

UAS

ALGORITMA DAN PEMROGAMAN II



Muhammad Raihan
231011401201
03TPLP029

No 1.

Buatlah sebuah program yang menerima input berupa string dan melakukan kompresi string menggunakan algoritma Huffman. Jelaskan bagaimana pohon Huffman dibangun dan bagaimana proses encoding dan decoding dilakukan dalam bahasa pemrograman C++

Source Code :

```
#include <iostream>
#include <string>
#include <queue>
#include <vector>
#include <map>

using namespace std;

// Struktur node pohon Huffman
struct Node {
    char ch;
    int freq;
    Node *left;
    Node *right;

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

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

// Membangun pohon Huffman
Node* buildHuffmanTree(map<char, int>& freq) {
    priority_queue<Node*, vector<Node*>, Compare> pq;

    for (auto pair : freq) {
        pq.push(new Node(pair.first, pair.second));
    }

    while (pq.size() > 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();
        Node* newNode = new Node('\0', left->freq + right->freq);
        newNode->left = left;
        newNode->right = right;
        pq.push(newNode);
    }
}
```

```

    return pq.top();
}

// Membuat kode Huffman
void buildHuffmanCode(Node* root, string code, map<char, string>& huffmanCode) {
    if (root == nullptr) return;

    if (root->ch != '\0') {
        huffmanCode[root->ch] = code;
    }

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

// Mengencode string
string encode(string text, map<char, string>& huffmanCode) {
    string encodedText = "";
    for (char c : text) {
        encodedText += huffmanCode[c];
    }
    return encodedText;
}

// Mendekode string (memerlukan pohon Huffman yang sama)
string decode(string encodedText, Node* root) {
    string decodedText = "";
    Node* current = root;
    for (char bit : encodedText) {
        if (bit == '0') {
            current = current->left;
        } else {
            current = current->right;
        }

        if (current->left == nullptr && current->right == nullptr) {
            decodedText += current->ch;
            current = root;
        }
    }
    return decodedText;
}

int main() {

```

```

        cout << "Nama : Muhammad Raihan" << endl;
        cout << "NIM : 231011401201 " << endl;
        cout << "Kelas : 03TLP029" << endl << endl;

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

    // Menghitung frekuensi karakter
    map<char, int> freq;
    for (char c : text) {
        freq[c]++;
    }

    // Membangun pohon Huffman
    Node* root = buildHuffmanTree(freq);

    // Membuat kode Huffman
    map<char, string> huffmanCode;
    buildHuffmanCode(root, "", huffmanCode);

    // Menampilkan kode Huffman
    cout << "Kode Huffman:\n";
    for (auto pair : huffmanCode) {
        cout << pair.first << " : " << pair.second << endl;
    }

    // Mengencode teks
    string encodedText = encode(text, huffmanCode);
    cout << "Teks terencode: " << encodedText << endl;

    // Mendekode teks
    string decodedText = decode(encodedText, root);
    cout << "Teks terdecode: " << decodedText << endl;

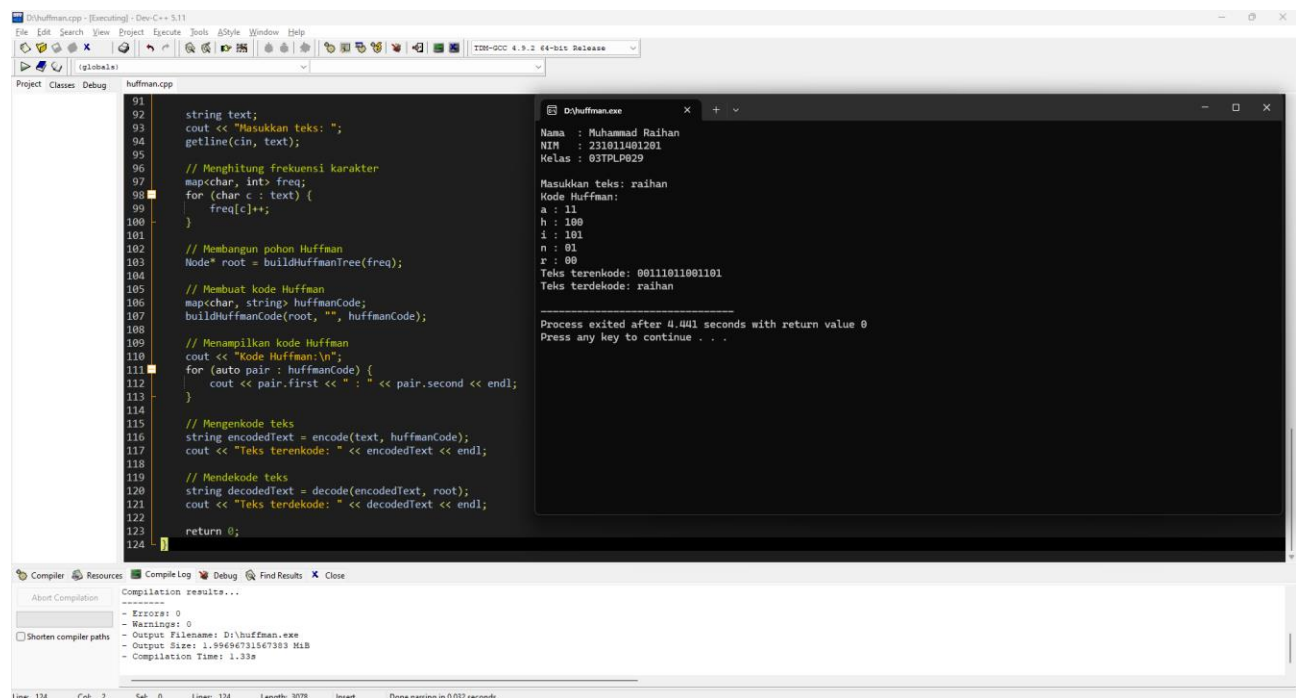
    return 0;
}

