

Algoritma



Name: Muhammad Dzakwan Fadhlurrohman

NIM :231011403237

Class : 02TPLP029

DEPARTEMEN TEKNOLOGI INFORMASI

FAKULTAS ILMU KOMPUTER

UNIVERSITAS PAMULANG

Jalan Raya Puspitek, Buaran, Kecamatan Pamulang, Kota Tangsel, Banten

Jawaban soal nomor 1.

Tanggal: 08-01-2025

Judul: kompresi string menggunakan algoritma Huffman

Penjelasan Algoritma Huffman:

1. Pembangunan Pohon Huffman:

- Hitung frekuensi setiap karakter dalam string.
- Buat node untuk setiap karakter dan masukkan ke dalam min-heap (priority queue).
- Iterasi selama lebih dari satu node dalam heap:
 - Ambil dua node dengan frekuensi terendah.
 - Buat node baru dengan frekuensi total dari dua node tersebut sebagai parent, dan kedua node menjadi anaknya.
 - Masukkan node baru kembali ke heap.
- Hasil akhirnya adalah root dari pohon Huffman.

2. Proses Encoding:

- Traversal pohon Huffman untuk mendapatkan kode biner untuk setiap karakter.
- Simpan kode biner untuk setiap karakter dalam tabel.

3. Proses Decoding:

Gunakan kode biner dan pohon Huffman untuk mengubah kode biner kembali menjadi string aslinya.

Source code:

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

// Node Huffman
struct HuffmanNode {
    char character;
    int frequency;
    HuffmanNode* left;
    HuffmanNode* right;

    HuffmanNode(char ch, int freq) : character(ch), frequency(freq), left(nullptr),
right(nullptr) {}
};

// Comparator untuk priority queue
struct Compare {
    bool operator()(HuffmanNode* a, HuffmanNode* b) {
        return a->frequency > b->frequency;
    }
};
```

```

};

// Membuat pohon Huffman
HuffmanNode* buildHuffmanTree(const string& text) {
    unordered_map<char, int> frequencyMap;
    for (char ch : text) frequencyMap[ch]++;

    priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare> pq;
    for (auto& pair : frequencyMap) {
        pq.push(new HuffmanNode(pair.first, pair.second));
    }

    while (pq.size() > 1) {
        HuffmanNode* left = pq.top(); pq.pop();
        HuffmanNode* right = pq.top(); pq.pop();
        HuffmanNode* parent = new HuffmanNode('\0', left->frequency + right->frequency);
        parent->left = left;
        parent->right = right;
        pq.push(parent);
    }

    return pq.top();
}

// Traversal pohon untuk menghasilkan kode Huffman
void generateHuffmanCodes(HuffmanNode* root, string code, unordered_map<char,
string>& huffmanCodes) {
    if (!root) return;

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

    generateHuffmanCodes(root->left, code + "0", huffmanCodes);
    generateHuffmanCodes(root->right, code + "1", huffmanCodes);
}

// Encoding string
string encode(const string& text, unordered_map<char, string>& huffmanCodes) {
    string encodedString = "";
    for (char ch : text) {
        encodedString += huffmanCodes[ch];
    }
    return encodedString;
}

// Decoding string
string decode(HuffmanNode* root, const string& encodedString) {
    string decodedString = "";
    HuffmanNode* current = root;

```

```

for (char bit : encodedString) {
    if (bit == '0') current = current->left;
    else current = current->right;

    if (!current->left && !current->right) {
        decodedString += current->character;
        current = root;
    }
}
return decodedString;
}

int main() {
    // Menampilkan informasi Nama, Kelas, dan NIM
    cout << "=====\n";
    cout << "Nama : Muhammad Dzakwan F\n";
    cout << "Kelas: 03TPLP029\n";
    cout << "Nim : 231011403237\n";
    cout << "=====\n\n";

    string text;
    cout << "Masukkan string untuk dikompresi: ";
    cin >> text;

    HuffmanNode* root = buildHuffmanTree(text);

    unordered_map<char, string> huffmanCodes;
    generateHuffmanCodes(root, "", huffmanCodes);

    cout << "Kode Huffman untuk setiap karakter:\n";
    for (auto& pair : huffmanCodes) {
        cout << pair.first << ": " << pair.second << endl;
    }

    string encodedString = encode(text, huffmanCodes);
    cout << "\nString terkompresi: " << encodedString << endl;

    string decodedString = decode(root, encodedString);
    cout << "String setelah dekompresi: " << decodedString << endl;

