PRAKTIKUM ALGORITMA DAN STUKTUR DATA

“Quick Sort, Merge Sort”

Oleh :

Fina Salsabila Pramudita (5223600006)

Program Studi Sarjana Terapan Teknologi Game

Departemen Teknologi Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2024

PERCOBAAN

1. Implementasi pengurutan dengan metode quick sort non rekursif

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#define MAX 10
#define MaxStack 10

int Array[MAX]; // Mengganti nama array dari Data menjadi Array
// Function to swap data
void Swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function for non-recursive quicksort
void QuickSortNonRecursive() {
    struct Stack {
        int left;
        int right;
    } Stack[MaxStack];

    int top = 0, i, j, L, R, pivot;
    Stack[top].left = 0;
    Stack[top].right = MAX - 1;

    while (top >= 0) {
        L = Stack[top].left;
        R = Stack[top].right;
        top--;

        while (R > L) {
```

```
            i = L;
            j = R;
            pivot = Array[(L + R) / 2]; // Mengganti nama array
dari Data menjadi Array

            while (i <= j) {
                while (Array[i] < pivot) i++; // Mengganti nama
array dari Data menjadi Array
                while (Array[j] > pivot) j--; // Mengganti nama
array dari Data menjadi Array
                if (i <= j) {
                    Swap(&Array[i], &Array[j]); // Mengganti nama
array dari Data menjadi Array
                    i++;
                    j--;
                }
            }

            if (L < j) {
                top++;
                Stack[top].left = L;
                Stack[top].right = j;
            }

            L = i;
        }
    }
}

int main() {
    srand(static_cast<unsigned>(time(nullptr)));

    std::cout << "DATA SEBELUM DIURUTKAN"; // Mengganti pesan
sebelum data diurutkan
    for (int i = 0; i < MAX; i++) {
```

```
        Array[i] = rand() % 100 + 1; // Mengganti nama array dari
Data menjadi Array

        std::cout << "\nArray[" << i << "]: " << Array[i]; //
Mengganti nama array dari Data menjadi Array

    }


    QuickSortNonRecursive();


    std::cout << "\n\nDATA SETELAH DIURUTKAN"; // Mengganti pesan
setelah data diurutkan

    for (int i = 0; i < MAX; i++) {

        std::cout << "\nArray[" << i << "]: " << Array[i]; //
Mengganti nama array dari Data menjadi Array

    }


    return 0;

}
```

2. Implementasi pengurutan dengan metode quick sort rekursif

```
#include <iostream>

#include <cstdlib>

#include <ctime>

#define MAX 10


int Array[MAX]; // Mengubah nama array dari Data menjadi Array

// Function to swap data

void Tukar(int *a, int *b) { // Mengubah nama fungsi Swap menjadi
Tukar

    int temp = *a;

    *a = *b;

    *b = temp;

}


// Function to perform QuickSort recursively

void QuickSortRekursif(int L, int R) { // Mengubah nama fungsi
QuickSortRecursive menjadi QuickSortRekursif

    int i = L, j = R;
```

```
    int pivot = Array[(L + R) / 2]; // Mengubah nama array dari
Data menjadi Array


    while (i <= j) {

        while (Array[i] < pivot) i++; // Mengubah nama array dari
Data menjadi Array

        while (Array[j] > pivot) j--; // Mengubah nama array dari
Data menjadi Array

        if (i <= j) {

            Tukar(&Array[i], &Array[j]); // Menggunakan nama
fungsi Tukar yang sudah diubah

            i++;

            j--;

        }

    }


    if (L < j) QuickSortRekursif(L, j); // Mengubah nama fungsi
QuickSortRecursive menjadi QuickSortRekursif

    if (i < R) QuickSortRekursif(i, R); // Mengubah nama fungsi
QuickSortRecursive menjadi QuickSortRekursif

