Praktikum 08

Praktikum Quick Sort, Merge Sort

Nama : Tora Sandhi Kamulian

NRP : 5223600013

Percobaan 1 :

Input :

```
#include <iostream>
#include <cstdlib>
#define MAX 10
#define MaxStack 10
using namespace std;

int Data[MAX];

// Prosedur menukar data
void Tukar(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

// Prosedur pengurutan metode Quick Sort non-rekursif
void QuickSortNonRekursif()
{
    // Deklarasi tipe data struct untuk tumpukan
    struct Tumpukan {
        int Kiri;
        int Kanan;
    } Tumpukan[MaxStack];

    int i, j, L, R, x, ujung = 1;

    // Inisialisasi tumpukan
    Tumpukan[1].Kiri = 0;
    Tumpukan[1].Kanan = MAX - 1;

    // Loop utama untuk pengurutan
    while (ujung != 0) {
        L = Tumpukan[ujung].Kiri;
        R = Tumpukan[ujung].Kanan;
        ujung--;

        // Proses pembagian data menjadi dua bagian
        while (R > L) {
            i = L;
            j = R;
            x = Data[(L + R) / 2];
            while (i <= j) {
                while (Data[i] < x)
                    i++;
```

```
                while (x < Data[j])
                    j--;
                if (i <= j) {
                    Tukar(&Data[i], &Data[j]);
                    i++;
                    j--;
                }
            }

            // Memeriksa dan memasukkan indeks baru ke tumpukan
            if (L < i) {
                ujung++;
                Tumpukan[ujung].Kiri = i;
                Tumpukan[ujung].Kanan = R;
            }
            R = j;
        }
    }
}

int main()
{
    srand(0);

    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT" << endl;
    for (int i = 0; i < MAX; i++)
    {
        Data[i] = (int)rand() / 1000 + 1;
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }

    // Memanggil fungsi QuickSortNonRekursif untuk mengurutkan data
    QuickSortNonRekursif();

    // Menampilkan data setelah diurutkan
    cout << "\nDATA SETELAH TERURUT" << endl;
    for (int i = 0; i < MAX; i++)
    {
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }

    return 0;
}
```

Output :

```
DATA SEBELUM TERURUT
Data ke 0 : 1804290
Data ke 1 : 846931
Data ke 2 : 1681693
Data ke 3 : 1714637
Data ke 4 : 1957748
Data ke 5 : 424239
Data ke 6 : 719886
Data ke 7 : 1649761
Data ke 8 : 596517
Data ke 9 : 1189642

DATA SETELAH TERURUT
Data ke 0 : 424239
Data ke 1 : 596517
Data ke 2 : 719886
Data ke 3 : 846931
Data ke 4 : 1189642
Data ke 5 : 1649761
Data ke 6 : 1681693
Data ke 7 : 1714637
Data ke 8 : 1804290
Data ke 9 : 1957748


=== Code Execution Successful ===
```

Percobaan 2 :

Input :

```cpp
#include <iostream>
#include <cstdlib>
#define MAX 10
using namespace std;

int Data[MAX];

// Prosedur menukar data
void Tukar(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

// Prosedur pengurutan metode Quick Sort rekursif
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) {
            Tukar(&Data[i], &Data[j]);
            i++;
            j--;
        }
    }
    if (L < j)
        QuickSortRekursif(L, j);
    if (i < R)
        QuickSortRekursif(i, R);
}

int main()
{
    srand(0);

    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT" << endl;
    for (int i = 0; i < MAX; i++)
    {
        Data[i] = (int)rand() / 1000 + 1;
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }

    // Memanggil fungsi QuickSortRekursif untuk mengurutkan data
    QuickSortRekursif(0, MAX - 1);

    // Menampilkan data setelah diurutkan
    cout << "\nDATA SETELAH TERURUT" << endl;
    for (int i = 0; i < MAX; i++)
    {
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }

    return 0;
}
```

Output :

```
DATA SEBELUM TERURUT
Data ke 0 : 1804290
Data ke 1 : 846931
Data ke 2 : 1681693
Data ke 3 : 1714637
Data ke 4 : 1957748
Data ke 5 : 424239
Data ke 6 : 719886
Data ke 7 : 1649761
Data ke 8 : 596517
Data ke 9 : 1189642

DATA SETELAH TERURUT
Data ke 0 : 424239
Data ke 1 : 596517
Data ke 2 : 719886
Data ke 3 : 846931
Data ke 4 : 1189642
Data ke 5 : 1649761
Data ke 6 : 1681693
Data ke 7 : 1714637
Data ke 8 : 1804290
Data ke 9 : 1957748


=== Code Execution Successful ===
```

