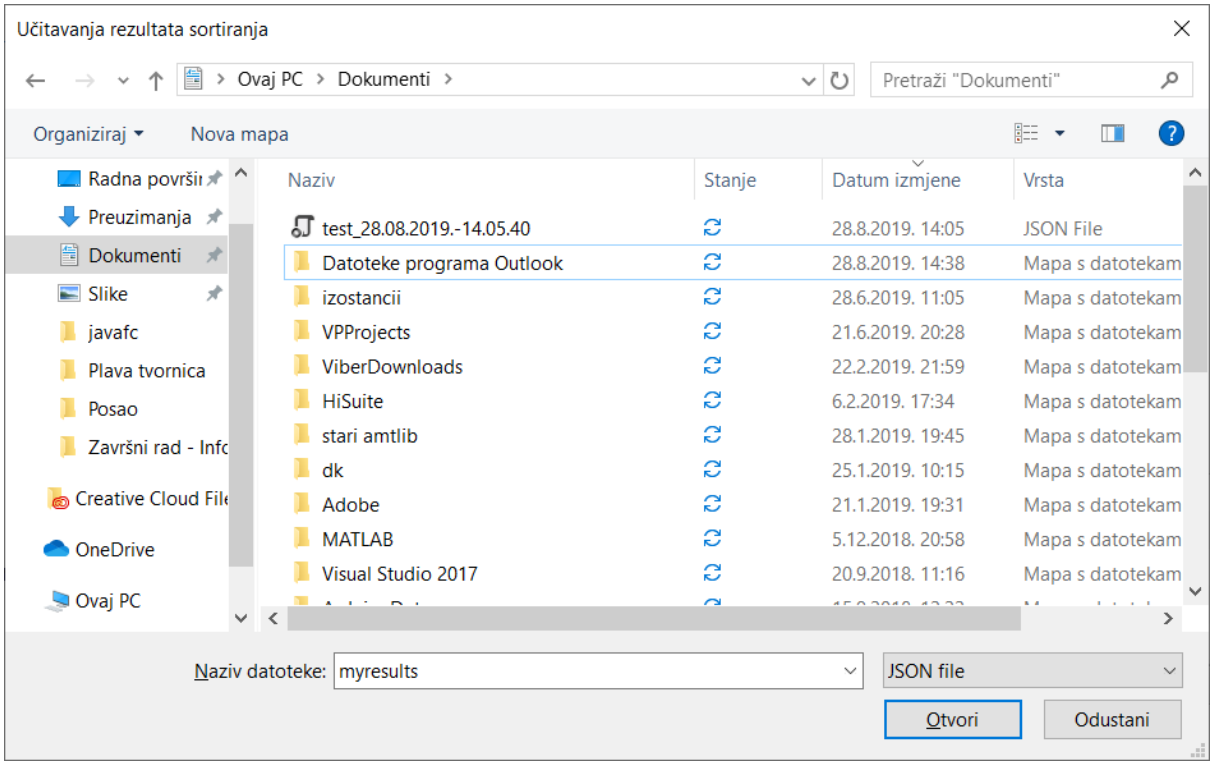
Slika 4.22. *Prozor za odabir direktorija.*

Nakon uspješnog odabira direktorija i spremanja rezultata u JSON datoteku prikazuje se dijaloški okvir o uspješnom spremanju (Slika 4.23.).



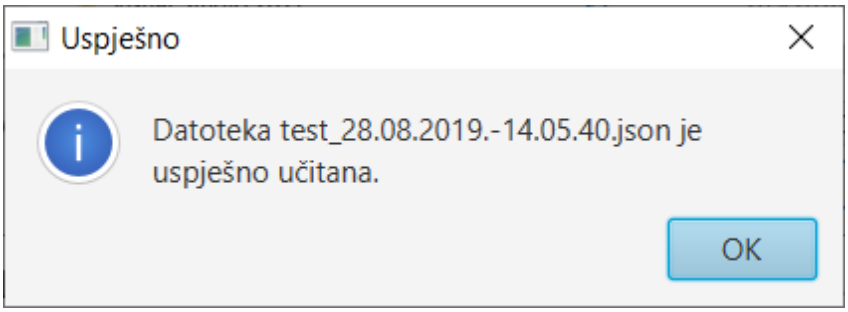
Slika 4.23. *Obavijest o uspješnom spremanju rezultata u datoteku.*

