Vasile Denisa-Georgiana
Grupa 204
Informatica ZI
Anul I
Facultatea de Matematica si Informatica

Tema

1. Se da un vector de numere intregi pozitive de dimensiune “dim”. Sa se sorteze elementele palindroame din vectorul dat.

```
#include <iostream>
#include <cmath>
using namespace std;

void Citire_Vector(int vector[], int marime_vector)
{
	for (int index = 0; index < marime_vector; index++)
	{
		cin >> vector[index];
	}
}

bool Verificare_Palindrom(int numar, int dimensiune)
{
	int numar1 = numar / pow(10, dimensiune / 2);
	int numar2 = numar % ((int)pow(10, dimensiune / 2));
	int oglindit_numar1 = 0;
	if (dimensiune % 2 == 1)
		numar1 /= 10;
	while (numar1 != 0)
	{
		oglindit_numar1 = oglindit_numar1 * 10 + numar1 % 10;
		numar1 /= 10;
	}
	if (oglindit_numar1 == numar2)
		return true;
	else
		return false;
}

void Afisare(int vector[], int marime_vector)
{

	for (int index = 0; index < marime_vector; index++)
	{
		cout << vector[index] << " ";
	}
}

void Sortare_vector(int vector[], int marime_vector, int dimensiune)
{
	for (int index1 = 0; index1 < marime_vector - 1; index1++)
	{
```

```
		for (int index2 = index1 + 1; index2 < marime_vector; index2++)
		{
			if (vector[index1] > vector[index2] &&
Verificare_Palindrom(vector[index1], dimensiune) && Verificare_Palindrom(vector[index2],
dimensiune))
			{
				int auxiliar = vector[index1];
				vector[index1] = vector[index2];
				vector[index2] = auxiliar;
			}
		}
	}
	Afisare(vector, marime_vector);
}

int main()
{
	int vector[100], marime_vector, dim;
	cin >> marime_vector >> dim;
	Citire_Vector(vector, marime_vector);
	Sortare_vector(vector, marime_vector, dim);
}
```

2. Se da un vector de numere intregi pozitive de dimensiune “dim”. Sa se sorteze elementele crescator pana la jumatatea sirului, iar de la jumatate la final toate elementele sa se sorteze descrescator.

```
#include <iostream>

using namespace std;

void Citire_Vector(int vector[], int marime_vector)
{
	for (int index = 0; index < marime_vector; index++)
	{
		cin >> vector[index];
	}
}

void Afisare(int vector[], int marime_vector)
{

	for (int index = 0; index < marime_vector; index++)
	{
		cout << vector[index] << " ";
	}
}

void Sortare_vector(int vector[], int marime_vector)
{
	for (int index1 = 0; index1 < marime_vector/2 -1; index1++)
```

```
	{
		for (int index2 = index1 + 1; index2 < marime_vector/2; index2++)
		{
			if (vector[index1] > vector[index2])
			{
				int auxiliar = vector[index1];
				vector[index1] = vector[index2];
				vector[index2] = auxiliar;
			}
		}
	}
	for (int index1 = marime_vector / 2; index1 < marime_vector- 1; index1++)
	{
		for (int index2 = index1 + 1; index2 < marime_vector; index2++)
		{
			if (vector[index1] < vector[index2])
			{
				int auxiliar = vector[index1];
				vector[index1] = vector[index2];
				vector[index2] = auxiliar;
			}
		}
	}
	Afisare(vector, marime_vector);
}

int main()
{
	int dim,vector[100];
	cin >> dim;
	Citire_Vector(vector, dim);
	Sortare_vector(vector, dim);
}
```

3. Sa se testeze daca un vector de numere intregi da teste sortat crescator.

a. Daca da, atunci fiind dat un element “elem”, sa se insereze elementul la locul lui, adica astfel incat sa ramana un sir sortat crescator.

