Qt GUI aplikacija za testiranje algoritama sortiranja

Klen, Vlatko

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

Stručni studij

QT GUI APLIKACIJA ZA TESTIRANJE ALGORITAMA SORTIRANJA

Završni rad

Vlatko Klen

Osijek, 2019.

SADRŽAJ

1. UVOD

Aplikacija je pri izradi podijeljena u dva dijela, statički i dinamički dio aplikacije. Statički dio predstavljaju podatci, dok dinamički predstavljaju algoritmi koji definiraju ponašanje aplikacije. Algoritam mora biti sastavljen od konačnog broja koraka koji ukazuju na slijed operacija koje treba obaviti nad objektima kako bi se dobili završni objekti ili rezultati[5]. Svaki korak algoritma se opisuje instrukcijom, a obavljanje algoritama naziva se algoritamskim procesom .

U ovom radu koristiti ćemo neke od jednostavnijih algoritama sortiranja sa ciljem njihove lakše usporedbe. Algoritmi koje ćemo koristiti su pisani u programskom jeziku C++. Razmatrat će se sljedeći algoritmi:

- Mjehuričasto sortiranje
- Sortiranje izborom
- Sortiranje umetanjem
- Brzo sortiranje
- Sortiranje spajanjem
- Sortiranje pomoću hrpe

Sortiranje podataka je proces uređenja određenog niza redanjem po nekim kriterijima. Algoritmi za sortiranje se značajno razlikuju po brzini izvođenja te na koje načine vrše sortiranje, njihova brzina izvođenja ovisi i o parametrima polja koje sortiraju. Primjena sortiranja podataka vidi se u određenim računalnim procesima, u sustavima koji koriste baze podataka, sortiranje daje mogućnost prikaza podataka redom prema željenim kriterijima.

Cilj rada je usporediti algoritme sortiranja, raspraviti njihove prednosti i mane, vidjeti kako se ponašaju sortirajući ulazne podatke različitih veličina i raspona, definirati i objasniti njihove složenosti.

2. ALGORITMI SORTIRANJA

Potreba za sortiranjem podataka je veoma učestala, svi programi koji rukuju velikom količinom podataka pružaju opciju sortiranja prema nekim kriterijima. Sortiranje podataka značajno olakšava rad s njima, pretraživanje sortiranih podataka je brže nego pretraživanje nasumičnih.
Najčešća sortiranja su leksikografska (sortiranje riječi i slova) i brojevna. Leksikografsko sortiranje se vrši prema abecednom poretku, te je moguće sortirati silazno – od većeg prema manjem, te uzlazno – od manjeg prema većem.

Algoritme analiziramo kako bi utvrdili koji algoritam je bolji, to jest koji algoritam troši manje resursa te da je pri tome učinkovit, ti resursi su prostor i vrijeme. Prostor se odnosi na memoriju računala, dok vrijeme predstavlja potrebno vrijeme da algoritam sortira podatke. Pošto ulazni podaci ne ovise samo o veličini, nego i drugim svojstvima, postoje tri slučaja učinkovitosti:

- Najbolja učinkovitost – ulazni podaci takvi da algoritam najkraće traje
- Najlošija učinkovitost – ulazni podatci takvi da algoritam najduže traje
- Prosječna učinkovitost – ulazni podaci su nasumični

U programiranju, složenost algoritma ovisna je o broju instrukcija koje algoritam treba napraviti da dobije krajnji rezultat. Složenost algoritma se zapisuje kao ovisnost broja elemenata koje je potrebno sortirati, te vrijeme potrebno da se to napravi. Složenost označavamo O (big-O) notacijom, gdje n predstavlja broj elemenata za sortirati.

Složenosti algoritama poredane od manje prema većoj:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n_2) < O(n_3) < \ldots < O(2_n) < O(n!)$$

2.1. Mjehuričasto sortiranje

Mjehuričasto sortiranje (engl. *Bubble sort*) je jednostavan algoritam koji sortiranje vrši prolaskom kroz cijelo polje ili listu elemenata, te svaki uspoređuje sa prethodnim. Ukoliko im poredak nije dobar, on zamjenjuje ta 2 elementa, tako da već prvim prolaskom postavlja pripadajući element na prvo mjesto. Polje koje sortiramo prolazimo n-1 puta, jednom za svaki element osim posljednjeg, koji će automatski biti sortiran nakon što su svi drugi elementi. Prilikom sortiranja polje se dijeli na 2 dijela: sortirani i ne sortirani dio, prilikom svakog prolaska

algoritam samo prolazi nesortiranim dijelom. U programskom kôdu 2.1. prikazan je algoritam mjehuričasto sortiranje.

```
void BubbleSort::Sort(vector<int> array, int n)
{
   int i, j;
   bool swapped;
   for (i = 0; i < n-1; i++)
   {
     swapped = false;
     for (j = 0; j < n-i-1; j++)
     {
        if (array[j] > array[j+1])
        {
           Swap(&array[j], &array[j+1]);
           swapped = true;
        }
     }
     if (swapped == false)
        break;
   }
}
```

Programski kôd 2.1. *Mjehuričasto sortiranje implementirano u jeziku C++ u Qt okruženju. [6]*

„*BubbleSort*“ funkcija kao parametre prima polje koje želimo sortirati, te broj elemenata u tom polju. Koristi se za sortiranje manjih skupova podataka. Najbolja učinkovitost je kad je polje već sortirano te je potreban samo jedan prolazak kroz polje, vremenska složenost u tom slučaju je O(n). Učinkovitost je najgora u slučaju gdje je naopako sortirano te je tad vremenska složenost O(n^2). U slučaju nasumičnog polja, prosječna vremenska složenost isto iznosi O(n^2).

2.2. Sortiranje izborom

Sortiranje izborom (engl. *Selection sort*) spada u jednostavne algoritme sortiranja. Radi tako da prolazi kroz polje elemenata te bira element s najmanjom vrijednost, potom ga zamjeni sa vrijednosti koja stoji na zadnjem mjestu, taj element je onda već na svom krajnjem mjestu. Taj postupak se ponavlja, s tim da pri prolasku kroz polje nisu uključeni već sortirani elementi. Ako je polje dugo n elemenata, nakon n-1 obilazaka će polje elemenata biti u potpunosti sortirano. U programskom kôdu 2.2. prikazan je algoritam sortiranje izborom.

```
void SelectionSort::Sort(vector<int> &arr, int n)
{
    int i, j, min_idx;
    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Programski kôd 2.2. *Sortiranje izborom implementiran u jeziku C++ u Qt okruženju. [6]*

„SelectionSort“ funkcija kao parametre prima polje koje želimo sortirati, te broj elemenata u tom polju. U sebi prilikom zamjene elemenata poziva funkciju *swap* koja zamjeni vrijednosti elemenata koje funkcija preda.

Vremenska analiza složenosti, tj. ukupni broj operacija ovisi o broju usporedbi i pridruživanja. Pri prvom prolasku broj usporedbi je n-1, te se svakim prolaskom smanjuje za jedan, konačni broj je *n(n-1)/2*. Prilikom prolaska kroz polje elementi mijenjaju pozicije, pa operacija pridruživanja ima *3(n-1)*. Ukupan broj operacija je: : *n(n-1)/2+3(n-1)=O(n2) [5]*.

2.3. Sortiranje umetanjem

Sortiranje umetanjem (engl. *Insertion sort*) je jednostavan algoritam za sortiranje koji je u praksi učinkovitiji od drugih jednostavnih algoritama (mjehuričasto sortiranje i sortiranje izborom). Princip se temelji na dijeljenju polja na dva dijela: nesortirani i sortirani dio. U početku prvi element je smatran sortiranim, a ostatak ne sortiranim. Svakim prolaskom polja, algoritam uzima prvi element iz dijela koji nije sortiran te ga stavlja na odgovarajuće mjesto u sortiranom dijelu, time je sortirani dio polja povećan za jedan, a ne sortirani manji. U programskom kôdu 2.3. prikazan je algoritam sortiranje umetanjem.

```
void InsertionSort::Sort(vector<int> arr, int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Programski kôd 2.3. *Sortiranje umetanjem algoritam u jeziku c++ u Qt okruženju. [6]*

„*InsertionSort*“ funkcija prima polje koje želimo sortirati, te njegovu duljinu. Učinkovitost je najveća kad je zadano polje već sortirano, u tom slučaju ima samo jedna usporedba po iteraciji kroz petlju, znači n-1 usporedbi. Tada je vremenska složenost O(n). Učinkovitost je najmanja u slučaju najvećeg broja usporedbi za svaki iteraciju kroz petlju, tada je vremenska složenost ista kao i kod prosječnog slučaja a to je $O(n^2)$. Kvadratna složenost u prosječnom slučaju znači da algoritam nije dobar za sortiranje velikih polja. Međutim ima neke prednosti:

- Dobra učinkovitost pri sortiranju manjih polja
- Učinkovitiji od ostalih algoritama iste složenosti
- Jednostavan za implementirati
- Stabilan, tj. ostavlja redosljed elemenata s istim vrijednostima nepromjenjenim

2.4. Sortiranje pomoću hrpe

Sortiranje pomoću hrpe (engl. *Heap sort*) za razliku od prethodnih algoritama koristi posebnu strukturu podataka prilikom sortiranja zvanu hrpa (engl. *Heap*). Ideja je da sve elemente smjestimo u vrstu stabla kojeg nazivamo hrpa ili gomila. Hrpu punimo tako da se sve razine osim zadnje popunje, a posljednja se popuni s lijeva na desno. Za hrpu vrijedi da svi vrhovi imaju manju vrijednost od svoja 2 nastavka. Novi element dodajemo na kraj stabla te uspoređujemo s roditeljima, ako ima manju vrijednost zamjenjujemo ta 2 elementa. Taj postupak ponavljamo sve dok novi element ne dobije roditeljski element koji je manji od njega ili ne postane korijen stabla. Radi na principu unutarnjeg sortiranja, znači da sve elemente sprema u

radnu memoriju, te ih zatim sortira. U programskom kôdu 2.4. prikazan je algoritam sortiranje pomoću hrpe.

```
void HeapSort::Heapify(vector<int> &arr, int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        Swap(&arr[i], &arr[largest]);
        Heapify(arr, n, largest);
    }
}
void HeapSort::Sort(vector<int> &arr, int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        Heapify(arr, n, i);
    for (int i=n-1; i>=0; i--)
    {
        Swap(&arr[0], &arr[i]);
        Heapify(arr, i, 0);
    }
}
```