```

Penjelasan

1. **Struktur Node:** Merepresentasikan node dalam pohon Huffman, berisi karakter, frekuensi, dan pointer ke anak kiri dan kanan.
2. **buildHuffmanTree:**
 - Menerima map frekuensi karakter.
 - Membuat priority queue (min-heap) untuk menyimpan node berdasarkan frekuensi.
 - Mengambil dua node dengan frekuensi terkecil, membuat node baru sebagai parent-nya, dan memasukkannya kembali ke queue.
 - Proses ini diulang hingga hanya tersisa satu node (root).
3. **buildHuffmanCode:**
 - Melakukan traversal rekursif pada pohon Huffman.
 - Setiap cabang kiri diberi kode "0" dan cabang kanan diberi kode "1".
 - Ketika mencapai leaf node (karakter), kode yang terbentuk disimpan dalam huffmanCode.
4. **encode:** Mengganti setiap karakter dalam teks asli dengan kode Huffman yang sesuai.
5. **decode:** Menerima teks terencode dan pohon Huffman. Melakukan traversal pada pohon berdasarkan bit dalam teks terencode untuk mendapatkan karakter aslinya.

Output :



The screenshot displays a C++ IDE with a source code editor on the left and a console window on the right. The source code implements a Huffman tree algorithm, including functions for building the tree, generating codes, and encoding/decoding text. The console output shows the user's input and the program's results.

```
91 string text;
92 cout << "Masukkan teks: ";
93 getline(cin, text);
94
95 // Menghitung frekuensi karakter
96 map<char, int> freq;
97 for (char c : text) {
98     freq[c]++;
99 }
100
101 // Membangun pohon Huffman
102 Node* root = buildHuffmanTree(freq);
103
104 // Membuat kode Huffman
105 map<char, string> huffmanCode;
106 buildHuffmanCode(root, "", huffmanCode);
107
108 // Menampilkan kode Huffman
109 cout << "Kode Huffman:\n";
110 for (auto pair : huffmanCode) {
111     cout << pair.first << " : " << pair.second << endl;
112 }
113
114 // Mengencode teks
115 string encodedText = encode(text, huffmanCode);
116 cout << "Teks terencode: " << encodedText << endl;
117
118 // Mendekode teks
119 string decodedText = decode(encodedText, root);
120 cout << "Teks terdecode: " << decodedText << endl;
121
122 return 0;
123
124
```

