TEHNIČKA ŠKOLA RUĐERA BOŠKOVIĆA

GETALDIĆEVA 4, ZAGREB

ZAVRŠNI STRUČNI RAD:

Algoritmi za sortiranje s primjerima

Mentorica: Sanja Vehabović-Hadžić UČENIK: Matija Aleksić

RAZRED: 4.A

Zagreb, travanj 2021.

Sadržaj

Sadržaj

[1. Uvod 4](#_Toc70401871)

[2. Ukratko o programskom jeziku c# 5](#_Toc70401872)

[3. Ukratko o Microsoft Visual Studio Express 2012 5](#_Toc70401873)

[4. Izgled (dizajn) forms aplikacije 5](#_Toc70401874)

[5. Objašnjenje koda 8](#_Toc70401875)

[5.1 Form1 8](#_Toc70401876)

[5.2 Kod koji se ponavlja 9](#_Toc70401877)

[5.3 Bubble Sort 10](#_Toc70401878)

[5.3.1 Kod 10](#_Toc70401879)

[5.3.2 Objašnjenje 11](#_Toc70401880)

[5.4 Insertion sort 11](#_Toc70401881)

[5.4.1 Kod 11](#_Toc70401882)

[5.4.2 Objašnjenje 12](#_Toc70401883)

[5.5 Shell sort 12](#_Toc70401884)

[5.5.1 Kod 12](#_Toc70401885)

[5.5.2 Objašnjenje 13](#_Toc70401886)

[5.6 Radix sort 16](#_Toc70401887)

[5.6.1 Kod 16](#_Toc70401888)

[5.6.2 Objašnjenje 16](#_Toc70401889)

[5.7 Quick sort 19](#_Toc70401890)

[5.7.1 Kod 19](#_Toc70401891)

[5.7.2 Objašnjenje 19](#_Toc70401892)

[5.8 Counting sort 20](#_Toc70401893)

[5.8.1 Kod 20](#_Toc70401894)

[5.8.2 Objašnjene 20](#_Toc70401895)

[5.9 Selection sort 20](#_Toc70401896)

[5.9.1 Kod 20](#_Toc70401897)

[5.9.2 Objašnjenje 20](#_Toc70401898)

[Izvori 21](#_Toc70401899)