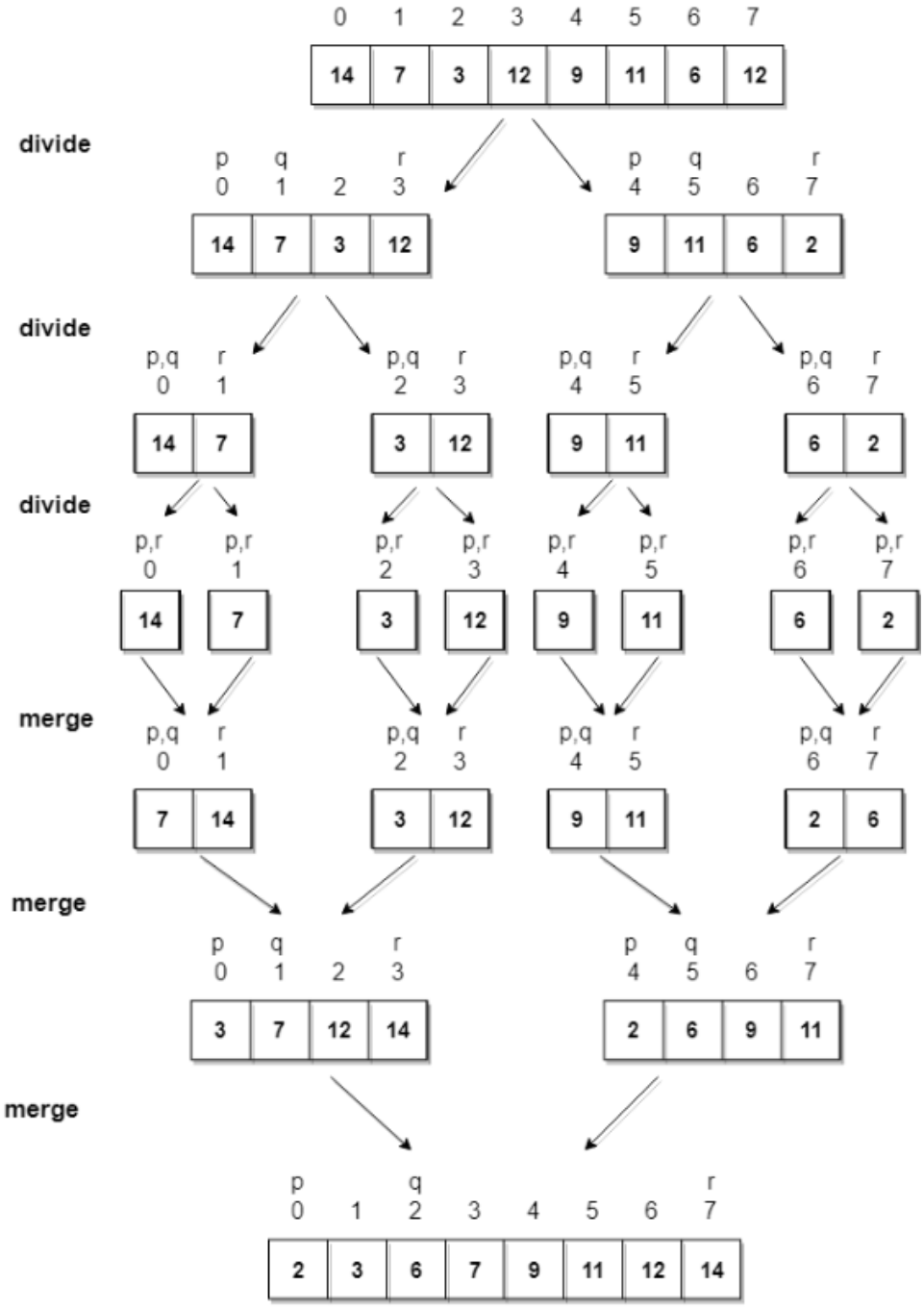
Programski kôd 2.4. *Sortiranje pomoću hrpe algoritam u jeziku c++ u Qt okruženju. [6]*

Sa svojom vremenskom složenosti *O(n log(n))*, algoritam sortiranje pomoću hrpe je optimalan. Ne zahtjeva nikakvu dodatnu memoriju, ali nije stabilan, tj. mijenja poredak elemenata iste vrijednosti.

2.5. Sortiranje spajanjem

Sortiranje spajanjem (engl. *Merge sort*) je algoritam koji sortiranje vrši podjelom polja koristeći rekurziju. „Algoritam je rekurzivan ako poziva sam sebe za obavljanje dijela posla. Kako bi to bilo uspješno, svakim pozivom problem treba podijeliti na manji nego početni.“[2]. Algoritam radi na principu podijeli pa vladaj. Zadano polje je podjeljeno na dva manja podjednaka dijela, zatim se rekurzivno manji dijelovi ponovo dijele. Nakon maksimalnog razdjeljivanja polja, najmanje dijelove sažima u jedno sortirano polje koristeći algoritam za

spajanje dva sortirana polja u jedno isto tako sortirano polje. Algoritam sortiranje spajanjem je prikazan na slici 2.1. i programskom kôdu 2.5.



Slika 2.1. *Vizualizacija algoritma sortiranje spajanjem.*

```
void MergeSort::Merge(vector<int> &arr, int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;
    int R[n2], L[n1];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

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

void MergeSort::Sort(vector<int> &arr, int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;
        Sort(arr, l, m);
        Sort(arr, m+1, r);
        Merge(arr, l, m, r);
    }
}
```

Programski kôd 2.5. *Implementacija algoritma sortiranje spajanjem u C++ jeziku, Qt okruženju. [6]*

Prilikom rekurzivnih poziva, vrijeme je proporcionalno veličini polja koje nastaje. Svi rekurzivni pozivi rade s poljima čija je veličina jednaka početnom polju. Izračun vremena za sve rekurzivne pozive je *O(n),* ima ih $log_2(n+1)$ pa je složenost algoritma u najgorem slučaju *O(n log n)* [1].

2.6. Brzo sortiranje

Brzo sortiranje (engl. *Quick sort*) je složen algoritam koji radi na načelu podijeli pa vladaj, te koristi rekurziju. Nastao je pokušajem ubrzavanja koraka spajanja polja, kod njega korak spajanja je izbjegnut. Prvo se ističe jedan element liste – „pivot“. Taj pivot može biti bilo koji element liste, te se prema njegovoj vrijednosti izvorna lista podijeli na dva dijela. Bitno je da se pivot izračuna u O(1) vremenu. Lijevo od pivot broja se slažu elementi koji su manji od njega, desno od njega slažu se veće vrijednosti. Te liste opet na isto podijeli dok ne dobije listu sa jednim elementom, te kao rezultat dobije sortirano polje. Brzo sortiranje ima više implementacija, ovisno o odabiru pivot elementa. Algoritam koji koristimo izabire zadnji element polja kao pivot. Na slici 2.2. i programskom kôdu 2.6. je prikazan algoritam brzo sortiranje.

Slika 2.2. *Vizualni prikaz algoritma brzo sortiranje [1].*

```
int QuickSort::partition (vector<int> &arr, int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] <= pivot)
        {
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void QuickSort::Sort(vector<int> &arr, int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        Sort(arr, low, pi - 1);
        Sort(arr, pi + 1, high);
    }
}
```

Programski kôd 2.6. *Implementacija algoritma brzo sortiranje u C++ jeziku, Qt okruženju. [6]*

Brzo sortiranje algoritam u najgorem slučaju ima složenost $O(n^2)$. Taj najgori slučaj se događa kad je pivot početni element, te je polje već sortirano. Lagano je dokazati da prosječno vrijeme izvršavanja iznosi $O(n \log n)$. Najbolji slučaj je kad pivot razdvoji niz na 2 jednaka dijela. U najboljem slučaju biti će $\log n$ podjela.

3. QT PROGRAMSKO OKRUŽENJE

Qt je besplatan i višeplatformski (engl. *Cross platform)* skup alata za stvaranje grafičkih sučelja i cross-platform aplikacija koji se mogu pokretati na raznim softverskim i hardverskim platformama kao što su Linux, Windows, MacOS, Andriod ili ugrađenim sustavima sa malo ili bez promjene u originalnom kôdu.

3.1. Qt koncepti

Qt je izgrađen na sljedećim konceptima[7]:

- Kompletna apstrakcija grafičkog sučelja – koristi aplikacijska programska sučelja platformi na kojima se koristi, na platformama koji imaju svoje *widgete* (*widget* je element grafičkog korisničkog sučelja koji prikazuje informaciju ili omogućava korisniku interakciju s aplikacijom).
- Signali i utori – koncept stvoren u *Qt*-u za komunikaciju među objektima. Koncept je da *widgeti* grafičkog sučelja mogu slati signale koji sadržavaju informacije o događaju, koje mogu primati druge kontrole koristeći specijalne funkcije koje nazivamo utori.
- *Qt* ima mogućnost korištenja s nekoliko drugih programskih jezika kao što su Python, Javascript, C#, Rust.
- Sustav meta objekata – omogućava korištenje signala i utora za komunikaciju između objekata, pruža informacije o podacima za vrijeme izvođenja te omogućava korištenje sustava dinamičkih svojstva.

3.2. Značajni alati

U sljedećim potpoglavljima objašnjeni su osnovi koncepti unutar Qt-a.

