Praktikum Algoritma Struktur Data

Praktikum 08: Quick Sort dan Merge Sort

Oleh:

Bintang Bimantara/ 5223600025

Program Studi Sarjana Terapan Teknologi Game

Departemen Teknologi Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2024

Percobaan

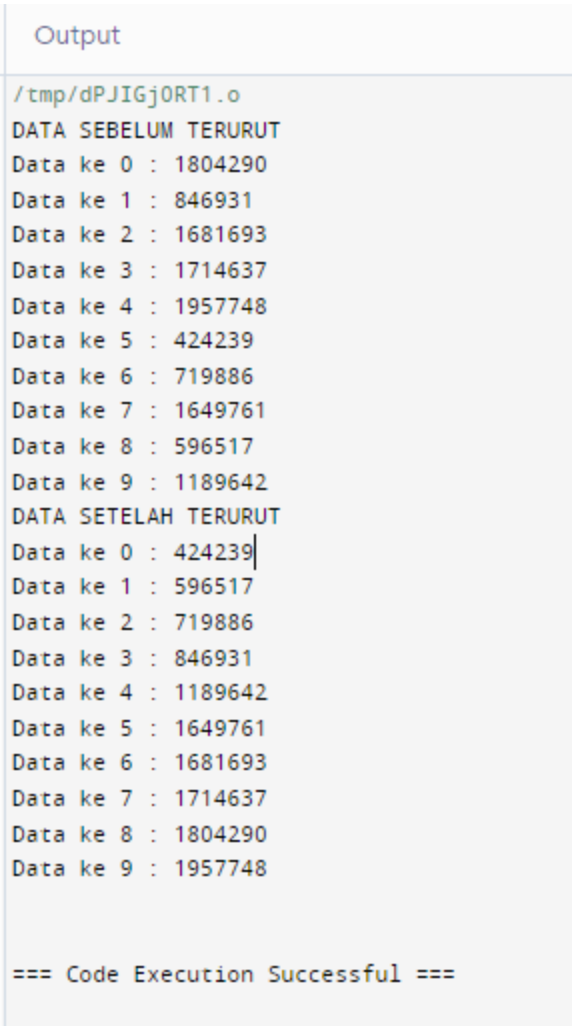
1. Percobaan 1 : Implementasi pengurutan dengan metode *Quick Sort* non rekursif.

```
#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
void QuickSortNonRekursif()
{
    struct tump {
        int Kiri;
        int Kanan;
    } Tumpukan[MaxStack];
    int i, j, L, R, x, ujung = 1;
    Tumpukan[1].Kiri = 0;
    Tumpukan[1].Kanan = MAX - 1;
    while (ujung != 0)
```

```
30          L = Tumpukan[ujung].Kiri;
31          R = Tumpukan[ujung].Kanan;
32          ujung--;
33          while (R > L)
34          {
35              i = L;
36              j = R;
37              x = Data[(L + R) / 2];
38              while (i <= j)
39              {
40                  while (Data[i] < x)
41                      i++;
42                  while (x < Data[j])
43                      j--;
44                  if (i <= j)
45                  {
46                      Tukar(&Data[i], &Data[j]);
47                      i++;
48                      j--;
49                  }
50              }
51              if (L < i)
52              {
53                  ujung++;
54                  Tumpukan[ujung].Kiri = i;
55                  Tumpukan[ujung].Kanan = R;
56              }
```

```
            R = j;
        }
    }
}

int main()
{
    int i;
    srand(0);
    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT" << endl;
    for (i = 0; i < MAX; i++)
    {
        Data[i] = rand() / 1000 + 1;
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }
    QuickSortNonRekursif();
    // Data setelah terurut
    cout << "DATA SETELAH TERURUT" << endl;
    for (i = 0; i < MAX; i++)
    {
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }
    return 0;
}
```

Hasil Output:



2. Percobaan 2 : Implementasi pengurutan dengan metode penyisipan Quick Sort rekursif.

```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
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);
}

