Ujian Akhir Semester

Algoritma dan Pemrogramman II

Disusun Oleh:

Nama: Abdil Rambhani

NIM: 231011401210

Kelas: 03TPLP029

FAKULTAS ILMU KOMPUTER

PROGRAM STUDI TEKNIK INFORMATIKA

JAWABAN

1. Berikut adalah langkah-langkahnya:

   Bangun Pohon Huffman:
   Hitung frekuensi setiap karakter dalam string.
   Gunakan struktur seperti min-heap untuk menyimpan node berdasarkan frekuensinya.
   Gabungkan dua node dengan frekuensi terendah menjadi satu hingga hanya tersisa satu node (root pohon).

   Proses Encoding:
   Traversal pohon untuk menghasilkan kode Huffman (0 untuk kiri, 1 untuk kanan).

   Proses Decoding:
   Gunakan string biner dan traversal pohon untuk mendekode.

Tanggal: 08/01/2025	Pohon Huffman (Soal No 1)

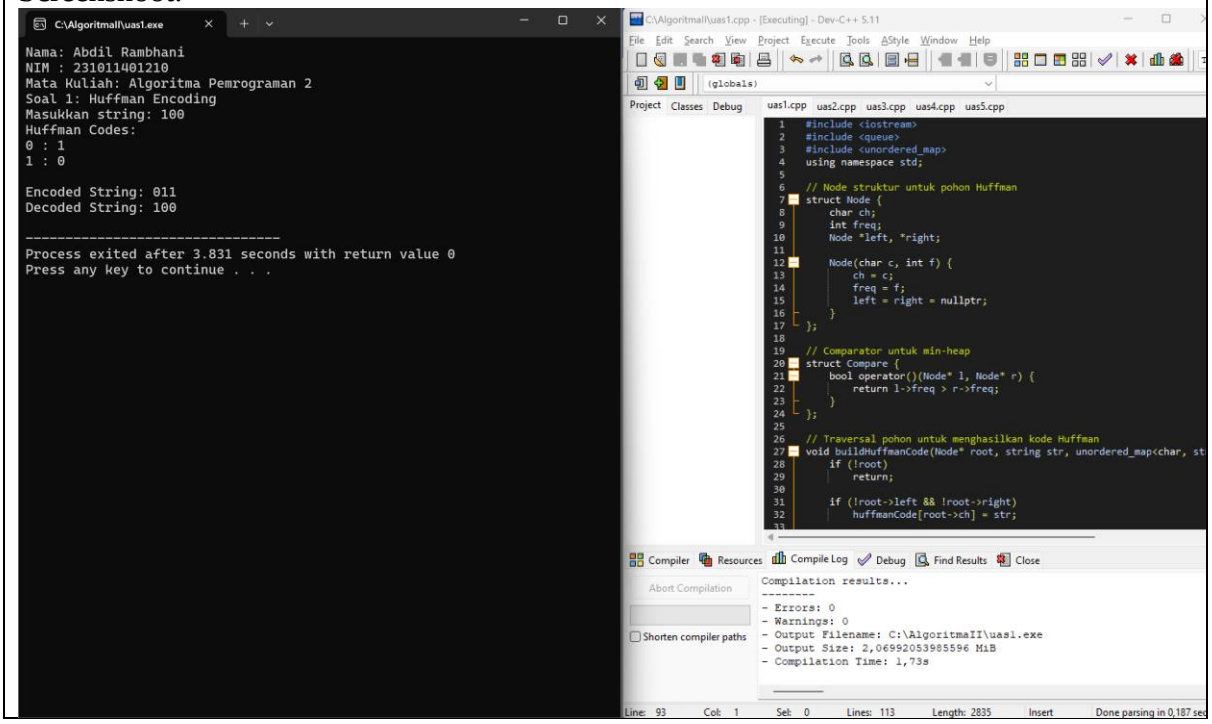
Source Code:

```
#include <iostream>
#include <queue>
#include <unordered_map>
using namespace std;

// Node struktur untuk pohon Huffman
struct Node {
    char ch;
    int freq;
    Node *left, *right;

    Node(char c, int f) {
        ch = c;
        freq = f;
        left = right = nullptr;
    }
};

// Comparator untuk min-heap
struct Compare {
    bool operator()(Node* l, Node* r) {
        return l->freq > r->freq;
    }
};

// Traversal pohon untuk menghasilkan kode Huffman
```

```
void buildHuffmanCode(Node* root, string str, unordered_map<char, string>&
huffmanCode) {
    if (!root)
        return;

    if (!root->left && !root->right)
        huffmanCode[root->ch] = str;

    buildHuffmanCode(root->left, str + "0", huffmanCode);
    buildHuffmanCode(root->right, str + "1", huffmanCode);
}

// Dekode string biner menggunakan pohon Huffman
string decodeHuffman(Node* root, string encodedStr) {
    string decodedStr = "";
