Qt GUI aplikacija za testiranje algoritama sortiranja

Almaši, Matija

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

Stručni studij

WINDOWS FORMS GUI APLIKACIJA ZA TESTIRANJE ALGORITAMA SORTIRANJA

Završni rad

Matija Almaši

Osijek, 2019.

SADRŽAJ

1. UVOD 1
2. KORIŠTENE TEHNOLOGIJE 2
2.1. Microsoft Visual Studio 2
2.2. Windows Forms (.NET framework) 2
2.3. C# programski jezik 3
2.4. Json.NET dodatak za .NET framework 3
3. SORTIRANJE I ALGORITMI SORTIRANJA 4
3.1. Sortiranje 4
3.2. Složenost algoritama 4
3.3. Vrste sortiranja 5
4. OPIS I IZRADA WINDOWS FORMS APLIKACIJE 20
4.1. Izrada Windows Forms Aplikacije 20
4.2. Opis i implementacija Windows Forms aplikacije 21
5. KORIŠTENJE APLIKACIJE 31
5.1. Postavljanje testiranja 31
5.2. Rezultati testiranja 34
6. ZAKLJUČAK 39
LITERATURA 40
SAŽETAK 41
ABSTRACT 42
ŽIVOTOPIS 43
PRILOZI 44

1. UVOD

Cilj ovoga rada je izrada Windows Forms GUI (engl. *Graphical User Interface*) aplikacije za testiranje algoritama sortiranja. Temeljem parametara koje korisnik unosi u grafičko sučelje: broj testova, odabir algoritama, duljina polja za pojedine testove, minimalne i maksimalne vrijednosti elemenata polja, mjeri se vrijeme potrebno određenom algoritmu za sortiranje zadanog polja te prikaz rezultata na stupčastom i linijskom grafu. Po završetku testiranja rezultate je moguće spremiti u vanjsku datoteku.

Svrha ove aplikacije je pomoć pri pravilnom odabiru algoritma ovisno o duljini polja s djelomično (pošto su poznate minimalna i maksimalna vrijednost) nepredvidivim elementima. Odabir algoritma se u prvu ruku čini jednostavnim, jer je cilj uzeti najbrži algoritam, no univerzalno najbrži algoritam ne postoji. Također treba uzeti u obzir brzinu i jednostavnost implementacije samog algoritma, pošto i ovome području postoje velike razlike. Primjerice ako postoji potreba za sortiranjem malih polja duljine do 1000 elemenata, izbor algoritma se svodi na jednostavnost implementacije više nego na brzinu izvođenja, ali ako govorimo o poslužitelju koji sortira bilijune brojeva dnevno, brzina izvođenja postaje najvažnija stavka pri izboru algoritma.

Podjela rada po poglavljima glasi:

- U poglavlju 2 opisane su tehnologije korištene za izradu aplikacije.
- U poglavlju 3 opisani su korišteni algoritmi.
- U poglavlju 4 opisana je aplikacija, te koraci prilikom njezine izrade
- U poglavlju 5 prikazane su upute za korištenje aplikacije, te rezultati testiranja dobiveni izrađenom aplikacijom.

1.1. Zadatak završnog rada

Zadatak završnog rada je kreirati aplikaciju za testiranje različitih algoritama za sortiranje koja omogućava prikaz rezultata testiranja stupčastim i linijskim dijagramom. Potrebno je kreirati GUI koristeći Windows Forms, 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.

2. KORIŠTENE TEHNOLOGIJE

U svijetu informacijskih tehnologija, broj programskih jezika, kao i alata koji uključuju uređivače teksta i razvojna okruženja postoji mnogo. No, Microsoft znatno ograničava mogućnost kreiranja aplikacija koje koriste njihove tehnologije na vlastite alate, kao što je Visual Studio. Ovo poglavlje baviti će se alatima koji su korišteni u izradi ovog završnog rada.

2.1. Microsoft Visual Studio

Microsoft Visual Studio (dalje u tekstu: Visual Studio) je razvojno okruženje (engl. *Integrated Development Environment, IDE*) kreirano od strane Microsoft Corporation-a. Prema [1] korisnicima omogućava razvoj računalnih programa, internetskih stranica, servisa i aplikacija, kao i mobilnih aplikacija.

Prva inačica Visual Studia puštena je u javnost 1997. godine, a trenutno zadnja inačica je Visual Studio 2019. Prema službenim Microsoftovim podacima iz 2016. godine [2], 3.1 milijun programera je aktivno koristilo neki od Microsoftovih razvojnih alata, od čega je preko 2 milijuna koristilo C# programski jezik.

Na Microsoftovim stranicama mogu se pronaći najnovije, ali isto tako i starije inačice Visual Studia, uključujući besplatnu inačicu Visual Studio Community i besplatni uređivač koda (engl. *Code Editor*) Visual Studio Code.

2.2. Windows Forms (.NET framework)

Windows Forms je knjižnica grafičkih klasa koja je dio Microsoftovog .NET razvojnog okvira. Prema [3] korisnicima pruža platformu za izgradnju bogatih aplikacija s grafičkim korisničkim sučeljem za stolna računala, laptope i tablete. Svrha Windows Forms-a je lakši prikaz podataka, olakšano korištenje aplikacija i povećana sigurnost.

Svaka kontrola (element na korisničkom sučelju) je instanca klase. Rasporedom i ponašanjem kontrola se upravlja metodama i pristupnicima (engl. *accessor*). Windows Forms pruža velik izbor kontrola, kao što su: kućice za unos teksta ili brojeva, tipke, grafove, trake za napredak, oznake, kalendare, padajuće izbornike itd.

Također je moguće proširivati postojeće klase iz knjižnice kako bi se postigla visoka razina apstrakcije i višekratnog korištenja koda.

2.3. C# programski jezik

C# (engl. izgovor *„See Sharp“*) je jednostavan, moderan, objektno orijentiran programski jezik široke namjene koji omogućava razvoj aplikacija koristeći više programskih paradigmi. Prema [4] C# polaže svoje korijene u „C“ obitelji programskih jezika, te je vrlo lako shvatljiv programerima koji su koristili C, C++, Javu ili JavaScript. Razvijen je od strane Microsofta 2000. godine, a trenutna zadnja stabilna inačica je 7.3.

Na Microsoftovim stranicama mogu se pronaći detaljni vodiči za pisanje C# koda, od najjednostavnije aplikacije, kao što je „*Hello World*“, pa sve do znatno kompleksnijih implementacija.

2.4. Json.NET dodatak za .NET framework

Json.NET je dodatak otvorenog koda (engl. *open source*) za Visual Studio razvijen od strane Newtonsoft-a. Prema [5], korisnicima omogućava serijalizaciju .NET objekata, odnosno spremanje objekta u JSON, te deserijalizaciju, odnosno kreiranje objekta koristeći podatke iz JSON datoteke.

3. SORTIRANJE I ALGORITMI SORTIRANJA

Iako algoritama za sortiranje postoji na stotine, ovo poglavlje bavi se sa šest algoritama odabranih za svrhe izrade ovog rada, te sortiranjem općenito.

3.1. Sortiranje

Iako se sortiranje koristi znatno više otkad postoje moderna računala, sama potreba za sortiranjem se pojavila još u dalekoj prošlosti. Sortirati se može skoro sve, od ljudi – po abecedi, visini, do npr. voća po veličini ili kovanica po vrijednosti. Pojavom modernih računala se primjena sortiranja proširila na digitalne podatke. Vrlo često se sortiraju popisi korisnika po korisničkoj oznaci abecedno ili po jedinstvenoj oznaci od manje prema većoj. Jednako često se sortiraju i polja (engl. *array*) brojeva, što zbog raznoraznih analiza ili drugih razloga. Na slici 3.1. prikazan je jedan primjer sortiranja, gdje se države sortiraju uzlazno u ovisnosti o broju stanovnika.

Država	Populacija
Hrvatska	4.298.889
SAD	327.167.434
Njemačka	83.019.200
Finska	5.519.586
Rusija	146.793.744
Japan	124.800.000

Država	Populacija
Hrvatska	4.298.889
Finska	5.519.586
Njemačka	83.019.200
Japan	124.800.000
Rusija	146.793.744
SAD	327.167.434

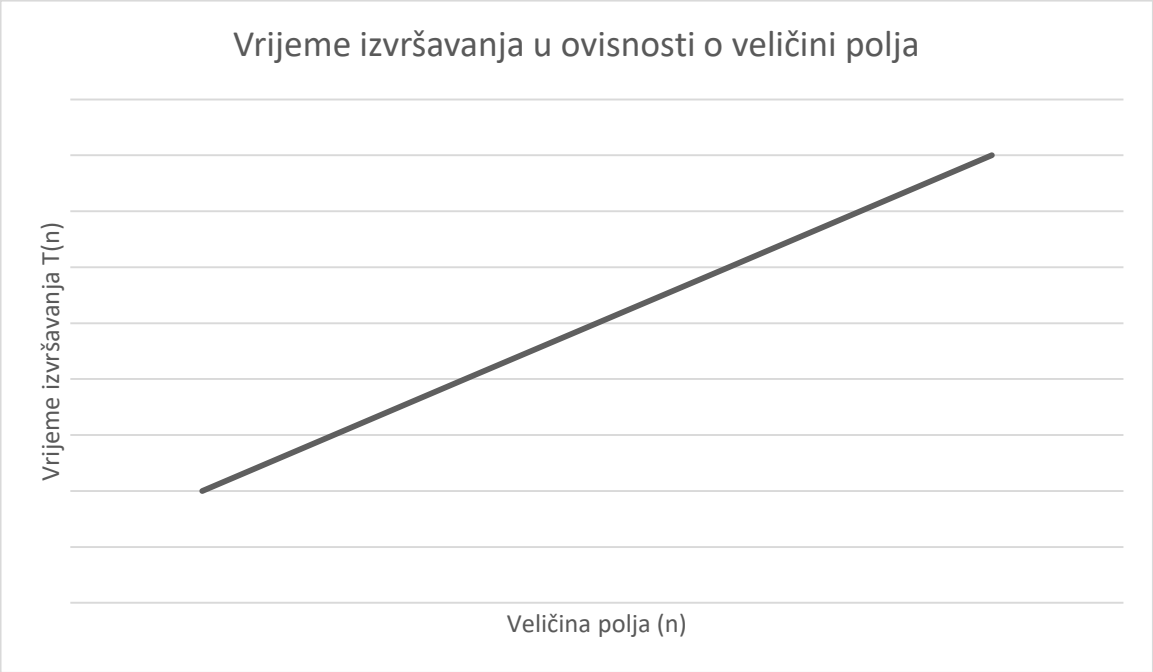
Slika 3.1. *Primjer sortiranja država prema njihovoj populaciji*

3.2. Složenost algoritama

Učinkovitost algoritama sortiranja, kao i svakog drugog dijela programskoga koda, može se analizirati ili usporediti s različitom implementacijom koda, dok god je njihova svrha ista, odnosno dok god daju identičan rezultat, ako im predamo identične ulazne podatke. Ovo poglavlje se bavi teoretskom stranom algoritama, bez rezultata testiranja.

Kriterij po kojemu se algoritmi uspoređuju je cijena. Cijenom se u svijetu programiranja uglavnom ne označava materijalna vrijednost, već procesorsko vrijeme i zauzeće memorije. Procesorsko vrijeme, kao što samo ime govori, je količina vremena koja je potrebna procesoru za izvršavanje cijelog algoritma. Zauzeće memorije je fizička količina glavne memorije koju program

koristi. Slika 3.2. prikazuje ovisnost vremena izvršavanja algoritma sortiranja u ovisnosti o veličini polja koje mu se preda.



Slika 3.2. *Vrijeme izvršavanja algoritma u ovisnosti o veličini polja*

„Vrijeme izražavamo kao funkciju *T(n)* gdje je neka pogodna mjera za veličinu skupa ulaznih podataka. Ako promatramo algoritam koji sortira niz brojeva tada njegovo vrijeme izražavamo kao *T(n)* gdje je n duljina toga niza brojeva“ prema [6].