3.2.1. Sustav meta objekata

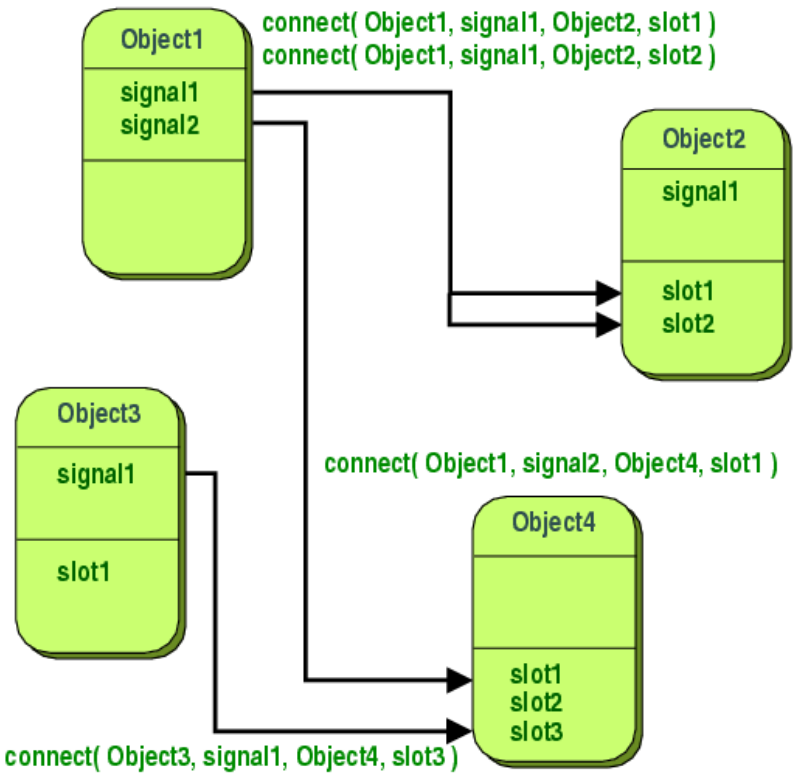
Osnova sustava meta objekata su : *QObject* klasa, *Q_OBJECT macro* naredba te prevoditelj meta objekata (engl. *Meta Object Compiler*). On nudi i uslugu provjere klasnih imena, nasljeđivanja, stvaranje objekata, prevođenje stringova na različite jezike te dohvaćanje i postavljanje svojstva objekta.

Q_OBJECT macro sadržan je u definicijama svih klasa koje koriste signale i utore ili neke druge usluge sustava meta objekata. Kad prevoditelj meta objekata u zaglavlju naiđe na ovaj *macro*, stvara datoteku koja sadrži opis klasa.[3]

QObject je osnovna klasa svih *Qt* objekata. Svi objekti u *Qt* aplikaciji imaju objekt roditelja i objekt dijete. Prilikom uništenja objekta sva djeca se uništavaju. Najlakše je vidjeti tu vezu korištenjem *QWidget* klase, koja je osnovna klasa svih elemenata koji se prikazuju na ekranu. Ako glavni prozor sadrži razne tipke i prozore za unos teksta, zatvaranjem glavnog prozora svi ostali elementi na njemu se brišu. Taj sustav se pokazao dobrim za izradu aplikacija s grafičkim korisničkim sučeljem.

3.2.2. Signali i utori

Qt za međuobjektnu komunikaciju koristi signale i utore. Jedan objekt emitira signal prilikom nekog događaja, npr. pritiskom na tipku ili upisivanje teksta. Utori (engl. *Slots*) su funkcije koje se pozivaju u slučaju emitiranja signala. Shema sustava signala i utora je prikazana na slici 3.1. [4]



Slika 3.1. *Shema sustava signala i utora[4].*

3.2.3. Qt Creator

Qt Creator je integrirano razvojno okruženje (engl. *Integrated development environment - IDE*) za izradu *Qt* aplikacija i dio je *Qt* skupa alata za softverski razvoj. *Qt editor* je alat za uređivanje teksta koji pruža korisne funkcionalnosti za olakšanje pisanja kôda, kao automatska dopuna, provjera greški u kodu tokom samog kodiranja, provjera uvlačenja teksta, nekorištene varijable i slično.

Qt Designer uređivač je alat koji se koristi za izradu grafičkog sučelja. Omogućava izuzetno lagano dodavanje novih *widgeta* jednostavnim povlačenjem s alatne trake. Dodanom *widgetu* se jednako tako lagano mogu promijeniti svojstva u prozoru koji ih sve prikazuje jednostavnim odabirom *widgeta*.

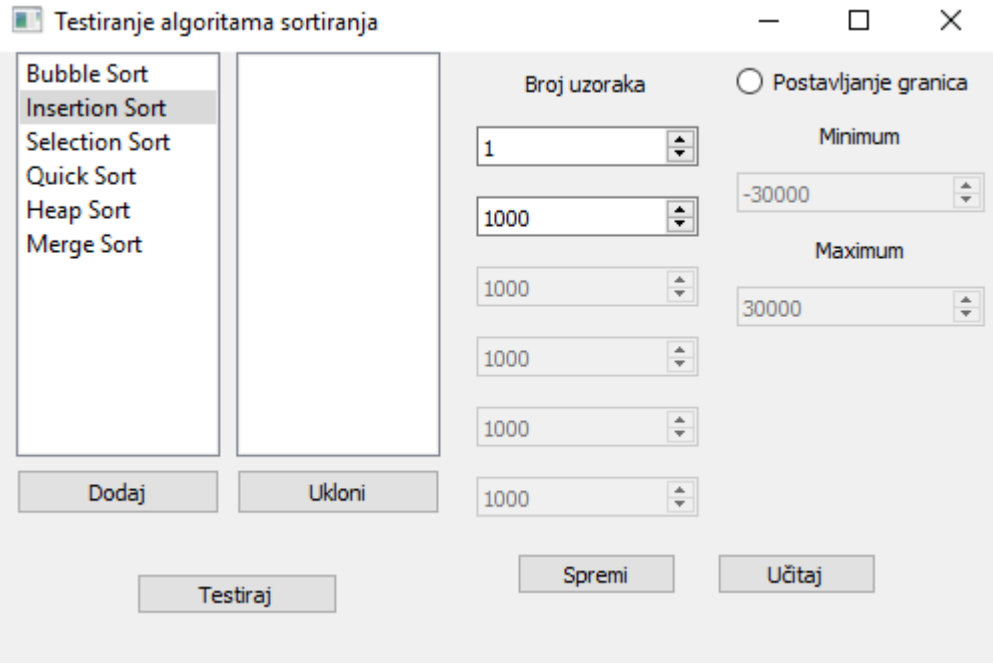
4. APLIKACIJA ZA TESTIRANJE ALGORITAMA

Aplikacija je rađena u *Qt* programskom okruženju koristeći *C++* programski jezik. Cilj je bio napraviti aplikaciju za usporedbu algoritama sortiranja te njihov grafički prikaz. Test ili testove korisnik može napraviti kakve god želi s parametrima koji su mu izloženi, to su:

- Vrste algoritama sortiranja koje želi testirati
- Broj testova kojih želi napraviti (maksimalno 5)
- Veličina uzoraka za pojedini test
- Postavljanje granica između kojih će se generirati vrijednosti za sortiranje

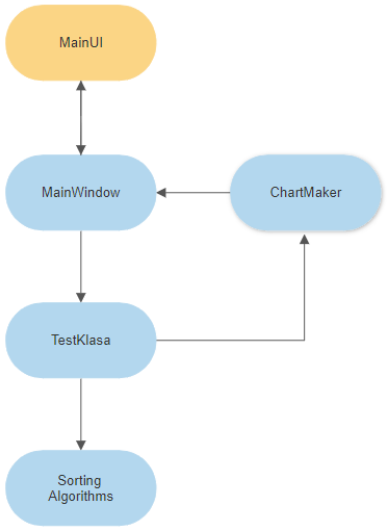
Nakon postavljanja parametara rezultati sortiranja se prikazuju korisniku u obliku stupčastog i linijskog dijagrama. Napravljene testove je moguće spremiti, te jednako tako ih ponovno učitati.

Aplikacija uz pomoć jednostavnog grafičkog sučelja te jednostavnog korištenja omogućava razne kombinacije raznih vrsta algoritama sortiranja te ulaznih parametara. Prikaz rezultata – dijagrami koji se otvore u posebnim prozorima su jednako lagani za iščitavanje te pružaju informacije o vremenu potrebnom za izvršavanje algoritama te broju operacija napravljenih prilikom izvođenja algoritama. Na slici 4.1. se može vidjeti grafičko korisničko sučelje.



Slika 4.1. *Grafičko korisničko sučelje aplikacije.*

Tok aplikacije se može vidjeti na slici 4.2. Na njemu možemo vidjeti protok informacija prilikom testiranja algoritama. S grafičkog sučelja se pozivaju metode u *MainWindow* koji prosljeđuje podatke i poziva metodu *testiraj* u klasi *TestKlasa*. *TestKlasa* pokreće i mjeri algoritme sortiranja, nakon toga rezultate šalje u *ChartMaker* koji stvori potrebne dijagrame, pošalje ih *MainWindow* te se onda prikažu.



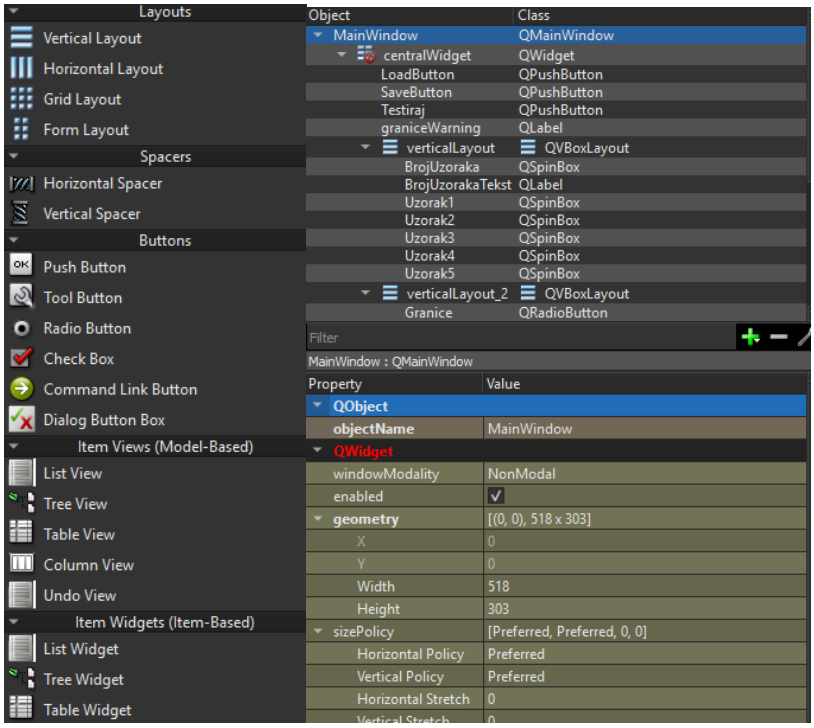
Slika 4.2. *Prikaz toka aplikacije.*

4.1. Izrada aplikacije

Aplikacija je izrađena u *Qt* programskom okruženju, u besplatnoj verziji. Aplikacija je podijeljena na dva dijela :

- Vidljivi dio (grafički dio aplikacije s kojim korisnik ima interakciju)
- Funkcionalni dio (sadrži logiku aplikacije, zadužen za sortiranje i obradu podataka)

Oba dijela su definirana u programskom jeziku C++. Vidljivi dio je generirao *Qt* tako da jednostavnim povlačenjem i ispuštanjem, stvorimo i organiziramo grafičke elemente koji su nam potrebni. Te elemente možemo grupirati po rasporedu *(eng. Layout)* te im postavljati/mijenjati svojstva u izborniku koji nam *Qt* pruža (Slika 4.3.).

