MAKALAH

TENTANG SORTING

Disusun oleh :

Nama	: L Hafid Alkhair
NIM	: 2023903430060
Kelas	: TRKJ 1.C
Jurusan	: Teknologi Informasi dan Komputer
Program Studi	: Teknologi Rekayasa Komputer Jaringan
Dosen Pengajar	: Indrawati, SST. MT

JURUSAN TEKNOLOGI INFORMASI KOMPUTER

PRODI TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEUMAWE

TAHUN AJARAN 2023/2024

DAFTAR ISI

BAB I
PEMBAHASAN

A. DEFENISI SORTING

Dalam konteks bahasa pemrograman C, "sorting" mengacu pada proses pengurutan elemen-elemen dalam sebuah array atau struktur data berdasarkan kriteria tertentu. Tujuan utama dari pengurutan adalah untuk menyusun elemen-elemen data sehingga dapat diakses atau dicari lebih efisien.

Definisi umum untuk "sorting" dalam bahasa C dapat dijelaskan sebagai berikut:

Sorting (Pengurutan):

Pengurutan adalah proses menyusun elemen-elemen data dalam suatu struktur data, seperti array, secara teratur berdasarkan kriteria tertentu. Kriteria pengurutan dapat berupa urutan numerik (ascending atau descending) atau berdasarkan aturan khusus lainnya. Pengurutan dilakukan untuk mempermudah pencarian, pengambilan data terurut, dan peningkatan efisiensi akses data.

Dalam implementasi bahasa C, algoritma pengurutan seperti Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, dan Quick Sort sering digunakan untuk mengatur urutan elemen-elemen dalam array atau struktur data lainnya. Proses pengurutan memerlukan perbandingan dan pertukaran elemen-elemen data sehingga mereka dapat diatur sesuai dengan kriteria yang diinginkan.

B. Pentingnya Sorting Dalam Pengembangan Perangkat Lunak

Sorting memiliki peran krusial dalam pengembangan perangkat lunak dan sistem komputer. Proses pengurutan memungkinkan implementasi algoritma pencarian yang lebih efisien, memperbolehkan penggunaan metode seperti binary search untuk pencarian yang cepat pada data yang terurut. Selain itu, sorting juga mengoptimalkan berbagai algoritma dan operasi pada struktur data,

seperti merge pada merge sort, yang menjadi lebih efisien ketika menerima input yang sudah diurutkan.

Efisiensi pengaksesan dan manipulasi data juga meningkat dengan data yang diurutkan, menghasilkan kinerja sistem yang lebih baik. Dalam banyak kasus, data yang diurutkan mempermudah penyusunan, pemahaman laporan, dan tampilan visual lainnya. Selain itu, sorting dapat dioptimalkan untuk proses pemrosesan data, seperti penggabungan antar tabel dalam basis data atau pengelompokan data.

Penerapan sorting juga memberikan manfaat dalam meningkatkan responsivitas aplikasi, memudahkan pengguna dalam menjelajahi dan memanipulasi data yang sudah diurutkan. Dalam konteks aplikasi real-time, sorting mendukung penyajian data dengan cepat dan akurat, yang esensial dalam pengambilan keputusan yang cepat. Pemilihan algoritma pengurutan yang sesuai dengan karakteristik data dan kebutuhan aplikasi menjadi kunci untuk mencapai keseimbangan optimal antara waktu eksekusi dan kebutuhan sumber daya.

C. Jenis Jenis Sorting

Dalam pengembangan perangkat lunak menggunakan bahasa pemrograman C, terdapat beberapa jenis algoritma sorting yang umumnya digunakan. Jenis-jenis sorting ini berperan dalam menyusun elemen-elemen data dalam suatu struktur data, seperti array, dengan berbagai metode. Berikut adalah beberapa jenis sorting yang umum digunakan:

a) Bubble Sort

Bubble Sort adalah salah satu algoritma sorting yang sederhana dan mudah dipahami. Prinsip dasar Bubble Sort adalah dengan secara berulang membandingkan dan menukar pasangan elemen yang berdekatan dalam array hingga seluruh array terurut. Pada setiap iterasi, elemen-elemen dibandingkan, dan jika ditemukan bahwa elemen ke-i

lebih besar dari elemen ke-(i+1), keduanya ditukar. Proses ini berlanjut hingga tidak ada lagi pertukaran yang perlu dilakukan, menandakan bahwa array sudah terurut.