    return 0;
}

```

Algoritma:

Input: String yang akan dikompresi.

Output: Kode Huffman untuk setiap karakter, string terkompresi, dan hasil dekompresi.

Langkah-langkah:

1. Hitung Frekuensi Karakter:

- Ambil setiap karakter dalam string.
- Hitung berapa kali setiap karakter muncul dan simpan dalam struktur data seperti hashmap atau array.

2. Bangun Min-Heap:

- Buat sebuah min-heap (priority queue) yang menyimpan node-noda pohon. Setiap node berisi:
 - Karakter.
 - Frekuensi kemunculan.
 - Pointer ke anak kiri dan kanan (awalnya `null`).
- Masukkan semua karakter beserta frekuensinya ke dalam min-heap.

3. Bangun Pohon Huffman:

- Ulangi proses berikut sampai hanya tersisa satu node dalam heap:
 1. Ambil dua node dengan frekuensi terendah dari heap.
 2. Buat node baru yang menjadi parent kedua node tersebut:
 - Frekuensi node baru adalah jumlah frekuensi kedua node anak.
 - Node kiri adalah node dengan frekuensi lebih kecil.
 - Node kanan adalah node lainnya.
 3. Masukkan node baru ke dalam heap.
- Node terakhir yang tersisa adalah root dari pohon Huffman.

4. Buat Kode Huffman:

- Traversal pohon Huffman secara rekursif untuk menghasilkan kode Huffman:
 - Beri nilai 0 pada cabang kiri dan 1 pada cabang kanan.
 - Simpan kode biner untuk setiap karakter dalam tabel (hashmap).

5. Encode String:

- Ganti setiap karakter dalam string dengan kode biner yang sesuai dari tabel kode Huffman.
- Gabungkan hasilnya menjadi string terkompresi.

6. Decode String:

- Mulai dari root pohon Huffman.
- Baca setiap bit dalam string terkompresi:
 - Jika 0, pindah ke anak kiri.
 - Jika 1, pindah ke anak kanan.
- Jika mencapai daun, tambahkan karakter ke string hasil dekompresi dan kembali ke root.

Screenshot source code:

```
uas-1.cpp
1 #include <iostream>
2 #include <queue>
3 #include <unordered_map>
4 #include <vector>
5 using namespace std;
6
7 // Node Huffman
8 struct HuffmanNode {
9     char character;
10    int frequency;
11    HuffmanNode* left;
12    HuffmanNode* right;
13
14    HuffmanNode(char ch, int freq) : character(ch), frequency(freq), left(nullptr), right(nullptr) {}
15 };
16
17 // Comparator untuk priority queue
18 struct Compare {
19     bool operator()(HuffmanNode* a, HuffmanNode* b) {
20         return a->frequency > b->frequency;
21     }
22 };
23
24 // Membuat pohon Huffman
25 HuffmanNode* buildHuffmanTree(const string& text) {
26     unordered_map<char, int> frequencyMap;
27     for (char ch : text) frequencyMap[ch]++;
28
29     priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare> pq;
30     for (auto& pair : frequencyMap) {
31         pq.push(new HuffmanNode(pair.first, pair.second));
32     }
33
34     while (pq.size() > 1) {
35         HuffmanNode* left = pq.top(); pq.pop();
36         HuffmanNode* right = pq.top(); pq.pop();
37         HuffmanNode* parent = new HuffmanNode('\0', left->frequency + right->frequency);
38         parent->left = left;
39         parent->right = right;
40         pq.push(parent);
41     }
42
43     return pq.top();
44 }
45
```

Screenshot source code:

```

45 // Traversal pohon untuk menghasilkan kode Huffman
46 void generateHuffmanCodes(HuffmanNode* root, string code, unordered_map<char, string>& huffmanCodes) {
47     if (!root) return;
48     if (root->character != '\0') {
49         huffmanCodes[root->character] = code;
50     }
51     generateHuffmanCodes(root->left, code + "0", huffmanCodes);
52     generateHuffmanCodes(root->right, code + "1", huffmanCodes);
53 }
54
55 // Encoding string
56 string encode(const string& text, unordered_map<char, string>& huffmanCodes) {
57     string encodedString = "";
58     for (char ch : text) {
59         encodedString += huffmanCodes[ch];
60     }
61     return encodedString;
62 }
63
64 // Decoding string
65 string decode(HuffmanNode* root, const string& encodedString) {
66     string decodedString = "";
67     HuffmanNode* current = root;
68     for (char bit : encodedString) {
69         if (bit == '0') current = current->left;
70         else current = current->right;
71
72         if (!current->left && !current->right) {
73             decodedString += current->character;
74             current = root;
75         }
76     }
77     return decodedString;
78 }
79
80
81 }
82

```

Screenshot source code:

```

82
83 int main() {
84     // Menampilkan informasi Nama, Kelas, dan NIM
85     cout << "=====\n";
86     cout << "Nama : Muhammad Dzakwan F\n";
87     cout << "Kelas : 03TPLP029\n";
88     cout << "Nim : 231011403237\n";
89     cout << "=====\n\n";
90
91     string text;
92     cout << "Masukkan string untuk dikompresi: ";
93     cin >> text;
94
95     HuffmanNode* root = buildHuffmanTree(text);
96
97     unordered_map<char, string> huffmanCodes;
98     generateHuffmanCodes(root, "", huffmanCodes);
99
100     cout << "Kode Huffman untuk setiap karakter:\n";
101     for (auto& pair : huffmanCodes) {
102         cout << pair.first << " : " << pair.second << endl;
103     }
104
105     string encodedString = encode(text, huffmanCodes);
106     cout << "\nString terkompresi: " << encodedString << endl;
107
108     string decodedString = decode(root, encodedString);
109     cout << "String setelah dekompresi: " << decodedString << endl;
110
111     return 0;
112 }
113

```

