**Ministerul Educaţiei și Cercetării al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică și Microelectronică**

**RAPORT**

Lucrarea de laborator nr.2

*la Analiza și Proiectarea Algoritmilor*

A efectuat:

st. gr. TI-216 Roșca Dorin

A verificat : Lisnic Inga

Chişinău - 2022

**Lucrare de laborator nr 2**

**Tema:** Metoda divide et impera.

**Scopul lucrării**:

1. Studierea metodei divide et impera.

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

**Rezumat succint la tema lucrării de laborator:**

Tehnica divide et impera Divide et impera este o tehnica de elaborare a algoritmilor care constă în:

1. Descompunerea cazului ce trebuie rezolvat într-un număr de subcazuri mai mici ale aceleiaşi probleme.

2. Rezolvarea succesivă şi independentă a fiecăruia din aceste subcazuri.

3. Recompunerea subsoluţiilor astfel obţinute pentru a găsi soluţia cazului iniţial. Să presupunem că avem un algoritm A cu timp pătratic. Fie c o constantă, astfel încât timpul pentru a rezolva un caz de mărime n este tA(n) ≤ cn2 . Să presupunem că este posibil să rezolvăm un astfel de caz prin descompunerea în trei subcazuri, fiecare de mărime [n/2]. Fie d o constantă, astfel încât timpul necesar pentru descompunere şi recompunere este t(n) ≤ dn. Folosind vechiul algoritm şi ideea de descompunere-recompunere a subcazurilor, obţinem un nou algoritm B, pentru care: tB(n) = 3tA(n/2)+t(n) < 3c((n+1)/2)2+dn = 3/4cn2+(3/2+d)n+3/4c

Termenul 3/4cn2 domină pe ceilalţi când n este suficient de mare, ceea ce înseamnă ca algoritmul B este în esenţă cu 25% mai rapid decât algoritmul A.

Nu am reuşit însă să schimbăm ordinul timpului, care rămâne pătratic. Putem să continuăm în mod recursiv acest procedeu, împărţind subcazurile în subsubcazuri etc. Pentru subcazurile care nu sunt mai mari decât un anumit prag n0, vom folosi tot algoritmul A. Obţinem astfel algoritmul C, cu timpul tC(n) este în ordinul lui n lg 3 . Deoarece lg 3 ≅ 1,59, înseamnă că de această dată am reuşit să îmbunătăţim ordinul timpului. Algoritmul formal al metodei divide et impera: funcţion divimp(x) {returnează o soluţie pentru cazul x} if x este suficient de mic then return adhoc(x) {descompune x în subcazurile x1, x2, …, xk} for i  1 to k do yi  divimp(xi) {recompune y1, y2, …, yk în scopul obţinerii soluţiei y pentru x} return y unde adhoc este subalgoritmul de bază folosit pentru rezolvarea micilor subcazuri ale problemei în cauză (în exemplul nostru, acest subalgoritm este A).

**Codul programului in C**

**Mergesort:**

**#include** <iostream>

**using** **namespace** std;

void **merge**(int \*, int, int, int);

void **merge\_sort**(int \*tab, int mic, int mare)

{

    int mij;

**if** (mic **<** mare)

    {

// *dividez tabloul la mijloc si sortez independent utilizand mergesort*

        mij = (mic + mare) / 2;

**merge\_sort**(tab, mic, mij);

**merge\_sort**(tab, mij + 1, mare);

// *merge or conquer sorted tabays*

**merge**(tab, mic, mare, mij);

    }

}

// *Merge sort*

void **merge**(int \*tab, int mic, int mare, int mij)

{

    int i, j, k, c[50];

    i = mic;

    k = mic;

    j = mij + 1;

**while** (i **<=** mij **&&** j **<=** mare)

    {

**if** (tab[i] **<** tab[j])

        {

            c[k] = tab[i];

            k++;

            i++;

        }

**else**

        {

            c[k] = tab[j];

            k++;

            j++;

        }

    }

**while** (i **<=** mij)

    {

        c[k] = tab[i];

        k++;

        i++;

    }

**while** (j **<=** mare)

    {

        c[k] = tab[j];

        k++;

        j++;

    }

**for** (i = mic; i **<** k; i++)

    {

        tab[i] = c[i];

    }

}

// *citesc rezultatul si afisez*

int **main**()

{

    int mytabay[] = {8, 7, 6, 1, 0, 9, 2};

    int num = 7;

**merge\_sort**(mytabay, 0, num - 1);

    cout **<<** "Tabelul sortat:\n";

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

    {

        cout **<<** mytabay[i] **<<** "\t";

    }

}

**Quicksort:**

**#include** <iostream>

**using** **namespace** std;

// *functia pentru a schimba elementele*

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

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

// *functie de afisare tablou*

void **printtabel**(int tabel[], int size)

{

    int i;

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

        cout **<<** tabel[i] **<<** " ";

    cout **<<** **endl**;

}

// *functe de rearanjare (gasire punctul de repartizare)*

int **partition**(int tabel[], int mic, int mare)

{

// *selectare cel mai din dreapta ca  pivot*

    int pivot = tabel[mare];

// *pointer pentru cel mai mare numar*

    int i = (mic - 1);

// *traversez fiecare element al tabloului*

// *le compar cu pivotul*

**for** (int j = mic; j **<** mare; j++)

    {

**if** (tabel[j] **<=** pivot)

        {

// *daca element mai mic ca pivotul e gasit*

// *il schimb cu cel mai mare i*

            i++;

// *schimb elementele intre j si i*

**swap**(&tabel[i], &tabel[j]);

        }

    }

// *schimb pivotul cu cel mai mare element*

**swap**(&tabel[i + 1], &tabel[mare]);

// *returnez elementul de repartizare*

**return** (i + 1);

}

void **quickSort**(int tabel[], int mic, int mare)

{

**if** (mic **<** mare)

    {

// *gasesc elemnt c*

// *elemente mai mici ca pivot sunt la stanga de acesta*

// *elemente mai mare ca pivot este la dreapta de acesta*

        int pi = **partition**(tabel, mic, mare);

// *apel recursiv la stanga de pivot*

**quickSort**(tabel, mic, pi - 1);

// *apel recursiv la dreapta de pivot*

**quickSort**(tabel, pi + 1, mare);

    }

}

int **main**()

{

    int data[] = {8, 7, 6, 1, 0, 9, 2};

    int n = **sizeof**(data) / **sizeof**(data[0]);

    cout **<<** "tabel nesortat: \n";

**printtabel**(data, n);

// *quicksort*

**quickSort**(data, 0, n - 1);

    cout **<<** "Tabel sortat: \n";

**printtabel**(data, n);

}

**Concluzie**

Efectuînd lucrarea dată am implementat algoritmi de sortare, quicksort si mergesort, bazaţi pe analizind timpul de executie al algoritmilor daţi. Aceşti algoritmi sunt foarte asemanatori deoarece ambii se bazeaza pe principiul divide ed impera adica,masivul este impartit in mai multe parti egale,care fiind sortate intre ele,la unire arata tabloul sortat

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