Kelebihan Bubble Sort:

- **Sederhana dan Mudah Dipahami:** Implementasi Bubble Sort sangat sederhana dan mudah dipahami, cocok untuk pemahaman dasar tentang algoritma sorting bagi pemula.
- **Membutuhkan Sedikit Ruang Penyimpanan Tambahan:** Bubble Sort hanya memerlukan satu variabel tambahan untuk pertukaran, sehingga membutuhkan sedikit ruang penyimpanan tambahan.

Kekurangan Bubble Sort:

- **Inefisien pada Data Besar:** Bubble Sort memiliki kinerja yang kurang efisien pada data yang besar. Algoritma ini memiliki kompleksitas waktu rata-rata O(n^2), di mana n adalah jumlah elemen dalam array.
- **Performa Terbatas pada Kasus Terbaik:** Meskipun sederhana, Bubble Sort memiliki performa yang buruk pada kasus terbaik karena melakukan pertukaran pada setiap iterasi, bahkan jika array sudah terurut.
- **Tidak Efisien untuk Penggunaan Nyata:** Bubble Sort umumnya tidak digunakan dalam aplikasi dunia nyata karena ada algoritma sorting lain yang lebih efisien, terutama pada kasus data yang besar atau kompleks.

Penting untuk mempertimbangkan kelebihan dan kekurangan ini ketika memilih algoritma sorting untuk implementasi dalam bahasa C, terutama jika efisiensi waktu eksekusi adalah faktor penting dalam aplikasi yang dikembangkan.

b) Insertion Sort

Insertion Sort adalah algoritma sorting yang membagi array menjadi dua bagian: bagian yang sudah diurutkan dan bagian yang belum diurutkan. Algoritma ini memproses satu elemen pada satu waktu dari bagian yang belum diurutkan dan menyisipkannya ke dalam bagian yang sudah diurutkan. Selama proses ini, elemen-elemen yang lebih besar dari elemen yang sedang diproses bergeser ke posisi yang lebih tinggi dalam array.

Kelebihan Insertion Sort:

- **Sederhana dan Mudah Dipahami:** Insertion Sort adalah salah satu algoritma yang paling mudah dipahami dan diimplementasikan. Cocok untuk data yang sudah hampir terurut atau jumlah elemen yang sedikit.
- **Efisien untuk Data yang Hampir Terurut:** Jika data sudah hampir terurut, Insertion Sort dapat bekerja dengan sangat efisien. Algoritma ini memiliki kompleksitas waktu O(n) pada kasus terbaik.
- **Membutuhkan Sedikit Ruang Penyimpanan Tambahan:** Insertion Sort membutuhkan sedikit ruang penyimpanan tambahan, yaitu satu variabel tambahan untuk menyimpan nilai yang sedang diproses.

Kekurangan Insertion Sort:

- **Kurang Efisien untuk Data Besar:** Pada data yang besar, Insertion Sort menjadi kurang efisien karena memiliki kompleksitas waktu rata-rata dan terburuk O(n^2).
- **Tidak Cocok untuk Data Terbalik Urut:** Jika data terurut secara terbalik, Insertion Sort dapat menjadi kurang efisien karena setiap elemen baru harus disisipkan ke posisi awal array.

- **Kinerja Tergantung pada Urutan Data Awal:** Kinerja Insertion Sort sangat tergantung pada urutan awal data. Semakin terurut data, semakin baik performa Insertion Sort.

Meskipun Insertion Sort memiliki kelebihan pada kasus data yang hampir terurut atau jumlah elemen yang sedikit, dalam banyak kasus, algoritma sorting lain seperti Merge Sort atau Quick Sort lebih dipilih untuk kinerja yang lebih baik pada data yang lebih besar dan kasus umum.

c) Selection Sort

Selection Sort adalah algoritma sorting sederhana yang membagi array menjadi dua bagian: bagian yang sudah diurutkan dan bagian yang belum diurutkan. Algoritma ini secara berulang memilih elemen dengan nilai terkecil (pada implementasi ascending) atau terbesar (pada implementasi descending) dari bagian yang belum diurutkan dan menukarnya dengan elemen pertama dari bagian yang belum diurutkan. Proses ini diulang hingga seluruh array terurut.

Kelebihan Selection Sort:

- **Sederhana dan Mudah Dipahami:** Selection Sort adalah algoritma yang sangat sederhana dan mudah dipahami. Implementasinya melibatkan perulangan sederhana dan menukar elemen.