```

C:\Users\sasus\Documents\semester 3\tugas cpp\...
Nama : Muhammad Dzakwan F
Kelas : 03TPLP029
Nim : 231011403237
=====
Masukkan string untuk dikompresi: 1
Kode Huffman untuk setiap karakter:
1:
String terkompresi:
String setelah dekompresi:
-----
Process exited after 25.01 seconds with return value
0
Press any key to continue . . .

```

Jawaban soal nomor 2.

Tanggal: 08-01-2025	Judul: Analisis kompleksitas waktu dan ruang.
Analisis kompleksitas waktu dan ruang.	
1. Kompleksitas Waktu:	
<ul style="list-style-type: none">• Langkah 1: Menyimpan Elemen dari Array A dalam Hash Map<ul style="list-style-type: none">○ Kita mengiterasi seluruh elemen dalam array A sekali. Jika panjang array A adalah n, maka waktu yang dibutuhkan untuk menyimpan elemen-elemen A dalam hash map adalah $O(n)$.○ Operasi Penyimpanan dalam Hash Map memiliki waktu rata-rata $O(1)$ per elemen, karena operasi <code>mapA[a]++</code> dilakukan dalam waktu konstan.• Langkah 2: Iterasi melalui Array B dan Cek di Hash Map<ul style="list-style-type: none">○ Kita mengiterasi seluruh elemen dalam array B. Jika panjang array B adalah m, maka kita melakukan iterasi sebanyak m kali.○ Setiap kali kita melakukan pengecekan <code>mapA.find(needed)</code>, yang dilakukan di dalam hash map, waktu rata-rata untuk pencarian adalah $O(1)$ per elemen.• Total Kompleksitas Waktu:<ul style="list-style-type: none">○ Waktu yang dibutuhkan untuk proses ini adalah $O(n + m)$, di mana n adalah panjang array A dan m adalah panjang array B.	
2. Kompleksitas Ruang:	
<ul style="list-style-type: none">• Ruang untuk Hash Map:<ul style="list-style-type: none">○ Hash map <code>mapA</code> menyimpan elemen-elemen dari array A, sehingga ruang yang dibutuhkan oleh hash map adalah $O(n)$, di mana n adalah panjang array A.• Ruang untuk Hasil Pasangan:<ul style="list-style-type: none">○ Pada worst-case, semua pasangan dalam array B dapat ditemukan dalam array A. Jadi, ruang yang dibutuhkan untuk menyimpan pasangan-pasangan hasil adalah $O(\min(n, m))$, di mana n adalah panjang array A dan m adalah panjang array B.• Total Kompleksitas Ruang:<ul style="list-style-type: none">○ $O(n)$ untuk hash map dan $O(\min(n, m))$ untuk menyimpan hasil pasangan, sehingga total ruang yang dibutuhkan adalah $O(n + \min(n, m))$.	
Source code: <pre>#include <iostream> #include <unordered_map> #include <vector> using namespace std; // Fungsi untuk menampilkan informasi nama, kelas, dan NIM void displayInfo() { cout << "=====\\n"; cout << "Nama : Muhammad Dzakwan F\\n"; cout << "Kelas: 03TPLP029\\n"; }</pre>	

```

    cout << "Nim : 231011403237\n";
    cout << "=====\n\n";
}

// Fungsi untuk menemukan pasangan bilangan
vector<pair<int, int>> findPairs(const vector<int>& A, const vector<int>& B, int K) {
    unordered_map<int, int> mapA;
    vector<pair<int, int>> result;

    // Simpan elemen-elemen dari array A ke dalam hash map
    for (int a : A) {
        mapA[a]++;
    }

    // Iterasi melalui array B
    for (int b : B) {
        int needed = K - b; // Cari nilai yang diperlukan dari A
        if (mapA.find(needed) != mapA.end()) {
            result.push_back({needed, b});
        }
    }

    return result;
}

int main() {
    // Tampilkan informasi nama, kelas, dan NIM
    displayInfo();

    vector<int> A = {1, 2, 3, 4, 5};
    vector<int> B = {4, 5, 6, 7};
    int K = 9;

    vector<pair<int, int>> pairs = findPairs(A, B, K);

    cout << "Pasangan bilangan dengan jumlah " << K << ":\n";
    for (auto& pair : pairs) {
        cout << "(" << pair.first << ", " << pair.second << ")\n";
    }

    return 0;
}

```

Algoritma:

- **Inisialisasi:**

- Siapkan dua array: A dan B, serta nilai K yang merupakan jumlah yang dicari.
- Buat sebuah hash map (`unordered_map`) untuk menyimpan elemen-elemen dari array A.

- **Memasukkan Elemen Array A ke dalam Hash Map:**

- Iterasi seluruh elemen array A dan masukkan setiap elemen ke dalam hash map `mapA` (key = elemen array A, value = jumlah kemunculan).

- **Mencari Pasangan:**

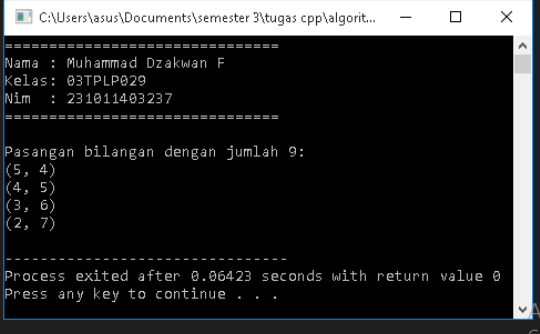
- Iterasi seluruh elemen array B.
- Untuk setiap elemen `b` di array B, hitung nilai yang diperlukan ($needed = K - b$).
- Jika `needed` ada dalam hash map `mapA`, maka pasangan (`needed`, `b`) adalah pasangan yang valid yang jumlahnya sama dengan `K`.

- **Simpan dan Tampilkan Pasangan:**

- Simpan pasangan yang ditemukan dalam sebuah vector atau list.
- Tampilkan semua pasangan yang ditemukan.

Screenshot source code:

