TUGAS

PERANCANGAN DAN ANALISIS ALGORITMA

“Mergesort”

Dosen Pengampu :

Randi Proska Sandra, M.Sc.

Disusun Oleh :

Nama : Fattan Naufan Islami

NIM : 23343037

PROGRAM STUDI INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

UNIVERSITAS NEGERI PADANG

2025

A. Penjelasan Program / Algoritma

Mergesort adalah algoritma pengurutan yang menggunakan teknik divide-and-conquer, yaitu membagi masalah besar menjadi submasalah yang lebih kecil, menyelesaikan submasalah tersebut secara rekursif, dan kemudian menggabungkan solusi-solusi kecil untuk memperoleh solusi dari masalah besar. Algoritma ini bekerja dengan cara membagi array yang akan diurutkan menjadi dua bagian yang lebih kecil, mengurutkan kedua bagian tersebut secara rekursif, dan akhirnya menggabungkan dua bagian yang sudah terurut menjadi satu array yang terurut penuh.

Proses Dasar dari Mergesort

1. Pembagian (Divide) : Array dibagi menjadi dua bagian yang kira-kira memiliki ukuran yang sama. Proses ini berlanjut secara rekursif hingga setiap bagian hanya berisi satu elemen. Hal ini sesuai dengan prinsip divide-and-conquer yang dijelaskan dalam dokumen Divide and Conquer, di mana masalah besar dibagi menjadi submasalah yang lebih kecil.

2. Pengurutan (Conquer) : Setiap subarray yang dihasilkan dari pembagian tersebut kemudian diurutkan secara rekursif menggunakan Mergesort itu sendiri. Setiap subarray yang telah terpecah akan diurutkan kembali. Ini adalah aplikasi rekursif dari teknik divide-and-conquer, yang menyelesaikan submasalah dengan cara yang sama.

3. Penggabungan (Combine) : Setelah setiap subarray terurut, langkah selanjutnya adalah menggabungkan dua subarray yang sudah terurut menjadi satu array yang terurut. Penggabungan dilakukan dengan cara membandingkan elemen-elemen dari kedua subarray, dan elemen yang lebih kecil dimasukkan ke dalam array hasil gabungan. Proses ini dilakukan hingga semua elemen digabungkan menjadi satu array yang terurut penuh. Tahap penggabungan ini sangat penting, karena ini adalah cara untuk menggabungkan solusi dari submasalah menjadi solusi untuk masalah utama, sesuai dengan prinsip combine dalam teknik divide-and-conquer.

Penerapan Divide-and-Conquer pada Mergesort

Penerapan prinsip divide-and-conquer dalam Mergesort terlihat jelas dalam proses rekursinya. Jika kita mewakili proses tersebut dengan sebuah rekursi, kita mendapatkan rumus sebagai berikut:

$$T(n)=2T(n/2)+O(n)$$

Di mana :

a. ***T(n/2)*** menggambarkan pembagian array menjadi dua bagian yang lebih kecil, yang kemudian diurutkan secara rekursif.
b. ***O(n)*** menggambarkan langkah penggabungan dua subarray yang sudah terurut menjadi satu array yang terurut.

Berdasarkan rekursi ini, dengan menggunakan Teorema Master, dapat dihitung bahwa kompleksitas waktu Mergesort adalah ***O(n log n)***, yang memberikan kestabilan kinerja yang lebih baik dibandingkan dengan algoritma pengurutan lainnya yang memiliki kompleksitas waktu $O(n^2)$**,** seperti Bubble Sort dan Selection Sort.

Kelebihan Mergesort

1. Algoritma pengurutan yang stabil, yang artinya elemen-elemen yang memiliki nilai yang sama akan tetap berada pada urutan yang sama setelah pengurutan.

2. Kompleksitas waktu yang konsisten ***O(n log n)*** pada kasus terbaik, rata-rata, dan terburuk.

Kekurangan Mergesort

1. Memerlukan ruang tambahan untuk menyimpan array sementara selama proses penggabungan, yang membuatnya lebih boros dalam penggunaan memori dibandingkan dengan algoritma pengurutan lain seperti QuickSort yang lebih efisien dalam penggunaan ruang.