- **Tidak Memerlukan Ruang Penyimpanan Tambahan**: Selection Sort hanya memerlukan satu variabel tambahan untuk menyimpan nilai sementara selama pertukaran. Oleh karena itu, membutuhkan sedikit ruang penyimpanan tambahan.

Kekurangan Selection Sort:

- **Inefisien untuk Data Besar:** Selection Sort memiliki kompleksitas waktu rata-rata dan terburuk O(n^2), sehingga kurang efisien untuk data yang besar.
- **Kinerja Konstan pada Setiap Kasus:** Meskipun data sudah terurut atau hampir terurut, Selection Sort tetap melibatkan perbandingan dan pertukaran elemen, sehingga kinerjanya tetap konstan pada setiap kasus.
- **Tidak Stabil:** Selection Sort tidak stabil, yang berarti jika terdapat elemen-elemen dengan nilai yang sama, urutan relatif mereka dapat berubah setelah proses pengurutan.

Meskipun Selection Sort mudah dipahami dan digunakan untuk jumlah elemen yang kecil, algoritma sorting seperti Merge Sort atau Quick Sort lebih disukai untuk data yang lebih besar dan kasus umum karena memiliki kompleksitas waktu yang lebih baik.

d) Merge Sort

Merge Sort adalah algoritma sorting yang menggunakan pendekatan divide and conquer. Algoritma ini membagi array menjadi dua bagian, mengurutkan masing-masing bagian secara rekursif, dan kemudian menggabungkan kedua bagian yang sudah terurut menjadi satu array utuh. Proses penggabungan (merge) dilakukan dengan membandingkan elemen-elemen dari kedua bagian dan menempatkannya dengan urutan yang benar.

Kelebihan Merge Sort:

- **Stabilitas:** Merge Sort adalah algoritma stabil, yang berarti jika ada dua elemen dengan nilai yang sama, urutan relatif mereka akan tetap sama setelah pengurutan.
- **Efisien pada Data Besar:** Merge Sort memiliki kompleksitas waktu O(n log n), menjadikannya efisien pada data yang besar.

- **Kinerja Konsisten:** Kinerja Merge Sort tidak dipengaruhi oleh keadaan awal data. Algoritma ini memiliki kinerja yang konsisten pada setiap kasus, termasuk pada data yang hampir terurut atau terbalik urut.
- **Cocok untuk Struktur Data Terhubung:** Merge Sort tidak memerlukan akses acak ke elemen, sehingga cocok untuk struktur data terhubung seperti linked list.

Kekurangan Merge Sort:

- **Membutuhkan Ruang Tambahan:** Merge Sort memerlukan ruang tambahan yang setara dengan ukuran array yang akan diurutkan. Hal ini bisa menjadi kendala pada situasi di mana ruang penyimpanan tambahan terbatas.
- **Kompleksitas Implementasi:** Implementasi Merge Sort memerlukan rekursi dan logika penggabungan yang kadang-kadang sulit diimplementasikan untuk pemula.

Merge Sort sangat cocok untuk pengurutan data dalam skenario di mana stabilitas dan efisiensi pada data besar diperlukan. Meskipun memerlukan ruang tambahan, keuntungan dalam hal stabilitas dan performa membuatnya menjadi pilihan yang baik untuk aplikasi yang membutuhkan pengurutan yang handal dan efisien.

e) Merge Sort

Quick Sort adalah algoritma sorting yang menggunakan pendekatan divide and conquer. Algoritma ini memilih elemen pivot dari array dan membagi array menjadi dua bagian: elemen yang lebih kecil dari pivot dan elemen yang lebih besar dari pivot. Setelah itu, Quick Sort diterapkan secara rekursif pada kedua bagian tersebut. Proses ini berlanjut hingga seluruh array terurut.

Kelebihan Quick Sort:

- **Efisien pada Data Besar:** Quick Sort memiliki kompleksitas waktu O(n log n) pada kasus rata-rata, menjadikannya efisien untuk data yang besar.
- **In-Place Sorting:** Quick Sort dapat diimplementasikan sebagai in-place sorting, yaitu melakukan pengurutan tanpa memerlukan alokasi memori tambahan, kecuali untuk variabel pivot.
- **Kemampuan Mengatasi Data yang Hampir Terurut:** Quick Sort dapat memiliki kinerja yang baik pada data yang sudah hampir terurut, terutama jika implementasinya memilih pivot dengan cerdas.
- **Kinerja Rata-rata Tinggi:** Quick Sort umumnya memiliki kinerja rata-rata yang tinggi dan menjadi salah satu algoritma sorting yang sering digunakan.