Console Output:

```
Nama : Muhammad Raihan
NIM : 231010401201
Kelas : 03TLP029

Masukkan teks: raihan
Kode Huffman:
a : 11
h : 100
i : 101
n : 01
r : 00
Teks terencode: 00111011001101
Teks terdecode: raihan

Process exited after 4.041 seconds with return value 0
Press any key to continue . . .
```

Compiler Output:

```
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: D:\huffman.exe
- Output Size: 1.89696731567303 KiB
- Compilation Time: 1.33s
```

No 2.

Diberikan dua buah array yang berisi bilangan bulat. Tuliskan algoritma yang efisien untuk menemukan semua pasangan bilangan dari kedua array tersebut yang jumlahnya sama dengan suatu nilai K yang diberikan dan buat program dalam C++. Analisis kompleksitas waktu dan ruang dari algoritma yang Anda buat.

Source Code :

```
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;

void findPairs(vector<int>& arr1, vector<int>& arr2, int K) {
    unordered_map<int, bool> map1; // Menyimpan elemen arr1

    // Memasukkan elemen arr1 ke hash map
    for (int num : arr1) {
        map1[num] = true;
    }

    cout << "Pasangan yang jumlahnya " << K << ":\n";
    for (int num2 : arr2) {
        int complement = K - num2;
        if (map1.count(complement)) { // Memeriksa apakah complement ada di map
            cout << "(" << complement << ", " << num2 << ")\n";
        }
    }
}

int main() {

    cout << "Nama : Muhammad Raihan" << endl;
    cout << "NIM : 231011401201" << endl;
    cout << "Kelas : 03TPLP029" << endl << endl;

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

    findPairs(arr1, arr2, K);

    arr1 = {1, 2, 3, 4, 5};
    arr2 = {1, 2, 3, 4, 5};
    K = 6;
    findPairs(arr1, arr2, K);
}
```

```
    return 0;  
}
```

Penjelasan:

Kompleksitas Waktu:

- Memasukkan elemen ke hash map: $O(m)$, di mana m adalah ukuran `arr1`.
- Iterasi melalui `arr2`: $O(n)$, di mana n adalah ukuran `arr2`.
- Pencarian di hash map (rata-rata): $O(1)$.

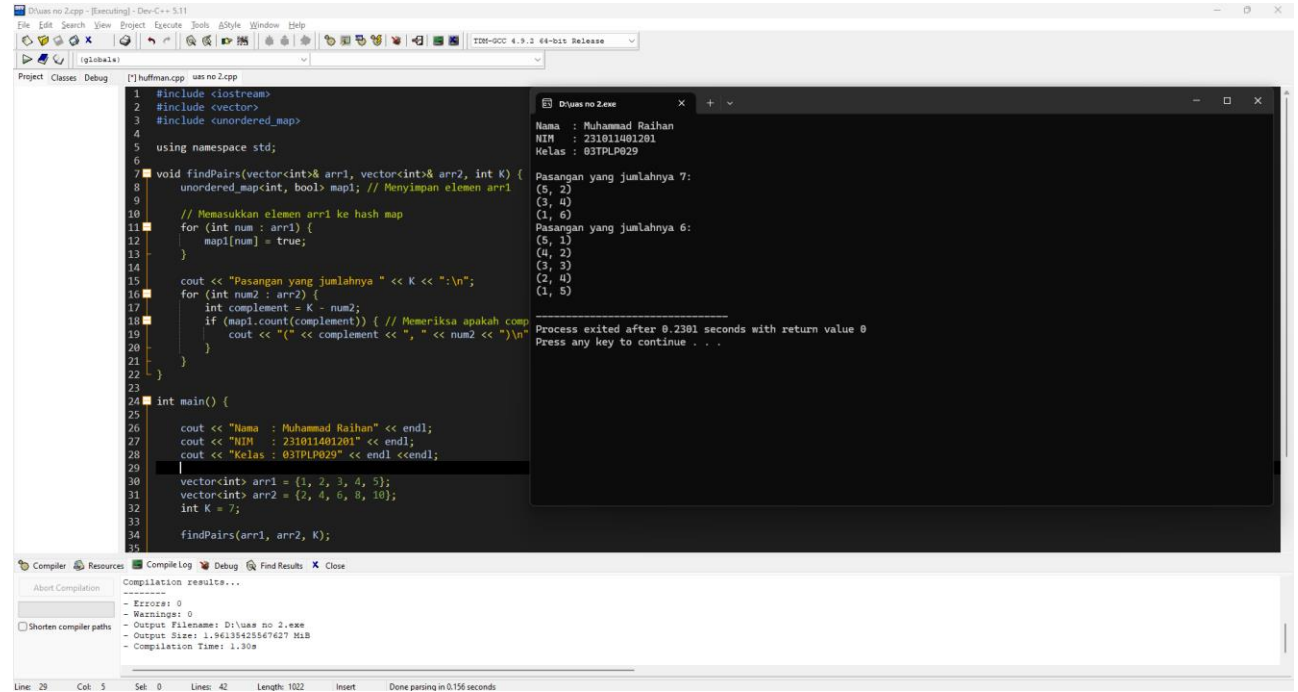
Jadi, kompleksitas waktu totalnya adalah $O(m + n)$, yang merupakan kompleksitas linear. Ini jauh lebih efisien daripada solusi brute-force $O(m * n)$.