}


int main() {

    srand(static_cast<unsigned>(time(nullptr))); // Menggunakan
waktu saat ini sebagai seed untuk generator bilangan acak


    std::cout << "DATA SEBELUM DIURUTKAN"; // Mengubah pesan
sebelum pengurutan

    for (int i = 0; i < MAX; i++) {

        Array[i] = rand() % 100 + 1; // Mengubah nama array dari
Data menjadi Array, serta mengubah cara menginisialisasi

        std::cout << "\nArray[" << i << "]: " << Array[i]; //
Mengubah nama array dari Data menjadi Array

    }


    QuickSortRekursif(0, MAX - 1); // Mengubah nama fungsi
QuickSortRecursive menjadi QuickSortRekursif
```

```
    std::cout << "\n\nDATA SETELAH DIURUTKAN"; // Mengubah pesan
setelah pengurutan
    for (int i = 0; i < MAX; i++) {
        std::cout << "\nArray[" << i << "]: " << Array[i]; //
Mengubah nama array dari Data menjadi Array
    }

    return 0;
}
```

3. Implementasi pengurutan dengan metode merge sort

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#define MAX 10

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

// Fungsi untuk menggabungkan array
void gabung(int Data[], int temp[], int kiri, int tengah, int
kanan) {
    int i, kiri_akhir, jumlah_elemen, pos_sementara;
    kiri_akhir = tengah - 1;
    pos_sementara = kiri;
    jumlah_elemen = kanan - kiri + 1;

    while ((kiri <= kiri_akhir) && (tengah <= kanan)) {
        if (Data[kiri] <= Data[tengah]) {
            temp[pos_sementara] = Data[kiri];
            pos_sementara++;
            kiri++;
        } else {
            temp[pos_sementara] = Data[tengah];
            pos_sementara++;
```

```
            tengah++;
        }
    }

    while (kiri <= kiri_akhir) {
        temp[pos_sementara] = Data[kiri];
        kiri++;
        pos_sementara++;
    }

    while (tengah <= kanan) {
        temp[pos_sementara] = Data[tengah];
        tengah++;
        pos_sementara++;
    }

    for (i = 0; i < jumlah_elemen; i++) {
        Data[kanan] = temp[kanan];
        kanan--;
    }
}

// Fungsi rekursif untuk mengurutkan data
void m_sort(int Data[], int temp[], int kiri, int kanan) {
    int tengah;
    if (kanan > kiri) {
        tengah = (kiri + kanan) / 2;
        m_sort(Data, temp, kiri, tengah);
        m_sort(Data, temp, tengah + 1, kanan);
        gabung(Data, temp, kiri, tengah + 1, kanan);
    }
}
```

```
// Antarmuka publik untuk merge sort
void mergeSort(int Data[], int temp[], int ukuran_array) {
    m_sort(Data, temp, 0, ukuran_array - 1);
}


int main() {
    srand(static_cast<unsigned int>(time(nullptr))); // Penentuan seed yang lebih baik


    std::cout << "\nDATA BELUM DIURUTKAN: ";
    for (int i = 0; i < MAX; i++) {
        Data[i] = rand() % 100 + 1; // Rentang data yang lebih masuk akal dan terlihat
        std::cout << Data[i] << " ";
    }


    mergeSort(Data, temp, MAX);


    std::cout << "\nDATA SUDAH DIURUTKAN: ";
    for (int i = 0; i < MAX; i++) {
        std::cout << Data[i] << " ";
    }
    std::cout << "\n";


    return 0;
}
```

LATIHAN

1. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan quick sort dan merge sort.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#define MAX 10
```

```
std::vector<int> Data(MAX);
std::vector<int> temp(MAX);


void tukar(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}


void cetakArray(const std::vector<int>& array, const std::string&
prefix) {
    std::cout << prefix;
    for (int num : array) {
        std::cout << num << " ";
    }
    std::cout << "\n";
}


void quickSort(int kiri, int kanan) {
    if (kiri >= kanan) return;
    int pivot = Data[(kiri + kanan) / 2];
    int i = kiri, j = kanan;

    while (i <= j) {
        while (Data[i] < pivot) i++;
        while (Data[j] > pivot) j--;
        if (i <= j) {
            tukar(&Data[i], &Data[j]);
            i++;
            j--;
        }
    }
```