Kekurangan Quick Sort:

- **Tidak Stabil:** Quick Sort tidak stabil, yang berarti urutan relatif elemen-elemen dengan nilai yang sama dapat berubah setelah pengurutan.
- **Sensitif terhadap Pemilihan Pivot:** Kinerja Quick Sort dapat terpengaruh oleh pemilihan pivot. Pemilihan pivot yang buruk dapat menghasilkan kinerja yang buruk.
- **Tidak Cocok untuk Struktur Data Terhubung:** Quick Sort memerlukan akses acak ke elemen array, sehingga tidak selalu cocok untuk struktur data terhubung seperti linked list.
- **Kompleksitas Implementasi:** Implementasi Quick Sort bisa lebih kompleks daripada algoritma sorting sederhana seperti Bubble Sort atau Insertion Sort.

Quick Sort sangat efisien pada data yang besar dan sering digunakan dalam praktiknya. Pemilihan pivot yang cerdas dan implementasi yang baik dapat memaksimalkan kelebihan algoritma ini.

D. Analisis Kinerja Sorting Algoritma

Pada tahap analisis kinerja algoritma pengurutan, beberapa faktor penting perlu dipertimbangkan untuk memilih algoritma yang paling sesuai dengan kebutuhan spesifik. Berikut adalah aspek-aspek kunci yang dapat dievaluasi:

a) Kompleksitas Waktu:

Kasus Terbaik, Rata-rata, dan Terburuk: Evaluasi waktu eksekusi algoritma pada kasus terbaik, rata-rata, dan terburuk. Perbandingan ini membantu memahami bagaimana algoritma berkinerja pada berbagai skenario input.

b) Kompleksitas Ruang:

Alokasi Memori Tambahan: Perhatikan seberapa banyak ruang memori tambahan yang diperlukan oleh setiap algoritma. Algoritma in-place (tanpa alokasi memori tambahan) dapat lebih efisien dalam penggunaan sumber daya.

c) Stabilitas:

Stabilitas Pengurutan: Stabilitas mengacu pada kemampuan algoritma untuk mempertahankan urutan relatif elemen dengan nilai yang sama. Algoritma stabil akan menjaga urutan ini, sementara algoritma yang tidak stabil bisa mengubahnya.

d) Responsif Terhadap Jenis Data:

Data Hampir Terurut atau Sudah Terurut: Analisis sejauh mana algoritma mempertahankan efisiensi pada data yang hampir terurut atau sudah terurut. Beberapa algoritma mungkin lebih cocok untuk skenario ini.

e) Kemampuan Mengatasi Data Besar:

Kompleksitas Waktu pada Data Besar: Perhatikan kompleksitas waktu algoritma pada skala data yang besar. Beberapa algoritma mungkin lebih efisien ketika menangani data dengan jumlah elemen yang besar.

f) Kemampuan Mengatasi Data Terbalik Urut:

Kinerja pada Data Terbalik Urut: Beberapa algoritma memiliki kinerja yang lebih buruk ketika menghadapi data yang sudah terbalik urut. Pahami sejauh mana algoritma dapat beradaptasi dengan kondisi ini.

g) Adaptabilitas terhadap Kondisi:

Dinamika Perubahan Data: Evaluasi sejauh mana algoritma dapat menyesuaikan diri dengan perubahan kondisi data. Beberapa algoritma mungkin lebih fleksibel dalam menanggapi variasi input.

h) Keterbacaan dan Kepahaman:

Kesederhanaan Implementasi: Pertimbangkan seberapa sederhana algoritma dapat diimplementasikan dan dipahami. Keterbacaan kode yang baik dapat mempermudah pengembangan dan pemeliharaan aplikasi.

Analisis menyeluruh terhadap aspek-aspek ini membantu menghasilkan pemahaman yang lebih baik tentang karakteristik masing-masing algoritma pengurutan, memungkinkan pemilihan yang lebih bijak sesuai dengan konteks penggunaan yang diinginkan.

E. Notasi Big-O

Notasi Big-O adalah cara untuk mengekspresikan kinerja atau kompleksitas waktu suatu algoritma. Notasi ini memberikan batasan atas (upper bound) pertumbuhan algoritma seiring dengan peningkatan ukuran input. Dalam konteks sorting algorithms atau algoritma pengurutan, Notasi Big-O sangat penting untuk memahami bagaimana algoritma tersebut akan berkinerja terhadap jumlah data yang meningkat.

