*Colegiul Național „Ion C. Brătianu”*

*Pitești, Argeș*

**Metode de sortare. Analiză comparativă**

*Lucrare pentru susținerea atestatului profesional la disciplina informatică*

Elev: Graure Vlad-Ștefan

Profesor îndrumător: Enache Nicoleta

Clasa a XII-a C

Specializarea matematică-informatică, informatică intensiv

Sesiunea MAI 2022

Cuprins

[Introducere 3](#_Toc100609813)

[Descrierea limbajului 4](#_Toc100609814)

[Originea numelui 5](#_Toc100609815)

[Istorie 5](#_Toc100609816)

[Operatori 6](#_Toc100609817)

[Sintaxa 7](#_Toc100609818)

[Metode și funcții 8](#_Toc100609819)

[Biblioteci 9](#_Toc100609820)

[Constructori 11](#_Toc100609821)

[Interfețe 11](#_Toc100609822)

[Descrierea proiectului 13](#_Toc100609823)

[Funcționalitatea butoanelor 14](#_Toc100609824)

[Butoanele din partea superioară 14](#_Toc100609825)

[Butoanele din partea mediană 15](#_Toc100609826)

[Apelarea funcției constructor 18](#_Toc100609827)

[Interfața 19](#_Toc100609828)

[Algoritmi de sortare 19](#_Toc100609829)

[1. Metoda bulelor 20](#_Toc100609830)

[2. Sortarea prin numărare 20](#_Toc100609831)

[3. Sortarea prin interschimbare 21](#_Toc100609832)

[4. Heap Sort 22](#_Toc100609833)

[5. Sortarea prin inserție 23](#_Toc100609834)

[6. Sortarea prin interclasare 23](#_Toc100609835)

[7. Quick Sort 24](#_Toc100609836)

[8. Sortarea prin selecție 25](#_Toc100609837)

[9. Shell Sort 25](#_Toc100609838)

[Bibliografie 27](#_Toc100609839)

# Introducere

Algoritmii de sortare reprezintă un tip de algoritmi care este utilizat pentru a poziționa elementele unui set conform unei secvențe stabilite printr-o relație de ordine, astfel încât fiecare element sa fie mai mic sau mai mare decât cel care urmează. Fiecare tip de algoritm se sortare are avantajele si dezavantajele sale, vizând complexitatea, viteza de realizare a sortării pentru seturi de elemente de mărime diferită și dimensiunea pe disc.

Algoritmii de sortare sunt unul dintre cei mai de baza algoritmi, astfel importanța lor este crucială. Majoritatea aplicațiilor conțin cel puțin o formă de sortare: fie că este vorba despre modul în care realizează un set de operații sau de modul în care afișează rezultatele.

Am ales această temă deoarece consider că o bună stăpânire a acestui tip de algoritmi constituie un mare avantaj pentru oricine dorește să urmeze programarea în continuare. Vizualizarea modului în care aceștia operează, alături de cunoașterea principiului pe care aceștia operează constituie, în opinia mea, una dintre cele mai eficiente modalități de înțelegere logică a algoritmilor.

Aplicația este programată în C# și folosește o interfață Windows Forms. Ea conține 9 tipuri diferite de algoritmi de sortare și are rolul de a prezenta o vizualizare a modului în care aceștia operează prin arătarea interschimbărilor pe care algoritmul le realizează în timpul unei sortări.

Așadar, aplicația are rol didactic, putând fi utilizată atât de profesori pentru predare, cât și de elevi sau orice persoană care dorește o mai bună înțelegere a acestor algoritmi.

# Descrierea limbajului

C# este un limbaj de programare cu scop general. C# cuprinde tastarea statică, tastarea puternică, imperativ, declarativ, funcțional, generic, orientat pe obiecte (bazate pe clase) și discipline de programare orientate spre componente.

C# a fost proiectat de Anders Hejlsberg de la Microsoft în 2000 și mai târziu a fost aprobat ca standard internațional de Ecma (ECMA-334) în 2002 și ISO (ISO/IEC 23270) în 2003. Microsoft a introdus C# împreună cu .NET Framework și Visual Studio. La acea vreme, Microsoft nu avea produse open-source. Patru ani mai târziu, în 2004, a început un proiect gratuit și open-source numit Mono, oferind un compilator cross-platform și un mediu runtime pentru limbajul de programare C#. Un deceniu mai târziu, Microsoft a lansat Visual Studio Code (editor de cod), Roslyn (compilator) și platforma unificată .NET, toate acestea suportând C# și sunt gratuite, open-source și cross-platform.

Începând cu 2021, cea mai recentă versiune a limbajului este C# 10.0, care a fost lansată în 2021 în .NET 6.0.

Standardul Ecma enumeră următoarele obiective de proiectare pentru C#:

* Limbajul se dorește a fi un limbaj de programare simplu, modern, de uz general, orientat pe obiecte.
* Limbajul și implementările acestora ar trebui să ofere suport pentru principiile ingineriei software, cum ar fi verificarea puternică a tipului, verificarea limitelor matricei, detectarea încercărilor de utilizare a variabilelor neinițializate. Robustețea software-ului, durabilitatea și productivitatea programatorului sunt importante.
* Limbajul este destinat utilizării în dezvoltarea de componente software adecvate pentru implementarea în medii distribuite.
* Portabilitatea este foarte importantă pentru codul sursă și programatori, în special pentru cei deja familiarizați cu C și C ++.
* Sprijinul pentru internaționalizare este foarte important.
* C# se dorește a fi potrivit pentru scrierea aplicațiilor atât pentru sistemele găzduite, cât și pentru cele încorporate, variind de la cele foarte mari care utilizează sisteme de operare sofisticate, până la cele foarte mici care au funcții dedicate.
* Deși aplicațiile C# sunt destinate să fie economice în ceea ce privește cerințele de memorie și putere de procesare, limbajul nu a fost destinat să concureze direct pe performanță și dimensiune cu C sau Assembly.

