Ciobanu Rares-Constantin

ATP – SEMINAR 7

1. Să se scrie un subprogram ce conține doi vectori **ordonaţi** și să se obţină un al treilea vector ordonat care va conţine elementele din cei doi vectori in ordine crescătoare(sortare prin interclasare). *Timpul este de ordinul O(n * logn).*

```
#include<stdio.h>
//tablou = vector....folosesc ambii termeni in speranta ca vor fi mai clare
explicatiile.

//x-tabloul care va fi sortat
//p-pozitia de inceput a primului "vector" din tabloul X
//m-pozitia de final a primului "vector" din tabloul X

//m+1-pozitia de inceput a celui de-al doilea "vector" din tabloul X
//u-pozitia de final a celui de-al doilea "vector" din tabloul X
void interclasare(int x[], int p, int m, int u)
{
	int i, j, k, l; //indecsi de parcurgere a tabloului X
	//i - va fi fol. pt primul "vector"; j - va fi fol. pt al 2 lea "vector" ; k-
index folosit la popularea vectorului temporal Z

	//l - un simplu index de parcurgere...folosit la adaugarea elementelor ramase
din X in Z

	int z[100];//vector temporar

	i = p; j = m + 1; k = 0;//aveti explicatii mai sus.

	//in acest while se parcurg in acelasi timp acei 2 "vectori" din X
	while ((i <= m) && (j <= u))
	{
		//Se compara "vectorii" din tabloul X intre ei si se creeaza un nou
tablou Z sortat.
		if (x[i] < x[j])
		{
			z[k] = x[i]; i++; k++;
		}
		else
		{
			z[k] = x[j]; j++; k++;
		}
	}

	//daca raman elemente in plus in "vectori" se adauga in Z....primul vector
poate avea 3 elem. si al 2 lea poate avea 7 elem.=> 4 elemente trebuie completate
	if (i <= m)
		for (l = i; l <= m; l++)
		{
			z[k] = x[l];
			k++;
		}

	if (j <= u)
		for (l = j; l <= u; l++)
		{
			z[k] = x[l];
			k++;
```

```
			}

	//se suprascrie X cu valorile ordonate din Z..astfel X va fi modificat si in
main....
	//la vectori nu trebuie sa puneti "&" in fata ca sa se vada modificarile si in
main....doar la variabilele simple trebuie "&"
	int t = p;//adica t=0;
	for (k = 0; k < (u - p) + 1; k++)// pe exemplul nostru din main:   cat timp
t<(6-0)+1
			x[t++] = z[k];
}

void main() {
	int N = 3, M = 4;
	int x[] = { 1,3,7,2,5,8,9 };//sa presupunem ca vectorul x a fost format din 2
vectori initiali {1,3,7} si {2,5,8,9}

	interclasare(x, 0, N - 1, N + M - 1); //observati cu atentie aceasta
apelare....

	//afisare vector dupa sortare
	for (int i = 0; i < N + M; i++) {
		printf("%d ", x[i]);
	}

	printf("\n");
}
```

2. Să se sorteze un vector dat prin intermediul sortării Merge sort. Aveși nevoie de metoda de la exercițiul 1 pentru rezolvarea acestui exercițiu. *Timpul este de ordinul O(n * logn).*

```
//MERGE SORT = DIVIDE ET IMPERA + interclasare
//x-tabloul ; p - poz de inceput a tabloului X; u - poz de sf. a tabloului X;
//aici tabloul nu mai este ordonat....e un tablou oarecare


//se imparte vectorul in secvente din ce in ce mai mici, astfel incat
//fiecare secventa sa fie ordonata la un moment dat si interclasata
//cu o alta secventa din vector corespunzatoare.
void mergeSort(int x[], int p, int u)
{
	if (p < u)
	{
		int m = (p + u) / 2;//mijlocul tabloului X
		mergeSort(x, p, m); // se tot imparte tabloul X / 2 recursiv
		mergeSort(x, m + 1, u);
		interclasare(x, p, m, u);
	}
}

void main() {
	int N = 3, M = 4;

	int T = N + M;
	int x[] = { 1,31,17,2,5,18,9 };

	mergeSort(x, 0, T-1);//pana la N-1 aici !! pe pozitia x[7] o sa gasiti valori
gunoi (-8932932932)

	//afisare vector dupa sortare
	for (int i = 0; i < T; i++) {
```

```
			printf("%d ", x[i]);
		}

		printf("\n");
}
```

