**Lucrare de laborator**

**la disciplina**

**Structuri de Date și Algoritmi**

**Lucrare de laborator nr. 5:**

**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. **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 tabelul de mai jos:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **100** | **1000** | **10000** | **100000** |
| **Metoda de sortare 1 ( ex. selection sort)** | **0.000000 s** | **0.000000 s** | **0.068000 s** | **6.701000 s** |
| **Metoda de sortare 2 ( ex. heap sort)** | **0.000000 s** | **0.000000 s** | **0.000000 s** | **0.029000 s** |

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 tabelul de mai jos:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **100** | **1000** | **10000** | **100000** |
| **Linear search** | **0.000000 s** | **0.000000 s** | **0.000000 s** | **0.004000 s** |
| **Binary search** | **0.000000 s** | **0.000000 s** | **0.000000 s** | **0.005000 s** |

**Rezultatul**

Rezultaul lucrării se va plasa într-un raport, unde va fi inclusă foaia de titlu și codul sursă a programului cu o poză de execuție a acestuia și tabelele cu analiza empirică efectuată.

Code :

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <time.h>**

**#include <conio.h>**

**int** \***create\_init\_array**(**int** size)

{

srand(time(NULL));

**int** i;

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

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

{

arr[i] = rand() % size + 1;

}

**return** arr;

}

**void** **print\_array**(**int** \*arr, **int** size)

{

**int** i;

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

{

printf("|%d\t",arr[i]);

}

}

**void** **insertion\_sort**(**int** \*arr,**int** size)

{

**int** i,j;

**int** curr, pred;

printf("[Insertion\_Sort]\nFor %d elements-\n",size);

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

{

curr = arr[i];

pred = i - 1;

**while** (curr < arr[pred] && pred >= 0) {

arr[pred + 1] = arr[pred];

--pred;

}

arr[pred + 1] = curr;

}

}

**void** **swap**(**int** \*a, **int** \*b)

{

**int** temp = \*b;

\*b = \*a;

\*a = temp;

}

**void** **heapify**(**int** \*arr, **int** n, **int** i) {

*// Find largest among root, left child and right child*

**int** largest = i;

**int** left = 2 \* i + 1;

**int** right = 2 \* i + 2;

**if** (left < n && arr[left] > arr[largest])

largest = left;

**if** (right < n && arr[right] > arr[largest])

largest = right;

*// Swap and continue heapifying if root is not largest*

**if** (largest != i) {

swap(&arr[i], &arr[largest]);

heapify(arr, n, largest);

}

}

*// Main function to do heap sort*

**void** **heap\_sort**(**int** \*arr,**int** size)

{

printf("[Heap\_Sort]\nFor %d elements-\n",size);

**for** (**int** i = size / 2 - 1; i >= 0; i--)

{

heapify(arr, size, i);

}

*// Heap sort*

**for** (**int** i = size - 1; i >= 0; i--)

{

swap(&arr[0], &arr[i]);

*// Heapify root element to get highest element at root again*

heapify(arr, i, 0);

}

}

**void** **linear\_search**(**int** \*arr, **int** size, **int** key) {

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

**int** counter=0;

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

{

**if** (arr[i] == key)

{

result[counter] = i+1;

counter++;

}

}

printf("Pozitiile pe care se afla numarul [%d] sunt : ",key);

print\_array(result,counter);

printf("\n");

free(result);

result = NULL;

}

**void** **binary\_search**(**int** \*arr, **int** start, **int** end, **int** key)

{

**int** \* result = malloc(end\***sizeof**(**int**));

**int** counter = 0;

**while** (start <= end){

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

**if** (arr[mid] == key)

{

result[counter] = mid+1;

counter++;

}

**if** (arr[mid] < key)

start = mid + 1;

**else**

end = mid - 1;

}

printf("Pozitiile pe care se afla numarul [%d] sunt : ",key);

print\_array(result,counter);

printf("\n");

free(result);

result = NULL;

}

**int** **menu**()

{

**int** error\_input\_found = 0;

menu\_start:

system("cls");

printf("%+206s\n","| Menu |\n");

printf("%+205s\n","| ------------------------------------ |");

printf("%+205s\n","| 1.init\_array |");

printf("%+205s\n","| 2.print\_array |");

printf("%+205s\n","| 3.free\_arr |");

printf("%+205s\n","| 4.insertion\_sort |");

printf("%+205s\n","| 5.heap\_sort |");

printf("%+205s\n","| 6.linear\_search |");

printf("%+205s\n","| 7.binary\_search |");

printf("%+205s\n","| 0.exit |");

