Математичка гимназија

МАТУРСКИ РАД

-из предмета Програмирање и програмски језици-

Графички приказ алгоритама за сортирање

Ученик: Вања Крстајић 4ц Ментор: Снежана Јелић

Београд, мај 2025.

Садржај

1 Увод..... 2

2 Упутство за коришћење апликације..... 3

3 Алгоритми за сортирање..... 6

3.1 Selection sort..... 6

3.2 Insertion sort..... 7

3.3 Bubble sort..... 9

3.4 Quick sort..... 10

3.5 Остали алгоритми..... 11

4 Програмирање..... 12

4.1 Форма..... 12

4.2 Класа Сорт..... 13

4.3 Класа Елемент..... 14

5 Закључак..... 15

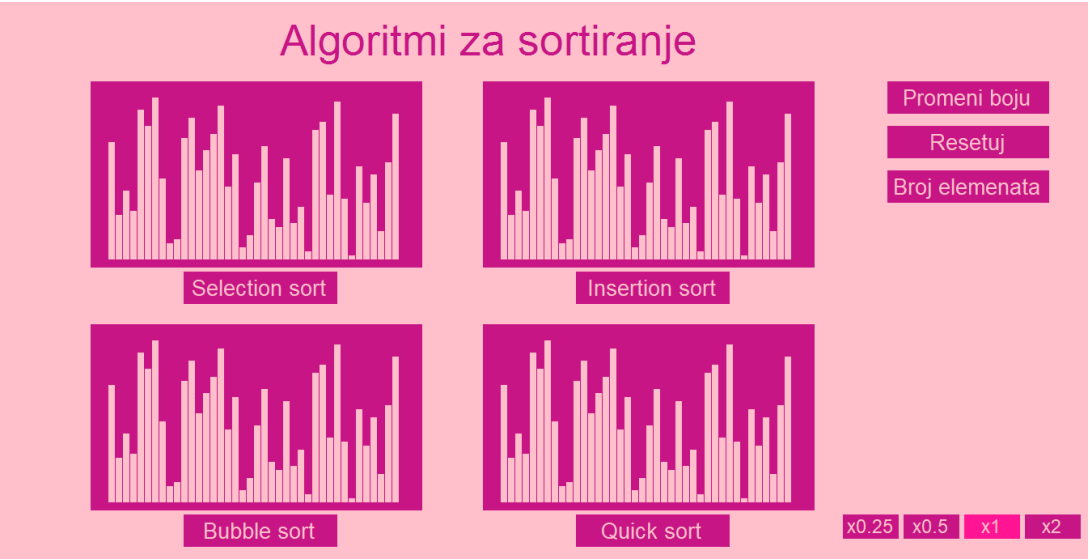
Литература..... 16

1 Увод

Идеја сортирања постоји много пре него што су се појавили први модерни рачунари. Још у старим цивилизацијама, људи су имали потребу да организују податке – било да су то спискови, роба или записи. Међутим, развој алгоритама за сортирање почиње у правом смислу тек са појавом програмабилних рачунара у 20. веку.

Као што сам већ напоменула, развој алгоритама за сортирање је директно повезан не само са развојем рачунара, него и са потребом за што ефикаснијом обрадом велике количине података. Неке методе су интуитивне и лако разумљиве, као што је **Bubble sort**, али са растом количине података, ови алгоритми постају неефикасни. Са друге стране, постоје и сложенији алгоритми као што су **Merge sort** и **Quick sort** који нуде много боље перформансе, али су и теже за разумевање и имплементацију. Због тога сам одлучила да креирам графички приказ који ће визуализовати рад ових алгоритама, како би процес сортирања био лакше схватљив и интерактиван.

У оквиру овог матурског рада развијена је апликација у програмском језику **C#**, чији је циљ графичка визуализација рада алгоритама. Апликација омогућава кориснику да на интуитиван и визуелно прегледан начин прати како поједини алгоритми функционишу, корак по корак. Одабрала сам четири алгоритма - **Selection sort**, **Insertion sort**, **Bubble sort** и **Quick sort**. То су алгоритми који су ми били најзанимљивији као и које смо највише обрађивали у школи, тако да би овај програм могао да послужи и као помоћ у учењу.



