

Algoritmi de Sortare – DEMO

PROIECT DISCIPLINA POO

Autor: Vizitiu Valentin Iulian

Grupa 3112B

Suceava - 2023

TEMA SI MOTIVATIA ALEGERII

Tema proiectului:

Vizualizarea performanțelor diferitor algoritmi de sortare, prin afișarea grafica a vectorilor ce trebuie sortați, si testarea performanței unor algoritmi de sortare.

Motivația alegerii:

Am ales această temă deoarece am realizat un proiect similar în primul an al facultății, însă acel proiect nu a implicat utilizarea conceptelor de programare orientată pe obiect și a fost implementat în limbajul C, spre deosebire de proiectul actual care este implementat în limbajul C++. Prin acest nou proiect, doresc să compar progresul realizat în acești doi ani de studiu la facultate.

De asemenea, consider că este extrem de interesant să observăm cum diferiți algoritmi de sortare funcționează în spate și să putem vizualiza procesul de sortare în timp real. Prin acest proiect, ne propunem să evaluăm și să comparăm viteza de execuție a diferiților algoritmi de sortare.

CUPRINS:

Tema si motivația alegerii	2
----------------------------------	---

CAP I – ELEMENTE TEORETICE

Descrierea problemei	4
Abordarea teoretica a problemei	4
Elemente specifice POO folosite	6
Algoritmi de sortare implementați	7

CAP II – IMPLEMENTARE

Tehnologii folosite	9
Diagrama de clase	9

CAP III – ANALIZA SOLUTIEI IMPLEMENTATE

Formatul datelor de I/O	11
Studii de caz date I/O	11
Performante obținute	13
Complexitate timp algoritmi sortare	14

CAP IV – MANUALUL DE UTILIZARE

Navigarea meniului	16
Contribuțiile mele	17

CAP V – CONCLUZII

Concluzii implementare	17
------------------------------	----

CAPITOLUL I – ELEMENTE TEORETICE

1. Descrierea problemei:

Necesitatea algoritmilor de sortare în programare este fundamentală și esențială în dezvoltarea software-ului. Algoritmii de sortare sunt utilizați pentru a organiza și ordona datele într-o manieră eficientă și coerentă.

În majoritatea aplicațiilor software, manipularea și gestionarea datelor reprezintă o componentă crucială. Oricând lucrăm cu un volum mare de date sau când avem nevoie să prezentăm informațiile într-o anumită ordine, algoritmii de sortare devin indispensabili.

Prin utilizarea algoritmilor de sortare, putem obține avantaje semnificative în eficiența și performanța aplicațiilor noastre. Un algoritm de sortare bine ales și implementat poate reduce timpul de execuție și poate optimiza procesele de căutare, filtrare sau prezentare a datelor.

De asemenea, algoritmii de sortare sunt utilizați în rezolvarea unei varietăți de probleme complexe, precum căutarea elementului maxim sau minim într-o listă, eliminarea duplicatelor sau găsirea celor mai frecvente elemente. Ei oferă o bază solidă pentru dezvoltarea altor algoritmi și structuri de date.

Deci algoritmii de sortare joacă un rol esențial în programare, contribuind la organizarea și manipularea eficientă a datelor. Prin selectarea și implementarea corespunzătoare a acestor algoritmi, putem obține aplicații mai rapide, mai eficiente și mai fiabile, ceea ce duce la o experiență utilizator îmbunătățită și la optimizarea performanței generale a sistemelor software.

2. Abordarea teoretica a problemei:

Deoarece în acest an am avut recomandarea ca proiectul să ruleze în terminal, și să folosească limbajul C++, a fost o provocare să implementezi o asemenea aplicație doar în consolă, având un rang limitat de caractere și culori, și de asemenea ceea ce a fost un obstacol major a fost lipsa de spațiu.

Să luăm un exemplu:

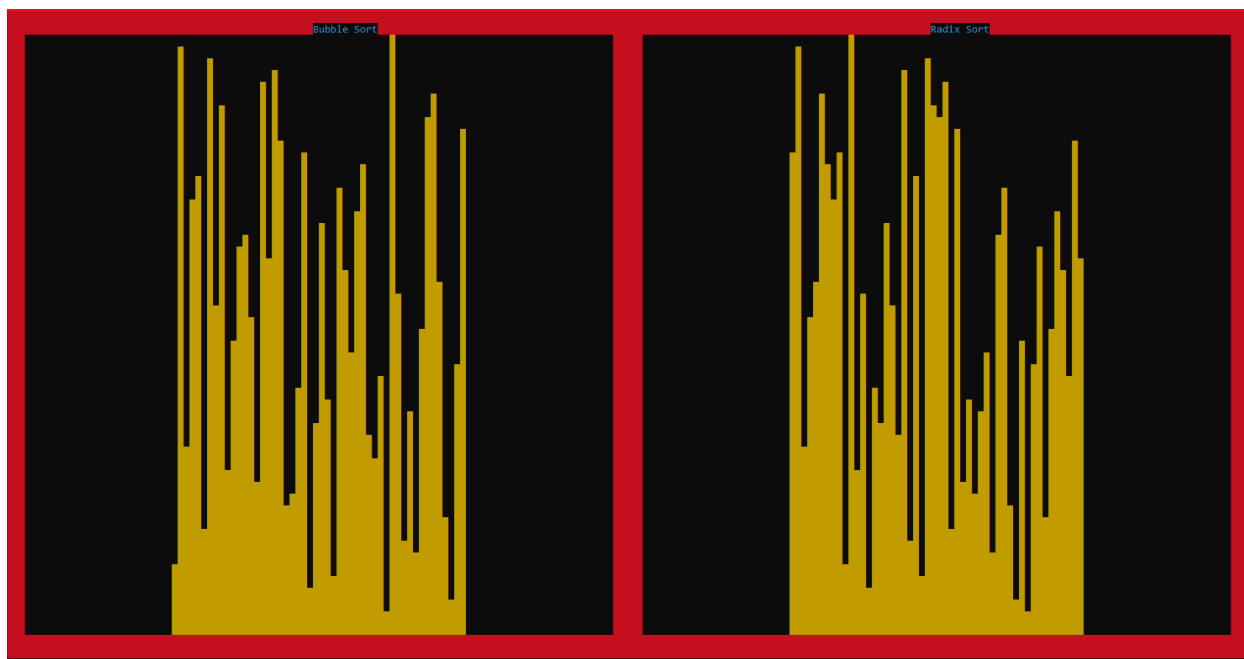
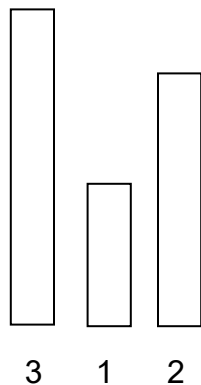


Figura 1 – Exemplu de doi algoritmi de sortare afișați side-by-side

În acest program vom avea de sortat vectori de numere (`std::vector<int>`), în reprezentarea grafică, fiecare linie galbenă reprezintă un număr, cu cât acest număr este mai mare, cu atât linie este mai înaltă pe verticală.



Acum problema apare: terminalele pot avea un număr limitat de caractere pe fiecare linie, chiar și pe un ecran de 1920x1080, maximul de caractere este limitat la aproximativ 230 caractere pe orizontală, și ~60 pe verticală.

Din această cauză, în vizualizarea grafică a algoritmilor, fiecare algoritm va sorta un vector de 50 elemente, dar chiar și în aceste restricții, font-ul terminal-ului trebuie să fie setat la cel puțin **13**.

Deci tot programul, inclusiv grafică va fi în terminal.

Salvarea in fişare – CSV:

In program va fi posibila salvarea in fişiere a stării curente a sortării algoritmilor.

Din cauza modului inteligent de lucru al aplicaţiei (sortarea nu are loc de fapt in timp real, acest lucru ar fi imposibil, deoarece nu putem defini ce înseamnă „un pas” pentru toţi algoritmi, de exemplu pentru bubble sort ar putea fi schimbarea a doua element, dar ce înseamnă un pas pentru un algoritm ca radix sort?, de asemenea ar fi prea complex sa oprim sortarea si sa vizualizam starea vectorilor). Din acest motiv am venit cu următoarea soluţie: Sortarea se face in prealabil, înainte de a fi afişata in mod grafic, iar fiecare „pas” din algoritm va fi salvat intr-un vector.

Deoarece in final toata sortarea va fi intr-un vector de vectori de numere, putem sa salvam acest vector intr-un fişier!

Am ales formatul CSV pentru acest lucru.

Aşa arata un exemplu de sortare salvata in fişier:

Bubble Sort

1, 3, 4, 5

1, 2, 4, 5

1, 3, 5, 6

Selection Sort

1, 3, 4, 5

3, 4, 5 6

Doua linii vor fi numele algoritmilor folosiţi, care servesc si rolul de delimitator pentru cei doi vectori de vectori de numere salvaţi.

Citirea din fişiere este simpla:

1. Citeşte numele primului algoritm
2. Citeşte fiecare linie din vectorul de vectori de numere si separa numerele de virgula, si converteşte-le la int.
3. Repeta pana la numele celui de-al doilea algoritm.

3. Elemente specifice POO:

Toata aplicaţia foloseşte clasele, nu exista funcţii care nu aparţin nici unei clase.

Daca a fost nevoia de a crea o funcţie mai generala, folosita in toata aplicaţia, sau care nu se potrivea unei clase, aceasta a fost declarata ca funcţie statica in clasa [Utilities](#).

Abstractizarea:

Fiecare clasa din program arata public doar metodele pe care utilizatori ar trebui sa le folosească, iar clasele fac lucruri in spate de care noi nu știm.

De exemplu clasa ce este responsabila cu salvarea si citirea datelor din fișier verifica daca folder-ul pentru salvări este creat, iar daca nu îl creează, acest lucru se întâmpla in constructorul clasei.

Unii algoritmi de sortare au mai multe metode ajutătoare declarate private.

Exista o clasa speciala pentru logare, abstractizează cum se va afișa textul colorat si informațiile despre logare, noi doar trebuie sa-i furnizam un sir de caractere.