Kompleksitas Ruang:

- Ruang yang dibutuhkan untuk hash map: $O(m)$, karena kita menyimpan elemen-elemen dari `arr1`.

Jadi, kompleksitas ruangnya adalah $O(m)$.

Output :



The screenshot shows a C++ IDE with a source file named `huffman.cpp` and an output window titled `Dijus no 2.exe`. The code defines a function `findPairs` that takes two vectors of integers and an integer `K`, and prints pairs of numbers from the first vector whose sum is `K`. The `main` function initializes two vectors, `arr1` and `arr2`, and calls `findPairs` with `K=7`.

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4
5 using namespace std;
6
7 void findPairs(vector<int>& arr1, vector<int>& arr2, int K) {
8     unordered_map<int, bool> map1; // Menyimpan elemen arr1
9
10    // Memasukkan elemen arr1 ke hash map
11    for (int num : arr1) {
12        map1[num] = true;
13    }
14
15    cout << "Pasangan yang jumlahnya " << K << " : \n";
16    for (int num2 : arr2) {
17        int complement = K - num2;
18        if (map1.count(complement)) { // Memeriksa apakah comp
19            cout << "(" << complement << ", " << num2 << ") \n";
20        }
21    }
22 }
23
24 int main() {
25     cout << "Nama : Muhammad Raihan" << endl;
26     cout << "NIM : 231011401201" << endl;
27     cout << "Kelas : 03TLP029" << endl << endl;
28
29     vector<int> arr1 = {1, 2, 3, 4, 5};
30     vector<int> arr2 = {2, 4, 6, 8, 10};
31     int K = 7;
32
33     findPairs(arr1, arr2, K);
34 }
35
```

The output window displays the following text:

```
Nama : Muhammad Raihan
NIM : 231011401201
Kelas : 03TLP029

Pasangan yang jumlahnya 7:
(5, 2)
(3, 4)
(1, 6)
Pasangan yang jumlahnya 6:
(5, 1)
(4, 2)
(3, 3)
(2, 4)
(1, 5)