b. Daca nu este sortat crescator, atunci sa se identifice daca exista un eventual element care strica “ordinea”, iar daca exista unul singur, atunci sa se elimine acel element.

```
#include <iostream>

using namespace std;

void Citire_Vector(int vector[], int marime_vector)
{
```

```
	for (int index = 0; index < marime_vector; index++)
	{
		cin >> vector[index];
	}
}

void Afisare(int vector[], int marime_vector)
{

	for (int index = 0; index < marime_vector; index++)
	{
		cout << vector[index] << " ";
	}
}

int Aflare_Pozitie_ELEM(int vector[], int marime_vector, int elem)
{
	for (int index = 0; index < marime_vector-1; index++)
	{
		if (vector[index] <= elem && vector[index + 1] > elem)
			return index + 1;
	}
}

void Inserare_ELEM(int vector[], int marime_vector,int elem)
{
	int pozitie = Aflare_Pozitie_ELEM(vector, marime_vector, elem);
	for (int index = marime_vector; index > pozitie; index--)
	{
		vector[index] = vector[index - 1];
	}
	marime_vector++;
	vector[pozitie] = elem;
	Afisare(vector, marime_vector);
}

void Eliminare_element(int vector[], int& dim, int pozitie)
{
	for (int index = pozitie; index < dim; index++)
	{
		vector[index] = vector[index + 1];
	}
	dim--;
}

int Verificare_Sortare(int vector[], int& marime_vector,int elem)
{
	for (int index1 = 0; index1 < marime_vector-1; index1++)
	{
		for (int index2 = index1 + 1; index2 < marime_vector; index2++)
		{
			if (vector[index1] > vector[index2])
			{
				Eliminare_element(vector, marime_vector, index1);
				return Verificare_Sortare(vector, marime_vector, elem);
			}
		}
	}
```

```
}

int main()
{
	int dim, vector[100],elem;
	cin >> dim>>elem;
	Citire_Vector(vector, dim);
	Verificare_Sortare(vector, dim,elem);
	Inserare_ELEM(vector, dim, elem);
}
```

4. Se citeste un vector de dim elemente dintr-un fisier text, “cautari.txt”. Sa se sorteze crescator vectorul si sa se returneze pozitiile fiecarui element sortat din vectorul initial folosind cautarea secventiala.

Exemplu: *35, 21, 8, 71, 93, 4, 20. Se afiseaza: 6, 3, 7, 2, 1, 4, 5*

Metoda 1(vector de pozitie):

```
#include <iostream>
#include <fstream>

using namespace std;

ifstream fin("cautari.txt");

void Afisare(int vector[], int marime_vector)
{

	for (int index = 1; index <= marime_vector; index++)
	{
		cout << vector[index] << " ";
	}
}

void Interschimbare(int& numar1, int& numar2)
{
	int auxiliar = numar1;
	numar1 = numar2;
	numar2 = auxiliar;
}

void Ordonare_Vector(int vector[], int dim)
{
	int vector_pozitie[100];
	for (int index = 1; index <= dim; index++)
		vector_pozitie[index] = index;
	for (int index1 = 1; index1 < dim; index1++)
	{
		for (int index2 = index1 + 1; index2 <= dim; index2++)
		{
			if (vector[index1] > vector[index2])
```

```
                        {
                                Interschimbare(vector[index1], vector[index2]);
                                Interschimbare(vector_pozitie[index1],
vector_pozitie[index2]);
                        }
                }
        }
        Afisare(vector_pozitie, dim);
}

int main()
{
        int vector[100], element_vector, index = 1;
        while (fin >> element_vector)
        {
                vector[index++] = element_vector;
        }
        int dim = --index;
        fin.close();
        Ordonare_Vector(vector, dim);
}
```

Metoda 2:

```
#include <iostream>
#include <fstream>

using namespace std;

void Cautare_Maxim(int& maxim,int& index_maxim,int vector[],int dim)
{
        maxim = vector[1];
        for (int index = 2; index <= dim; index++)
        {
                if (maxim < vector[index])
                {
                        maxim = vector[index];
                        index_maxim = index;
                }
        }
}

void Cautare_Minim(int vector[], int dim)
{
        int minim, index_minim;
        Cautare_Maxim(minim, index_minim, vector, dim);
        for (int index = 1; index <= dim; index++)
        {
                if (minim > vector[index] && vector[index] != -1)
                {
                        minim = vector[index];
                        index_minim = index;
                }
        }
        vector[index_minim] = -1;
        cout << index_minim << " ";
```

```
}

void Ordonare_Vector(int vector[], int dim)
{
	for (int index = 1; index <= dim; index++)
	{
		Cautare_Minim(vector, dim);
	}
}

int main()
{
	ifstream fin("cautari.txt");

	int vector[100], element_vector, dim = 1;
	while (fin >> element_vector)
	{
		vector[dim++] = element_vector;
	}
	fin.close();
	Ordonare_Vector(vector, --dim);
}
```

