PRAKTIKUM 4

ALGORITMA DAN STRUKTUR DATA

Dosen Pengampu : Anugrayani Bustamin., ST., MT

Disusun Oleh:

Kelompok 4 (Merge Sort)

Nama : Ady Ulil Amri & Adrian

NIM : D121231080 & D121231047

Kelas : B

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK UNIVERSITAS HASANUDDIN

2024

Pseudocode :

Algoritma Merge Sort

Definisi variable

```
Int main()
    arr_size: integer
    arr [arr_size] : integer
    kalah : integer
    total_poin : float
SUBROUTINE merge(int arr[], int l, int m, int r)
    i, j, k : integer
    n1, n2 : integer
    L[n1], R[n2] : integer
SUBROUTINE mergeSort(int arr[], int l, int r)
    m : integer
SUBROUTINE printArray(int A[], int size)
    i : integer
```

Rincian

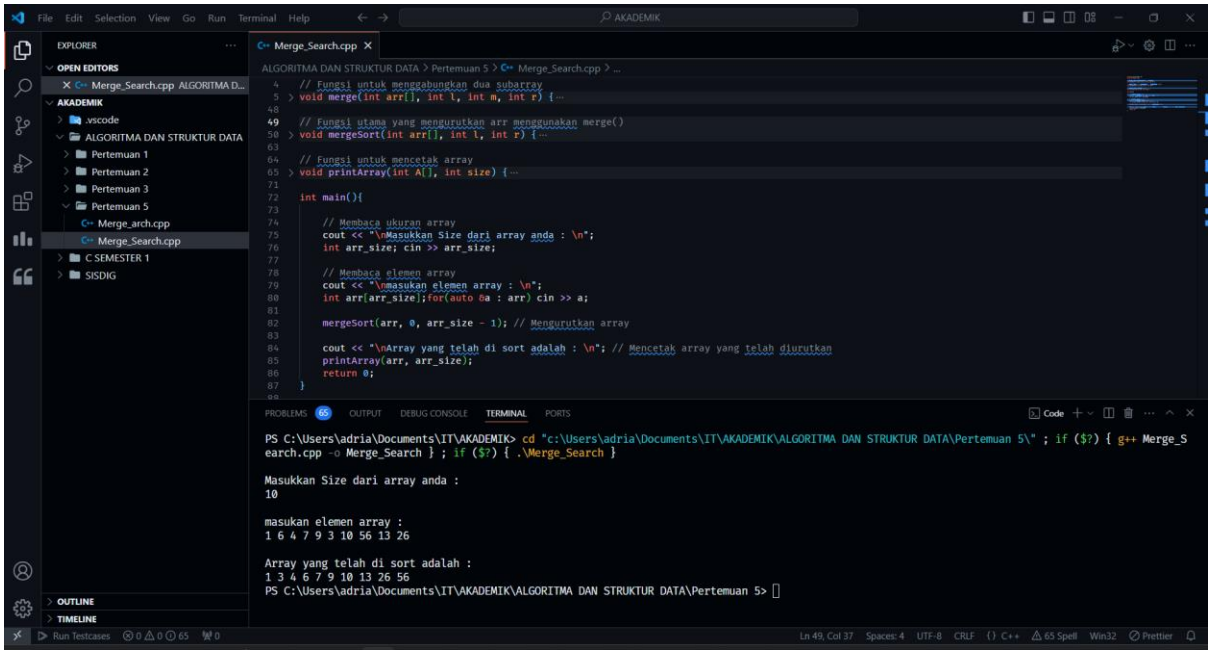
```
START
OUTPUT “Masukkan Size dari array anda : "
READ(arr_size)
OUTPUT “masukan elemen array : “
FOR int i = 0 , i < arr_size
    Read(&arr[i])
    i++
SUBROUTINE mergeSort(arr, 0, arr_size - 1)
    IF l < r
        m = l+(r-l)/2
        SUBROUTINE mergeSort(arr, l, m)
        // rekursif
        SUBROUTINE mergeSort(arr, m+1, r)
        // rekursif
```

```
        SUBROUTINE merge(arr. l, m, r)
            n1 = m - l + 1
            n2 = r – m
            FOR i = 0, I < n1
                L[i] = arr[l + i]
            FOR j = 0. j < n2
                R[j] = arr[m + 1 + j]
            i = 0. j = 0. k = l;
            WHILE i < n1 && j < n2
                IF L[i] <= R[j]
                    arr[k] = L[i];
                    i++;
                ELSE
                    rr[k] = R[j]
                    j++;
                k++
            WHILE i < n1
                arr[k] = L[i]
                i++
                k++
            WHILE j < n2
                arr[k] = R[j]
                j++
                k++
END SUBROUTINE
OUTPUT “Array yang telah di sort adalah : ”
SUBROUTINE printArray(arr, arr_size)
    FOR int i = 0, I < size
        OUTPUT(A[i])
        i++
FINISH
```