## Originea numelui

Microsoft a folosit pentru prima dată numele C# în 1988 pentru o variantă a limbajului C concepută pentru compilare incrementală. Acel proiect nu fost niciodată finalizat, dar numele a rămas.

Numele „C Sharp” a fost inspirat de notația muzicală prin care un diez indică faptul că nota scrisă ar trebui făcută cu un semiton mai înalt în înălțime. Acesta este similar cu numele limbajului C++, unde „++” indică faptul că o variabilă ar trebui să fie incrementată cu 1 după ce a fost evaluată. Simbolul ascuțit seamănă, de asemenea, cu o legătură de patru simboluri „+” (într-o grilă două câte două), implicând că limbajul este un increment de C++.

Sufixul „sharp” a fost folosit de o serie de alte limbaje .NET care sunt variante ale limbilor existente, inclusiv J# (un limbaj .NET conceput și de Microsoft, care este derivat din Java 1.1), A# (de la Ada) și limbajul de programare funcțional F#. Implementarea inițială a Eiffel pentru .NET a fost numită Eiffel#, un nume retras deoarece limbajul Eiffel complet este acum acceptat. Sufixul a fost folosit și pentru biblioteci, cum ar fi Gtk# (un wrapper .NET pentru GTK și alte biblioteci GNOME) și Cocoa# (un wrapper pentru Cocoa).

## Istorie

În timpul dezvoltării .NET Framework, bibliotecile de clasă au fost scrise inițial folosind un sistem de compilare de cod gestionat numit „Simple Managed C” (SMC). În ianuarie 1999, Anders Hejlsberg a format o echipă pentru a construi un nou limbaj la acea vreme, numit Cool, care înseamnă „C-like Object Oriented Language”. Microsoft a luat în considerare păstrarea numelui „Cool” ca denumire finală a limbii, dar a ales să nu facă acest lucru din motive legate de marca comercială. În momentul în care proiectul .NET a fost anunțat public la Conferința dezvoltatorilor profesionali din iulie 2000, limbajul fusese redenumit C#, iar bibliotecile de clasă și runtime-ul ASP.NET fuseseră portate în C#.

## Operatori

1. Adunare (și plus unar) și scădere (și minus unar): +, -.

c = a + b;

c = a - b;

1. Înmulțire: \*.

c = a \* b;

1. Împărțire Modulo: %.

c = a % b; // calculează restul împărțirii lui a la b

1. Incrementare și decrementare: ++, --.

c = ++a; // îl crește pe a cu o unitate

c = --a; // îl scade pe a cu o unitate

1. Egal si diferit de: ==, !=.

if (bgw == null)

if (bgw != null)

1. Mai mic și mai mic sau egal: <, <=.

if (a < b)

if (a <= b)

1. Mai mare și mai mare sau egal: >, >=.

if (a > b)

if (a >= b)

1. Operatori logici (negație, și, sau): !, &&, ||.

if (!a)

if (a && b)

if (a || b)

1. De atribuire: =, +=, -=, /=, \*=.

c += a; // echivalent cu c = c + a;

1. Operatorul condițional:  expresie1? expresie2: expresie3.

Returnează expresie 2 dacă expresie 1 este adevărată sau expresie 3 dacă este falsă.

## Sintaxa

Sintaxa de bază a limbajului C # este similară cu cea a altor limbaje în stil C, cum ar fi C, C++ și Java, în special:

* Punctul și virgula este folosit pentru a indica sfârșitul unei declarații sau al unei comenzi.

int spacing = 0;

* Parantezele sunt folosite pentru a grupa declarațiile. Instrucțiunile sunt grupate în mod obișnuit în metode (funcții), metodele în clase și clasele în spații de nume.

if (bgw != null && !bgw.IsBusy || bgw == null)

* Variabilele sunt atribuite folosind un semn egal, dar comparate folosind două semne egal consecutive.

spacing = 0;

*spre deosebire de*

if (bgw != null && !bgw.IsBusy || bgw == null)

* Parantezele pătrate sunt folosite cu tablouri, atât pentru a le declara, cât și pentru a obține o valoare la un index dat într-unul dintre ele.

int[] array, start\_pos;

*sau*

array[i]

* C # acceptă declarații de variabile implicit tipizate prin cuvântul cheie var și matrice tip implicit cu cuvântul cheie new [] urmat de un inițializator de colecție.

array = new int[NumEntries];

* C # acceptă un tip de date booleane, bool. Declarațiile care iau condiții, cum ar fi while și if, necesită o expresie de tip care implementează operatorii true și false, cum ar fi tipul boolean. În timp ce C ++ are, de asemenea, un tip boolean, poate fi convertit liber în și din numere întregi, iar expresii precum if (a) necesită doar ca a să fie convertibil în bool, permițând a să fie un int sau un pointer. C # nu permite această abordare „întreg care înseamnă adevărat sau fals”, pe motiv că forțarea programatorilor să folosească expresii care returnează exact bool poate preveni anumite tipuri de greșeli de programare, cum ar fi if (a = b) (folosirea atribuirii = în loc de egalitate = = ).

bool Paused = false;

C# este mai „prietenos” la tastare decât C ++. Singurele conversii implicite sunt cele care sunt considerate sigure, cum ar fi extinderea numerelor întregi. Acest lucru este aplicat în timpul compilării, în timpul JIT și, în unele cazuri, în timpul execuției. Nu au loc conversii implicite între booleeni și numere întregi, nici între membrii enumerare și numere întregi (cu excepția literalului 0, care poate fi convertit implicit în orice tip enumerat). Orice conversie definită de utilizator trebuie să fie marcată explicit ca explicită sau implicită, spre deosebire de constructorii de copiere C++ și operatorii de conversie.

dly= Convert.ToInt32(numericUpDown1.Value);

C # are suport explicit pentru covarianță și contravarianță în tipurile generice, spre deosebire de C ++ care are un anumit grad de suport pentru contravarianță pur și simplu prin semantica tipurilor returnate pe metodele virtuale.

