ALGORITMA DAN STUKTUR DATA

“Bubble Sort, Shell 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 gelembung (bubble sort)

```
#include <iostream>
#include <cstdlib>
#include <ctime>

#define JUMLAH_DATA 10 // Menggunakan nama variabel JUMLAH_DATA
untuk menyatakan jumlah data

int Data[JUMLAH_DATA]; // Mengubah variabel MAX menjadi
JUMLAH_DATA untuk menyatakan jumlah data

// Fungsi untuk pertukaran data
void Tukar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Fungsi untuk melakukan pengurutan menggunakan Bubble Sort
void UrutBubble() {
    for(int i = 1; i < JUMLAH_DATA; i++) {
        for(int j = JUMLAH_DATA - 1; j >= i; j--) {
            if(Data[j-1] > Data[j]) {
                Tukar(&Data[j-1], &Data[j]); // Memanggil fungsi
Tukar untuk menukar data
            }
        }
    }
}

int main() {
    std::srand(static_cast<unsigned int>(std::time(nullptr))); //
Menetapkan seed random dengan waktu saat ini untuk mendapatkan
keacakan yang lebih baik

    // Membangkitkan bilangan acak
    std::cout << "DATA SEBELUM DIURUTKAN";
    for(int i = 0; i < JUMLAH_DATA; i++) {
```

```
        Data[i] = std::rand() % 100 + 1; // Menghasilkan bilangan
acak antara 1 dan 100
        std::cout << "\nData ke " << i << " : " << Data[i];
    }

    UrutBubble(); // Melakukan pengurutan menggunakan Bubble Sort

    // Menampilkan data setelah diurutkan
    std::cout << "\nDATA SETELAH DIURUTKAN";
    for(int i = 0; i < JUMLAH_DATA; i++) {
        std::cout << "\nData ke " << i << " : " << Data[i];
    }

    return 0;
}
```

2. Implementasi pengurutan dengan metode shell (shell sort)

```
#include <iostream>
#include <cstdlib>
#include <ctime>

#define JUMLAH_DATA 10 // Menentukan jumlah data yang akan
diurutkan
int Data[JUMLAH_DATA];

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

// Fungsi untuk melakukan pengurutan menggunakan Shell sort
void UrutShell() {
    int celah, i, j;
    bool tukar;
    celah = JUMLAH_DATA;

    while (celah > 1) {
        celah = celah / 2;
```

```
            tukar = false;

            while (!tukar) {
                tukar = true;
                for (j = 0; j < JUMLAH_DATA - celah; j++) {
                    i = j + celah;
                    if (Data[j] > Data[i]) {
                        Tukar(&Data[j], &Data[i]);
                        tukar = false;
                    }
                }
            }
        }
    }

    int main() {
        std::srand(static_cast<unsigned int>(std::time(nullptr))); //
    Menetapkan seed random dengan waktu saat ini untuk mendapatkan
    keacakan yang lebih baik

        // Membangkitkan bilangan acak
        std::cout << "DATA SEBELUM DIURUTKAN";
        for (int i = 0; i < JUMLAH_DATA; i++) {
            Data[i] = std::rand() % 100 + 1; // Menghasilkan bilangan
    acak antara 1 dan 100
            std::cout << "\nData ke " << i << " : " << Data[i];
        }

        UrutShell(); // Melakukan pengurutan menggunakan Shell sort

        // Menampilkan data setelah diurutkan
        std::cout << "\nDATA SETELAH DIURUTKAN";
        for (int i = 0; i < JUMLAH_DATA; i++) {
            std::cout << "\nData ke " << i << " : " << Data[i];
        }

        return 0;
    }
```

LATIHAN

1. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan metode gelembung dan shell.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>

#define JUMLAH_DATA 10 // Menentukan jumlah data yang akan
diurutkan

std::vector<int> data(JUMLAH_DATA);