Metoda 3(cautare secventiala):

```
#include <iostream>
#include <fstream>

using namespace std;

void Copie_Vector(int* vector, int* copie_vector, int dimensiune)
{
	for (int index = 1; index <= dimensiune; index++)
	{
		copie_vector[index] = vector[index];
	}
}

void Citire_Vector(int* vector, int& dimensiune,int* vector_ordonat)
{
	ifstream fin("cautari.txt");
	int element_vector;
	while (fin >> element_vector)
	{
		vector[dimensiune++] = element_vector;
	}
	dimensiune--;
	fin.close();
	Copie_Vector(vector, vector_ordonat, dimensiune);
}

void Interschimbare(int& numar1, int& numar2)
{
	int auxiliar = numar1;
	numar1 = numar2;
	numar2 = auxiliar;
}
```

```
void Ordonare_Vector(int* vector, int dimensiune)
{
	for (int index = 1; index < dimensiune; index++)
	{
		for (int index2 = index + 1; index2 <= dimensiune; index2++)
		{
			if (vector[index] > vector[index2])
			{
				Interschimbare(vector[index], vector[index2]);
			}
		}
	}
}

void Afisare_Pozitii(int* vector, int* vector_ordonat, int dimensiune)
{
	Ordonare_Vector(vector_ordonat, dimensiune);
	for (int index = 1; index <= dimensiune; index++)
	{
		bool cautare_element = false;
		for (int index2 = 1; index2 <= dimensiune && cautare_element==false;
index2++)
		{
			if (vector_ordonat[index] == vector[index2])
			{
				cautare_element = true;
				cout << index2 << " ";
			}
		}
	}
}

int main()
{
	int vector[100],vector_ordonat[100],  dim = 1;
	Citire_Vector(vector, dim, vector_ordonat);
	Afisare_Pozitii(vector, vector_ordonat, dim);
}
```

5. Se citeste un vector de n numere intregi ordonat descrescator din fisier. Se citeste un al doilea vector de m numere intregi din consola. Folosind cautarea binara sa se determine care elemente din al doilea vector apar in primul si pe ce poziții. Rezultatul se va scrie in fisier.

```
#include <iostream>
#include <fstream>

using namespace std;

void Citire_Vector_din_FISIER(int* vector, int& dimensiune)
{
	ifstream fin("cautari.in");
	int element_vector;
	while (fin >> element_vector)
	{
		vector[dimensiune++] = element_vector;
	}
	dimensiune--;
	fin.close();
}

void Citire_Vector(int vector[], int& dimensiune)
{
	cin >> dimensiune;
	for (int index = 0; index < dimensiune; index++)
	{
		cin >> vector[index];
	}
}


void Interschimbare(int& numar1, int& numar2)
{
	int auxiliar = numar1;
	numar1 = numar2;
	numar2 = auxiliar;
}

int Cautare_Binara(int* vector, int dimensiune, int element_cautat)
{
	int stanga = 0, dreapta = dimensiune - 1, pozitie = -1;
	while (stanga < dreapta && pozitie == -1)
	{
		int mijloc = (stanga + dreapta) / 2;
		if (vector[mijloc] == element_cautat)
		{
			pozitie = mijloc;
		}
		else
		{
			if (vector[mijloc] > element_cautat)
```

```
                        {
                                stanga = ++mijloc;
                        }
                        else
                        {
                                dreapta = --mijloc;
                        }
                }
        }
        if (vector[stanga] == element_cautat)
        {
                pozitie = stanga;
        }
        return pozitie;
}

void Afisare_in_Fisier(int* vector1_din_fisier, int* vector2_din_consola, int
dimensiune1, int dimensiune2)
{
        ofstream fout("cautari.out");

        for (int index = 0; index < dimensiune2; index++)
        {
                int pozitie = Cautare_Binara(vector1_din_fisier, dimensiune2,
vector2_din_consola[index]);
                if (pozitie != -1)
                {
                        fout << "Elementul " << vector2_din_consola[index] << " din al
doilea vector apare si in primul pe pozitia " << pozitie << endl;;
                }
                else
                {
                        fout << "Elementul " << vector2_din_consola[index] << " din al
doilea vector NU apare si in primul vector" << endl;;
                }
        }
        fout.close();
}

int main()
{
        int vector1[100], vector2[100];
        int dim1 = 0, dim2;
        Citire_Vector_din_FISIER(vector1, dim1);
        Citire_Vector(vector2, dim2);
        Afisare_in_Fisier(vector1, vector2, dim1, dim2);
}
```