Membrii de enumerare sunt plasați în domeniul lor propriu.

Limbajul C # nu permite variabile sau funcții globale. Toate metodele și membrii trebuie să fie declarați în cadrul claselor. Membrii statici ai claselor publice pot înlocui variabilele și funcțiile globale.

Variabilele locale nu pot umbri variabilele blocului care le încadrează, spre deosebire de C și C++.

## Metode și funcții

O metodă în C # este un membru al unei clase care poate fi invocată ca funcție (o secvență de instrucțiuni), mai degrabă decât simpla capacitate de păstrare a valorii unei proprietăți de clasă. Ca și în alte limbaje similare din punct de vedere sintactic, cum ar fi C++ și ANSI C, semnătura unei metode este o declarație care cuprinde în ordine: orice cuvinte cheie opționale de accesibilitate (cum ar fi private), specificarea explicită a tipului său de returnare (cum ar fi int, sau cuvântul cheie void dacă nu se returnează nicio valoare), numele metodei și, în final, o secvență între paranteze de specificații ale parametrilor separate prin virgulă, fiecare constând dintr-un tip de parametru, numele său formal și, opțional, o valoare implicită care trebuie utilizată oricând nu este furnizat niciunul. Anumite tipuri specifice de metode, cum ar fi cele care pur și simplu obțin sau stabilesc o proprietate de clasă prin valoarea returnată sau atribuire, nu necesită o semnătură completă, dar în cazul general, definiția unei clase include declarația completă a semnăturii metodelor sale.

public bool IsSorted()

{

for (int i = 0; i < array.Count() - 1; i++)

{

if (array[i] > array[i + 1]) return false;

}

return true;

}

La fel ca C ++ și spre deosebire de Java, programatorii C# trebuie să folosească domeniul de aplicare al modificatorului de cuvinte cheie virtuale pentru a permite metodelor să fie suprascrise de subclase.

Metodele de extensie în C# permit programatorilor să folosească metode statice ca și cum ar fi metode din tabelul de metode ale unei clase, permițând programatorilor să adauge metode la un obiect despre care cred că ar trebui să existe pe acel obiect și derivatele sale.

Tipul dinamic permite legarea metodei în timpul rulării, permițând apeluri de metodă asemănătoare JavaScript și compoziția obiectelor în timpul rulării.

C# are suport pentru indicatori de funcții cu tastare puternică prin cuvântul cheie delegat. La fel ca semnalul și slot-ul pseudo-C++ al cadrului Qt, C# are o semantică care înconjoară în mod specific evenimentele de stil de publicare-abonare, deși C # folosește delegați pentru a face acest lucru.

C# oferă apeluri de metodă sincronizate asemănătoare Java, prin atributul [MethodImpl (MethodImplOptions.Synchronized)] și are suport pentru blocări care se exclud reciproc prin intermediul cuvântului cheie blocare.

## Biblioteci

* using System;

Namespace-ul System este rădăcină pentru tipurile fundamentale din .NET. Acest namespace include clase care reprezintă tipurile de date de bază utilizate de toate aplicațiile, de exemplu, Object (rădăcina ierarhiei de moștenire), Byte, Char, Array, Int32 și String. Multe dintre aceste tipuri corespund tipurilor de date primitive pe care le folosește limbajul de programare.

* using System.Collections.Generic;

Conține interfețe și clase care definesc colecții generice, care permit utilizatorilor să creeze colecții puternic tipizate care oferă o siguranță și o performanță mai bună a tipului decât colecțiile negenerice puternic tipizate.

* using System.ComponentModel;

Oferă clase care sunt utilizate pentru a implementa comportamentul în timpul rulării și în timpul proiectării componentelor și controalelor. Acest namespace include clasele de bază și interfețele pentru implementarea atributelor și convertoarelor de tip, legarea la sursele de date și componentele de licențiere.

* using System.Data;

Oferă acces la clasele care reprezintă arhitectura ADO.NET. ADO.NET și permite construcția unor componente care gestionează eficient datele din mai multe surse.

* using System.Drawing;

Oferă acces la GDI + funcționalitatea grafică de bază. Namespace-urile System.Drawing.Drawing2D, System.Drawing.Imaging și System .Drawing.Text oferă funcționalități mai avansate.

* using System.Linq;

Oferă clase și interfețe care acceptă interogări care utilizează Interogare Integrată în Limbaj (LINQ).

* using System.Text;

Conține clase care reprezintă codificări de caractere ASCII și Unicode, clase de bază abstracte pentru conversia blocurilor de caractere în și din blocuri de octeți și o clasă de ajutor care manipulează și formatează obiectele String fără a crea instanțe intermediare ale String-urilor.

* using System.Windows.Forms;

Conține clase pentru crearea de aplicații bazate pe Windows care profită din plin de caracteristicile bogate ale interfeței utilizator disponibile în sistemul de operare Microsoft Windows.

* using System.Threading.Tasks;

Oferă tipuri care simplifică munca de scriere simultană și cod asincron. Principalele tipuri sunt Task, care reprezintă o operație asincronă care poate fi așteptată și anulată și Task <TResult>, care este o sarcină care poate returna o valoare.

## Constructori

Ori de câte ori o clasă sau o structură este creată, constructorul acesteia este apelat. O clasă sau o structură poate avea mai mulți constructori care iau argumente diferite. Constructorii permit programatorului să seteze valori implicite, să limiteze instanțierea și să scrie cod flexibil și ușor de citit.