```

```
43  int main()
44  {
45      int i;
46      srand(0);
47      // Membangkitkan bilangan acak
48      cout << "DATA SEBELUM TERURUT" << endl;
49      for (i = 0; i < MAX; i++)
50      {
51          Data[i] = rand() / 1000 + 1;
52          cout << "Data ke " << i << " : " << Data[i] << endl;
53      }
54      QuickSortRekursif(0, MAX - 1);
55      // Data setelah terurut
56      cout << "DATA SETELAH TERURUT" << endl;
57      for (i = 0; i < MAX; i++)
58      {
59          cout << "Data ke " << i << " : " << Data[i] << endl;
60      }
61      return 0;
62  }
63
```

Hasil Output:

```
Output
/tmp/2oBXmaFDHz.o
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 ===
```

3. Percobaan 3 : Implementasi pengurutan dengan metode penyisipan Merge Sort.

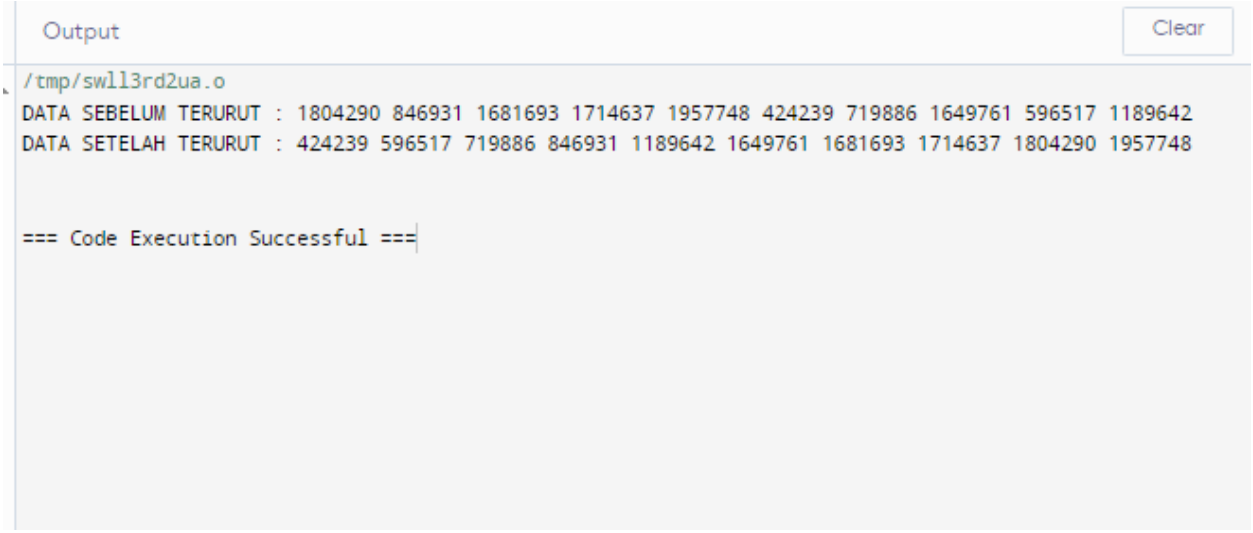
```cpp
#include <iostream>
#include <cstdlib>
#define MAX 10
using namespace std;
//Nama: Bintang Bimantara
//NRP: 5223600025

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

```

```cpp
// 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)
    {
```

```
41          temp[tmp_pos] = Data[tengah];
42          tengah = tengah + 1;
43          tmp_pos = tmp_pos + 1;
44      }
45      for (i = 0; i < num_elements; i++)
46      {
47          Data[kanan] = temp[kanan];
48          kanan = kanan - 1;
49      }
50  }
51
52  // Prosedur membuat kumpulan data
53  void m_sort(int Data[], int temp[], int kiri, int kanan)
54  {
55      int tengah;
56      if (kanan > kiri)
57      {
58          tengah = (kanan + kiri) / 2;
59          m_sort(Data, temp, kiri, tengah);
60          m_sort(Data, temp, tengah + 1, kanan);
61          merge(Data, temp, kiri, tengah + 1, kanan);
62      }
63  }
64
65  void mergeSort(int Data[], int temp[], int array_size)
66  {
67      m_sort(Data, temp, 0, array_size - 1);
68  }
69
```