    Node* current = root;

    for (char bit : encodedStr) {
        current = (bit == '0') ? current->left : current->right;

        if (!current->left && !current->right) {
            decodedStr += current->ch;
            current = root;
        }
    }

    return decodedStr;
}

void huffmanEncoding(string text) {
    // Hitung frekuensi setiap karakter
    unordered_map<char, int> freq;
    for (char ch : text)
        freq[ch]++;

    // Bangun min-heap
    priority_queue<Node*, vector<Node*>, Compare> pq;
    for (auto pair : freq)
        pq.push(new Node(pair.first, pair.second));

    // Bangun pohon Huffman
    while (pq.size() > 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push(new Node('\0', sum));
        pq.top()->left = left;
        pq.top()->right = right;
    }
```

```
    Node* root = pq.top();

    // Bangun kode Huffman
    unordered_map<char, string> huffmanCode;
    buildHuffmanCode(root, "", huffmanCode);

    cout << "Huffman Codes:\n";
    for (auto pair : huffmanCode)
        cout << pair.first << " : " << pair.second << endl;

    // Encoding teks
    string encodedStr = "";
    for (char ch : text)
        encodedStr += huffmanCode[ch];

    cout << "\nEncoded String: " << encodedStr << endl;

    // Decoding teks
    string decodedStr = decodeHuffman(root, encodedStr);
    cout << "Decoded String: " << decodedStr << endl;
}

int main() {
    cout << "Nama: Abdil Rambhani" << endl;
    cout << "NIM : 231011401210" << endl;
    cout << "Mata Kuliah: Algoritma Pemrograman 2" << endl;
    cout << "Soal 1: Huffman Encoding" << endl;

    string text;
    cout << "Masukkan string: ";
    getline(cin, text);

    huffmanEncoding(text);

    return 0;
}
```