B. Pseudocode

```
BEGIN

STRUCT array:
   DECLARE n as integer
   DECLARE array[n] as integer

FUNCTION gabungkan(array, kiri, tengah, kanan):
   DECLARE n1 as integer = tengah - kiri + 1
   DECLARE n2 as integer = kanan - tengah

   CREATE bagian_kiri[n1], bagian_kanan[n2] as temporary arrays

   // Salin data ke array sementara
   FOR i from 0 to n1 - 1 DO
      bagian_kiri[i] = array[kiri + i]
   END FOR
   FOR j from 0 to n2 - 1 DO
      bagian_kanan[j] = array[tengah + 1 + j]
   END FOR

   DECLARE i = 0, j = 0, k = kiri

   // Gabungkan dua bagian yang sudah terurut
   WHILE i < n1 AND j < n2 DO
      IF bagian_kiri[i] <= bagian_kanan[j] THEN
         array[k] = bagian_kiri[i]
         INCREMENT i by 1
      ELSE
         array[k] = bagian_kanan[j]
         INCREMENT j by 1
      END IF
      INCREMENT k by 1
   END WHILE

   // Salin elemen yang tersisa, jika ada
   WHILE i < n1 DO
      array[k] = bagian_kiri[i]
      INCREMENT i by 1
      INCREMENT k by 1
   END WHILE
```

```
    WHILE j < n2 DO
        array[k] = bagian_kanan[j]
        INCREMENT j by 1
        INCREMENT k by 1
    END WHILE
END FUNCTION

FUNCTION urutkan(array, kiri, kanan):
    IF kiri < kanan THEN
        DECLARE tengah as integer = kiri + (kanan - kiri) / 2

        CALL urutkan(array, kiri, tengah) // Rekursif untuk bagian kiri
        CALL urutkan(array, tengah + 1, kanan) // Rekursif untuk bagian kanan

        CALL gabungkan(array, kiri, tengah, kanan) // Gabungkan kedua bagian
    END IF
END FUNCTION

FUNCTION main():
    DECLARE array[n] as integer = {12, 11, 13, 5, 6, 7}
    DECLARE n as integer = SIZE OF array

    CALL urutkan(array, 0, n - 1)

    DISPLAY "Array yang terurut: "
    FOR i from 0 to n - 1 DO
        DISPLAY array[i]
    END FOR
END FUNCTION

END
```

C. Source Code

```
/*
    Nama File : mergeSort.cpp
    Judul     : Implementasi Algoritma Mergesort dalam Bahasa C++
    Nama      : Fattan Naufan Islami
    NIM       : 23343037
    Prodi     : Informatika
*/

#include <iostream>
#include <vector>

using namespace std;

void gabungkan(vector<int>& array, int kiri, int tengah, int kanan) {
    int n1 = tengah - kiri + 1;
    int n2 = kanan - tengah;

    vector<int> bagian_kiri(n1), bagian_kanan(n2);

    // Salin data ke array sementara
```

```
    for (int i = 0; i < n1; i++)
        bagian_kiri[i] = array[kiri + i];
    for (int j = 0; j < n2; j++)
        bagian_kanan[j] = array[tengah + 1 + j];

    int i = 0, j = 0, k = kiri;

    // Gabungkan dua bagian yang sudah terurut
    while (i < n1 && j < n2) {
        if (bagian_kiri[i] <= bagian_kanan[j]) {
            array[k] = bagian_kiri[i];
            i++;
        } else {
            array[k] = bagian_kanan[j];
            j++;
        }
        k++;
    }

    // Salin elemen yang tersisa, jika ada
    while (i < n1) {
        array[k] = bagian_kiri[i];
        i++;
        k++;
    }

    while (j < n2) {
        array[k] = bagian_kanan[j];
        j++;
        k++;
    }
}

void urutkan(vector<int>& array, int kiri, int kanan) {
    if (kiri < kanan) {
        int tengah = kiri + (kanan - kiri) / 2; // Temukan titik tengah

        urutkan(array, kiri, tengah); // Rekursif untuk bagian kiri
        urutkan(array, tengah + 1, kanan); // Rekursif untuk bagian kanan

        gabungkan(array, kiri, tengah, kanan); // Gabungkan kedua bagian
    }
}

int main() {
    vector<int> array = {12, 11, 13, 5, 6, 7};
    int n = array.size();

    urutkan(array, 0, n - 1);

    cout << "Array yang terurut: ";
    for (int i = 0; i < n; i++)
        cout << array[i] << " ";
    cout << endl;

    return 0;
}
```