Spremljene rezultate moguće je ponovo učitati klikom na stavku „*Učitaj*“ u padajućem izborniku. Nakon klika na stavku automatski se otvara prozor za odabir JSON datoteke (Slika 4.24.).

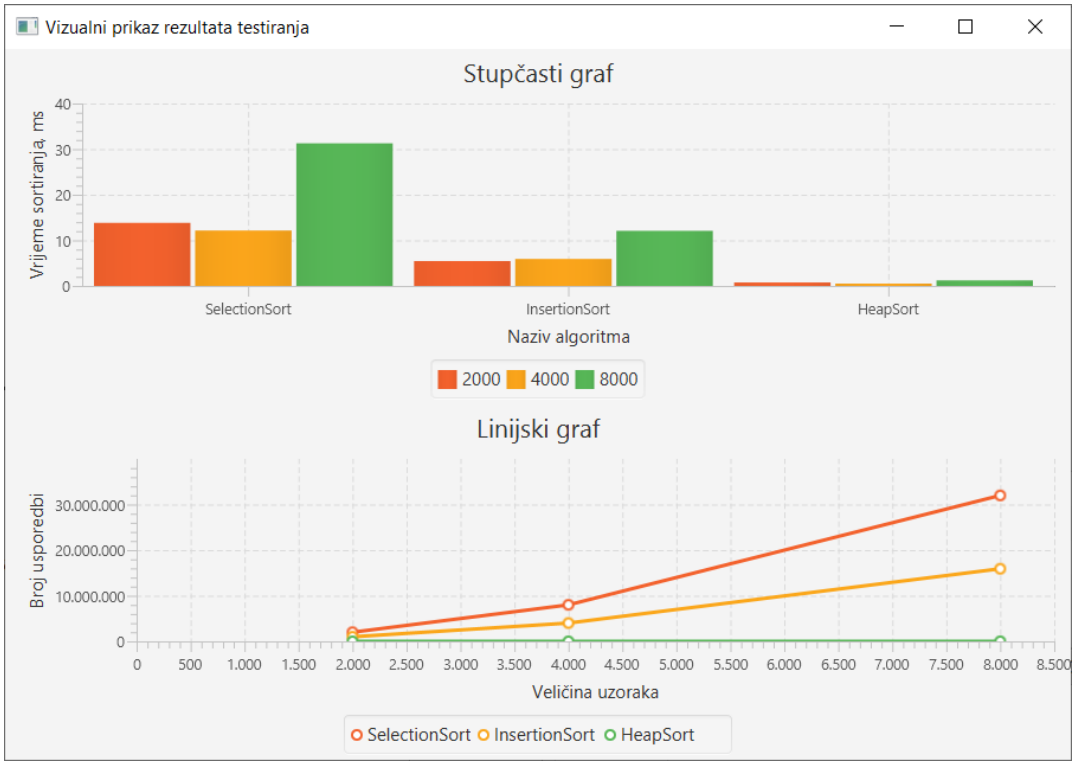


Slika 4.24. *Prozor za odabir rezultata u obliku JSON datoteke.*

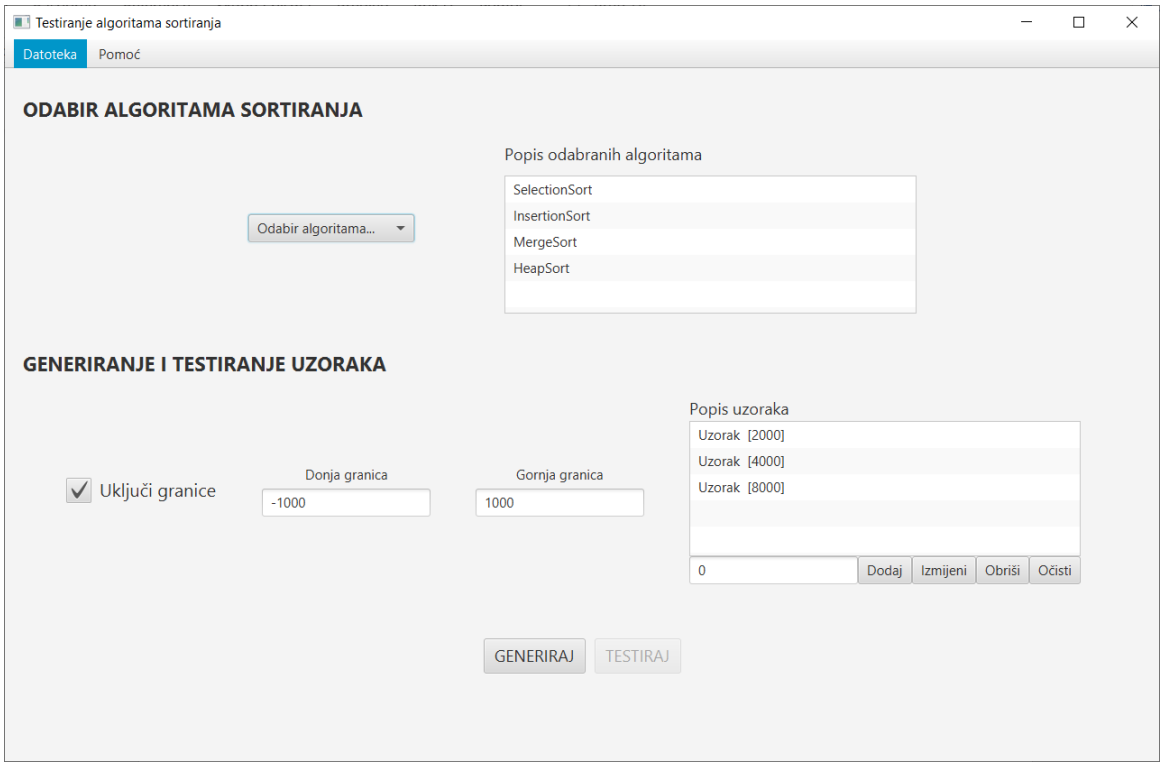
Dvostrukim klikom na željenu datoteku ili odabirom željene datoteke i klikom na „*Otvori*“ izbacuje se obavijest o uspješnom otvaranju datoteke (Slika 4.25.), prikazuju se grafovi sa spremljenim rezultatima (Slika 4.26.) te se početni prozor popunjava podacima spremljenim u JSON datoteku (Slika 4.27.).



Slika 4.25. *Obavijest o uspješnom učitavanju rezultata iz datoteke*



Slika 4.26. *Stupčasti i linijski graf dobiven rezultatima učitanim iz datoteke.*