**int** command = 5;

**if**(error\_input\_found)

{

printf("\n\nVa rog sa selectati o comanda existenta.\n");

}

printf("\n\nCommand : ");

scanf("%d",&command);

**while**(command>7 || command<0)

{

printf("Command does not exist .\n");

error\_input\_found = 1;

scanf("%\*[^\n]");

**goto** menu\_start;

}

**return** command;

}

**int** **main**()

{

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

printf("\n\n\n");

**int** size,key;

**int** command = 7;

**int** \*array = NULL;

**int** choose[4] = {100,1000,10000,100000};

**while**(command)

{

command = menu();

**if**(command == 1)

{

**if**(array != NULL)

{

free(array);

}

choose\_again:

printf("\nChoose : \n1) 100\n2) 1000\n3) 10000\n4) 100000\n\n");

printf("Size[1,2,3,4]: ");

scanf("%d",&size);

**for**(**int** i=1;i<=4;i++)

{

**if**(i == size)

{

array = create\_init\_array(choose[i-1]);

**break**;

}

}

**if**(size>4 || size<0)

{

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

**goto** choose\_again;

}

}

**else** **if**(array != NULL)

{

**switch**(command)

{

**case** 2:

{

print\_array(array,choose[size-1]);

**break**;

}

**case** 3:

{

**if**(array!=NULL)

{

free(array);

printf("Tabloul a fost eliberat.\n\n");

array = NULL;

}

**break**;

}

**case** 4:

{

**clock\_t** begin = clock();

insertion\_sort(array,choose[size-1]);

**clock\_t** end = clock();

**float** time\_spent = (**float**)(end - begin) / CLOCKS\_PER\_SEC;

printf("The elapsed time is %f seconds\n", time\_spent);

**break**;

}

**case** 5:

{

**clock\_t** begin = clock();

heap\_sort(array,choose[size-1]);

**clock\_t** end = clock();

**float** time\_spent = (**float**)(end - begin) / CLOCKS\_PER\_SEC;

printf("The elapsed time is %f seconds\n", time\_spent);

**break**;

}

**case** 6:

{

printf("Introduceti numerul cautat : ");

scanf("%d",&key);

**clock\_t** begin = clock();

linear\_search(array,choose[size-1],key);

**clock\_t** end = clock();

**float** time\_spent = (**float**)(end - begin) / CLOCKS\_PER\_SEC;

printf("The elapsed time is %f seconds\n", time\_spent);

**break**;

}

**case** 7:

{

printf("Introduceti numerul cautat : ");

scanf("%d",&key);

**clock\_t** begin = clock();

binary\_search(array,0,choose[size-1],key);

**clock\_t** end = clock();

**float** time\_spent = (**float**)(end - begin) / CLOCKS\_PER\_SEC;

printf("The elapsed time is %f seconds\n", time\_spent);

**break**;

}

**case** 0:

{

**return** 0;

}

}

} **else**

{

printf("\nArray is NULL! Init please! \n");

}

printf("\n\nPress any Key to continue!\n");

\_getch();

