LAPORAN
PEMROGRAMAN TERSTRUKTUR
“ANALISIS ALGORITMA PENGURUTAN”

Disusun Oleh:

Wildan Mukmin 2317051080

Dosen Pengajar:

Didik Kurniawan M.kom

Tanggal Pengumpulan:

Link Github: https://github.com/WildanMukmin/AnalisisAlgoritmaSorting

Jurusan Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Lampung

2024

Daftar Isi

Bab 1: Pendahuluan

1.1 Latar Belakang

Pengurutan data merupakan salah satu operasi fundamental dalam ilmu komputer yang sering kali menjadi bagian dari berbagai aplikasi dan sistem. Pengurutan data tidak hanya mempercepat proses pencarian dan pengambilan data, tetapi juga memudahkan dalam analisis dan pengolahan data lebih lanjut. Seiring dengan berkembangnya teknologi informasi, kebutuhan akan algoritma pengurutan yang efisien dan cepat menjadi semakin penting.

Berbagai algoritma pengurutan telah dikembangkan untuk menangani berbagai jenis data dan kebutuhan spesifik. Beberapa di antaranya adalah algoritma pengurutan sederhana seperti Bubble Sort, Insertion Sort, dan Selection Sort, serta algoritma yang lebih kompleks seperti Merge Sort dan Quick Sort. Masing-masing algoritma memiliki karakteristik, kelebihan, dan kelemahan yang berbeda, yang mempengaruhi kinerja mereka dalam berbagai situasi.

1.2 Tujuan

Tujuan dari laporan ini adalah untuk menganalisis dan membandingkan kinerja dari kelima algoritma pengurutan tersebut berdasarkan berbagai faktor, seperti kompleksitas waktu, kompleksitas ruang, dan efisiensi dalam berbagai kondisi data. Analisis ini diharapkan dapat memberikan wawasan yang lebih dalam mengenai kapan dan bagaimana algoritma-algoritma ini dapat digunakan secara efektif dalam berbagai aplikasi praktis.

1.3 Algoritma yang Dianalisis

Pada laporan kali ini algoritma yang dianalisis adalah algortima bubble sort, insertion sort, selection sort, merge sort, dan quick sort.

Bab 2 : Metodologi

2.1 Lingkungan Pengembangan

Pengujian algoritma kali ini menggunakan beberapa tools sebagai berikut ini:

Bahasa Pemrograman : C++, python

Sistem Oprasi : Windows 10

Processor : AMD Ryzen 3 5300U with Radeon Graphics 2.60 GHz

RAM : 8.00GB

System Type : 64-bit operating system, x64-based processor

2.2 Implementasi Algoritma

Bubble Sort: Merupakan salah satu algoritma pengurutan yang paling sederhana. Algoritma ini bekerja dengan cara membandingkan pasangan elemen yang berdekatan dan menukar mereka jika mereka berada dalam urutan yang salah. Proses ini diulang sampai tidak ada lagi pasangan elemen yang perlu ditukar.

Insertion Sort : Algoritma ini bekerja dengan cara membangun array yang diurutkan satu per satu. Elemen dari array input diambil dan dimasukkan ke posisi yang tepat dalam array yang diurutkan.

Selection Sort : Algoritma ini bekerja dengan cara menemukan elemen terkecil dari array yang belum diurutkan dan menukarnya dengan elemen pertama dari array yang belum diurutkan. Proses ini diulang untuk bagian array yang tersisa.

Merge Sort : Algoritma ini menggunakan pendekatan divide and conquer, di mana array dipecah menjadi dua bagian yang lebih kecil, diurutkan secara rekursif, dan kemudian digabungkan kembali dalam urutan yang benar.

Quick Sort : Juga menggunakan pendekatan divide and conquer, algoritma ini memilih elemen pivot dan mempartisi array sedemikian rupa sehingga elemen yang lebih kecil dari pivot berada di sebelah kiri pivot dan elemen yang lebih besar berada di sebelah kanan. Proses ini diulang secara rekursif untuk sub-array di kiri dan kanan pivot.