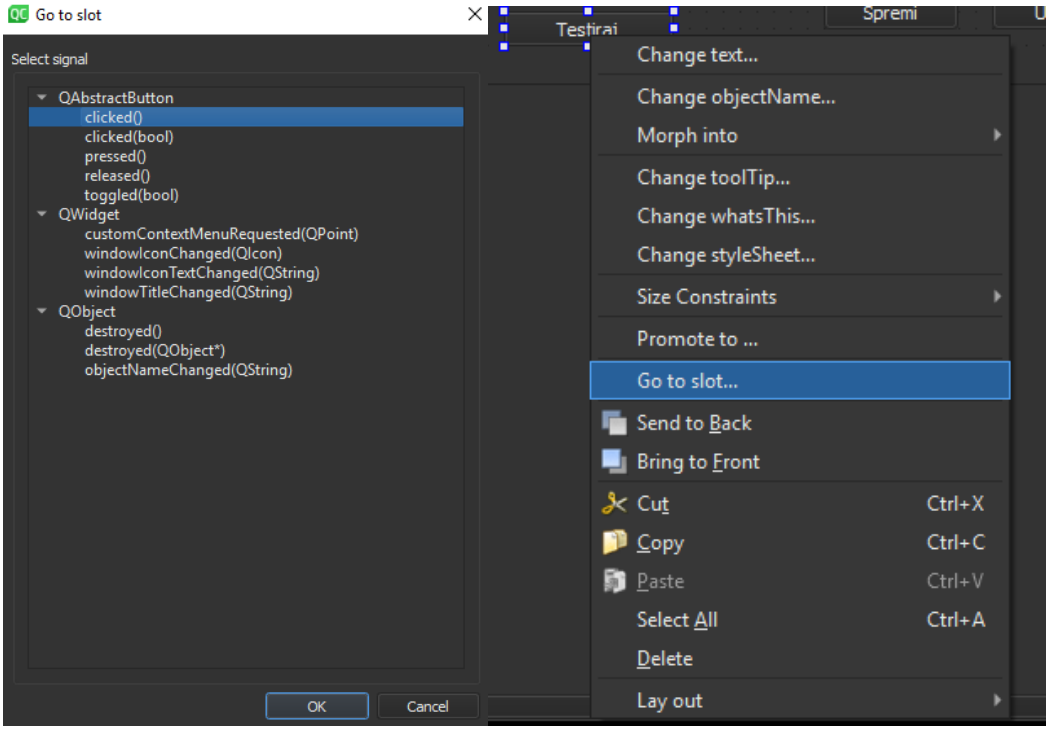


Slika 4.3. *Izbornik widgeta i prozor sa svojstvima.*

Odabirom elementa kojem želimo mijenjati svojstva nam se prikažu svojstva ovisna o elementu gdje možemo postaviti njegovu poziciju (apsolutnu, relativnu), po zadanom postavljene vrijednosti, naputak, font i slično.

Nakon postavljanja i organiziranja željenih grafičkih elemenata ih treba povezati s funkcionalnim dijelom, funkcionalni dio ima pristup svojstvima tih elemenata te ih može mijenjati. Za postavljanje ponašanja grafičkog elementa *Qt* pruža jednostavnu opciju stvaranja utora koji su poveznica vizualnog i funkcionalnog dijela (Slika 4.4.).

Desnim klikom na *widget* te odabirom „*Go to slot...*“ otvori se prozor koji nam daje opcije u kojim okolnostima želimo da se utor pozove, kakva interakcija s *widgetom* će ga pozvati. Generirani utori imaju opisna imena, ime *widgeta* koji ga poziva, te koja akcija ga poziva (Programski kôd 4.1.).

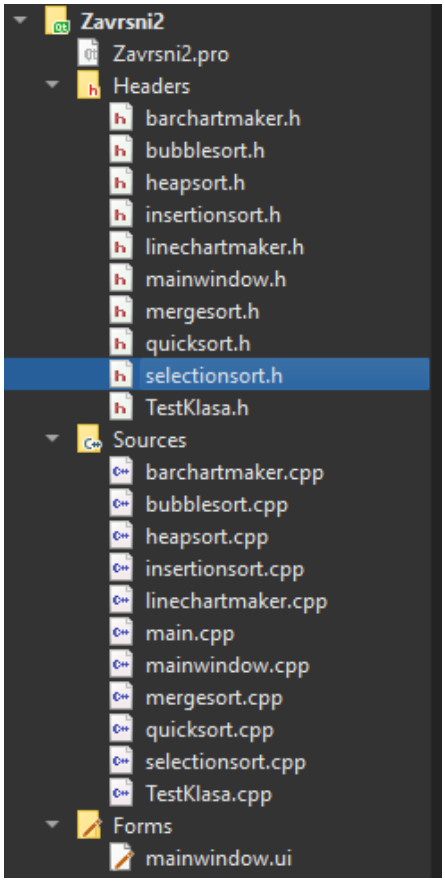


Slika 4.4. *Stvaranje utora za widgete.*

```
MainWindow::~MainWindow() {...}

void MainWindow::on_Testiraj_clicked() {...}

void MainWindow::on_Granice_toggled(bool checked) {...}

void MainWindow::on_BrojUzoraka_valueChanged(int arg1) {...}

void MainWindow::on_Minimum_valueChanged(int arg1) {...}
void MainWindow::on_Maksimum_valueChanged(int arg1) {...}

void MainWindow::on_pushButton_clicked() {...}

void MainWindow::on_pushButton_2_clicked() {...}

void MainWindow::on_SaveButton_clicked() {...}

void MainWindow::on_LoadButton_clicked() {...}
```

Programski kôd 4.1. *Stvoreni utori*

Funkcionalni dio aplikacije je organiziran po klasama te pripadajućim datotekama zaglavlja (Slika 4.5.), klase sadrže logiku dok datoteke zaglavlja sadrže deklaracije i uključivanje biblioteka te drugih datoteka zaglavlja koje se koriste u klasi.



Slika 4.5. *Organizacija datoteka u projektu.*

Klase služe za grupiranje funkcija i svojstava koji su logički povezani radi lakšeg održavanja te popravljanja kôda. Program kreće od *main.cpp* klase unutar kojeg se pokreće aplikacija te stvara početni prozor *mainwindow.cpp*. *Mainwindow* klasa je zadužena za inicijalizaciju vrijednosti koje se koriste u aplikaciji te sadrži *utore* koje aplikacija koristi.

Najbitnija klasa je *mainwindow.cpp* (Programski kôd 4.2.) koja sadrži sve utore *widgeta* te stvara glavni prozor. Prilikom pokretanja programa klasa *main.cpp* ju poziva te ona postavlja glavni prozor kao i naknadno definirane signale i utore. Registriraju se korisnički stvoreni tipovi podataka tipovi podataka kako bi se ti podatci mogli se koristit između niti bez opasnosti oštećivanja podataka.

U klasi *mainwindow.cpp* se nalaze svi utori, najbitniji utor *on_testiraj_clicked* je zadužen za provjeru granica te je li izabran algoritam.

```
int maksimum = 30000;
int minimum = -30000;
int brojUzoraka = 1;
bool granice = false;
bool bubbleSort = false;
bool heapSort = false;
bool insertionSort = false;
bool mergeSort= false;
bool quickSort = false;
bool selectionSort = false;


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    qRegisterMetaType<boolVector>();
    qRegisterMetaType<intVector>();
    qRegisterMetaType<arrayOfArrays>();

    connect(&testirajSort,SIGNAL(sendChart(arrayOfArrays,intVector,int,boolVector, arrayOfArrays)),
            this, SLOT(showChart(arrayOfArrays,intVector,int,boolVector,arrayOfArrays)));
    ui->setupUi(this);
    QStringList items;
    items << "Bubble Sort" << "Insertion Sort"<<"Selection Sort"<<"Quick Sort"<<"Heap Sort"<<"Merge Sort" ;
    ui->addSortList->addItems(items);
    ui->addSortList->setCurrentRow( 1 );
    ui->graniceWarning->hide();
    ui->working->hide();

}
```

Programski kôd 4.2. *Dio klase mainwindow.cpp.*

On također i skuplja podatke iz ostalih elemenata te ih priprema za slanje klasi u kojoj se obavlja sortiranje. Nakon provjere ispravnosti, te pripreme podataka asinkrono poziva funkciju *testiraj* (Programski kôd 4.3.) iz klase *TestKlasa.cpp* koja je zadužena za generiranje uzoraka te pokretanje algoritama sortiranja.

```
void KlasaTestiraj::testiraj(boolVector sortovi, int minimum, int maksimum, intVector velicinaUzoraka, int brojUzoraka){
    vector<vector<int>> uzorci(brojUzoraka);

    for(int i=0;i<brojUzoraka;i++){
        vector<int> uzorak(velicinaUzoraka[i]);
        uzorci[i] = uzorak;
    }

    uzorci[0] = RandomPolje(minimum,maksimum, velicinaUzoraka[0], uzorci[0]);
    if(brojUzoraka>1)uzorci[1] = RandomPolje(minimum,maksimum, velicinaUzoraka[1], uzorci[1]);
    if(brojUzoraka>2)uzorci[2] = RandomPolje(minimum,maksimum, velicinaUzoraka[2], uzorci[2]);
    if(brojUzoraka>3)uzorci[3] = RandomPolje(minimum,maksimum, velicinaUzoraka[3], uzorci[3]);
    if(brojUzoraka>4)uzorci[4] = RandomPolje(minimum,maksimum, velicinaUzoraka[4], uzorci[4]);

    QElapsedTimer timer;

    arrayOfArrays svaVremena;

     arrayOfArrays sveOperacije = {{{0}}} ;

    if(sortovi[0]){
         vector<int> BubbleArray;
         BubbleSort *bubbleSort = new BubbleSort;
         for(int i=0;i<brojUzoraka;i++){
         BubbleArray = uzorci[i];
         timer.restart();
         bubbleSort->Sort(BubbleArray,velicinaUzoraka[i], &sveOperacije[0][i]);
         svaVremena[0][i]=timer.nsecsElapsed()/1000;

     }
    }
```

