Praktikum 7

Bubble sort dan Shell sort

Nama: Dio Stania Adinata

NRP: 5223600024

latihan 1

Input:

#include <iostream>

#include <cstdlib>

#define MAX 10

using namespace std;

int Data[MAX];

int temp[MAX];

// fungsi merge sort

void merge(int Data[], int temp[], int kiri, int tengah, int kanan)

{

    int i, left\_end, num\_elements, tmp\_pos;

    left\_end = tengah - 1;

    tmp\_pos = kiri;

    num\_elements = kanan - kiri + 1;

    while ((kiri <= left\_end) && (tengah <= kanan))

    {

        if (Data[kiri] <= Data[tengah])

        {

            temp[tmp\_pos] = Data[kiri];

            tmp\_pos = tmp\_pos + 1;

            kiri = kiri + 1;

        }

        else

        {

            temp[tmp\_pos] = Data[tengah];

            tmp\_pos = tmp\_pos + 1;

            tengah = tengah + 1;

        }

    }

    while (kiri <= left\_end)

    {

        temp[tmp\_pos] = Data[kiri];

        kiri = kiri + 1;

        tmp\_pos = tmp\_pos + 1;

    }

    while (tengah <= kanan)

    {

        temp[tmp\_pos] = Data[tengah];

        tengah = tengah + 1;

        tmp\_pos = tmp\_pos + 1;

    }

    for (i = 0; i < num\_elements; i++)

    {

        Data[kanan] = temp[kanan];

        kanan = kanan - 1;

    }

}

// fungsi unutk membuat kumpulan data

void m\_sort(int Data[], int temp[], int kiri, int kanan)

{

    int tengah;

    if (kanan > kiri)

    {

        tengah = (kanan + kiri) / 2;

        m\_sort(Data, temp, kiri, tengah);

        m\_sort(Data, temp, tengah + 1, kanan);

        merge(Data, temp, kiri, tengah + 1, kanan);

    }

}

void mergeSort(int Data[], int temp[], int array\_size)

{

    m\_sort(Data, temp, 0, array\_size - 1);

}

// fungsu untuk pengurutan metode Quick Sort

void QuickSortRekursif(int L, int R)

{

    int i, j, x;

    x = Data[(L + R) / 2];

    i = L;

    j = R;

    while (i <= j) {

        while (Data[i] < x)

            i++;

        while (Data[j] > x)

            j--;

        if (i <= j) {

            swap(Data[i], Data[j]);

            i++;

            j--;

        }

    }

    if (L < j)

        QuickSortRekursif(L, j);

    if (i < R)

        QuickSortRekursif(i, R);

}

void displayArray(int arr[], int size)

{

    for (int i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    srand(0);

    // menginisiasi bilangan acak

    cout << "DATA SEBELUM TERURUT : ";

    for (int i = 0; i < MAX; i++)

    {

        Data[i] = rand() / 1000 + 1;

        cout << Data[i] << " ";

    }

    cout << endl;

    // fungsi Merge Sort

    cout << "\nMerge Sort Process:" << endl;

    for (int size = 1; size <= MAX; size \*= 2)

    {

        cout << "Array saat ini: ";

        mergeSort(Data, temp, size);

        displayArray(Data, MAX);

    }

    // fungsi Quick Sort

    cout << "\nQuick Sort Process:" << endl;

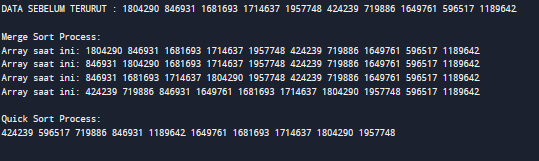
    QuickSortRekursif(0, MAX - 1);

    displayArray(Data, MAX);

    return 0;

}

Output:



Latihan 2

Input:

#include <iostream>

#include <cstdlib>

#define MAX 10

using namespace std;

int Data[MAX];

int temp[MAX];

int comparisonCountMerge = 0;

int shiftCountMerge = 0;

int comparisonCountQuick = 0;

int shiftCountQuick = 0;