3.3. Vrste sortiranja

Svaki problem se u programiranju može riješiti na više različitih načina tako da dobijemo isti rezultat. Prema [7], algoritama sortiranja možemo nabrojati više od 30, a neki od njih se mogu naći u više različitih inačica. Iako im je svima svrha ista, te će, ako su ispravno napisani, za identične ulazne podatke dati identične izlazne podatke - njihove performanse i jednostavnost su drastično različite. Osnovna podjela algoritama sortiranja prema Mangeru [6] glasi:

- Sortiranje zamjenom elemenata – ovdje pripadaju sortiranje izborom najmanjeg elementa (engl. *selection sort*) i sortiranje zamjenom susjednih elemenata (engl. *bubble sort*)
- Sortiranje umetanjem – dijelimo ga na jednostavno sortiranje umetanjem (engl. *insertion sort*) i višestruko sortiranje umetanjem (engl. *shell sort*)

- Rekurzivni algoritmi za sortiranje – prepoznajemo algoritam brzog sortiranja (engl. *quick sort*) i algoritam sortiranja pomoću sažimanja (engl. *merge sort*)
- Sortiranje pomoću binarnih stabala – ovdje pripadaju sortiranje obilaskom binarnog stabla traženja (engl. *tree sort*) i sortiranje pomoću hrpe (engl. *heap sort*)

3.3.1. Sortiranje izborom najmanjeg elementa

Jedan od najjednostavnijih, ali i najsporijih algoritama sortiranja. Princip rada je sljedeći: kroz polje se prolazi *n-1* puta. U svakom prolazu traži se najmanji element i njime zamjenjujemo prvi element koji uzimamo u obzir. Svakim prolaskom prestajemo uzimati u obzir jedan element s početka polja. Primjer sortiranja izborom najmanjeg elementa vidi se na slici 3.3. Pronađeni najmanji element je osjenčan u svakom prolazu, dok se promatrani elementi nalaze desno od vertikalne linije.

Početno polje	16	30	2	42	10	23	7
Nakon 1. izmjene	2	30	16	42	10	23	7
Nakon 2. izmjene	2	7	16	42	10	23	30
Nakon 3. izmjene	2	7	10	42	16	23	30
Nakon 4. izmjene	2	7	10	16	42	23	30
Nakon 5. izmjene	2	7	10	16	23	42	30
Nakon 6. izmjene	2	7	10	16	23	30	42

Slika 3.3. *Primjer sortiranja algoritmom izbora najmanjeg elementa*

Realizacija algoritma sortiranja izborom najmanjeg elementa u programskom jeziku C# vidljiva je na slici 3.4. koja prikazuje metodu *SelectionSort*. Argumenti koje metoda prima su polje cjelobrojnih elemenata *int[] arr* i cjelobrojna duljina polja *int n*. Metoda izmjenjuje predano polje tako da ono po završetku izvršavanja postaje sortirano. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float SelectionSort(int[] arr, int n)
        {
            float count = 0;
            for (int i = 0; i < n - 1; i++)
            {
                int min_idx = i;
                for (int j = i + 1; j < n; j++)
                {
                    if (arr[j] < arr[min_idx])
                        min_idx = j;
                    count++;
                }
                int temp = arr[min_idx];
                arr[min_idx] = arr[i];
                arr[i] = temp;
            }
            return count;
        }
```

Slika 3.4. *C# kod algoritma sortiranja izborom najmanjeg elementa, prema [8]*

Analiza vremena izvršavanja glasi:

- U prvom prolasku se događa $n-1$ usporedbi, a u svakom slijedećem prolazu je taj broj umanjen za 1. Ukupni broj usporedbi prema (3-1) :

$$(n-1)+(n-2)+(n-3)+\cdots+2+1=n(n-1)/2 \qquad (3\text{-}1)$$

- U svakom prolasku se također izvršava i zamjena, odnosno izvršava se $3(n-1)$ operacija pridruživanja.
- Ukupni broj operacija prema (3-2) glasi:

$$n(n-1)/2+3(n-1)=O(n^2) \qquad (3\text{-}2)$$

- Jednadžba za ukupni broj operacija nam govori kako algoritam uvijek obavlja istu količinu posla, pod uvjetom da je polje iste veličine, bez obzira na početno stanje polja.

3.3.2. Sortiranje zamjenom susjednih elemenata

Kao i sortiranje izborom najmanjeg elementa, ovaj je algoritam iznimno jednostavan, ali i iznimno spor. Ovaj algoritam u svakom prolasku kroz polje provjerava je li sljedeći element manji od trenutnog i ako je, zamjenjuje im vrijednosti. Svakim prolaskom skraćuje polje koje uspoređuje

za jedan element, pošto najveći promatrani element uvijek bude postavljen na zadnje mjesto. Moguće je izvesti varijaciju algoritma koja se zaustavlja ako utvrdi da više nema elemenata koje treba zamijeniti.

Primjer algoritma zamjenom susjednih elemenata vidi se na slici 3.5. U svakom koraku su osjenčani oni elementi koje treba zamijeniti. Za razliku od prošlog algoritma, ovdje se promatrani dio polja nalazi lijevo od vertikalne linije. Zadnji prolaz je kontrolni prolaz gdje algoritam provjerava jesu li svi elementi na ispravnom mjestu u polju.

Prvi prolaz	16	30	2	42	10	23	7
	16	2	30	42	10	23	7
	16	2	30	10	42	23	7
	16	2	30	10	23	42	7
	16	2	30	10	23	7 |	42
Drugi prolaz	16	2	30	10	23	7	42
	2	16	30	10	23	7	42
	2	16	10	30	23	7	42
	2	16	10	23	30	7	42
	2	16	10	23	7 |	30	42
Treći prolaz	2	16	10	23	7 |	30	42
	2	10	16	23	7 |	30	42
	2	10	16	7 |	23	30	42
Četvrti prolaz	2	10	16	7 |	23	30	42
	2	10	7 |	16	23	30	42
Peti prolaz	2	10	7 |	16	23	30	42
	2	7 |	10	16	23	30	42
Šesti prolaz	2	7	10	16	23	30	42

Slika 3.5. *Primjer sortiranja algoritmom zamjene susjednih elemenata*

Realizacija algoritma sortiranja zamjenom susjednih elemenata u programskom jeziku C# vidljiva je na slici 3.6. koja prikazuje metodu *Bubblesort*. Metoda kao argumente prima cjelobrojno polje *int[] arr* i cjelobrojnu duljinu polja *int n*. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float BubbleSort(int[] arr, int n)
        {
            float count = 0;
            for (int i = 0; i < n - 1; i++)
                for (int j = 0; j < n - i - 1; j++)
                {
                    if (arr[j] > arr[j + 1])
                    {
                        int temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                    count++;
                }
            return count;
        }
```

Slika 3.6. *C# kod algoritma sortiranja zamjenom susjednih elemenata prema [9]*

Analiza vremena izvršavanja glasi:

- U svakom m-tom prolazu algoritam izvršava u najgorem slučaju $n - m$ usporedbi i $n - m$ zamjena elemenata.
- U zadnjem prolazu se izvršava u najgorem slučaju jedna usporedba i jedna zamjena.
- Ukupan broj operacija u najgorem slučaju prema (3-3) glasi:

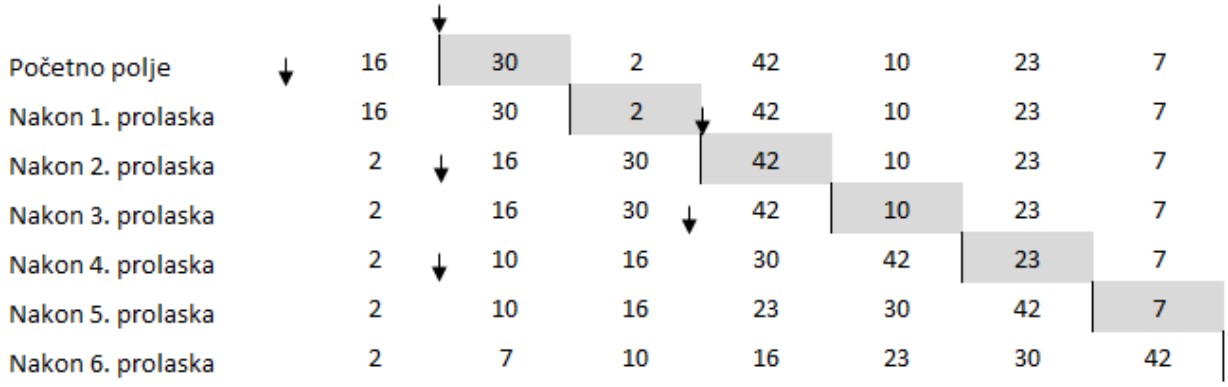
$$4(n-1) + 4(n-2) + \cdots + 4 * 1 = 4n(n-1)/2 = 2n(n-1) = O(n\text{^}2) \qquad (3\text{-}3)$$

- S obzirom da je nemoguće predvidjeti stanje ulaznog polja, broj operacija je bilo gdje između $(n-1)$ i $2n(n-1)$.

3.3.3. Jednostavno sortiranje umetanjem

Na početku algoritma prvi element polja je već sortiran, a ostatak polja nije. U svakom prolasku algoritam uzima prvi element nesortiranog dijela te odredi „pravo mjesto“ na kojemu taj element treba biti, te ga na isto umeće. Svakim prolaskom algoritam smanjuje nesortirani dio za 1, odnosno povećava sortirani dio za 1.

Primjer rada jednostavnog algoritma za sortiranje vidljiv je na slici 3.7. U svakom prolasku je promatrani element iz nesortiranog dijela polja osjenčan, dok je mjesto na koje ga algoritam umeće označen strelicom. Sortirani i nesortirani dio su odijeljeni vertikalnom linijom.



Slika 3.7. *Primjer jednostavnog sortiranja umetanjem*

Realizacija algoritma jednostavnog sortiranja umetanjem u programskom jeziku C# vidljiva je na slici 3.8. koja prikazuje metodu *InsertionSort*. Metoda kao argumente prima cjelobrojno polje *int[] arr* i cjelobrojnu duljinu polja *int n*. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float InsertionSort(int[] arr, int n)
        {
            float count = 0;
            for (int i = 1; i < n; ++i)
            {
                int key = arr[i];
                int j = i - 1;

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

Slika 3.8. *C# kod algoritma jednostavnog sortiranja umetanjem prema [10]*

3.3.4. Brzo sortiranje

„Metoda podijeli-pa-vladaj (engl. *divide-and-conquer*) sastoji se u tome da zadani primjerak problema razbijemo u nekoliko manjih (istovrsnih) primjeraka i to tako da se rješenje polaznog primjeraka može konstruirati iz rješenja manjih primjeraka“ (Manger, 2014, pp. 169-170). Ovo je metoda koju koristimo za algoritam brzog sortiranja.

U praksi on izgleda ovako:

- Odaberemo jedan element kojeg nazivamo stožer.
- Svi elementi koji su manji od stožera se postavljaju na lijevu stranu.
- Svi elementi koji su veći od stožera se postavljaju na desnu stranu.

Koriste se dva brojača, *i* i *j* (ili *L(left)* i *R(right)*) za usporedbu elemenata u odnosu na stožer:

- „Sve dok je *i* lijevo od *j* pomičemo *i* udesno preskačući sve elemente manje od stožera. Ako je za neki element utvrđeno da je veći od pivota *i* se zaustavlja.
- Sve dok je *j* desno od *i* pomičemo *j* ulijevo preskačući sve elemente koji su veći od stožera. Ako je za neki element utvrđeno da je manji od stožera *j* se zaustavlja.
- Kada su oboje *i* i *j* zaustavljeni elementi se zamjenjuju.
- Kada se *i* i *j* prekriže nije potrebno mijenjati elemente. Zaustave se prolasci kroz niz i element na koji pokazuje *i* zamjeni se sa stožerom te se zatim ponovno postavi.“

Primjer algoritma brzog sortiranja vidi se na slici 3.9. Brojevi na kojima se trenutno nalaze brojači su osjenčani. Dio polja koji trenutno promatramo nalazi se desno od vertikalne linije, a lijevo od linije je stožer. Uglatim zagradama su odvojeni isječci polja koji se sortiraju rekurzivnim pozivom algoritma. Detalji rekurzivnih poziva nisu prikazani.