Modul in care sortarea se face pentru fiecare algoritm este ascunsa in interiorul clasei, noi doar trebuie sa apelam metoda Sort(), iar apoi avem acces la vectorul sortat.

Încapsularea:

Clasele nu dau acces public la nici o data a claselor. Daca este nevoie de aceste date, ele sunt accesate folosind metode de get si set.

Moștenirea & Polimorfismul:

Acesta este unul dintre cele mai utilizate concepte folosite in program.

Fiecare algoritm de sortare moștenește din clasa abstracta [SorterBase](#). Aceasta clasa definește o metoda abstracta [Sort\(\)](#) ce trebuie implementata obligatoriu de orice algoritm de sortare, similar unor interfețe in alte limbaje de programare, ca C#.

Aceasta clasa si implementează câteva metode utile folosite de toți ceilalți algoritmi de sortare, cum ar fi [RandomizeArray\(\)](#), ce va amesteca numerele din vector, înainte de a fi sortate.

Acest lucru face programul mult mai organizat si rapid de scris, deoarece nu trebuie sa ne repetam din nou in fiecare clasa, iar modificările sunt făcute intr-un singur loc.

4. Algoritmi de sortare implementați:

Deoarece acest program a fost creat pentru vizualizarea sortării algoritmilor, nu am putut alege orice algoritm de sortare, a fost nevoie de niște algoritmi de sortare care se prezinta intr-un mod interesant când ne uitam la un vector de numere.

Algoritmi aleși sunt:

1. **Bubble Sort:** Bubble Sort este un algoritm simplu de sortare care compară repetat elementele adiacente și le interschimbă până când lista este complet sortată. [Link către Wikipedia](#)
2. **Selection Sort:** Selection Sort este un algoritm de sortare care selectează cel mai mic (sau cel mai mare) element și îl plasează în poziția corectă, repetând acest proces pentru restul listei. [Link către Wikipedia](#)
3. **Quick Sort:** Quick Sort este un algoritm de sortare eficient bazat pe principiul "împărțește și cucerește". Alege un element pivot și rearanjează lista astfel încât elementele mai mici să fie înaintea pivotului și cele mai mari să fie după pivot, apoi repetă procesul pentru subliste. [Link către Wikipedia](#)
4. **Radix Sort:** Radix Sort este un algoritm de sortare non-comparativă care sortează elementele pe baza cifrelor acestora. Se bazează pe principiul de a sorta elementele de la cifra cea mai puțin semnificativă către cea mai semnificativă. [Link către Wikipedia](#)
5. **Merge Sort:** Merge Sort este un algoritm de sortare recursiv care împarte lista în două părți egale, le sortează separat, iar apoi le combina într-o singură listă sortată. [Link către Wikipedia](#)
6. **Cocktail Sort:** Cocktail Sort, cunoscut și sub numele de "Shaker Sort" sau "Bidirectional Bubble Sort", este o variantă a Bubble Sort care parcurge lista în ambele sensuri, interschimbând elementele adiacente în funcție de ordinea dorită. [Link către Wikipedia](#)
7. **Comb Sort:** Comb Sort este un algoritm de sortare bazat pe comparații care îmbină elemente învecinate cu un factor de scalare pentru a realiza sortarea într-un timp mai eficient. [Link către Wikipedia](#)
8. **Gnome Sort:** Gnome Sort, cunoscut și sub numele de "Stupid Sort" sau "Slow Sort", este un algoritm de sortare care compară și interschimbă elementele adiacente, deplasându-se înapoi în listă atunci când ordinea este greșită. [Link către Wikipedia](#)
9. **Odd-Even Sort:** Odd-Even Sort, cunoscut și sub numele de "Brick Sort" sau "Pairwise Sorting Network", este un algoritm de sortare paralel care compară și interschimbă elementele adiacente într-un mod alternativ până când lista este complet sortată. [Link către Wikipedia](#)
10. **Cycle Sort:** Cycle Sort este un algoritm de sortare in-place care minimizează numărul de scrieri pe disc prin rearanjarea elementelor în pozițiile corecte într-un număr minim de cicluri. [Link către Wikipedia](#)

CAPITOLUL II – IMPLEMENTARE

1. Tehnologii folosite:

Limbaj de programare: C++

IDE: Visual Studio

UML Diagram: draw.io

Am avut alegerea între CodeBlocks și Visual Studio, dar am continuat cu Visual Studio deoarece are mult mai multe abilități, și este mai ușor de folosit, și am mai multă experiență cu IDE-uri similare.

2. Diagrama de clase:

În codul sursă există comentarii ce explică cum funcționează fiecare clasă. De asemenea, multe din metode au un număr mare de parametri, ceea ce face dificilă afișarea lor într-o diagramă UML. Pentru a vedea în mod detaliat rostul fiecărei metode, referiți-vă la codul sursă.