Programski kôd 4.3. *Dio metode testiraj.*

Metoda testiraj generira polja veličine uzoraka te ih zatim popunjava pozivom metode *RandomPolje* (Programski kôd 4.4.) kojoj proslijedi prazno polje te granice uzorka. Provjerava koji algoritmi su odabrani, potom svaki od algoritama sortiranja mjeri, koristeći *Qt*-ovu biblioteku *QElapsedTimer* koji mjeri vrijeme od njegovog pokretanja do poziva metode *nsecsElapsed*, te vrijednosti vremena se spremaju o polje polja koje se koristi kasnije za izradu dijagrama. Broj operacija se mjeri tako da se proslijedi referenca na varijablu *sveOperacije* koja je isto polje polja te se odgovarajuće povećava za svaku operaciju izvedenu unutar algoritma sortiranja.

```
vector<int> KlasaTestiraj::RandomPolje(int minimum, int maksimum, int velicina, intVector array){

        QRandomGenerator *generator = new QRandomGenerator();

        for(int i=0; i<velicina; i++){
            array[i] = generator->bounded(minimum,maksimum);
     }
    return array;

}
```

Programski kôd 4.4. *Metoda RandomPolje.*

Nakon izvedenih svih potrebnih sortiranja te spremanja vrijednosti vremena izvođenja i broja operacije odašilje se signal iz klase *TestKlasa.cpp* nazad na klasu *mainwindow.cpp.* Taj signal/utor *sendChart/showChart* je povezan pozivom *connect* u stvaranju *mainwindow* klase što se može vidjeti na slici 4.6. Nakon poziva tog signala na *mainwindow* klasi se izvršava metoda *showChart* (Programski kôd 4.5) koja prima odašiljane podatke, stvara potrebne klase *BarChartMaker* i *LineChartMaker*, poziva metode *ChartMaker* te kao rezultat dobiva *QChartView* kojima postavlja svojstva te ih prikazuje.

```
void MainWindow::showChart(arrayOfArrays svaVremena,intVector velicinaUzoraka,int brojUzoraka ,bo
    BarChartMaker *a = new BarChartMaker();
    LineChartMaker *b = new LineChartMaker();

    QChart *barChart = a->ChartMaker(svaVremena,velicinaUzoraka, brojUzoraka, sortovi);
    QChart *lineChart = b->ChartMaker(sveOperacije,velicinaUzoraka, brojUzoraka, sortovi);

    QChartView *barChartView = new QChartView(barChart);
    QChartView *lineChartView = new QChartView(lineChart);
    barChartView -> setRenderHint(QPainter::Antialiasing);
    lineChartView -> setRenderHint(QPainter::Antialiasing);
    barChartView->resize(800,800);
    lineChartView->resize(800,800);
    barChartView->setWindowTitle("Vremena sortova");
    lineChartView->setWindowTitle("Broj operacija sortova");
    ui->working->hide();
    barChartView->show();
    lineChartView->show();
}
```

Programski kôd 4.5. *Metoda showChart.*

Klasa *BarChartMaker* (Programski kôd 4.6.) u sebi sadrži logiku potrebnu za stvaranje stupčastog dijagrama. Stvori dijagram *QChart*, te *QBarSeries* koji predstavlja podatke za stupčasti dijagram. Stvara *x* i *y* osi te postavlja potrebna svojstva. Iterira nad podatcima koji sadrže vremena izvođenja dijagrama te ih dodaje u *QBarSet* koji predstavlja jedan skup podataka. Nakon stvaranja svih potrebnih skupova dodaje ih u *QChart* te novo stvoreni dijagram vraća kao povratnu vrijednost, nakon toga u *mainwindow* klasi se prikaže u novom prozoru.

```
QChart* BarChartMaker::ChartMaker(std::array<std::array<long,5>,6> svaV

    QStringList categories;
    QString imena[6]={
        "Bubble Sort",
        "Heap Sort",
        "Insertion Sort",
        "Merge Sort",
        "Quick Sort",
        "Selection Sort"
    };

    for (int i = 0;i<brojUzoraka;i++) {
        categories.append(QString::number(i+1));
    }

    QChart *chart = new QChart();
    QBarSeries *series = new QBarSeries();
    QBarCategoryAxis *axisX = new QBarCategoryAxis();
    axisX->append(categories);
    chart->addAxis(axisX, Qt::AlignBottom);
    QValueAxis *axisY = new QValueAxis();
    axisY->setLabelFormat("%.0f");
    axisX->setTitleText("Redni broj uzorka");
    axisY->setTitleText("Vrijeme sortiranja u mikro sekundama");
    chart->addAxis(axisY, Qt::AlignLeft);
    int max = 0;
    int min = INT_MAX;

    for (int i = 0;i<6;i++) {
        if(sortovi[i]){
            QBarSet *set = new QBarSet(imena[i]);
            for (int j=0;j<brojUzoraka;j++) {
                if(max<svaVremena[i][j]) max = svaVremena[i][j];
                if(min>svaVremena[i][j]) min = svaVremena[i][j];
                 set->append(svaVremena[i][j]);
            }
            series->append(set);
        }
    }
    axisY->setRange(0,max);
    chart -> addSeries(series);
    series->attachAxis(axisX);
    series->attachAxis(axisY);
    chart->legend()->setVisible(true);
    chart->legend()->setAlignment(Qt::AlignBottom);

    return chart;
};
```

Programski kôd 4.6. *Metoda ChartMaker u klasi BarChartMaker.*

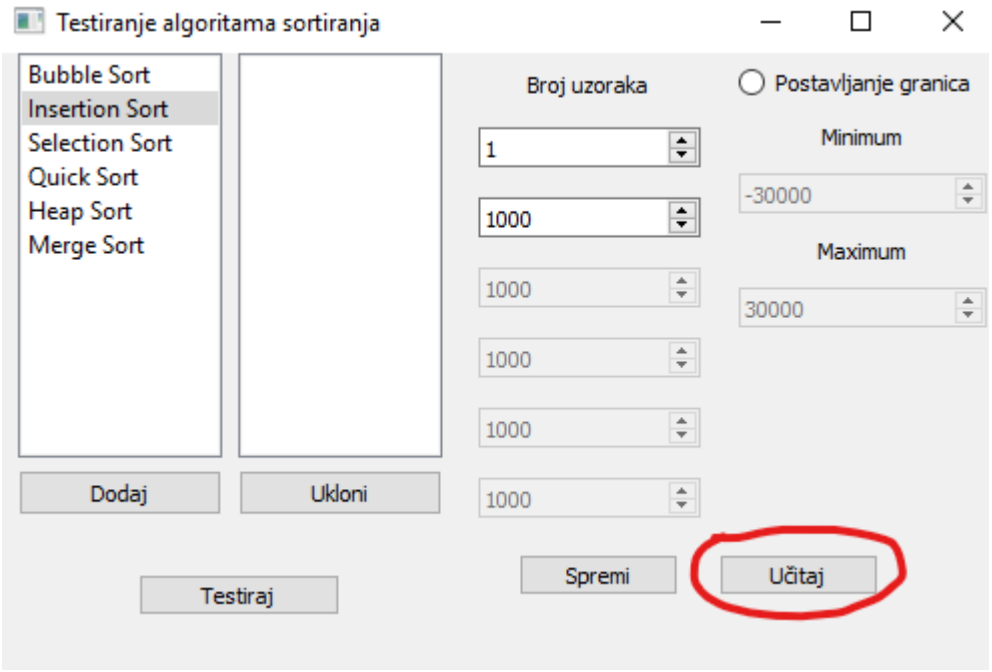
Klasa *LineChartMaker* (Programski kôd 4.7.) u sebi sadrži logiku potrebnu za stvaranje linijskog dijagrama, stvori chart, te *QLineSeries* koji predstavlja podatke za linijski dijagram. Stvara X i Y osi te postavlja potrebna svojstva. Iterira nad podatcima koji sadrže vremena izvođenja algoritama sortiranja te ih dodaje u *QLineSeries*. Nakon prolaska kroz sve potrebne podatke stvoreni dijagram vraća kao povratnu vrijednost koji se zatim prikaže u novom prozoru.

```
QChart* LineChartMaker::ChartMaker(std::array<std::array<long,5>,6> sveOperacije,st

    QString imena[6]={
        "Bubble Sort",
        "Heap Sort",
        "Insertion Sort",
        "Merge Sort",
        "Quick Sort",
        "Selection Sort"
    };

    QChart *chart = new QChart();
    QValueAxis *axisX = new QValueAxis();
    axisX->setTitleText("Veličina uzorka");
    chart->addAxis(axisX, Qt::AlignBottom);
    QValueAxis *axisY = new QValueAxis();
    axisX->setLabelFormat("%.0f");
    axisY->setLabelFormat("%.0f");
    axisY->setTitleText("Broj operacija");
    chart->addAxis(axisY, Qt::AlignLeft);
    int max = 0;
    int min = INT_MAX;
    for (int i = 0;i<6;i++) {
        if(sortovi[i]){

            QLineSeries *lineseries = new QLineSeries();
            lineseries->setPointsVisible();
            lineseries->setName(imena[i]);
            lineseries->append(0,0);
            for (int j=0;j<brojUzoraka;j++) {
                if(max<sveOperacije[i][j]) max = sveOperacije[i][j];
                if(min>sveOperacije[i][j]) min = sveOperacije[i][j];
                lineseries->append(QPoint(velicinaUzoraka[j],sveOperacije[i][j]));
            }

            chart->addSeries(lineseries);
            lineseries->attachAxis(axisX);
            lineseries->attachAxis(axisY);
        }
    }
    axisY->setRange(0,max + max*0.05);

    chart->legend()->setVisible(true);
    chart->legend()->setAlignment(Qt::AlignBottom);

    return chart;
};
```

Programski kôd 4.7. Metoda *ChartMaker u klasi LineChartMaker.*

4.2. Rad s aplikacijom

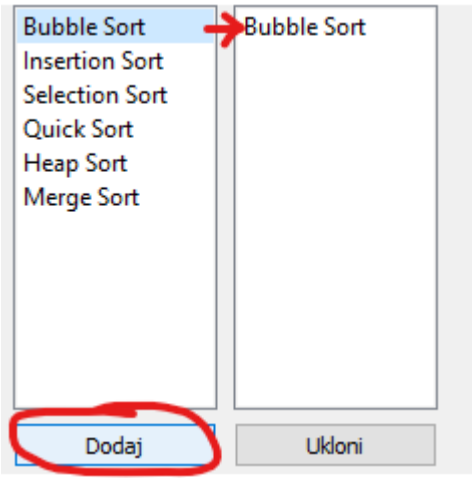
Prilikom pokretanja aplikacije stvori se glavni prozor sa svojim elementima, korisnik može učitati podatke iz *.txt* datoteke koja mora bit u specifičnom obliku kako bi se parametri sortiranja mogli pravilno ispuniti. Klikom na tipku "*Učitaj*" se otvori prozor za odabir datoteke . Na slici 4.6. se vidi početni zaslon aplikacije.



