RO: Tehnici de sortare.

EN: Sorting techniques.

Obiective:

- Înţelegerea tehnicilor de sortare.
- Familiarizarea cu metodele specifice de implementare a acestor tehnici în limbajul C/C++
- Functii de biblioteca pentru sortare: *qsort( )*

Objectives:

- Understanding the sorting techniques.
- Using the sorting techniques in specific C/C++ implementations
- Library sorting functions: *qsort( )*

Rezumat:

Pentru un set de obiecte $S=\{a_1, a_2, ..., a_n\}$ sortarea constă în rearanjarea obiectelor într-o ordine specifică prin permutarea acestora, rezultând setul $S1=\{a_{k1}, a_{k2}, ..., a_{kn}\}$, astfel că dându-se o funcţie de ordonare *f* să fie îndeplinită relaţia:

$f(a_{k1}) < f(a_{k2}) <...< f(a_{kn})$

Sortarea se face în raport cu o cheie asociată obiectelor.
Eficienţa unei metode de sortare se evaluează prin:
- numărul de comparaţii ale cheii;
- numărul de permutari ale unui obiect.

Aceste operaţii sunt dependente de numărul de elemente din set. O metodă bună de sortare necesită *nlog(n)* comparaţii.
Există metode de sortare:
- simple - efectuează *n*n* comparaţii; sortarea se face "in situ" adică pe loc, tabloul iniţial fiind modificat;
- avansate - permit reducerea numărului de comparaţii până la *nlog(n)*, însă prezintă o complexitate mai mare.

- Biblioteca standard *(stdlib.h)* pune la dispoziţie funcţia:

*void qsort(void *base, size_t nelem, size_t width,*
*int(*fcmp)(const void *, const void *));*

- Funcţia de comparare, definită de utilizator trebuie să returneze pentru ordonare crescatoare :

< 0	dacă	*v1 < *v2
0	dacă	*v1 == *v2
> 0	dacă	*v1 > *v2

Pentru ordonare crescatoare valoarile negativa si pozitiva returnata se vor inversa. Tabloul unidimensional de ordonat poate să conţină şi alte tipuri de date (şiruri de caractere sau structuri).

Exemple:

1. Codul functiilor de cautare si sortare la care este necesara scrierea partii de program principal in care se citeste sirul initial si se afiseaza sirul rezultat.

```
//*****************************************************************************
//Sortarea prin insertie
void sortIns(int *p, int n)
{
int i, j, temp;
        for(i=1; i<n; i++)  {
                temp = p[i];
                for(j=i-1; j>=0; j--) {
                    if(p[j] > temp)
                                                p[j+1] = p[j];
                        else
                                break;
                }//end for
                        p[j+1] = temp;
        }//end for
}//end sortIns
```

```
//*****************************************************************************
//Sortarea prin selectie
void sortSel(int *p, int n)
{
int i, j, pozmin, temp;
        for(i=0; i<n; i++) {
                // cautare pozitie cel mai mic element din sirul curent
                pozmin = i;
                for(j=i+1; j<n; j++) {
                        if(p[pozmin] > p[j])
                                pozmin = j;
                        }//end for
                // interschimbare cu elementul de pe prima pozitie
                temp = p[pozmin];
                p[pozmin] = p[i];
                p[i] = temp;
        }//end for
}//end sortSel

//*****************************************************************************
//Sortarea prin metoda bulelor (fara flag)
void sortBubble(int *p, int n)
{
int i, j, temp;
        for(i=1; i<n; i++) {
                for(j=n-1; j>=i; j--) {
                        if(p[j-1] > p[j]) {
                                temp = p[j];
                                p[j] = p[j-1];
                                p[j-1] = temp;
                        }//end if
                }//end for
        }//end for
}//end sortBubble

//*****************************************************************************
//Sortare prin metoda quickSort
void quickSort(int *p, int prim, int ultim){
int i, j, pivot, temp;
  i = prim;
  j = ultim;
  pivot = p[ultim];
        // partitionare
        do  {
                while(p[i] < pivot)
                        i++;
                while(p[j] > pivot)
                        j--;
                if(i < j) {
                        temp = p[i];
                        p[i] = p[j];
                        p[j] = temp;
                }//end if
                if(i <= j) {
                        j--;
                        i++;
                }//end if
        }while(i < j);//end do-while

     // apel recursiv
        if(prim < j)
```

```
                quickSort(p, prim, j);
        if(i < ultim)
                quickSort(p, i, ultim);
}//end quickSort
```

