Ministerul Educaţiei și Cercetării

al Republicii Moldova   
Universitatea Tehnică a Moldovei

Faculatea Calculatoare, Infromatică și Microelectronică   
  
  
  
  
  
  
  
RAPORT

# LUCRARE DE LABORATOR NR. 2

**La disciplina: Analiza şi Proiectarea Algoritmilor**

**Tema: Metoda divide et impera.**

A efectuat: st. gr. TI-211 Popa Cătălin   
  
A verificat: asist. Univ. Andrieschi-Bagrin Veronica

UTM, Chișinău 2022

**Tema:**

Metoda divide et impera.

**Scopul lucrării:**

1.Studierea metodei divide et impera.

2. Analiza şi implementarea algoritmilor bazaţi pe metoda divide et impera.

**Sarcina de bază:**

1.Studierea noțiunilor despre metoda divide ey impera.

2.Implementarea algoritmilor Mergesort, Bubblesort, Quicksort.

3.Efectuarea analizei empirică a algoritmilor.

4.Concluzie asupra lucrării.

**Sursa cod**

#include <iostream>

#include <array>

#include <conio.h>

#include <windows.h>

#include <limits>

#include <chrono>

using namespace std::chrono;

using namespace std;

long int it=0;

////////////////////////////////// Merge sort

void merge(int arr[], int left, int mid, int right)

{

    int size1 = mid - left + 1;

    it += 3;

    int size2 = right - mid;

    it += 2;

    int \*L = new int[size1];

    int \*R = new int[size2];

    for (int i = 0; i < size1; i++)

    {

        it += 2;

        L[i] = arr[left + i];

        it += 3;

    }

    for (int j = 0; j < size2; j++)

    {

        it += 3;

        R[j] = arr[mid + 1 + j];

        it += 3;

    }

    int i, j, k;

    i = 0;

    j = 0;

    k = left;

    it += 3;

    while (i < size1 && j < size2)

    {

        it += 2;

        if (L[i] <= R[j])

        {

            it += 2;

            arr[k] = L[i];

            i++;

            it += 1;

        }

        else

        {

            arr[k] = R[j];

            j++;

            it += 2;

        }

        k++;

        it++;

    }

    while (i < size1)

    {

        it++;

        arr[k] = L[i];

        i++;

        k++;

        it += 3;

    }

    while (j < size2)

    {

        it++;

        arr[k] = R[j];

        j++;

        k++;

        it += 3;

    }

}

void merge\_sort(int arr[], int begin, int end)

{

    if (begin < end)

    {

        it +=1;

        int midlle = ((begin + end) / 2);

        it += 3;

        merge\_sort(arr, begin, midlle);

        merge\_sort(arr, midlle + 1, end);

        it++;

        merge(arr, begin, midlle, end);

    }

}

////////////////////////////////////////////////////Bubble

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

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

    it+=3;

}

void bubble\_sort(int arr[], int size)

{

    int i, j;

    for (i = 0; i < size - 1; i++)

    {

        it += 4;

        for (j = 0; j < size - i - 1; j++)

        {

            it += 4;

            if (arr[j] > arr[j + 1])

            {

                it += 2;

                swap(arr[j], arr[j + 1]);

                it += 1;

            }

        }

    }

}

/// ///////////////////////////////////////////Quick

int partition (int arr[], int low, int high)

{

    int pivot = arr[high];

    int i = (low - 1);

    it += 3;

    for (int j = low; j <= high- 1; j++)

    {

        it += 5;

        if (arr[j] <= pivot)

        {

            i++;

            it += 3;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    it += 2;

    return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high)

    {

        it += 1;

        int pivot = partition(arr, low, high);

        quickSort(arr, low, pivot - 1);

        quickSort(arr, pivot + 1, high);

        it += 3;

    }

}

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

int \*new\_array(int size)

{

    int value;

    int j = 0;

    int \*ar = new int[size];

    int \*ar2 = new int[size];

    cout << "\n1.Cazul favorabil\n2.Random\n3.Cazul defavorabil\n";

    cin >> value;

    if (value == 1)

    {

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

        {

            ar[i] = i;

        }

    }

    else if (value == 2)

    {

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

        {

            int x = rand() % size;

            ar[i] = x;

        }

    }

    else if (value == 3)

    {

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

        {

            ar2[i] = i;

        }

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

        {

            ar[j++] = ar2[i--];

        }

    }

    return ar;

}

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

void printArray(int arr[], int size)

{

    for (auto i = 0; i < size; i++)

        cout << arr[i] << " ";

}

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

int menu()