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

// Fungsi untuk melakukan Bubble Sort dengan menampilkan setiap
iterasi
void UrutBubble() {
    bool tukar;
    for (int i = 0; i < JUMLAH_DATA - 1; i++) {
        tukar = false;
        for (int j = 0; j < JUMLAH_DATA - i - 1; j++) {
            if (data[j] > data[j + 1]) {
                Tukar(&data[j], &data[j + 1]);
                tukar = true;
            }
        }
        // Menampilkan status array setelah setiap iterasi
        std::cout << "Setelah iterasi " << i + 1 << ": ";
        for (int k = 0; k < JUMLAH_DATA; k++) {
            std::cout << data[k] << " ";
        }
        std::cout << "\n";
        if (!tukar)
```

```
            break; // Tidak ada pertukaran berarti array sudah
terurut
        }
    }

    // Fungsi untuk melakukan Shell Sort dengan menampilkan setiap
    iterasi
    void UrutShell() {
        int celah, i, j;
        bool tukar;

        for (celah = JUMLAH_DATA / 2; celah > 0; celah /= 2) {
            do {
                tukar = false;
                for (j = 0; j < JUMLAH_DATA - celah; j++) {
                    i = j + celah;
                    if (data[j] > data[i]) {
                        Tukar(&data[j], &data[i]);
                        tukar = true;
                    }
                }
                // Menampilkan status array setelah setiap
    perbandingan celah
                std::cout << "Setelah celah " << celah << ": ";
                for (int k = 0; k < JUMLAH_DATA; k++) {
                    std::cout << data[k] << " ";
                }
                std::cout << "\n";
            } while (tukar);
        }
    }

    // Fungsi untuk menghasilkan data acak
    void BangkitkanDataAcak() {
        for (int i = 0; i < JUMLAH_DATA; i++) {
            data[i] = std::rand() % 100 + 1;
        }
    }
```

```
// Fungsi untuk menampilkan data awal
void TampilkanData() {
    std::cout << "Data sebelum diurutkan: ";
    for (int i = 0; i < JUMLAH_DATA; i++) {
        std::cout << data[i] << " ";
    }
    std::cout << "\n";
}

int main() {
    int pilihan;
    bool berjalan = true;

    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    while (berjalan) {
        std::cout << "\nMenu:\n1. Bubble Sort\n2. Shell Sort\n3.
Keluar\nPilih (1-3): ";
        std::cin >> pilihan;

        if (pilihan == 1 || pilihan == 2) {
            BangkitkanDataAcak();
            TampilkanData();

            if (pilihan == 1) {
                std::cout << "Melakukan Bubble Sort...\n";
                UrutBubble();
            } else if (pilihan == 2) {
                std::cout << "Melakukan Shell Sort...\n";
                UrutShell();
            }

            std::cout << "Data setelah diurutkan: ";
            for (int i = 0; i < JUMLAH_DATA; i++) {
                std::cout << data[i] << " ";
            }
            std::cout << "\n";
        } else if (pilihan == 3) {
            berjalan = false; // Keluar dari loop
```

```
            } else {
                std::cout << "Pilihan tidak valid, silakan coba
    lagi.\n";
            }
        }

        return 0;
    }
```

2. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma gelembung dan shell

```
    #include <iostream>
    #include <cstdlib>
    #include <ctime>
    #include <vector>

    #define JUMLAH_DATA 10

    std::vector<int> data(JUMLAH_DATA);

    // Function to swap data
    void Tukar(int *a, int *b, int& tukar) {
        int temp = *a;
        *a = *b;
        *b = temp;
        tukar++;
    }

    // Function to perform Bubble Sort with display at each iteration
    void UrutBubble() {
        bool ditukar;
        int perbandingan = 0, tukar = 0;
        for (int i = 0; i < JUMLAH_DATA - 1; i++) {
            ditukar = false;
            for (int j = 0; j < JUMLAH_DATA - i - 1; j++) {
                perbandingan++;
                if (data[j] > data[j + 1]) {
                    Tukar(&data[j], &data[j + 1], tukar);
                    ditukar = true;
```