2.3 Pengujian Kompleksitas

Untuk pengujian pada laporan ini, penulis menggunakan konsep pengujian menggunakan data random, terurut, dan terurut terbalik. Dengan masing data sebanyak 10, 100, 500, 1000, dan 10000. Kemudian data uji yang didapatkan dengan cara mengenerate menggunakan python yang kemudian di simpan kedalam file txt. Setelah file txt yang berisi data n sudah dibuat, kemudian pada program utama c++, saya menggunakan ifsteam untuk membaca file tersebut. Setelah file dibaca maka sebuah variable akan menampung isi dari file tersebut, dimana pada pengujian kali ini penulis menggunakan sebuah stl vector. Vector yang sudah dibuat akan dilakukan pengujian algoritma sorting yang sudah dibuat. Waktu dari pengujian ini didapat dari sebuah library yang bernama *chrono*.

Bab 3 : Hasil Pengujian

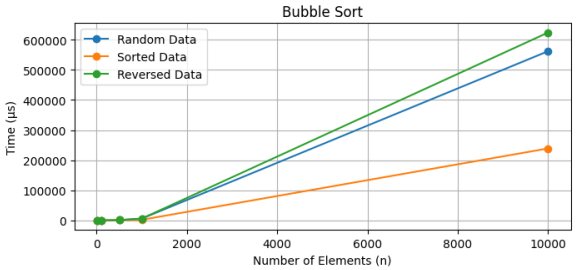
3.1 Setup Pengujian

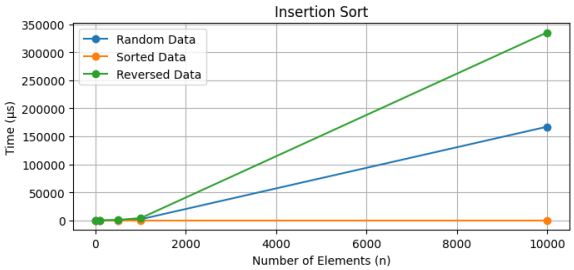
Pengujian dilakukan dengan memisahkan data yang akan diuji dengan file program utama. Seperti yang sudah dibahas pada bab metodelogi, data yang degenerate random, terurut, dan terurut terbalik dibuat menggunakan kode python.

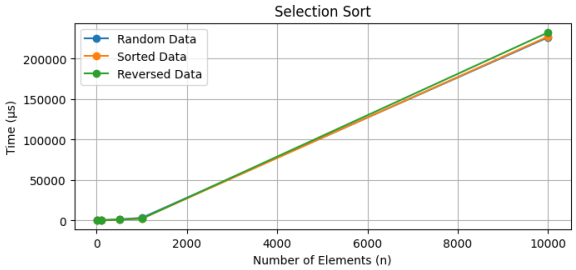
3.2 Tabel Hasil Pengujian

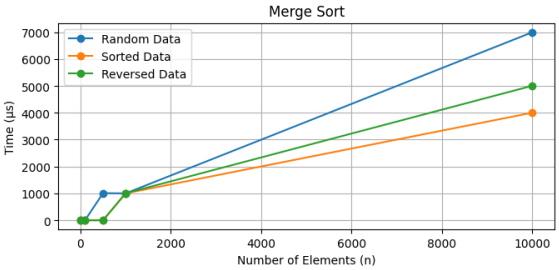
ALGORITMA	KONDISI DATA	N DATA / MICRO SECOND (µs)				
		10	100	500	1000	10000
Bubble Sort	Random Data	0	0	999	6017	562122
	Terurut	0	0	1005	1999	239068
	Terurut Terbalik	0	0	1014	6014	624126
Insertion Sort	Random Data	0	0	0	2000	167023
	Terurut	0	0	0	0	0
	Terurut Terbalik	0	0	1000	3998	335075
Selection Sort	Random Data	0	0	1004	3015	226050
	Terurut	0	0	995	2009	227036
	Terurut Terbalik	0	0	998	1999	232065
Merge Sort	Random Data	0	0	1010	1000	6990
	Terurut	0	0	0	1002	4000
	Terurut Terbalik	0	0	0	1005	5001
Quick Sort	Random Data	0	0	0	0	1001
	Terurut	0	0	1006	5001	501154
	Terurut Terbalik	0	0	1000	3000	307069