Početno polje	16	30	2	42	10	23	7
Pomicanje iteratora	16	30	2	42	10	23	7
Zamjena elemenata	16	7	2	42	10	23	30
Promicanje iteratora	16	7	2	42	10	23	30
Zamjena elemenata	16	7	2	10	42	23	30
Pomicanje iteratora	16	7	2	10	42	23	30
Iteratori se preklapaju	[7	2	10]	16	[42	23	30]
Dva potpolja se sortiraju	[2	7	10]	16	[23	30	42]

Slika 3.9. *Primjer brzog sortiranja*

Realizacija algoritma brzog sortiranja u programskom jeziku C# vidljiva je na slici 3.10. koja prikazuje metodu *QuickSort* i pomoćnu metodu *Partition()*. Metoda *QuickSort* kao argumente prima cjelobrojno polje *int[] arr*, cjelobrojni indeks prvog elementa polja *int low* i cjelobrojni indeks zadnjeg elementa polja *int high*. Metoda *Partition* prima iste argumente kao i *QuickSort*. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float QuickSort(int[] arr, int low, int high)
        {
            float count = 0;
            if (low < high)
            {
                var result = Partition(arr, low, high);
                int pi = result.Item1;
                count += result.Item2;
                count += QuickSort(arr, low, pi - 1);
                count += QuickSort(arr, pi + 1, high);
            }
            return count;
        }
        static Tuple<int, float> Partition(int[] arr, int low, int high)
        {
            int pivot = arr[high];
                float count = 0;

            int i = (low - 1);
            for (int j = low; j < high; j++)
            {
                if (arr[j] <= pivot)
                {
                    i++;

                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
                count++;
            }

            int temp1 = arr[i + 1];
            arr[i + 1] = arr[high];
            arr[high] = temp1;

            return Tuple.Create(i + 1, count);
        }
```

Slika 3.10. *C# kod algoritma brzog sortiranja prema [11]*

Analiza vremena izvršavanja glasi:

- U najgorem slučaju broj operacija prema (3-4) glasi:

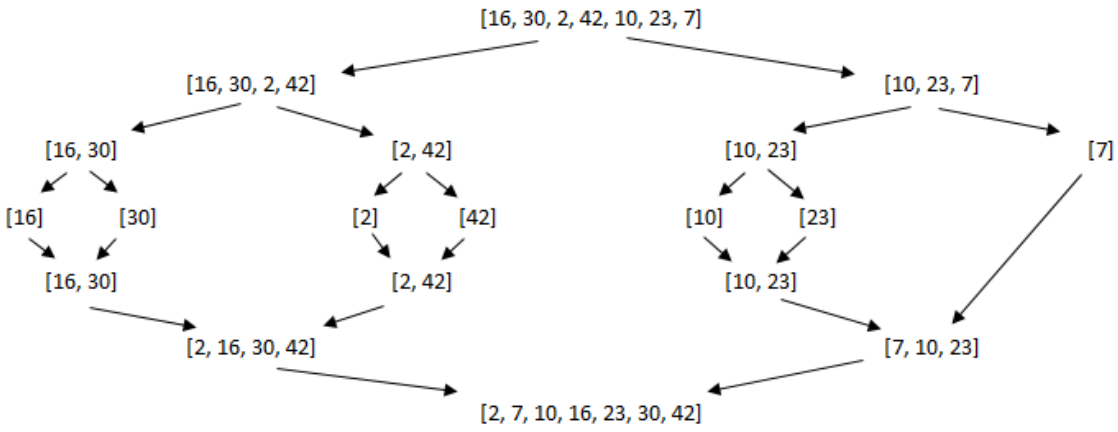
$$n + (n-1) + (n-2) + \cdots + 2 = (n + (n-1) + (n-2) + \cdots + 2 = (n+1)(n/2) - 1 \tag{3-4}$$

- U najboljem slučaju, koristeći veliko O notaciju prema (3-5) dobivamo:

$$\theta(nlog_2 n) \tag{3-5}$$

3.3.5. Sortiranje pomoću sažimanja

Ako se polje sastoji samo od jednog elementa, tada je već sortirano. U suprotnom, polje se dijeli na dva polja podjednakih duljina. Ona se sortiraju rekurzivnim pozivima istog algoritma. Kada su oba polja sortirana, ona se sažimaju u jedno sortirano polje uz pomoć algoritma sažimanja. Primjer sortiranja sažimanjem vidljiv je na slici 3.11. U uglatim zagradama nalaze se isječci polja koji nastaju tokom sortiranja. Strelice pokazuju iz čega su nastali isječci polja.

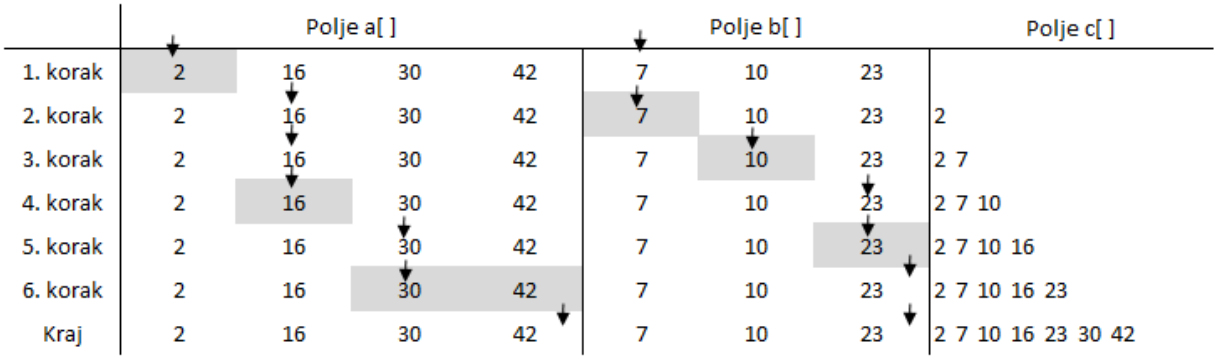


Slika 3.11. *Primjer sortiranja sažimanjem*

Postupak pomoćnog algoritma sažimanja 2 sortirana ulazna polja duljine n i m u 1 sortirano polje duljine $n + m$ glasi (za potrebe primjera koriste se ulazna polja *a[]* i *b[]* i sažeto polje *c[]*):

- Postavljamo brojače na početne elemente polja *a[]*, *b[]* i *c[]*.
- Ako je jedan od tih elemenata manji, zapisujemo ga u polje *c[]*, te povećavamo odgovarajući brojač za jedan.
- Ako su jednaki, zapisujemo oba u *c[]* i povećavamo oba brojača za jedan.
- Kod svakog zapisivanja povećavamo brojač u polju *c[]*.
- Kada zapišemo zadnji element polja *a[]* ili *b[]* u polje *c[]*, ostatak drugog polja prepisujemo u *c[]*.

Primjer pomoćnog algoritma sažimanja vidljiv je na slici 3.12. Položaji brojača su označeni strelicama, dok je element koji se prepisuje u polje *c[]* osjenčan.



Slika 3.12. *Prikaz pomoćnog algoritma sažimanja*

Realizacija algoritma sortiranja pomoću sažimanja u programskom jeziku C# vidljiva je na slici 3.13. koja prikazuje metodu *MergeSort* i pomoćnu metodu *Merge*. Metoda *MergeSort* kao argumente prima cjelobrojno polje brojeva *int[] arr*, cjelobrojni početni indeks polja *int p* i cjelobrojni krajnji indeks polja *int r*. Metoda *Merge* prima sve argumente koje prima i *MergeSort* uz dodatak cjelobrojnog argumenta *int q*, koji je aritmetička sredina argumenata *p* i *r*. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float MergeSort(int[] arr, int p, int r)
        {
            float count = 0;
            if (p < r)
            {
                int q = (p + r) / 2;
                count += MergeSort(arr, p, q);
                count += MergeSort(arr, q + 1, r);
                count += Merge(arr, p, q, r);
            }
            return count;
        }

static float Merge(int[] arr, int p, int q, int r)
        {
            int i, j, k;
            float count = 0;
            int n1 = q - p + 1;
            int n2 = r - q;
            int[] L = new int[n1];
            int[] R = new int[n2];
            for (i = 0; i < n1; i++)
                L[i] = arr[p + i];
            for (j = 0; j < n2; j++)
                R[j] = arr[q + 1 + j];
            i = 0;
            j = 0;
            k = p;
            while (i < n1 && j < n2)
            {
                if (L[i] <= R[j])
                {
                    arr[k] = L[i];
                    i++;
                }
                else
                {
                    arr[k] = R[j];
                    j++;
                }
                k++;
                count++;
            }
            while (i < n1)
            {
                arr[k] = L[i];
                i++;
                k++;
            }
            while (j < n2)
            {
                arr[k] = R[j];
                j++;
                k++;
            }
            return count;
        }
```

Slika 3.13. *C# kod algoritma sortiranja sažimanjem prema [12]*

Analiza vremena izvršavanja glasi:

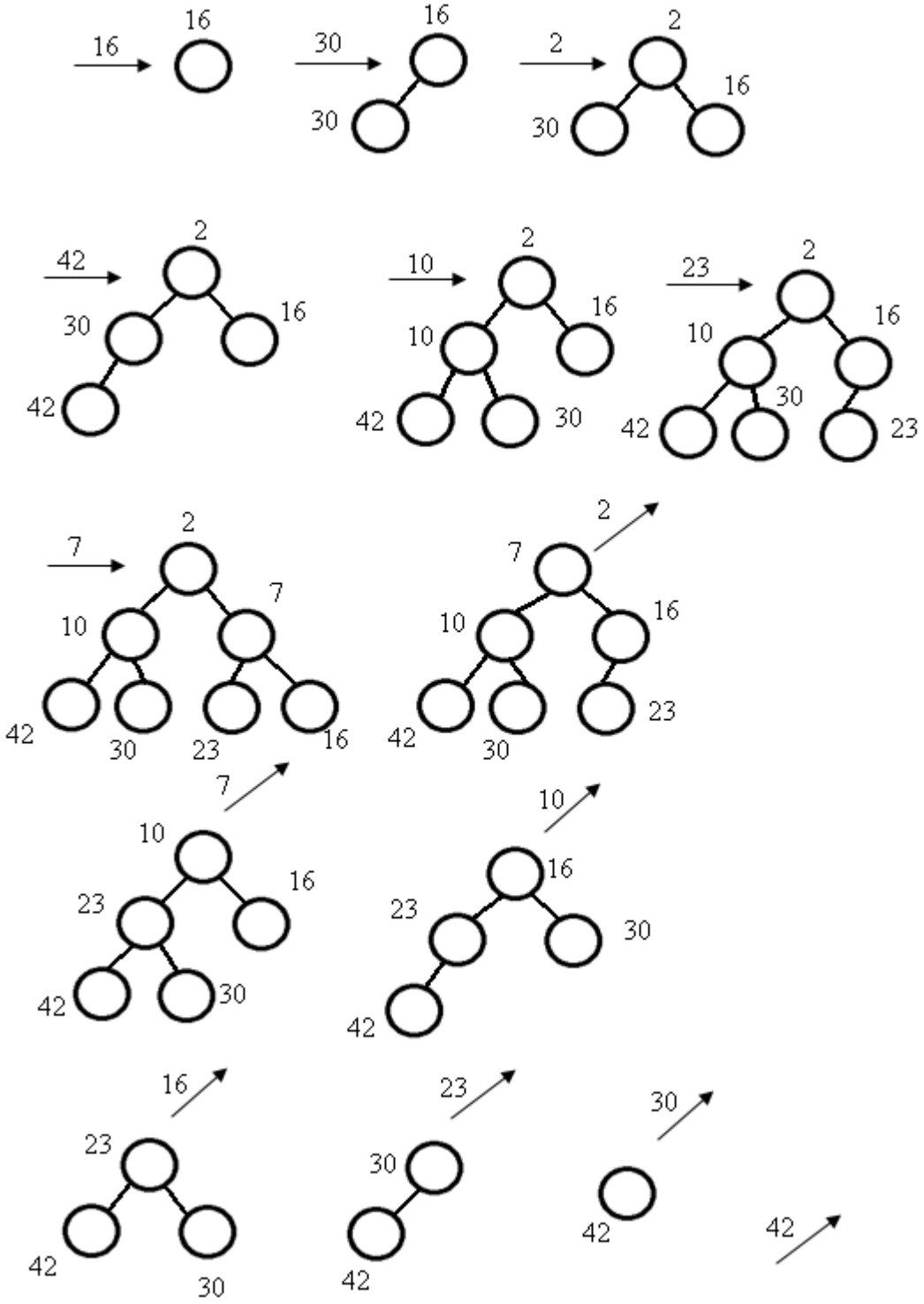
- S obzirom da je ukupan broj razina $log_2 n + 1$, u najgorem slučaju vrijeme izvršavanja prema (3-6) je:

$$log_2(n+1) * O(n) = O(nlogn) \tag{3-6}$$

- Prednost ovoga algoritma je mogućnost sortiranja velikih polja koja mogu biti pohranjena na vanjskoj memoriji.
- Mana ovoga algoritma je dodatni trošak memorije, koji se stvara prilikom prepisivanja polja tokom sažimanja.

3.3.6. Sortiranje pomoću hrpe

Za sortiranje pomoću hrpe koristimo binarna stabla. Prvi korak je stvaranje hrpe tako da redom ubacujemo podatke u istu. Nadalje s hrpe „skidamo“ najmanji element te ga vraćamo u polje. Svojstva hrpe su takva da ćemo elemente skidati s hrpe u sortiranom poretku, odnosno od najmanjeg do najvećeg. S obzirom da se u programima hrpa pohranjuje u 1D polje, operacije nad hrpom su zapravo manipulacija poljem. Primjer algoritma sortiranja pomoću hrpe vidljiv je na slici 3.14. Strelice označavaju trenutnu operaciju, odnosno koji element postavljamo na hrpu ili skidamo s hrpe.



