**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr. 5

la cursul ***„Structuri de date şi algoritmi”***

**Tema: Algoritmi de sortare şi căutare**

A efectuat :  **St. gr. CR-221FR Serba Cristina**

**A verificat: Asis.univ. Toma Olga**

**Chișinău 2023**

**Cuprins**

[INTRODUCERE 2](#_Toc125243894)

[REALIZAREA PRACTICĂ A SARCINII DE LUCRU 3](#_Toc125243895)

[CONCLUZII 7](#_Toc125243896)

[BIBLIOGRAFIE 8](#_Toc125243897)

# 

# INTRODUCERE

Algoritm - o procedura / o secventa de pasi care rezolva o problema data intr-un mod repetabil si cu resurse (timp si memorie) finite; primeste un set de date de intrare si returneaza o solutie – set de date de iesire

Analiza complexității unui algoritm are ca scop estimarea volumului de resurse de calcul necesare pentru execuția algoritmului. Prin resurse se înțelege:

* Spațiul de memorie necesar pentru stocarea datelor pe care le prelucrează algoritmul.
* Timpul necesar pentru execuția tuturor prelucrărilor specificate în algoritm.

Această analiză este utilă pentru a stabili dacă un algoritm utilizează un volum acceptabil de resurse pentru rezolvarea unei probleme. In acest fel timpul de executie va fi exprimat prin numarul de operatii elementare executate. Sunt considerate operatii elementare cele aritmetice (adunare, scadere, ınmulțire, ımpartire), comparatiile si cele logice (negatie, conjuncte și disjunctie).

Este așadar suficient sa se contorizeze doar anumite tipuri de operații elementare, numite operații de bază. Timpul de executie al ıntregului algoritm se obtine ınsumand timpii de executie ai prelucrarilor componente.

Programul dat urmareste sa traga concluzii de eficacitatea efectuarii a doua algoritme de sortare in dependenta de numarul de permutari si comparatii.

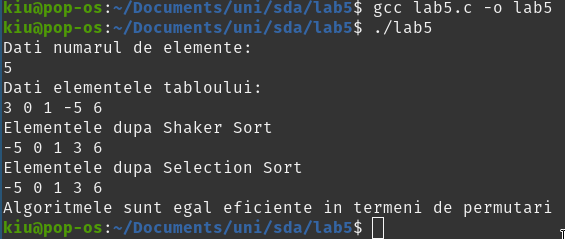
# REALIZAREA PRACTICĂ A SARCINII DE LUCRU

**Sarcina lucrării de laborator (varianta 13):**

Fie dat un tablou unidimensional din n elemente întregi. Să se ordoneze elementele tabloului în ordine ascendentă, utilizând metoda Shaker şi metoda de selecţie directă. Să se compare eficienţa metodelor în funcţie de numărul de comparaţii şi numărul de permutări.