Un constructor este o metodă al cărei nume este același cu numele tipului său. Semnătura metodei include doar un modificator de acces opțional, numele metodei și lista de parametri ai acesteia (nu include un tip de returnare).

public BubbleSort(int[] array\_In, int[] array\_copy\_In, int[] start\_pos\_In, Graphics g\_In, int MaxVal\_In, int dly\_In, int bar\_width\_In)

{

array = array\_In;

start\_pos = start\_pos\_In;

g = g\_In;

MaxVal = MaxVal\_In;

dly = dly\_In;

bar\_width = bar\_width\_In;

}

## Interfețe

O interfață definește un contract. Orice clasă sau structură care implementează acel contract trebuie să ofere o implementare a membrilor definiți în interfață. Începând cu C # 8.0, o interfață poate defini o implementare implicită pentru membri. De asemenea, poate defini membri statici pentru a oferi o implementare unică pentru funcționalitatea comună.

// bibliotecile necesare

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Vizualizare\_Sortari // numele algoritmului

{

internal interface Interface1 // denumirea interfeței

{

void NextStep(); // specificarea metodei

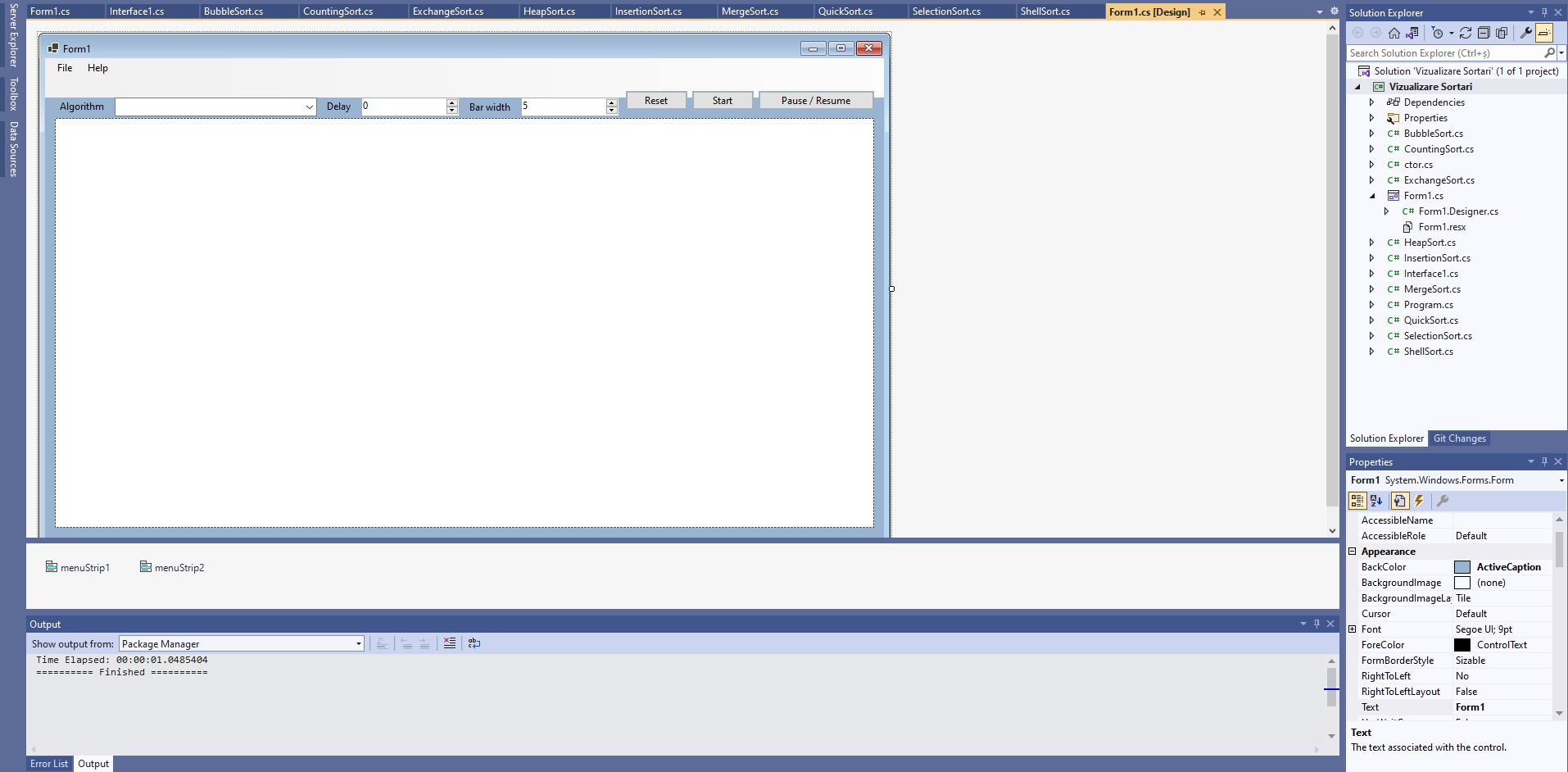
}

}

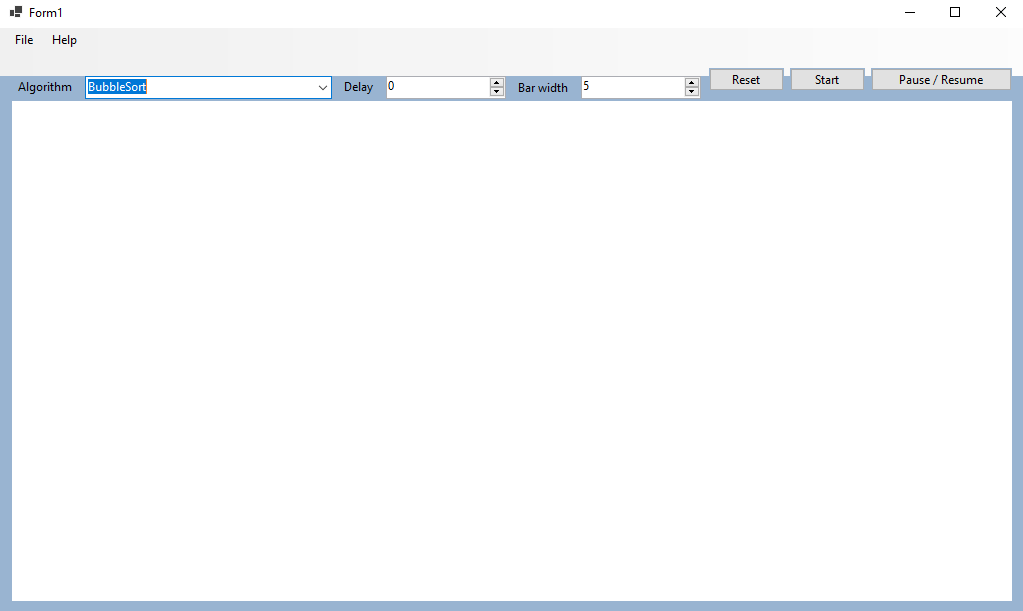
# Descrierea proiectului

Proiectul constă într-o aplicație de tipul Windows Froms realizată în Visual Studio 2022, scrisă în limbajul C#. Aceasta are rolul de a reprezenta grafic modul în care operează mai mulți algoritmi de sortare.

Interfața grafică folosește pachetul .NET specific Windows Forms. Aceasta este realizată cu ajutorul pachetului Design oferit în Visual Studio care permite editarea manuală, fără a folosi cod propriu-zis, a modului în care sunt așezate elementele aplicației.



În momentul rulării, aplicația arată astfel:



## Funcționalitatea butoanelor

### Butoanele din partea superioară

1. File - deschide un meniu din care se poate selecta opțiunea Exit, având ca efect închiderea programului;



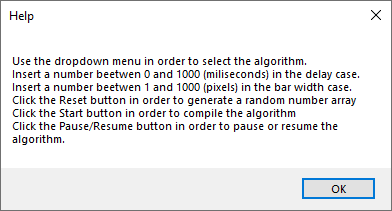
private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

1. Help – deschide o fereastră de tip Windows MessageBox. Rolul ei este de a familiariza utilizatorul cu funcționalitatea butoanelor.