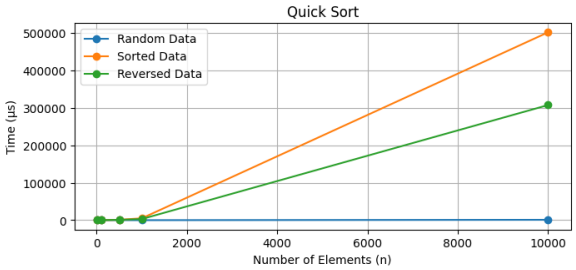
3.3 Grafik Kinerja











Bab 4 : Analisis dan Pembahasan

4.1 Analisis Kompleksitas Waktu

Bubble Sort:

- Random Data: Waktu eksekusi meningkat sangat cepat seiring bertambahnya jumlah elemen, menunjukkan kompleksitas waktu O(n^2).
- Sorted Data: Performa lebih baik dari data acak, tetapi masih O(n^2).
- Reversed Data: Hampir sama dengan data acak, juga menunjukkan O(n^2).

Insertion Sort:

- Random Data: Waktu eksekusi meningkat dengan jumlah elemen, tetapi lebih cepat daripada Bubble Sort. Kompleksitas waktu O(n^2).
- Sorted Data: Sangat cepat dengan waktu hampir nol untuk semua ukuran data, menunjukkan kinerja terbaik dengan O(n) karena hanya satu kali pengulangan.
- Reversed Data: Lebih cepat dari Bubble Sort tetapi masih O(n^2).

Selection Sort:

- Random Data: Waktu eksekusi meningkat dengan jumlah elemen, menunjukkan O(n^2).
- Sorted Data: Performa sedikit lebih baik tetapi masih O(n^2).
- Reversed Data: Serupa dengan data acak, juga O(n^2).

Merge Sort:

- Random Data: Waktu eksekusi meningkat secara linear-logarithmic, menunjukkan O(n log n).
- Sorted Data: Performa tetap konsisten, juga O(n log n).
- Reversed Data: Serupa dengan data acak dan terurut, O(n log n).

Quick Sort:

- Random Data: Sangat cepat dengan O(n log n), terbaik untuk data acak.
- Sorted Data: Performa menurun dengan data terurut, menunjukkan kinerja buruk O(n^2) pada kasus terburuk.
- Reversed Data: Lebih cepat daripada data terurut, tetapi masih bisa mencapai O(n^2).

4.2 Perbandingan Algoritma

Bubble Sort:

- Kelebihan: Implementasi sederhana.
- Kekurangan: Lambat dengan kompleksitas waktu O(n^2), tidak efisien untuk data besar.
- Kondisi Ideal: Ukuran data kecil atau ketika kemudahan implementasi lebih penting daripada performa.

Insertion Sort:

- Kelebihan: Efisien untuk dataset kecil dan hampir terurut, kinerja O(n) pada data terurut.
- Kekurangan: O(n^2) pada data acak atau terbalik.
- Kondisi Ideal: Ukuran data kecil hingga sedang, atau data hampir terurut.

Selection Sort:

- Kelebihan: Sederhana dan tidak tergantung pada urutan data.
- Kekurangan: O(n^2) untuk semua kasus.
- Kondisi Ideal: Ketika stabilitas dan jumlah penulisan minimal lebih penting daripada kecepatan.

Merge Sort:

- Kelebihan: Konsisten O(n log n) untuk semua jenis data, stabil dan efisien pada data besar.
- Kekurangan: Membutuhkan ruang tambahan O(n).
- Kondisi Ideal: Dataset besar, terutama ketika stabilitas penting.

Quick Sort:

- Kelebihan: Sangat cepat dengan O(n log n) pada kasus rata-rata, kinerja terbaik untuk data acak.
- Kekurangan: Kasus terburuk O(n^2) pada data terurut, tidak stabil.
- Kondisi Ideal: Dataset besar dengan data acak, ketika kecepatan lebih penting daripada stabilitas.

Bab 5: Ringkasan Hasil dan Rekomendasi

5.1 Ringkasan Hasil

Bubble Sort: Lambat dan tidak efisien untuk dataset besar dengan kompleksitas waktu O(n^2).