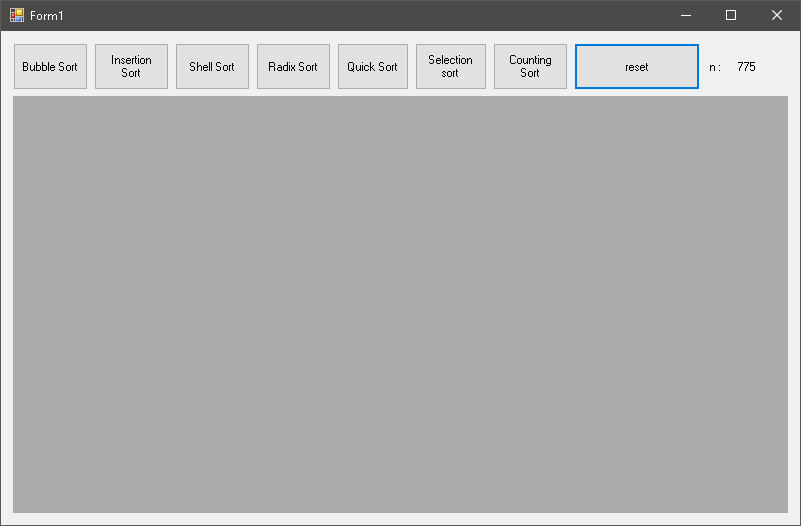
# Uvod

Gluposti

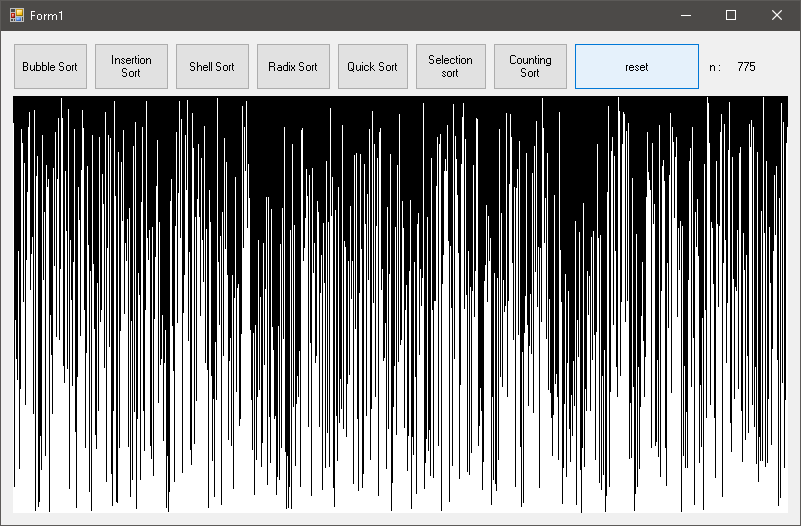
# Ukratko o programskom jeziku c#

# Ukratko o Microsoft Visual Studio Express 2012

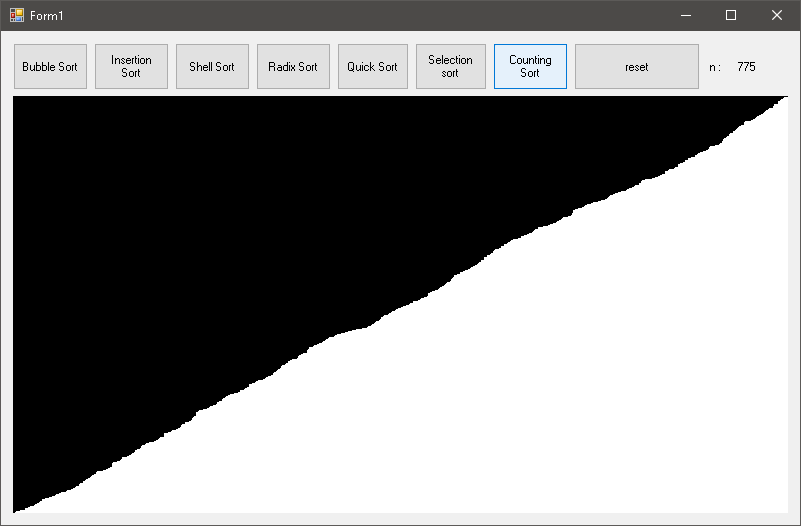
# Izgled (dizajn) forms aplikacije



4.1 Slika aplikacije na pokretanju



4.2 Slika aplikacije sa ne sortiranim grafom



4.3 Slika aplikacije sa sortiranim grafom

# Objašnjenje koda

## Form1

namespace AlgoZaSort

{

public partial class Form1 : Form

{

//inicalizacija arraya kojeg

//ćemo koristiti za sortiranje i

//graphics elementa za crtanje

int[] RandArray;

Graphics s;

//this.Resize koristimo da bi smo otkrili

//promjenu oblika prozora koja će pozivati

//funkciju Form1\_Resize;

public Form1()

{

InitializeComponent();

this.Resize += Form1\_Resize;

}

// funciju koristimo da refreshamo label koji

//prikazuje broj n i graf, tako da nam se graf

//odmah prilagodi veličini prozora (panela)

void Form1\_Resize(object sender, EventArgs e)

{

label1.Text = Convert.ToString(panel1.Width);

crtanje\_graf();

}

// ispis labela n čim se prozor pokrene

private void Form1\_Load(object sender, EventArgs e)

{

label1.Text = Convert.ToString(panel1.Width);

}

//pritiskom na gumb ponovo crtamo graf

//u slučaju da ne želimo mijenjat n

private void butReset\_Click(object sender, EventArgs e)

{

crtanje\_graf();

}

//funkcija koju koristimo za crtanje novog(svježeg)

//grafa

private void crtanje\_graf()

{

//refresh n

label1.Text = Convert.ToString(panel1.Width);

//stvaranje graphics objekta za panel1

s = panel1.CreateGraphics();

//uzimamo n koji ćemo koristiti za veličinu arraya

int n = panel1.Width;

//postavljamo najveću vrijednost za crtanje, visinu panela

int MaxVrijednost = panel1.Height;

//postavljanje dužine arraya

RandArray = new int[n];

//popunjavanje panel1 sa crnom bojom

s.FillRectangle(new System.Drawing.SolidBrush(System.Drawing.Color.Black), 0, 0, n, MaxVrijednost);

//punjenje arraya sa nasumičnim brojevima od 0

//do MaxVrijednost(visina) panela

Random rand = new Random();

for (int i = 0; i < n; i++)

{

RandArray[i] = rand.Next(0, MaxVrijednost);

}

//kada je array napunjen random brojevima crtamo ga

for (int i = 0; i < n; i++)

{

s.FillRectangle(new System.Drawing.SolidBrush(System.Drawing.Color.White), i, MaxVrijednost - RandArray[i], 1, MaxVrijednost);

}

}

//prilikom stiska gumba upotpunjujemo Vrijednosti

//aktualnim podatcima i pozivamo prikladajući sort algoritam

private void butStart\_Click(object sender, EventArgs e)

{

Vrijednost bs = new BubbleSort();

bs.Vrijednosti(RandArray, s, panel1.Height);

}

.