```
        cetakArray(Data, "Quick Sort Iterasi: ");

        quickSort(kiri, j);
        quickSort(i, kanan);
    }

    void merge(int kiri, int tengah, int kanan) {
        int i = kiri, j = tengah, k = kiri;

        while (i <= tengah - 1 && j <= kanan) {
            if (Data[i] <= Data[j])
                temp[k++] = Data[i++];
            else
                temp[k++] = Data[j++];
        }

        while (i <= tengah - 1)
            temp[k++] = Data[i++];

        while (j <= kanan)
            temp[k++] = Data[j++];

        for (i = kiri; i <= kanan; i++)
            Data[i] = temp[i];

        cetakArray(Data, "Merge Sort Iterasi: ");
    }

    void mergeSort(int kiri, int kanan) {
        if (kanan > kiri) {
            int tengah = (kiri + kanan) / 2;
            mergeSort(kiri, tengah);
            mergeSort(tengah + 1, kanan);
```

```
            merge(kiri, tengah + 1, kanan);
        }
    }

    void inisialisasiData() {
        srand(static_cast<unsigned int>(time(nullptr)));
        for (int i = 0; i < MAX; i++) {
            Data[i] = rand() % 100 + 1;
        }
    }

    int main() {
        char ulangi;
        int pilihan;

        do {
            inisialisasiData();
            std::cout << "\nDATA BELUM DIURUTKAN: ";
            cetakArray(Data, "");

            std::cout << "Pilih metode penyortiran:\n";
            std::cout << "1. Quick Sort\n";
            std::cout << "2. Merge Sort\n";
            std::cout << "Pilihan Anda: ";
            std::cin >> pilihan;

            switch (pilihan) {
                case 1:
                    quickSort(0, MAX - 1);
                    break;
                case 2:
                    mergeSort(0, MAX - 1);
                    break;
```

```
            default:
                std::cout << "Pilihan tidak valid!\n";
                continue;
        }

        std::cout << "\nDATA TELAH DIURUTKAN: ";
        cetakArray(Data, "");

        std::cout << "Ulangi program? (y/n): ";
        std::cin >> ulangi;
    } while (ulangi == 'y' || ulangi == 'Y');