6. Sa se genereze un sir de numere cu ajutorul funcției rand din stdlib.h. Sa se sorteze crescator toate elementele de pe poziții pare din acest sir de numere.

```
#include <iostream>
#include <stdlib.h>

using namespace std;

void Generare_Sir(int* vector, int& dimensiune)
{
	cin >> dimensiune;
	for (int index = 0; index < dimensiune; index++)
	{
		vector[index] = rand();
	}
}

void Interschimbare(int& numar1, int& numar2)
{
	int auxiliar = numar1;
	numar1 = numar2;
	numar2 = auxiliar;
}

void Sortare(int* vector,int dimensiune)
{
	for (int index1 = 0; index1 < dimensiune-1; index1 += 2)
	{
		for (int index2 = index1 + 2; index2 < dimensiune; index2 += 2)
		{
			if (vector[index1] > vector[index2])
			{
				Interschimbare(vector[index1], vector[index2]);
			}
		}
	}
}

void Afisare_Vector_Sortat(int* vector, int dimensiune)
{
	Sortare(vector, dimensiune);
	for (int index = 0; index < dimensiune; index++)
	{
		cout << vector[index] << " ";
	}
}

int main()
{
	int vector[100],dimensiune;
	Generare_Sir(vector, dimensiune);
	Afisare_Vector_Sortat(vector, dimensiune);
}
```

7. Se citeste o matrice cu n linii si m coloane. Sa se genereze un sir de 100 numere random si sa se afiseze câte dintre aceste elemente sunt pe prima linie a matricei.

Obs: Se va aplica o operație de sortare asupra liniilor matricei astfel incât prima coloana sa fie sortata crescator, ulterior se va apela cautarea binara.

```
#include <iostream>
#include <stdlib.h>

using namespace std;

void Citire_Matrice(int matrice[][100], int& dimensiune_linii, int& dimensiune_coloane)
{
	cin >> dimensiune_linii >> dimensiune_coloane;
	for (int index = 0; index < dimensiune_linii; index++)
	{
		for (int index2 = 0; index2 < dimensiune_coloane; index2++)
		{
			cin >> matrice[index][index2];
		}
	}
}

void Interschimbare_Linii(int matrice[][100], int dimensiune_coloane, int numar_linie1,
int numar_linie2)
{
	for (int index = 0; index < dimensiune_coloane; index++)
	{
		int auxiliar = matrice[numar_linie1][index];
		matrice[numar_linie1][index] = matrice[numar_linie2][index];
		matrice[numar_linie2][index] = auxiliar;
	}
}

void Sortare_dupa_Linii(int matrice[][100], int dimensiune_linii, int dimensiune_coloane)
{
	for (int index1 = 0; index1 < dimensiune_linii - 1; index1++)
	{
		for (int index2 = index1 + 2; index2 < dimensiune_linii; index2++)
		{
			if (matrice[index1][0] > matrice[index2][0])
			{
				Interschimbare_Linii(matrice, dimensiune_coloane, index1,
index2);
			}
		}
	}
}

bool Cautare_Binara(int matrice[][100], int dimensiune, int element_cautat)
{
	int stanga = 1, dreapta = dimensiune, pozitie = -1;
```

```
	while (stanga < dreapta && pozitie == -1)
	{
		int mijloc = (stanga + dreapta) / 2;
		if (matrice[0][mijloc] == element_cautat)
		{
			pozitie = mijloc;
		}
		else
		{
			if (matrice[0][mijloc] < element_cautat)
			{
				stanga = ++mijloc;
			}
			else
			{
				dreapta = --mijloc;
			}
		}
	}
	if (matrice[0][stanga] == element_cautat)
	{
		pozitie = stanga;
	}
	if (pozitie == -1) return false;
	return true;
}

void Generare_Sir_de_Numere(int matrice[][100], int dimensiune_linii, int
dimensiune_coloane)
{
	int Numar_de_Elemente = 0;
	Sortare_dupa_Linii(matrice, dimensiune_linii, dimensiune_coloane);
	for (int index = 0; index < 100; index++)
	{
		int element_random = rand(), verificare = 0;
		verificare = Cautare_Binara(matrice, dimensiune_linii, element_random);
		if (verificare == true)
		{
			cout << element_random << " ";
		}
	}

}

int main()
{
	int matrice[100][100], dimensiune_linii, dimensiune_coloane;
	Citire_Matrice(matrice, dimensiune_linii, dimensiune_coloane);
	Generare_Sir_de_Numere(matrice, dimensiune_linii, dimensiune_coloane);
}
```