Screenshoot:



Nomor 2:

Jawab :

Untuk masalah ini, kita bisa memanfaatkan struktur data yang efisien seperti hash set atau hash map untuk mempercepat pencarian pasangan angka. Dengan cara ini, kita dapat menemukan semua pasangan dengan jumlah tertentu (K) dalam waktu yang lebih cepat dibandingkan metode brute-force.

Algoritma

1.Langkah pertama: Buat hash set (atau hash map) untuk menyimpan elemen-elemen dari array pertama.

2.Langkah kedua: Iterasi melalui elemen-elemen array kedua. Untuk setiap elemen, cek apakah selisih antara K dan elemen tersebut sudah ada di hash set yang berisi elemen-elemen array pertama.

3.Langkah ketiga: Jika ada pasangan yang jumlahnya sama dengan K, simpan pasangan tersebut.

Kompleksitas Waktu:

•Membaca elemen-elemen array pertama ke dalam hash set membutuhkan waktu O(n) dengan n sebagai panjang array pertama.

•Iterasi melalui array kedua membutuhkan waktu O(m) dengan m sebagai panjang array kedua.

•Setiap pencarian dalam hash set memerlukan waktu O(1) rata-rata.

Jadi, kompleksitas waktu total adalah O(n + m), di mana n adalah panjang array pertama dan m adalah panjang array kedua.

Kompleksitas Ruang:

•Hash set memerlukan ruang O(n) untuk menyimpan elemen-elemen array pertama.

Jadi, kompleksitas ruang total adalah O(n).

Code Dalam C ++:

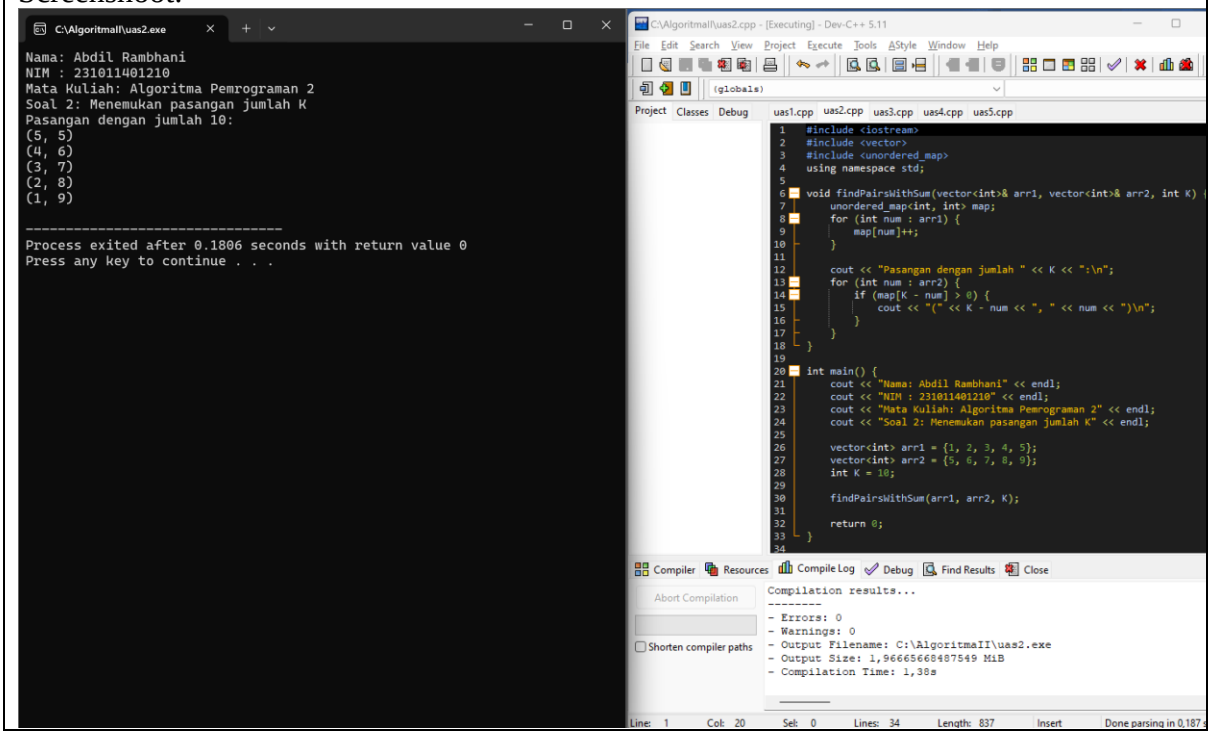
Tanggal: 08/01/2025	menemukan pasangan bilangan dengan jumlah tertentu K (Soal No 2)

Source Code:

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

void findPairsWithSum(vector<int>& arr1, vector<int>& arr2, int K) {
```

```
    unordered_map<int, int> map;
    for (int num : arr1) {
        map[num]++;
    }

    cout << "Pasangan dengan jumlah " << K << ":\n";
    for (int num : arr2) {
        if (map[K - num] > 0) {
            cout << "(" << K - num << ", " << num << ")\n";
        }
    }
}

int main() {
    cout << "Nama: Abdil Rambhani" << endl;
    cout << "NIM : 231011401210" << endl;
    cout << "Mata Kuliah: Algoritma Pemrograman 2" << endl;
    cout << "Soal 2: Menemukan pasangan jumlah K" << endl;

    vector<int> arr1 = {1, 2, 3, 4, 5};
    vector<int> arr2 = {5, 6, 7, 8, 9};
    int K = 10;

    findPairsWithSum(arr1, arr2, K);

    return 0;
}
```

Screenshoot:



Nomor 3: Quick Sort

Kode ini merupakan implementasi algoritma **Quick Sort** dengan gaya **fungsional** dalam bahasa C++. Algoritma Quick Sort ini memanfaatkan prinsip **divide and conquer**, di mana array dibagi menjadi dua bagian berdasarkan elemen pivot, kemudian kedua bagian tersebut diurutkan secara rekursif. Fungsi quickSortFunctional menerima sebuah array arr, kemudian memilih elemen pertama array sebagai **pivot**. Jika array memiliki 0 atau 1 elemen, fungsi langsung mengembalikan array tersebut karena sudah terurut. Selanjutnya, elemen-elemen yang lebih kecil dari atau sama dengan pivot dimasukkan ke dalam array less, sementara elemen yang lebih besar dimasukkan ke dalam array greater. Fungsi quickSortFunctional dipanggil secara rekursif untuk mengurutkan kedua bagian tersebut, dan hasil dari rekursi ini digabungkan dengan pivot di tengahnya, menghasilkan array yang terurut. Di dalam fungsi main, program mencetak informasi mengenai nama, NIM, dan mata kuliah, kemudian memanggil fungsi quickSortFunctional untuk mengurutkan array {3, 6, 8, 10, 1, 