3. Să se sorteze un vector dat prin intermediul sortării rapide(Quick Sort). *Timpul este de ordinul O(n * logn). Timpul este de ordinul O(n^2) – in cel mai nefavorbail caz(când vectorul e sortat deja crescător/descrescător).*

```
#include<stdio.h>

void interschimbaValori(int &a, int &b) {
	int aux = a;
	a  =  b;
	b = aux;
}

/*
   1) aceasta functie alege ca pivot ultimul element din tablou;
   2) se rearanjează elementele în așa fel încât, toate elementele care sunt mai mari decât pivotul merg în partea dreaptă a tabloului. Valorile egale cu pivotul pot sta în orice parte a tabloului. În plus, tabloul poate fi împărțit în părți care nu au aceeași dimensiune (nu sunt egale).

	Functia returneaza indexul unde se va imparti tabloul(se va imparti de la indexul pivotului);
*/
int impartire(int tablou[], int stanga, int dreapta)
{
	// pivot (elementul care va fi mutat la pozitia sa corecta in tabloul sortat)
	// practic ultimul element din tablou va fi mutat undeva....
	int pivot = tablou[dreapta];

	int i = (stanga - 1); // se pastreaza indexul ultimului element mai mic decât pivotul nostru

	for (int j = stanga; j <= dreapta - 1; j++)
	{
		// daca elementul curent este mai mic decat pivotul
		if (tablou[j] < pivot)
		{
			i++;    // se mareste indexul celui mai mic element
			interschimbaValori(tablou[i], tablou[j]);
		}
	}
	//se muta pivotul pe pozitia lui corespunzatoare in tablou;
	interschimbaValori(tablou[i + 1], tablou[dreapta]);

	return i + 1;
}

/*
	tablou  --> tabloul care trebuie sortat;
	stanga  --> Indexul de start;
	dreapta  --> Indexul de final;

	Algoritmul imparte tabloul in bucati(nu neaparat egale) din ce in ce mai mici si dupa care il sorteaza.
*/
void quickSort(int tablou[], int stanga, int dreapta)
```

```
{
	if (stanga < dreapta)
	{
		/*  pi este indexul de impartire
			practic se calculeaza unde se imparte tabloul(divide et impera)
		*/
		int pi = impartire(tablou, stanga, dreapta);

		quickSort(tablou, stanga, pi - 1); // sorteaza elementele pana la
indexul de impartire PI; pentru ca știi deja ca pivotul folosit mai sus e deja pus pe
poziția corespunzătoare in tablou…nu mai trebuie sortat încă o dată și el

		quickSort(tablou, pi + 1, dreapta); // sorteaza elementele dupa indexul
de impartire PI;
	}
}

void afisareTablou(int tablou[], int N)
{
	int i;
	for (i = 0; i < N; i++)
		printf("%d ", tablou[i]);
	printf("\n");
}

void main() {
	int tablou[] = { 10, 7, 8, 9, 1, 5 };
	int n = sizeof(tablou) / sizeof(tablou[0]);//asa se poate afla cate elemente
are un vector

	quickSort(tablou, 0, n - 1);//n-1 pt elementele sunt stocate in vector de la
pozitia 0;
	printf("Vector sortat:\n");
	afisareTablou(tablou, n);
	}
```

