Fauzi Faruq Nabbani

140810160007

Tugas 4

1. Merge Sort

Source Code

/\*

Nama : Fauzi Faruq Nabbani

NPM : 140810160007

Program : Merge Sort

\*/

#include <iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

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

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

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

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

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

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

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

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

void membuatArray(int A[] , int size){

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

A[i] = size - i;

}

}

/\* Driver program to test above functions \*/

int main()

{

int arr[10000];

int n = sizeof(arr)/sizeof(arr[0]);

membuatArray(arr,n);

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

mergeSort(arr, 0, n - 1);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

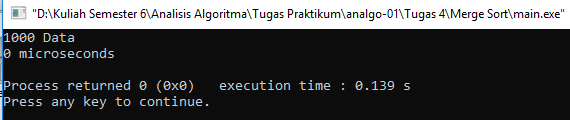
return 0;

}

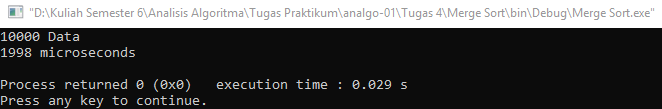
Running Time

Banyak data yang di uji : 1000, 10.000 , 20.000 , 50.000

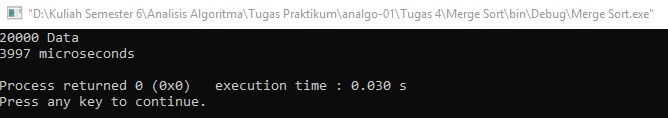
* 1000 Data



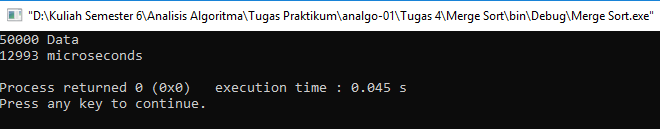
* 10.000 Data



* 20000 Data



* 50000 Data



Kompleksitas Waktu

Big-O = Big-Ω = Big-θ = n \* log n

1. Insertion Sort

Source Code :

/\*

Nama : Fauzi Faruq Nabbani

NPM : 140810160007

Program : Insertion Sort

\*/

#include <iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;

void insertionSort(int arr[], int n)

{

int i, key, j;

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

key = arr[i];

j = i - 1;

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

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

void membuatArray(int A[] , int size){

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

A[i] = size - i;

}

}

int main()

{

int arr[500];

int n = sizeof(arr) / sizeof(arr[0]);

membuatArray(arr,n);

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

insertionSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

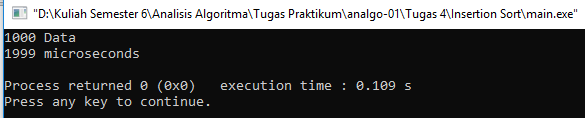
return 0;

}

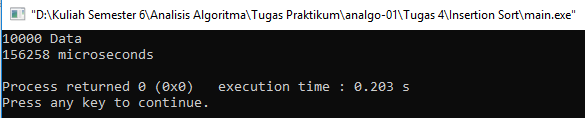
Running Time

Banyak data yang di uji : 10.000 , 20.000 , 50.000 , 100.000

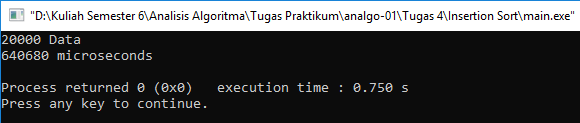
* 1000 Data



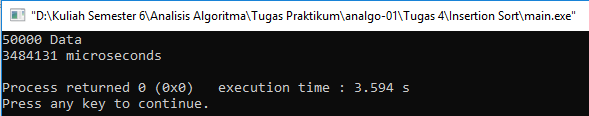
* 10.000 Data



* 20.000 Data



* 50.000 Data



Kompleksitas Waktu

Big-O = n

Big-Ω = Big-θ = n2

1. Selection Sort

Source code :

/\*

Nama : Fauzi Faruq Nabbani

NPM : 140810160007

Program : Selection Sort

\*/

#include <iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;

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

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[],int n)

{

int i , j ,min\_idx;

//One by one move boudary of unsorted subarray

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

{

//Find the minimum element in unsorted array

min\_idx = i;

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

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

min\_idx = j;

// Swap the found minimum element with the first element

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

}

}

void membuatArray(int A[] , int size){

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

A[i] = size - i;

}

}

void printArray(int arr[], int size)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{ cout<<"50000 Data";

int arr[50000] ;

int n = sizeof(arr)/sizeof(arr[0]);

membuatArray(arr,n);

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

selectionSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

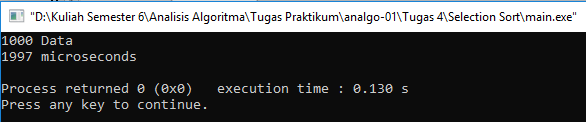
return 0;

}

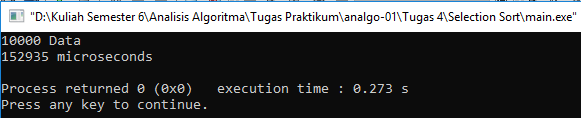
Running Time

Banyak data yang di uji : 1000, 10.000 , 20.000 , 50.000 ,

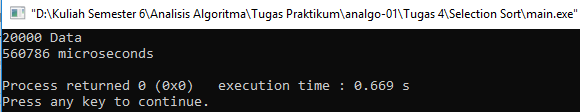
* 1000 Data



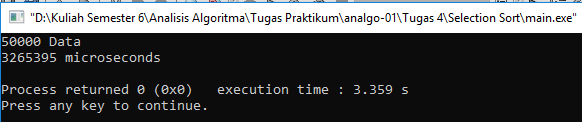
* 10000 Data



* 20000 Data



* 50000 Data



Big-O = Big-Ω = Big-θ = n2

1. Bubble Sort

Source Code

/\*

Nama : Fauzi Faruq Nabbani

NPM : 140810160007

Program : Bubble Sort

\*/

#include <iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;

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

int temp = \*xp;

\*yp = \*yp;

\*yp = temp;

}

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]);

}

}

}

void membuatArray(int A[] , int size){

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

A[i] = size - i;

}

}

int main()

{

cout<<"50000 Data";

int arr[50000];

int n = sizeof(arr)/sizeof(arr[0]);

membuatArray(arr,n);

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

bubbleSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

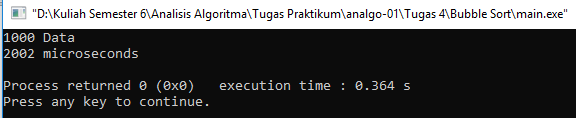
auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

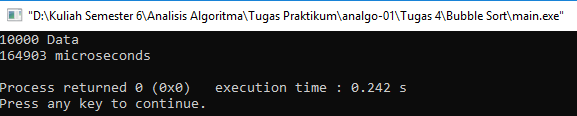
return 0;

}

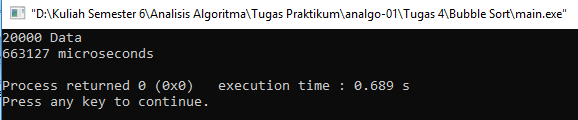
* 1000 Data



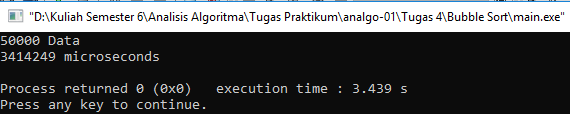
* 10000 Data



* 20000 Data



* 50000 Data



Big-O = n

Big-Ω = Big-θ = n2