UJIAN AKHIR SEMESTER

ALGORITMA PEMROGRAMMAN 2

Disusun Oleh :

Nama : Rio Permana Mardianto

NIM : 231011403458

Kelas : 03TPLP027

TEKNIK INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS PAMULANG

JL. RAYA PUSPITEK, BUARAN, KEC. PAMULANG, KOTA TANGGERANG SELATAN BANTEN.

SOAL

1. Anda diberikan sebuah array yang berisi data acak. Anda diminta untuk mengimplementasikan algoritma sorting yang paling efisien untuk data tersebut.
   Pertanyaan:
   a) Bagaimana cara Anda menentukan algoritma sorting yang paling tepat? (10 Poin)
   b) Jelaskan faktor-faktor yang perlu dipertimbangkan (misalnya, ukuran data, tingkat keterurutan awal, stabilitas algoritma). (10 Poin)
   c) Bandingkan dan kontraskan antara algoritma binary search dan linear search. Dalam situasi apa binary search lebih disukai dibandingkan linear search? Jelaskan dengan contoh program dalam C++ atau Python. (30 Poin)

2. Anda diberikan dua buah array yang sudah terurut.
   Pertanyaan dan Tugas:
   a) Bagaimana cara Anda menggabungkan kedua array tersebut menjadi satu array yang baru dan tetap terurut dalam waktu yang efisien? (10 Poin)
   b) Algoritma penggabungan mana yang akan Anda pilih dan mengapa? (10 Poin)
   c) Buatkan Program dalam C++ atau Python untuk algoritma sorting yang telah anda pilih! (30 Poin)

JAWABAN

NO 1

a) Untuk menentukan algoritma sorting yang paling tepat, langkah-langkah berikut dapat dilakukan:
   1. **Analisis Ukuran Data**
      Jika ukuran data kecil (misalnya, kurang dari 100 elemen), algoritma sederhana seperti **Insertion Sort** dapat dipertimbangkan. Jika ukuran data besar (misalnya, ratusan ribu atau lebih), algoritma efisien seperti **Quick Sort**, **Merge Sort**, atau **Heap Sort** lebih cocok.
   2. **Analisis Tingkat Keterurutan Awal**
      Jika data sudah hampir terurut, **Insertion Sort** atau **Bubble Sort** dengan optimasi dapat lebih efisien. Jika data sepenuhnya acak, algoritma berbasis **Divide and Conquer** seperti **Quick Sort** atau **Merge Sort** biasanya lebih efisien.
   3. **Kebutuhan Stabilitas**
      Jika stabilitas diperlukan (misalnya, elemen dengan kunci yang sama harus mempertahankan urutannya), algoritma seperti **Merge Sort** atau **Tim Sort** lebih cocok.
   4. **Memori yang Tersedia**
      Jika memori terbatas, gunakan algoritma **In-Place** seperti **Quick Sort** atau **Heap Sort**

      .

b) Faktor-faktor yang Perlu Dipertimbangkan yaitu :

1. **Ukuran Data**
   - Algoritma sederhana (seperti **Bubble Sort**) dapat bekerja untuk data kecil, tetapi tidak efisien untuk data besar.
   - Algoritma kompleks (seperti **Merge Sort**) memiliki kinerja lebih baik pada data besar.
2. **Tingkat Keterurutan Awal**
   Data yang hampir terurut dapat diproses lebih efisien dengan algoritma berbasis perbandingan sederhana, seperti **Insertion Sort**.
3. **Stabilitas Algoritma**
   Stabilitas penting jika elemen yang memiliki nilai kunci sama harus mempertahankan urutan aslinya.
4. **Kompleksitas Waktu dan Ruang**
   Beberapa algoritma (seperti **Merge Sort**) membutuhkan ruang tambahan untuk array temporer, sementara algoritma seperti **Quick Sort** menggunakan ruang secara in-place.
5. **Karakteristik Data**
   Jika data berupa angka dengan rentang kecil, **Counting Sort** atau **Radix Sort** lebih cocok dibanding algoritma berbasis perbandingan.

c) Contoh perbandingan antara Algoritma Binary Search dan Linear Search dalam program C++ :

uas.cpp > main()