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

{

string title = "Help";

string message =

"Use the dropdown menu in order to select the algorithm."

+ '\n' +

"Insert a number beetwen 0 and 1000 (miliseconds) in the delay case."

+ '\n' +

"Insert a number beetwen 1 and 1000 (pixels) in the bar width case."

+ '\n' +

"Click the Reset button in order to generate a random number array"

+ '\n' +

"Click the Start button in order to compile the algorithm"

+ '\n' +

"Click the Pause/Resume button in order to pause or resume the algorithm."

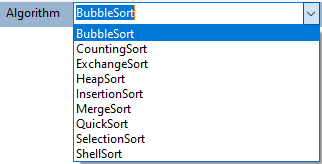
+ '\n';

MessageBox.Show(message, title, MessageBoxButtons.OK);

}

### Butoanele din partea mediană

1. Meniul pentru selectarea algoritmului – permite alegerea unuia dintre cei 9 algoritmi de sortare;



private void PopulateDropdown()

{

List<string> ClassList = AppDomain.CurrentDomain.GetAssemblies().Select

Many(x=>x.GetTypes()).Where(x=>typeof(Interface1).IsAssignableFrom(x) && !x.IsInterface && !x.IsAbstract).Select(x=>x.Name).ToList();

ClassList.Sort(); // extrage numele fișierelor în care sunt scrise sortările pentru popularea meniului

foreach (string entry in ClassList)

{

comboBox1.Items.Add(entry); // populează meniul

}

comboBox1.SelectedIndex = 0;

}

private void comboBox1\_SelectedIndexChanged(object sender,EventArgs e){}

// reține tipul selectat de sortare în momentul unei schimbări

1. Meniul pentru selectarea timpului de așteptare – permite selectarea unui număr cuprins între 0 și 1000, reprezentând numărul de milisecunde pe care programul le va aștepta după efectuarea fiecărei interschimbări;

public void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

dly= Convert.ToInt32(numericUpDown1.Value); // reține numarul în variabila dly

}

1. Meniul pentru selectarea lățimii barelor – permite selectarea unui număr cuprins între 2 și 1000 reprezentând lățimea barelor grafice, în pixeli. Afișarea are loc după ce butonul Reset este apăsat.

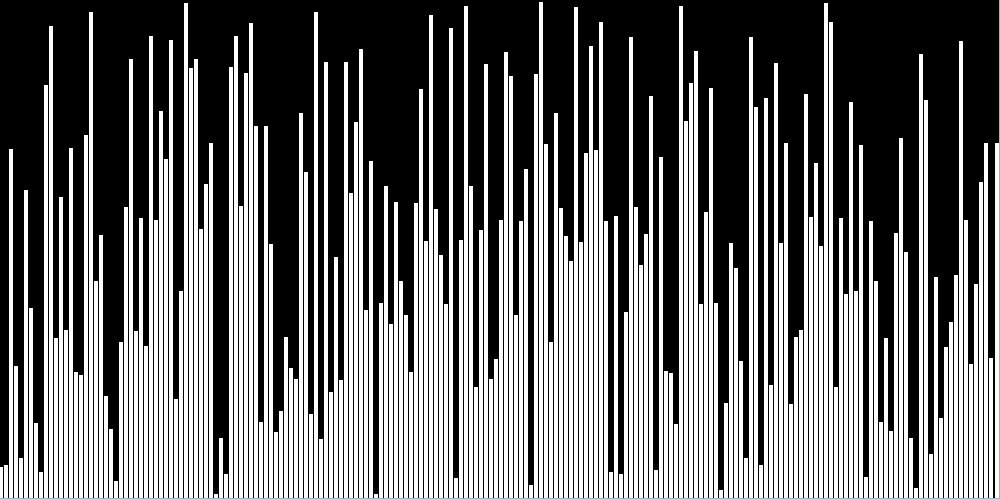
private void numericUpDown2\_ValueChanged(object sender, EventArgs e)