.

.

}

}

## Kod koji se ponavlja

Ovaj dio koda se ponavlja u svakom class-u algoritama za sortiranje i uvijek ima istu svrhu. Prikazan kao primjer.

//intamo kopije potrebnih vrijednosti

private static int[] RandArray;

private static Graphics s;

private static int MaxVrijednost;

static Brush WhiteBrush = new System.Drawing.SolidBrush(System.Drawing.Color.White);

static Brush BlackBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Black);

// spremanje nasljeđenih vrijednosti u kopije i pozivanje određenog sorta

public void Vrijednosti(int[] RandArrayIN, Graphics sIN, int MaxVrijdnostIN)

{

RandArray = RandArrayIN;

s = sIN;

MaxVrijednost = MaxVrijdnostIN;

PRIMJER\_Sort(RandArray);

}

PRIMJER\_Sort(int[] data){

...

swap(i, i + 1);

...

}

//funkcija zamjenjuje vrijednosti na određenim pozicijama arraya

//i pozziva funkciju za ponovo crtanje nekon zamjene

private static void swap(int i, int p)

{

int t = RandArray[i];

RandArray[i] = RandArray[p];

RandArray[p] = t;

DrawBar(p, RandArray[p]);

DrawBar(i, RandArray[i]);

}

private static void DrawBar(int i, int p)

{

//na poziciji i crta crni stupić do MaxVrijednosti tj. briše stari

s.FillRectangle(BlackBrush, i , 0, 1, MaxVrijednost);

//na poziciji i crta novi bijeli stupić

s.FillRectangle(WhiteBrush, i , MaxVrijednost - p, 1 , MaxVrijednost);

}

}

## Bubble Sort

### Kod

public static void Bubble\_Sort(int[] data)

{

int i, j;

int N = data.Length;

for (j = N - 1; j > 0; j--)

{

for (i = 0; i < j; i++)

{

if (data[i] > data[i + 1])

swap(i, i + 1);

}

}

}

### Objašnjenje

Bubble sort je najjednostavniji I samim time najsporiji algoritam za sortiranje. Njegova upotreba u pravom svijetu nije česta zbog njegove brzine, ali on je početni sort koji se daje kao primjer učenicima zbog njegove jednostavnosti za lakoću razumijevanja i uvod u algoritme za sortiranje.

Bubble sort radi tako da prolazi kroz cijeli array i zamjenjuje vrijednosti kako smo mi odredili (if (data[i] > data[i + 1])) uzlazno ili silazno. Svakim prolaskom kroz array algoritam uspoređuje vrijednosti na svakom koraku, uzima najveću vrijednost i pomjera ju (u našem slučaju) prema desno. Kako bi smo optimizirali ovaj sort mi ne moramo uvijek uspoređivati od početka do kraja arraya nego uspoređujemo od početka arraya do početka sortiranog dijela arraya. Tako ubrzavamo algoritam jer ne moramo uspoređivati i sortirani dio arraya. Bubble sort nije prikladan za sortiranja većeg broja vrijednosti.

## Insertion sort

### Kod

public static void IntArrayInsertionSort(int[] data)

{

int i, j;

int N = data.Length;

for (j = 1; j < N; j++)

{

for (i = j; i > 0 && data[i] < data[i - 1]; i--)

{

swap(i, i - 1);

}

}

}

### Objašnjenje

Insertion sort se često radi objašnjenja uspoređuje sa načinom kako mi sortiramo karte u rukama.

Insertion sort je algoritam dijeli array na dva dijela : sortirani i ne sortirani dio. Algoritam sortira tako da usporedi nadolazeću vrijednost (iz ne sortiranog dijela) sa svim prijašnjim vrijednostima koje su došle prije njega. Uspoređuje vrijednosti na isti princip kao bubble sort, umjesto da najveću vrijednost uzme i sortira ju na dalje, on prvu vrijednost iz ne sortiranog dijela pomjera do svog mjesta u sortiranom dijelu. Algoritam na taj način radi puno nepotrebnih usporedbi i pomjeranja vrijednosti samim time smanjuje svoju efikasnost. Isto kao bubble sort nije prikladan za sortiranje većeg broja vrijednosti.