Insertion Sort: Cepat untuk data kecil dan hampir terurut dengan kompleksitas terbaik O(n) pada data terurut, tetapi O(n^2) pada data acak atau terbalik.

Selection Sort: Sederhana, namun lambat untuk semua jenis data dengan O(n^2).

Merge Sort: Konsisten dan efisien dengan O(n log n) pada semua jenis data, membutuhkan ruang tambahan.

Quick Sort: Sangat cepat untuk data acak dengan O(n log n), tetapi bisa mencapai O(n^2) pada data terurut atau hampir terurut.

5.2 Rekomendasi

Untuk Dataset Kecil:

- Insertion Sort: Pilihan terbaik karena cepat pada data kecil dan hampir terurut, kompleksitas terbaik O(n) pada data terurut.

Untuk Dataset Besar:

- Merge Sort: Sangat cocok karena konsisten dengan O(n log n) pada semua jenis data dan stabil.
- Quick Sort: Pilihan terbaik untuk data acak dengan O(n log n), namun perlu berhati-hati dengan data yang terurut atau hampir terurut karena bisa mencapai O(n^2).

Ketika Kemudahan Implementasi Penting:

- Bubble Sort atau Selection Sort: Keduanya sangat sederhana untuk diimplementasikan, namun tidak efisien untuk data besar.

Ketika Stabilitas Penting:

- Merge Sort: Stabil dan efisien dengan O(n log n), ideal untuk dataset besar yang membutuhkan stabilitas.

Ketika Memori Terbatas:

- Quick Sort: Lebih efisien dalam penggunaan memori dibandingkan Merge Sort karena tidak membutuhkan ruang tambahan sebesar O(n).

Lampiran

Untuk beberapa hal penting yang baiknya di perhatikan, karna kode saya menggunakan file external, mungkin jika pembaca ingin menuji kebenaran kode saya, baiknya langsung clone code saya pada link github https://github.com/WildanMukmin/AnalisisAlgoritmaSorting .

Berikut ini beberapa cuplikan kode yang saya buat:

```python
import random

# Function to generate test data and save it to a file
def generate_test_data():
    sizes = [10, 100, 500, 1000, 10000]
    conditions = ['random', 'reversed', 'sorted']

    for size in sizes:
        for condition in conditions:
            if condition == 'random':
                arr = [random.randint(0, 10000) for _ in range(size)]
            elif condition == 'reversed':
                arr = list(range(size, 0, -1))
            elif condition == 'sorted':
                arr = list(range(size))

            filename = f"data_{condition}_{size}.txt"
            with open(filename, 'w') as f:
                for num in arr:
                    f.write(f"{num}\n")
            print(f"Generated {filename}")

# Generate test data
generate_test_data()
```

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

vector <int> generateData(string location){
    vector <int> arr;
    int temp;
    ifstream data;
    data.open("data/" + location + ".txt");
    while(!data.eof()){
        data >> temp;
        arr.push_back(temp);
    }
    return arr;
}

void insertionSort(string Location) {
    vector <int> arr = generateData(location);
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "waktu eksekusi untuk " << location << " adalah : " << duration.count() << endl;
}

int main() {
    insertionSort("data_random_10");
    insertionSort("data_random_100");
    insertionSort("data_random_500");
    insertionSort("data_random_1000");
    insertionSort("data_random_10000");
    cout << endl << endl;
    insertionSort("data_sorted_10");
    insertionSort("data_sorted_100");
    insertionSort("data_sorted_500");
    insertionSort("data_sorted_1000");
    insertionSort("data_sorted_10000");
    cout << endl << endl;
    insertionSort("data_reversed_10");
    insertionSort("data_reversed_100");
    insertionSort("data_reversed_500");
    insertionSort("data_reversed_1000");
    insertionSort("data_reversed_10000");