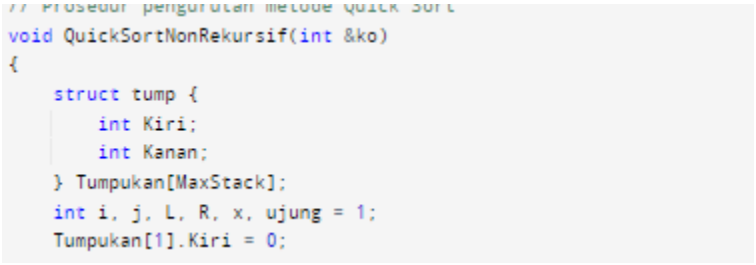
```
70  int main()
71  {
72      int i;
73      //pembangkit bilangan random
74      srand(0);
75      //membangkitkan bilangan integer random
76      cout << "DATA SEBELUM TERURUT : ";
77      for (i = 0; i < MAX; i++)
78      {
79          Data[i] = rand() / 1000 + 1;
80          cout << Data[i] << " ";
81      }
82      mergeSort(Data, temp, MAX);
83      cout << "\nDATA SETELAH TERURUT : ";
84      for (i = 0; i < MAX; i++)
85          cout << Data[i] << " ";
86      cout << endl;
87  }
88
```

Hasil Output:

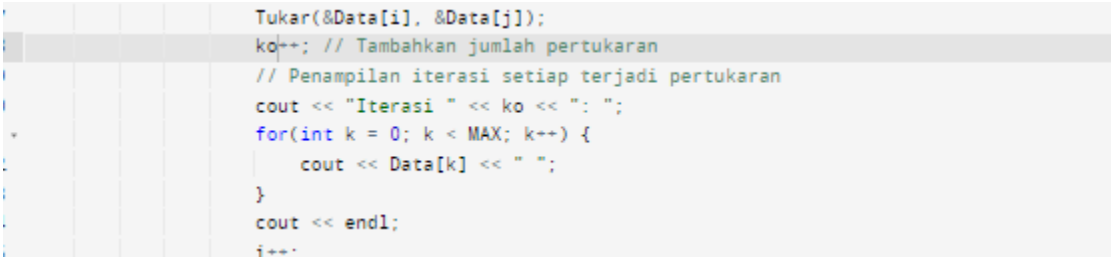


Latihan :

1. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan pada Quick Sort rekursif, Quick Sort non rekursif dan Merge Sort.



```
// Prosedur pengurutan metode Quick Sort
void QuickSortNonRekursif(int &ko)
{
    struct tump {
        int Kiri;
        int Kanan;
    } Tumpukan[MaxStack];
    int i, j, L, R, x, ujung = 1;
    Tumpukan[1].Kiri = 0;
```



```
                Tukar(&Data[i], &Data[j]);
                ko++; // Tambahkan jumlah pertukaran
                // Penampilan iterasi setiap terjadi pertukaran
                cout << "Iterasi " << ko << ": ";
                for(int k = 0; k < MAX; k++) {
                    cout << Data[k] << " ";
                }
                cout << endl;
                i++;
```

untuk menampilkan perubahan setiap iterasi dari proses pengurutan dalam penyisipan akan diberi kode ini yang akan menampilkan setiap iterasi yang terjadi dan menggunakan variabel “ko” sebagai patokan.

2. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma pengurutan penyisipan .

```
// Prosedur pengurutan metode Quick Sort
void QuickSortNonRekursif(int &jumlahPertukaran, int &jumlahPergeseran, int &jumlahPerbandingan)
{
    struct tump {
        int Kiri;
        int Kanan;
    } Tumpukan[MaxStack];
    int i, j, L, R, x, ujung = 1;
```

untuk menghitung berapa banyak perbandingan dan pergeseran maka pertama perlu mendeklarasi sebuah variabel untuk menghitung tersebut.

```
    x = Data[(L + R) / 2];
    while (i <= j)
    {
        jumlahPerbandingan++; // Tambahkan jumlah perbandingan
        while (Data[i] < x)
        {
            i++;
            jumlahPerbandingan++; // Tambahkan jumlah perbandingan
        }
        while (x < Data[j])
        {
            j--;
            jumlahPerbandingan++; // Tambahkan jumlah perbandingan
        }
        if (i <= j)
        {
            Tukar(&Data[i], &Data[j]);
```

```
    {
        Tukar(&Data[i], &Data[j]);
        jumlahPertukaran++; // Tambahkan jumlah pertukaran
        jumlahPergeseran += 2; // Tambahkan jumlah pergeseran (pertukaran dua elemen)
        // Penampilan iterasi setiap terjadi pertukaran
        cout << "Iterasi " << jumlahPertukaran << ": ";
        for(int k = 0; k < MAX; k++) {
            cout << Data[k] << " ";
        }
        cout << endl;
        i++;
        j--;
    }
```

