Angga Kresnabayu

140810160001

A

Kodingan:

#include <iostream>

#include <stdlib.h>

#include <ctime>

#include <chrono>

#include <algorithm>

using namespace std;

using namespace std::chrono;

//Algortima Merge Sort

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

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

{

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(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

//Algoritma Selection Sort

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int data[], int n)

{

int i, j, min\_idx;

for (i = 0; i < n-1; i++)

{

min\_idx = i;

for (j = i+1; j < n; j++)

if (data[j] < data[min\_idx])

min\_idx = j;

swap(&data[min\_idx], &data[i]);

}

}

//Algoritma Insertion Sort

void insertionSort(int data[], int n)

{

int i, key, j;

for (i = 1; i < n; i++) {

key = data[i];

j = i - 1;

while (j >= 0 && data[j] > key) {

data[j + 1] = data[j];

j = j - 1;

}

data[j + 1] = key;

}

}

//Algoritma Buble

void bubbleSort(int arr[], int n)

{

int i, j;

for (i = 0; i < n-1; i++)

for (j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

int main()

{

int n = 40000;

int dataMerge[n];

int dataSelec[n];

int dataInsert[n];

int dataBubble[n];

for (int i=0; i<n; i++){

dataMerge[i] = rand()%n;

dataSelec[i] = dataMerge [i];

dataInsert[i] = dataMerge [i];

dataBubble[i] = dataMerge [i];

}

auto start = high\_resolution\_clock::now();

mergeSort(dataMerge, 0, n-1);

auto stop = high\_resolution\_clock::now();

auto start1 = high\_resolution\_clock::now();

selectionSort(dataSelec, n);

auto stop1 = high\_resolution\_clock::now();

auto start2 = high\_resolution\_clock::now();

insertionSort(dataInsert, n);

auto stop2 = high\_resolution\_clock::now();

auto start3 = high\_resolution\_clock::now();

bubbleSort(dataBubble, n);

auto stop3 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto duration1 = duration\_cast<microseconds>(stop1 - start1);

auto duration2 = duration\_cast<microseconds>(stop2 - start2);

auto duration3 = duration\_cast<microseconds>(stop3 - start3);

cout << "Data sebanyak " << n <<endl;

cout << "Time Marge Sort: "<< duration.count() << " microseconds" << endl;

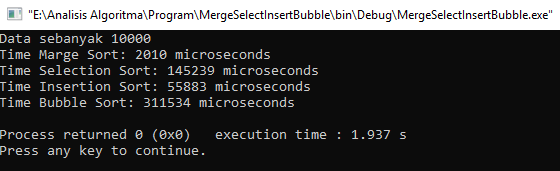
cout << "Time Selection Sort: "<< duration1.count() << " microseconds" << endl;

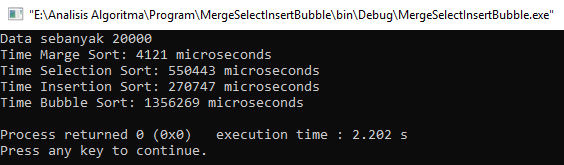
cout << "Time Insertion Sort: "<< duration2.count() << " microseconds" << endl;

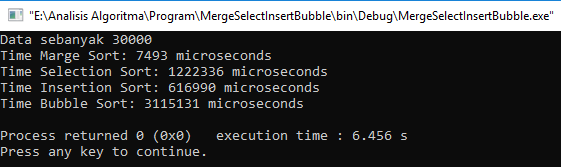
cout << "Time Bubble Sort: "<< duration3.count() << " microseconds" << endl;

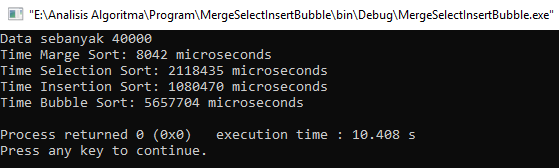
}

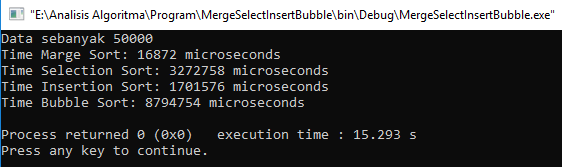
Data yang diuji 10.000, 20.000, 30.000, 40.000, dan 50.000











Hasil:

Kesimpulan:

1. Merge Sort

Untuk best case, worst case, dan average case

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

T(n) = n \* log n

Big-O = Big-Ω = Big-θ = n \* log n

1. Selection Sort

Untuk best case, worst case, dan average case

T(n) = 2(1)+2(2)+ … +2(n-1)

T(n) = 2

T(n) = n2 - n

Big-O = Big-Ω = Big-θ = n2

1. Insertion Sort

Untuk best case

T(n) = an + b

Untuk worst case dan average case

T(n) = an2 + bn + c

Big-O = n

Big-Ω = Big-θ = n2

1. Bubble Sort

Untuk best case

T(n) = n - 1

Untuk worst case dan average case

T(n) = n2 + n

Big-O = n

Big-Ω = Big-θ = n2