Слика 1- Приказ интерфејса

2 Упутство за коришћење апликације

Ова апликација омогућава кориснику да визуелно прати рад различитих алгоритама сортирања над низом бројева. У наставку је опис функционалности сваког елемента корисничког интерфејса:

1) Избор броја елемената у низу

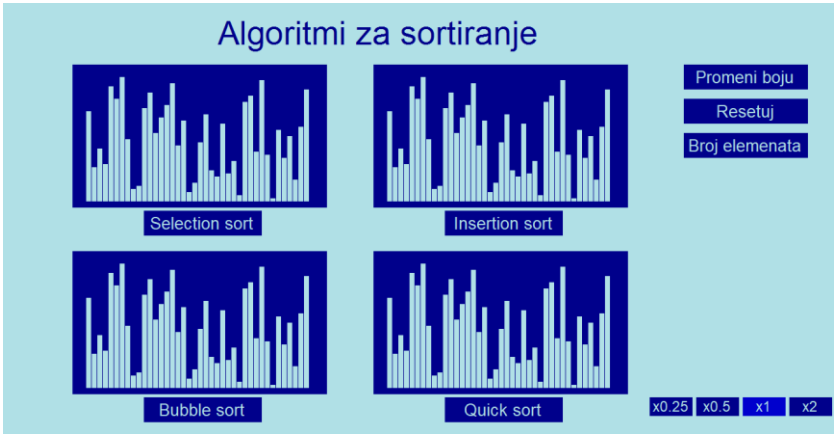
    - Кликом на дугме **Број елемената** отвара се падајући мени у коме корисник може да изабере једну од понуђених вредности: **10, 20, 30, 40 или 50** (када се апликација покрене има 40 елемената)
    - Избором било ког броја, креира се нови низ са тим бројем елемената
    - Након избора, падајући мени аутоматски нестаје
    - Падајући мени такође може да нестане кликом поново на дугме **Број елемената**

2) Ресетовање низа

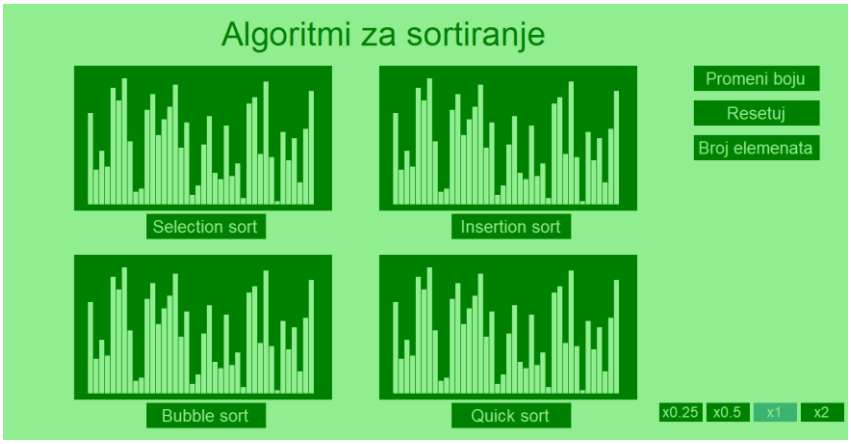
    - Дугме **Ресетуј** генерише нови случајни низ са исто елемената као претходни

3) Промена боје интерфејса - дугме **Промени боју** мења боју целе апликације у следећем редоследу:

    - Почетна боја: **розе**
    - Први клик: **плава**
    - Други клик: **зелена**
    - Након тога се циклус понавља