Process exited after 0.2201 seconds with return value 0
Press any key to continue . . .
```

The bottom status bar shows: Line: 29, Col: 5, Sel: 0, Lines: 42, Length: 1022, Insert, Done parsing in 0.156 seconds.

No 3.

Terjemahkan algoritma quick sort ke dalam gaya pemrograman fungsional menggunakan bahasa C++. Jelaskan bagaimana Anda mengimplementasikan rekursi dan menghindari penggunaan loop. Bagaimana kinerja versi fungsional ini dibandingkan dengan versi imperatif?

Source Code :

Output :

No 4.

Analisis kompleksitas waktu dan ruang dari algoritma radix sort. Bandingkan dengan algoritma quick sort dan merge sort dan buatlah satu program dalam C++ yang didalamnya terdapat fungsi radix sort, quick sort dan merge sort. Dalam kondisi apa radix sort lebih unggul?

Source Code :

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

using namespace std;
using namespace std::chrono;

// Fungsi untuk mendapatkan digit ke-i dari angka
int getDigit(int num, int i) {
    int divisor = 1;
    for (int j = 0; j < i; j++) {
        divisor *= 10;
    }
}
```

```

    return (num / divisor) % 10;
}

void radixSort(vector<int>& arr) {
    if (arr.empty()) return;
    int maxVal = *max_element(arr.begin(), arr.end());
    int numDigits = 0;
    while (maxVal > 0) {
        maxVal /= 10;
        numDigits++;
    }

    for (int i = 0; i < numDigits; i++) {
        vector<vector<int>> buckets(10);
        for (int num : arr) {
            int digit = getDigit(num, i);
            buckets[digit].push_back(num);
        }
        arr.clear();
        for (auto& bucket : buckets) {
            arr.insert(arr.end(), bucket.begin(), bucket.end());
        }
    }
}

// Fungsi partisi untuk Quick Sort
int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}

void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

```

```

    }
}

// Fungsi merge untuk Merge Sort
void merge(vector<int>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            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(vector<int>& arr, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
    }
}

```

```

        merge(arr, l, m, r);
    }
}

int main() {

    cout << "Nama : Muhammad Raihan" << endl;
    cout << "NIM : 231011401201" << endl;
    cout << "Kelas : 03TLP029" << endl << endl;

    vector<int> data = {170, 45, 75, 90, 802, 24, 2, 66};

    vector<int> data_radix = data;
    auto start = high_resolution_clock::now();
    radixSort(data_radix);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "Radix Sort Time: " << duration.count() << " microseconds" << endl;

    vector<int> data_quick = data;
    start = high_resolution_clock::now();
    quickSort(data_quick, 0, data_quick.size() - 1);
    stop = high_resolution_clock::now();
    duration = duration_cast<microseconds>(stop - start);
    cout << "Quick Sort Time: " << duration.count() << " microseconds" << endl;

    vector<int> data_merge = data;
    start = high_resolution_clock::now();
    mergeSort(data_merge, 0, data_merge.size() - 1);
    stop = high_resolution_clock::now();
    duration = duration_cast<microseconds>(stop - start);
    cout << "Merge Sort Time: " << duration.count() << " microseconds" << endl;

    return 0;
}

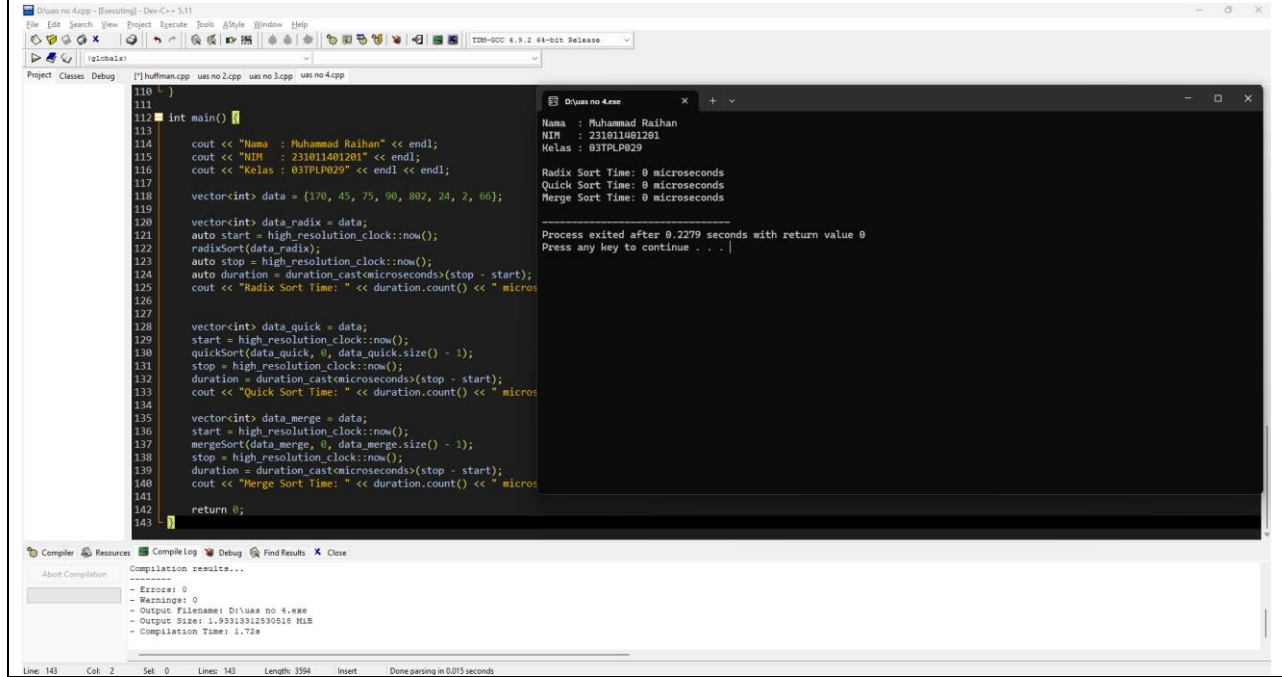
```