## Shell sort

### Kod

#### Shell Sort

public static void IntArrayShellSort(int[] data, int[] intervals)

{

int i, j, k, m;

int N = data.Length;

for (k = intervals.Length - 1; k >= 0; k--)

{

int interval = intervals[k];

for (m = 0; m < interval; m++)

{

for (j = m + interval; j < N; j += interval)

{

for (i = j; i >= interval && data[i] < data[i - interval]; i -= interval)

{

swap( i, i - interval);

}

}

}

}

}

#### Pomoćni kod za stvaranje intervala

static int[] GenerateIntervals(int n)

{

int t = Math.Max(1, (int)Math.Log(n, 3) - 1);

int[] intervals = new int[t];

intervals[0] = 1;

for (int i = 1; i < t; i++)

intervals[i] = 3 \* intervals[i - 1] + 1;

return intervals;

}

### Objašnjenje

Shell sort je prvi kompleksniji algoritam za sortiranje. Algoritam se bazira na stvaranju intervala u arrayu koji će se sortirati sve dok interval nije jednak nula na principu bubble sorta. Što je veći broj ulaznih vrijednosti to trebamo veći početni interval. Donald Knuth je predložio nekoliko metoda dobivanja intervala, koristiti ću sljedeću :

t=⌊log3n⌋−1

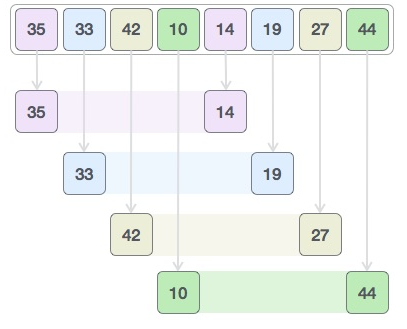
Ovom formulom generirat ćemo potrebne intervale. Prikaz implementacije formule možemo vidjeti u kodu 5.5.1.2.

Shell sort možemo vizualizirati za lakše objašnjavanje.



Slika 5.5.2.1 – početno stanje arraya

Slika 5.5.2.1 prikazuje početno stanje arraya za primjer. Prvi interval koji ćemo koristiti u primjeru je tri, array je mali i veći interval nije optimalan. Uspoređujemo sve brojeve u arrayu sa intervalom 3 :



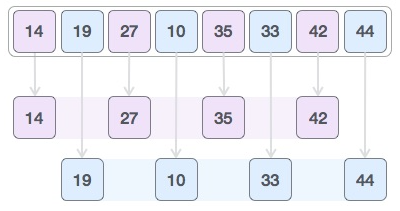
Slika 5.5.2.2

Sve brojeve usporedimo i zamijenimo im mjesta po potrebi. Nakon prve zamjene array izgleda ovako :



Slika 5.5.2.3

Idući interval je duplo manji od prošlog. 3/2 = 1.5 . U kodu 5.5.1.2 t je definiran kao integer I zato umjesto nemogućeg intervala 1.5 dobivamo interval 1. Array uspoređujemo sa intervalom 1 :



Slika 5.5.2.4

Nakon potrebnih zamjena vrijednosti dobivamo iduće stanje arraya :



Slika 5.5.2.5

Idući interval je integer od 1/2=0. Više ne možemo koristiti intervale i prebacujemo se na insertion sort koji će brzo sortirati već napola sortiran array i završni array izgleda ovako :



Slika 5.5.2.6

## Radix sort

### Kod

static void RadixSortAlgoritam(int[] data)

{

int i, j;

int[] tmp = new int[data.Length];

for (int shift = 31; shift > -1; --shift)

{

j = 0;

for (i = 0; i < data.Length; ++i)

{

bool move = (data[i] << shift) >= 0;

if (shift == 0 ? !move : move)

{

data[i - j] = data[i];

DrawBar(i-j,data[i-j]);

}

else

{

tmp[j++] = data[i];

}

}

Array.Copy(tmp, 0, data, data.Length - j, j);

}

}

### Objašnjenje

Radix sort je posebna vrsta algoritma za sortiranje zato što on ne uspoređuje vrijednosti direktno. On se može koristiti za sortiranje brojeva, slova, znakova pa čak i punch kartica. U ovom slučaju sortirati će brojeve. Radix sort radi po principu kantica u koje se spremaju redom vrijednosti arraya prvo po vrijednosti jedinica, pa po vrijednosti desetica itd. Kada prođemo kroz sve znamenke najvećeg broja array će biti sortiran.