2, 1}. Setelah itu, hasil array yang sudah terurut dicetak ke layar. Kode ini menggunakan gaya fungsional karena seluruh operasi pengurutan dilakukan secara rekursif, tanpa mengubah array secara langsung, melainkan dengan menggabungkan hasil rekursi untuk menghasilkan array yang terurut.

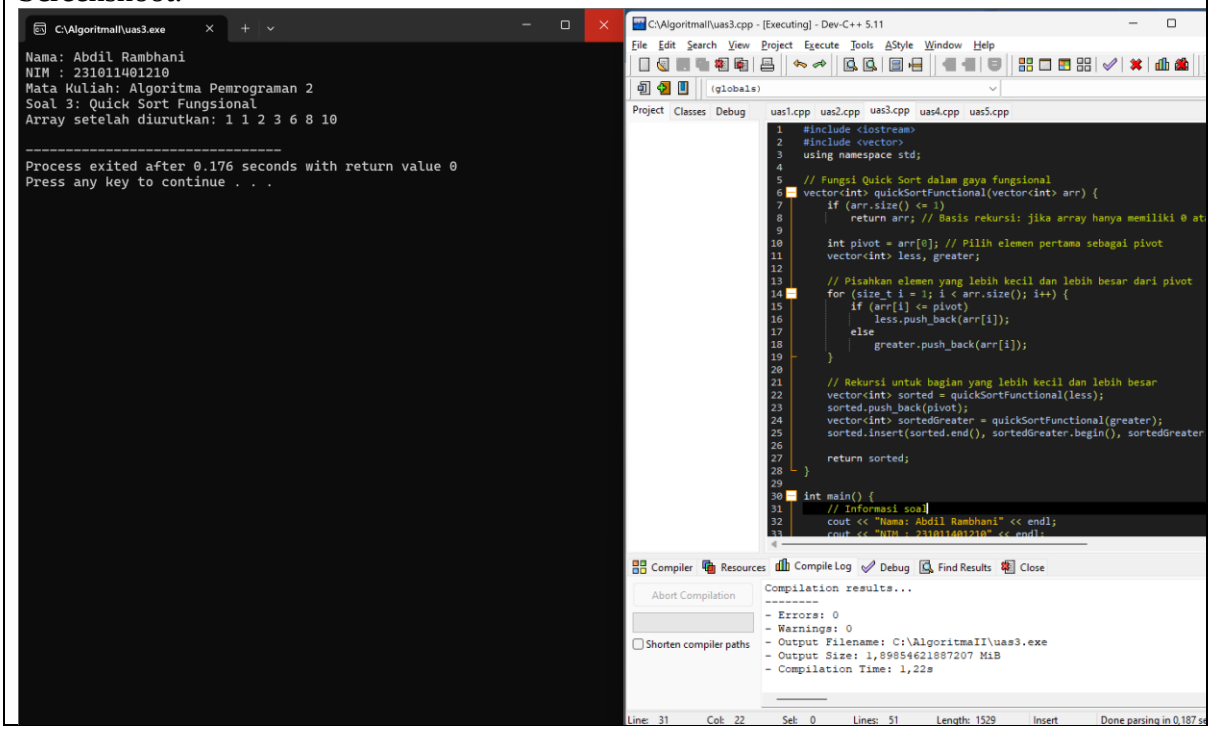
Tanggal: 08/01/2025	Quick Sort (Soal No 3)

Source Code:

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

// Fungsi Quick Sort dalam gaya fungsional
vector<int> quickSortFunctional(vector<int> arr) {
    if (arr.size() <= 1)
        return arr; // Basis rekursi: jika array hanya memiliki 0 atau 1 elemen, sudah terurut

    int pivot = arr[0]; // Pilih elemen pertama sebagai pivot
    vector<int> less, greater;

    // Pisahkan elemen yang lebih kecil dan lebih besar dari pivot
    for (size_t i = 1; i < arr.size(); i++) {
        if (arr[i] <= pivot)
            less.push_back(arr[i]);
        else
            greater.push_back(arr[i]);
    }

    // Rekursi untuk bagian yang lebih kecil dan lebih besar
    vector<int> sorted = quickSortFunctional(less);
    sorted.push_back(pivot);
    vector<int> sortedGreater = quickSortFunctional(greater);
    sorted.insert(sorted.end(), sortedGreater.begin(), sortedGreater.end());
```

```
    return sorted;
}

int main() {
    // Informasi soal
    cout << "Nama: Abdil Rambhani" << endl;
    cout << "NIM : 231011401210" << endl;
    cout << "Mata Kuliah: Algoritma Pemrograman 2" << endl;
    cout << "Soal 3: Quick Sort Fungsional" << endl;

    // Data yang akan diurutkan
    vector<int> arr = {3, 6, 8, 10, 1, 2, 1};

    // Panggil fungsi Quick Sort
    vector<int> sortedArr = quickSortFunctional(arr);

    // Cetak hasil pengurutan
    cout << "Array setelah diurutkan: ";
    for (int num : sortedArr)
        cout << num << " ";
    cout << endl;

    return 0;
}
```