D. Analisis Kebutuhan Waktu

1) Analisis menyeluruh dengan memperhatikan operasi/instruksi yang di eksekusi berdasarkan operator penugasan atau assignment (=, +=, *=, dan lainnya) dan operator aritmatika (+, %, * dan lainnya)

a. Menyalin Elemen ke Array Sementara

```
for (int i = 0; i < n1; i++)
bagian_kiri[i] = array[kiri + i];
for (int j = 0; j < n2; j++)
bagian_kanan[j] = array[tengah + 1 + j];
```

Di dalam fungsi ***gabungkan()***, elemen-elemen dari array utama disalin ke dalam dua array sementara yang terpisah (bagian_kiri dan bagian_kanan). Setiap elemen array disalin satu kali, dan karena ada 2 bagian array yang perlu disalin, jumlah operasi penugasan yang dilakukan adalah sebanyak ***2n*** pada setiap level rekursi, di mana ***n*** adalah jumlah elemen yang sedang diproses. Proses penyalinan ini penting untuk memastikan bahwa kedua bagian array yang sedang digabungkan dapat diakses dan dimodifikasi tanpa mengubah array asli.

b. Menyalin Elemen Kembali ke Array Utama

```
array[k] = bagian_kiri[i];
           i++;
           k++;
```

Setelah elemen-elemen dari kedua bagian array dibandingkan, hasil perbandingan tersebut disalin kembali ke dalam array utama agar elemen-elemen berada dalam urutan yang benar. Proses penyalinan ini memerlukan sebanyak ***n*** operasi pada setiap level rekursi, di mana ***n*** adalah jumlah elemen yang digabungkan pada level tersebut. Setiap elemen perlu ditempatkan kembali ke dalam array utama sesuai dengan urutan yang telah ditentukan selama proses penggabungan.

c. Operasi pada Indeks

Pada setiap pemanggilan rekursif fungsi urutkan, array dibagi menjadi dua bagian yang lebih kecil, dan untuk itu diperlukan penugasan pada indeks seperti kiri, tengah, dan kanan untuk menentukan batasan-batasan array yang sedang diproses. Operasi penugasan pada indeks ini dilakukan sebanyak ***O(log n)***, karena ada ***O(log n)*** tingkat rekursi yang terjadi (seiring array dibagi menjadi dua bagian setiap kali).

d. Perhitungan Indeks:

```
int tengah = kiri + (kanan - kiri) / 2;
```

Untuk menemukan titik tengah array, algoritma MergeSort menggunakan rumus ***tengah = kiri + (kanan - kiri) / 2;***. Operasi aritmatika ini melibatkan pengurangan, pembagian, dan penambahan yang memakan waktu konstan ***O(1)*** untuk setiap

pemanggilan fungsi rekursif. Meskipun hanya operasi aritmatika sederhana, perhitungan titik tengah sangat penting untuk membagi array menjadi dua bagian yang lebih kecil pada setiap tingkat rekursi.

e. Increment/Decrement Indeks

```
i++;
j++;
k++;
```

Selama proses penggabungan, indeks untuk iterasi melalui array sementara akan diincrement atau didecrement. Operasi seperti ***i***++, ***j***++, dan ***k***++ ini digunakan untuk memindahkan posisi dalam array sementara atau array utama. Setiap elemen array yang digabungkan memerlukan satu operasi increment atau decrement, yang berarti operasi ini dilakukan sebanyak ***O(n)*** pada setiap level rekursi, di mana ***n*** adalah jumlah elemen yang sedang digabungkan.