Слика 2 – Плави интерфејс



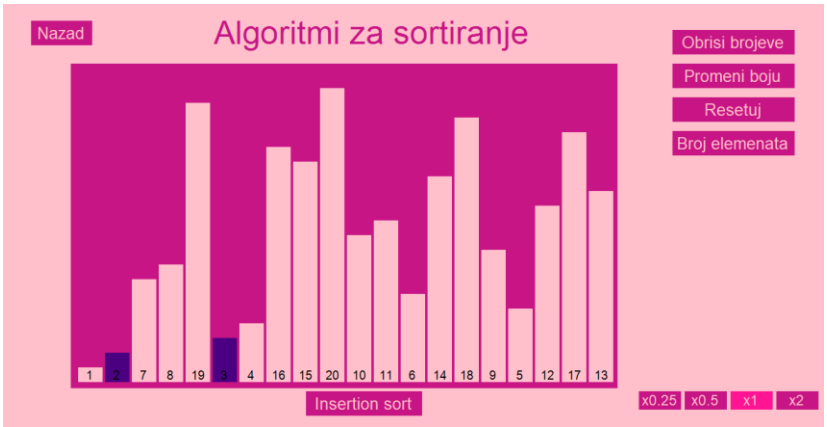
Слика 3 – Зелени интерфејс

4) Покретање и паузирање алгоритма

- Сваки алгоритам има своје дугме
- Кликом на дугме са називом алгоритма, алгоритам започиње сортирање приказаног низа у правоугаонику изнад
- Још један клик на исто дугме паузира извршавање алгоритма

5) Прелазак у режим једног алгоритма

- Кликом на било који од правоугаоника у главном менију, апликација прелази у режим приказа само тог једног алгоритма
- У режиму једног алгоритма постоји дугме **Назад**, којим се корисник враћа у **главни мени**



Слика 4 – Режим једног алгоритма

6) Приказивање бројева

- У оквиру режима једног алгоритма се појављује дугме **Прикажи бројеве** чијим кликом се на стубићима појављују њихове вредности

- Онда се оно претвара у дугме **Обриши бројеве**, чијим се кликом ови бројеви бришу

7) Контрола брзине извршавања - у доњем десном углу налазе се четири дугмета:

- **x0.25** — најспорије извршавање
- **x0.5** — спорије извршавање
- **x1** — нормална брзина *(подразумевана при покретању апликације)*
- **x2** — убрзано извршавање

Тренутна брзина је означена другом бојом.

3 Алгоритми за сортирање

Алгоритми за сортирање представљају основни део рачунарских наука и постоји велики број различитих приступа решавању овог проблема. Иако им је циљ исти — довести елементе неког скупа у одређени редослед (нпр. од најмањег до највећег) — они се међусобно разликују по начину рада, ефикасности и примени.

Битан фактор јесте **временска сложеност**. Она се мери у броју корака који алгоритам извршава у односу на број елемената n - **O(број корака)**. Она углавном иде од O(n log n) за сложеније и ефикасније алгоритме до $O(n^2)$ за не тако ефикасне за велики број података.

Још неке важне карактеристике алгоритама:

- **Просторна сложеност** – меморијска искоришћеност
- **Стабилност** - алгоритам је стабилан ако не мења релативни редослед елемената са истом вредношћу
- **Итеративни** или **рекурзивни** приступ
- **Прилагодљивост** – да ли брзина сортирања зависи од почетног стања низа
- **Паралелизација** – да ли је алгоритам погодан за паралелно извршавање неких корака

3.1 Selection sort

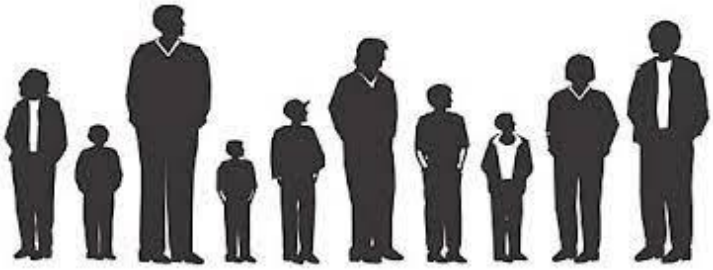
Selection sort (Селекционо сортирање) представља једно од једноставнијих алгоритама.

Принцип рада:

1. Проћи кроз цео низ и наћи најмањи елемент
2. Најмањи елемент заменити са првим елементом
3. Потом проћи кроз преостали део низа (без првог елемента) и поновити поступак – пронаћи најмањи елемент и заменити га са другим елементом у низу
4. Понављати поступак док цео низ не буде сортиран.

```
for (int i = 0; i < n - 1; i++)
{
    for (int j = i + 1; j < n; j++)
    {
        if (a[j].Vrednost < a[i].Vrednost)
            a[i].razmeni(a[j]);
    }
}
```

Слика 5 – Код за Selection sort



Слика 6 – Selection sort у сортирању ученика на физичком по висини

Упркос томе што је једноставан и за схватање и за програмирање, овај алгоритам није баш ефикасан. Временска сложеност је увек **O(n^2)**, и када је низ делимично сортиран, јер он увек пролази кроз преостали део низа. То га чини јединим алгоритмом код кога брзина сортирања не зависи од почетног стања низа. Просторна сложеност је **O(1)** – алгоритам је ***in-place***, што значи да не захтева додатни простор за сортирање.