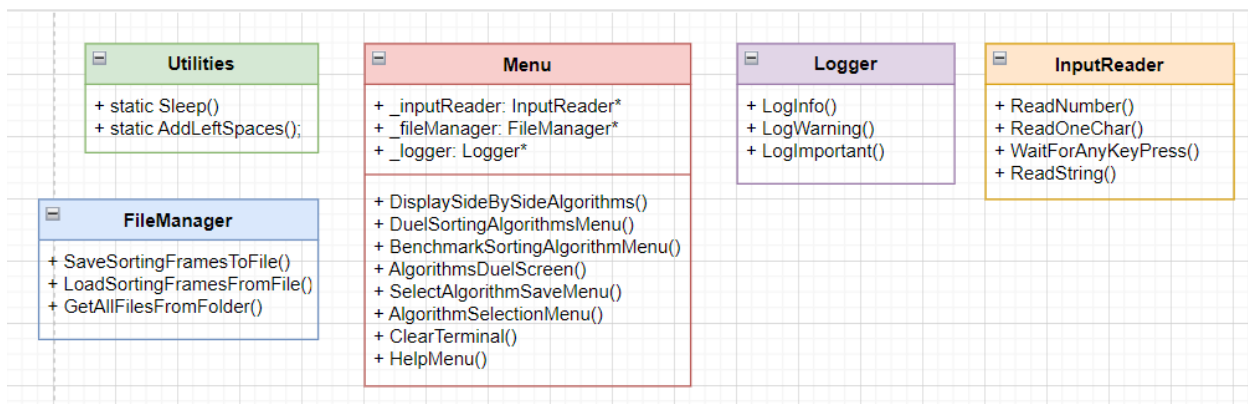


Figura 2 – Partea unu din clasele implementate

Aceste clase sunt de sine stătătoare și nu depind de alte clase pentru a funcționa.

Clasa Utilities:

- Această clasă este folosită pentru a încapsula metode ce nu se încadrează în nici o altă clasă

Clasa Menu:

- Această este clasa principală a programului, ea instanțiază toate celelalte obiecte, și dirijează flow-ul aplicației

- Conține metode pentru afișarea meniului, lucrul cu terminalul, si încapsulează logica complicata a programului
- Am folosit Dependency Injection pentru a rezolva dependențele clasei

Clasa Logger:

- Aceasta clasa este folosita pentru logarea diferitelor evenimente in consola

Clasa InputReader:

- Aceasta clasa este folosita pentru a prelua date de la utilizator, si de a le procesa pentru a fi folosite in aplicație

Clasa FileManager:

- Aceasta clasa este folosita pentru lucrul cu fișiere in program

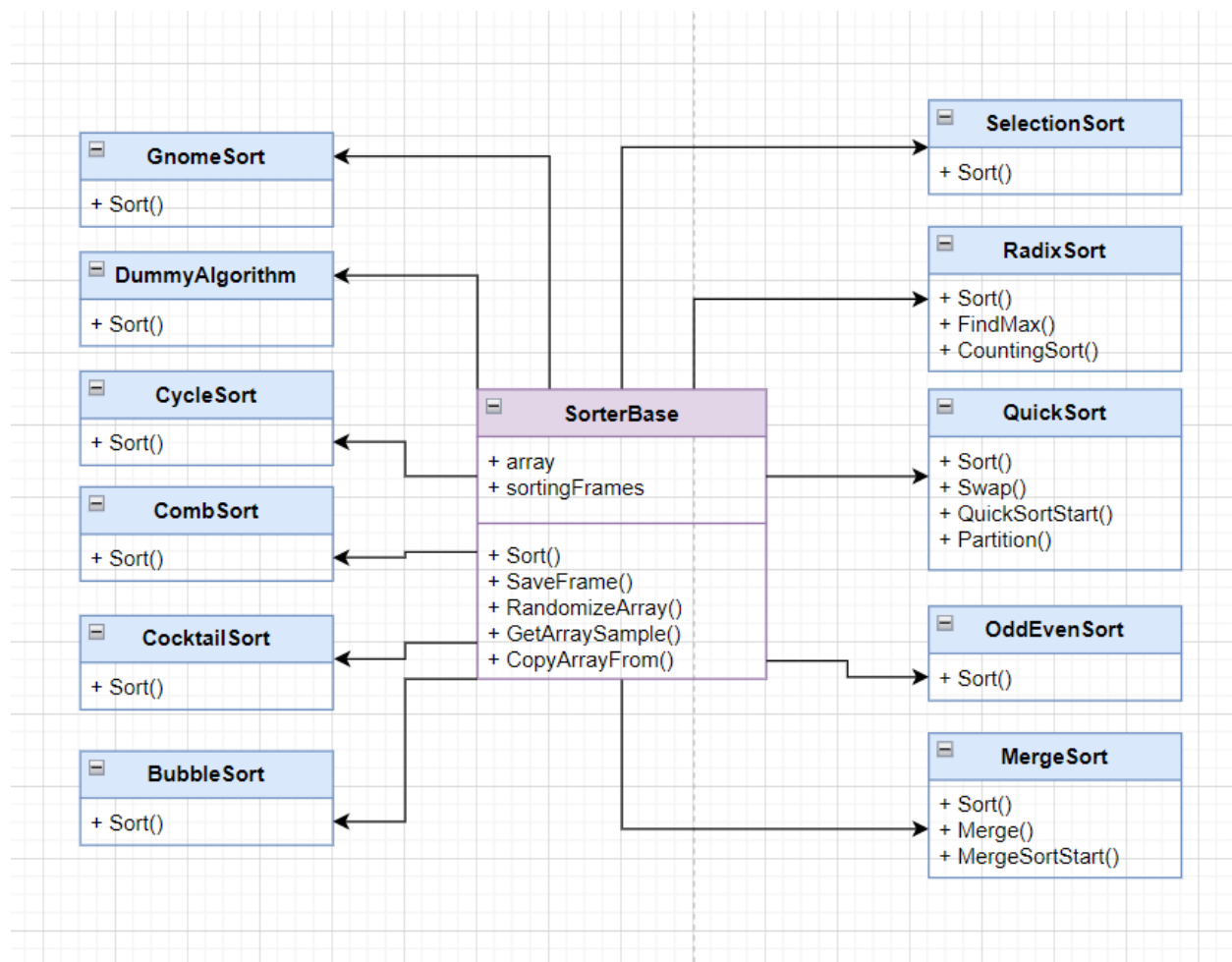


Figura 3 – Partea doua din clasele implementate

Aceste clase se ocupa cu sortarea din program. Toate aceste clase moștenesc din clasa **SorterBase**.

Am explicat mai multe in partea de moștenire si polimorfism de la capitolul 2.

CAPITOLUL III – ANALIZA SOLUTIEI IMPLEMENTATE

1. Formatul datelor de I/O:

Programul va primi date din exterior prin doua moduri:

- Date introduse de utilizator (alegerea opțiunilor in meniu, introducerea numelui unui fișier, etc);
- Date preluate din fișiere CSV, ce reprezintă salvări ale stării unor sortări precedente.

Pentru datele preluate de la utilizator, acestea sunt in principal numere, care reprezintă opțiunile de navigare din meniu. Un alt loc in care utilizatorii trebuie sa introducă date este la accesarea opțiunii de salvare in fișier in timpul rulării vizualizării sortării, când utilizatorii trebuie sa introducă un sir de caractere ce reprezinta numele fișierului in care se vor salva datele.

De asemenea utilizatorii vor fi rugați sa apese orice tasta pentru a continua rularea programului, lucru când apăsarea oricărei taste este suficienta.

Preluarea datelor de la utilizator se face cu ajutorul clasei [InputReader](#). Aceasta clasa validează toate datele, si le returnează intr-un format pe care programul le poate folosim, deci nu exista posibilitatea de a introduce date invalide, iar utilizatorul va primi un mesaj clar daca ceva este in neregula.

Pentru comunicarea cu exteriorul, datele de output sunt salvate ca fișiere CSV (coma separated values).

Numerele din vectorul de numere sunt salvate in fișier, separate de virgula. Doua linii din acest fișier sunt rezervate salvării numelui algoritmului de sortare utilizat.

Ne putem imagina un vector bidimensional, fiecare linie din vector conține acele numere separate de virgula, iar totalitatea liniilor reprezintă fiecare „pas” din sortarea făcută de algoritm.

2. Studii de caz – date I/O:

```
Please select an option:
1. Run two algorithms side by side
2. Benchmark two algorithms
3. Load previously saved algorithm duels
4. Help
5. Exit

>>> 1
```




Figura 4 – Introducerea opțiunii în meniu

Opțiunile în meniu sunt reprezentate de numere, utilizatorul doar trebuie să introducă numărul asociat opțiunii și să apese tasta ENTER.

```
1
Please give a name for the save:
>>> testsave
The state has been saved to a file. Load it again from the main menu.
Press any key to continue . . .
```




Figura 5 – Introducerea numelui unui fișier

La accesarea opțiunii de salvare, utilizatorii sunt rugați să introducă orice nume pentru un fișier. Lungimea maximă este de 255 caractere.

De asemenea după introducerea numelui fișierului, sunt rugați să apese orice tasta pentru a continua.