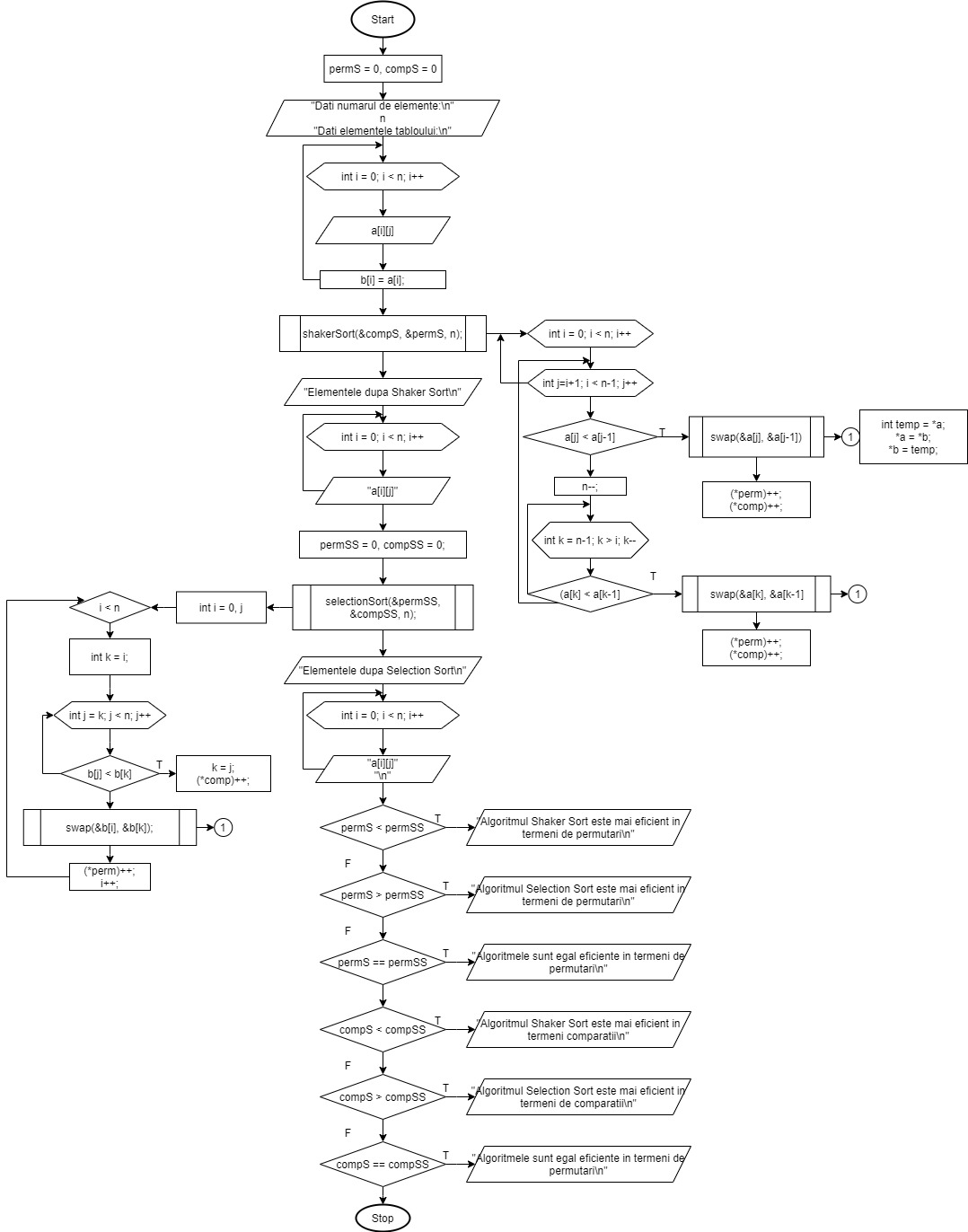
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int a[100], b[100];  void swap(int \*a, int \*b)  {  int temp = \*a;  \*a = \*b;  \*b = temp;  }  void shakerSort(int \*comp, int \*perm, int n)  {  for(int i = 0; i < n; i++)  for(int j=i+1; i < n-1; j++)  {  if(a[j] < a[j-1])  {  swap(&a[j], &a[j-1]);  (\*perm)++;  (\*comp)++;  }  n--;  for(int k = n-1; k > i; k--)  if(a[k] < a[k-1])  {  swap(&a[k], &a[k-1]);  (\*perm)++;  (\*comp)++;  }  }  }  int selectionSort(int \*perm, int \*comp, int n)  {  int i = 0, j;  while (i < n)  {  int k = i;  for(j = k; j < n; j++)  if(b[j] < b[k])  {  k = j;  (\*comp)++;  }  swap(&b[i], &b[k]);  (\*perm)++;  i++;  }  }  int main()  {  int permS = 0, compS = 0; //permutari si comparatii pentru shaker sort  int n;  printf("Dati numarul de elemente:\n");  scanf("%d", &n);  printf("Dati elementele tabloului:\n");  for(int i = 0; i < n; i++)  {  scanf("%d", &a[i]);  b[i] = a[i];  }  shakerSort(&compS, &permS, n);  printf("Elementele dupa Shaker Sort\n");  for(int i = 0; i < n; i++)  printf("%d ", a[i]);  int permSS = 0, compSS = 0;  selectionSort(&permSS, &compSS, n);  printf("\nElementele dupa Selection Sort\n");  for(int i = 0; i < n; i++)  printf("%d ", a[i]);  printf("\n");  if(permS < permSS) printf("Algoritmul Shaker Sort este mai eficient in termeni de permutari\n"); else  if(permS > permSS) printf("Algoritmul Selection Sort este mai eficient in termeni de permutari\n"); else  if(permS == permSS) printf("Algoritmele sunt egal eficiente in termeni de permutari\n");  if(compS < permSS) printf("Algoritmul Shaker Sort este mai eficient in termeni de comparatii\n"); else  if(compS > permSS) printf("Algoritmul Selection Sort este mai eficient in termeni de comparatii\n"); else  if(compS == compSS) printf("Algoritmele sunt egal eficiente in termeni de comparatii\n");  return 0;  } |

Algoritmul selection sort funcționează prin găsirea în mod repetat a elementului minim din porțiunea nesortată a matricei și schimbându-l cu primul element al porțiunii nesortate. Acesta ține evidența numărului de comparații și permutări făcute și le stochează în variabilele corespunzătoare. La sfârșitul funcției, matricea este sortată în ordine crescătoare. Funcția shakerSort folosește bucle pentru a itera peste matrice și compară elementele adiacente, schimbându-le dacă elementul din dreapta este mai mic decât cel din stânga. Numărul de comparații și permutări este crescut de fiecare dată când are loc o schimbare. Primul ciclu for din bucla părinte este pentru creșterea valorii celei mai mari până la cel mai mare indice nesortat, iar doua fază pentru descreșterea valorii celei mai mici la cel mai mic indice nesortat.



**Fig.1.1 Rezultatul executiei**

Figura 1.1 prezintă rezultatul executiei programului



**Fig.1.2 *Schema bloc a programului***

Figura 1.2 prezintă schema bloc a programului

# CONCLUZII

Scopul lucrării date este de a determina eficacitatea a două algoritme de sortare. Astfel, au fost create două tablouri de valori întregi pentru a fi modificate, ulterior, de funcțiile algoritmelor respective. Utilizând două variabile, perm și comp, au fost determinate câte permutații și comparații, respectiv, au fost efectuate la execuția fiecărui algoritm în parte. Am constatat că, în general, metoda Shaker este mai eficientă decât metoda de selecție directă, deoarece necesită mai puține comparații și permutări pentru a ordona un tablou unidimensional. Totuși, este important să se țină cont de faptul că acest lucru poate varia în funcție de specificul fiecărui caz particular.

# BIBLIOGRAFIE

[***https://ocw.cs.pub.ro/courses/sda-ab/laboratoare/03***](https://ocw.cs.pub.ro/courses/sda-ab/laboratoare/03)

[***https://app.diagrams.net/***](https://app.diagrams.net/)