```
            }
        }
        // Menampilkan status array setelah setiap iterasi
        std::cout << "Setelah iterasi " << i + 1 << ": ";
        for (int k = 0; k < JUMLAH_DATA; k++) {
            std::cout << data[k] << " ";
        }
        std::cout << "\n";
        if (!ditukar)
            break; // Tidak ada pertukaran berarti array sudah
terurut
    }
    std::cout << "Total perbandingan: " << perbandingan << ",
Total pertukaran: " << tukar << "\n";
}

// Function to perform Shell Sort with display at each iteration
void UrutShell() {
    int celah, i, j, perbandingan = 0, tukar = 0;
    for (celah = JUMLAH_DATA / 2; celah > 0; celah /= 2) {
        bool ditukar;
        do {
            ditukar = false;
            for (j = 0; j < JUMLAH_DATA - celah; j++) {
                perbandingan++;
                i = j + celah;
                if (data[j] > data[i]) {
                    Tukar(&data[j], &data[i], tukar);
                    ditukar = true;
                }
            }
            // Menampilkan status array setelah setiap
perbandingan celah
            std::cout << "Setelah celah " << celah << ": ";
            for (int k = 0; k < JUMLAH_DATA; k++) {
                std::cout << data[k] << " ";
            }
            std::cout << "\n";
        } while (ditukar);
```

```
    }
    std::cout << "Total perbandingan: " << perbandingan << ",
Total pertukaran: " << tukar << "\n";
}

// Function to generate random data
void BangkitkanDataAcak() {
    for (int i = 0; i < JUMLAH_DATA; i++) {
        data[i] = std::rand() % 100 + 1;
    }
}

// Function to display the initial data
void TampilkanData() {
    std::cout << "Data sebelum terurut: ";
    for (int i = 0; i < JUMLAH_DATA; i++) {
        std::cout << data[i] << " ";
    }
    std::cout << "\n";
}

int main() {
    int pilihan;
    bool berjalan = true;

    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    while (berjalan) {
        std::cout << "\nMenu:\n1. Urut Bubble\n2. Urut Shell\n3.
Keluar\nPilih (1-3): ";
        std::cin >> pilihan;

        if (pilihan == 1 || pilihan == 2) {
            BangkitkanDataAcak();
            TampilkanData();

            if (pilihan == 1) {
                std::cout << "Melakukan Urut Bubble...\n";
                UrutBubble();
```

```
            } else if (pilihan == 2) {
                std::cout << "Melakukan Urut Shell...\n";
                UrutShell();
            }

            std::cout << "Data setelah terurut: ";
            for (int i = 0; i < JUMLAH_DATA; i++) {
                std::cout << data[i] << " ";
            }
            std::cout << "\n";
        } else if (pilihan == 3) {
            berjalan = false; // Keluar dari loop
        } else {
            std::cout << "Pilihan tidak valid, silakan coba
lagi.\n";
        }
    }