Source Code

```cpp
#include <bits/stdc++.h>
using namespace std;

// Fungsi untuk menggabungkan dua subarray
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1; // Ukuran subarray pertama
    int n2 = r - m; // Ukuran subarray kedua

    // Membuat array sementara
    int L[n1], R[n2];

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

    // Menggabungkan array sementara kembali ke array asli
    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++;
    }

    // Menyalin elemen yang tersisa dari L[], jika ada
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Menyalin elemen yang tersisa dari R[], jika ada
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
```

```
}

// Fungsi utama yang mengurutkan arr menggunakan merge()
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Mencari titik tengah untuk membagi array menjadi dua setengah
        int m = l+(r-l)/2;

        // Mengurutkan setengah pertama dan kedua
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        // Menggabungkan dua setengah yang telah diurutkan
        merge(arr, l, m, r);
    }
}

// Fungsi untuk mencetak array
void printArray(int A[], int size) {
    int i;
    for (i=0; i < size; i++)
        cout << A[i] << " ";
    cout << endl;
}

int main(){

    // Membaca ukuran array
    cout << "\nMasukkan Size dari array anda : \n";
    int arr_size; cin >> arr_size;

    // Membaca elemen array
    cout << "\nmasukan elemen array : \n";
    int arr[arr_size];for(auto &a : arr) cin >> a;

    mergeSort(arr, 0, arr_size - 1); // Mengurutkan array

    cout << "\nArray yang telah di sort adalah : \n"; // Mencetak array yang
telah diurutkan
    printArray(arr, arr_size);
    return 0;
}
```



NOTASI

Kompleksitas waktu (time complexity) dari algoritma Merge Sort adalah O(n log n) dalam semua kasus (terbaik, rata-rata, dan terburuk), di mana n adalah jumlah elemen dalam array. Ini karena algoritma ini membagi array menjadi dua bagian hingga hanya tersisa satu elemen, lalu menggabungkannya kembali sambil mengurutkannya. Proses ini melibatkan logaritma basis 2 dari n tahap pembagian dan setiap tahap membutuhkan operasi sebanding dengan n untuk penggabungan.

Kompleksitas ruang (space complexity) dari algoritma Merge Sort adalah O(n), karena membutuhkan ruang tambahan untuk array sementara saat proses penggabungan.

Jadi, notasi Big O untuk algoritma ini adalah O(n log n) untuk waktu dan O(n) untuk ruang.

WORST CASE DAN BEST CASE

Untuk algoritma Merge Sort, kasus terbaik dan terburuknya memiliki kompleksitas waktu yang sama, yaitu O(n log n). Ini berarti bahwa efisiensi algoritma tidak dipengaruhi oleh seberapa terurutnya array input, namun meskipun array input tidak mempengaruhi kompleksitas waktu Merge Sort, beberapa implementasi Merge Sort dapat dioptimalkan untuk menangani kasus di mana bagian dari array input sudah diurutkan, sehingga dapat berjalan lebih cepat pada array semacam itu. Namun, implementasi standar Merge Sort tidak memiliki optimisasi semacam itu.

https://drive.google.com/file/d/1mxZ6C3cv8ySwdG3g0xqiNNe3lVJwU9j0/view?usp=drive_link