Percobaan 3 :

Input :

```cpp
#include <iostream>
#include <cstdlib>
#define MAX 10
using namespace std;

int Data[MAX];
int temp[MAX];

// Prosedur 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;
    }
}

// Prosedur 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);
}

int main()
{
    srand(0);

    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT : ";
    for (int i = 0; i < MAX; i++)
    {
        Data[i] = rand() / 1000 + 1;
        cout << Data[i] << " ";
    }

    mergeSort(Data, temp, MAX);

    // Menampilkan data setelah diurutkan
    cout << "\nDATA SETELAH TERURUT : ";
    for (int i = 0; i < MAX; i++)
        cout << Data[i] << " ";
```

```
    cout << endl;

    return 0;
}
```

Output :

```
DATA SEBELUM TERURUT : 1804290 846931 1681693 1714637 1957748 424239 719886 1649761 596517 1189642
DATA SETELAH TERURUT : 424239 596517 719886 846931 1189642 1649761 1681693 1714637 1804290 1957748


=== Code Execution Successful ===
```

Latihan 1 :

Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses

pengurutan dengan quick sort dan merge sort.

Input :

```
#include <iostream>
#include <cstdlib>
#define MAX 10
using namespace std;

int Data[MAX];
int temp[MAX];

// Prosedur 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;
    }
}

// Prosedur 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);
}

// Prosedur 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);

    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT : ";
    for (int i = 0; i < MAX; i++)
    {
        Data[i] = rand() / 1000 + 1;
        cout << Data[i] << " ";
    }
    cout << endl;

    // 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);
    }

    // Quick Sort
    cout << "\nQuick Sort Process:" << endl;
    QuickSortRekursif(0, MAX - 1);
    displayArray(Data, MAX);

    return 0;
}
```

Output :

```
DATA SEBELUM TERURUT : 1804290 846931 1681693 1714637 1957748 424239 719886 1649761 596517 1189642

Merge Sort Process:
Array saat ini: 1804290 846931 1681693 1714637 1957748 424239 719886 1649761 596517 1189642
Array saat ini: 846931 1804290 1681693 1714637 1957748 424239 719886 1649761 596517 1189642
Array saat ini: 846931 1681693 1714637 1804290 1957748 424239 719886 1649761 596517 1189642
Array saat ini: 424239 719886 846931 1649761 1681693 1714637 1804290 1957748 596517 1189642

Quick Sort Process:
424239 596517 719886 846931 1189642 1649761 1681693 1714637 1804290 1957748


=== Code Execution Successful ===
```

Latihan 2:

Tambahkan kode program untuk menghitung banyaknya perbandingan dan

pergeseran pada algoritma quick sort dan merge sort.

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;

// Prosedur 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;
    }
}

// Prosedur 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);
}

// Prosedur 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);

    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT : ";
    for (int i = 0; i < MAX; i++)
    {
        Data[i] = rand() / 1000 + 1;
        cout << Data[i] << " ";
    }
    cout << endl;

    // 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;

    // Quick Sort
    cout << "\nQuick Sort Process:" << endl;
    QuickSortRekursif(0, MAX - 1);
    displayArray(Data, MAX);
    cout << "Perbandingan: " << comparisonCountQuick << ", Pergeseran: " <<
shiftCountQuick << endl;

    return 0;
}
```

Output :

```
DATA SEBELUM TERURUT : 1804290 846931 1681693 1714637 1957748 424239 719886 1649761 596517 1189642

Merge Sort Process:
Array saat ini: 1804290 846931 1681693 1714637 1957748 424239 719886 1649761 596517 1189642
Perbandingan: 0, Pergeseran: 0
Array saat ini: 846931 1804290 1681693 1714637 1957748 424239 719886 1649761 596517 1189642
Perbandingan: 1, Pergeseran: 2
Array saat ini: 846931 1681693 1714637 1804290 1957748 424239 719886 1649761 596517 1189642
Perbandingan: 6, Pergeseran: 7
Array saat ini: 424239 719886 846931 1649761 1681693 1714637 1804290 1957748 596517 1189642
Perbandingan: 22, Pergeseran: 21