{

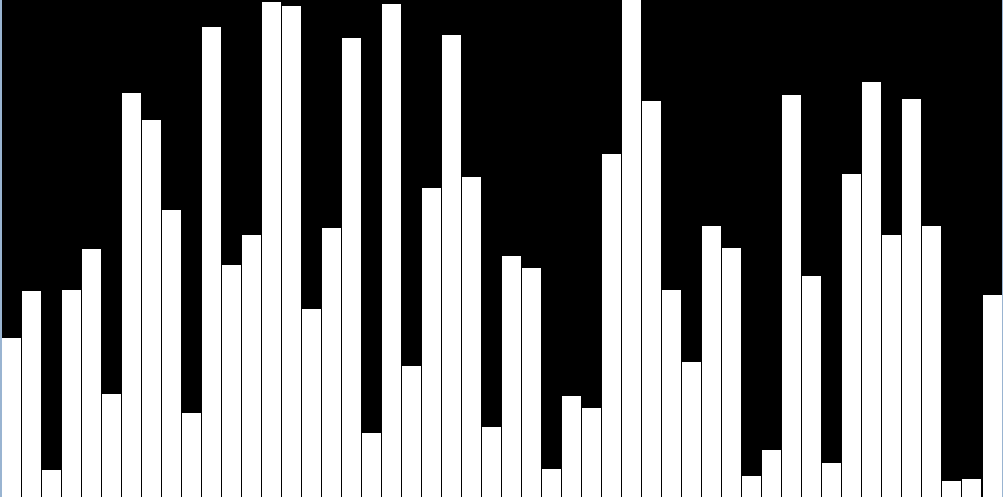
bar\_width= Convert.ToInt32(numericUpDown2.Value); // reține numarul în variabila bar\_width

if(bar\_width>panel1.Width)bar\_width=panel1.Width; // întrucât dimensiunile panoului grafic depind de cele ale ferestrei în care rulează aplicația, dacă numărul introdus de utilizator este mai mare decât cel care poate fi afișat, acesta va lua automat dimensiunea panoului

}



5 pixeli



20 pixeli

1. Reset – generează o listă nouă de numere și o afișează pe panoul grafic;

private void button1\_Click(object sender, EventArgs e)

{

if (bgw != null&&!bgw.IsBusy||bgw==null)

{

g = panel1.CreateGraphics(); // inițializează variabila grafică

int NumEntries = panel1.Width / bar\_width; // calulează numărul de elemente din listă pe baza dimensiunilor panoului

int MaxVal = panel1.Height;

current\_pos = 0;

Paused = false;

array = new int[NumEntries];

array\_copy = new int[NumEntries];

start\_pos = new int[NumEntries];

g.FillRectangle(new System.Drawing.SolidBrush(System.Drawing.

Color.Black), 0, 0, panel1.Width, MaxVal); // funcția de desenare

Random rand = new Random();

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

{

array[i] = rand.Next(0, MaxVal); // generează un număr random

array\_copy[i] = array[i];

}

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

{

start\_pos[i] = current\_pos; // reține ordonata fiecărei bare pentru eficientizarea afișărilor ulterioare

int spacing = 0;

if (NumEntries - i - 1 != 0) spacing = (1+panel1.Width –

current\_pos - (NumEntries - i) \* bar\_width) / (NumEntries - i - 1); // folosește o formulă matematică pentru a determina coordonatele fiecărei bare

g.FillRectangle(new System.Drawing.SolidBrush(System.Drawing.Color.White), current\_pos, MaxVal - array[i], bar\_width-1, MaxVal);

current\_pos += spacing + bar\_width;

}

}

}

1. Start – începe sortarea corespunzătoare algoritmului selectat în meniul aferent;

private void button2\_Click(object sender, EventArgs e)

{

if (bgw == null || !bgw.IsBusy) // verifică dacă nu cumva algoritmul deja rulează

{

if (array == null) button1\_Click(null, null);

bgw = new BackgroundWorker(); // variabilă care permite multi- threading-ul necesar pentru disponibilitatea utilizării butoanelor în timpul executării sortării

bgw.WorkerSupportsCancellation = true; // permite utilizarea butonului Pause/ Resume

bgw.DoWork += new DoWorkEventHandler(bgw\_DoWork);

bgw.RunWorkerAsync(argument: comboBox1.SelectedItem); // începe executarea sortării selectate

}

}

1. Pause / Resume – permite stoparea temporară, respectiv reluarea operațiunii de sortare. Această funcționalitate este implementată doar pentru 5 din cei 9 algoritmi ( bubble, prin numărare, interschimbare, inserție și selecție) întrucât ceilalți algoritmi folosesc funcții recursive ce nu pot fi întrerupte până la finalizare.

private void button3\_Click(object sender, EventArgs e)

{

if (bgw != null) //dacă nicio operație nu este în desfășurare

{

if (!Paused) //dacă algoritmul nu este deja pe pauză

{

bgw.CancelAsync(); //oprește executarea

Paused = true;

}

else //dacă este deja pe pauză

{

if (bgw.IsBusy) return;

int NumEntries = panel1.Width / bar\_width;

int MaxVal = panel1.Height;

Paused = false;

current\_pos = 0;

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

{

g.FillRectangle(new System.Drawing.SolidBrush

(System.Drawing.Color.Black), start\_pos[i], 0, bar\_width - 1, MaxVal);

g.FillRectangle(new System.Drawing.SolidBrush

(System.Drawing.Color.White), start\_pos[i], MaxVal - array[i], bar\_width - 1, MaxVal);

}

bgw.RunWorkerAsync(argument: comboBox1.SelectedItem);

//reîncepe sortarea

}

}

}

## Apelarea funcției constructor

public void bgw\_DoWork(object sender, System.ComponentModel.DoWorkEventArgs e)

{

BackgroundWorker bw = sender as BackgroundWorker;

string SortingName = (string)e.Argument;