Slika 3.14. *Primjer sortiranja pomoću hrpe*

Realizacija algoritma sortiranja pomoću hrpe u programskom jeziku C# vidljiva je na slici 3.15. koja prikazuje metodu *HeapSort* i pomoćnu metodu H*eapify*. Metoda *HeapSort* kao argumente prima cjelobrojno polje brojeva *int[] arr* i cjelobrojnu duljinu polja *int n*. Pomoćna

metoda *Heapify* prima iste argumente kao *HeapSort* uz dodatak cjelobrojnog argumenta *int i* koji označava trenutni položaj brojača u metodi *HeapSort*. Uz to, algoritam broji koliko je usporedbi izvršeno da bi se polje sortiralo.

```
public static float HeapSort(int[] arr, int n)
        {
            float count = 0;
            for (int i = n / 2 - 1; i >= 0; i--)
                count += Heapify(arr, n, i);

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

                count += Heapify(arr, i, 0);
            }
            return count;
        }
static float Heapify(int[] arr, int n, int i)
        {
            float count = 0;
            int largest = i;
            int l = 2 * i + 1;
            int r = 2 * i + 2;

            if (l < n && arr[l] > arr[largest])
                largest = l;
            count++;

            if (r < n && arr[r] > arr[largest])
                largest = r;
            count++;

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

                count += Heapify(arr, n, largest);
            }
            count++;

            return count;
        }
```

Slika 3.15. *C# kod algoritma sortiranja pomoću hrpe prema [13]*

Analiza složenosti glasi:

- Izvršavanje algoritma se sastoji od n operacija ubacivanja elemenata u hrpu i n operacija izbacivanja elemenata s hrpe.
- S obzirom da jedna operacija ubacivanja odnosno izbacivanja traje $O(logn)$, ukupno trajanje algoritma u najgorem slučaju iznosi $O(nlogn)$.

4. OPIS I IZRADA WINDOWS FORMS APLIKACIJE

Ovo poglavlje se bavi izradom Windows Forms aplikacije izrađene za potrebe ovog završnog rada. Također, ovdje su prikazani i detaljnije opisani obrasci aplikacije, kao i njihove sastavnice.

4.1. Izrada Windows Forms Aplikacije

Aplikacija je izrađena u programskom okruženju Microsoft Visual Studio Enterprise 2017, koja je studentima besplatno dostupna za akademske ili istraživačke svrhe u sklopu usluge *MSDN-AA*. Za komercijalne svrhe aplikaciju je moguće preuzeti s Microsoftove stranice uz mjesečnu pretplatu čiji iznos ovisi o inačici.

Svaki obrazac (engl. *form*) se sastoji od korisniku nevidljivog dijela kojim se određuje funkcionalnost i kontrola - odnosno korisniku vidljivog dijela. Oba dijela se definiraju programskim jezikom C#.

U domenu funkcionalnosti ulazi - što se dogodi kad se aplikacija pokrene, kada korisnik nešto klikne ili promjeni vrijednost, odabere element u padajućem izborniku i slično. Programerima se predlaže korištenje dijaloških okvira prije bilo koje „važnije“ operacije kako korisnici ne bi slučajno izvršili neku neželjenu akciju.

Vidljivi dio, iako je definiran programskim jezikom C#, može se definirati i povlačenjem i ispuštanjem (engl. *drag-and-drop*) potrebne kontrole na prozor aplikacije. Tada Visual Studio automatski generira kod za odabranu kontrolu. Za početak, potrebno je rasporediti kontrole po prozoru. Lokacija, veličina, naziv, identifikator i druga svojstva kontrola se mogu slobodno odrediti u okviru sa svojstvima svake kontrole ili programski.

Kada su sve potrebne kontrole postavljene, potrebno ih je povezati s kodom, jer su kontrole inače beskorisne. Moguće je iščitavati podatke iz kućica za unos teksta ili brojeva, kao i indeksa odabranog elementa u padajućem izborniku, postaviti slušatelj (engl. *listener*) klika ili promjene odabranog elementa itd. Kontrole se u kodu pozivaju prema njihovom nazivu koje se postavlja u svojstvima. Da bi se dodao slušatelj klika moguće je dvostrukim klikom na kontrolu automatski generirati praznu metodu te dodati *onclick* svojstvo kontroli.

Ispitivanje aplikacije radi se u više koraka. Prvi korak je kompiliranje (engl. *compiling*) u kojemu će kompajler korisniku prikazati greške u kodu, ako postoje. Ovdje se većinom radi o greškama u pisanju, pogrešnim referencama, pokušajima zapisivanja vrijednosti netočnog tipa podatka u varijablu i slično. Nakon ispravka svih eventualnih grešaka u kodu, kod se uspješno

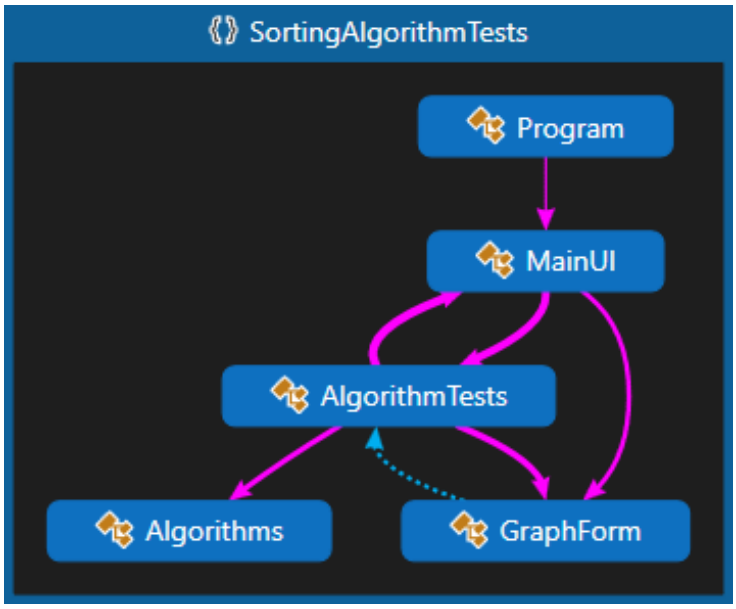
kompilira i pokreće u načinu za ispravljanje pogrešaka (engl. *debug, debugging*) gdje je u svakom koraku izvršavanja moguće pratiti stanja varijabli, argumenata, objekata itd. Kada su sve greške otklonjene, aplikacija se kompilira u načinu za objavu (engl. *release*) koja primjenjuje razne optimizacije na kod s ciljem bržeg izvršavanja.

4.2. Opis i implementacija Windows Forms aplikacije

Windows Forms aplikacija treba omogućavati:

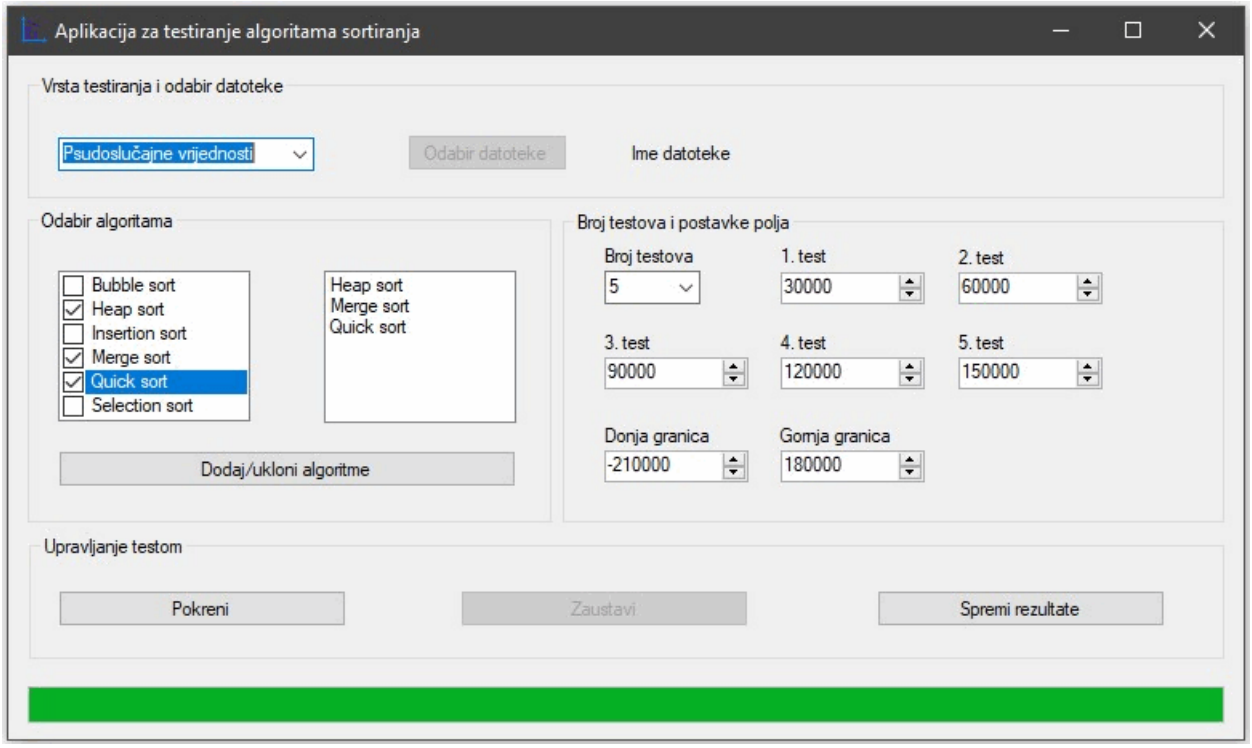
- odabir algoritama koji će se testirati
- odabir veličine polja na kojemu se testira sortiranje
- automatsko generiranje polja koristeći pseudo slučajne vrijednosti
- prikaz rezultata testiranja koristeći stupčasti i linijski graf,
- spremanje rezultata testiranja u vanjsku datoteku
- učitavanje rezultata iz vanjske datoteke.

Tok aplikacije prikazan je na slici 4.1. Na njemu su prikazane sve klase aplikacije i njihove veze. Ljubičaste pune linije označavaju pozive metoda, dok plave isprekidane linije označavaju čitanje podataka. Na slici se također vidi i hijerarhija klasa, odnosno klasa koja se nalazi više na prikazu kreira objekte klase koja se nalazi niže na prikazu.



Slika 4.1. *Prikaz toka aplikacije*

Korisniku vidljiv dio aplikacije sastoji se od glavnog obrasca koji sadrži sve kontrole i nekoliko dijaloških okvira koji korisniku prikazuju greške ili druge povratne informacije, te obrasca koji sadrži grafove s rezultatima testiranja. Glavni obrazac vidljiv je na slici 4.2.



Slika 4.2. *Prikaz glavnog obrasca s kontrolama*

Ovaj obrazac opisan je klasom *MainUI*. Sve kontrole ove klase dodane su povlačenjem i ispuštanjem koristeći Designer Visual Studia, te su po potrebi prilagođene koristeći okvir sa svojstvima, gdje je moguće promijeniti identifikator, naziv, dimenzije, poziciju i razna druga svojstva specifična svakoj komponenti. U Designeru su također dodane funkcionalnosti koje je moguće tim putem dodati. U slučaju klase *MainUI* to su dijalozi za učitavanje i spremanje datoteke, tajmer i pozadinski radnik (engl. *background worker*) koji se inicijalizira prema metodi *InitializeBackgroundWorkerHandlers* koja je prikazana ispod.

```
private void InitializeBackgroundWorkerHandlers()
{
backgroundWorker.DoWork += (s, e) =>
        {
        test.RunTests();
        };
backgroundWorker.RunWorkerCompleted += (s, e) =>
        {
        timer.Stop();
        swatch.Stop();
        ShowCompleteMessage();
        btnStopTesting.Enabled = false;
        btnSaveTestResults.Enabled = true;
        btnStartAlgorithmTests.Enabled = true;
        };
}
```

Programski kod 4.1. *Metoda InitializeBackgroundWorkerHandlers koja inicijalizira pozadinskog radnika*

Funkcionalni dio klase *MainUI* ne radi testiranje već kontrolira korisničko sučelje, što uključuje provjeru je li neki element prazan, omogućavanje i onemogućavanje kontrola ovisno o korisnikovim odabirima, prikazuje dijaloške okvire ili greške i slično. Također ova klasa iščitava podatke o prijašnjim testiranjima iz datoteke, sprema rezultate testiranja u datoteku. Konačno, ova klasa kreira objekt klase *AlgorithmTests* koja vrši testiranje.

Pri vrhu obrasca nalazi se okvir „Vrsta testiranja i odabir datoteke“, a u njemu se nalazi padajući izbornik označen „Vrsta testiranja“ na kojemu su ponuđene opcije: „Pseudo slučajne vrijednosti“ i „Podaci iz vanjske datoteke“.