Quick Sort Process:
424239 596517 719886 846931 1189642 1649761 1681693 1714637 1804290 1957748
Perbandingan: 28, Pergeseran: 10
```

Latihan 3 :

Implementasikan pengurutan data Pegawai pada tugas pendahuluan dengan

ketentuan :

a. Metode pengurutan dapat dipilih.

b. Pengurutan dapat dipilih secara urut naik atau turun.

c. Pengurutan dapat dipilih berdasarkan NIP dan NAMA.

d. Gunakan struktur data array.

Input :

```
#include <iostream>
```

```cpp
#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() {
    // Membuat data Pegawai
    vector<Pegawai> dataPegawai = {
        {"123", "John Doe", 30, 5000},
        {"456", "Jane Smith", 35, 6000},
        {"789", "Alice Johnson", 28, 4500},
        {"234", "Bob Williams", 32, 5500},
        {"567", "Emily Davis", 40, 7000}
    };

    // Memilih metode pengurutan
    int choice;
    cout << "Pilih metode pengurutan:" << endl;
    cout << "1. Berdasarkan NIP" << endl;
    cout << "2. Berdasarkan Nama" << endl;
    cin >> choice;

    // Memilih urutan pengurutan
    int order;
    cout << "Pilih urutan pengurutan:" << endl;
    cout << "1. Urut naik" << endl;
    cout << "2. Urut turun" << endl;
    cin >> order;

    // Menggunakan fungsi pengurutan sesuai dengan pilihan
    if (choice == 1) {
        if (order == 1) {
            // Pengurutan berdasarkan NIP secara naik
            sort(dataPegawai.begin(), dataPegawai.end(), compareByNIP);
        }
        else {
```

```
            // Pengurutan berdasarkan NIP secara turun
            sort(dataPegawai.begin(), dataPegawai.end(), [](const Pegawai& a,
const Pegawai& b) {
                return a.NIP > b.NIP;
                });
        }
    }
    else {
        if (order == 1) {
            // Pengurutan berdasarkan Nama secara naik
            sort(dataPegawai.begin(), dataPegawai.end(), compareByNama);
        }
        else {
            // Pengurutan 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 :

a. Bisa Pilih Metode Pengurutan

```
// Memilih metode pengurutan
int choice;
cout << "Pilih metode pengurutan:" << endl;
cout << "1. Berdasarkan NIP" << endl;
cout << "2. Berdasarkan Nama" << endl;
cin >> choice;
```

```
Pilih metode pengurutan:
1. Berdasarkan NIP
2. Berdasarkan Nama
```

b. Bisa Pilih urutan Pengurutan (Naik/Turun)

```
// Memilih urutan pengurutan
int order;
cout << "Pilih urutan pengurutan:" << endl;
cout << "1. Urut naik" << endl;
cout << "2. Urut turun" << endl;
cin >> order;
```

```
Pilih urutan pengurutan:
1. Urut naik
2. Urut turun
1
```

c. Bisa berdasarkan Nama dan NIP

```
// Menggunakan fungsi pengurutan sesuai dengan pilihan
if (choice == 1) {
    if (order == 1) {
        // Pengurutan berdasarkan NIP secara naik
        sort(dataPegawai.begin(), dataPegawai.end(), compareByNIP);
    }
    else {
        // Pengurutan berdasarkan NIP secara turun
        sort(dataPegawai.begin(), dataPegawai.end(), [](const Pegawai& a, const Pegawai& b) {
            return a.NIP > b.NIP;
            });
    }
}
else {
    if (order == 1) {
        // Pengurutan berdasarkan Nama secara naik
        sort(dataPegawai.begin(), dataPegawai.end(), compareByNama);
    }
    else {
        // Pengurutan 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);
```

d. menggunakan struktur data array

```
// Membuat data Pegawai
vector<Pegawai> dataPegawai = {
    {"123", "John Doe", 30, 5000},
    {"456", "Jane Smith", 35, 6000},
    {"789", "Alice Johnson", 28, 4500},
    {"234", "Bob Williams", 32, 5500},
    {"567", "Emily Davis", 40, 7000}
};
```

Latihan 4:

Kesimpulan :

Dalam percobaan ini, implementasi Quick Sort dan Merge Sort berhasil dilakukan pada data Pegawai dengan kemampuan untuk memilih metode pengurutan (berdasarkan NIP atau Nama) serta urutan pengurutan (naik atau turun), memberikan fleksibilitas dan efisiensi dalam proses pengurutan data.