Berikut adalah beberapa contoh Notasi Big-O untuk berbagai jenis algoritma pengurutan:

a) **Bubble Sort:**
   - Notasi Big-O: O(n^2)
   - Penjelasan: Kinerja Bubble Sort pada kasus terburuk adalah kuadratik, di mana waktu eksekusi tumbuh sebanding dengan kuadrat jumlah elemen dalam array.

b) **Insertion Sort:**
   - Notasi Big-O: O(n^2)
   - Penjelasan: Seperti Bubble Sort, Insertion Sort memiliki kompleksitas waktu kuadratik dalam kasus terburuk.

c) **Selection Sort:**
   - Notasi Big-O: O(n^2)
   - Penjelasan: Kompleksitas waktu Selection Sort juga kuadratik, karena pada setiap iterasi, elemen minimum harus dicari dan ditukar.

d) **Merge Sort:**
   - Notasi Big-O: O(n log n)
   - Penjelasan: Merge Sort memiliki kompleksitas waktu yang lebih baik, yaitu logaritmik, karena menggunakan pendekatan divide and conquer.

e) **Quick Sort:**
   - Notasi Big-O: O(n^2) dalam kasus terburuk, O(n log n) pada kasus rata-rata.
   - Penjelasan: Meskipun memiliki kompleksitas waktu yang lebih rendah pada kasus rata-rata, Quick Sort dapat memiliki kompleksitas waktu kuadratik dalam kasus terburuk jika pivot dipilih dengan buruk.

Notasi Big-O memberikan gambaran tentang seberapa efisien suatu algoritma pengurutan pada skala data yang besar. Dalam konteks ini, algoritma

dengan kompleksitas waktu O(n log n) cenderung lebih efisien pada data yang besar dibandingkan dengan yang memiliki kompleksitas O(n^2).

F. Implementasi Bubble Sort

Contoh Program

```
#include <stdio.h>

// Fungsi untuk melakukan Bubble Sort
void bubbleSort(int arr[], int n) {
    int i, j;

    // Iterasi melalui semua elemen array
    for (i = 0; i < n-1; i++) {

        // Setiap iterasi akan "menggelembungkan" elemen terbesar ke posisi yang benar
        for (j = 0; j < n-i-1; j++) {

            // Jika elemen ke-j lebih besar dari elemen ke-(j+1), tukar keduanya
            if (arr[j] > arr[j+1]) {
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
```

```
            }
        }
    }
}

// Fungsi untuk menampilkan elemen-elemen array
void printArray(int arr[], int size) {
    int i;

    // Iterasi melalui semua elemen array dan menampilkan mereka
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Program utama untuk menguji Bubble Sort
int main() {
    // Inisialisasi array yang akan diurutkan
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Menampilkan array sebelum diurutkan
```

```
    printf("Array sebelum diurutkan: \n");
    printArray(arr, n);

    // Panggil fungsi Bubble Sort
    bubbleSort(arr, n);

    // Menampilkan array setelah diurutkan
    printf("Array setelah diurutkan: \n");
    printArray(arr, n);

    return 0;
}
```

Penjelasannya :

- Fungsi bubbleSort melakukan Bubble Sort pada array yang diberikan. Iterasi pertama mengurutkan elemen terbesar ke posisi terakhir, iterasi kedua mengurutkan elemen kedua terbesar ke posisi kedua dari belakang, dan seterusnya.
- Fungsi printArray digunakan untuk menampilkan elemen-elemen array.
- Dalam fungsi main, array awal ditampilkan, kemudian fungsi Bubble Sort dipanggil, dan array setelah diurutkan ditampilkan kembali.

G. Implementasi Inesertion Sort

Contoh Program

```
#include <stdio.h>


// Fungsi untuk melakukan Insertion Sort
void insertionSort(int arr[], int n) {
    int i, key, j;

    // Iterasi mulai dari elemen kedua hingga elemen terakhir
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Pindahkan elemen-elemen yang lebih besar dari key ke posisi satu lebih ke kanan
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }

        // Tempatkan key pada posisi yang benar dalam array yang diurutkan
        arr[j + 1] = key;
    }
```

```
}

// Fungsi untuk menampilkan elemen-elemen array
void printArray(int arr[], int size) {
    int i;

    // Iterasi melalui semua elemen array dan menampilkan mereka
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Program utama untuk menguji Insertion Sort
int main() {
    // Inisialisasi array yang akan diurutkan
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Menampilkan array sebelum diurutkan
    printf("Array sebelum diurutkan: \n");
    printArray(arr, n);
```

```
    // Panggil fungsi Insertion Sort
    insertionSort(arr, n);

    // Menampilkan array setelah diurutkan
    printf("Array setelah diurutkan: \n");
    printArray(arr, n);

    return 0;
}
```