Screenshoot:



Nomor 4: Analisis dan Implementasi Algoritma Radix Sort, Quick Sort, dan Merge Sort

Penjelasan Singkat:
Kompleksitas Waktu dan Ruang:

Radix Sort:
Waktu: O(d * (n + b)), di mana d adalah jumlah digit terbesar dan b adalah basis (radix). Efisien untuk data integer dalam rentang terbatas.
Ruang: O(n + b) untuk array dan penyimpanan tambahan.

Quick Sort:
Waktu: O(n log n) rata-rata, $O(n^2)$ dalam kasus terburuk.
Ruang: O(log n) karena rekursi.

Merge Sort:
Waktu: O(n log n) di semua kasus.
Ruang: O(n) karena memerlukan array tambahan.
Perbandingan:

Radix Sort lebih unggul saat bekerja dengan data integer dalam rentang sempit karena tidak menggunakan operasi perbandingan.

Quick Sort sering lebih cepat untuk data yang besar karena overhead kecil, meskipun ada risiko kasus terburuk.

Merge Sort stabil dan selalu memiliki kompleksitas waktu O(n log n), tetapi memerlukan lebih banyak memori.

Kode C++: Berikut implementasi fungsi Radix Sort, Quick Sort, dan Merge Sort dalam satu program:

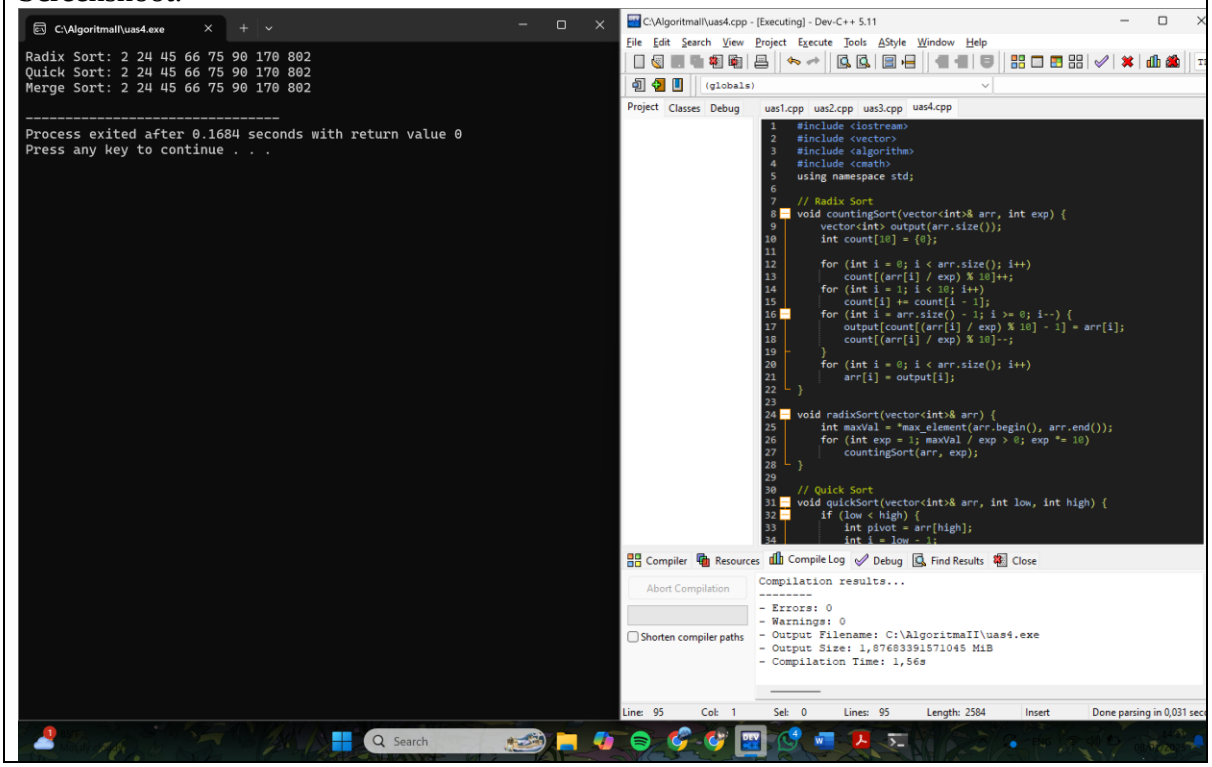
Tanggal: 08/01/2025	Analisis dan Implementasi Algoritma Radix Sort, Quick Sort, dan Merge Sort (Soal No 4)

Source Code:

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

// Fungsi Quick Sort dalam gaya fungsional
vector<int> quickSortFunctional(vector<int> arr) {
    if (arr.size() <= 1)
        return arr; // Basis rekursi: jika array hanya memiliki 0 atau 1 elemen, sudah terurut

    int pivot = arr[0]; // Pilih elemen pertama sebagai pivot
```

```
    vector<int> less, greater;