Kemudian menambahkan isi dari variabel tersebut pada fungsi yang melakukan pengurutan.

```
int main()
{
    int i;
    int jumlahPertukaran = 0; // Variabel untuk melacak jumlah pertukaran
    int jumlahPergeseran = 0; // Variabel untuk melacak jumlah pergeseran
    int jumlahPerbandingan = 0; // Variabel untuk melacak jumlah perbandingan
    srand(0);
    // Membangkitkan bilangan acak
    cout << "DATA SEBELUM TERURUT" << endl;
    for (i = 0; i < MAX; i++)
    {
        Data[i] = rand() / 1000 + 1;
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }
    QuickSortNonRekursif(jumlahPertukaran, jumlahPergeseran, jumlahPerbandingan);
    // Data setelah terurut
    cout << "DATA SETELAH TERURUT" << endl;
    for (i = 0; i < MAX; i++)
    {
        cout << "Data ke " << i << " : " << Data[i] << endl;
    }
    cout << "Jumlah pergeseran: " << jumlahPergeseran << endl; // Menampilkan jumlah pergeseran
    cout << "Jumlah perbandingan: " << jumlahPerbandingan << endl; // Menampilkan jumlah perbandingan
    return 0;
}
```

Setelah itu tampilkan hasil variabel tersebut dengan cout pada int main.

3. Buatlah project baru untuk Latihan dan 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.

```cpp
#include <iostream>
#include <cstring>
#include <cctype>

using namespace std;

struct Pegawai {
    string NIP;
    string Nama;
    string Alamat;
    char Golongan;
};

const int MAX_PEGAWAI = 5;
Pegawai dataPegawai[MAX_PEGAWAI] = {
    {"ST01", "Suep", "Surabaya", 'A'},
    {"ST02", "Sari", "Malang", 'B'},
    {"ST05", "Agung", "Sidoarjo", 'B'},
    {"ST04", "Fathur", "Surabaya", 'O'},
    {"ST09", "Kholil", "Sidoarjo", 'A'}
};

void swap(Pegawai& a, Pegawai& b) {
    Pegawai temp = a;
    a = b;
    b = temp;
}
```

```
void quickSortNIP(Pegawai arr[], int left, int right, bool ascending) {
    int i = left, j = right;
    string pivot = arr[(left + right) / 2].NIP;

    while (i <= j) {
        while (ascending ? arr[i].NIP < pivot : arr[i].NIP > pivot)
            i++;
        while (ascending ? arr[j].NIP > pivot : arr[j].NIP < pivot)
            j--;

        if (i <= j) {
            swap(arr[i], arr[j]);
            i++;
            j--;
        }
    }

    if (left < j)
        quickSortNIP(arr, left, j, ascending);
    if (i < right)
        quickSortNIP(arr, i, right, ascending);
}

void quickSortNama(Pegawai arr[], int left, int right, bool ascending) {
    int i = left, j = right;
    string pivot = arr[(left + right) / 2].Nama;

    while (i <= j) {
        while (ascending ? arr[i].Nama < pivot : arr[i].Nama > pivot)
            i++;
        while (ascending ? arr[j].Nama > pivot : arr[j].Nama < pivot)
            j--;

        if (i <= j) {
```