Penjelasannya :

- Fungsi insertionSort melakukan Insertion Sort pada array yang diberikan. Pada setiap iterasi, elemen yang belum diurutkan diambil satu per satu dan ditempatkan pada posisi yang benar dalam array yang sudah diurutkan.
- Fungsi printArray digunakan untuk menampilkan elemen-elemen array.
- Dalam fungsi main, array awal ditampilkan, kemudian fungsi Insertion Sort dipanggil, dan array setelah diurutkan ditampilkan kembali.

H. Implementasi Selection Sort

Contoh Program

```
#include <stdio.h>

// Fungsi untuk melakukan Selection Sort
void selectionSort(int arr[], int n) {
    int i, j, min_idx;
```

```
    // Iterasi melalui semua elemen array
    for (i = 0; i < n-1; i++) {
        // Temukan indeks elemen terkecil di bagian belum diurutkan
        min_idx = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx])
                min_idx = j;
        }

        // Tukar elemen terkecil dengan elemen pertama di bagian belum diurutkan
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

// Fungsi untuk menampilkan elemen-elemen array
void printArray(int arr[], int size) {
    int i;

    // Iterasi melalui semua elemen array dan menampilkan mereka
```

```
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Program utama untuk menguji Selection Sort
int main() {
    // Inisialisasi array yang akan diurutkan
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Menampilkan array sebelum diurutkan
    printf("Array sebelum diurutkan: \n");
    printArray(arr, n);

    // Panggil fungsi Selection Sort
    selectionSort(arr, n);

    // Menampilkan array setelah diurutkan
    printf("Array setelah diurutkan: \n");
    printArray(arr, n);
```

```
    return 0;
}
```

Penjelasannya

- Fungsi selectionSort melakukan Selection Sort pada array yang diberikan. Pada setiap iterasi, mencari elemen terkecil di bagian array yang belum diurutkan dan menukarnya dengan elemen pertama di bagian belum diurutkan.
- Fungsi printArray digunakan untuk menampilkan elemen-elemen array.
- Dalam fungsi main, array awal ditampilkan, kemudian fungsi Selection Sort dipanggil, dan array setelah diurutkan ditampilkan kembali.

I. Implementasi Merge Sort

Contoh Program

```
#include <stdio.h>
#include <stdlib.h>

// Fungsi untuk menggabungkan dua bagian terurut dari array
void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Buat array sementara untuk dua bagian
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));
```

```
// Salin data ke array sementara L[] dan R[]
for (i = 0; i < n1; i++)
    L[i] = arr[left + i];
for (j = 0; j < n2; j++)
    R[j] = arr[mid + 1 + j];


// Gabungkan array sementara ke dalam array utama
i = 0;
j = 0;
k = left;
while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}
```

```
    // Salin sisa elemen dari L[], jika ada
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Salin sisa elemen dari R[], jika ada
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    // Bebaskan memori dari array sementara
    free(L);
    free(R);
}

// Fungsi utama untuk melakukan Merge Sort
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
```

```
        // Temukan titik tengah array
        int mid = left + (right - left) / 2;

        // Panggil rekursif untuk dua bagian yang terpisah
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Gabungkan dua bagian yang terpisah
        merge(arr, left, mid, right);
    }
}

// Fungsi untuk menampilkan elemen-elemen array
void printArray(int arr[], int size) {
    int i;

    // Iterasi melalui semua elemen array dan menampilkan mereka
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

```
// Program utama untuk menguji Merge Sort
int main() {
    // Inisialisasi array yang akan diurutkan
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Menampilkan array sebelum diurutkan
    printf("Array sebelum diurutkan: \n");
    printArray(arr, n);

    // Panggil fungsi Merge Sort
    mergeSort(arr, 0, n - 1);