    // Pisahkan elemen yang lebih kecil dan lebih besar dari pivot
    for (size_t i = 1; i < arr.size(); i++) {
        if (arr[i] <= pivot)
            less.push_back(arr[i]);
        else
            greater.push_back(arr[i]);
    }

    // Rekursi untuk bagian yang lebih kecil dan lebih besar
    vector<int> sorted = quickSortFunctional(less);
    sorted.push_back(pivot);
    vector<int> sortedGreater = quickSortFunctional(greater);
    sorted.insert(sorted.end(), sortedGreater.begin(), sortedGreater.end());

    return sorted;
}

int main() {
    // Informasi soal
    cout << "Nama: Abdil Rambhani" << endl;
    cout << "NIM : 231011401210" << endl;
    cout << "Mata Kuliah: Algoritma Pemrograman 2" << endl;
    cout << "Soal 3: Quick Sort Fungsional" << endl;

    // Data yang akan diurutkan
    vector<int> arr = {3, 6, 8, 10, 1, 2, 1};

    // Panggil fungsi Quick Sort
    vector<int> sortedArr = quickSortFunctional(arr);

    // Cetak hasil pengurutan
    cout << "Array setelah diurutkan: ";
    for (int num : sortedArr)
        cout << num << " ";
    cout << endl;

    return 0;
}
```

Screenshoot:



Nomor 5: Program untuk Gambar Fractal

Kode ini digunakan untuk menghasilkan visualisasi Segitiga Sierpinski menggunakan metode permainan kekacauan (Chaos Game). Pertama, tiga titik sudut segitiga sama sisi didefinisikan, yaitu (0, 0), (1, 0), dan (0.5, 0.866). Titik awal dipilih di tengah segitiga, yaitu di koordinat (0.5, 0.5). Kemudian, program memilih secara acak salah satu dari tiga titik sudut tersebut, lalu titik saat ini bergerak setengah jalan menuju titik yang terpilih. Proses ini diulang sebanyak iterations kali (misalnya 50.000 kali) untuk menghasilkan pola titik yang membentuk fraktal Segitiga Sierpinski. Titik-titik yang dihasilkan kemudian diplot menggunakan matplotlib dengan warna biru dan ukuran kecil. Plot ditampilkan tanpa axis agar tampak lebih bersih, dan diberi judul "Sierpinski Triangle". Semakin banyak iterasi yang dilakukan, semakin halus dan mendetail fraktalnya.

Tanggal: 08/01/2025	Analisis dan Implementasi Algoritma Radix Sort, Quick Sort, dan Merge Sort (Soal No 4)

```
import matplotlib.pyplot as plt
import random

def sierpinski_triangle(iterations):
    # Define the vertices of the triangle
    vertices = [(0, 0), (1, 0), (0.5, 0.866)]  # Equilateral triangle

    # Start with an initial point
    x, y = 0.5, 0.5

    # Lists to store the points
    x_vals = [x]
    y_vals = [y]

    for _ in range(iterations):
        # Randomly choose a vertex
        vx, vy = random.choice(vertices)

        # Move halfway towards the chosen vertex
        x = (x + vx) / 2
        y = (y + vy) / 2

        # Append the new point to the lists
        x_vals.append(x)
        y_vals.append(y)

    # Plot the points
    plt.figure(figsize=(6, 6))
    plt.scatter(x_vals, y_vals, s=0.1, color='blue')
    plt.axis('off')  # Hide axes for a cleaner look
```

```
    plt.title("Sierpinski Triangle", fontsize=14)
    plt.show()

# Generate the fractal with 50,000 points
if __name__ == "__main__":
    print("Generating Sierpinski Triangle...")
    sierpinski_triangle(50000)
```

Screenshoot: