Ministerul Educaţiei, Culturii și Cercetării

al Republicii Moldova   
  
Universitatea Tehnică a Moldovei   
  
  
  
  
  
  
  
RAPORT

Lucrarea de laborator nr. 5   
la Structuri de date si algoritmi

A efectuat: st. gr. TI-211 Popa Cătălin   
  
A verificat: Stratulat Ștefan

UTM, Chișinău 2021

**Tema:**

Implementarea și analiza algoritmilor de sortare și de căutare în limbajul C.

**Scopul lucrării**

Scopul lucrării este de a familiariza studentul cu implementarea și analiza algoritmilor, utilizînd pentru aceasta limbajul C.

**Problema**

1. **Să se elaboreze un program ce va aloca dinamic un tablou unidimensional de numere întregi și va implementa următoarele funcții, funcțiile vor fi organizate sub forma unui meniu:**

* inițializarea tabloului cu numere aleatorii;
* afișarea tabloului;
* eliberarea memoriei tabloului;
* sortarea tabloului utilizînd una din metode ( selection sort, insertion sort, bubble sort ), se va afișa în cît timp a fost executată sortarea;
* sortarea tabloului utilizînd una din metode ( merge sort, quick sort, heap sort ), se va afișa în cît timp a fost executată sortarea.

1. **1 Să se realizeze o analiză empirică a timpului necesar pentru a sorta un tablou aleator de lungimea N unde N va avea valori [100, 1000, 10000, 100000], rezultatele obținute se vor plasa în tabel.**

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <time.h>

void red () {

printf("\033[0;31m");

}

void green () {

printf("\033[0;32m");

}

////////////////////////////////////////////////////////////////////////

int \*create\_array(int n)

{

srand(time(NULL));

int \*array = malloc(n \* sizeof(int));

for(int i=0; i<n; i++)

{

array[i] = rand() % n + 1;

}

return array;

}

////////////////////////////////////////////////////////////////////////

void print\_array(int \*array, int n)

{

{

for(int i=0; i<n; i++)

printf("arr[%d]: |%d|\n",i,\*(array+i));

}

}

////////////////////////////////////////////////////////////////////////

void free\_array(int \*array)

{

free(array);

}

////////////////////////////////////////////////////////////////////////

void insertionSort(int \*array, int n)

{

int key, j;

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

{

key = array[i];

j = i - 1;

while (j >= 0 && array[j] > key)

{

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = key;

}

}

////////////////////////////////////////////////////////////////////////

void quick\_sort(int \*array, int n)

{

if (n < 2)

{

return;

}

int mid = array[n / 2];

int \*left = array;

int \*right = array + n - 1;

while (left <= right) {

if (\*left < mid) {

left++;

}

else if (\*right > mid) {

right--;

}

else {

int t = \*left;

\*left = \*right;

\*right = t;

left++;

right--;

}

}

quick\_sort(array, right - array + 1);

quick\_sort(left, array + n - left);

}

////////////////////////////////////////////////////////////////////////

int menu()

{

int error\_input\_found = 0;

menu\_start:

system("cls");

printf("\033[0;33m Laborator: \033[0;31m nr.5\n");

printf("\033[0;33m Grupa: \033[0;31m TI-211 \n");

printf("\033[0;33m A efectuat: \033[0;31m Popa Catalin \n");

printf("\033[0;33m A verificat:\033[0;31m Stratulat Stefan\n\n\n\n");

green();

printf("|----------------------//Menu\\--------------------------|\n");

printf("|1.Initializeaza tabloul |\n");

printf("|2.Afisarea tabloului |\n");

printf("|3.Eliberarea memoriei tabloului |\n");

printf("|4.Sortarea tabloului - insertion sort |\n");

printf("|5.Sortarea tabloului - quick sort |\n");

printf("|0.Inchide programul |\n");

printf("|-------------------------------------------------------|\n");

int command = 10;

if(error\_input\_found)

{

printf("\n\nCommanda trebuie sa fie o optiune de mai sus!!![0-5]\n");

}

red();

printf("\n\nSelectati commanda: ");

scanf("%d",&command);

if((command>5)||(command<0))

{

error\_input\_found = 1;

goto menu\_start;

}

return command;

}

//////////////////////////////////////////////////////////////////

int main()

{

int n,i;

int \*ptr=0;

int \*array;

int command=10;

while(command)

{

command = menu();

if(command == 1)