Slika 4.27. *Početni prozor popunjen podacima iz učitane datoteke.*

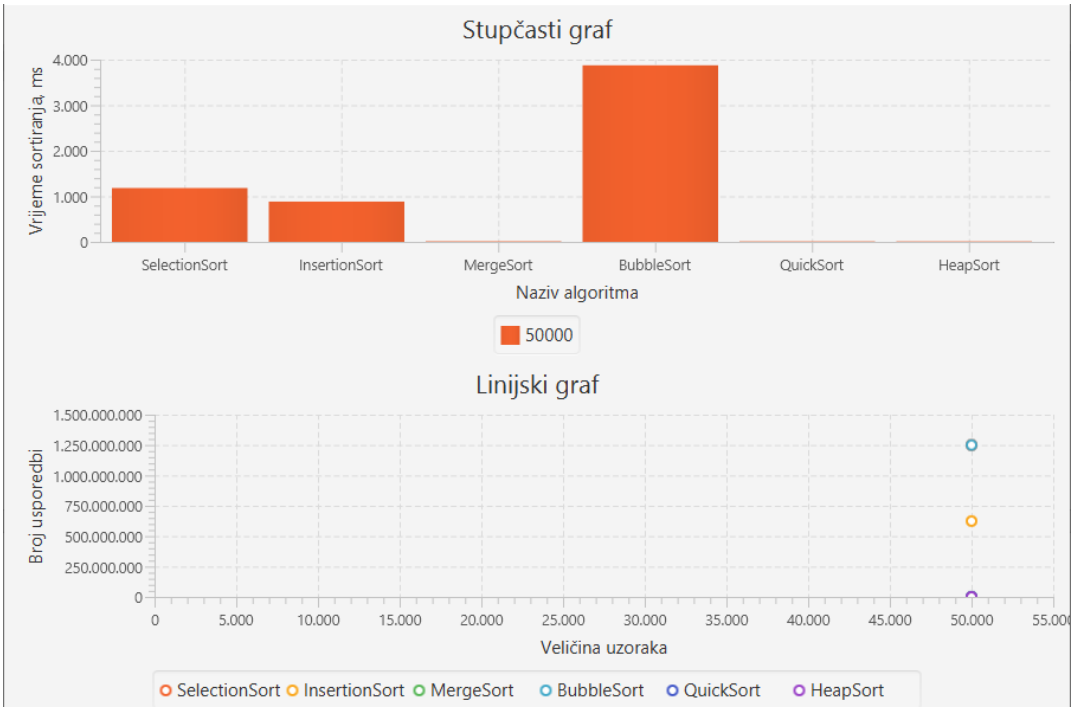
Učitana datoteka ne sadrži vrijednosti testnih uzoraka za koje su dobiveni rezultati već samo listu algoritama na kojima je izvršeno testiranje, granice u kojima su generirani pseudo-slučajni brojevi te broj uzoraka i veličinu svakog uzorka. Iz toga razloga nije moguće izvršiti testiranje bez generiranja testnih uzoraka pseudo-slučajnim brojevima. Učitane postavke iz glavnog prozora moguće je mijenjati i brisati.

4.3. Rezultati testiranja

U svrhu prikaza prednosti odnosno nedostataka jednog algoritma za sortiranje u odnosu na drugi izvršene su usporedbe rezultata dobivenih za različite parametre testiranja. Testiranja su izvršena na sljedećoj konfiguraciji računala:

- Procesor: Intel Core™ i7-6700HQ 2.60GHz
- Radna memorija: 8GB DDR4
- Grafička kartica: NVIDIA GeForce GTX 950M 2GB
- Pohrana podataka: 1000GB HDD, 256GB Samsung SSD 850 EVO