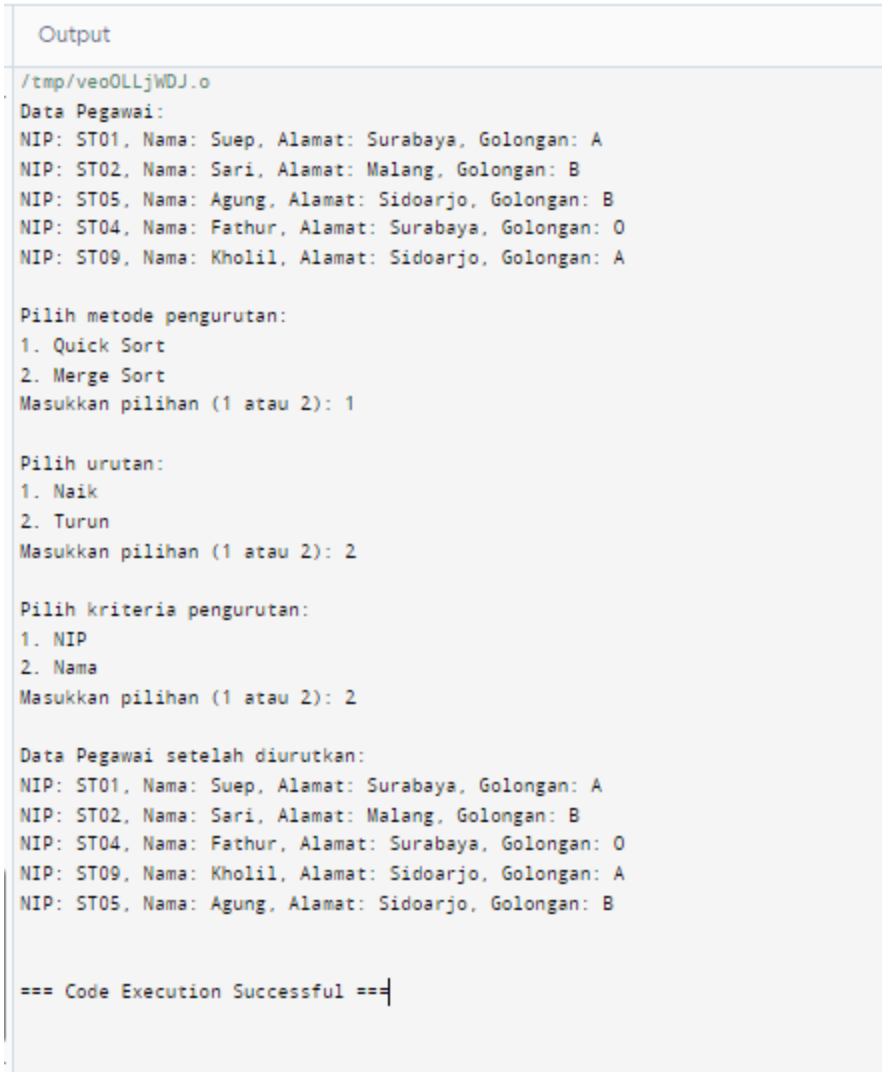
```
61
62          if (i <= j) {
63              swap(arr[i], arr[j]);
64              i++;
65              j--;
66          }
67      }
68
69      if (left < j)
70          quickSortNama(arr, left, j, ascending);
71      if (i < right)
72          quickSortNama(arr, i, right, ascending);
73  }
74
75  void merge(Pegawai temp[], int left, int mid, int right, bool ascending, bool byNIP) {
76      int i = left, j = mid + 1, k = left;
77
78      while (i <= mid && j <= right) {
79          if (byNIP) {
80              if (ascending ? temp[i].NIP < temp[j].NIP : temp[i].NIP > temp[j].NIP)
81                  dataPegawai[k++] = temp[i++];
82              else
83                  dataPegawai[k++] = temp[j++];
84          }
85          else {
86              if (ascending ? temp[i].Nama < temp[j].Nama : temp[i].Nama > temp[j].Nama)
87                  dataPegawai[k++] = temp[i++];
88              else
89                  dataPegawai[k++] = temp[j++];
90          }
91      }
92
93      while (i <= mid)
94          dataPegawai[k++] = temp[i++];
95
```

```
    while (j <= right)
        dataPegawai[k++] = temp[j++];
}

void mergeSortNIP(Pegawai temp[], int left, int right, bool ascending) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSortNIP(temp, left, mid, ascending);
        mergeSortNIP(temp, mid + 1, right, ascending);
        merge(temp, left, mid, right, ascending, true);
    }
}

void mergeSortNama(Pegawai temp[], int left, int right, bool ascending) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSortNama(temp, left, mid, ascending);
        mergeSortNama(temp, mid + 1, right, ascending);
        merge(temp, left, mid, right, ascending, false);
    }
}
```

```
int main() {
    int choice, sortBy;
    bool ascending;
    Pegawai temp[MAX_PEGAWAI];

    cout << "Data Pegawai:" << endl;
    for (int i = 0; i < MAX_PEGAWAI; i++) {
        cout << "NIP: " << dataPegawai[i].NIP << ", Nama: " << dataPegawai[i].Nama
            << ", Alamat: " << dataPegawai[i].Alamat << ", Golongan: " << dataPegawai[i].Golongan << endl;
    }

    cout << "\nPilih metode pengurutan:" << endl;
    cout << "1. Quick Sort" << endl;
    cout << "2. Merge Sort" << endl;
    cout << "Masukkan pilihan (1 atau 2): ";
    cin >> choice;

    if (choice != 1 && choice != 2) {
        cout << "Pilihan tidak valid!" << endl;
        return 0;
    }

    cout << "\nPilih urutan:" << endl;
    cout << "1. Naik" << endl;
    cout << "2. Turun" << endl;
    cout << "Masukkan pilihan (1 atau 2): ";
    cin >> choice;
    ascending = (choice == 1);

    cout << "\nPilih kriteria pengurutan:" << endl;
    cout << "1. NIP" << endl;
    cout << "2. Nama" << endl;
    cout << "Masukkan pilihan (1 atau 2): ";
    cin >> sortBy;

