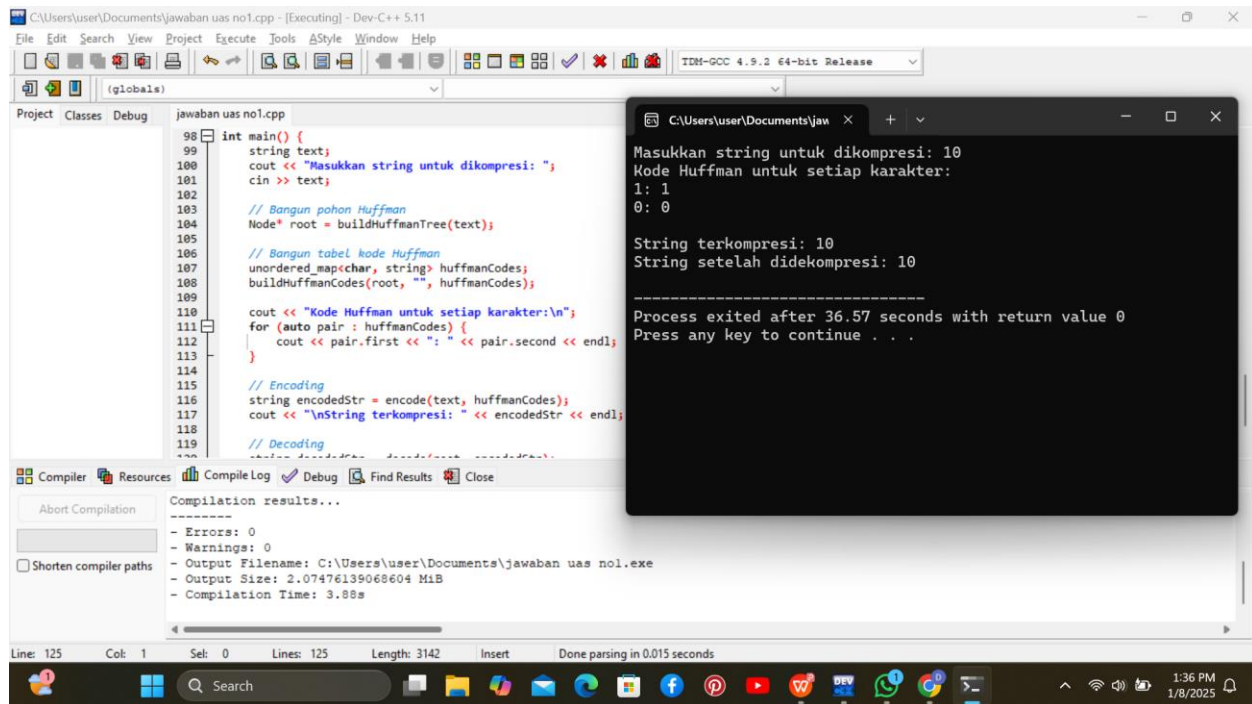


**NAMA: DELVIN HIDAYAH TANJUNG**

**KELAS:03 TPLP 029**

**JAWABAN UAS : ALGORITMA DAN PEMROGRAMAN II**

**1.)**



The screenshot shows a C++ program in Dev-C++ implementing Huffman coding. The program is named 'jawaban uas no1.cpp'. It includes the following code:

```
98 int main() {
99     string text;
100     cout << "Masukkan string untuk dikompresi: ";
101     cin >> text;
102
103     // Bangun pohon Huffman
104     Node* root = buildHuffmanTree(text);
105
106     // Bangun tabel kode Huffman
107     unordered_map<char, string> huffmanCodes;
108     buildHuffmanCodes(root, "", huffmanCodes);
109
110     cout << "Kode Huffman untuk setiap karakter:\n";
111     for (auto pair : huffmanCodes) {
112         cout << pair.first << ": " << pair.second << endl;
113     }
114
115     // Encoding
116     string encodedStr = encode(text, huffmanCodes);
117     cout << "\nString terkompresi: " << encodedStr << endl;
118
119     // Decoding
120     string decodedStr = decode(encodedStr, root);
121     cout << "\nString setelah didekompresi: " << decodedStr << endl;
122 }
```

The program's output is shown in a separate window:

```
Masukkan string untuk dikompresi: 10
Kode Huffman untuk setiap karakter:
1: 1
0: 0

String terkompresi: 10
String setelah didekompresi: 10

-----
Process exited after 36.57 seconds with return value 0
Press any key to continue . . .
```

The compilation results show 0 errors and 0 warnings. The output filename is 'C:\Users\user\Documents\jawaban uas no1.exe', the output size is 2.07476139068604 MiB, and the compilation time is 3.88s.

**Penjelasan:**

**1. Pembangunan Pohon Huffman:**

- Hitung frekuensi setiap karakter dalam string.
- Buat simpul daun untuk setiap karakter dan tambahkan ke dalam priority queue berdasarkan frekuensi.
- Gabungkan dua simpul dengan frekuensi terendah menjadi simpul baru. Frekuensi simpul baru adalah jumlah dari dua simpul yang digabungkan.
- Ulangi hingga hanya tersisa satu simpul dalam queue, yang menjadi akar pohon Huffman.

**2. Proses Encoding:**

- Lakukan traversal pada pohon Huffman (DFS atau rekursif) untuk membuat kode Huffman untuk setiap karakter.
- Gantikan setiap karakter dalam string asli dengan kodenya.

**3. Proses Decoding:**

- Gunakan string hasil encoding dan pohon Huffman untuk mendekodekan kembali ke string asli dengan traversing dari akar ke simpul daun sesuai bit dalam kode.

2.)

```
23 }
24
25 return result;
26 }
27
28 int main() {
29     vector<int> arr1 = {1, 2, 3, 4, 5};
30     vector<int> arr2 = {6, 7, 8, 9, 10};
31     int K = 10;
32
33     // Temukan pasangan
34     vector<pair<int, int>> pairs = findPairs(arr1, arr2, K);
35
36     // Tampilkan hasil
37     cout << "Pasangan dengan jumlah " << K << ":\n";
38     for (const auto& p : pairs) {
39         cout << "(" << p.first << ", " << p.second << ")\n";
40     }
41
42     return 0;
43 }
44
```

```
Pasangan dengan jumlah 10:
(4, 6)
(3, 7)
(2, 8)
(1, 9)

-----
Process exited after 11.98 seconds with return value 0
Press any key to continue . . .
```

Compilation results...

```
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\user\Documents\jawaban uas no2.exe
- Output Size: 1.98207963867188 MiB
- Compilation Time: 3.78s
```

## Penjelasan Program

### 1. Input:

- Dua array `arr1` dan `arr2`.
- Nilai target `KKK`.

### 2. Proses:

- Elemen dari `arr1` dimasukkan ke dalam hash map bersama frekuensinya.
- Iterasi setiap elemen di `arr2`, dan hitung nilai pasangan yang melengkapi `KKK`.
- Jika pasangan ditemukan dalam hash, tambahkan pasangan ke hasil dan perbarui frekuensi elemen di hash.

### 3. Output:

- Tampilkan semua pasangan  $(x, y)$  di mana  $x \in arr1$  dan  $y \in arr2$ , serta  $x + y = K$ .

3.)

The screenshot shows the Dev-C++ IDE with a C++ program for quicksort. The code is in a file named 'jawaban uas no3.cpp'. The program defines a `quickSort` function that uses recursive calls to sort an array. The `main` function initializes an array `data = {3, 6, 8, 10, 1, 2, 1}` and calls `quickSort(data)`. The output of the program is displayed in a separate window, showing the sorted data: `1 1 2 3 6 8 10`. The compilation results at the bottom indicate that the program compiled successfully with 0 errors and 0 warnings. The output file is `C:\Users\user\Documents\jawaban uas no3.exe` and the compilation time was 7.05s.

```
28 sortedLess.push_back(pivot);
29 sortedLess.insert(sortedLess.end(), sortedGreater.begin(), sortedGreater.end());
30
31 return sortedLess;
32
33
34 int main() {
35     vector<int> data = {3, 6, 8, 10, 1, 2, 1};
36
37     // Quick Sort fungsional
38     vector<int> sortedData = quickSort(data);
39
40     // Tampilkan hasil
41     cout << "Data yang sudah diurutkan: ";
42     for (int num : sortedData) {
43         cout << num << " ";
44     }
45     cout << endl;
46
47     return 0;
48
49 }
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\user\Documents\jawaban uas no3.exe
- Output Size: 1.90190505981445 MiB
- Compilation Time: 7.05s

## Penjelasan Kode

### 1. Pivot:

- o Elemen pertama dipilih sebagai pivot.
- o Bagian array selain pivot diproses untuk membentuk dua kelompok: lebih kecil dan lebih besar.

### 2. Rekursi:

- o Fungsi `quickSort` dipanggil untuk subset array yang lebih kecil dan lebih besar.
- o Gabungkan hasil rekursi dengan pivot untuk membentuk array yang sudah diurutkan.

### 3. Penggabungan:

- o Hasil rekursi (dua array yang terurut) digabung menggunakan operasi `push_back` dan `insert`.

4.)

The screenshot shows a C++ IDE with a project named 'jawaban uas no4.cpp'. The code implements three sorting algorithms: Radix Sort, Quick Sort, and Merge Sort. The output window displays the results for an array: 170 45 75 90 802 24 2 66. The output shows the array after each sort, all resulting in the same sorted order: 2 24 45 66 75 90 170 802. The process exited after 17.16 seconds with a return value of 0.

```
// Radix Sort
radixSort(radixArr);
cout << "Setelah Radix Sort: ";
for (int num : radixArr) cout << num << " ";
cout << endl;

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

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

return 0;
```

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\user\Documents\jawaban uas no4.exe
- Output Size: 1.89209079742432 MiB
- Compilation Time: 4.08s

## Analisis Kompleksitas

### Radix Sort

#### 1. Waktu:

- Radix Sort bekerja dengan **digit-by-digit sorting**. Jika  $d$  adalah jumlah digit dalam bilangan terbesar dan  $n$  adalah jumlah elemen:
  - Kompleksitas waktu per iterasi (pengurutan berdasarkan 1 digit menggunakan counting sort):  $O(n)O(n)O(n)$ .
  - Kompleksitas total:  $O(d \cdot n)O(d \cdot n)O(d \cdot n)$ .
- Jika bilangan memiliki  $k$  digit, maka  $d \approx \log_{10}(k) \approx \log_{10}(k)$ , sehingga kompleksitas dapat menjadi  $O(n \cdot \log_{10}(k))O(n \cdot \log_{10}(k))O(n \cdot \log_{10}(k))$ .

#### 2. Ruang:

- Ruang tambahan untuk counting sort:  $O(n+k)O(n+k)O(n+k)$ , di mana  $k$  adalah rentang nilai digit (misalnya, 0–9 untuk bilangan desimal).
- Total:  $O(n+k)O(n+k)O(n+k)$ .

### Quick Sort

#### 1. Waktu:

- Rata-rata:  $O(n \log n)O(n \log n)O(n \log n)$ .
- Kasus terburuk (jika pivot buruk):  $O(n^2)O(n^2)O(n^2)$ .

#### 2. Ruang:

- Rekursi memerlukan  $O(\log n)O(\log n)O(\log n)$  ruang untuk stack (in-place).

## Merge Sort

1. **Waktu:**
  - Selalu  $O(n \log n)$  karena selalu membagi array menjadi dua bagian.
2. **Ruang:**
  - Membutuhkan  $O(n)$  ruang tambahan untuk array sementara.

5.)

The screenshot shows a C++ IDE with the following components:

- Source File:** `jawaban uas no5.cpp` (Executing) - Dev-C++ 5.11
- Code:**

```
15
16 // Rekursi untuk 3 bagian segitiga
17 drawSierpinski(canvas, x, y, half);
18 drawSierpinski(canvas, x - half, y + half);
19 drawSierpinski(canvas, x + half, y + half);
20 }
21
22 int main() {
23     int size = 32; // Ukuran sisi segitiga
24     vector<vector<char>> canvas(size, vector<char>(size, ' '));
25
26     // Panggil fungsi untuk menggambar Sierpinski
27     drawSierpinski(canvas, size - 1, 0, size);
28
29     // Tampilkan hasil di konsol
30     for (const auto& row : canvas) {
31         for (char ch : row) cout << ch;
32         cout << endl;
33     }
34
35 }
36
```
- Output Window:** Displays the Sierpinski triangle pattern and the message: "Process exited after 15.77 seconds with return value 0. Press any key to continue . . ."
- Compiler Output:** Shows compilation results: "Compilation results... - Errors: 0 - Warnings: 0 - Output Filename: C:\Users\user\Documents\jawaban uas no5.exe - Output Size: 1.87460994720459 MiB - Compilation Time: 4.19s"
- Taskbar:** Shows the system clock at 2:22 PM on 1/8/2025.