Ministerul Educaţiei al Republicii Moldova

Universitatea Tehnică a Moldovei

Catedra Informatica Aplicată

**RAPORT**

Lucrarea de laborator nr.6

*la Structuri de date si algoritmi*

A efectuat:

st. gr. C-171 D.Melniciuc

A verificat:

dr., conf.univ. M. Kulev

Chişinău 2018

**Lucrarea de laborator nr.6**

**Tema:** Analiza emperică a algoritmilor de sortare și de căutare

**Scopul lucrării:** Obținerea deprinderilor de analiză emperică a algoritmilor

**Sarcina de lucru:** De scris un program in limbajul C pentru analiza emperică a algoritmului propus (după variantă) care afișarea meniului cu următoarele obțiuni:

1. Tabloul demonstrativ de n elemente (n<=20)

2. Tablou cu valori aleatoare pentru trei numere n1, n2, n3 de elemente mari (10000<n1<n2=2\*n1<n3=2\*n2)

3. Tablou sortat crescător

4. Tablou sortat invers (descrescător)

5. Analiza algoritmului funcției standard de sortare **qsort()** sau funcției standard de căutare **bsearch()**

Analiza emperică a algoritmului constă în: determinarea timpului de rulare, numărului de compărări, numărului de interschimbări (muțări), compărarea rezultatelor obținute și aranjate în

tabela și formularea concluziilor.

**Varianta 10: Insertion sort**

Sortarea prin insertie

Metoda

Acest tip de algoritm simplu de sortare se comporta eficient in lucrul cu tablouri de date de dimensiuni mici sau in cadrul tablourilor sortate partial, el fiind adesea intalnit ca facand parte din alti algoritmi de sortare mai sofisticati. El procedeaza prin aducerea termenilor (elementelor) din tabloul unul cate unul si inserarea lor in pozitia corecta intr-un nou tablou. Sortarea prin insertie lucreaza astfel: insereaza fiecare element sortat din tabloul de intrare (initial) intr-un tablou iesire (rezultat) la locul potrivit. Simplitatea algoritmului consta prin necesitatea de a avea doar doua tablouri structurate: tabloul de intrare (nesortat) si tabloul de iesire in care termenii care se sorteaza sunt inserati. Pentru a prevenii un numar prea mare de accese la memorie, cele mai multe implementari folosesc metoda sortarii interioare ("pe loc") care functioneaza prin mutarea termenului curent in partea termenilor sortati si secventa se tot repeta pana cand termenului i s-a gasit valoarea de loc. Sortarea prin insertie este mult mai eficienta decat sortare prin interschimbare (bubble sort), desi trebuie sa se tine cont de anumite proprietati.

Complexitatea

Este un algoritm de sortare simplu, liniar: O(N) pentru tabele care contin N elemente aproape sortate. Timpul de executie al algoritmului depinde de numãrul inversãrilor, deci de ordinea initialã al elementelor. În caz general sunt necesare N 2 /4 comparatii si interschimbãri, N 2 /8 deci ordinea magnitudinii este O(N 2). În caz cel mai nefavorabil, când tabela este initial sortatã în ordine inverse sortarea este de O(N2). Dacã tabela A contine înregistrãri mari atunci e preferabil folosirea unei "tabele de index": P, accesând tabela originalã doar pentru comparare, astfel costul este mai putin. Altã modalitate de îmbunãtãtire constã în folosirea unei matrici de pointeri pentru înregistrãri.

***Codul programului***

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <time.h>*

*long insertSort(int \*a,int n)*

*{*

*int i, k, t;*

*long nr\_i = 0, zx;*

*for(i=1; i<n; i++)*

*{*

*zx=a[i];*

*for(k=i-1; k>=0 && zx<a[k]; k--)*

*{*

*a[k+1] = a[k];*

*a[k+1] = zx;*

*nr\_i++;*

*}*

*}*

*return nr\_i;*

*}*

*int main()*

*{*

*int size, m = 1, comand;*

*float timp;*

*int d = 0;*

*int \*arr = NULL;*

*start:*

*while(m)*

*{ m = 0;*

*system("cls");fflush(stdin);*

*printf("Introduceti numarul de elemente: ");*

*scanf("%d", &size);*

*arr = (int\*) malloc (size \* sizeof(int));*

*if(!arr)*

*{*

*printf("\n[-] Memoria nu a fost alocata\n\n");*

*m = 1;*

*}*

*}*

*for(;;*

*){*

*system("cls"); fflush(stdin);*

*printf("[1] Tablou deja sortat\n");*

*printf("[2] Tablou random\n");*

*printf("[3] Tablou sortat invers\n\n");*

*printf("[0] Inapoi?\n\n");*

*printf("[9] Exit\n\n");*

*printf(">> ");*

*scanf("%d",&comand); system("cls");*

*switch(comand)*

*{*

*case 0 : break;*

*case 9 : return 0;*

*case 1 :*

*for(int i=0; i<size; i++)*

*arr[i] = i;*

*timp = clock();*

*printf("Nr de schimbari: %d\n", insertSort(arr, size));*

*printf("Timpul: %f secunde \n", (clock()-timp)/CLOCKS\_PER\_SEC);*

*system("pause");*

*break;*

*case 2 :*

*srand(time(NULL));*

*for(int i=0; i<size; i++)*

*arr[i] = rand() % 100;*

*timp = clock();*

*printf("Nr de schimbari: %d\n", insertSort(arr, size));*

*printf("Timpul: %f secunde \n", (clock()-timp)/CLOCKS\_PER\_SEC);*

*system("pause");*

*break;*

*case 3 :*

*for(int i=0; i<size; i++)*

*arr[i] = size - i;*

*timp = clock();*

*printf("Nr de schimbari: %d\n", insertSort(arr,size));*

*printf("Timpul: %f secunde \n", (clock()-timp)/CLOCKS\_PER\_SEC);*

*system("pause"); break;*

*default : printf("\n[-] Comanda gresita\n"); break;*

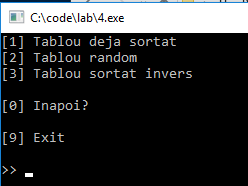
*}*

*}*

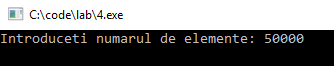
*return 0;*

*}*

**Rezultatele obtinute**

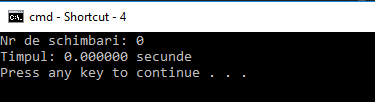
**Meniu**

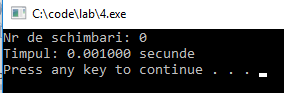
***Introducerea***

***1500 elemente 50.000 elemente***

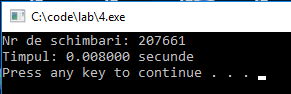
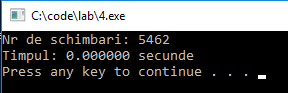


**Sortarea unui tablou deja sortat**

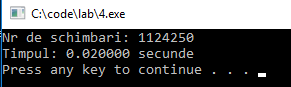
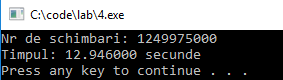


****

**Sortarea unui tablou cu elemente random**



**Sortarea unui tablou invers**



**Concluzii**: In urma acestei lucrari de laborator am ajuns la ideia ca insertion sort este un algoritm de sortare bun, insa nui asa de rapid ca qsort. Pentru sirurile deja aranjate se face 0 (zero) mutari, ceia ce ii un rezultat bun pentru viteza efectuarii. Inca un plus pentru acest algoritm este ca nu necesita memorie adaugatoare fata de qsort.