```
15 // Fungsi untuk menemukan pasangan bilangan
16 vector<pair<int, int>> findPairs(const vector<int>& A, const vector<int>& B, int K) {
17     unordered_map<int, int> mapA;
18     vector<pair<int, int>> result;
19
20     // Simpan elemen-elemen dari array A ke dalam hash map
21     for (int a : A) {
22         mapA[a]++;
23     }
24
25     // Iterasi melalui array B
26     for (int b : B) {
27         int needed = K - b; // Cari nilai yang diperlukan dari A
28         if (mapA.find(needed) != mapA.end()) {
29             result.push_back({needed, b});
30         }
31     }
32
33     return result;
34 }
35
36 int main() {
37     // Tampilkan informasi nama, kelas, dan NIM
38     displayInfo();
39
40     vector<int> A = {1, 2, 3, 4, 5};
41     vector<int> B = {4, 5, 6, 7};
42     int K = 9;
43
44     vector<pair<int, int>> pairs = findPairs(A, B, K);
45
46     cout << "Pasangan bilangan dengan jumlah " << K << ":\n";
47     for (auto& pair : pairs) {
48         cout << "(" << pair.first << ", " << pair.second << ")\n";
49     }
50
51     return 0;
52 }
```



C:\Users\asus\Documents\semester 3\tugas cpp\algorit...

=====

Nama : Muhammad Dzakwan F
Kelas: 03TPLP029
NIM : 231011403237
=====

Pasangan bilangan dengan jumlah 9:

(5, 4)
(4, 5)
(3, 6)
(2, 7)

Process exited after 0.06423 seconds with return value 0
Press any key to continue . . .

Jawaban soal nomor 3.

Tanggal:08-11-2025	Judul: algoritma quick sort
Penjelasan Implementasi: 1. Fungsi Partition: <ul style="list-style-type: none">o Fungsi <code>partition</code> membagi array ke dalam dua bagian: elemen yang lebih kecil dari pivot (<code>left</code>) dan elemen yang lebih besar dari pivot (<code>right</code>).o Fungsi ini menggunakan rekursi untuk memecah array berdasarkan pivot dan membentuk tiga bagian: bagian kiri, pivot, dan bagian kanan. 2. Fungsi QuickSort: <ul style="list-style-type: none">o Fungsi <code>quickSort</code> adalah fungsi rekursif yang mengurutkan array.o Jika ukuran array kurang dari atau sama dengan 1 (basis rekursi), array akan langsung dikembalikan tanpa perubahan.o Pivot dipilih dari elemen tengah array.o Array dibagi menjadi sub-array kiri dan kanan, kemudian kedua sub-array diurutkan secara rekursif menggunakan panggilan fungsi <code>quickSort</code>.o Hasil rekursi pada sub-array kiri dan kanan digabungkan dengan pivot untuk membentuk array yang terurut. Penghindaran Penggunaan Loop: <ul style="list-style-type: none">• Dalam implementasi ini, semua iterasi dilakukan menggunakan rekursi, baik untuk membagi array dalam fungsi <code>quickSort</code> maupun untuk mempartisi array dalam fungsi <code>partition</code>.• Dengan cara ini, loop <code>for</code> dalam gaya pemrograman imperatif digantikan dengan rekursi, yang merupakan prinsip utama dalam pemrograman fungsional.	
Kinerja Versi Fungsional vs Imperatif: <ul style="list-style-type: none">• Kompleksitas Waktu:<ul style="list-style-type: none">o Baik implementasi fungsional maupun imperatif dari Quick Sort memiliki kompleksitas waktu rata-rata yang sama, yaitu $O(n \log n)$ untuk kasus terbaik dan rata-rata, dan $O(n^2)$ untuk kasus terburuk (ketika pivot selalu elemen terkecil atau terbesar).• Kompleksitas Ruang:<ul style="list-style-type: none">o Versi fungsional membutuhkan ruang ekstra untuk menyimpan array yang dibagi, yang berarti ruang yang digunakan akan lebih besar dibandingkan dengan versi imperatif yang dapat mengurutkan array di tempat (in-place).o Oleh karena itu, versi fungsional mungkin memerlukan lebih banyak ruang dibandingkan dengan versi imperatif yang menggunakan in-place partitioning, yang memiliki $O(\log n)$ ruang rekursi.• Keuntungan Pemrograman Fungsional:<ul style="list-style-type: none">o Kode Lebih Deklaratif: Pemrograman fungsional lebih mengutamakan apa yang harus dilakukan daripada bagaimana melakukannya. Ini membuat kode lebih mudah dibaca dan dipahami.o Immutabilitas: Data tidak diubah, sehingga lebih aman dalam penggunaan multi-threading dan lebih mudah untuk ditangani secara paralel.• Kekurangan Pemrograman Fungsional:<ul style="list-style-type: none">o Kinerja dan Memori: Penggunaan rekursi dan pembentukan array baru di setiap	

langkah dapat menyebabkan overhead kinerja dan penggunaan memori yang lebih tinggi.

- **Keberlanjutan Rekursi:** Jika tidak dilakukan dengan hati-hati, rekursi dapat menyebabkan tumpukan panggilan (stack overflow) pada input yang sangat besar, meskipun dalam implementasi ini tidak ada tumpukan eksplisit.

Source code:

```
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
void profile(){
    cout << "Nama: Muhammad Dzakwan F" << endl;
    cout << "NIM : 231011403237" << endl;
    cout << "Kelas: 03TPLP029" << endl << endl;
}

template <typename T>
std::vector<T> quickSort(const std::vector<T> & arr) {
    // Basis: Jika array kosong atau memiliki satu elemen, kembalikan langsung.
    if (arr.size() <= 1) {
        return arr;
    }

    // Pilih elemen pivot.
    T pivot = arr[0];

    // Gunakan fungsi lambda untuk memisahkan elemen lebih kecil dan lebih besar dari pivot.
    std::vector<T> less, greater;
    std::copy_if(arr.begin() + 1, arr.end(), std::back_inserter(less), [pivot](T x) { return x <=
pivot; });
    std::copy_if(arr.begin() + 1, arr.end(), std::back_inserter(greater), [pivot](T x) { return x
> pivot; });

    // Rekursi untuk bagian kiri dan kanan, lalu gabungkan hasilnya.
    std::vector<T> sortedLess = quickSort(less);
    std::vector<T> sortedGreater = quickSort(greater);

    // Gabungkan hasil (less + pivot + greater).
    sortedLess.push_back(pivot);
    sortedLess.insert(sortedLess.end(), sortedGreater.begin(), sortedGreater.end());

    return sortedLess;
}

// Contoh penggunaan.
```

```

int main() {
    profile();
    std::vector<int> data = {10, 7, 8, 9, 1, 5};
    std::vector<int> sortedData = quickSort(data);

    for (int num : sortedData) {
        std::cout << num << " ";
    }

    return 0;
}

```

Screenshot source code:

The screenshot shows a C++ source code editor with three tabs: uas-1.cpp, uas-2.cpp, and uas-3.cpp. The code in uas-2.cpp implements a quicksort algorithm. It includes necessary headers, a profile function for timing, and a recursive quicksort function. The main function in uas-3.cpp uses the quicksort function to sort an array of integers and prints the sorted result.

```

1 #include <vector>
2 #include <algorithm>
3 #include <iostream>
4 using namespace std;
5 void profile() {
6     cout << "Nama: Muhammad Dzakwan F" << endl;
7     cout << "NIM : 231011403237" << endl;
8     cout << "Kelas: 03TLP029" << endl << endl;
9 }
10
11 template <typename T>
12 std::vector<T> quickSort(const std::vector<T>& arr) {
13     // Basis: Jika array kosong atau memiliki satu elemen, kembalikan langsung.
14     if (arr.size() <= 1) {
15         return arr;
16     }
17
18     // Pilih elemen pivot.
19     T pivot = arr[0];
20
21     // Gunakan fungsi lambda untuk memisahkan elemen lebih kecil dan lebih besar dari pivot.
22     std::vector<T> less, greater;
23     std::copy_if(arr.begin() + 1, arr.end(), std::back_inserter(less), [pivot](T x) { return x <= pivot; });
24     std::copy_if(arr.begin() + 1, arr.end(), std::back_inserter(greater), [pivot](T x) { return x > pivot; });
25
26     // Rekursi untuk bagian kiri dan kanan, lalu gabungkan hasilnya.
27     std::vector<T> sortedLess = quickSort(less);
28     std::vector<T> sortedGreater = quickSort(greater);
29
30     // Gabungkan hasil (less + pivot + greater).
31     sortedLess.push_back(pivot);
32     sortedLess.insert(sortedLess.end(), sortedGreater.begin(), sortedGreater.end());
33
34     return sortedLess;
35 }
36
37 // Contoh penggunaan.
38 int main() {
39     profile();
40     std::vector<int> data = {10, 7, 8, 9, 1, 5};
41     std::vector<int> sortedData = quickSort(data);
42
43     for (int num : sortedData) {
44         std::cout << num << " ";
45     }
46
47     return 0;
48 }

```

The output window shows the following text:

```

C:\Users\asus\Documents\semester 3\tugas cpp\algor...
Nama: Muhammad Dzakwan F
NIM : 231011403237
Kelas: 03TLP029