Slika 4.6. *Početni zaslon te naznačenu tipku „Učitaj".*

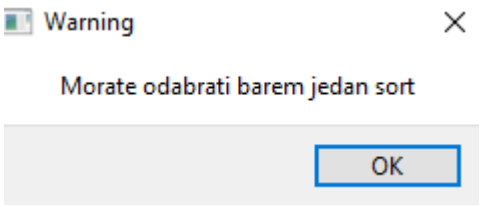
Ako se podaci iz vanjske datoteke uspješno učitaju, korisnik može pokrenuti test i vidjeti rezultate, ali može i promijeniti učitane parametre. Ako se korisnik odluči spremiti parametre sortiranja pritiskom na tipku „*Spremi*" se otvori prozor za spremanje datoteke, datoteka se sprema u *.txt* formatu.

Ukoliko korisnik želi postaviti svoje parametre ili izmijeniti postojeće, može odabrati koji algoritmi sortiranja će se izvršavati i mjeriti dodavanjem i uklanjanjem iz liste. Pritiskom na algoritam koji želimo dodati te na tipku „*dodaj*" algoritam se pojavi u prozoru pored koji predstavlja listu algoritama koji će se izvršiti. Klikom na element iz liste algoritama za izvršavanje te pritiskom na tipku „*Ukloni*" se algoritam briše iz liste te se neće izvršiti. Na slici 4.7. se vidi postupak dodavanja jednog od sortova.



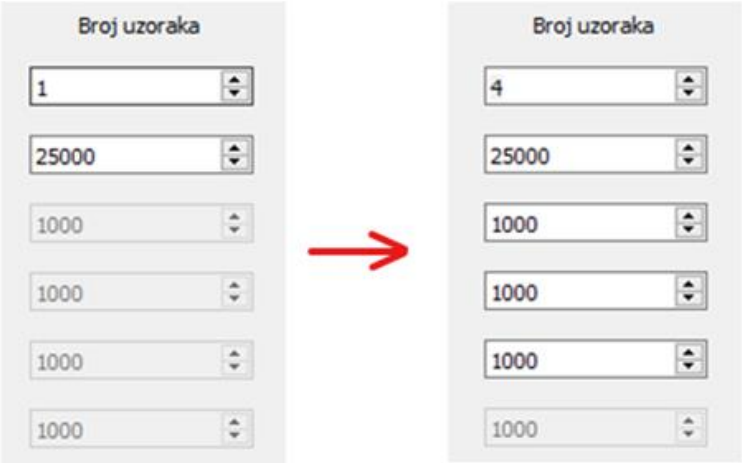
Slika 4.7. *Dodavanje sorta u listu za izvršavanje.*

U slučaju da nije odabran niti jedan algoritam te se testiranje pokuša pokrenuti izbaciti će se poruka upozorenja koja obavještava korisnika da mora biti odabran barem jedan algoritam za sortiranje prije testiranja (Slika 4.8.).



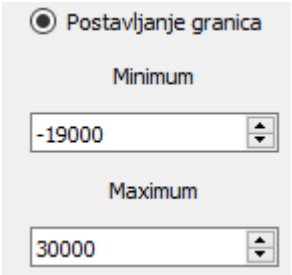
Slika 4.8. *Upozorenje za odabir barem jednog algoritma.*

Nakon izabira algoritama koji će se testirati korisnik ima opciju odabrati broj uzoraka za koji će se testirati te veličinu samih uzoraka. Minimalni broj uzoraka za testiranje je 1, a maksimalni 5. Veličina uzoraka je ograničena između 1.000 elemenata i 5.000.000 elemenata, svi unosi manji od 1000 se automatski postave na minimalnu vrijednost, a veći od 5.000.000 se automatski postave na maksimalnu vrijednost. Preporuča se postavljanje relativno bliskih vrijednosti radi lakšeg pregleda i interpretacije rezultata, prevelika razlika napravi testove s malom veličinom uzoraka nebitnima. Kako korisnik mijenja broj uzoraka za koje želi provoditi testove, tako se polja za unos veličine pojedinih uzoraka uključuju/isključuju (Slika 4.9.).



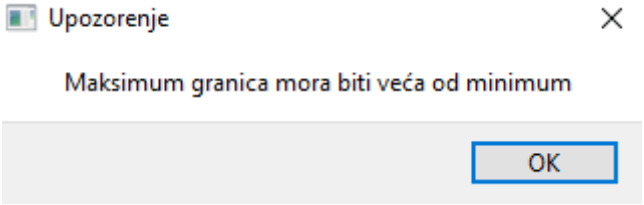
Slika 4.9. *Prikaz mijenjanja broja uzoraka.*

Korisnik može postaviti granice između kojih će se generirati elementi polja za sortiranje (Slika 4.10.). S isključenim granicama se generiraju vrijednosti između -30.000 i 30.000.



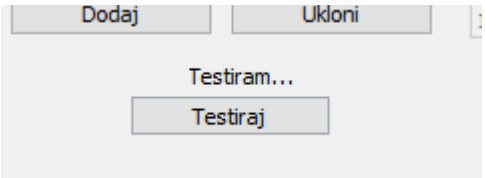
Slika 4.10. *Uključivanje postavljanja granica*

Prilikom postavljanja granica, obje vrijednosti su ograničene između -30.000 i 30.000, te minimum ne smije biti veći ili jednak maksimumu inače prilikom pokretanja testiranja korisnik dobije upozorenje (Slika 4.11.).



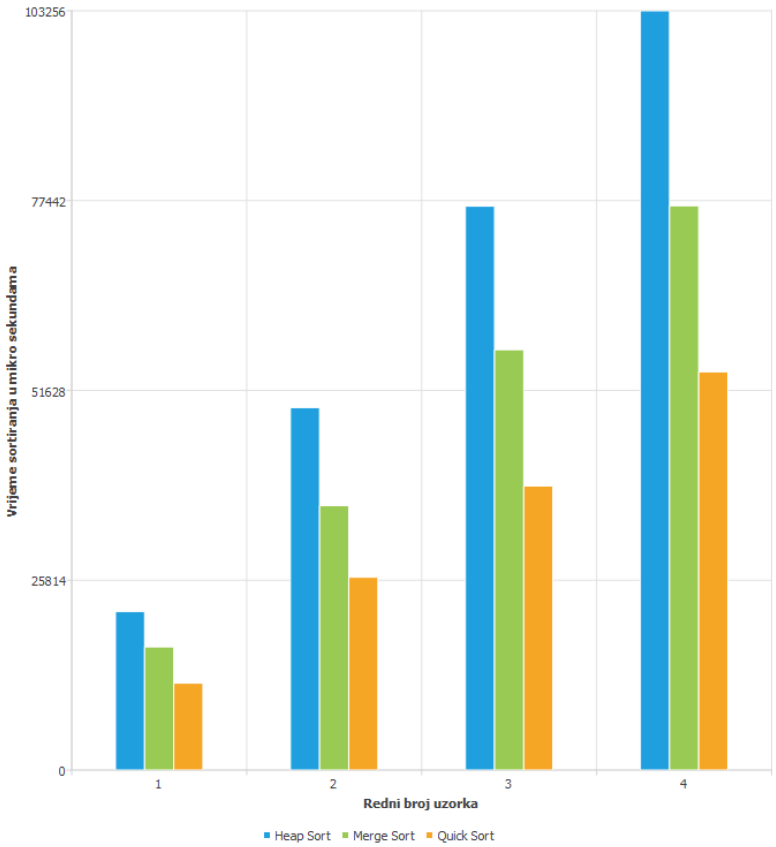
Slika 4.11. *Upozorenje za neispravne granice.*

Nakon unošenja ispravnih parametara korisnik može spremiti postavke na ranije opisani način i/ili testirati odabrane algoritme prema zadanim parametrima pritiskom na tipku „*Testiraj*“. Dok se testiranje izvodi prozor je interaktivan te korisnik može urediti parametre za sljedeće testiranje, tokom izvođenja iznad tipke se prikazuje tekst „*Testiram...*“ (Slika 4.12.).