    return 0;
}
```

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

vector <int> generateData(string location){
    vector <int> arr;
    int temp;
    ifstream data;
    data.open("data/" + location + ".txt");
    while(!data.eof()){
        data >> temp;
        arr.push_back(temp);
    }
    return arr;
}

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];
            ++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 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);
    }
}

void testingTime(string location){
    vector <int> arr = generateData(location);
    auto start = high_resolution_clock::now();
    mergeSort(arr, 0, arr.size() - 1);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "waktu eksekusi untuk mergeSort " << location << " adalah : " << duration.count() << endl;
}


int main() {
    testingTime("data_random_10");
    testingTime("data_random_100");
    testingTime("data_random_500");
    testingTime("data_random_1000");
    testingTime("data_random_10000");
    cout << endl << endl;
    testingTime("data_sorted_10");
    testingTime("data_sorted_100");
    testingTime("data_sorted_500");
    testingTime("data_sorted_1000");
    testingTime("data_sorted_10000");
    cout << endl << endl;
    testingTime("data_reversed_10");
    testingTime("data_reversed_100");
    testingTime("data_reversed_500");
    testingTime("data_reversed_1000");
    testingTime("data_reversed_10000");
    return 0;
}

```

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

vector <int> generateData(string location){
    vector <int> arr;
    int temp;
    ifstream data;
    data.open("data/" + location + ".txt");
    while(!data.eof()){
        data >> temp;
        arr.push_back(temp);
    }
    return arr;
}

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;
}

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);
    }
}

void testingTime(string location){
    vector <int> arr = generateData(location);
    auto start = high_resolution_clock::now();
    quickSort(arr, 0, arr.size() - 1);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "waktu eksekusi untuk quicksort " << location << " adalah : " << duration.count() << endl;
}



int main() {
    testingTime("data_random_10");
    testingTime("data_random_100");
    testingTime("data_random_500");
    testingTime("data_random_1000");
    testingTime("data_random_10000");
    cout << endl << endl;
    testingTime("data_sorted_10");
    testingTime("data_sorted_100");
    testingTime("data_sorted_500");
    testingTime("data_sorted_1000");
    testingTime("data_sorted_10000");
    cout << endl << endl;
    testingTime("data_reversed_10");
    testingTime("data_reversed_100");
    testingTime("data_reversed_500");
    testingTime("data_reversed_1000");
    testingTime("data_reversed_10000");
    return 0;
}
```

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

vector <int> generateData(string location){
    vector <int> arr;
    int temp;
    ifstream data;
    data.open("data/" + location + ".txt");
    while(!data.eof()){
        data >> temp;
        arr.push_back(temp);
    }
    return arr;
}

void selectionSort(string Location) {
    vector <int> arr = generateData(location);
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        swap(arr[i], arr[minIndex]);
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "waktu eksekusi untuk " << location << " adalah : " << duration.count() << endl;
}

int main() {
    selectionSort("data_random_10");
    selectionSort("data_random_100");
    selectionSort("data_random_500");
    selectionSort("data_random_1000");
    selectionSort("data_random_10000");
    cout << endl << endl;
    selectionSort("data_sorted_10");
    selectionSort("data_sorted_100");
    selectionSort("data_sorted_500");
    selectionSort("data_sorted_1000");
    selectionSort("data_sorted_10000");
    cout << endl << endl;
    selectionSort("data_reversed_10");
    selectionSort("data_reversed_100");
    selectionSort("data_reversed_500");
    selectionSort("data_reversed_1000");
    selectionSort("data_reversed_10000");

    return 0;
}

```

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

vector<int> generateData(const string& location) {
    vector<int> arr;
    int temp;
    ifstream data("data/" + location + ".txt");
    while (data >> temp) {
        arr.push_back(temp);
    }
    return arr;
}

void bubbleSort(string location) {
    vector <int> arr = generateData(location);
    auto start = high_resolution_clock::now();
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "waktu eksekusi untuk " << location << " adalah : " << duration.count() << endl;
}

int main() {

    bubbleSort("data_random_10");
    bubbleSort("data_random_100");
    bubbleSort("data_random_500");
    bubbleSort("data_random_1000");
    bubbleSort("data_random_10000");
    cout << endl << endl;
    bubbleSort("data_sorted_10");
    bubbleSort("data_sorted_100");
    bubbleSort("data_sorted_500");
    bubbleSort("data_sorted_1000");
    bubbleSort("data_sorted_10000");
    cout << endl << endl;
    bubbleSort("data_reversed_10");
    bubbleSort("data_reversed_100");
    bubbleSort("data_reversed_500");
    bubbleSort("data_reversed_1000");
    bubbleSort("data_reversed_10000");

    return 0;
}
```