Type type = Type.GetType("Vizualizare\_Sortari." + SortingName);

var ctors = type.GetConstructors();

try

{

Interface1 se = (Interface1)ctors[0].Invoke(new object[] {array,array\_copy,start\_pos,g,panel1.Height,dly,bar\_width});

// transmite parametrii necesari fișierelor secundare

while (!IsSorted() && (!bgw.CancellationPending))

{

se.NextStep(); // rezultă o complexitate de O(n) la care se va adăuga complexitatea funcției NextStep() specifică fiecărei sortări

}

} // apelarea interfeței

catch (Exception ex)

{

}

}

## Interfața

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Vizualizare\_Sortari

{

internal interface Interface1 // generează metoda de lucru pentru fiecare fișier secundar al programului

{

void NextStep();

}

}

## Algoritmi de sortare

Fiind metoda impusă prin interfață, fiecare fișier al sortărilor este alcătuit dintr-o funcție constructor care poartă numele tipului de sortare și dintr-o funcție NextStep() care reprezintă sortarea propriu-zisă, specifică fiecărui tip de sortare.

Funcția DrawBar(), identică în toate fișierele, care realizează afișarea barelor.

private void DrawBar(int pos, int h)

{

g.FillRectangle(GreenBrush, start\_pos[pos], 0, bar\_width-1, MaxVal); // pentru o mai bună vizualizare, bara este mai întâi colorată în verde

Task.Delay(dly).Wait(); // oprește programul pentru observarea mai bună a modului în care acesta funcționează

g.FillRectangle(BlackBrush, start\_pos[pos], 0,bar\_width-1,MaxVal);

g.FillRectangle(WhiteBrush, start\_pos[pos], MaxVal-h, bar\_width-1, MaxVal);

}

O altă funcție care se regăsește în majoritatea sortărilor( mai puțin sortarea prin numărare, prin interclasare și Shell Sort) este Swap() care realizează interschimbările și apelează funcția DrawBar().

private void Swap(int i)

{

int copy=array[i];

array[i]=array[i+1];

array[i+1]=copy;

DrawBar(i, array[i]);

DrawBar(i + 1, array[i + 1]);

}

#### Metoda bulelor – sau Bubble Sort, se bazează pe parcurgerea vectorului și interschimbarea oricăror două elemente învecinate care nu sunt în ordinea dorită.

public void NextStep()

{

for(int i = 0; i < array.Count()-1;i++)

{

if (array[i] > array[i + 1]) Swap(i);

}

}

Complexitatea este de O().

#### Sortarea prin numărare – sau Counting Sort, este o metodă de sortare a [vectorilor](https://infogenius.ro/vectori-cpp/) ce se bazează pe utilizarea unui [vector de frecvență](https://infogenius.ro/vectori-caracteristici-vectori-de-frecventa/). Acest algoritm de sortare poate fi extrem de eficient în anumite situații, în funcție de intervalul de valori al elementelor vectorului.

public CountingSort(int[] array\_In, int[] array\_copy\_In, int[] start\_pos\_In, Graphics g\_In, int MaxVal\_In, int dly\_In, int bar\_width\_In) // funcția constructor

{

array = array\_In;

array\_copy = array\_copy\_In;

start\_pos = start\_pos\_In;

g = g\_In;

MaxVal = MaxVal\_In;

dly = dly\_In;

frq = new int[MaxVal];

for (int j = 0; j < array\_copy.Count(); j++)

{

frq[array\_copy[j]]++; // construcția vectorului de frecvență în constructor

}

bar\_width = bar\_width\_In;

}

public void NextStep()

{

while (frq[step] != 0)

{

if (array[k] != step) // comparație pentru eficientizarea - dacă elementul nu este deja în această poziție

{

frq[step]--;

DrawBar(k, step);

array[k++] = step;

}

else

{

k++;

frq[step]--;

}

}

step++;

}

Are o complexitate de O(n+k), unde k este diferența dintre minim și maxim. Nu poate fi utilizată dacă k este foarte mare din cauza restricțiilor date de lungimea maximă a unui vector. Poate fi utilizată totuși folosind noțiunea de liste liniare, însă eficiența rămâne scăzută dacă k este foarte mare.

#### Sortarea prin interschimbare – sau Exchange Sort, constă în compararea unui termen cu fiecare termen ce îl urmează până la ultimul din vector, iar dacă cei doi termeni nu îndeplinesc criteriul, sunt interschimbați.

public void NextStep()

{

for (int i = step + 1; i < array.Count(); i++)

{

if (array[step] > array[i]) Swap(i, step);

}

step++;

}

Complexitatea este de O().

#### Heap Sort - este o tehnică de sortare bazată pe structura de date Binary Heap. Este similară cu sortarea prin selecție unde găsim mai întâi elementul minim și îl plasăm la început. Repetăm același proces pentru elementele rămase.

Un arbore binar complet este un arbore binar în care fiecare nivel, cu excepția eventualului ultim, este complet umplut și toate nodurile sunt cât mai departe posibil la stânga.

Un Binary Heap este un arbore binar complet în care elementele sunt stocate într-o ordine specială, astfel încât valoarea dintr-un nod părinte este mai mare (sau mai mică) decât valorile din cele două noduri secundare ale acestuia. Primul se numește max heap, iar cel de-al doilea se numește min heap. Heap-ul poate fi reprezentat printr-un arbore binar sau o matrice.

public void NextStep()

{

Sort(array, array.Count()); // vectorul va fi sortat după execuția funcției

}

private void CreateHeap(int[] array, int n, int i) // crearea Binary Heap

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

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

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

// determinarea fiului stâng și drept

if (largest != i)

{

Swap(i, largest); // aranjarea crescătoare

CreateHeap(array, n, largest); // funcție recursivă

}

}

private void Sort(int[] array, int n)

{

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

CreateHeap(array, n, i);

for (int i = n - 1; i > 0; i--)

{

Swap(0, i);

CreateHeap(array, i, 0);

}

}

Complexitatea este de O(n\*log(n)), indiferent de ordinea inițială a vectorului.