Penjelasan:

Radix Sort lebih unggul dalam kondisi

- Data berupa bilangan bulat atau string dengan range yang terbatas dan diketahui.
- Jumlah digit/karakter (k) relatif kecil dibandingkan jumlah elemen (n).
- Kinerja linear $O(n)$ sangat dibutuhkan.

Output:



The screenshot shows a C++ IDE with a source code editor on the left and a console window on the right. The source code is a C++ program that sorts an array of integers using Radix, Quick, and Merge Sort algorithms and measures their execution times. The console window displays the output of the program, including the user's name, NIM, and the execution times for each sorting algorithm.

```
110 }
111
112 int main() {
113
114     cout << "Nama : Muhammad Raihan" << endl;
115     cout << "NIM : 231011401201" << endl;
116     cout << "Kelas : 03TLP029" << endl << endl;
117
118     vector<int> data = {170, 45, 75, 90, 802, 24, 2, 60};
119
120     vector<int> data_radix = data;
121     auto start = high_resolution_clock::now();
122     radixSort(data_radix);
123     auto stop = high_resolution_clock::now();
124     auto duration = duration_cast<microseconds>(stop - start);
125     cout << "Radix Sort Time: " << duration.count() << " micros";
126
127
128     vector<int> data_quick = data;
129     start = high_resolution_clock::now();
130     quickSort(data_quick, 0, data_quick.size() - 1);
131     stop = high_resolution_clock::now();
132     duration = duration_cast<microseconds>(stop - start);
133     cout << "Quick Sort Time: " << duration.count() << " micros";
134
135     vector<int> data_merge = data;
136     start = high_resolution_clock::now();
137     mergeSort(data_merge, 0, data_merge.size() - 1);
138     stop = high_resolution_clock::now();
139     duration = duration_cast<microseconds>(stop - start);
140     cout << "Merge Sort Time: " << duration.count() << " micros";
141
142     return 0;
143 }
```

Output:

```
Nama : Muhammad Raihan
NIM : 231011401201
Kelas : 03TLP029

Radix Sort Time: 0 microseconds
Quick Sort Time: 0 microseconds
Merge Sort Time: 0 microseconds

Process exited after 0.2279 seconds with return value 0
Press any key to continue . . .
```

Compilation results...

```
- Errors: 0
- Warnings: 0
- Output Filename: D:\uas no 4.exe
- Output Size: 1.9313312530518 MiB
- Compilation Time: 1.72s
```

No 5.

Buatlah sebuah program yang dapat menghasilkan gambar fractal sederhana, seperti Sierpinski triangle atau Mandelbrot set. Jelaskan prinsip rekursif yang mendasari pembentukan fractal dan bagaimana Anda mengimplementasikannya dalam C++.

Source Code :**Output :**