// fungsi untuk merge sort

void merge(int Data[], int temp[], int kiri, int tengah, int kanan)

{

    int i, left\_end, num\_elements, tmp\_pos;

    left\_end = tengah - 1;

    tmp\_pos = kiri;

    num\_elements = kanan - kiri + 1;

    while ((kiri <= left\_end) && (tengah <= kanan))

    {

        comparisonCountMerge++;

        if (Data[kiri] <= Data[tengah])

        {

            temp[tmp\_pos] = Data[kiri];

            tmp\_pos = tmp\_pos + 1;

            kiri = kiri + 1;

        }

        else

        {

            temp[tmp\_pos] = Data[tengah];

            tmp\_pos = tmp\_pos + 1;

            tengah = tengah + 1;

            shiftCountMerge++;

        }

    }

    while (kiri <= left\_end)

    {

        temp[tmp\_pos] = Data[kiri];

        kiri = kiri + 1;

        tmp\_pos = tmp\_pos + 1;

        shiftCountMerge++;

    }

    while (tengah <= kanan)

    {

        temp[tmp\_pos] = Data[tengah];

        tengah = tengah + 1;

        tmp\_pos = tmp\_pos + 1;

        shiftCountMerge++;

    }

    for (i = 0; i < num\_elements; i++)

    {

        Data[kanan] = temp[kanan];

        kanan = kanan - 1;

    }

}

// fungsi untuk membuat kumpulan data

void m\_sort(int Data[], int temp[], int kiri, int kanan)

{

    int tengah;

    if (kanan > kiri)

    {

        tengah = (kanan + kiri) / 2;

        m\_sort(Data, temp, kiri, tengah);

        m\_sort(Data, temp, tengah + 1, kanan);

        merge(Data, temp, kiri, tengah + 1, kanan);

    }

}

void mergeSort(int Data[], int temp[], int array\_size)

{

    m\_sort(Data, temp, 0, array\_size - 1);

}

// fungsi untuk pengurutan metode Quick Sort

void QuickSortRekursif(int L, int R)

{

    int i, j, x;

    x = Data[(L + R) / 2];

    i = L;

    j = R;

    while (i <= j) {

        comparisonCountQuick++;

        while (Data[i] < x)

        {

            i++;

            comparisonCountQuick++;

        }

        while (Data[j] > x)

        {

            j--;

            comparisonCountQuick++;

        }

        if (i <= j) {

            swap(Data[i], Data[j]);

            i++;

            j--;

            shiftCountQuick++;

        }

    }

    if (L < j)

        QuickSortRekursif(L, j);

    if (i < R)

        QuickSortRekursif(i, R);

}

void displayArray(int arr[], int size)

{

    for (int i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    srand(0);

    // menginisiasi bilangan acak

    cout << "DATA SEBELUM TERURUT : ";

    for (int i = 0; i < MAX; i++)

    {

        Data[i] = rand() / 1000 + 1;

        cout << Data[i] << " ";

    }

    cout << endl;

    // fungsi Merge Sort

    cout << "\nMerge Sort Process:" << endl;

    for (int size = 1; size <= MAX; size \*= 2)

    {

        mergeSort(Data, temp, size);

        cout << "Array saat ini: ";

        displayArray(Data, MAX);

        cout << "Perbandingan: " << comparisonCountMerge << ", Pergeseran: " << shiftCountMerge << endl;

    }

    cout << endl;

    // fungsi Quick Sort

    cout << "\nQuick Sort Process:" << endl;

    QuickSortRekursif(0, MAX - 1);

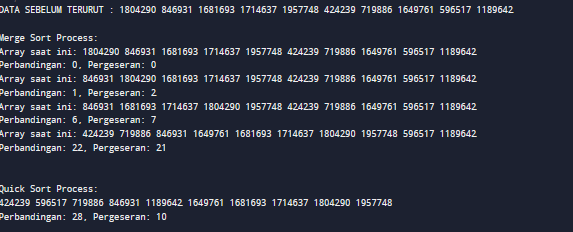
    displayArray(Data, MAX);

    cout << "Perbandingan: " << comparisonCountQuick << ", Pergeseran: " << shiftCountQuick << endl;

    return 0;