4. Să se scrie un subprogram care să sorteze elementele unui vector dat in ordine crescătoare prin metoda bulelor și să se testeze in main pe un vector generat de programator.

```
#include<stdio.h>
//Sortare prin metoda bulelor
//Primim ca parametrii vectorul si numarul de elemente din vector
//Se compara pe rand fiecare doua elemente consecutive, de la prima pana
//la ultima pereche din vector.
//Daca ordinea a doua elemente consecutive nu este cea corecta se face
interschimbarea.
//Daca la parcurgerea vectorului s-a produs cel putin o interschimbare se repeta
procesul.
void sortareBule(int a[], int n) {
	bool gata = false; //variabila cu care verificam daca mai parcurgem inca o data
vectorul sau nu
		int aux;
	while (!gata) {
		gata = true;
		for (int i = 0; i < n - 1; i++)
			if (a[i] > a[i + 1]) {
				aux = a[i];
				a[i] = a[i + 1];
				a[i + 1] = aux;
```

```
                        gata = false; //s-a produs cel putin o interschimbare, deci
marcam faptul ca algoritmul nu este gata si trebuie sa se mai faca cel putin o
parcurgere a vectorului
                }
        }
}
void main() {
        int arr[] = { 10, 7, 8, 9, 1, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        sortareBule(arr, n);
        printf("Array-ul sortat: ");
        for (int i = 0; i < n; i++)
                printf("%d ", arr[i]);
}
```

5. Să se scrie un subprogram care să sorteze elementele unui vector dat in ordine crescătoare prin metoda selecției și să se testeze in main pe un vector generat de programator.

```
//Sortare prin metoda selectiei
//Metoda selectiei presupune parcurgerea vectorului de la primul element pana la
penultimul element.
//Pentru fiecare pozitie i din cadrul parcurgerii se cauta elementul minim din cadrul
pozitiilor ramase(de la i + 1 pana la n).
//Se face interschimbarea intre pozitia curenta i si elementul minim gasit in
pozitiile ramase din vector.
void sortareSelectie(int a[], int n) {
        int aux;
        int poz; //utilizat pentru salvarea pozitiei elementului minim gasit
        for (int i = 0; i < n - 1; i++) { //parcurgem vectorul de la primul element
pana la penultimul
                poz = i;
                //parcurgem restul vectorului de la i + 1 pana la final pentru a gasi
pozitie elementului de valoare minima
                for (int j = i + 1; j < n; j++)
                        if (a[j] < a[poz])
                                poz = j;
                //facem interschimbarea intre pozitia curenta i si pozitia elementului
de valoare minima
                aux = a[poz];
                a[poz] = a[i];
                a[i] = aux;
        }
}

void main() {
        int arr[] = { 10, 7, 8, 9, 1, 5 };
        int n = sizeof(arr) / sizeof(arr[0]);
        sortareSelectie(arr, n);
        printf("Array-ul sortat: ");
        for (int i = 0; i < n; i++)
                printf("%d ", arr[i]);
}
```