Setiap tingkat rekursi dalam algoritma MergeSort ini melibatkan penyalinan elemen ke array sementara, perbandingan elemen, dan penyalinan kembali elemen ke array utama, yang semuanya memerlukan waktu ***O(n)*** per level rekursi. Dengan kedalaman rekursi sebesar ***O(log n)***, jumlah total operasi yang dilakukan oleh algoritma adalah ***O(n log n)***. Hal ini menunjukkan bahwa meskipun ada banyak operasi perbandingan dan penyalinan, jumlah total operasi tumbuh secara logaritmik dengan ukuran input, yang menjadikan MergeSort lebih efisien dibandingkan dengan algoritma pengurutan lainnya yang memiliki kompleksitas waktu lebih tinggi, seperti Bubble Sort atau Insertion Sort.

2) Analisis berdasarkan jumlah operasi abstrak atau operasi khas

a. Perbandingan Elemen

```
if (bagian_kiri[i] <= bagian_kanan[j]) {
       array[k] = bagian_kiri[i];
                  i++;
                } else {
      array[k] = bagian_kanan[j];
                  j++;
                  }
```

Selama proses penggabungan dua array yang sudah terurut, elemen-elemen dari kedua array tersebut dibandingkan untuk menentukan urutan yang benar. Setiap perbandingan antara elemen ***(bagian_kiri[i] <= bagian_kanan[j])*** adalah satu operasi abstrak yang terjadi pada setiap elemen yang sedang digabungkan. Pada setiap tingkat rekursi, sebanyak n elemen akan dibandingkan, sehingga operasi perbandingan ini terjadi sebanyak ***O(n)*** per level rekursi.

b. Penggabungan Subarray

```
        while (i < n1) {
  array[k] = bagian_kiri[i];
             i++;
             k++;
              }
        while (j < n2) {
```

```
array[k] = bagian_kanan[j];
           j++;
           k++;
            }
```

Proses penggabungan dua subarray yang sudah terurut memerlukan iterasi melalui seluruh elemen array dan melakukan perbandingan elemen. Selain perbandingan, elemen yang sudah terurut juga harus dipindahkan ke array utama dalam urutan yang benar, yang juga merupakan operasi abstrak. Proses ini dilakukan pada setiap tingkat rekursi, memerlukan waktu ***O(n)*** pada setiap level, karena setiap elemen harus dipindahkan dan ditempatkan pada posisi yang benar di array utama.

3) Analisis menggunakan pendekatan best-case (kasus terbaik), worst-case (kasus terburuk), dan average-case (kasus rata-rata)

a. Best-Case

Terjadi ketika array sudah terurut. Meskipun array sudah terurut, algoritma tetap akan membagi array menjadi dua bagian dan menggabungkannya kembali. Walaupun tidak ada pertukaran elemen, algoritma tetap melaksanakan pembagian dan penggabungan untuk setiap level rekursi. Oleh karena itu, kompleksitas waktu dalam kasus terbaik tetap ***O(n log n)***, meskipun tidak ada banyak pekerjaan yang dilakukan pada saat penggabungan (hanya melakukan perbandingan).

b. Worst-Case

Worst-case terjadi ketika elemen-elemen array dalam urutan yang paling tidak menguntungkan (misalnya, array terbalik). Dalam hal ini, setiap pembagian rekursif akan dilakukan, dan penggabungan akan memerlukan waktu ***O(n)*** pada setiap level rekursi. Seperti halnya pada best-case, meskipun dalam worst-case kita melakukan lebih banyak perbandingan, kompleksitas waktu MergeSort tetap ***O(n log n)***, karena proses pembagian dan penggabungan tidak terpengaruh oleh urutan awal elemen.

c. Average-Case

Pada kasus rata-rata, elemen-elemen array terdistribusi secara acak. Algoritma tetap membagi array menjadi dua bagian dan menggabungkannya kembali. Proses ini memerlukan ***O(n)*** operasi per level rekursi, dan kedalaman rekursi adalah ***O(log n)***. Kompleksitas waktu untuk average-case juga tetap ***O(n log n)***.

E. Referensi

Levitin, A. (2012). Introduction to the design & analysis of algorithms (3rd ed.). Pearson Education, Inc

F. Lampiran Link Github

https://github.com/Rainy1502/FattanNaufanIslami-mergeSort