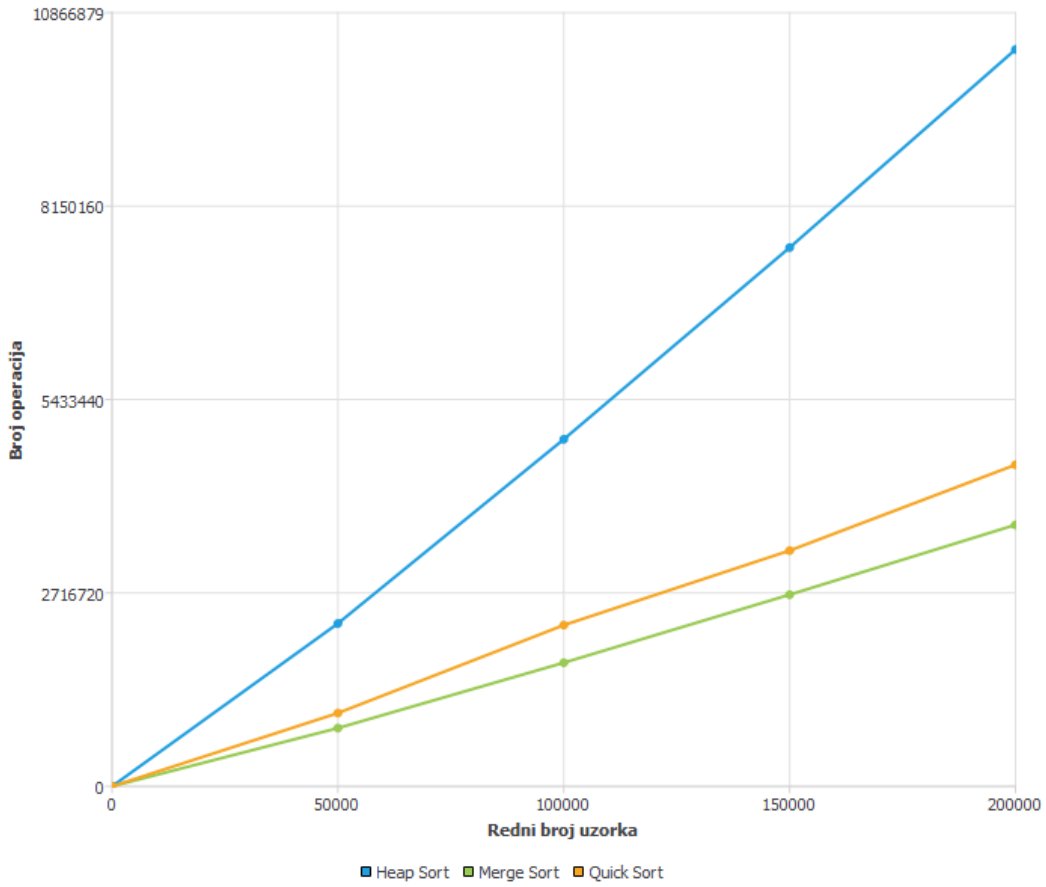
Slika 4.12. *Tipka za testiranje.*

Prilikom završetka testiranja, program izbacuje 2 nova prozora koji sadrže dijagrame koji imaju informacije o provedenim testiranjima. Jedan prozor sadrži stupčasti dijagram, prikazan na slici 4.13.



Slika 4.13. *Stupčasti dijagram.*

Stupčasti dijagram prikazuje vrijeme potrebno za izvršenje pojedinog algoritma za pojedini uzorak. Na *y* osi se nalazi vrijeme potrebno za izvršavanje algoritma u mikrosekundama. Na *x* osi je redni broj algoritma, svaki od algoritama prikazan je bojom koja mu je pridružena na legendi na dnu dijagrama. Uz prozor stupčastog dijagrama otvori se i prozor linijskog dijagrama, prikazan na slici 4.14.



Slika 4.14. *Linijski dijagram.*

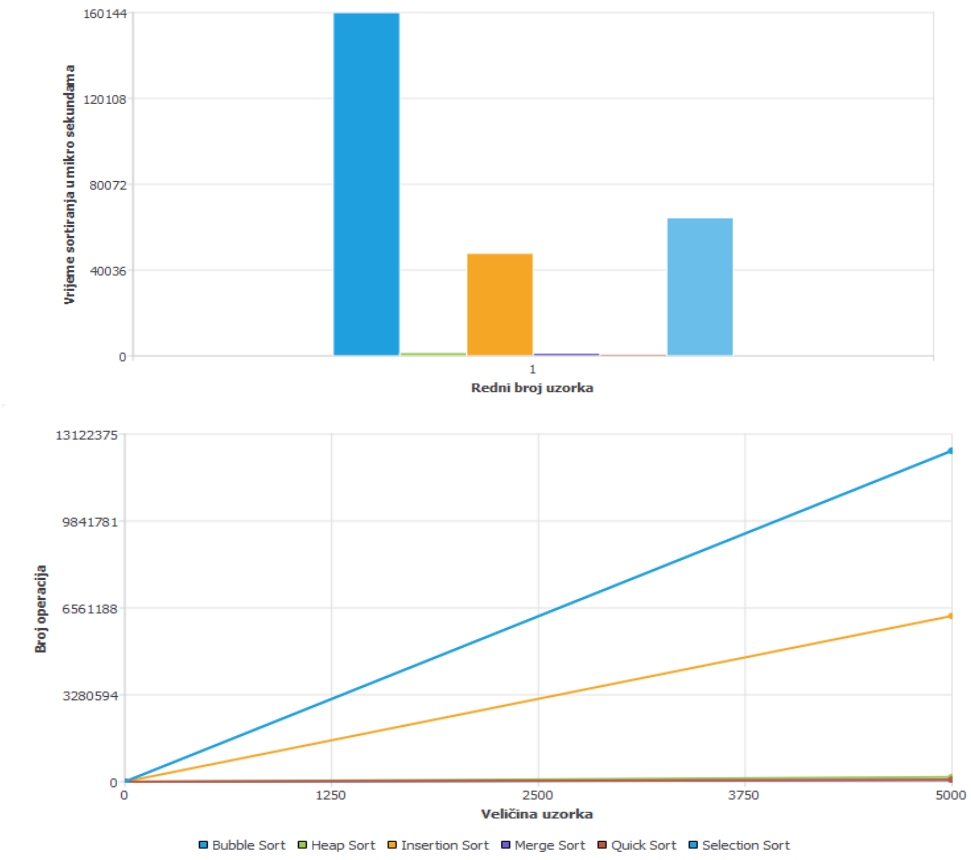
Linijski dijagram prikazuje broj izvedenih operacija u ovisnosti o veličini uzorka. Na osi *y* je broj operacije izvedenih prilikom izvođenja algoritma sortiranja, a na osi *x* je veličina uzoraka. Linija svakog algoritma je kodirana bojom, legenda boja je na dnu prozora. Točke na linijama označavaju vrijednosti uzoraka za koje se algoritam izvodio.

4.3. Rezultati testiranja

Za prikaz razlika između pojedinih algoritama za sortiranje napravljene su neke usporedbe rezultata dobivenih testiranjem algoritama s određenim parametrima. Testiranja su provedena na računalo sa sljedećom konfiguracijom:

- Procesor: Intel Core™ i5-6500 Processor, 6M Cache, up to 3.20 GHz
- Radna memorija: 16GB DDR4 2133MHz
- Grafička kartica: Radeon RX 580 8GB GDDR5
- Pohrana podataka**:** ADATA Premier SP550 SSD 240GB

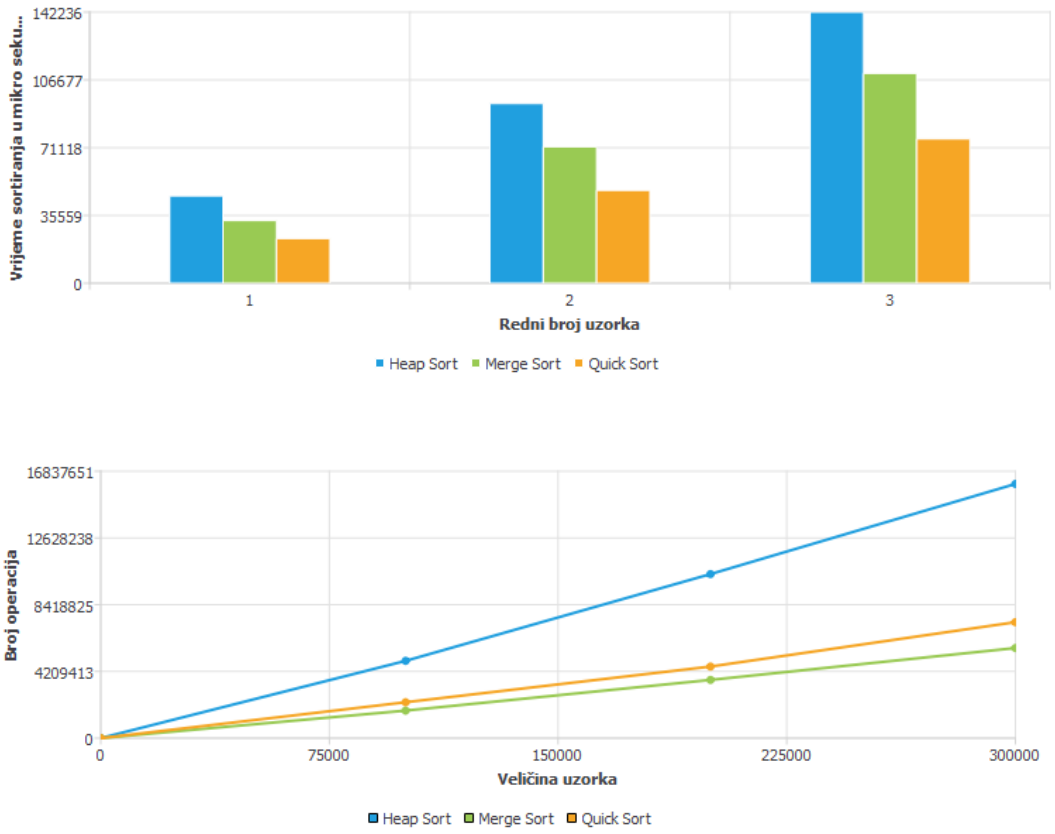
U prvom testu su uključeni svi algoritmi sortiranja, postavljen je jedan uzorak veličine 5000 elemenata. Granice vrijednosti generiranih elemenata su ostavljene na vrijednostima od -30.000 do 30.000. Rezultati testiranja se mogu vidjeti na slici 4.15.



Slika 4.15. *Prikaz rezultata sortiranja svih algoritmima s uzorkom od 5000 elemenata.*

U prvom testu jasno vidimo brzinu sortiranja između brzih algoritama (brzo sortiranje, sortiranje spajanjem i sortiranje pomoću hrpe) te sporih (mjehuričasto sortiranje, sortiranje umetanjem, sortiranje izborom), te ih nema puno smisla uspoređivati jer ih je teško prikazati smisleno. Mjehuričasto sortiranje je bio najsporiji algoritam sa 159 milisekundi, a najbrži algoritam je brzo sortiranje sa 1 milisekundi, znači 159 puta brži. Brzo sortiranje je za 5000 elemenata najbrži brzi sort, a sortiranje umetanjem najbrži spori sort.