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Binary Search Implementation
int binarySearch(vector<int>& arr, int target) {
    int left = 0, right = arr.size() - 1;
    int langkah = 1; // Variabel untuk mencatat langkah

    while (left <= right) {
        cout << "Langkah " << langkah++ << ": Mencari di rentang indeks [" << left << ", " << right << "]\n";
        int mid = left + (right - left) / 2; // Menghindari overflow
        cout << "   Indeks tengah: " << mid << ", Nilai tengah: " << arr[mid] << endl;

        if (arr[mid] == target) {
            cout << "   Target ditemukan di indeks " << mid << endl;
            return mid;
        }
        else if (arr[mid] < target) {
            cout << "   Target lebih besar dari nilai tengah, pindah ke kanan\n";
            left = mid + 1;
        } else {
            cout << "   Target lebih kecil dari nilai tengah, pindah ke kiri\n";
            right = mid - 1;
        }
    }

    cout << "   Target tidak ditemukan dalam array.\n";
    return -1; // Target tidak ditemukan
}
```

```cpp
// Linear Search Implementation
int linearSearch(vector<int>& arr, int target) {
    cout << "Memulai Linear Search...\n";
    for (int i = 0; i < arr.size(); ++i) {
        cout << "Langkah " << i + 1 << ": Memeriksa indeks " << i << ", Nilai: " << arr[i] << endl;
        if (arr[i] == target) {
            cout << "   Target ditemukan di indeks " << i << endl;
            return i;
        }
    }

    cout << "   Target tidak ditemukan dalam array.\n";
    return -1; // Target tidak ditemukan
}
```

```cpp
int main() {
    cout<<"Nama : Rio Permana Mardianto"<<endl;
    cout<<"NIM : 231011403458"<<endl;
    cout<<"Kelas : 03TPLP027"<<endl;
    cout<<"UJIAN AKHIR SEMESTER\n"<<endl;

    vector<int> data = {3, 6, 8, 10, 15, 20};
    int target = 10;

    // Sort data untuk Binary Search
    sort(data.begin(), data.end());
    cout << "Array setelah diurutkan: ";
    for (int val : data) cout << val << " ";
    cout << "\n\n";

    // Binary Search
    cout << "Melakukan Binary Search untuk target " << target << "...\n";
    int binaryResult = binarySearch(data, target);
    if (binaryResult != -1)
        cout << "Hasil Binary Search: Ditemukan di indeks " << binaryResult << "\n\n";
    else
        cout << "Hasil Binary Search: Tidak ditemukan\n\n";

    // Linear Search
    cout << "Melakukan Linear Search untuk target " << target << "...\n";
    int linearResult = linearSearch(data, target);
    if (linearResult != -1)
        cout << "Hasil Linear Search: Ditemukan di indeks " << linearResult << "\n";
    else
        cout << "Hasil Linear Search: Tidak ditemukan\n";

    return 0;
}
```

OUTPUT :

```
PS D:\Perkodingan\c++\semester3> ./uas1
Nama : Rio Permana Mardianto
NIM : 231011403458
Kelas : 03TPLP027
UJIAN AKHIR SEMESTER

Array setelah diurutkan: 3 6 8 10 15 20

Melakukan Binary Search untuk target 10...
Langkah 1: Mencari di rentang indeks [0, 5]
   Indeks tengah: 2, Nilai tengah: 8
   Target lebih besar dari nilai tengah, pindah ke kanan
Langkah 2: Mencari di rentang indeks [3, 5]
   Indeks tengah: 4, Nilai tengah: 15
   Target lebih kecil dari nilai tengah, pindah ke kiri
Langkah 3: Mencari di rentang indeks [3, 3]
   Indeks tengah: 3, Nilai tengah: 10
   Target ditemukan di indeks 3
Hasil Binary Search: Ditemukan di indeks 3