    // Menampilkan array setelah diurutkan
    printf("Array setelah diurutkan: \n");
    printArray(arr, n);

    return 0;
}
```

Penjelasannya

- Fungsi merge digunakan untuk menggabungkan dua bagian terurut dari array menjadi satu bagian terurut.

- Fungsi mergeSort adalah implementasi rekursif dari algoritma Merge Sort. Ini membagi array menjadi dua bagian, memanggil dirinya sendiri untuk setiap bagian, dan kemudian menggabungkan dua bagian tersebut.
- Fungsi printArray digunakan untuk menampilkan elemen-elemen array.
- Dalam fungsi main, array awal ditampilkan, kemudian fungsi Merge Sort dipanggil, dan array setelah diurutkan ditampilkan kembali.

J. Implementasi Quick Sort

Contoh Program

```
#include <stdio.h>


// Fungsi untuk menukar dua elemen dalam array
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}


// Fungsi untuk menentukan posisi akhir dari pivot dan mengatur array di sekitarnya
int partition(int arr[], int low, int high) {
    // Pilih pivot (di sini, kami memilih elemen terakhir sebagai pivot)
    int pivot = arr[high];


    // Indeks elemen yang kurang dari atau sama dengan pivot
```

```
    int i = (low - 1);

    // Iterasi melalui semua elemen, membandingkan masing-masing dengan pivot
    for (int j = low; j <= high - 1; j++) {
        // Jika elemen saat ini lebih kecil dari atau sama dengan pivot
        if (arr[j] <= pivot) {
            // Tingkatkan indeks elemen yang lebih kecil atau sama dengan pivot
            i++;
            // Tukar arr[i] dan arr[j]
            swap(&arr[i], &arr[j]);
        }
    }

    // Tukar arr[i+1] dan arr[high] (pivot)
    swap(&arr[i + 1], &arr[high]);
    // Kembalikan indeks di mana pivot sekarang berada
    return (i + 1);
}

// Fungsi utama untuk melakukan Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
```

```
        // Temukan posisi pivot seperti yang dijelaskan di atas
        int pi = partition(arr, low, high);

        // Panggil rekursif untuk bagian sebelum dan setelah pivot
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Fungsi untuk menampilkan elemen-elemen array
void printArray(int arr[], int size) {
    int i;

    // Iterasi melalui semua elemen array dan menampilkan mereka
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Program utama untuk menguji Quick Sort
int main() {
    // Inisialisasi array yang akan diurutkan
```

```
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Menampilkan array sebelum diurutkan
    printf("Array sebelum diurutkan: \n");
    printArray(arr, n);

    // Panggil fungsi Quick Sort
    quickSort(arr, 0, n - 1);

    // Menampilkan array setelah diurutkan
    printf("Array setelah diurutkan: \n");
    printArray(arr, n);