6. Să se scrie un subprogram care să sorteze elementele unui vector dat in ordine crescătoare prin metoda numărării și să se testeze in main pe un vector generat de programator.

```
#include<stdio.h>
#include<stdlib.h>
//Sortare prin metoda numararii
```

```
//Prin aceasta metoda se mai construieste un vector (cu aceeasi dimensiune ca si
vectorul original)
//care va contine pozitiile sortate ale elementelor din vectorul original.
//Fie „a” vectorul original si „num” vectorul de pozitii, num[i] va contine pozitia
finala a elementului a[i].
//Pozitia finala a fiecarui element este data de numarul de elemente mai mici decat el
din vector.
//Daca avem doua elemente mai mici decat a[i] atunci acesta se afla pe a 3-a pozitie
//Algoritmul va numara cate numere sunt mai mici decat elementul curent.
void sortareNumarare(int a[], int n) {
	//vectorul num este vectorul care va contine pozitiile
	//vectorul aux este cel folosit pentru pozitionarea elementelor din „a”

	//acesti vectori sunt alocati dinamic pentru a avea dimensiunea vectorului
primit „a” ….sau se putea lucra si fara pointeri declarand vectorii „num” si „aux”:
//int num[100],aux[100]….

	int *num, *aux;
	num = (int*)calloc(n, sizeof(int)); // alocam si initializam cu 0 fiecare
element din vector..calloc!=malloc…remember?
	aux = (int*)malloc(n * sizeof(int)); //e doar un vector temporar…folosit pentru
interschimbarea valorilor

	//calculam numarul de elemente mai mici pentru combinatia pozitiilor i si j
	for (int i = 0; i < n - 1; i++)
		for (int j = i + 1; j < n; j++)
			if (a[j] < a[i])
				num[i]++;
			else
				num[j]++;
	//punem in aux elementele din a la pozitiile specificate in num
	for (int i = 0; i < n; i++)
		aux[num[i]] = a[i];
	//punem in vectorul original elementele pozitionate
	for (int i = 0; i < n; i++)
		a[i] = aux[i];
	//eliberam din HEAP vectorii num si aux
	free(aux);
	free(num);
}
void main() {
	int arr[] = { 10, 7, 8, 9, 1, 5 };
	int n = sizeof(arr) / sizeof(arr[0]);
	sortareNumarare(arr, n);
	printf("Array-ul sortat: ");
	for (int i = 0; i < n; i++)
		printf("%d ", arr[i]);
}
```

7. Să se scrie un subprogram care să sorteze elementele unui vector dat in ordine crescătoare prin metoda interschimbării și să se testeze in main pe un vector generat de programator

```
#include<stdio.h>
//Sortare prin metoda interschimbarii
//Aceasta metoda presupune verificarea a cate doua elemente. Daca ordinea nu este cea
corecta se produce o interschimbare intre aceste doua elemente.
//Aceasta metoda este similara cu metoda selectiei, doar ca la metoda selectiei se
cauta pozitia elementului de valoare minima(pentru sortare crescatoare) si se producea
o singura interschimbare. Aici se va produce cate o interschimbare de fiecare data
cand valorile nu sunt in ordinea corecta.
```

```
void sortareInterschimbare(int a[], int n) {
	int aux;
	for (int i = 0; i < n - 1; i++) //parcurgem pana la penultimul element
		for (int j = i + 1; j < n; j++) //comparam urmatoarele elemente de dupa
pozitia i cu elementul de pe pozitia i
			//se face interschimbare de fiecare data cand gasim un element j
mai mic decat i
			//astfel aducem elementul minim la finalul iteratiilor
			if (a[i] > a[j]) {
				aux = a[i];
				a[i] = a[j];
				a[j] = aux;
			}
}
void main() {
	int arr[] = { 10, 7, 8, 9, 1, 5 };
	int n = sizeof(arr) / sizeof(arr[0]);
	sortareInterschimbare(arr, n);
	printf("Array-ul sortat: ");
	for (int i = 0; i < n; i++)
		printf("%d ", arr[i]);
}
```

Algoritm de sortare	Cel mai bun caz	Cazul obișnuit	Cel mai rău caz	Pe loc
Bule	$O(n)$	$O(n^2)$	$O(n^2)$	✓
Selectie	$O(n^2)$	$O(n^2)$	$O(n^2)$	✓
Inserare	$O(n)$	$O(n^2)$	$O(n^2)$	✓
Shell	$O(n)$	$\geq O(n \cdot \log(n))$	$O(n^2)$	✓
Interclasare	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	-
Heap	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	✓
Quick	$O(n \cdot \log(n))$	$O(n \cdot \log(n))$	$O(n^2)$	✓
Prin numărare	$O(n)$	$O(n)$	$O(n)$	-
Bucket	$O(n+k)$	$O(n+k)$	$O(n^2)$	-
Radix	$O(d \cdot n)$	$O(d \cdot n)$	$O(d \cdot n)$	-