Postupak :

Imamo brojeve : 13,124,52,773 i 536

I kantice : od 0 do 9(za svaku znamenku broja)

Prvi korak je da brojeve poredamo u odgovarajuće kantice prema njihovoj zadnjoj znamenki pazeći da ih upisujemo redom

0

1

2 52

3 13 773

4 124

5

6 536

7

8

9

Dobivamo idući raspored : 52,13,773,124,536 i ponavljamo postupak ali sad na znamenkama desetica.

0

1 13

2 124

3 536

4

5 52

6

7 773

8

9

Dobivamo raspored : 13,124,536,52,773. Sad bi mislili da ne možemo dalje sortirati zato što neke vrijednosti nemaju znamenke stotica, ali imaju, nula.

0 13,52

1 124

2

3

4

5 536

6

7 773

8

9

I na kraju dobijemo sortiran array bez direktne usporedbe varijabla : 13,52,124,536,773.

## Quick sort

### Kod

public static void IntArrayQuickSort(int[] data, int l, int r)

{

int i, j;

int x;

i = l;

j = r;

x = data[(l + r) / 2];

while (true)

{

while (data[i] < x)

i++;

while (x < data[j])

j--;

if (i <= j)

{

swap(i, j);

i++;

j--;

}

if (i > j)

break;

}

if (l < j)

IntArrayQuickSort(data, l, j);

if (i < r)

IntArrayQuickSort(data, i, r);

}

### Objašnjenje

Quick sort je jedan od najpoznatijih algoritama za sortiranja današnjice. Quick sort je rekurzivan tj. Poziva sam sebe kod sortiranja. Program dijeli na dva dijela preko pivot-a. Za prvi point najčešće uzimamo vrijednost na sredini arraya.

Primjer arraya :

9,3,7,1,8,6,4,2,5,10,11

Prvi pivot point uzimamo na sredini arraya :

9,3,7,1,8,6,4,2,5,10,11

Zatim sve brojeve manje od pivota pomaknemo lijevo od njega a veće desno :3,1,4,2,5,6,9,7,8,10,11

Sad uzimamo novi pivot na sredini brojeva manjih od pivota(rekurzija), na primjeru to je broj 4 :

3,1,4,2,5,6,9,7,8,10,11

i ponovimo prijašnji korak. Dobivamo :

1,2,3,4,5,6,9,7,8,10,11

Istu stvar ponavljamo na desno strani :

1,2,3,4,5,6,9,7,8,10,11

1,2,3,4,5,6,7,8,9,10,11

Primjećujemo da nam na desnoj strani od orginalnog pivota trebala je jedna manja rekurzija od lijeve strane i isto tako brojeve sa lijeve strane nikad ne uspoređujemo sa brojevima na desnoj. Manje usporedba znači brži i efikasniji sort.Zbog toga quick sort ima i upotreba u pravom svijetu.

## Counting sort

### Kod

### Objašnjene

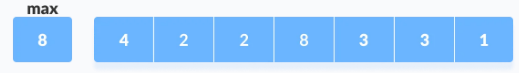
Uvod

Početni array :



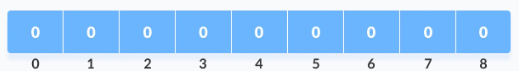
Slika 5.8.2.1 Početni array

Prvo iz arraya pronalazimo maximalnu vrijednost.



Slika 5.8.2.2

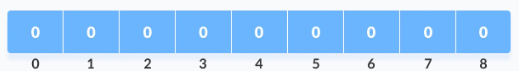
Inicijaliziramo novi array dužine max+1¸i upotpunimo ga nulama. Njega ćemo koristiti za brojanje koliko puta se neki broj ponavlja. Od tuda dobivamo i ime sorta.



Slika 5.8.2.3

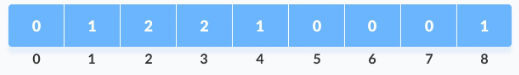
U novi array upisujemo koliko puta se znamenka pojavljuje u orginalnom arrayu na prikladno mjesto u novom arrayu





Slika 5.8.2.4

Završno stanje novog arraya :



Slika 5.8.2.5

## Selection sort

### Kod

### Objašnjenje

# Zaključak

# Izvori

Shell sort vizualizacija slike : <https://www.tutorialspoint.com/data_structures_algorithms/shell_sort_algorithm.html>

Radix Sort uvod povijest : <https://en.wikipedia.org/wiki/Radix_sort>

Counting sort objašnjenje : <https://www.programiz.com/dsa/counting-sort>