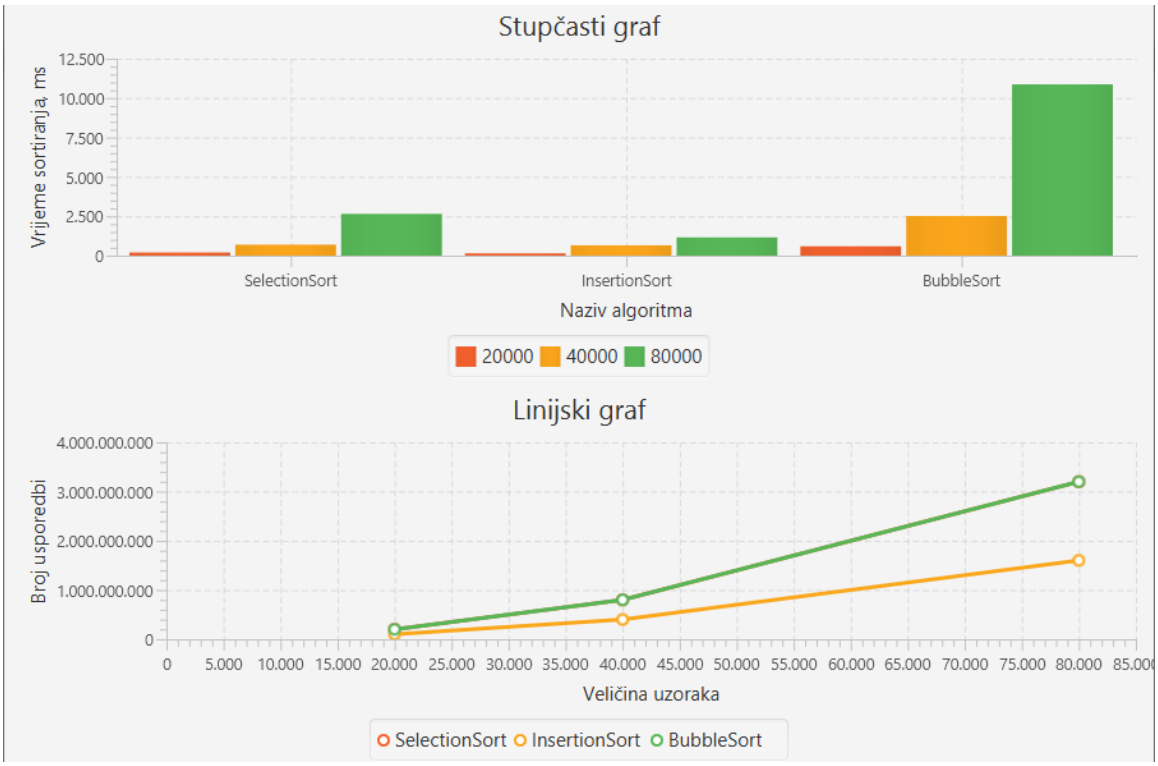
U prvom testiranju dodan je uzorak veličine 50 000 te su odabrani svi algoritmi za sortiranje. Granice generiranja pseudo-slučajnih brojeva postavljene su na zadanu vrijednost od -65536 do 65535. Rezultat prvog testiranja prikazan je na Slika 4.28.



Slika 4.28. *Rezultati prvog testiranja.*

Na Slici 4.28. može se uočiti razlog zbog kojeg se algoritmi *Merge*, *Quick* i *Heap sort* svrstavaju pod brze algoritme, a algoritmi *Selection*, *Insertion* i *Bubble sort* pod spore algoritme. Zbog velike razlike u vremenu izvršavanja između skupine sporih i skupine brzih algoritama teško je iz grafa očitati koji algoritam ima najmanje vrijeme izvršavanja, no lako je vidljivo da je *Bubble sort* najsporiji algoritam odnosno da je za sortiranje polja *Bubble sort* algoritmom bilo potrebno 1 250 000 000 usporedbi te 3875ms.