2. Ordonarea unui sir de valori prin metodele *quick sort* si *bubble sort*. Se afiseaza timpii de lucru specifici executarii diferitelor secvente de cod.

```
#include <time.h>

#include <iostream>
using namespace std;

void init(int numere[ ], int);
void afis(int numere[ ], int);
void bubble(int numere[ ], int);
void quick(int numere[ ], int);
int comparare(const void *arg1, const void *arg2);

int main( ){
        int dim, *numere;
        cout << "\nIntrodu nr. elemente: ";
        cin >> dim;
        numere = new int[dim];
        init(numere, dim);
        //afis(numere, dim);
        bubble(numere, dim);
        //afis(numere, dim);
        init(numere, dim);
        //afis(numere, dim);
        quick(numere, dim);
        //afis(numere, dim);

        delete[ ]numere;
        }

void init(int numere[ ], int dim)
{
        clock_t start, end;
        double dif;

        start = clock( );
        srand((unsigned)time(NULL));

        for (int i = 0; i<dim; i++) {
                numere[i] = rand();
        }

        end = clock( );
        dif = (double)(end - start);
        cout << "\nGenerarea numerelor a durat (clicks) " << dif << " si (seconds) " << ((float)dif) /
CLOCKS_PER_SEC << endl;
}

void afis(int numere[ ], int dim)
{
        clock_t start, end;
        double dif;

        start = clock( );
        for (int i = 0; i<dim; i++)
                cout << " " << numere[i];
```

```
	end = clock( );
	dif = (double)(end - start);
	cout << "\nAfisarea numerelor a durat (clicks) " << dif << "  si (seconds) " << ((float)dif) /
CLOCKS_PER_SEC << endl;
}

void bubble(int numere[ ], int dim)
{
	clock_t start, end;
	double dif;
	int aux, ok;

	start = clock( );
	do {
		ok = 1;
		for (int i = 0; i<dim - 1; i++) {
			if (numere[i]>numere[i + 1]) {
				aux = numere[i];
				numere[i] = numere[i + 1];
				numere[i + 1] = aux;
				ok = 0;
			}
		}
	} while (ok == 0);

	end = clock( );
	dif = (double)(end - start);
	cout << "\nOrdonarea cu \"bubble sort\" a numerelor a durat (clicks) " << dif << " si (seconds) " <<
((float)dif) / CLOCKS_PER_SEC << endl;
}

void quick(int numere[ ], int dim) {

	clock_t start, end;
	double dif;

	start = clock( );
	qsort((int*)numere, dim, sizeof(int), comparare);

	end = clock( );
	dif = (double)(end - start);
	cout << "\nOrdonarea cu \"quick sort\" a numerelor a durat (clicks) " << dif << " si (seconds) " <<
((float)dif) / CLOCKS_PER_SEC << endl;
}

int comparare(const void *arg1, const void *arg2)
{
	if (*(int *)arg1<*(int *)arg2)
		return -1;
	if (*(int *)arg1 == *(int *)arg2)
		return 0;
	if (*(int *)arg1>*(int *)arg2)
		return 1;
}
```