    return 0;
}
```

Penjelasannya

- Fungsi swap digunakan untuk menukar dua elemen dalam array.
- Fungsi partition menentukan posisi akhir dari pivot dan mengatur array di sekitarnya.
- Fungsi quickSort adalah implementasi rekursif dari algoritma Quick Sort. Ini memilih pivot, mempartisi array, dan kemudian memanggil dirinya sendiri untuk bagian sebelum dan setelah pivot.
- Fungsi printArray digunakan untuk menampilkan elemen-elemen array.

- Dalam fungsi main, array awal ditampilkan, kemudian fungsi Quick Sort dipanggil, dan array setelah diurutkan ditampilkan kembali.

K. Penerapan Sorting Dalam Proyek Nyata

Penerapan algoritma pengurutan (sorting) dalam proyek nyata dapat memberikan manfaat yang signifikan dalam pengolahan dan analisis data. Berikut adalah beberapa contoh penerapan sorting dalam konteks proyek nyata:

a) Pengurutan Data Pelanggan:

Dalam proyek manajemen pelanggan, data pelanggan dapat diurutkan berdasarkan kriteria tertentu, seperti nama belakang atau ID pelanggan. Hal ini memudahkan pencarian data pelanggan dan penanganan transaksi.

b) Pengelolaan Data Transaksi Keuangan:

Pada proyek keuangan, data transaksi dapat diurutkan berdasarkan tanggal atau jumlah transaksi. Sorting mempermudah analisis tren keuangan, identifikasi anomali, dan penyusunan laporan keuangan.

c) Pengurutan Produk dalam Inventori:

Dalam sistem manajemen inventori, produk dapat diurutkan berdasarkan kategori, nama, atau jumlah stok. Sorting membantu pengelolaan inventori, penentuan kebutuhan restok, dan pemantauan ketersediaan produk.

d) Analisis Data Sensor atau IoT:

Dalam proyek berbasis sensor atau Internet of Things (IoT), data sensor dapat diurutkan berdasarkan waktu pengambilan data. Sorting memungkinkan analisis temporal, deteksi pola, dan pemantauan perubahan kondisi secara kronologis.

e) Penyusunan Jadwal Acara:

Dalam proyek manajemen acara, jadwal acara dapat diurutkan berdasarkan waktu atau jenis acara. Sorting membantu penyusunan jadwal yang efisien dan mempermudah peserta acara untuk merencanakan kehadiran.

f) Pengolahan Data Pribadi dalam Aplikasi Sosial:

Dalam aplikasi media sosial, daftar teman atau kontak dapat diurutkan berdasarkan nama atau interaksi terakhir. Sorting memfasilitasi pencarian teman dan pengelolaan hubungan sosial.

g) Analisis Data Geospasial:

Dalam proyek yang melibatkan data geospasial, seperti peta atau pemantauan lingkungan, data dapat diurutkan berdasarkan koordinat geografis. Sorting memungkinkan penyajian data yang lebih terstruktur dan efisien.

h) Rangking Produk atau Layanan:

Dalam platform e-commerce atau ulasan produk, produk atau layanan dapat diurutkan berdasarkan peringkat atau ulasan pengguna. Sorting membantu pembeli dalam mengambil keputusan berdasarkan preferensi atau popularitas.

Penerapan sorting ini mencerminkan penggunaan yang umum dalam berbagai bidang, membantu meningkatkan keteraturan, efisiensi pencarian, dan kemampuan analisis data.

BAB II

PENUTUP

A. Kesimpulan

Penerapan algoritma pengurutan (sorting) dalam berbagai proyek nyata membuktikan bahwa sorting bukan hanya suatu langkah proses rutin, melainkan fondasi penting dalam mengoptimalkan pengolahan data. Dalam proyek manajemen pelanggan, sorting mempermudah pencarian data pelanggan dan meningkatkan efisiensi transaksi. Dalam konteks keuangan, sorting berperan krusial dalam analisis tren keuangan dan penyusunan laporan keuangan yang akurat. Selain itu, pada tingkat logistik dan inventori, sorting menjadi landasan untuk pengelolaan inventori yang efektif dan pemantauan ketersediaan produk.

Proyek-proyek berbasis sensor dan IoT juga menemukan nilai tambah dari sorting, memungkinkan analisis temporal dan pemantauan perubahan kondisi secara kronologis. Dalam pengaturan jadwal acara, sorting mendukung penyusunan jadwal yang efisien dan memudahkan peserta acara untuk merencanakan kehadiran. Begitu juga, dalam dunia media sosial, sorting berfungsi sebagai elemen kunci dalam manajemen teman dan kontak, memastikan pengguna dapat dengan cepat menemukan dan berinteraksi dengan orang-orang terdekat.

Pentingnya sorting tidak hanya terbatas pada pemrosesan data. Dalam konteks data geospasial, sorting memainkan peran penting dalam penyajian data yang terstruktur dan efisien, memudahkan pemahaman dan pengambilan keputusan. Selanjutnya, pada platform e-commerce dan ulasan produk, sorting menjadi landasan untuk menyusun peringkat dan ulasan pengguna, memberikan panduan berharga bagi pembeli dalam mengambil keputusan berdasarkan preferensi atau popularitas.

Dengan demikian, kesimpulan utama yang dapat diambil adalah bahwa sorting bukan hanya alat teknis, tetapi fondasi yang mendasari efisiensi operasional dan analisis mendalam dalam proyek-proyek nyata. Keberhasilan implementasi algoritma sorting tidak hanya memastikan ketertiban data, tetapi juga membuka

pintu untuk pemahaman yang lebih baik, pengelolaan yang lebih efektif, dan pengambilan keputusan yang lebih tepat dalam berbagai konteks proyek. Sehingga, pemahaman mendalam tentang algoritma sorting dan penerapannya secara kontekstual dapat memperkuat landasan pengembangan proyek-proyek berkelanjutan dan efisien.

DAFTAR PUSTAKA

https://www.sobatambisius.com/2021/09/belajar-bahasa-c-12-sorting.html

https://www.educba.com/sorting-in-c/

https://www.geeksforgeeks.org/sorting-algorithms/

https://www.sanfoundry.com/c-program-sort-array-ascending-order/

https://id.scribd.com/doc/310956602/Sorting-Array-Pada-Bahasa-C