Ako se odabere opcija „Podaci iz vanjske datoteke“, korisniku se omogućuje tipka koja pokreće dijaloški okvir za izbor datoteke. Vanjska datoteka mora biti u $.json$ obliku i ispravno ispisana. Uspješnim učitavanjem datoteke na oznaku „Ime datoteke“ dodaje se puna adresa datoteke i njezino ime. Također se popunjavaju sve kontrole s podacima o testiranju, uključujući odabrane algoritme, broj testova, duljine polja i minimalna odnosno maksimalna vrijednost elemenata polja. Metoda koja učitava podatke iz datoteke nalazi se u nastavku.

```
private void load_file_Click(object sender, EventArgs e)
{
if(openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
        string array_file = openFileDialog1.FileName;
        string json = File.ReadAllText(array_file);
        _label_file_name.Text = array_file;
        test = new AlgorithmTests();
        test = JsonConvert.DeserializeObject<AlgorithmTests>(json);
        test.GetMainReference(this);
        FillUIElements();
        test.DisplayDataFromFile();
        }
}
```

Programski kod 4.2. *Metoda load_file_Click koja sprema rezultate testiranja u datoteku*

Odabirom opcije „Testiranje s pseudo slučajnim vrijednostima“, u okviru „Odabir algoritma“ moguće je odabrati koji algoritmi se testiraju. Ponuđeni algoritmi su: *bubble sort, insertion sort, heap sort, merge sort, quick sort* i *selection sort* koji su opisani u 3. poglavlju ovog rada. Jedina kontrola koja ovdje posjeduje funkcionalnost je gumb „Dodaj/ukloni algoritme“, koji na klik poziva metodu *clbAlgorithmsChecked*. Ta metoda provjerava koji algoritmi su označeni te ih dodaje u kućicu za popis i postavlja polje *bool algorithms* ovisno o tome koji su algoritmi odabrani.

U okviru „Broj testova i postavke polja“ nalaze se kontrole za broj testova, veličinu polja u svakom testu i donju, odnosno gornju granicu elemenata u poljima. Broj testova se određuje odabirom elementa 1-5 u padajućem izborniku, dok se veličina polja i donja odnosno gornja granica određuju upisivanjem brojeva u za to predviđene kućice. Ove kontrole posjeduju samo automatski dodijeljenu funkcionalnost koja automatski postavlja uneseni broj na minimalnu odnosno maksimalnu dozvoljenu vrijednost ako se unese broj koji je manji odnosno veći od tih vrijednosti.

U nastavku je prikazana metoda *btnStartAlgorithmTests_Click* koja se izvršava na klik tipke za pokretanje testova. Ova metoda provjerava jesu li ispunjeni svi uvjeti potrebni za pokretanje testiranja, ako nisu korisniku se prikazuje greška. U suprotnom pridružuje varijablama vrijednosti s kontrola, stvara novi objekt *test* klase *AlgorithmTests* koristeći konstruktor kojemu se predaju sve varijable s unesenim vrijednostima, zatim istom objektu predaje referencu na glavni obrazac. Nadalje, stvara se ime datoteke za spremanje rezultata testiranja tako da se niti „test_“ pridruži trenutni datum i vrijeme, inicijalizira se obrazac za prikaz rezultata testiranja na grafovima,

pokreće se tajmer i konačno poziva se metoda *RunAlgorithmTests* koja pokreće testiranje na pozadinskom radniku što omogućava interakciju s korisničkim sučeljem za vrijeme testiranja.

```
public void btnStartAlgorithmTests_Click(object sender, EventArgs e)
{
if (dropdownTestType.SelectedItem == null)
       {
       MessageBox.Show("Niste odabrali vrstu testa!", "Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else if (lbAlgs.Items.Count == 0 && dropdownTestType.SelectedIndex == 0)
       {
       MessageBox.Show("Niste odabrali nijedan algoritam!", "Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else if (!CheckTestCount() && dropdownTestType.SelectedIndex == 0)
       {
       MessageBox.Show("Niste odabrali broj testova!", "Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else if (!CheckTextBoxes() && dropdownTestType.SelectedIndex == 0)
       {
       MessageBox.Show("Niste unijeli veličinu polja!", "Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else if (tbMinValue.Value > tbMaxValue.Value)
       {
       MessageBox.Show("Minimalna vrijednost u polju mora biti manja od
              maksimalne!", "Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else if (tbMinValue.Value == tbMaxValue.Value)
       {
       MessageBox.Show("Minimalna i maksimalna vrijednost ne smiju biti iste!",
"Greška",
       MessageBoxButtons.OK,
       MessageBoxIcon.Error);
       }
else
       {
       GetDataFromUI();
       test = new AlgorithmTests(algorithms, testRepetitions, array_size, min,
              max);
       test.GetMainReference(this);
       newJsonFileName = "test_" + DateTime.Now.ToString("ddMMyy_HHmmss") +
              ".json";
       btnStartAlgorithmTests.Enabled = false;
       btnStopTesting.Enabled = true;
       InitializeGraphForm();
       graphForm.EnableChartSeries();
       SetTimer();
       RunAlgorithmTests();
       }
}
```

Programski kod 4.3. *Metoda btnStartAlgorithmTests_Click koja se pokreće na klik tipke „Pokreni“*

Testiranje se izvodi, kako je već ranije navedeno, koristeći objekt *test* klase *AlgorithmTests*. Konstruktor klase i primjer testa prikazani u kodovima 4.4. i 4.5 su glavni dio programa, odnosno oni odrađuju namjenu ove aplikacije. Osim već navedenog, klasa sadrži nekoliko pomoćnih metoda za prikaz poruke kada korisnik prekine testiranje klikom na gumb „Zaustavi“, zatim metodu za prikaz podataka testiranja na grafu ako je korisnik učitao podatke iz datoteke, te metodu koja stvara polje koje će biti sortirano i popunjava ga pseudoslučajnim vrijednostima.

```
public AlgorithmTests(bool[] _algorithms, int _n_rep, int[]
       _array_size, int _min, int _max)
        {
            algorithms = new bool[6];
            arraySize = new int[5];
            timeBubble = new double[5];
            timeHeap = new double[5];
            timeInsertion = new double[5];
            timeMerge = new double[5];
            timeQuick = new double[5];
            timeSelection = new double[5];
            comparisonCounterBubble = new float[5];
            comparisonCounterHeap = new float[5];
            comparisonCounterInsertion = new float[5];
            comparisonCounterMerge = new float[5];
            comparisonCounterQuick = new float[5];
            comparisonCounterSelection = new float[5];
            algorithms = _algorithms;
            arraySize = _array_size;
            testRepetitions = _n_rep;
            min = _min;
            max = _max;
            testOrdinal = 0;
            complete = true;
        }
```

Programski kod 4.4. *Konstruktor klase AlgorithmTests*

Prije pokretanja samih algoritama testiranja, aplikacija postavlja traku za napredak tako da trenutni položaj postavi na početak (0), te za krajnji položaj uzima vrijednost jednaku umnošku broju testova i broja odabranih algoritama. Zatim se pokreće petlja koja se ponavlja onoliko puta koliki je broj testova odabran.

Prvi korak testiranja je stvaranje polja pseudo slučajnih cijelih brojeva. Duljinu polja, kao i minimalnu i maksimalnu vrijednost određuju vrijednosti unesene na grafičko sučelje od strane korisnika. Zatim se za svaki algoritam provjerava je li odabran, te je li korisnik putem sučelja

zatražio prekid testiranja. Ako obje provjere dozvole testiranje algoritma - aplikacija stvara kopiju generiranog polja, pokreće štopericu nakon što je vrati na početak (0), te iz statične pomoćne klase *Algorithms* pokreće algoritam za sortiranje, te mu predaje potrebne argumente. Po završetku algoritma zaustavlja se štoperica, sprema se vrijeme trajanja algoritma i broj usporedbi koje je algoritam izvršio. Metoda *Thread.Sleep* vidljiva na kraju testa poziva se da bi se spriječile greške prilikom prikazivanja rezultata testiranja na grafove, koja se događa ako su testovi iznimno kratki.

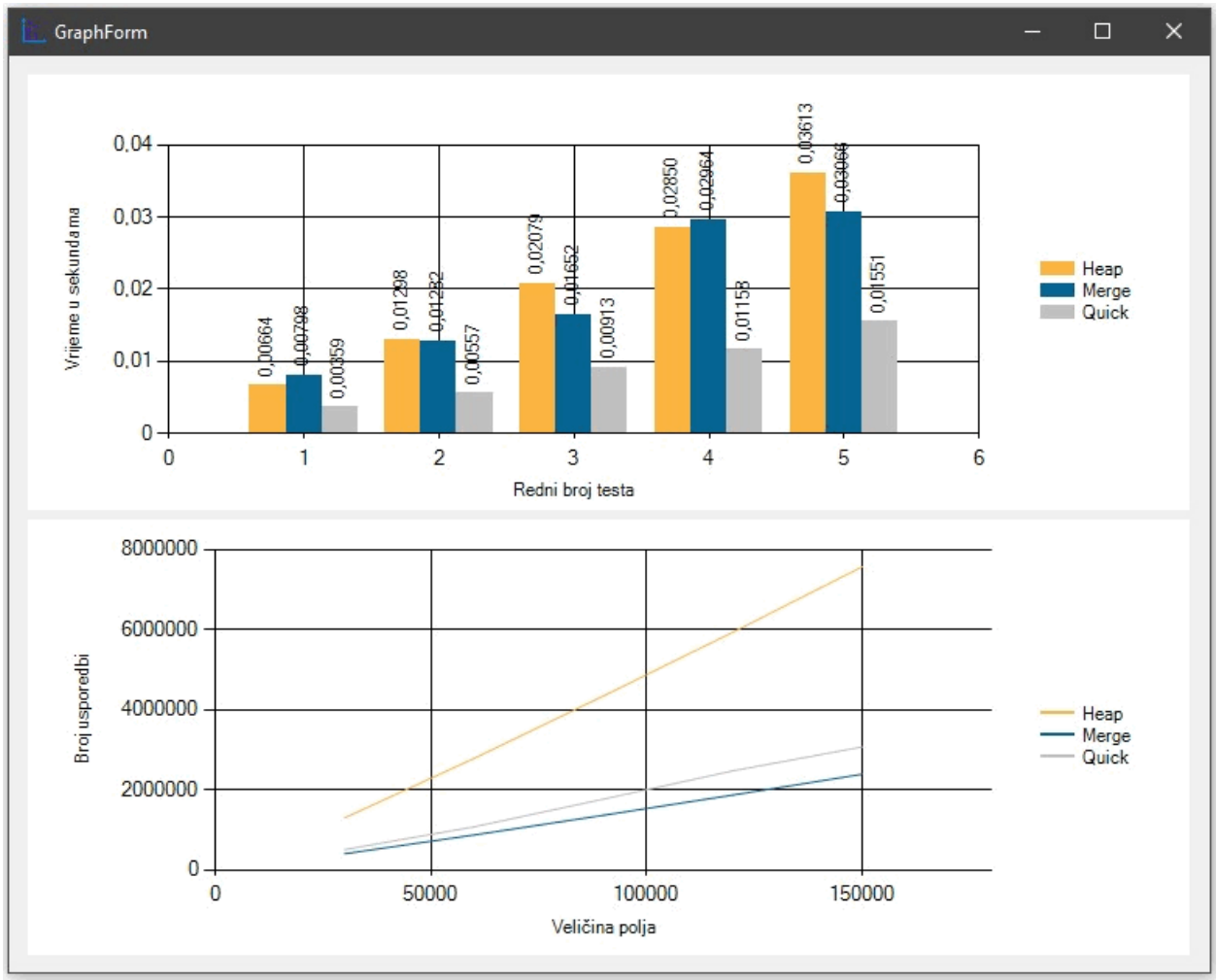
```
public void RunTests()
{
main.SetProgressBar();
while (testOrdinal < testRepetitions && !main.backgroundWorker.CancellationPending)
       {
       CreateArray();
       if(algorithms[0] && !main.backgroundWorker.CancellationPending)
              {
              array_copy = (int[])array.Clone();
              swatch.Reset();
              swatch.Start();
              comparisonCounterBubble[testOrdinal] =
                     Algorithms.BubbleSort(array_copy, arraySize[testOrdinal]);
              swatch.Stop();
              timeBubble[testOrdinal] = swatch.Elapsed.TotalSeconds;
              main.graphForm.DisplayDataToGraph("Bubble", testOrdinal,
                     timeBubble[testOrdinal], arraySize[testOrdinal],
                     comparisonCounterBubble[testOrdinal]);
              main.testProgress.PerformStep();
              Thread.Sleep(100);
              }
```

Programski kod 4.5. *Isječak metode RunTests. U prikazanom dijelu nazali se poziv metode CreateArray koja stvara polje brojeva koji se sortiraju, te primjer poziva algoritma Bubblesort*

Rezultati testiranja prikazuju se pomoću obrasca odnosno klase *GraphForm* prikazane na slici 4.3. Ovaj obrazac sadrži dva grafa, jedan stupčasti i jedan linijski. Na stupčastom grafu prikazuju se redni broj testiranja na *x* osi i vrijeme izvršavanja algoritma na *y* osi. Na linijskom grafu prikazuju se duljina polja na *x* osi i broj usporedbi na *y* osi. Funkcionalno, klasa *GraphForm* sadrži tri metode, od kojih dvije prikazuju podatke na grafovima, jedna rezultate testova koji se izvršavaju u stvarnom vremenu, druga podatke učitane iz datoteke. Zadnja metoda, prikazana programskim kodom 4.6., koristi se za prikaz legende na grafu, tako da bi se vidjela samo legenda za odabrane algoritme.

```
public void EnableChartSeries()
{
for (int i = 0; i < 6; i++)
        {
        if (testRef.algorithms[i])
                {
                barChart.Series[i].Enabled = true;
                lineChart.Series[i].Enabled = true;
                }
        }
}
```