3. Sortare structura folosind qsort( )

```
//Example struct sorted with qsort()
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
struct Datac {
        int an;
        int luna;
        int zi;
};
Struct Pers {
        char numep[12];
        struct Datac datan;
};
int cmp( const struct Pers *a, const struct Pers *b);

int main( )
{
        struct Pers angaj[ ] = {
                {"x3", {1980, 2,28}},
                {"x2", {1960, 5, 5}},
                {"x3", {1960, 1,5}},
                {"x2", {1961, 12, 31}},
                {"x1", {1980, 2, 28}}
        };

        int  i;
        int nang = sizeof(angaj)/sizeof(struct Pers);
        // apel functie de sortare
        qsort((Pers *)angaj, nang, sizeof(angaj[0]), (int (*)(const void*, const void*))cmp);
        printf("Datele sortate :\n");
        for (i = 0; i < nang; i++)
        printf("\t%s, %d, %d, %d\n", angaj[i].numep, angaj[i].datan.an, angaj[i].datan.luna, angaj[i].datan.zi);
    }

int cmp(const struct Pers *a, const struct Pers *b)
{
        if((a->datan).an > (b->datan).an)  return 1;
        else
                if((a->datan).an < (b->datan.an))   return -1;
                else {
                        if((a->datan).luna > (b->datan).luna) return 1;
                        else    if((a->datan).luna < (b->datan).luna) return -1;
                                else  {
                                        if((a->datan).zi > (b->datan).zi)
                                                return 1;
                                        else    if((a->datan).zi < (b->datan).zi)
                                                        return -1;
                                                 else
                                        if((strcmp(a->numep, b->numep) >0)) return 1;
                                else    if((strcmp(a->numep, b->numep) < 0)) return -1;
                                                        return 0;
                                        }
                }
}
```

Teme:

1. Implementaţi metoda bulelor (Bubble-Sort) care foloseşte un indicator flag şi optimizează ciclul interior. Se cere atât scrierea funcţiei, cât şi partea de program care face citirea şi afişarea şirului iniţial şi a celui ordonat.
2. Modificaţi programul care exemplifică metoda de sortare rapidă (Quick-Sort) aşa încât să ordoneze şirul iniţial în ordine descrescătoare.

3. Folosiţi funcţiile de bibliotecă pentru sortări (*qsort( )*) pentru a aranja un şir de înregistrări cu nume, prenume, cod numeric personal, data angajării după două câmpuri la alegere (un exemplu ar fi: crescător după nume şi descrescător după data angajării).
4. Scrieţi o aplicaţie C/C++ in care plecând de la două tablouri (unidimensionale) de numere naturale să se obţină un al treilea tablou care să conţină toate elementele tablourilor sursă fără a se repeta, aranjate în ordine crescătoare.
5. Completati codul problemei date ca exemplu (2) cu alte metode de sortare (sortarea prin selectie, sortarea shell, etc.). Cititi de la tastatura numarul de elemente al sirurilor de valori si apoi trimiteţi-l ca parametru la fiecare functie. Comparati timpii de lucru ai fiecarui algoritm.
6. Cititi de la tastatura *m* elemente de tip intreg intr-un tablou unidimensional si o valoare intreaga *n*<*m*. Impartiti tabloul citit in doua sub-tablori astfel:
   a) primul subtablou va contine primele *n* elemente din tabloul initial
   b) al doilea subtablou va contine restul elementelor din tablul initial.
   Sa se realizeze urmatoarele operatii:
   1. sa se ordoneze crescator cele doua subtablouri
   2. sa se sorteze tabloul initial, prin interclasarea celor doua subtablori ordonate. (merge-sort)
7. Să se scrie un program care permite sortarea unui stoc de calculatoare. Acestea să se reprezinte în program ca o structură formată din caracteristicile calculatoarelor (nume (sir caractere), tip de procesor (sir caractere), frecventa de tact (long int), dimensiunea memoriei RAM (int), preț (float). Sortarea se va face, la alegerea utilizatorului, după: pret, memorie, tact sau tip de procesor. Folosiți, de preferință, funcția de bibliotecă pentru sortări *qsort( )* sau o altă metodă la alegere. Sortați apoi considerând un câmp sir de caractere si unul numeric. Afișați rezultatele.
8. Preluați din două fișiere doua tablouri unidimensionale ce conțin valori reale. Generați un al treilea tablou care să conțină toate valorile din cele doua tablouri și toate valorile obținute prin medierea valorilor de pe aceeași poziție din cele doua tablouri inițiale. Dacă numărul de elemente ale tablourilor diferă, media se va face considerând valoarea 0.0 pentru elementele lipsă. Ordonați al treilea tablou și numărați câte valori neunice sunt în șir.
9. Generati in mod aleatoriu un tablou de maxim 200 valori intregi, valori nu mai mari de 100. Determinati si afisati valoarea minima, mediana si maxima generata, sortand elementele printr-o metoda la alegere. Determinati valoarea medie si comparati aceasta valoare cu cea mediana (afisati diferenta). Verificati daca valoarea medie este in tabloul initial generat.
10. Generati printr-un mecanism aleatoriu un tablou de maxim 200 de valori reale (prin doua tablouri de aceiaşi dimensiune, primul fiind partea intreaga (nu mai mare de 100), al doilea partea fractionara (limitata la 20 ca intreg ce devine .20 fractionar), tabloul real fiind obtinut prin combinarea partii intregi şi fractionare. Afişati tablourile generate, cel real obtinut.
    Sortati folosind functia *qsort( )* tabloul real si afisati rezultatul obtinut.
11. Alocati in mod dinamic un tablou de *n* numere intregi, care vor fi citite si afisate. Cititi o valoare cheie de la intrarea standard. Folosind functia *_lfind( )* cautati si afisati toate pozitiile in care aceasta cheie se gaseste in tabloul citit. Tratati cazul in care sunt valori multiple, sau valoarea nu e in tablou. Folosind functia *qsort( )* sortati apoi acest tablou, pe care il afisati. Cautati folosind functia *bsearch()* aceiasi valoare in tabloul sortat si afisati pozitia ei.