1 5 7 8 9 10
-----
Process exited after 0.04709 seconds with return value 0
Press any key to continue . . .

```

Jawaban soal nomor 4

Tanggal:08-01-2025

Judul: Analisis Kompleksitas Waktu dan Ruang

Analisis Kompleksitas Waktu dan Ruang

Radix Sort:

1. Kompleksitas Waktu:

- o Jika kita mengurutkan n elemen dengan digit maksimal d dan basis k , kompleksitas waktu adalah $O(d * (n + k))$.
- o Dalam kasus biasa, kita mengambil $k = 10$ (basis desimal), sehingga waktu menjadi $O(d * n)$, di mana d tergantung pada panjang maksimum digit dalam elemen.

2. Kompleksitas Ruang:

- o Radix Sort membutuhkan ruang tambahan untuk menyimpan elemen dalam bucket pada setiap digit. Jadi, kompleksitas ruang adalah $O(n + k)$.

Quick Sort:

1. Kompleksitas Waktu:

- o **Best Case:** $O(n \log n)$ (pivot selalu memisahkan array menjadi dua bagian yang hampir sama).
- o **Worst Case:** $O(n^2)$ (pivot selalu elemen terkecil/terbesar).
- o **Average Case:** $O(n \log n)$.

2. Kompleksitas Ruang:

- o $O(\log n)$ untuk rekursi jika partisi optimal.
- o Dalam kasus terburuk, ruang tambahan untuk rekursi bisa mencapai $O(n)$.

Merge Sort:

1. Kompleksitas Waktu:

- o $O(n \log n)$ untuk semua kasus karena array selalu dibagi dua dan digabungkan kembali dengan operasi linear.

2. Kompleksitas Ruang:

- o $O(n)$ karena memerlukan array tambahan untuk proses penggabungan.

Perbandingan Kinerja

Algoritma	Waktu Terbaik	Waktu Terburuk	Waktu Rata-rata	Ruang Tambahan
-----------	---------------	----------------	-----------------	----------------

Radix Sort	$O(n * d)$	$O(n * d)$	$O(n * d)$	$O(n + k)$
------------	------------	------------	------------	------------

Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(\log n)$
------------	---------------	----------	---------------	-------------

Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
------------	---------------	---------------	---------------	--------

Kapan Radix Sort Lebih Unggul?

- Jumlah elemen yang besar dengan nilai yang kecil dan digit tetap: Radix Sort sangat cepat

untuk data seperti angka positif kecil, karena tidak tergantung pada perbandingan elemen.

- **Input data yang terdistribusi merata:** Radix Sort bekerja dengan baik jika elemen memiliki distribusi digit yang serupa.

Source code:

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

// Fungsi untuk menampilkan informasi nama, kelas, dan NIM
void displayInfo() {
    cout << "=====\n";
    cout << "Nama : Muhammad Dzakwan F\n";
    cout << "Kelas: 03TPLP029\n";
    cout << "Nim : 231011403237\n";
    cout << "=====\n\n";
}

// Fungsi Radix Sort
void radixSort(vector<int> & arr) {
    int maxVal = *max_element(arr.begin(), arr.end());
    int exp = 1;

    while (maxVal / exp > 0) {
        vector<int> output(arr.size());
        vector<int> count(10, 0);

        // Hitung frekuensi digit
        for (int num : arr)
            count[(num / exp) % 10]++;

        // Hitung posisi elemen
        for (int i = 1; i < 10; i++)
            count[i] += count[i - 1];

        // Tempelkan elemen berdasarkan digit
        for (int i = arr.size() - 1; i >= 0; i--) {
            int digit = (arr[i] / exp) % 10;
            output[count[digit] - 1] = arr[i];
            count[digit]--;
        }

        // Salin hasil ke array asli
        arr = output;
        exp *= 10;
    }
}
```

// Fungsi Partition 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; j++) {  
        if (arr[j] < pivot) {  
            i++;  
            swap(arr[i], arr[j]);  
        }  
    }  
    swap(arr[i + 1], arr[high]);  
    return i + 1;  
}
```

// Fungsi Quick Sort

```
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 left, int mid, int right) {  
    int n1 = mid - left + 1;  
    int n2 = right - mid;  
    vector<int> L(n1), R(n2);  
  
    for (int i = 0; i < n1; i++) L[i] = arr[left + i];  
    for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];  
  
    int i = 0, j = 0, k = left;  
    while (i < n1 && j < n2) {  
        if (L[i] <= R[j]) arr[k++] = L[i++];  
        else arr[k++] = R[j++];  
    }  
    while (i < n1) arr[k++] = L[i++];  
    while (j < n2) arr[k++] = R[j++];  
}
```

// Fungsi Merge Sort

```
void mergeSort(vector<int>& arr, int left, int right) {  
    if (left < right) {  
        int mid = left + (right - left) / 2;  
        mergeSort(arr, left, mid);  
        mergeSort(arr, mid + 1, right);  
        merge(arr, left, mid, right);  
    }  
}
```

```

    }
}

// Main Program
int main() {
    // Menampilkan informasi nama, kelas, dan NIM
    displayInfo();

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

    vector<int> arrRadix = arr;
    vector<int> arrQuick = arr;
    vector<int> arrMerge = arr;

    // Radix Sort
    cout << "Radix Sort: ";
    radixSort(arrRadix);
    for (int num : arrRadix) cout << num << " ";
    cout << endl;

    // Quick Sort
    cout << "Quick Sort: ";
    quickSort(arrQuick, 0, arrQuick.size() - 1);
    for (int num : arrQuick) cout << num << " ";
    cout << endl;

    // Merge Sort
    cout << "Merge Sort: ";
    mergeSort(arrMerge, 0, arrMerge.size() - 1);
    for (int num : arrMerge) cout << num << " ";
    cout << endl;