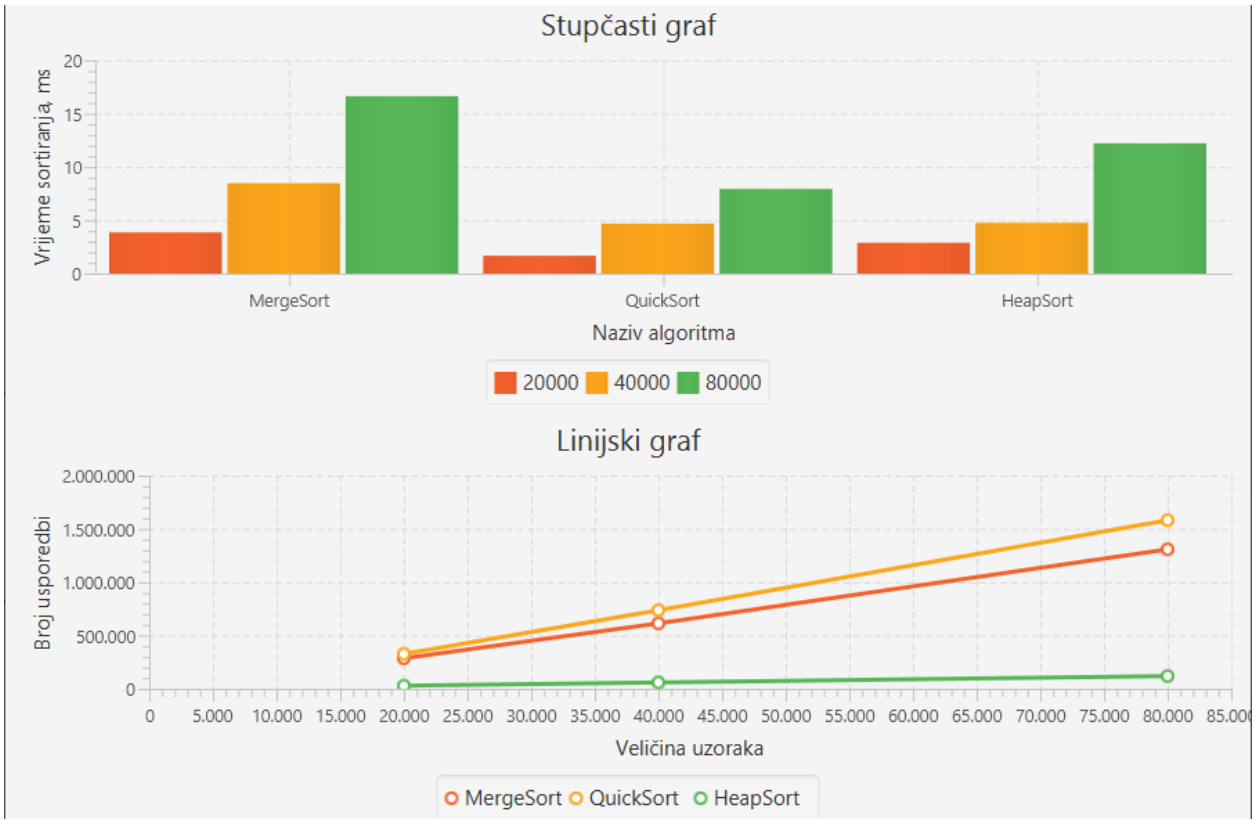
U drugom testiranju koriste se samo *Selection*, *Insertion* i *Bubble sort* algoritmi zbog jasnijeg prikaza rezultata. Testiranje se obavlja na uzorcima veličine 20 000, 40 000 i 80 000 te zadanim granicama generiranja pseudo-slučajnih brojeva. Rezultat drugog testiranja prikazan je na Slici 4.29.



Slika 4.29. *Rezultati drugog testiranja.*

Iz Slike 4.29. se može uočiti da je *Insertion sort* algoritam najbrži u sva tri slučaja s vremenom izvršavanja od 144.5841ms za 20 000 uzoraka, 653.4755ms za 40 000 uzoraka i 1155.0135ms za 80 000 uzoraka za razliku od *Bubble sort* algoritma koji je najsporiji u sva tri slučaja s vremenom izvršavanja od 592.6784ms za 20 000 uzoraka, 2514.3846ms za 40 000 uzoraka i 10871.5798ms za 80 000. Isto tako vidimo da broj usporedbi *Bubble sorta* ide čak i do 3 200 000 000 za razliku od *Insertion sorta* koji je obavio duplo manje usporedbi odnosno 1 600 000 000.

U trećem testiranju koriste se *Quick*, *Merge* i *Heap sort* algoritmi odnosno algoritmi iz skupine brzih algoritama. Testiranja su također obavljana na uzorcima veličine 20 000, 40 000 i 80 000 te zadanim granicama generiranja pseudo-slučajnih brojeva. Rezultat trećeg testiranja prikazan je na Slici 4.30.