Programski kod 4.6. *Metoda EnableChartSeries*



Slika 4.3. *Obrazac GraphForm na kojemu se prikazuju rezultati testiranja*

Po uspješnom završetku testiranja, prikazuje se poruka koja obavještava korisnika o završetku testiranja i trajanju testiranja. Također se omogućuje gumb za ponovno pokretanje testa, te gumb

za spremanje rezultata testiranja. Rezultati testiranja se spremaju u datoteku tako da se objekt *test* serijalizira u *.json* oblik. Spremanje vrši metoda *WriteJSON*, vidljiva u kodu 4.7.

```
private void WriteJSON(AlgorithmTests test)
{
saveFileDialog.FileName = newJsonFileName;
saveFileDialog.InitialDirectory =
       Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
string json = JsonConvert.SerializeObject(test);
if (saveFileDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
       {
       TextWriter writeDataToFile = new StreamWriter(saveFileDialog.OpenFile());
       writeDataToFile.Write(json);
       writeDataToFile.Close();
       }
}
```

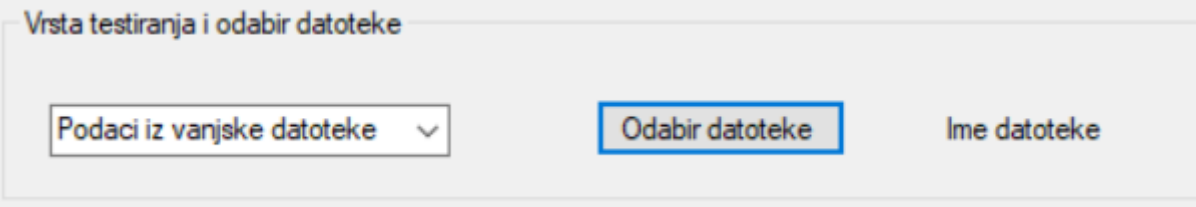
Programski kod 4.7. *Metoda WriteJSON*

5. KORIŠTENJE APLIKACIJE

U ovom poglavlju opisan je način korištenja aplikacija, odnosno koja je svrha svake kontrole na korisničkom sučelju. Također su prikazani i opisani rezultati dobiveni ovom aplikacijom.

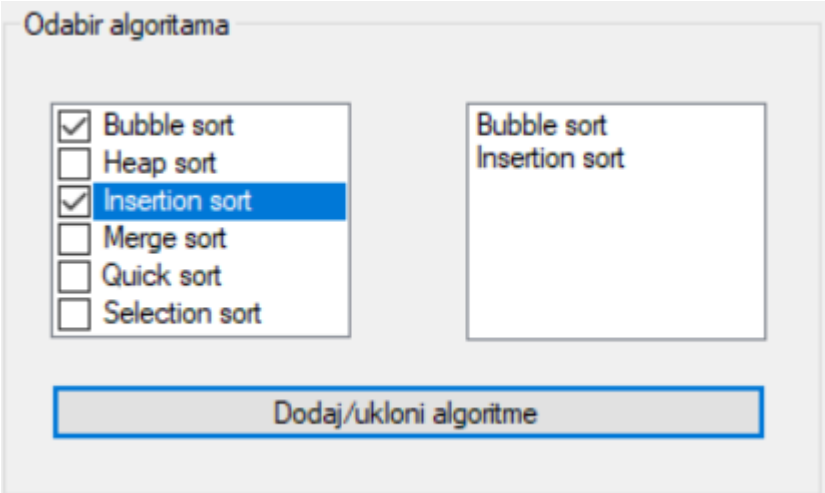
5.1. Postavljanje testiranja

Odabirom „Podaci iz vanjske datoteke“ (slika 5.1.) tipka „Odabir datoteke“ postaje omogućena. Klikom na isti otvara se dijaloški okvir za odabir vanjske datoteke. Kao što je navedeno u poglavlju 4, datoteka mora biti u *.json* obliku i ispravno popunjena da bi je aplikacija mogla učitati. Ako je učitavanje datoteke uspješno, korisničko sučelje se popunjava s vrijednostima iz datoteke, te se rezultati testiranja prikazuju na grafovima.

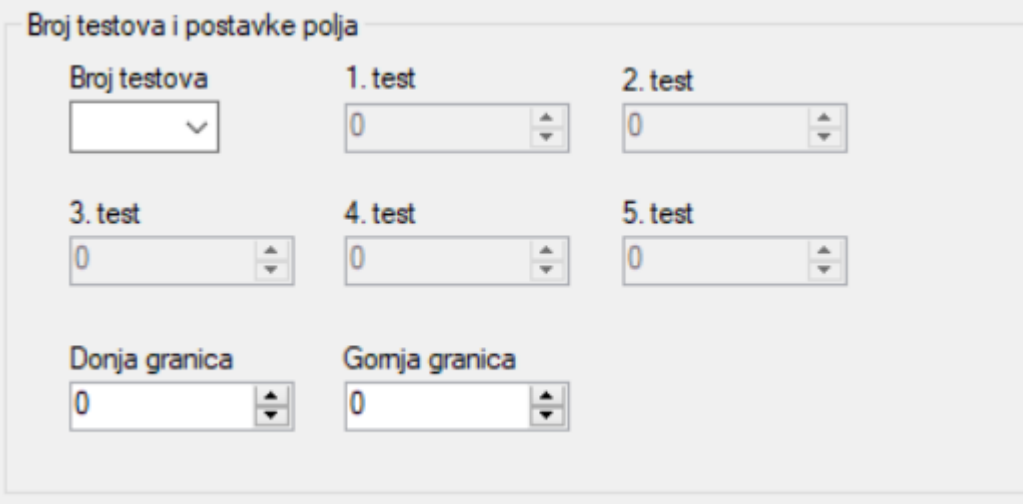


Slika 5.1. *Grupa kontrola „Vrsta testiranja i odabir datoteke“ s pripadajućim padajućim izbornikom i gumbom*

Odabirom „Pseudo slučajni brojevi“ omogućuje se odabir algoritama (slika 5.2.), te padajući izbornik za odabir broja testova (slika 5.3.). Algoritme je potrebno označiti na popisu algoritama, a da bi oni bili odabrani potrebno je kliknuti na tipku „Dodaj/ukloni algoritme“. Odabirom neke vrijednosti u broju testova omogućuje se unos duljine polja, za onoliko polja koliki je odabran broj testova, odnosno ako je odabran broj testova 2, omogućiti će se unos u prve dvije kućice za unos duljine polja. Također se omogućuju kućice za unos minimalne i maksimalne vrijednosti polja, te tipka za pokretanje testa. U slučaju da je neka od omogućenih kontrola neispunjena ili neispravno ispunjena, aplikacija prikazuje dijaloški okvir pogreške, te upućuje korisnika kako je ispraviti.



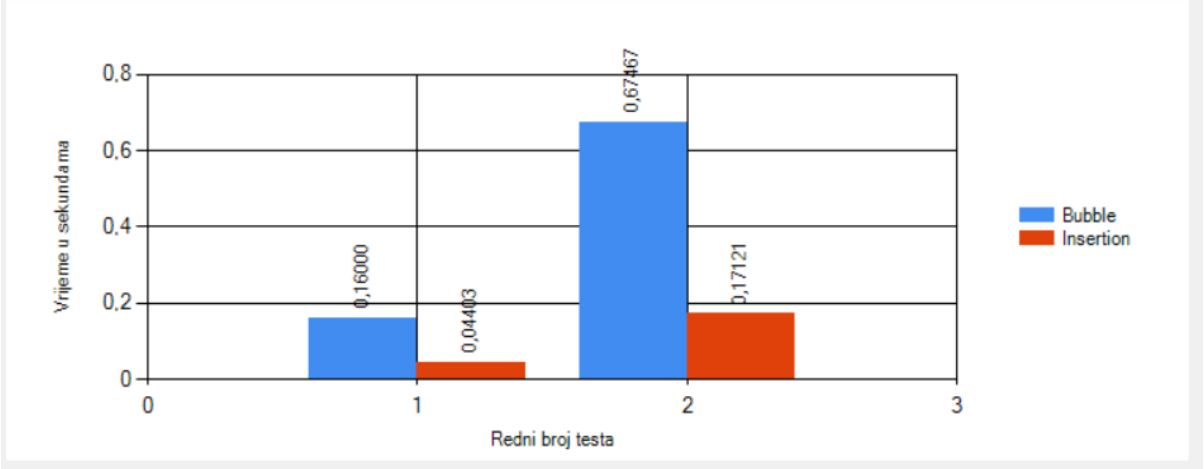
Slika 5.2. *Grupa kontrola „Odabir algoritama“ s pripadajućim popisom algoritama, prikazom odabranih algoritama i gumbom za dodavanje ili uklanjanje*



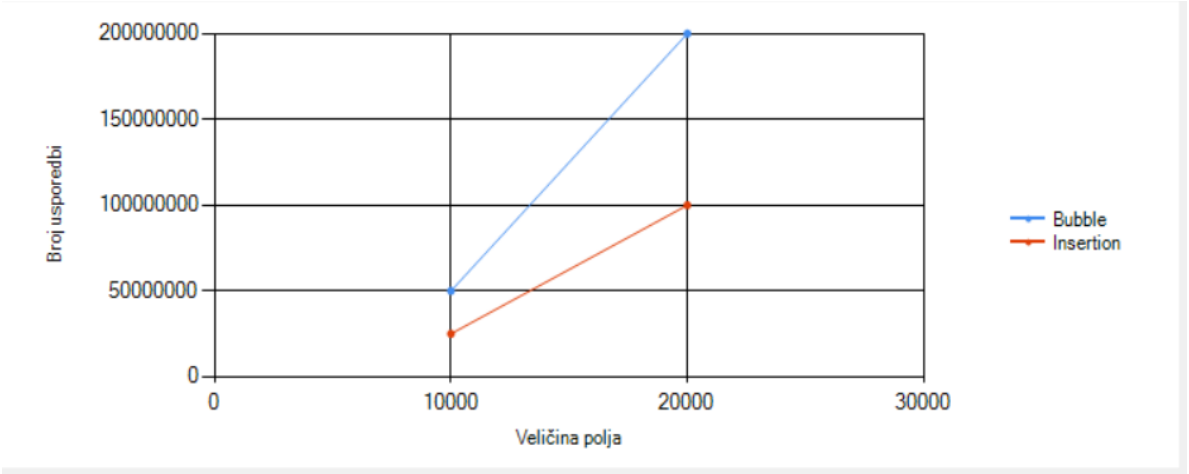
Slika 5.3. *Grupa kontrola „Broj testova i postavke polja“ s pripadajućim padajućim izbornikom za broj testova, te poljima za unos brojeva za veličinu polja i gornju odnosno donju granicu*

U slučaju da je test uspješno pokrenut (slika 5.4.), aplikacija učitava podatke iz kontrola s korisničkog sučelja, po potrebi se omogućuju ili onemogućuju tipke, postavlja se brojač koji mjeri ukupno vrijeme izvršavanja testa, te se testiranje asinkrono pokreće na pozadinskoj niti što korisniku omogućuje interakciju s obrascem za vrijeme testiranja. Podaci se zatim prikazuju na novom obrascu s grafovima (slike 5.5. i 5.6.), dok se traka za napredak pomiče za 1 korak.

Slika 5.4. *Grupa kontrola „Upravljanje testom“ s pripadajućim gumbima za pokretanje i zaustavljanje testa, te gumbom za spremanje rezultata testa*



Slika 5.5. *Stupčasti graf za prikaz rezultata testiranja, na x osi prikazuje se redni broj testa, dok se na y osi prikazuje trajanje testa u sekundama*



Slika 5.6. *Linijski graf za prikaz rezultata testiranja, na x osi prikazuje se veličina ulaznog polja, dok se na y osi prikazuje broj usporedbi koje je algoritam izvršio*

Korisnik nema nikakvu interakciju s obrascem s grafovima, već on služi samo za prikaz rezultata testiranja. Ovaj obrazac postaje vidljiv tek kada korisnik pokrene testiranje, te se po završetku izvršavanja svakog pojedinog algoritma na grafovima prikazuju rezultati testiranja. U slučaju stupčastog grafa, na *x* osi nalazi se redni broj testa, a na *y* osi vrijeme sortiranja. Na *x* osi

linijskog grafa prikazuje se veličina polja, dok se na *y* osi istog prikazuje broj usporedbi brojeva koje je algoritam izvršio da bi sortirao polje.

Za razliku od glavnog obrasca koji se otvara samo jednom i koji mora ostati otvoren da bi aplikacija ispravno radila, o obrascu s grafovima ne ovisi nikakva funkcionalnost aplikacija, te se prilikom svakog pokretanja testiranja stvara novi obrazac da bi korisnik mogao usporediti rezultate više različitih testova.