}

**return** 0;

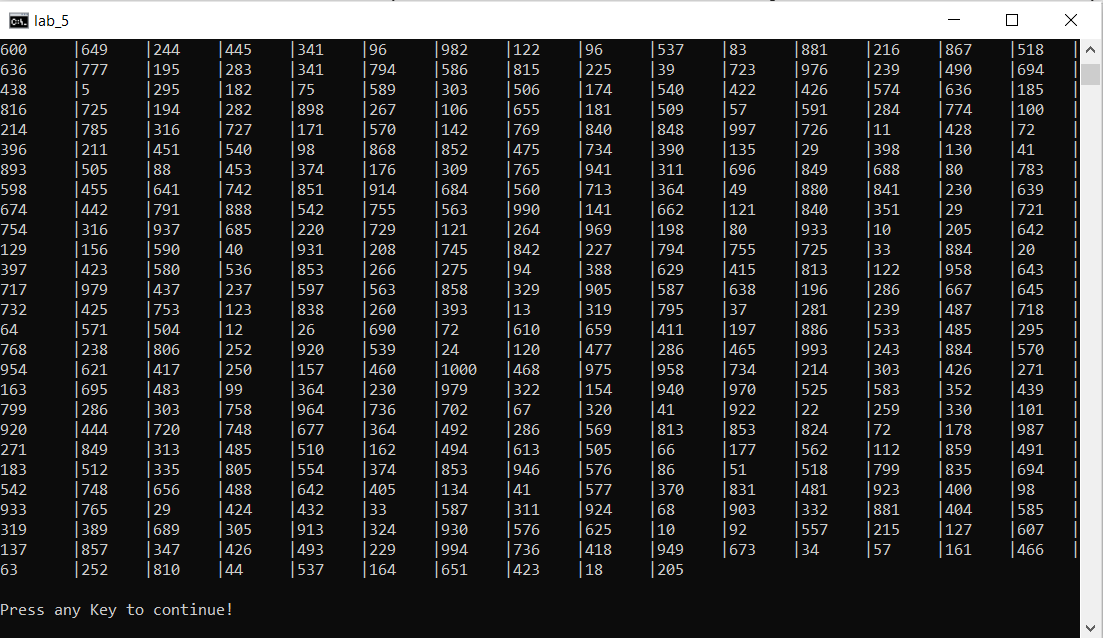
}

Rezultat :

Initializarea :



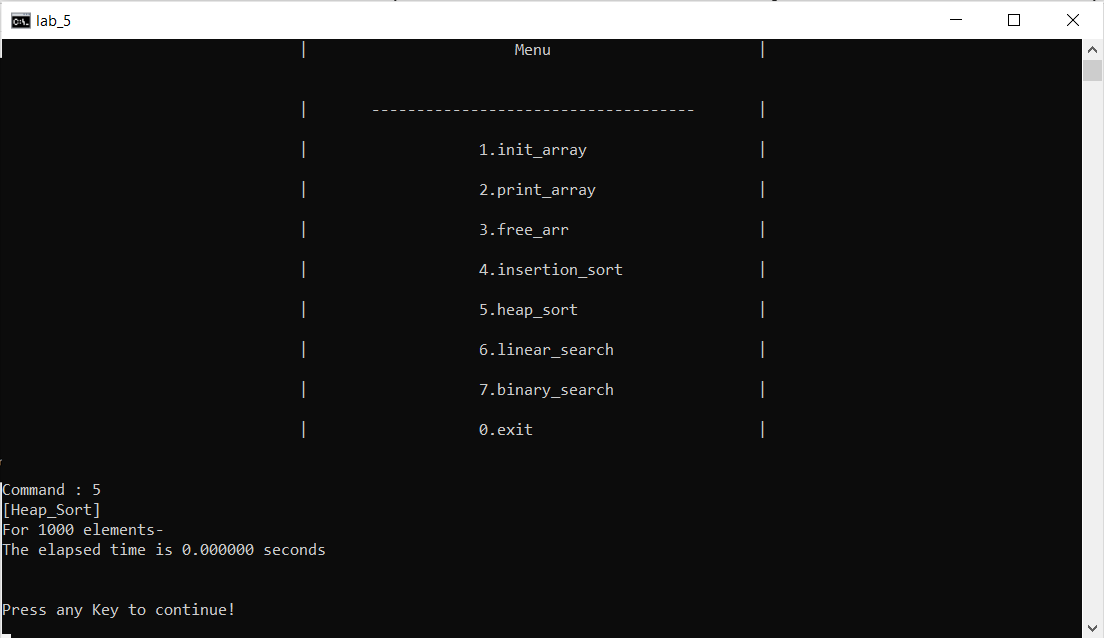
Printarea :