}

Output:



Latihan 3

Input:

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

// Struktur data Pegawai

struct Pegawai {

    string NIP;

    string Nama;

    int Usia;

    double Gaji;

};

// Fungsi untuk membandingkan dua Pegawai berdasarkan NIP

bool compareByNIP(const Pegawai& a, const Pegawai& b) {

    return a.NIP < b.NIP;

}

// Fungsi untuk membandingkan dua Pegawai berdasarkan Nama

bool compareByNama(const Pegawai& a, const Pegawai& b) {

    return a.Nama < b.Nama;

}

// Fungsi untuk menampilkan data Pegawai

void displayPegawai(const vector<Pegawai>& pegawai) {

    for (const auto& p : pegawai) {

        cout << "NIP: " << p.NIP << ", Nama: " << p.Nama << ", Usia: " << p.Usia << ", Gaji: " << p.Gaji << endl;

    }

}

int main() {

    // fungsi untuk Membuat data Pegawai

    vector<Pegawai> dataPegawai = {

        {"123", "Dio", 20, 5500},

        {"456", "Nauval", 21, 6060},

        {"789", "Rakha", 28, 4550},

        {"234", "Rifath", 32, 8500},

        {"567", "Adelia", 18, 5700}

    };

    // fungsi untuk memilih metode pengurutan

    int choice;

    cout << "Pilih metode pengurutan:" << endl;

    cout << "1. Berdasarkan NIP" << endl;

    cout << "2. Berdasarkan Nama" << endl;

    cout << "Pilih: " << endl;

    cin >> choice;

    // Memilih urutan pengurutan

    int order;

    cout << "Pilih urutan pengurutan:" << endl;

    cout << "1. Urut naik" << endl;

    cout << "2. Urut turun" << endl;

    cout << "Pilih: " << endl;

    cin >> order;

    // fungsi untuk menggunakan fungsi pengurutan sesuai dengan pilihan

    if (choice == 1) {

        if (order == 1) {

            // fungsi untuk mengurutkan berdasarkan NIP secara naik

            sort(dataPegawai.begin(), dataPegawai.end(), compareByNIP);

        }

        else {

            // fungsi untuk mengurutkan berdasarkan NIP secara turun

            sort(dataPegawai.begin(), dataPegawai.end(), [](const Pegawai& a, const Pegawai& b) {

                return a.NIP > b.NIP;

                });

        }

    }

    else {

        if (order == 1) {

            // fungsi untuk mengurutkan berdasarkan Nama secara naik

            sort(dataPegawai.begin(), dataPegawai.end(), compareByNama);

        }

        else {

            // fungsi untuk mengurutkan berdasarkan Nama secara turun

            sort(dataPegawai.begin(), dataPegawai.end(), [](const Pegawai& a, const Pegawai& b) {

                return a.Nama > b.Nama;

                });

        }

    }

    // Menampilkan data Pegawai setelah diurutkan

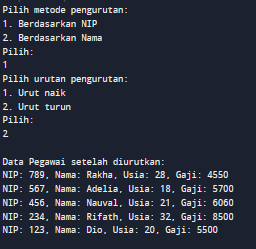
    cout << "\nData Pegawai setelah diurutkan:" << endl;

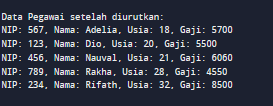
    displayPegawai(dataPegawai);

    return 0;

}

Output:





Latihan 4

Implementasi Quick Sort dan Merge Sort pada data Pegawai menunjukkan hasil yang memuaskan. Pengguna dapat memilih metode pengurutan (berdasarkan NIP atau Nama) dan urutan pengurutan (naik atau turun), sehingga proses pengurutan menjadi fleksibel dan efisien.