Po završetku svih odabranih algoritama zaustavlja se štoperica svih testova, prikazuje se dijaloški okvir s porukom o uspješnom završetku testiranja te ukupnom vremenu koje je prošlo od početka testiranja. Kao i pri početku testiranja, određene tipke se omogućuju, dok se druge onemogućuju.

Također se omogućuje i tipka za spremanje rezultata testa. Pritiskom na ovu tipku se spremaju postavke isključivo zadnjeg testa (broj algoritama, duljina polja itd.), ali se spremaju i rezultati svih testova koji su vidljivi na grafovima. Svi podaci se spremaju u datoteku u *.json* obliku. Zadano ime datoteke je test_datum_vrijeme (datum i vrijeme se dohvaćaju na pritisak tipke za pokretanje testova), a zadana adresa je radna površina (engl. *desktop*) trenutno prijavljenog korisnika, no korisnik može promijeniti i ime i adresu koristeći dijaloški okvir za spremanje datoteke.

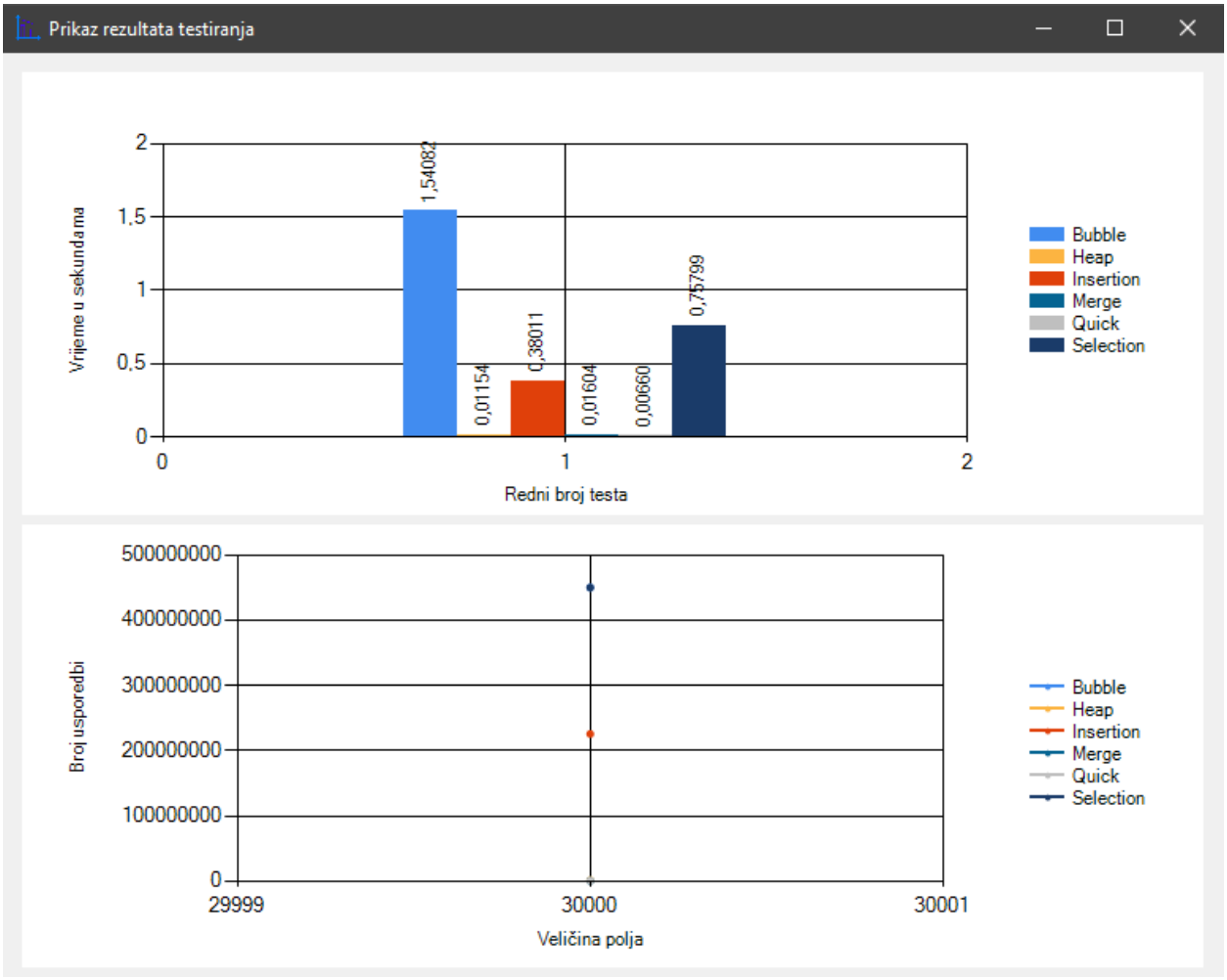
5.2. Rezultati testiranja

Ovo potpoglavlje bavi se problematikom testiranja aplikacije izrađene u sklopu završnog rada. U sklopu testiranja aplikacije izvršena su četiri testa, svi na istom računalu čije specifikacije su navedene u nastavku:

- Proizvođač i model: HP Pavilion g7-2003sm
- Procesor: Intel i5-3210M 2.5GHz
- Radna memorija: 8GB DDR3 1600MHz
- Grafičke kartice: Intel HD4000 (integrirana)
  AMD Radeon 7670M 1GB (diskretna)
- Pohrana podataka: Samsung 860 Evo SSD

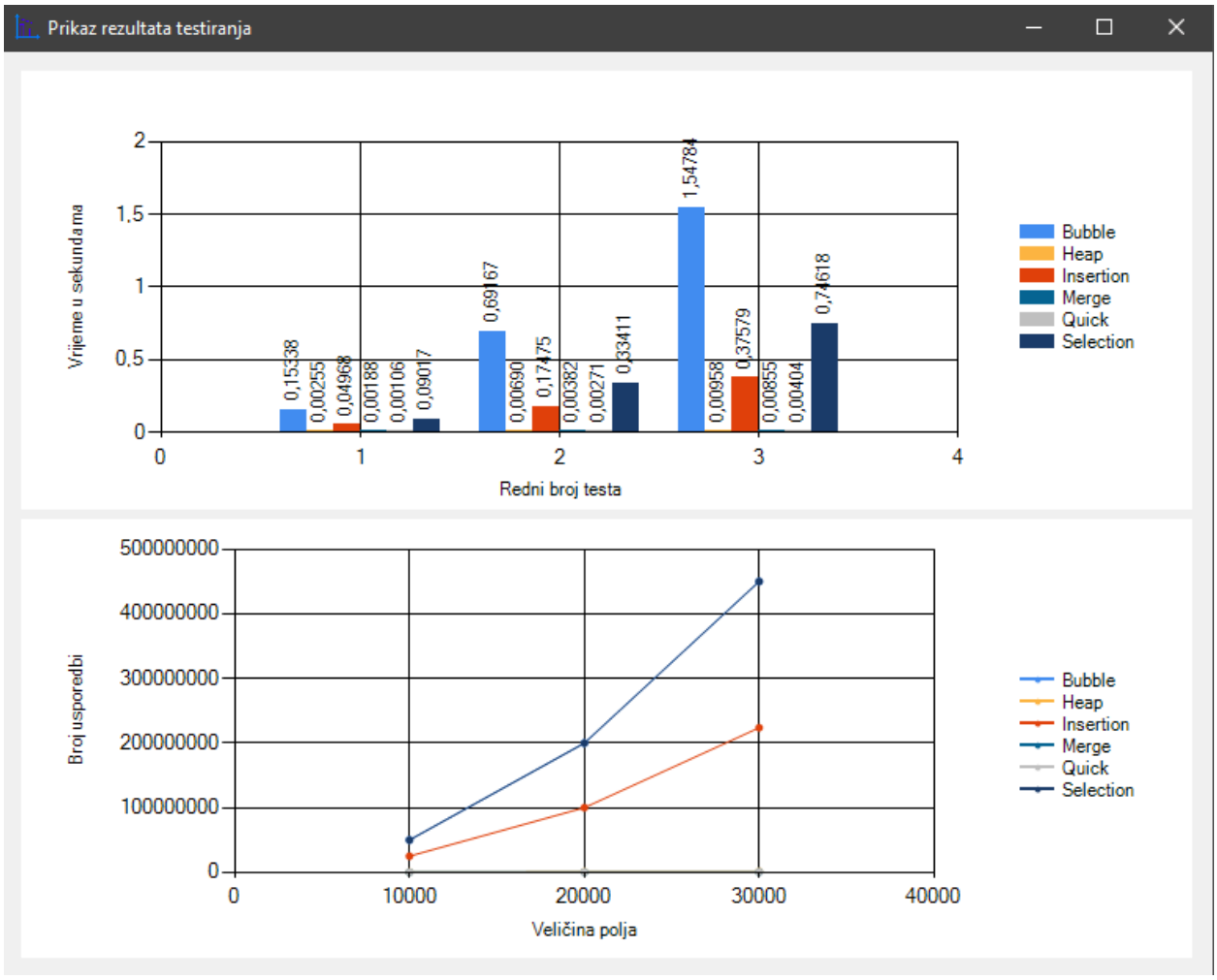
U prvom testu prikazanom na slici 5.7. odabrani su svi algoritmi sortiranja, te jedno ulazno polje duljine 30.000 brojeva. Ako se prisjetimo veliko O notacije svakog algoritma, rezultati testa postaju očiti. „Brzi“ algoritmi – quick sort, merge sort i heap sort su neusporedivo brži od „sporih“ algoritama – bubble sorta, selection sorta i insertion sorta. Najsporiji u testu je bio bubble sort,

kojemu je trebalo 1,54 sekundi za sortiranje polja od 30.000 brojeva, dok je najbrži bio quick sort kojemu je trebalo svega 6,6 milisekundi za sortiranje istog polja – što je čak 233 puta brže.



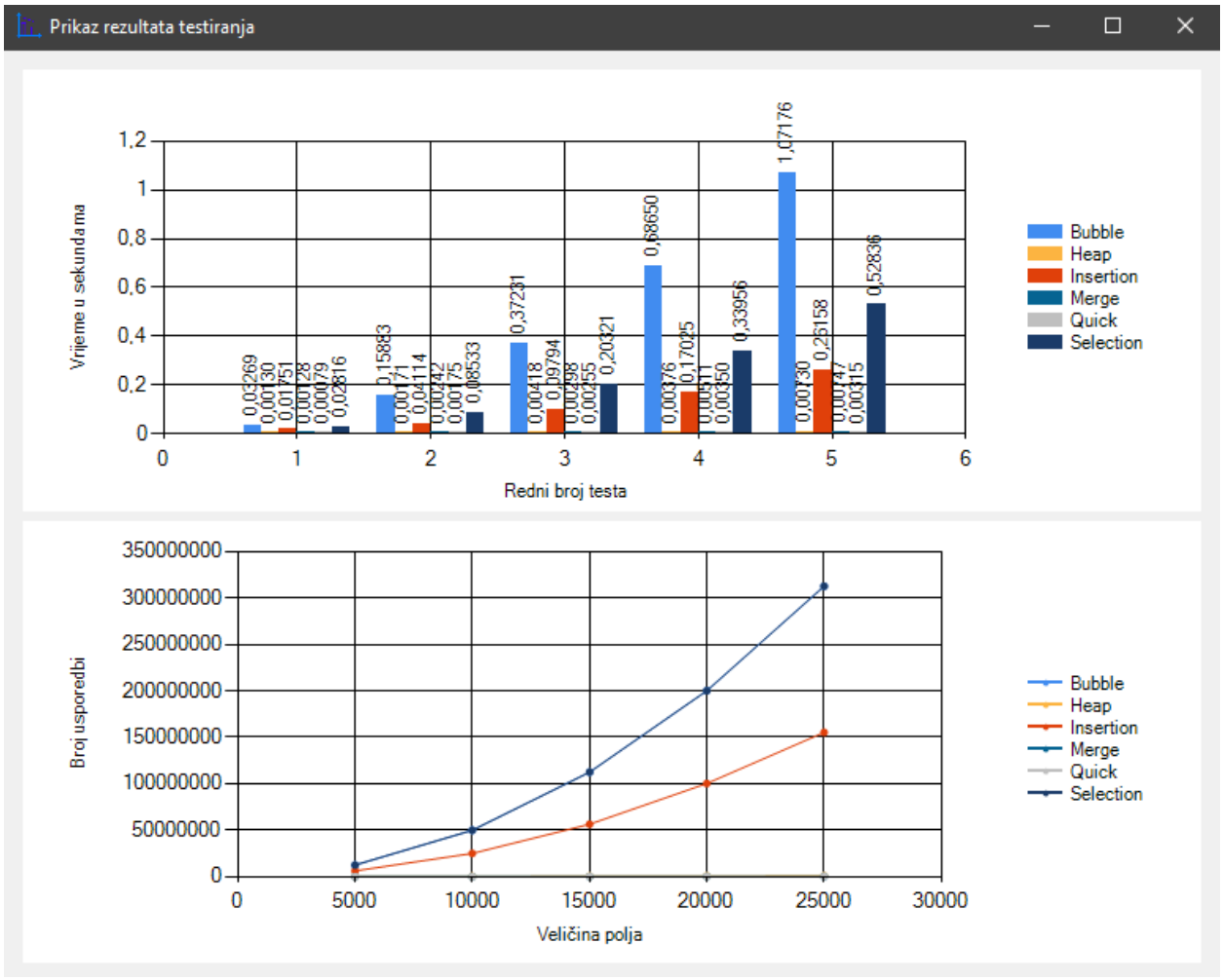
Slika 5.7. *Prikaz rezultata testiranja, svi algoritmi, jedno ulazno polje duljine 30000 elemenata*

U drugom testu prikazanom na slici 5.8. odabrani su svi ponuđeni algoritmi, te 3 ulazna polja veličina 10.000, 20.000 te 30.000. U ovome testu nastavlja se isti trend kao i u prvom testu, odnosno „brzi“ algoritmi su ponovno neusporedivo brži od „sporih“. Iako su rezultati naizgled identični onima u prvom testu, u ovome testu se može primijetiti da smanjenjem duljine polja smanjujemo i razliku između najbržeg i najsporijeg algoritma, koja sada iznosi 144 puta.