```

```
    for (int i = 0; i < MAX_PEGAWAI; i++)
        temp[i] = dataPegawai[i];

    if (sortBy == 1) {
        if (choice == 1)
            quickSortNIP(dataPegawai, 0, MAX_PEGAWAI - 1, ascending);
        else
            mergeSortNIP(temp, 0, MAX_PEGAWAI - 1, ascending);
    }
    else if (sortBy == 2) {
        if (choice == 1)
            quickSortNama(dataPegawai, 0, MAX_PEGAWAI - 1, ascending);
        else
            mergeSortNama(temp, 0, MAX_PEGAWAI - 1, ascending);
    }
    else {
        cout << "Pilihan tidak valid!" << endl;
        return 0;
    }

    cout << "\nData Pegawai setelah diurutkan:" << endl;
    for (int i = 0; i < MAX_PEGAWAI; i++) {
        cout << "NIP: " << dataPegawai[i].NIP << ", Nama: " << dataPegawai[i].Nama
            << ", Alamat: " << dataPegawai[i].Alamat << ", Golongan: " << dataPegawai[i].Golongan << endl;
    }

    return 0;
}
```

Hasil Output:



```
Output
/tmp/veoOLLjWDJ.o
Data Pegawai:
NIP: ST01, Nama: Suep, Alamat: Surabaya, Golongan: A
NIP: ST02, Nama: Sari, Alamat: Malang, Golongan: B
NIP: ST05, Nama: Agung, Alamat: Sidoarjo, Golongan: B
NIP: ST04, Nama: Fathur, Alamat: Surabaya, Golongan: O
NIP: ST09, Nama: Kholil, Alamat: Sidoarjo, Golongan: A

Pilih metode pengurutan:
1. Quick Sort
2. Merge Sort
Masukkan pilihan (1 atau 2): 1

Pilih urutan:
1. Naik
2. Turun
Masukkan pilihan (1 atau 2): 2

Pilih kriteria pengurutan:
1. NIP
2. Nama
Masukkan pilihan (1 atau 2): 2

Data Pegawai setelah diurutkan:
NIP: ST01, Nama: Suep, Alamat: Surabaya, Golongan: A
NIP: ST02, Nama: Sari, Alamat: Malang, Golongan: B
NIP: ST04, Nama: Fathur, Alamat: Surabaya, Golongan: O
NIP: ST09, Nama: Kholil, Alamat: Sidoarjo, Golongan: A
NIP: ST05, Nama: Agung, Alamat: Sidoarjo, Golongan: B


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

4. Berikan kesimpulan dari percobaan dan latihan yang telah Anda lakukan.

   Quick Sort dan Merge Sort adalah dua algoritma pengurutan yang efisien. Quick Sort membagi daftar menjadi dua bagian berdasarkan pivot, sementara Merge Sort membaginya menjadi dua bagian sama besar. Meskipun berbeda dalam pendekatannya, keduanya dapat digunakan untuk mengurutkan daftar besar dengan cepat.