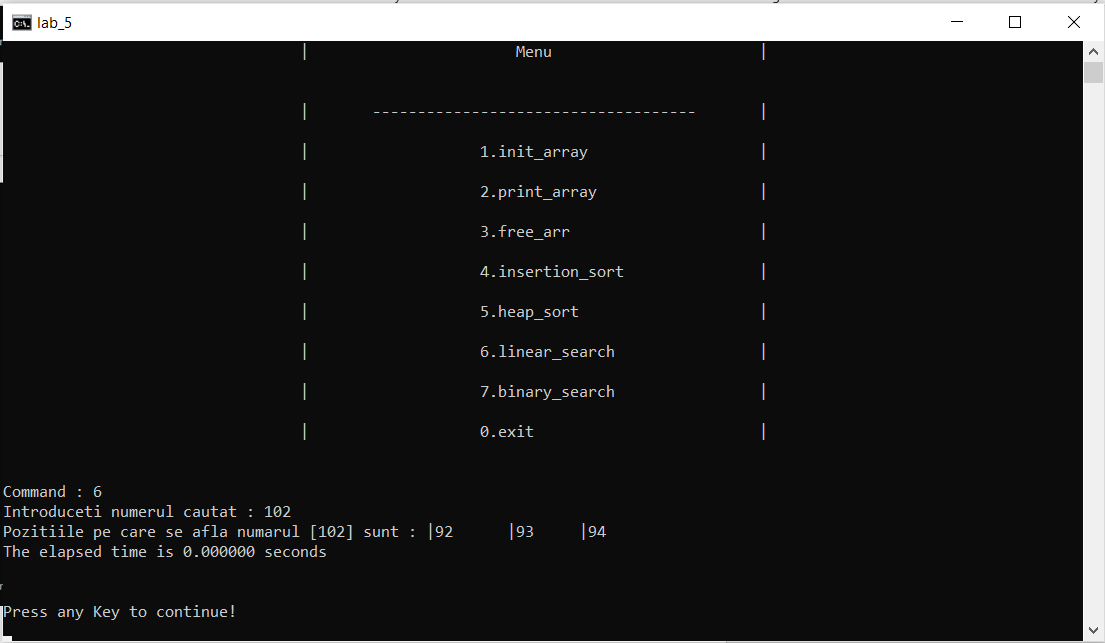
Insertion sort :



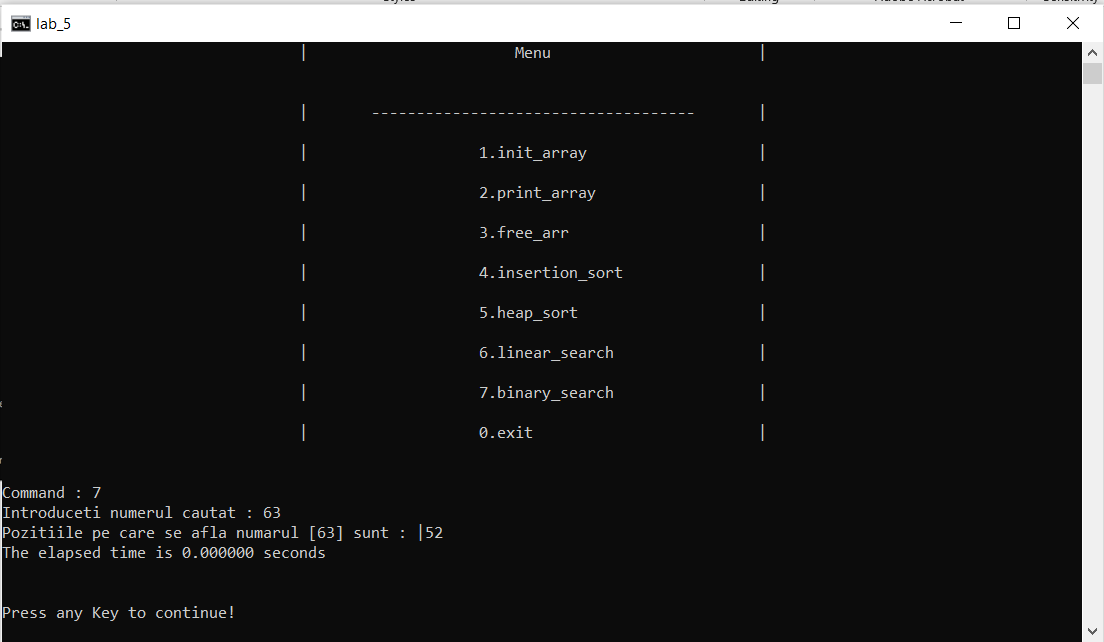
Heap Sort :



Linear search :



Binary Search :



Free array :