Особине:

- **Лоша временска сложеност** - O(n^2)
- **Није прилагодљив**
- **Добра просторна сложеност** - O(1)
- **Није стабилан**

3.2 Insertion sort

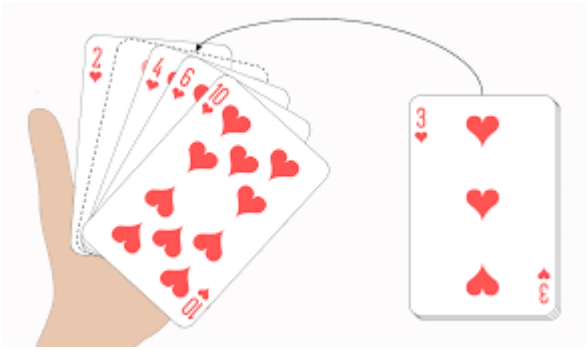
Insertion sort (Сортирање уметањем) се заснива на уметању једног по једног елемента на одговарајуће место унутар већ сортираног дела низа. Сличним начином на пример сортирамо карте у руци.

Принцип рада:

1. Узима се један по један елемент из несортираног дела
2. Убацује се на одговарајуће место у већ сортираном делу низа
3. Сваки наредни елемент се гура док не нађе своје место

```
for (int i = 1; i < n; i++)
{
    for (int j = 0; j < i; j++)
    {
        if (a[j].Vrednost > a[i].Vrednost)
            a[j].razmeni(niz_insertion[i]);
    }
}
```

Слика 7 – Код за Insertion sort



Слика 8 - Insertion sort у картању

Просечна временска сложеност је иста као код Selection sort-a, али је разлика прилагодљивост. Уколико Insertion sort као улаз прими већ сортиран алгоритам, он ће само једном прећи кроз цео низ и временска сложеност ће уместо $O(n^2)$ бити $O(n)$.

Особине:

- **Лоша временска сложеност** – у просеку $O(n^2)$
- **Прилагодљив је** – најбржи кад је већ сортиран низ, а најспорији ако су елементи сортирани у супротном поретку.
- **Добра просторна сложеност** - $O(1)$
- **Стабилан је**

Начин да овај алгоритам буде мало ефикаснији јесте коришћењем **бинарне претраге**, такозвани **tree sort**. Пошто је леви део низа већ сортиран, бинарна претрага брзо проналази место где елемент треба да се убаци. То смањује број поређења са $O(n)$ на $O(\log n)$ по елементу, али не и број замена (он и даље остаје $O(n^2)$).

3.3 Bubble sort

Bubble sort (Сортирање мехуром) се заснива на замени места суседних елемената уколико нису у добром међусобном поретку.

Принцип рада:

1. Креће се од десне стране према левој.
2. Упоређују се суседни елементи и ако је десни елемент мањи од левог, мењају им места.
3. Тако најмањи елемент испливава на левој страни после сваке итерације.
4. Поновити поступак, али без првог елемента (јер је он већ на свом месту)
5. Понављати овако док низ не буде сортиран

Ово је такозвани *обрнути Bubble sort* који се од обичног разликује само по томе да ли најмањи елемент испливава на левој страни или највећи на десној.

```
for (int i = 0; i < n - 1; i++)
{
    for (int j = n - 1; j > i; j--)
    {
        if (a[j].Vrednost < a[j - 1].Vrednost)
            a[j].razmeni(a[j - 1]);
    }
}
```

Слика 9 – Код за обрнути Bubble sort

Особине:

- **Лоша временска сложеност** – у просеку $O(n^2)$
- **Прилагодљив је** - најбржи кад је већ сортиран низ (O(n)) , а најспорији ако су елементи сортирани у супротном поретку ($O(n^2)$)
- **Добра просторна сложеност** - O(1)
- **Стабилан је**

Постоји унапређена верзија Bubble sort-а која се назива **Shaker sort** или **Cocktail sort** (а некад и **двосмерни Bubble sort**). Принцип рада је најпре гурање највећег елемента на крај (као класични Bubble sort), а одмах затим враћање са десна на лево ради изгуривања најмањег елемента на почетак (као обрнути Bubble sort). Ова два корака се понављају док се низ не сортира. Пошто гура елементе на обе стране има мањи број пролаза, али је опет иста сложеност јер се број замена не смањује. Погодан је за коришћење уколико су мали бројеви *заглављени* на крају.