{

    int error\_input\_found = 0;

menu\_start:

    system("cls");

    cout << "|           Menu                    |\n";

    printf("|    1.Dati lungimea tabloului       |\n");

    printf("|           2.Merge                  |\n");

    printf("|           3.Bubble                 |\n");

    printf("|           4.Quick                  |\n");

    printf("|           5.Afisarea               |\n");

    int chose = 9;

    if (error\_input\_found)

    {

        printf("\n\nSelectati o comanda : \n");

    }

    printf("\n\nCommand : ");

    cin >> chose;

    while (1)

    {

        if (cin.fail())

        {

            cin.clear();

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            system("cls");

            goto menu\_start;

        }

        if (!cin.fail())

            return chose;

        break;

    }

}

int main()

{

    int \*ar = NULL;

    int size;

    int j = 0;

    int \*ar2 = new int[size];

    int chose = 5;

    while (chose != 0)

    {

        chose = menu();

        switch (chose)

        {

        case 1:

            cout << "\n----\n";

            cin >> size;

            break;

        case 2:

        {

            it = 0;

            auto start = high\_resolution\_clock::now();

            cout << "\nMerge sort: \n";

            ar = new\_array(size);

            merge\_sort(ar, 0, size);

            auto stop = high\_resolution\_clock::now();

            cout << "\nNumarul de iteratii: "<< it;

            auto duration = duration\_cast<microseconds>(stop - start);

            cout << "\nTimpul: "<< duration.count() << " microsecunde" << endl;

            break;

        }

        case 3:

        {

            it = 0;

            auto start = high\_resolution\_clock::now();

            cout << "\nBubble sort: \n";

            ar = new\_array(size);

            bubble\_sort(ar, size);

            auto stop = high\_resolution\_clock::now();

            cout << "\nNumarul de iteratii: "<< it;

            auto duration = duration\_cast<microseconds>(stop - start);

             cout << "\nTimpul: "<< duration.count() << " milesecunde" << endl;

            break;

        }

        case 4:

        {

            it = 0;

            auto start = high\_resolution\_clock::now();

            cout << "\nQuick sort: \n";

            ar = new\_array(size);

            quickSort(ar, 0, size-1);

            auto stop = high\_resolution\_clock::now();

            cout << "\nNumarul de iteratii: "<< it;

            auto duration = duration\_cast<microseconds>(stop - start);

             cout << "\nTimpul: "<< duration.count() << " milesecunde" << endl;

            break;

        }

        case 5:

        {

            cout << "\nAfisarea tabloului:\n";

            printArray(ar, size);

        }

        break;

        case 0:

            exit(0);

            break;

        default:

            break;

        }

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

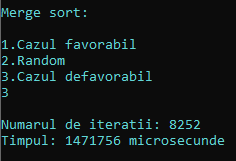
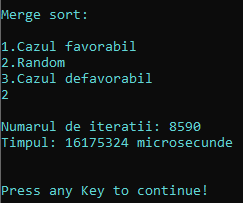
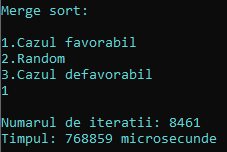
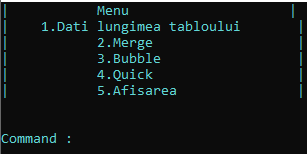
        \_getch();

    }

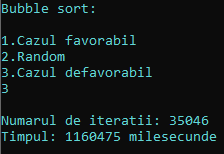
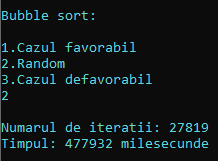
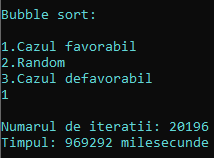
    return 0;

}

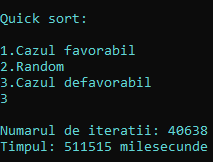
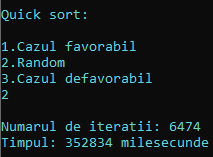
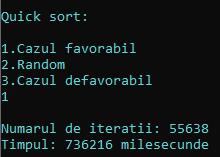
1.Menu 2.Merge sort



3.Bubble sort



4. Quick sort



**Cazuri favorabile**

**Cazuri defavorabile**

**Concluzie**

Efectuînd lucrarea dată am implementat algoritmi de sortare, quicksort , mergesort si bubblesort, bazaţi pe analizind timpul de executie al algoritmilor daţi. Aceşti algoritmi au un timp de execuţie diferit, de aceea este important de a alege algoritmul cel mai eficient, adică algoritmul care are un timp de execuţie mic,se observa :

1.Din cele analizate se vede ca quicksort este un algoritm mai rapid,la quick sort se efectueaza mai multe iteratii.

2.Dar se observa ca dupa analiza si executare cel mai incet lucreaza sortarea dupa mergesort.