1	Bubble Sort		
2	3, 30, 49, 1, 17, 14, 10, 50, 41, 35, 11, 47, 20, 38, 16, 24, 7, 2, 34, 18, 29, 12, 45, 48, 32, 8, 22, 28, 4, 23, 36, 25, 44, 21, 42, 39, 40, 46, 33, 15, 26, 27, 37, 31, 19, 9, 6, 5, 13, 43		
3	3, 30, 1, 49, 17, 14, 10, 50, 41, 35, 11, 47, 20, 38, 16, 24, 7, 2, 34, 18, 29, 12, 45, 48, 32, 8, 22, 28, 4, 23, 36, 25, 44, 21, 42, 39, 40, 46, 33, 15, 26, 27, 37, 31, 19, 9, 6, 5, 13, 43		
4	3, 30, 1, 17, 49, 14, 10, 50, 41, 35, 11, 47, 20, 38, 16, 24, 7, 2, 34, 18, 29, 12, 45, 48, 32, 8, 22, 28, 4, 23, 36, 25, 44, 21, 42, 39, 40, 46, 33, 15, 26, 27, 37, 31, 19, 9, 6, 5, 13, 43		
5	Merge Sort		
6	20, 45, 42, 28, 12, 36, 34, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
7	20, 45, 28, 42, 12, 36, 34, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
8	20, 28, 42, 45, 12, 36, 34, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
9	20, 28, 42, 45, 12, 36, 34, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
10	20, 28, 42, 45, 12, 36, 34, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
11	12, 20, 28, 34, 36, 42, 36, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
12	12, 20, 28, 34, 36, 42, 45, 24, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
13	12, 20, 28, 34, 36, 42, 45, 10, 24, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
14	12, 20, 28, 34, 36, 42, 45, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
15	12, 20, 28, 34, 36, 42, 45, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
16	12, 20, 28, 34, 36, 42, 45, 10, 22, 31, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
17	12, 20, 28, 34, 36, 42, 45, 10, 15, 18, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
18	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 22, 24, 31, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
19	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 24, 31, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
20	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 31, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
21	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
22	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
23	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 37, 44, 46, 21, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
24	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 37, 44, 21, 46, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
25	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 37, 44, 21, 46, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
26	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
27	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 29, 26, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
28	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 26, 29, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
29	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 26, 29, 35, 32, 40, 49, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
30	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 26, 29, 35, 32, 49, 40, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
31	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 26, 29, 35, 32, 40, 49, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		
32	10, 12, 15, 18, 20, 22, 24, 28, 31, 34, 36, 42, 45, 5, 21, 37, 44, 46, 48, 26, 29, 32, 35, 40, 49, 3, 23, 6, 7, 14, 9, 27, 33, 16, 17, 43, 4, 2, 30, 13, 38, 41, 11, 50, 39, 19, 25, 1, 8, 47		

Figura 6 – Modul de salvare a datelor in fişierul CSV

Pentru mai multe explicații, verificați subcapitolul precedent.

3. Performante obținute:

Aplicația rulează perfect, dar încă exista câteva probleme:

La desenarea pe ecran (rendering) a tuturor caracterelor, de pot observa pâlpâieli ale ecranului, sau putem vedea cum sunt introduse caracterele unul cate unul in consola. Acest lucru se datorează faptului ca consola nu a fost creata pentru a șterge si rescrie așa multe caractere intr-un timp foarte scurt, ea trebuie sa rămână o interfața doar pentru introducerea de comenzi simple.

O modalitate in care am diminuat aceasta problema este folosirea redusa de culori, si concatenarea tuturor caracterelor de afișat într-un string înainte de a fi afișate.

Performantele algoritmilor:

Pentru mai multe detalii in privința cu performantele fiecărui algoritm, consultați paginile de wikipedia de la capitolul 1.4.

Dar in principiu, fiecare algoritm de sortare poate fi exprimat in complexitatea in timp folosind [notația O mare](#).

Iată un exemplu:

```
[INFO] Shuffling the arrays to sort...
[IMPORTANT] The array looks like this before the sorting: [42
[WARNING] Sorting started for Cycle Sort, please wait...
[WARNING] Sorting started for Quick Sort, please wait...
[INFO] Displaying the arrays to make sure the sorting was cor
[IMPORTANT] Array sorted by Cycle Sort: [1, 2, 3, 4, 5, 6, 7,
[IMPORTANT] Array sorted by Quick Sort: [1, 2, 3, 4, 5, 6, 7,
[INFO] THESE ARE THE RESULTS :

Algorithm          |      Time [seconds]
-----
Cycle Sort         |      32.794378
Quick Sort         |      0.018961

Press any key to continue . . . _
```

Figura 7 – Exemplu performate algoritmi de sortare

Folosind opțiunea de testarea a performatelor algoritmilor din aplicație, putem vedea ca doar pentru un vector de 50 de mii de numere, Cycle sort este MULT mai încet decât Quick sort, care a sortat vectorul într-o fracțiune de secunda.

Acestea sunt complicitățile in timp pentru toți algoritmi implementați in program:

- **Bubble Sort:** $O(n^2)$
- **Selection Sort:** $O(n^2)$
- **Quick Sort:** $O(n \log n)$ în cazul mediu, $O(n^2)$ în cel mai rău caz
- **Radix Sort:** $O(d * (n + b))$, unde d reprezintă numărul de cifre din cel mai mare număr, n este numărul de elemente și b este baza sistemului de numerație (în mod obișnuit 10 pentru zecimal)
- **Merge Sort:** $O(n \log n)$
- **Cocktail Sort:** $O(n^2)$
- **Comb Sort:** $O(n^2)$
- **Gnome Sort:** $O(n^2)$
- **Odd-Even Sort:** $O(n^2)$
- **Cycle Sort:** $O(n^2)$

Rețineți că aceste complexități temporale reprezintă scenariul cel mai nefavorabil, cu excepția cazului în care se specifică altfel. În plus, unele dintre aceste algoritme de sortare au variații sau optimizări care pot îmbunătăți performanța lor în anumite cazuri.

CAPITOLUL IV – MANUALUL DE UTILIZARE

Acest program permite vizualizarea in mod grafic a sortării in timp real a unor vectori de numere, folosind oricare 10 algoritmi de sortare. Aceste vizualizări de fac in perechi de cate 2 algoritmi odată, pentru a putea compara care din algoritmi este mai eficient, dar si pentru a vedea cum se sortează de fapt niște date.

Programul permite de asemenea sa cronometram performatele algoritmilor de sortare pe vectori de numere de orice mărime (rezonabila) dorim, iar la final putem vedea timpul necesar de sortare pentru fiecare algoritm.

Navigarea meniului:



```
Sorting Algorithms DEMO

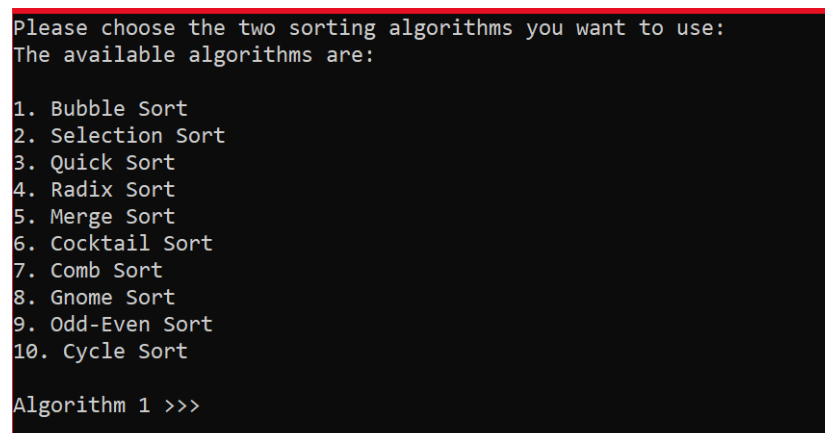
Please select an option:
1. Run two algorithms side by side
2. Benchmark two algorithms
3. Load previously saved algorithm duels
4. Help
5. Exit

>>>
```

Figura 8 – Meniul Principal al aplicației

La pornirea aplicației veti fi prezentat cu acest meniu.

Opțiunea 1 permite vizualizarea grafica a doi algoritmi aleși de dumneavoastră.



```
Please choose the two sorting algorithms you want to use:
The available algorithms are:

1. Bubble Sort
2. Selection Sort
3. Quick Sort
4. Radix Sort
5. Merge Sort
6. Cocktail Sort
7. Comb Sort
8. Gnome Sort
9. Odd-Even Sort
10. Cycle Sort

Algorithm 1 >>>
```

Figura 9 – Meniul de selecție algoritmi

În figura 9 putem observa cum arată sub meniul pentru alegerea celor doi algoritmi care vor fi utilizați, pur și simplu introduceți două numere pentru a selecta algoritmi.

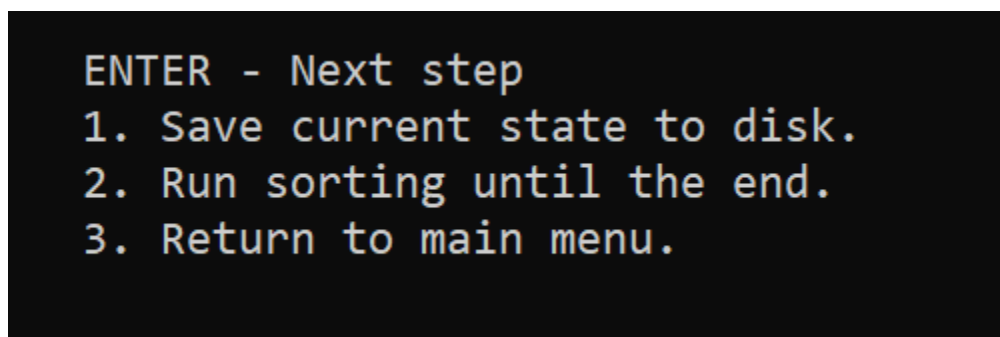


Figura 10 – Sub meniul din vizualizarea algoritmilor

După alegerea celor doi algoritmi, veți fi prezentați cu modul grafic în care se vor reprezenta vectorii, citiți capitolul unu pentru mai multe informații.

Sub meniul din figura 10, explica cum să derulam vizualizarea sortării.

Apăsând tasta ENTER, programul va rula „un pas” din sortare.

Sub-opțiunea 1 permite salvarea într-un fișier CSV a întregii vizualizări (sub forma de șiruri de numere!), pentru a putea fi reluată în mod ulterior. La accesarea acestei opțiuni veți fi rugat să introduceți numele salvării, pentru a o recunoaște mai târziu.

Puteți vedea un exemplu în figura 5.

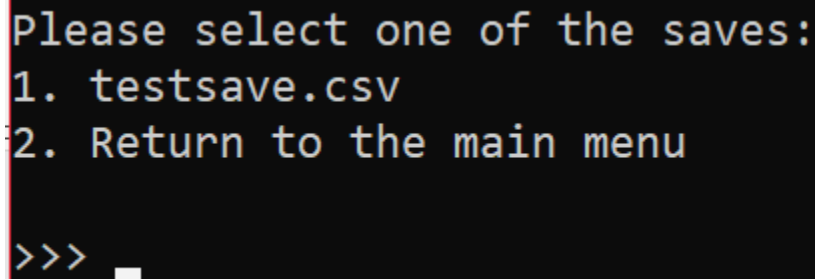
Sub-opțiunea 2 va lua controlul utilizatorului de la program, până când sortarea se va derula până la capăt. ATENȚIE! Dacă folosiți algoritmi de sortare lăniți, acest lucru ar putea dura foarte mult. Aveți grijă când accesați această opțiune.

Sub-opțiunea 3 se explică de la sine.

Opțiunea 2 a meniului principal permite cronometrarea performanțelor a doi algoritmi de sortare. După accesarea opțiunii veți fi prezentat cu meniul din figura 9, iar după alegerea celor doi algoritmi de sortare, programul va rula în fundal sortările, iar la final veți vedea ecranul din figura 7.

Opțiunea 3 a meniului principal permite re-încărcarea unei vizualizări precedente.

La accesarea acestei opțiuni, trebuie să introduceți numărul ce corespunde salvării.



```
Please select one of the saves:
1. testsave.csv
2. Return to the main menu

>>> _
```

Figura 11 – Meniul de reîncărcare al unei vizualizări precedente

Opțiunea 4 a meniului principal va afișa un text scurt ce explica cum se utilizează programul.

Contribuțiile mele:

Am realizat acest program pe cont propriu, programul ar putea fi o continuare a proiectelor făcute de mine în anul 1 de facultate, când am implementat 7 algoritmi de sortare în limbajul C, și am făcut vizualizarea similară în consolă, sau a unui alt proiect scris în Python, ce permite vizualizarea în mod grafic a sortării, folosind librăria pygame.

CAPITOLUL V – CONCLUZII

Avantajele acestei implementări este faptul că vizualizarea se face pur și simplu în consolă, ceea ce a făcut implementarea mult mai rapidă.

Programul permite utilizatorilor să vizualizeze sortările făcute de algoritmi și performanțele acestora.

Programul prezintă și dezavantaje, consola nu a fost creată pentru acest lucru, din acest motiv prezintă pâlpâieli la vizualizarea sortării.

De asemenea, dacă programul sortează „repede” vectorul când vizualizăm grafic, acest lucru nu înseamnă nimic, deoarece vectorii sunt sortați înainte ca vizualizarea să înceapă, aceasta vizualizare nu prezintă nimic mai mult decât un mod interesat de a vizualiza algoritmi.

Nici nu este posibil pentru a face această vizualizare să prezinte performanțele actuale

dintre doi algoritmi, deoarece noțiunea de „un pas” în sortare este diferită pentru toți algoritmi.

Programul doar rulează fiecare sortare rând pe rând, și salvează starea vectorului de sortat în alt vector („sorting frames”), ne putem imagina un video, iar fiecare stare este un cadru în acest video.

Dar acest lucru a fost perfect când s-a propus salvarea datelor în fișiere text, deoarece noi având deja toate cadrele pre-procesate, doar trebuie să le salvăm în fișier, și să le citim ulterior.

Dar acest lucru nu este o așa mare problemă! Deoarece programul are și opțiunea de sortare a unor vectori de mărimi imense (de fapt numărul de elemente din vector este dat de către utilizator), și ne prezintă cu timpul în secunde pentru care a durat sortarea, ceea ce ne permite să vedem mai bine performanțele.

Un lucru de notat este că timpi dați de acest program nu trebuie luați ca atare, programul sortează doar numere, dar în viață reală vom avea de sortat mult mai multe tipuri de date, cum ar fi șiruri de caractere, date calendaristice, etc. Deci înainte de a decide ce algoritm să folosiți, documentați-vă.

Un alt lucru este că acești algoritmi au o implementare simplă, doar bazele, se poate face de asemenea optimizări fiecărui algoritm, ceea ce îl poate face mult mai rapid.

În continuare se pot face multe optimizări și îmbunătățiri.

Un prim lucru ar fi folosirea unei librării grafice pentru afișarea vizualizării sortării. Acest lucru deja ne-ar permite să afișăm vectori mult mai mari, și într-un mod mult mai ușor de înțeles.

De asemenea programul ar putea rula un sunet de fiecare dată când algoritmul schimbă două elemente între ele, iar sunetul ar putea avea volumul proporțional cu mărimea numărului ce a fost schimbat. Linia ce reprezintă numărul de schimbat ar putea de asemenea să fie colorată cu o altă culoare pentru a ține pasul cu sortarea mai ușor.

De asemenea am putea adăuga mult mai mulți algoritmi de sortare, și versiuni diferite ale acestora. Un lucru interesant ar fi să vizualizăm și comparăm implementările algoritmilor de sortare din diferite limbaje de programare.

In concluzie, acest program este util pentru compararea vitezelor algoritmilor de sortare, și vizualizarea în timp real a sortării a doi algoritmi. Implementarea acestui program m-a ajutat să înțeleg mai bine limbajul C++ și cum funcționează diferiți algoritmi de sortare.

BIBLIOGRAFIE:

Visual Studio - <https://visualstudio.microsoft.com/>

Draw.io - <https://app.diagrams.net/>

C++ - <https://en.wikipedia.org/wiki/C%2B%2B>