    return 0;
}

```

Algoritma Radix Sort, Quick Sort, dan Merge Sort

1. Radix Sort:

- Temukan elemen terbesar untuk menentukan jumlah digit maksimum.
- Untuk setiap digit dari LSD (Least Significant Digit) ke MSD (Most Significant Digit):
 - Kelompokkan elemen ke dalam bucket sesuai digit tersebut.
 - Rekonstruksi array dari bucket.
- Ulangi hingga semua digit selesai.

2. Quick Sort:

- Pilih elemen pivot (bisa elemen terakhir atau elemen tengah).
- Partisi array menjadi dua bagian:
 - Elemen lebih kecil dari pivot.
 - Elemen lebih besar dari pivot.
- Rekursi pada kedua bagian tersebut.
- Kombinasikan hasilnya.

3. Merge Sort:

- Bagi array menjadi dua bagian hingga setiap bagian terdiri dari satu elemen.
- Gabungkan dua bagian tersebut secara berurutan menggunakan proses merge.
- Rekursi hingga array tersortir sepenuhnya.

Screenshot source code:

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <cmath>
5 using namespace std;
6
7 // Fungsi untuk menampilkan informasi nama, kelas, dan NIM
8 void displayInfo() {
9     cout << "=====\n";
10    cout << "Nama : Muhammad Dzakwan F\n";
11    cout << "Kelas: 03TLP020\n";
12    cout << "Nim : 231011403237\n";
13    cout << "=====\n\n";
14 }
15
16 // Fungsi Radix Sort
17 void radixSort(vector<int>& arr) {
18     int maxVal = *max_element(arr.begin(), arr.end());
19     int exp = 1;
20
21     while (maxVal / exp > 0) {
22         vector<int> output(arr.size());
23         vector<int> count(10, 0);
24
25         // Hitung frekuensi digit
26         for (int num : arr)
27             count[(num / exp) % 10]++;
28
29         // Hitung posisi elemen
30         for (int i = 1; i < 10; i++)
31             count[i] += count[i - 1];
32
33         // Tempelkan elemen berdasarkan digit
34         for (int i = arr.size() - 1; i >= 0; i--) {
35             int digit = (arr[i] / exp) % 10;
36             output[count[digit] - 1] = arr[i];
37             count[digit]--;
38         }
39
40         // Salin hasil ke array asli
41         arr = output;
42         exp *= 10;
43     }
44 }
45

```

```

Select C:\Users\asus\Documents\semester 3\tugas cpp\algorit...
=====
Nama : Muhammad Dzakwan F
Kelas: 03TLP020
Nim : 231011403237
=====
Radix Sort: 2 24 45 66 75 90 170 802
Quick Sort: 2 24 45 66 75 90 170 802
Merge Sort: 2 24 45 66 75 90 170 802
-----
Process exited after 0.03851 seconds with return value 0
Press any key to continue . . .

```

Screenshot source code:

```

46 // Fungsi Partition untuk Quick Sort
47 int partition(vector<int>& arr, int low, int high) {
48     int pivot = arr[high];
49     int i = low - 1;
50
51     for (int j = low; j < high; j++) {
52         if (arr[j] < pivot) {
53             i++;
54             swap(arr[i], arr[j]);
55         }
56     }
57     swap(arr[i + 1], arr[high]);
58     return i + 1;
59 }
60
61 // Fungsi Quick Sort
62 void quickSort(vector<int>& arr, int low, int high) {
63     if (low < high) {
64         int pi = partition(arr, low, high);
65         quickSort(arr, low, pi - 1);
66         quickSort(arr, pi + 1, high);
67     }
68 }
69
70 // Fungsi Merge untuk Merge Sort
71 void merge(vector<int>& arr, int left, int mid, int right) {
72     int n1 = mid - left + 1;
73     int n2 = right - mid;
74     vector<int> L(n1), R(n2);
75
76     for (int i = 0; i < n1; i++) L[i] = arr[left + i];
77     for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];
78
79     int i = 0, j = 0, k = left;
80     while (i < n1 && j < n2) {
81         if (L[i] <= R[j]) arr[k++] = L[i++];
82         else arr[k++] = R[j++];
83     }
84     while (i < n1) arr[k++] = L[i++];
85     while (j < n2) arr[k++] = R[j++];
86 }
87

```

Screenshot source code:

```

87
88 // Fungsi Merge Sort
89 void mergeSort(vector<int>& arr, int left, int right) {
90     if (left < right) {
91         int mid = left + (right - left) / 2;
92         mergeSort(arr, left, mid);
93         mergeSort(arr, mid + 1, right);
94         merge(arr, left, mid, right);
95     }
96 }
97
98 // Main Program
99 int main() {
100     // Menampilkan informasi nama, kelas, dan NIM
101     displayInfo();
102
103     vector<int> arr = {170, 45, 75, 90, 802, 24, 2, 66};
104
105     vector<int> arrRadix = arr;
106     vector<int> arrQuick = arr;
107     vector<int> arrMerge = arr;
108
109     // Radix Sort
110     cout << "Radix Sort: ";
111     radixSort(arrRadix);
112     for (int num : arrRadix) cout << num << " ";
113     cout << endl;
114
115     // Quick Sort
116     cout << "Quick Sort: ";
117     quickSort(arrQuick, 0, arrQuick.size() - 1);
118     for (int num : arrQuick) cout << num << " ";
119     cout << endl;
120
121     // Merge Sort
122     cout << "Merge Sort: ";
123     mergeSort(arrMerge, 0, arrMerge.size() - 1);
124     for (int num : arrMerge) cout << num << " ";
125     cout << endl;
126
127     return 0;
128 }
129

```

Jawaban soal nomor 5

Tanggal:08-01-2025

Judul: Prinsip Rekursif pada Sierpinski Triangle

Fractal adalah pola geometris kompleks yang dapat dihasilkan melalui prinsip rekursif, yaitu proses di mana suatu fungsi memanggil dirinya sendiri dengan skala atau parameter yang lebih kecil. Fraktal seperti **Sierpinski Triangle** atau **Mandelbrot Set** dibuat menggunakan prinsip ini untuk membentuk pola yang self-similar (mirip pada berbagai skala).

Berikut adalah penjelasan prinsip rekursif dan implementasi program untuk menghasilkan **Sierpinski Triangle** dalam C++:

Prinsip Rekursif pada Sierpinski Triangle

1. **Basis Rekursi:** Pada level tertentu (misalnya, ketika ukuran segitiga sangat kecil), fungsi berhenti memanggil dirinya sendiri.
2. **Rekursi:** Segitiga besar dibagi menjadi tiga segitiga kecil, kemudian proses rekursi diterapkan pada masing-masing segitiga kecil.

Source code:

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

// Fungsi untuk menampilkan informasi
void displayInfo() {
    cout << "===== " << endl;
    cout << "Nama : Muhammad Dzakwan F" << endl;
    cout << "Kelas: 03TPLP029" << endl;
    cout << "Nim : 231011403237" << endl;
    cout << "===== " << endl;
}

// Fungsi rekursif untuk menggambar segitiga
void drawSierpinski(vector<vector<char>>& canvas, int x, int y, int size) {
    if (size == 1) { // Base case: ukuran terkecil
        canvas[y][x] = '*'; // Gambar titik
        return;
    }

    // Bagian tengah
    int half = size / 2;

    // Rekursi untuk 3 bagian segitiga
    drawSierpinski(canvas, x, y, half); // Segitiga atas
    drawSierpinski(canvas, x - half, y + half, half); // Segitiga kiri bawah
    drawSierpinski(canvas, x + half, y + half, half); // Segitiga kanan bawah
}
```

```

int main() {
    // Menampilkan informasi
    displayInfo();

    int size = 32; // Ukuran sisi segitiga (harus pangkat 2)
    vector<vector<char>> canvas(size, vector<char>(size * 2, ' ')); // Kanvas

    // Panggil fungsi untuk menggambar Sierpinski Triangle
    drawSierpinski(canvas, size - 1, 0, size);

    // Tampilkan hasil di konsol
    for (const auto& row : canvas) {
        for (char ch : row) cout << ch;
        cout << endl;
    }

    return 0;
}

```

Algoritma Sierpinski Triangle dengan Rekursi:

1. Inisialisasi:

- Tentukan ukuran segitiga (harus berupa pangkat dua).
- Buat sebuah grid (kanvas) dengan ukuran yang cukup besar untuk menampung gambar **Sierpinski Triangle**.
- Tentukan fungsi untuk menampilkan informasi (nama, kelas, dan NIM).

2. Fungsi **displayInfo**:

- Tampilkan informasi identitas seperti nama, kelas, dan NIM.

3. Fungsi Rekursif **drawSierpinski**:

- Jika ukuran segitiga (**size**) = 1 (base case):
 - Gambar titik di posisi (x, y) di grid.
 - Kembalikan untuk menghentikan rekursi.
- Jika ukuran segitiga lebih besar dari 1:
 - Tentukan ukuran setengah dari segitiga saat ini (**half** = **size** / 2).
 - Panggil fungsi rekursif untuk menggambar tiga bagian segitiga kecil:
 1. Segitiga bagian atas: Panggil **drawSierpinski** dengan posisi (x, y) dan ukuran **half**.
 2. Segitiga bagian kiri bawah: Panggil **drawSierpinski** dengan posisi (x - **half**, y + **half**) dan ukuran **half**.
 3. Segitiga bagian kanan bawah: Panggil **drawSierpinski** dengan posisi (x + **half**, y + **half**) dan ukuran **half**.

4. Menampilkan Gambar:

- Setelah selesai menggambar menggunakan fungsi rekursif, tampilkan grid yang berisi **Sierpinski Triangle** pada terminal.

5. Output:

- Tampilkan hasil gambar segitiga dengan karakter * di terminal.

Screenshot source code:

```
3 #include <vector>
4 using namespace std;
5
6 // Fungsi untuk menampilkan informasi
7 void displayInfo() {
8     cout << "===== " << endl;
9     cout << "Nama : Muhammad Dzakwan F" << endl;
10    cout << "Kelas : 03TPLP029" << endl;
11    cout << "Nim : 231011403237" << endl;
12    cout << "===== " << endl;
13 }
14
15 // Fungsi rekursif untuk menggambar segitiga
16 void dravSierpinski(vector<vector<char>>& canvas, int x, int y, int size) {
17     if (size == 1) { // Base case: ukuran terkecil
18         canvas[y][x] = '*'; // Gambar titik
19         return;
20     }
21
22     // Bagian tengah
23     int half = size / 2;
24
25     // Rekursi untuk 3 bagian segitiga
26     dravSierpinski(canvas, x, y, half); // Segitiga atas
27     dravSierpinski(canvas, x - half, y + half, half); // Segitiga kiri bawah
28     dravSierpinski(canvas, x + half, y + half, half); // Segitiga kanan bawah
29 }
30
31 int main() {
32     // Menampilkan informasi
33     displayInfo();
34
35     int size = 32; // Ukuran sisi segitiga (harus pangkat 2)
36     vector<vector<char>> canvas(size, vector<char>(size * 2, ' ')); // Canvas
37
38     // Panggil fungsi untuk menggambar Sierpinski Triangle
39     dravSierpinski(canvas, size - 1, 0, size);
40
41     // Tampilkan hasil di konsol
42     for (const auto& row : canvas) {
43         for (char ch : row) cout << ch;
44         cout << endl;
45     }
46
47     return 0;
48 }
```