Homework:

1. Implement the Bubble-Sort method using a flag indicator and optimize the inner loop. Write the function that orders an array of integer values read from the keyboard. Display the original and the sorted arrays.
2. Modify the program that implements the Quick-Sort algorithm so that it will sort decreasingly the initial array of values.
3. Use the library function "*qsort( )*" for sorting an array of records that contain a name, a surname, a personal identification code and an employment date. The sorting is based on the data stored in some specific fields (like name, employment date, etc.).
4. Write a C/C++ application that reads from the keyboard 2 arrays of positive numbers. The program determines a 3-rd array that contains all the elements in the initial arrays, increasingly ordered. The elements that have the same value must appear a single time in the ordered array.
5. Add some new sorting methods to the code presented in the examples area (example 2 - selection sort, shell sort, etc.). Read from the keyboard the number of elements from the array and pass it as parameter to each sorting function. Compare the working times scored by each implemented sorting algorithm.
6. Read from the keyboard a string of *m* integers and an integer value *n*<*m*. Split the string in 2 substrings as it follows:
   a) the first substring will contain the first n elements of the initial string
   b) the second substring will contain the rest elements of the initial string

Realize the following operations:
- sort increasingly the substrings
- sort the initial string, by interlacing the sorted substrings (merge-sort)

7. Write a program that sorts a stock of computers, represented in the program as objects created from a structure that stores the computers' characteristics: name (string of characters), processor type(string of characters), frequency (long int), RAM memory (int), price (float). The sorting is performed, as specified by the user, by price, memory amount, frequency, or processor type. Use the *qsort( )* library function or any other sorting technique. Next sort the data using a character string and a numerical field. Display the results.
8. Read from two files two one dimensional arrays composed of real values. Generate a third array that contains all the values from the initial arrays and all the values obtained by calculating the average of the corresponding numbers. If the initial arrays have different numbers of values, the average is calculated between the existent values and 0.0. Order the last array and count the number of non-unique elements.
9. Generate in a random mode maximum 200 smaller than 100 integer numbers and store them into an array. Determine and display the minimum, the median and the maximum value and sort the array in order to accomplish that. Determine the average value and display the difference between it and the calculated median. Check if the average value is part of the initial array.
10. Generate 200 random float values. Store the integer parts (not bigger than 100) into an array. The fractional parts (limited at 20 as an integer value representing a 0.20 fractional part) are stored into another array with the same size as the first one. The initial values re-calculated by combining the elements stored into the previously described arrays are stored into another vector. Display all the arrays. Use the *qsort( )* library function for storing the float values and display the final result.
11. Allocate an array of *n* integer numbers that will be read from the keyboard. Read another value that represents a searching key. Using the *_lfind( )* function, search and display all the positions in which the key appears in the initial array. Handle the cases in which the key is found on more than 1 position or is not found at all. Sort the array using the *qsort( )* function and display the result. Use the *bsearch( )* function for searching the same key in the ordered array and print on the screen its positions.