{

printf("Dati numarul de elemente:\n");

scanf("%d",&n);

ptr = create\_array(n);

}

switch (command)

{

case 2:

{

if(ptr == NULL)

{

printf("Tabloul este null!!!\n");

}

else

{

print\_array(ptr, n);

}

break;

}

case 3:

{

free(ptr);

printf("Memoria a fost eliberata cu succes!\n");

ptr = NULL;

break;

}

case 4:

{

clock\_t start = clock();

insertionSort(ptr, n);

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f", time\_t);

break;

}

case 5:

{

clock\_t start = clock();

quick\_sort(ptr, n);

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f", time\_t);

break;

}

}

printf("\n\nTastati orice buton pentru continuare!\n");

\_getch();

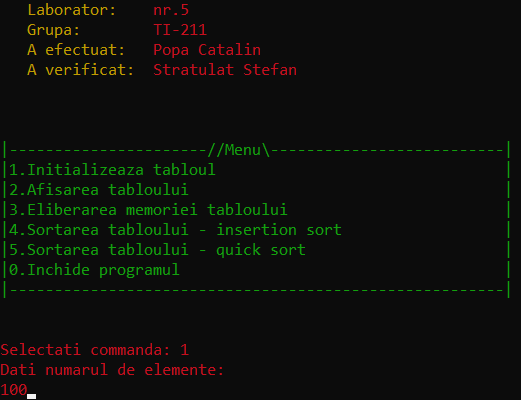
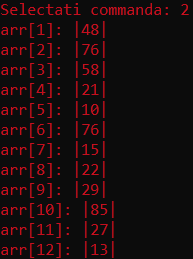
}

return 0;

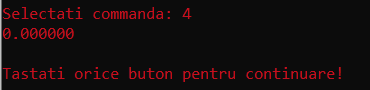
}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **100** | **1000** | **10000** | **100000** |
| **Insertion sort** | 0.000000 | 0.000000 | 0.062 | 6.056 |
| **Quick sort** | 0.000000 | 0.000000 | 0.000000 | 0.019 |

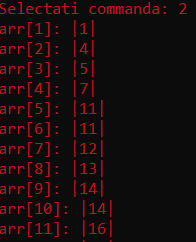
**Initializarea Afisarea**

**Sortarea Eliberarea memoriei**

**Afisarea dupa sortare**



1. **Să se elaboreze un program ce va aloca dinamic un tablou unidimensional de numere întregi și va implementa următoarele funcții, funcțiile vor fi organizate sub forma unui meniu:**

* inițializarea tabloului cu numere aleatorii;
* afișarea tabloului;
* eliberarea memoriei tabloului;
* căutarea unui număr din tablou utilizînd metoda de căutare linear search, se va afișa în cît timp se execută căutarea.
* căutarea unui număr din tablou utilizînd metoda de căutare binary search, se va afișa în cît timp se execută căutarea.

**2.2 Să se realizeze o analiză empirică a timpului necesar pentru a efectua 10000 de căutări într-un tablou aleator de lungimea N unde N va avea valori [100, 1000, 10000, 100000], rezultatele obținute se vor plasa în tabel.**

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <time.h>

void red () {

printf("\033[0;31m");

}

void green () {

printf("\033[0;32m");

}

////////////////////////////////////////////////////////////////////////

int \*create\_array(int n)

{

srand(time(NULL));

int \*array = malloc(n \* sizeof(int));

for(int i=0; i<n; i++)

{

array[i] = rand()% n + 1;

}

// array=(int\*)realloc(array, n\*sizeof(int));

return array;

}

////////////////////////////////////////////////////////////////////////

// int \*create\_array(int n)

// {

// srand(time(NULL));

// int \*array = malloc(n \* sizeof(int));

// int i, p, count;

// int temp = 0;

//

// for(count=0;count<n;count++)

// {

// array[count]=rand()% n+1;

// }

//

// while(temp < n)

// {

// int r = rand()% n + 1;

//

// for(i = 0; i < temp; i++)

// {

// if(array[i] == r)

// {

// break;

// }

// }

// if(i == temp)

// {

// array[temp++] = r;

// }

// }

//

// array=(int\*)realloc(array, n\*sizeof(int));

// return array;

// }

////////////////////////////////////////////////////////////////////////

void print\_array(int \*array, int n)

{

{

for(int i=0; i<n; i++)

printf("arr[%d]: |%d|\n",i+1,\*(array+i));

}

}

////////////////////////////////////////////////////////////////////////

void free\_array(int \*array)