    return 0;
}
```

3. Tambahkan pada project Latihan pada praktikum 7 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.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

struct Pegawai {
    std::string nama;
    int id;
};

// Prototipe fungsi
```

```
void urutkanPegawaiBubble(std::vector<Pegawai>& pegawai, bool
berdasarkanID, bool naik);
void urutkanPegawaiShell(std::vector<Pegawai>& pegawai, bool
berdasarkanID, bool naik);
void tukarPegawai(Pegawai* a, Pegawai* b);
void tampilkanPegawai(const std::vector<Pegawai>& pegawai);

int main() {
    // Data contoh
    std::vector<Pegawai> pegawai = {
        {"John Doe", 112233},
        {"Jane Smith", 223344},
        {"Alice Johnson", 334455},
        {"Bob Brown", 445566}
    };

    int pilihan, pilihanUrut, pilihanOrder;
    bool berjalan = true;

    while (berjalan) {
        std::cout << "\nMenu:\n1. Urutkan dengan Bubble
Sort\n2. Urutkan dengan Shell Sort\n3. Keluar\nPilih metode
pengurutan (1-3): ";
        std::cin >> pilihan;

        if (pilihan == 3) {
            berjalan = false;
            continue;
        }

        std::cout << "Pilih atribut pengurutan:\n1.
Berdasarkan ID\n2. Berdasarkan Nama\nPilih opsi (1-2): ";
        std::cin >> pilihanUrut;

        std::cout << "Pilih urutan:\n1. Naik\n2. Turun\nPilih
urutan (1-2): ";
        std::cin >> pilihanOrder;

        bool berdasarkanID = pilihanUrut == 1;
```

```
            bool naik = pilihanOrder == 1;

            switch (pilihan) {
                case 1:
                    urutkanPegawaiBubble(pegawai, berdasarkanID,
    naik);
                    break;
                case 2:
                    urutkanPegawaiShell(pegawai, berdasarkanID,
    naik);
                    break;
                default:
                    std::cout << "Pilihan tidak valid, silakan
    coba lagi.\n";
                    continue;
            }

            std::cout << "Data setelah diurutkan:\n";
            tampilkanPegawai(pegawai);
        }

        return 0;
    }

    void urutkanPegawaiBubble(std::vector<Pegawai>& pegawai, bool
    berdasarkanID, bool naik) {
        int n = pegawai.size();
        bool ditukar;
        for (int i = 0; i < n - 1; i++) {
            ditukar = false;
            for (int j = 0; j < n - i - 1; j++) {
                bool kondisi = berdasarkanID ? (pegawai[j].id >
    pegawai[j + 1].id) : (pegawai[j].nama > pegawai[j + 1].nama);
                if (naik ? kondisi : !kondisi) {
                    tukarPegawai(&pegawai[j], &pegawai[j + 1]);
                    ditukar = true;
                }
            }
            if (!ditukar)
```

```
            break;
        }
    }

    void urutkanPegawaiShell(std::vector<Pegawai>& pegawai, bool
    berdasarkanID, bool naik) {
        int n = pegawai.size();
        for (int gap = n / 2; gap > 0; gap /= 2) {
            for (int i = gap; i < n; i++) {
                Pegawai temp = pegawai[i];
                int j;
                for (j = i; j >= gap; j -= gap) {
                    bool kondisi = berdasarkanID ? (pegawai[j -
    gap].id > temp.id) : (pegawai[j - gap].nama > temp.nama);
                    if (naik ? kondisi : !kondisi) {
                        pegawai[j] = pegawai[j - gap];
                    } else {
                        break;
                    }
                }
                pegawai[j] = temp;
            }
        }
    }

    void tukarPegawai(Pegawai* a, Pegawai* b) {
        Pegawai temp = *a;
        *a = *b;
        *b = temp;
    }

    void tampilkanPegawai(const std::vector<Pegawai>& pegawai) {
        for (const auto& p : pegawai) {
            std::cout << "ID: " << p.id << ", Nama: " << p.nama <<
    "\n";
        }
    }
```

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

- Bubble Sort

  Bubble Sort adalah algoritma pengurutan sederhana yang secara berulang membandingkan setiap pasangan elemen berturut-turut dan menukar mereka jika mereka berada dalam urutan yang salah.

  - Kelebihan:

    Sederhana untuk dipahami dan diimplementasikan.

    Cocok untuk digunakan pada jumlah data yang kecil atau hampir terurut.
  - Kekurangan:

    Tidak efisien untuk jumlah data yang besar karena memiliki kompleksitas waktU

    Memerlukan banyak perbandingan dan pertukaran bahkan jika data sudah terurut.

- Shell Sort

  Shell Sort adalah pengembangan dari insertion sort yang lebih cepat, di mana data diurutkan dengan membandingkan elemen yang berjarak tertentu jauhnya.

  - Kelebihan:

    Lebih efisien daripada Bubble Sort, terutama pada jumlah data yang besar.

    Meskipun bukan yang tercepat, Shell Sort merupakan algoritma pengurutan yang relatif mudah diimplementasikan.
  - Kekurangan:

    Meskipun lebih cepat daripada Bubble Sort, Shell Sort masih kalah cepat dibandingkan dengan algoritma pengurutan yang lebih canggih seperti Quick Sort atau Merge Sort.

    Performanya dapat bervariasi tergantung pada pilihan penyesuaian gap, yang merupakan tantangan dalam mengoptimalkan algoritmanya.