Slika 4.30. *Rezultati trećeg testiranja.*

Iz Slike 4.30. je vidljivo da je *Quick sort* algoritam sva tri uzorka sortirao u najkraćem vremenu u odnosu na *Merge* i *Heap sort*, no isto tako je vidljivo kako je *Heap sort* algoritam obavio daleko najmanji broj usporedbi odnosno 1 400 000 manje u odnosu na *Quick sort* za veličinu uzorka 80 000. *Merge sort* se pokazao kao najsporiji algoritam iz skupine brzih algoritama, a *Quick sort* kao najbrži.

Zaključujemo kako je *Bubble sort* uvjerljivo najsporiji algoritam i da obavlja najveći broj usporedbi za istu veličinu uzoraka u odnosu na ostale algoritme. Isto tako zaključujemo kako je *Quick sort* najbrži algoritam u odnosu na ostale na kojima je izvršeno testiranje, no ne i najmanjeg broja usporedbi.

5. ZAKLJUČAK

Cilj ovog završnog rada je izrada GUI aplikaciju za testiranje algoritama sortiranja pomoću *JavaFX* grafičkih paketa te prikaz i usporedba rezultata testiranja algoritama sortiranja nad generiranim testnim uzorcima. Za implementaciju *backend-a* aplikacije korištena je MVC arhitektura kôda, dok je korisničko sučelje organizirano na dva dijela, odabir algoritama sortiranja i generiranje testnih uzoraka te testiranje algoritama sortiranja nad testnim uzorcima. Ta dva dijela implementirana su koristeći tek nekoliko UI komponenti čime se ostvarila maksimalna jednostavnost korištenja aplikacije.

Ovaj završni rad osim dijela vezanog za izrađenu aplikaciju sadrži i dio vezan za algoritme sortiranja koji su korišteni u aplikaciji. U tom dijelu završnog rada objašnjeni su principi rada algoritama prema programskom kôdu koji se nalazi ispod vizualnog prikaza svakog algoritma. Izrađena aplikacija generira testne uzorke odnosno polja popunjena pseudo-slučajnim cjelobrojnim brojevima na temelju parametara podešenih u glavnom prozoru aplikacije te na tim uzorcima provodi testiranja različitih algoritama sortiranja kako bi izmjerio vrijeme potrebno za sortiranje i broj usporedbi tijekom sortiranja. Rezultati testiranja se prikazuju u obliku stupčastog i linijskog grafa za svaki korišteni algoritam. Grafovi omogućuju jednostavno očitavanje rezultata testiranja te daju ljepši i pregledniji prikaz rezultata.

Aplikacija može biti korisna studentima, ali i ostalim korisnicima kako bi mogli procijeniti u kojim uvjetima odabrati koji algoritam sortiranja. Npr. ukoliko je korisniku potreban algoritam za sortiranje koji će u najkraćem mogućem vremenu sortirati polje od nekoliko tisuća podataka onda će definitivno izabrati *Quick sort*, no ukoliko je korisniku važnije da manje optereti procesor nego da što prije obavi zadatak izabrat će *Heap sort* algoritam. Kao poboljšanje ove aplikacije moguće je dodati virtualni prikaz sortiranja polja tijekom izvršavanja sortiranja.

LITERATURA

[1] J. Potts, N. Hildebrandt, J. Gordon, C. Castillo, JavaFX Scene Builder: Using JavaFX Scene Builder with Java IDEs, Oracle, Redwood City, 2014., dostupno na https://docs.oracle.com/javase/8/javafx/JFXST.pdf [13.06.2019.]

[2] C. Castillo, JavaFX: JavaFX Getting Started with JavaFX, Oracle, Redwood City, 2014., dostupno na https://docs.oracle.com/javase/8/scene-builder-2/JSBID.pdf [13.06.2019.]

[3] R. Manager, Struktura podataka i algoritmi: Nastavni materijal, Sveučilište u Zagrebu, Zagreb, 2013.