    return 0;
}
```

2. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma quick sort dan merge sort.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#define MAX 10

std::vector<int> Data(MAX);
std::vector<int> temp(MAX);

int perbandingan, perpindahan;

void tukar(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
    perpindahan++;
}
```

```
void cetakArray(const std::vector<int>& array, const std::string&
prefix) {
    std::cout << prefix;
    for (int num : array) {
        std::cout << num << " ";
    }
    std::cout << "\n";
}

void bubbleSort() {
    bool swapped;
    for (int i = 0; i < MAX - 1; i++) {
        swapped = false;
        for (int j = 0; j < MAX - i - 1; j++) {
            perbandingan++;
            if (Data[j] > Data[j + 1]) {
                tukar(&Data[j], &Data[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

void quickSort(int kiri, int kanan) {
    if (kiri >= kanan) return;
    int pivot = Data[(kiri + kanan) / 2];
    int i = kiri, j = kanan;

    while (i <= j) {
        while (Data[i] < pivot) { i++; perbandingan++; }
        while (Data[j] > pivot) { j--; perbandingan++; }
        if (i <= j) {
            tukar(&Data[i], &Data[j]);
```

```
                i++;
                j--;
            }
        }

        quickSort(kiri, j);
        quickSort(i, kanan);
    }

    void merge(int kiri, int tengah, int kanan) {
        int i = kiri, j = tengah, k = kiri;

        while (i <= tengah - 1 && j <= kanan) {
            perbandingan++;
            if (Data[i] <= Data[j]) {
                temp[k++] = Data[i++];
            } else {
                temp[k++] = Data[j++];
            }
            perpindahan++;
        }

        while (i <= tengah - 1) {
            temp[k++] = Data[i++];
            perpindahan++;
        }

        while (j <= kanan) {
            temp[k++] = Data[j++];
            perpindahan++;
        }

        for (i = kiri; i <= kanan; i++) {
```

```
            Data[i] = temp[i];
            perpindahan++;
        }
    }

    void mergeSort(int kiri, int kanan) {
        if (kanan > kiri) {
            int tengah = (kiri + kanan) / 2;
            mergeSort(kiri, tengah);
            mergeSort(tengah + 1, kanan);
            merge(kiri, tengah + 1, kanan);
        }
    }

    void inisialisasiData() {
        srand(static_cast<unsigned int>(time(nullptr)));
        for (int i = 0; i < MAX; i++) {
            Data[i] = rand() % 100 + 1;
        }
        perbandingan = 0;
        perpindahan = 0;
    }

    int main() {
        char ulangi;
        int pilihan;

        do {
            inisialisasiData();
            std::cout << "\nDATA BELUM DIURUTKAN: ";
            cetakArray(Data, "");

            std::cout << "Pilih metode pengurutan:\n";
```

```
            std::cout << "1. Quick Sort\n";
            std::cout << "2. Merge Sort\n";
            std::cout << "3. Bubble Sort\n";
            std::cout << "Pilihan Anda: ";
            std::cin >> pilihan;

            switch (pilihan) {
                case 1:
                    quickSort(0, MAX - 1);
                    break;
                case 2:
                    mergeSort(0, MAX - 1);
                    break;
                case 3:
                    bubbleSort();
                    break;
                default:
                    std::cout << "Pilihan tidak valid!\n";
                    continue;
            }

            std::cout << "\nDATA TELAH DIURUTKAN: ";
            cetakArray(Data, "");
            std::cout << "Jumlah perbandingan: " << perbandingan <<
"\n";
            std::cout << "Jumlah perpindahan: " << perpindahan <<
"\n";

            std::cout << "Ulangi program? (y/n): ";
            std::cin >> ulangi;
        } while (ulangi == 'y' || ulangi == 'Y');

        return 0;
    }
```