3.4 Quick sort

Quick sort (Брзо сортирање) је доста ефикаснији од претходно споменутих алгоритама, а користи се једноставном идејом – да су сви елементи из првог дела низа мањи од свих елемената из другог дела, а поступак се примењује док низ не буде сортиран. Назив је добио тако што је његов творац, **Сер Чарлс Ентони Ричард Хор**, био одушевљен његовом брзином.

Принцип рада:

1. Бирање пивота – бира се један елемент, на пример последњи, да буде **пивот** (елемент на основу кога ће се други елементи поредити)
2. Пролазак кроз низ најпре са леве стране и памћење индекса првог елемента који је већи од пивота (такозвани леви индекс)
3. Пролазак кроз низ са десне стране и памћење индекса првог елемента који није већи од пивота (такозвани десни индекс)
4. Уколико је леви индекс мањи од десног, елементи на тим позицијама се мењају и поступак се наставља све док индекси не буду једнаки
5. Мењају се елементи на позицијама левог индекса и пивота
6. Добијамо исти проблем мање димензије, па се опет позива функција за ова два низа
7. Поступак се понавља док низови имају више од једног елемента

```
void sortiraj_quick(int levi, int desni)
{   if (levi < desni)
    {
        int li = levi, di = desni, pivot = a[desni].Vrednost;
        do
        {
            while (a[li].Vrednost < pivot)
              li++;
            while a[di].Vrednost >= pivot && li < di)
              di--;
            if (li < di)
              a[li].razmeni(a[di]);
        }
        while (li != di);
        a[li].razmeni(a[desni]);
        sortiraj_quick(levi, li - 1);
        sortiraj_quick(li + 1, desni);
    }
}
```

Слика 10 – Код за Quick sort

Пошто се овај алгоритам своди на решавање проблема мање димензије, он спада у класу такозваних *Завади па владај* алгоритама.

Особине:

- **Добра временска сложеност** – у просеку O(n log n)
- **Прилагодљив је**

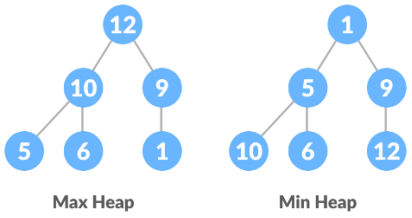
- **Није стабилан**
- **Добра просторна сложеност** - O(log n)
- **Рекурзиван приступ**

Иако је просечна временска сложеност O(n log n), што је јако добро, у најгорем случају она може достићи и $O(n^2)$. То је случај када се сваки пут за пивота бира највећи или најмањи елемент, то јест када је низ већ сортиран. То се може побољшати тако што ће за пивот бити изабран средњи по вредности од три насумична елемента.

3.5 Остали алгоритми

У овом раду сам споменула само четири алгоритма —**Selection sort, Insertion sort , Bubble sort** и **Quick sort**, који су довољни да покажу разлике између једноставнијих и напреднијих приступа сортирању. Неки од осталих алгоритама су

- **Merge sort –** алгоритам који дели низ на половине док не остане један елемент у сваком низу, и онда их спаја у сортиран низ – временска сложеност је O(n log n) у сваком случају
- **Shell sort** – унапређена верзија Insertion sort-а – пореди удаљене елементе, па постепено смањује растојање до суседа, чиме постиже брже сортирање него обичан Insertion sort – просечна временска сложеност је O(n log n) али у најгорем случају може бити и $O(n^2)$
- **Heap sort** – прави се бинарно стабло где сваки родитељ има већу вредност од своје деце, и одатле се извлачи највећи елемент и ставља на крај низа – временска сложеност је O(n log n) у сваком случају