[4] Sorting Algorithms, GeeksForGeeks, 2017., https://www.geeksforgeeks.org/sorting-algorithms/ [13.06.2019.]

[5] Z. Kojčić, Završni rad: Pregled algoritama sortiranja, Filozofski fakultet, Osijek, 2012.

[6] UML osnove, Učim programiranje, 2012., http://www.ucim-programiranje.com/2012/12/uml-osnove/ [27.06.2019.]

SAŽETAK

U sklopu ovog završnog rada izrađena je *JavaFX* GUI aplikacija za testiranje odabranih algoritama sortiranja koja rješava problem odabira najbržeg odnosno najefikasnijeg algoritma. Programski kôd je pisan Java programskim jezikom u *Eclipse* razvojnom okruženju koji je povezan sa *Scene Builder* programskim alatom za generiranje *JavaFX* komponenti korisničkog sučelja. Korisničko sučelje na intuitivan način omogućava korisniku unos parametara za generiranje uzoraka na temelju kojih se provodi testiranje algoritama sortiranja. Dobiveni rezultati testiranja se mogu spremiti u JSON datoteku te ih je kasnije moguće ponovo učitati radi vizualnog prikaza rezultata u obliku grafova, ali i radi novog testiranja. Na temelju rezultata dobivenih izrađenom aplikacijom moguće je algoritme podijeliti na spore (*Bubble*, *Insertion* i *Selection sort*) i brze (*Quick*, *Merge* i *Heap sort*), te je moguće jednostavno s grafa očitati vrijeme potrebno za sortiranje kao i broj usporedbi tijekom sortiranja odabranim algoritmima.

Ključne riječi: algoritmi, *Eclipse*, *JavaFX*, *SceneBuilder*, sortiranje

ABSTRACT

Title: JavaFX GUI application for testing sorting algorithms

As part of this final thesis, a JavaFX application was developed for testing selected sorting algorithms. The application solves the problem of selecting the fastest i.e. the most efficient algorithm. The code is written in Java programming language using Eclipse development environment which is linked to the Scene Builder software tool for generation of the JavaFX user interface components. The user interface is interactive with user in such a way that user enters sample generation parameters for which the test is being performed. The resulting test data can be saved to a JSON file so that it can be later reloaded for visual display in the form of graphs, as well as for new testing. The results obtained by using the application created for the purpose of this final thesis make it possible to classify sorting algorithms into slow (Bubble, Insertion and Selection sort) and fast (Quick, Merge and Heap sort) and the time required for sorting can be easily read from the graph as well as the number of comparisons sorting the selected algortihm.

Keywords: algorithms, *Eclipse*, *JavaFX*, *SceneBuilder*, sorting

ŽIVOTOPIS

Dominik Tkalčec, rođen 13. listopada 1997. godine u Virovitici. Odrastao je u Otrovancu, selu pokraj Pitomače gdje živi i danas. Školovanje započinje 2004. godine u Područnoj školi Otrovanec od 1. do 4. razreda, a ostatak osnovnoškolskog obrazovanje dovršio je u Osnovnoj školi Petra Preradovića u Pitomači. Pohađao je Tehničku školu u Virovitici te 2016. godine stekao zvanje Računalni tehničar za strojarstvo. Iste godine sudjeluje i na Državnom natjecanju učenika strojarskih zanimanja u kategoriji „CNC tehnologije – glodanje/tokarenje“. Obrazovanje nastavlja upisom stručnog studija Informatike na Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2016. godine. Na 3. godini radi kao demonstrator na FERIT-u dajući studentima savjete i pomažući im riješiti problem iz C programiranja te Arduino programiranja. Praksu je odradio 2019. godine kao back-end developer u Atos-u Osijek gdje upotpunjuje svoje znanje iz Java programskog jezika.

Dominik Tkalčec

PRILOZI

Projektna mapa s izvornim kôdom nalazi se na optičkom disku koji je priložen uz printanu verziju završnog rada.