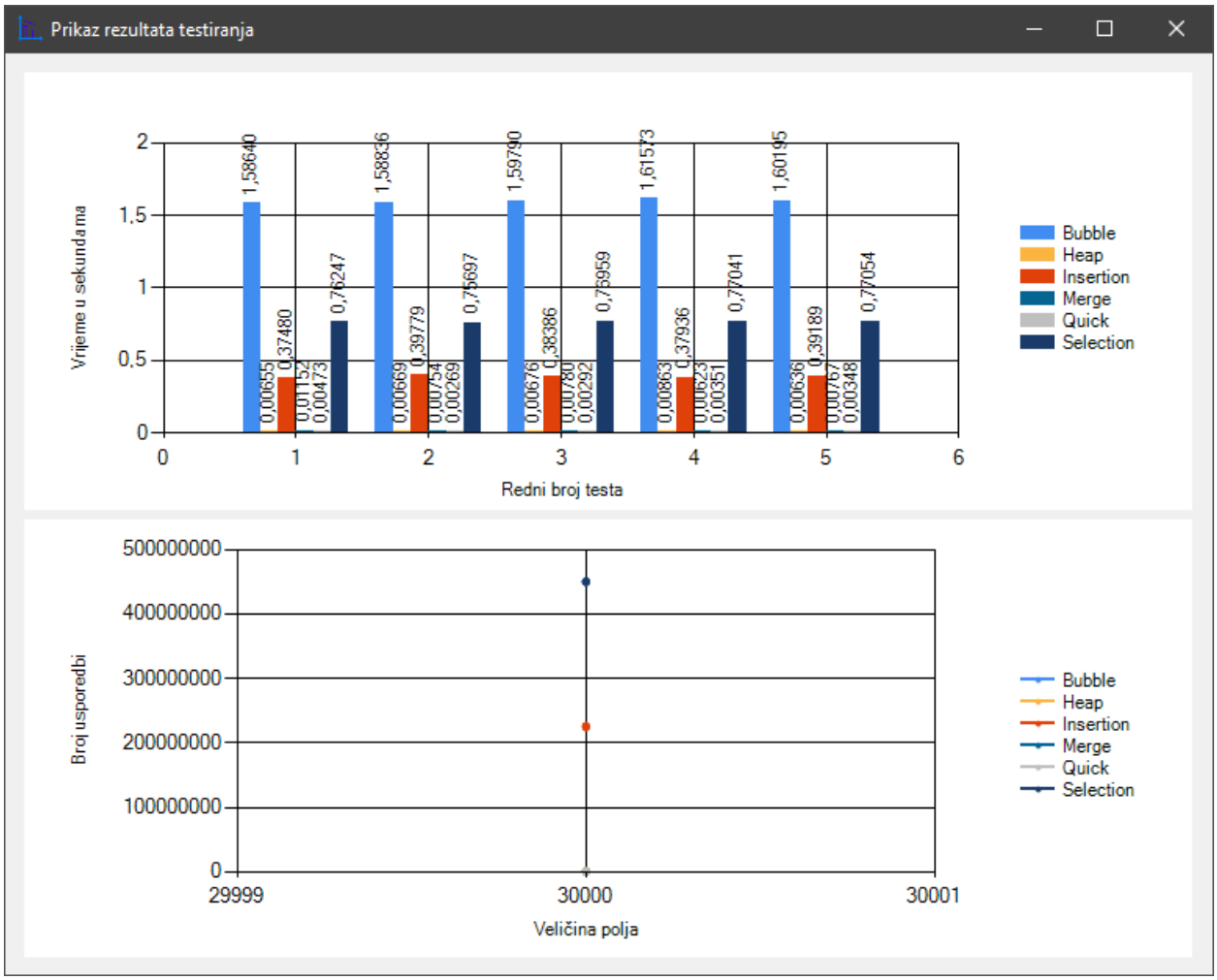
Slika 5.8. *Prikaz rezultata testiranja, svi algoritmi, tri ulazna polja duljina 10000, 20000 i 30000 elemenata*

U trećem testu prikazanom na slici 5.9. odabrani su svi ponuđeni algoritmi, te 5 ulaznih polja duljina 5.000, 10.000, 15.000, 20.000 i 25.000. Ovaj test nam potvrđuje rezultate dobivene prošlim testovima – razlika između brzih i sporih algoritama je još uvijek velika, ali smo ponovno smanjenjem duljine polja dobili i manju razliku, koja u slučaju duljine polja 5.000 iznosi 41 puta.



Slika 5.9. *Prikaz rezultata testiranja, svi algoritmi, pet ulaznih polja duljina 5000, 10000, 15000, 20000 i 25000 elemenata*

Zadnji test izvršen u sklopu ovog završnog rada prikazan na slici 5.10. služi kao prikaz izgleda obrasca s grafovima ako odaberemo sve ponuđene opcije i ponovimo istu duljinu polja u svih pet testova. Tako su u ovom testu odabrani svi algoritmi, te pet polja identične duljine – 30.000. Rezultati ovog testa također prikazuju da su mjerenja izvršena ovom aplikacijom izrazito točna, iako je nemoguće kontrolirati ulazno polje. Tako je standardna devijacija za *bubble sort* 0.0106 što je manje od 1%, dok je varijanca $1{,}11*10^{-4}$ što je manje od $7*10^{-5}$%.



Slika 5.10. *Prikaz rezultata testiranja, svi algoritmi, pet ulaznih polja duljine 30000*

6. ZAKLJUČAK

Cilj ovog završnog rada bio je izrada Windows Forms aplikacije koja će testirati šest odabranih algoritama sortiranja i prikazati rezultate izvršenih testova. Aplikacija izrađena u sklopu ovog završnog rada vrši zadanu funkciju, odnosno uspješno generira i sortira ulazna polja pomoću raznih algoritama sortiranja prema parametrima podešenim od strane korisnika, te uspješno prikazuje dobivene rezultate na grafovima i omogućava spremanje rezultata testiranja u datoteku. Koristeći ovu aplikaciju moguće je odrediti koji algoritam sortiranja najbolje odgovara određenoj situaciji i određenom korisniku, što je bitno jer odabirom krivog algoritma sortiranje se može izvršavati u nedogled. Na primjer, student koji treba sortirati polje s malom količinom podataka će vjerojatno odabrati *bubble sort*, pogotovo ako tek uči programirati, jer je najlakši za implementirati. Nasuprot tome – iskusni će programer uvijek posegnuti za nekim od brzih algoritama, posebno ako je potrebno sortirati iznimno veliku bazu koja sadrži stotine milijuna ili čak i više podataka. Također, aplikaciju je iznimno jednostavno koristiti – korisničko sučelje je intuitivno posloženo, te nema zbunjujućih opcija.

Aplikacija je izrađena u Visual Studio razvojnom okruženju, funkcionalni dio u C# programskom jeziku, a korisničko sučelje povlačenjem i ispuštanjem, te naknadnim uređivanjem svojstava kontrola. Radi lakšeg snalaženja na korisničkom sučelju, kontrole su grupirane u okvire, tako da je lako prepoznati što koja kontrola radi. Na glavnom obrascu se nalaze sve kontrole aplikacije. Aplikacija omogućava spremanje određenih podataka na korisnički odabranu lokaciju u $.json$ obliku te ih je moguće učitati samo s ovom aplikacijom.

LITERATURA

[1] Microsoft Docs – Visual Studio Documentation, https://docs.microsoft.com/en-us/visualstudio/?view=vs-2017, 23.6.2019.

[2] Dr. Z's Blog - How Many Developers Use C++ vs. C# vs. Other Programming Languages, https://blogs.msdn.microsoft.com/zxue/2016/10/24/how-many-developers-use-c-vs-c-vs-other-programming-languages/, 19.6.2019.

[3] Technopedia – Windows Forms, https://www.techopedia.com/definition/24300/windows-forms-net, 23.6.2019.

[4] Microsoft Docs – A Tour of the C# Language, https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/index, 23.6.2019.

[5] Newtonsoft – Json.NET, https://www.newtonsoft.com/json, 23.6.2019

[6] R. Manger, Strukture podataka i algoritmi, ELEMENT d.o.o., Zagreb, 2014.

[7] R. Manger, Strukture podataka i algoritmi, nastavni materijal, Sveučilište u Zagrebu, Zagreb, 2013.

[8] Geeks for Geeks – Selection Sort, https://www.geeksforgeeks.org/selection-sort/, 9.9.2019.

[9] Geeks for Geeks – Bubble Sort, https://www.geeksforgeeks.org/bubble-sort/, 9.9.2019.

[10] Geeks for Geeks – Insertion Sort, https://www.geeksforgeeks.org/insertion-sort/, 9.9.2019.

[11] Geeks for Geeks – Quick Sort, https://www.geeksforgeeks.org/quick-sort/, 9.9.2019.

[12] Geeks for Geeks – Merge Sort, https://www.geeksforgeeks.org/merge-sort/, 9.9.2019.

[13] Geeks for Geeks – Heap Sort, https://www.geeksforgeeks.org/heap-sort/, 9.9.2019.

SAŽETAK

Naslov: Windows Forms GUI aplikacija za testiranje algoritama sortiranja

U sklopu ovog završnog rada izrađena je Windows Forms aplikacija pomoću koje je moguće testirati šest različitih algoritama sortiranja. Za izradu je korišteno razvojno okruženje Microsoft Visual Studio i C# programski jezik. Ovisno o podacima koje korisnik unese u grafičko sučelje, aplikacije će testirati odabrane algoritme prema zadanim parametrima te prikazati dobivene rezultate na grafovima. Rezultati se također mogu spremiti u vanjsku datoteku, te kasnije učitati nazad u aplikaciju. Svrha ove aplikacije je pravilan izbor algoritma sortiranja za implementaciju, ovisno o potrebama samog programera.

Ključne riječi: algoritmi sortiranja, C#, sortiranje, Visual Studio, Windows Forms

ABSTRACT

Title: Windows Forms GUI application for testing sorting algorithms

A Windows Forms application was created as a part of this graduation paper. The application was developed using Microsoft Visual Studio IDE and C# programming language. It (the application) allows the user to test up to six different sorting algorithms with varying array lengths and min/max values depending on the user input into the graphical user interface. Test results will be displayed on two graphs contained on the user interface and can later be stored to an external file, which can be read by the same application. The goal of this application is to help users correctly choose a sorting algorithm depending on their needs.

Keywords: C#, sorting, sorting algorithms, Visual Studio, Windows Forms

ŽIVOTOPIS

Matija Almaši rođen je 1995. godine u Virovitici. Većinu svog života proveo je u Pakracu, gdje je pohađao osnovnu i srednju školu – smjer opća gimnazija. Po završetku srednje škole, upisao je preddiplomski studij računarstva na tadašnjem Elektrotehničkom fakultetu u Osijeku, sada poznatom kao Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. Na prvoj godini, prebacio se na preddiplomski stručni studij elektrotehnike – smjer Informatika, koji je završio 2019. godine. Na ljeto 2019. godine pohađao je stručnu praksu u Mono d.o.o.-u gdje je u timu s drugim studentima izradio ASP .NET Core web aplikaciju. Također je izradio više manjih projekata u raznim jezicima, što u svrhu polaganja kolegija ili ostvarivanja bodova, tako i u privatne svrhe.

Matija Almaši

PRILOZI

Na optičkom disku, priloženom uz ispisanu verziju završnog rada, nalazi se projektna mapa s izvornim kodom, te završni rad u .docx i .pdf obliku.