1. Sortarea prin inserție – sauInsertion Sort, se bazează pe ideea: dacă secvența cu indicii 0,1, ..., i-1 este ordonată, atunci putem insera elementul cu indicele i în această secvență astfel încât să fie ordonată secvența cu indicii 0, 1, ..., i.

public void NextStep()

{

int pos = step;

while (pos > 0 && array[pos] < array[pos - 1]) // căutare poziției de inserare

{

Swap(pos, pos - 1);

pos--;

}

step++;

}

Complexitatea este de O(). În cel mai favorabil caz, complexitatea este de O(n).

#### Sortarea prin interclasare - sau **Merge Sort,** este o metodă eficientă de sortare a elementelor unui tablou, bazată pe următoarea idee: dacă prima jumătate a tabloului are elementele sortate și a doua jumătate are de asemenea elementele sortate, prin [interclasare](https://www.pbinfo.ro/articole/5588/interclasarea-tablourilor) se va obține tabloul sortat. Sortarea prin interclasare este un exemplu tipic de algoritm [divide et impera](https://www.pbinfo.ro/articole/7651/divide-et-impera): se sortează o secvență delimitată de indicii stânga și dreapta.

public void NextStep()

{

Merge(array,0, array.Count() - 1); // vectorul va fi sortat după execuția funcției

}

private void Merge(int[] array, int left, int right)

{

if (left < right) // cazul de oprire al funcției recursive

{

// determinarea indicilor și apelul recursiv

int middle = (left + right) / 2;

Merge(array, left, middle);

Merge(array, middle + 1, right);

i = left, j = middle + 1, k = -1;

// algoritmul de interclasare

while (i <= middle && j <= right)

if (array[i] < array[j])

{

copy[++k] = array[i++];

}

else

{

copy[++k] = array[j++];

}

while (i <= middle)

{

copy[++k] = array[i++];

}

while (j <= right)

{

copy[++k] = array[j++];

}

// afișarea

for (i = left, j = 0; i <= right; i++, j++)

{

DrawBar(i, copy[j]);

array[i] = copy[j];

}

}

}

Complexitatea este de O(n\*log(n)), indiferent de ordinea inițială a vectorului.

#### Quick Sort – este un algoritm de tip [divide et impera](https://www.pbinfo.ro/articole/7651/divide-et-impera) care sortează o secvență a tabloului (inițial întreg tabloul) astfel: se alege un element special al listei, numit **pivot**, se ordonează elementele listei, astfel încât toate elementele din stânga pivotului să fie mai mici sau egale cu acesta, și toate elementele din dreapta pivotului să fie mai mari sau egale cu acesta, se continuă recursiv cu secvența din stânga pivotului și cu cea din dreapta lui.

public void NextStep()

{

Sort(array,0,array.Count()-1); // vectorul va fi sortat după execuția funcției

}

private void Sort(int[] array, int left, int right)

{

if (left < right) // condiția de oprire a algoritmului recursiv

{

int middle = (left + right) / 2; // determinarea indicilor

Swap(left, middle);

int i = left, j = right, d = 0;

while (i < j)

{

if (array[i] > array[j]) // realizarea ordonării în jurul pivotului

{

Swap(i, j);

d = 1 - d;

}

i += d;

j -= 1 - d;

}

// apelarea recursivă

Sort(array, left, i - 1);

Sort(array, i + 1, right);

}

}

Complexitatea variază între O(n\*log(n)) și O(), în funcție de ordonarea inițială a vectorului.

#### Sortarea prin selecție – sau Selection Sort, este o metodă de sortare care determină minimul dintr-o secvență și îl plasează la începutul acesteia. Repetarea acestui procedeu pentru fiecare secvență de tipul i, i+1, ..., n va conduce la sortarea vectorului.

public void NextStep()

{

posi=DetMin(step); // reține poziția minimului din secvența step, step+1, ..., n

if(array[posi]!=array[step]) // optimizează numărul de afisări

{

Swap(step,posi);

}

step++;

}

private int DetMin(int position) // determină poziția minimului din secvența step, step+1, ..., n

{

for (int i = position+1; i < array.Count(); i++)

{

if (array[i] < array[position]) position = i;

}

return position;

}

Complexitatea este de O().

#### Shell Sort - este în principal o variantă a sortării prin inserție. În sortarea prin inserție, mutăm elementele cu o singură poziție înainte. Când un element trebuie mutat mult înainte, sunt necesare multe mișcări. Ideea Shell Sort este de a permite interschimbarea de elemente îndepărtate. În Shell Sort, vectorul trebuie sa fie sortat-h pentru o valoare mare a h. Continuăm să reducem valoarea lui h până când aceasta devine 1. Se spune că un tablou este sortat-h dacă toate sublistele fiecărui element h’ sunt sortate.

public void NextStep()

{

Sort(array, array.Count()); // vectorul va fi sortat după execuția funcției

}

private void Sort(int[] array, int n)

{

for (int gap = n / 2; gap > 0; gap /= 2) // gap este h-ul din explicație

{

for (int i = gap; i < n; i += 1) // reazizarea sortării-h

{

int temp = array[i];

int j;

for (j = i; j >= gap && array[j - gap] > temp; j -= gap)

// shiftarea elementerlor

{

array[j] = array[j - gap];

DrawBar(j, array[j]);

}

array[j] = temp;

DrawBar(j,temp);

}

}

}

Complexitatea variază între O(n\*log(n)) și O(), în funcție de ordonarea inițială a vectorului.

# Bibliografie

* <https://en.wikipedia.org/wiki/C_Sharp_(programming_language)>
* <https://ro.wikipedia.org/wiki/.NET_Framework>
* <https://www.geeksforgeeks.org/shellsort/>
* <https://docs.microsoft.com/en-us/dotnet/csharp/>
* <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface>
* <https://docs.microsoft.com/en-us/dotnet/>