Melakukan Linear Search untuk target 10...
Memulai Linear Search...
Langkah 1: Memeriksa indeks 0, Nilai: 3
Langkah 2: Memeriksa indeks 1, Nilai: 6
Langkah 3: Memeriksa indeks 2, Nilai: 8
Langkah 4: Memeriksa indeks 3, Nilai: 10
   Target ditemukan di indeks 3
Hasil Linear Search: Ditemukan di indeks 3
PS D:\Perkodingan\c++\semester3>
```

NO 2

a) **Cara Menggabungkan Kedua Array yang Terurut**
Cara efisien untuk menggabungkan dua array terurut menjadi satu array baru yang tetap terurut adalah dengan menggunakan **algoritma penggabungan (merge)**. Algoritma ini memanfaatkan fakta bahwa kedua array sudah dalam keadaan terurut, sehingga kita dapat membandingkan elemen-elemen dari kedua array secara bersamaan.

b) **Algoritma yang Dipilih: Merge Algorithm**
Saya akan memilih **algoritma merge** karena:

- **Efisiensi Waktu,** Algoritma ini bekerja dalam waktu O(n+m)O(n + m)O(n+m), di mana nnn adalah panjang array pertama, dan mmm adalah panjang array kedua. Ini

jauh lebih efisien dibandingkan menggabungkan array terlebih dahulu dan kemudian menggunakan algoritma sorting yang umum seperti quicksort atau mergesort (O((n+m) log(n+m))).

- **Sederhana,** Implementasi algoritma ini langsung memanfaatkan keterurutan kedua array tanpa langkah tambahan.
- **Stabilitas,** Algoritma ini menjaga urutan elemen yang setara jika diperlukan.

c) Contoh program dengan Bahasa pemrograman C++ :

uas2.cpp > mergeArrays(const vector<int>&, const vector<int>&)

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Fungsi untuk menggabungkan dua array yang terurut
vector<int> mergeArrays(const vector<int>& arr1, const vector<int>& arr2) {
    vector<int> merged; // Array hasil gabungan
    int i = 0, j = 0;

    // Menggabungkan elemen dari kedua array
    while (i < arr1.size() && j < arr2.size()) {
        if (arr1[i] <= arr2[j]) {
            merged.push_back(arr1[i]);
            cout << "Menambahkan " << arr1[i] << " dari array pertama\n";
            i++;
        } else {
            merged.push_back(arr2[j]);
            cout << "Menambahkan " << arr2[j] << " dari array kedua\n";
            j++;
        }
    }

    // Menambahkan sisa elemen dari array pertama (jika ada)
    while (i < arr1.size()) {
        merged.push_back(arr1[i]);
        cout << "Menambahkan sisa " << arr1[i] << " dari array pertama\n";
        i++;
    }

    // Menambahkan sisa elemen dari array kedua (jika ada)
    while (j < arr2.size()) {
        merged.push_back(arr2[j]);
        cout << "Menambahkan sisa " << arr2[j] << " dari array kedua\n";
        j++;
    }

    return merged;
}
```

```cpp
int main() {
    cout<<"Nama : Rio Permana Mardianto"<<endl;
    cout<<"NIM : 231011403458"<<endl;
    cout<<"Kelas : 03TPLP027"<<endl;
    cout<<"UJIAN AKHIR SEMESTER\n"<<endl;
    // Contoh array terurut
    vector<int> array1 = {1, 3, 5, 7};
    vector<int> array2 = {2, 4, 6, 8};

    cout << "Menggabungkan kedua array terurut...\n";
    vector<int> result = mergeArrays(array1, array2);

    // Menampilkan hasil penggabungan
    cout << "Array hasil penggabungan: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT :

```
PS D:\Perkodingan\c++\semester3> ./uas2
Nama : Rio Permana Mardianto
NIM : 231011403458
Kelas : 03TPLP027
UJIAN AKHIR SEMESTER

Menggabungkan kedua array terurut...
Menambahkan 1 dari array pertama
Menambahkan 2 dari array kedua
Menambahkan 3 dari array pertama
Menambahkan 4 dari array kedua
Menambahkan 5 dari array pertama
Menambahkan 6 dari array kedua
Menambahkan 7 dari array pertama
Menambahkan sisa 8 dari array kedua
Array hasil penggabungan: 1 2 3 4 5 6 7 8
PS D:\Perkodingan\c++\semester3>
```