Слика 11 – Приказ бинарног стабла у Heap sort-у

Поред ових класичних алгоритама постоје и **хибридни алгоритми**. То су алгоритми који комбинују више различитих метода како би постигли што боље перформансе у различитим ситуацијама. Ови хибридни алгоритми се најчешће користе у већ направљеним функцијама за сортирање у програмским језицима као што су Python, Java, C++ и други. У C++ се на пример користи **Intro sort**:

- Почиње са Quick sort-ом
- Ако примети да сортирање иде ка лошем случају ($O(n^2)$), прећи на **Heap sort**
- За мале делове низа користи **Insertion sort**, јер је он ефикасан на малим количинама података.

4 Програмирање

Графички приказ је направљен у програмском језику C#. Апликација се састоји из једне форме и две класе.

4.1 Форма

Форма садржи један панел (на ком се све исцртава) и неколико лабела (које служе као дугмићи за одабир боје, брзине...). Када се покрене програм најпре се иницијализује **класа сорт**. Исцртава се наслов, исписују лабеле и позивају се методе **postavi()** и **nacrtaj()** које припремају и цртају стубиће. Иницијализују се променљиве које за сваки од алгоритама прате да ли су покренути и да ли су паузирани. Форма садржи доста обезбеђивача догађаја (*event handlers*):

- **private void panel1_MouseClick(...)** – Користи се уколико се кликне на неки од правоугаоника да би се програм променио у режим једног алгоритма
- **private async void quick_sort_MouseClick(...)** – Кликом на лабелу (али кориснику изгледа као дугме) се алгоритам покреће/паузира/наставља (апсолутно исто и за остале алгоритме)

```
private async void quick_sort_MouseClick(object sender, MouseEventArgs e)
{
    if (pokrenut4 == 0 && dalijeutoku2 == 0 && dalijeutoku3 == 0 && dalijeutoku1 == 0) //pokrenuti sortiranje
    {
        dalijeutoku4 = 1;
        pokrenut4 = 1;
        s.Pauzirano4 = false;
        await s.sortiraj_quick(0, brel - 1,cts.Token);
        pokrenut4 = 0;
        dalijeutoku4 = 0;
    }
    else if (dalijeutoku4 == 1) //pauzirati
    {
        s.Pauzirano4 = true;
        dalijeutoku4 = 0;
    }
    else if (pokrenut4 == 1 && dalijeutoku2 == 0 && dalijeutoku3 == 0 && dalijeutoku1 == 0)//nastaviti sortiranje
    {
        s.Pauzirano4 = false;
        dalijeutoku4 = 1;
    }
}
```

Слика 12 - Код за private async void quick_sort_MouseClick(...)

- **private void resetuj_MouseClick(...)** – Кликом на дугме **Ресетуј** се заустављају алгоритми уколико је неки био започет и поставља се нови низ елемената
- **private void promeni_boju_MouseClick(...)** – Кликом на дугме **Промени боју** се све исцртава у другој боји

4.2 Класа Сорт

Класа сорт као атрибуте има

- **график** и **боје** за цртање
- **број елемената** које је корисник изабрао
- променљиву **стање** која узима вредности од **0** до **4** у зависности од тога да ли је тренутно приказан **главни мени(0)** или **режим једног алгоритма(1,2,3,4)**
- **5 низова класе елемент** (један помоћни и по један за сваки од алгоритама сортирања)
- променљиву **брзина** која представља број милисекунди на колико се програм заустави (да би се видели постепени кораци алгоритма)
- индикаторе за сваки од сортова да ли је у тренутку **паузиран**

Методе које садржи:

- **конструктор – public sort(…)** - Поставља број елемената и конфигурише графичке параметре (висина, ширина, брзина) у зависности од унете величине низа (**n**).
- **public void postavi(…)** - Генерише насумични низ без понављања. Сваки члан свих низова добија своју позицију на екрану, као и вредност.
- **public void promeni_boju(…) –** Поставља нову тему – нови сет боја
- **public void promeni_velicinu(…) –** Поставља ново стање, на основу ког мења графичке параметре
- **public void nacrtaj(…) –** На основу стања исцртава или главни мени или режим једног алгоритма
- **public async Task sortiraj_bubble(…) –** Класичан алгоритам за сортирање са додатом графиком (апсолутно исто и за остале алгоритме)

```
public async Task sortiraj_bubble(CancellationToken token)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = n - 1; j > i; j--)
        {
            bs1 = i;
            bs2 = j;
            niz_bubble[j].nacrtaj(g, c3, c2,c, d, brojevi, n);
            niz_bubble[j - 1].nacrtaj(g, c3, c2, c, d, brojevi, n);
            await Task.Delay(brzina);
            while (pauzirano3)
                await Task.Delay(100);
            if (token.IsCancellationRequested) return;
            if (niz_bubble[j].Vrednost < niz_bubble[j - 1].Vrednost)
                niz_bubble[j].razmeni(g, niz_bubble[j - 1], c2, c, d, brojevi, n);
            niz_bubble[j - 1].nacrtaj(g, c1, c2, c, d, brojevi, n);
            niz_bubble[j].nacrtaj(g, c1, c2, c, d, brojevi, n);
        }
    }
    bs1 = -1;
    bs2 = -1;
}
```

Слика 13 - метода public async Task sortiraj_bubble(…)

Методе за алогритме за сортирање, поред класичног кода, садрже и графичке елементе. Сваки пут кад се уђе у петљу два елемента која се тренутно обрађују/упоређују исцртавају се у другој боји како би се истакли. Онда се помоћу контроле **await Task.Delay(brzina)** програм зауставља на неко време (у зависности од броја елемената и брзине коју је корисник одабрао). Након тога се проверава да ли је потребно да се направи прекид сортирања – уколико је корисник кликнуо паузу, ресетовање, промену броја елемената… Упоређују се вредности ова два елемента низа, и уколико је потребно врши се њихова замена тако што се позива метода из **класе елемент**. На крају се само исцртавају елементи након замене. Уместо контроле **await Task.Delay(brzina)** на почетку сам користила **Thread.Sleep(brzina)**. Једноставна је за коришћење али је проблем што се цео програм *успава* то јест док се тај алгоритам не заврши није могућ његов прекид или истовремено извршавање нечег другог.

4.3 Класа Елемент

Класа елемент садржи атрибуте за **вредност**, као и координате **x** и **y**. Од метода, осим својстава за малопре споменуте атрибуте има:

- **public void nacrtaj(…) –** Црта елемент у одређеној боји (то може бити и боја позадине уколико је потребно да се елемент *обрише* са тог места ради замене са неким другим елементом) и исписује вредности уколико је то потребно
- **public void razmeni(…)** – Размењује два елемента (њихове вредности и координате)

5 Закључак

Радећи на овој апликацији стекла сам дубље разумевање алгоритама за сортирање, не само кроз учење њихових особина, већ и кроз практичну примену и визуелизацију. Иако сам већ познавала основне принципе алгоритама као што су **Selection**, **Insertion**, **Bubble** и **Quick sort**, тек када сам их *оживела* у апликацији и видела како корак по корак мењају низ, схватила сам њихове разлике у брзини, стабилности и понашању у различитим ситуацијама.

Осим алгоритама, овај пројекат ми је помогао да се боље снађем у раду са формама, контролама, асинхроним извршавањем и генерално у **C#** окружењу. Посебно ми је било занимљиво радити на делу који се односи на интеракцију са корисником — избор броја елемената, брзине, боје и појединачног алгоритма.

Иако је било изазова, највећа вредност овог рада је то што сам кроз практичан пројекат повезала различите области знања — од теорије алгоритама до графичког програмирања. Верујем да ми је овај пројекат дао добро полазиште за даље усавршавање и рад на још сложенијим задацима у будућности.

Литература

[1] C/C++ збирка задатака, Милан Чабаркапа и Станка Матковић, Круг Београд

[2] http://www.webnstudy.com/tema.php?id=insertion-sort, последњи пут приступљено 26.05.2025.

[3] https://sr.wikipedia.org/sr-ec/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%B0%D1%9A%D0%B0, последњи пут приступљено 26.05.2025.

[4] https://skolakoda.github.io/redjanje-umetanjem, последњи пут приступљено 26.05.2025.

[5] http://www.webnstudy.com/tema.php?id=quicksort, последњи пут приступљено 26.05.2025.