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.

```
#include <iostream>

#include <vector>

#include <string>

#include <algorithm> // Untuk std::swap


struct Karyawan {

std::string kode;

std::string nama;

};


std::vector<Karyawan> karyawan;


void cetakKaryawan() {

for (auto& k : karyawan) {

std::cout << "Kode: " << k.kode << ", Nama: " << k.nama <<
      "\n";

}

}


int partition(std::vector<Karyawan>& arr, int rendah, int
      tinggi, bool naik, bool menurutKode) {

Karyawan pivot = arr[tinggi];

int i = (rendah - 1);


for (int j = rendah; j <= tinggi - 1; j++) {

if (naik == (menurutKode ? (arr[j].kode < pivot.kode) :
      (arr[j].nama < pivot.nama))) {

i++;

std::swap(arr[i], arr[j]);

}
```

```
}
std::swap(arr[i + 1], arr[tinggi]);
return (i + 1);
}

void quickSort(std::vector<Karyawan>& arr, int rendah, int
      tinggi, bool naik, bool menurutKode) {
if (rendah < tinggi) {
int pi = partition(arr, rendah, tinggi, naik, menurutKode);

quickSort(arr, rendah, pi - 1, naik, menurutKode);
quickSort(arr, pi + 1, tinggi, naik, menurutKode);
}
}

void merge(std::vector<Karyawan>& arr, int kiri, int tengah,
      int kanan, bool naik, bool menurutKode) {
int i, j, k;
int n1 = tengah - kiri + 1;
int n2 = kanan - tengah;

std::vector<Karyawan> L(n1), R(n2);

for (i = 0; i < n1; i++)
L[i] = arr[kiri + i];
for (j = 0; j < n2; j++)
R[j] = arr[tengah + 1 + j];

i = 0; j = 0; k = kiri;
while (i < n1 && j < n2) {
if (naik == (menurutKode ? (L[i].kode <= R[j].kode) :
      (L[i].nama <= R[j].nama))) {
arr[k] = L[i];
i++;
} else {
```

```
arr[k] = R[j];
j++;
}
k++;
}

while (i < n1) {
arr[k] = L[i];
i++;
k++;
}

while (j < n2) {
arr[k] = R[j];
j++;
k++;
}
}

void mergeSort(std::vector<Karyawan>& arr, int kiri, int
      kanan, bool naik, bool menurutKode) {
if (kiri < kanan) {
int tengah = kiri + (kanan - kiri) / 2;

mergeSort(arr, kiri, tengah, naik, menurutKode);
mergeSort(arr, tengah + 1, kanan, naik, menurutKode);
merge(arr, kiri, tengah, kanan, naik, menurutKode);
}
}

int main() {
char ulangi;
int pilihan, metodeSort, urutanSort;
do {
```

```
    std::cout << "Masukkan data Karyawan (Kode dan Nama, ketik
          'selesai' untuk selesai):\n";
    std::string kode, nama;
    while (true) {
    std::cout << "Kode: ";
    std::getline(std::cin, kode);
    if (kode == "selesai") break;
    std::cout << "Nama: ";
    std::getline(std::cin, nama);
    karyawan.push_back({kode, nama});
    }

    std::cout << "Pilih metode penyusunan:\n";
    std::cout << "1. Pengurutan Cepat\n";
    std::cout << "2. Pengurutan Gabungan\n";
    std::cin >> metodeSort;
    std::cin.ignore();

    std::cout << "Pilih urutan penyusunan:\n";
    std::cout << "1. Menurut Kode\n";
    std::cout << "2. Menurut Nama\n";
    std::cin >> urutanSort;
    std::cin.ignore();

    std::cout << "Pilih arah penyusunan:\n";
    std::cout << "1. Naik (Ascending)\n";
    std::cout << "2. Turun (Descending)\n";
    int arah;
    std::cin >> arah;
    std::cin.ignore();

    bool naik = (arah == 1);
    bool menurutKode = (urutanSort == 1);
```

```
    std::cout << "\nData Karyawan sebelum disusun:\n";
    cetakKaryawan();

    if (metodeSort == 1) {
    quickSort(karyawan, 0, karyawan.size() - 1, naik,
          menurutKode);
    } else {
    mergeSort(karyawan, 0, karyawan.size() - 1, naik,
          menurutKode);
    }

    std::cout << "\nData Karyawan setelah disusun:\n";
    cetakKaryawan();

    std::cout << "\nUlangi program? (y/n): ";
    std::cin >> ulangi;
    std::cin.ignore();

    karyawan.clear();  // Kosongkan vektor untuk input baru jika
          diulang
    } while (ulangi == 'y' || ulangi == 'Y');

    return 0;
    }
```

4. Mkk
   - Bubble Sort:
     Bubble Sort adalah algoritma pengurutan sederhana yang secara berulang membandingkan pasangan elemen berturut-turut dan menukar mereka jika mereka berada dalam urutan yang salah.
     Kelebihan:
     - Mudah dipahami dan diimplementasikan.
     - Cocok untuk daftar yang sudah hampir terurut.

     Kekurangan:
     - Kurang efisien untuk daftar yang besar karena memiliki kompleksitas waktu yang tinggi (O(n^2)).
     - Tidak efisien untuk daftar dengan elemen yang sudah terurut secara terbalik.
     - Jumlah perbandingan dan pergeseran cenderung tinggi, terutama untuk daftar besar.

- Merge Sort:
  Merge Sort adalah algoritma pengurutan yang menggunakan teknik divide and conquer dengan cara membagi daftar menjadi sub-daftar yang lebih kecil, mengurutkan masing-masing sub-daftar secara terpisah, dan kemudian menggabungkannya kembali.
  Kelebihan:
  - Sangat efisien untuk daftar yang besar karena memiliki kompleksitas waktu yang stabil (O(n log n)).
  - Tidak terpengaruh oleh keadaan awal daftar dan cocok untuk semua jenis data.

  Kekurangan:
  - Membutuhkan alokasi memori tambahan untuk array sementara saat penggabungan.
  - Lebih kompleks untuk diimplementasikan dibandingkan dengan Bubble Sort.