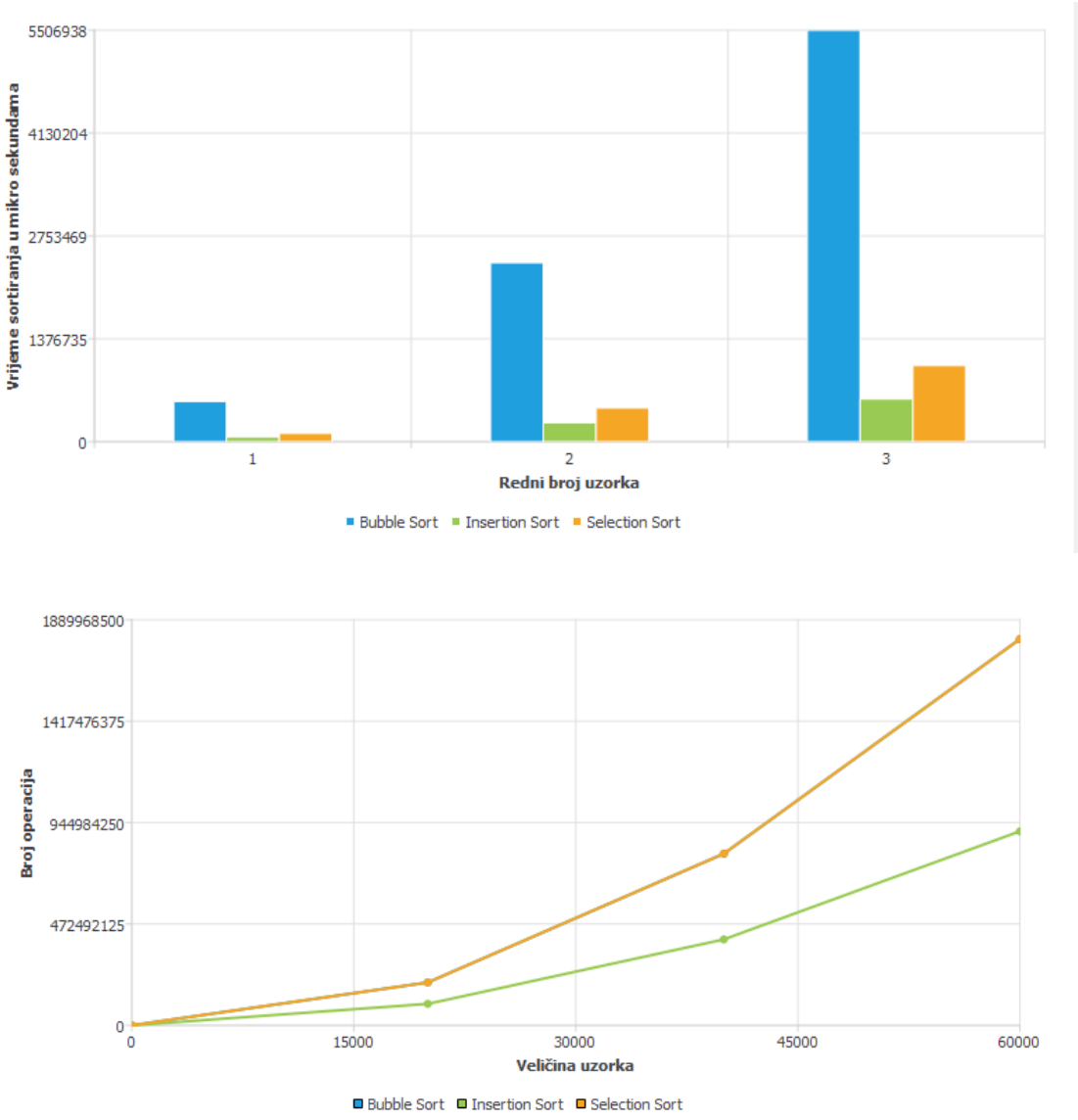
Drugi test uključuje brze algoritme, 3 uzorka veličine 100.000, 200.000 i 300.000 tisuća elemenata, granice od -30.000 do 30.000. Rezultati se mogu vidjeti na slici 4.16.



Slika 4.16. *Prikaz rezultata sortiranja brzih sortova s 3 uzorka različitih veličina.*

Na dijagramu vidimo da je brzo sortiranje najbrži algoritam sortiranja za zadane parametre, ali se izvrši u većem broju operacija od sortiranje spajanjem. Brzo sortiranje sortira uzorak od 100.000 elemenata u 22 ms, 200.000 u 46ms, 300.000 u 73ms. Ostali algoritmi su relativno blizu po vremenu izvođenja, iako su jednake složenosti, konstanta im je drugačija.

U trećem testu uključeni su spori algoritmi. Test je proveden za 3 uzorka veličina 20.000, 40.000 i 60.000, te su granice od -30.000 do 30.000. Rezultati se vide na slici 4.17.



Slika 4.17. *Prikaz rezultata sortiranja sporih sortova s 3 uzorka.*

Iz rezultata testiranja vidimo da je sortiranje umetanjem najbrži od sporih algoritama dok je mjehuričasto sortiranje daleko najsporiji. Za uzorak od 60.000 elemenata sortiranje umetanjem treba 950 ms dok mjehuričasto sortiranje treba 5500 milisekundi. Sortiranje umetanjem isto tako zahtjeva duplo manji broj operacija dok mjehuričasto sortiranje i sortiranje izborom trebaju jednak broj.

5. ZAKLJUČAK

Algoritmi sortiranja su značajni u informacijskom svijetu, često ih se ne primjećuje no olakšavaju snalaženje u velikom broju podataka. U programskim okruženjima koje imaju ugrađene funkcije za sortiranje najčešće se primjenjuje kombinacija algoritama za sortiranje, što znači da ne postoji jedan najbolji. Ovisno o ulaznim parametrima, specifikacijama računala različiti algoritmi poprimaju različite vremenske složenosti. Prilikom testiranja kao i očekivano vidi se da su „spori“ algoritmi višestruko sporiji od „brzih“ algoritama, no prednost im je što su veoma lagani za shvatiti te je njihova implementacija puno jednostavnija. Istovremeno testiranje svih algoritama koristeći aplikaciju nije praktično jer su razlike među njima dovoljno velike da ih je grafički teško prikazati na istom rasponu. „Brzi“ algoritmi svi imaju relativno sličnu duljinu izvođenja, zajedničko im je da sortiranje vrše korištenjem rekurzije, ali su i među njima implementacije te količina memorije koja im je potrebna različite. Kao poboljšanje aplikacije moguće je napraviti detaljnije dijagrame, proširiti dozvoljene veličine parametara, omogućiti izvođenje više testova u slučaju da korisnik želi sortirati velike uzorke. Jedno od mogućih poboljšanja bi bilo dodavanje vizualizacije sortiranja u pravom vremenu.

Qt okruženje je veoma lagano za upotrebu te ima veliki broj alata za lakšu izradu višeplatformskih aplikacija. Za lakše korištenje treba posjedovati adekvatne vještine *C++* jezika kao i načela na kojima se temelji. Kod *Qt* okruženja prepreka za učinkovito korištenje je potreba za dobrim poznavanjem ugrađenih biblioteka i alata koji nam mogu značajno smanjiti kôd koji je potrebno napisati te samim time omogućiti izradu kvalitetnijeg programa. *Qt* sadrži veliki broj alata koji se godinama aktivno razvijaju i nadograđuju, time što je otvorenog izvora (engl. *Open source*).

LITERATURA

[1]Oljica, M. (2014). *Vizualizacija osnovnih algoritama za sortiranje*.
http://mapmf.pmfst.unist.hr/~ani/radovi/zavrsni/Oljica_Mate_zavrsni.pdf

[2]Shaffer, C.A. (2010). *A Practical Introduction to Data Structures and Algorithm Analysis.*
Preuzeto s https://people.cs.vt.edu

[3]*The Meta-Object System, Qt Documentation,*
https://doc.qt.io/qt-5/moc.html 01.07.2019

[4]*Signals & slots, Qt Documentation* stranice,
https://doc.qt.io/qt-5/signalsandslots.html 01.07.2019

[5]Manger, Robert; Marušić, Miljenko. *Strukture podataka i algoritmi.*
Zagreb: Prirodoslovno matematički fakultet, 2007.

[6]Algoritmi za sortiranje. URL : https://www.geeksforgeeks.org 01.07.2019

[7]Veseli, Robert. *Izrada aplikacije koristeći Qt skup programskih alata*.
Osijek: Fakultet elektrotehnike računarsva i informacijskih tehnologija, 2017.

SAŽETAK

Naslov: *Qt* GUI aplikacija za testiranja algoritama sortiranja

Kao dio završnog rada je napravljena *Qt* GUI aplikacija za testiranje algoritama sortiranja koja prikazuje potrebno vrijeme, te broj operacija za izvršavanje nekog algoritma za određene ulazne podatke. Programski kôd je pisan u jeziku C++ i korištene su biblioteke ugrađene u *Qt* programsko okruženje. Grafički dio je izrađen pomoću *Qt* designera s kojim su ujedno povezani kôd i grafičko sučelje. U poglavljima je pružen pregled nekih jednostavnijih algoritama koji sortiraju samo iteracijom, te složenijih koji koriste i rekurzije. Opisana je oznaka kojom definiramo vremensku kompleksnost algoritma te što to znači. Sadrži dio s uputama kako koristiti aplikaciju i opisane su sve mogućnosti aplikacije.

Ključne riječi: algoritmi sortiranja, *C++*, *GUI*, sortiranje, *Qt*

ABSTRACT

Title: *Qt GUI* application for testing sorting algorithms

As a part of graduation paper, a *Qt GUI* application for testing sorting algorithms was created. The application shows time and number of operations needed for execution of the sorting algorithm. Program code was written in *C++* programming language and uses libraries from *Qt* integrated development environment. Graphical interface was created using *Qt designer* which is used to connect *GUI* and the code. In the chapters of the paper, an overview is given of some of the simpler sorting algorithms which use iteration and some of more complex ones which use recursion. Time complexity and system of marking algorithms is described. Paper also contains instructions on how to use the application.

Key words: *C++*, *GUI*, sorting, sorting algorithms, *Qt*

ŽIVOTOPIS

Vlatko Klen, rođen 27. Studenog 1996. godine u Osijeku gdje je i odrastao, te tamo trenutno živi. Školovanje je započeo 2003. godine u Osnovnoj školi Grigora Viteza u Osijeku. Nakon toga je pohađao Elektrotehničku i prometnu školu Osijek, te 2015. godine stekao zvanje Elektrotehničar. Obrazovanje je nastavio upisom Preddiplomskog studija Računarstva, ali je promjenio smjer sljedeće godine na stručni studij Informatike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku 2016. godine. Praksu je odradio 2019. godine kao developer u Mono d.o.o. gdje je odmah i nastavio raditi preko studentskog ugovora.

_____

Vlatko Klen

PRILOZI

Projektna mapa s izvornim kôdom i priloženim word i pdf dokumentom nalazi se na optičkom disku koji je priložen uz printanu verziju završnog rada.