{

free(array);

}

////////////////////////////////////////////////////////////////////////

int linear\_search(int \*array, int n, int values)

{

for (int i = 0; i < n; i++)

{

if (array[i] == values)

return i+1;

}

return -1;

}

////////////////////////////////////////////////////////////////////////

void quick\_sort(int \*array, int n)

{

if (n < 2)

{

return;

}

int mid = array[n / 2];

int \*left = array;

int \*right = array + n - 1;

while (left <= right) {

if (\*left < mid) {

left++;

}

else if (\*right > mid) {

right--;

}

else {

int t = \*left;

\*left = \*right;

\*right = t;

left++;

right--;

}

}

quick\_sort(array, right - array + 1);

quick\_sort(left, array + n - left);

}

////////////////////////////////////////////////////////////////////////

int binary\_search(int \*array,int variable, int start, int end)

{

while(start <= end)

{

int mid = start + (end - start) / 2;

if(array[mid] == variable)

{

return mid;

}

if(array[mid] < variable)

{

start = mid + 1;

}

else

{

end = mid - 1;

}

}

return -1;

}

////////////////////////////////////////////////////////////////////////

int menu()

{

int error\_input\_found = 0;

menu\_start:

system("cls");

printf("\n\n");

printf("\033[0;33m Laborator: \033[0;31m nr.5\n");

printf("\033[0;33m Grupa: \033[0;31m TI-211 \n");

printf("\033[0;33m A efectuat: \033[0;31m Popa Catalin \n");

printf("\033[0;33m A verificat:\033[0;31m Stratulat Stefan\n\n\n\n");

green();

printf("|----------------------//Menu\\--------------------------|\n");

printf("|1.Initializeaza tabloul |\n");

printf("|2.Afisarea tabloului |\n");

printf("|3.Eliberarea memoriei tabloului |\n");

printf("|4.Cautare dupa linear search |\n");

printf("|5.Cautare dupa binary search |\n");

printf("|0.Inchide programul |\n");

printf("|-------------------------------------------------------|\n");

int command = 10;

if(error\_input\_found)

{

printf("\n\nCommanda trebuie sa fie o optiune de mai sus!!![0-5]\n");

}

red();

printf("\n\nSelectati commanda: ");

scanf("%d",&command);

if((command>5)||(command<0))

{

error\_input\_found = 1;

goto menu\_start;

}

return command;

}

//////////////////////////////////////////////////////////////////

int main()

{

int n,i;

int \*ptr;

int command=100;

while(command)

{

command = menu();

if(command == 1)

{

printf("Dati numarul de elemente:\n");

scanf("%d",&n);

ptr = create\_array(n);

// ptr = array;

}

switch (command)

{

case 2:

{

if(ptr == NULL)

{

printf("Tabloul este null!!!\n");

}

else

{

print\_array(ptr, n);

}

break;

}

case 3:

{

free(ptr);

printf("\nMemoria a fost eliberata cu succes!\n");

ptr = NULL;

break;

}

case 4:

{

int variable,res;

printf("Introduceti variabila cautata: \n");

scanf("%d",&variable);

clock\_t start = clock();

res = linear\_search(ptr, n, variable);

if(res == -1)

{

printf("Elementul nu este prezent\n");

}

else

{

printf("Elementul se afla pe pozitia: |%d|\n", res);

}

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f", time\_t);

break;

}

case 5:

{

int variable,res;

printf("Introduceti variabila cautata: \n");

scanf("%d",&variable);

clock\_t start = clock();

quick\_sort(ptr, n);

res = binary\_search(ptr, variable, 0, n );

if(res != -1)

{

printf("Elementul se afla pe pozitia: |%d|\n", res+1);

}

else

{

printf("Elementul nu este prezent\n");

}

clock\_t end = clock();

float time\_t = (float)(end-start)/CLOCKS\_PER\_SEC;

printf("%f", time\_t);

break;

}

}

red();

printf("\n\nTastati orice buton pentru continuare!\n");

\_getch();

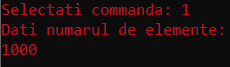
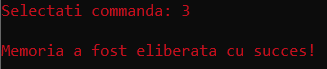
}

return 0;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **100** | **1000** | **10000** | **100000** |
| **Linear search** | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **Binary search** | 0.000000 | 0.000000 | 0.003 | 0.023 |

**Initializare Eliberarea memoriei**

**Binary search Linear search**

