**TUGAS MODUL PRAKTIKUM 4**

**ANALISIS ALGORITMA**



Disusun oleh :

Victor Wijaya

140810170057

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

# I. Studi Kasus 1: Merge Sort

## Source code:

#include <iostream>

using namespace std;

#define N 5;

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

int n1 = m - l + 1;

int n2 = r-m;

int L[n1], R[n2];

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

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

for(int i=0; i < n2; i++)

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

int i = 0;

int j = 0;

int 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);

}

}

void cetakArray(int arr[], int n){

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

cout << arr[i] << " ";

cout << endl;

}

int main(){

int arr[] = { 44,76,58,24,34,89,1};

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

cout << "Masukkan Array \t\t: "; cetakArray(arr,n);

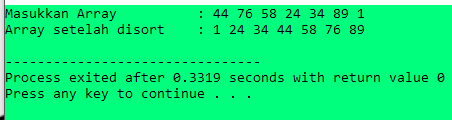
mergeSort(arr, 0, n-1);

cout << "Array setelah disort\t: ";cetakArray(arr,n);

return 0;

}

## Hasil Program:



**Kompleksitas Algoritma:**

Kompleksitas waktu algoritma merge sort adalah O(n lg n). Cari tahu kecepatan komputer Anda dalam memproses program. Hitung berapa running time yang dibutuhkan apabila input untuk merge sort-nya adalah 20?

T(26

# II. Studi Kasus 2: Selection Sort

## Source code:

#include <stdio.h>

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

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], 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 (arr[j] < arr[min\_idx])

min\_idx = j;

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

}

}

void printArray(int arr[], int size){

int i;

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

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

printf("\n");

}

int main(){

int arr[] = {45,76,11,23,41};

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

printf("Masukkan Array \t\t: ");

printArray(arr, n);

selectionSort(arr, n);

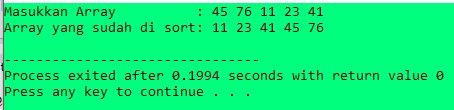
printf("Array yang sudah di sort: ");

printArray(arr, n);

return 0;

}

## Hasil Program:



**Kompleksitas Algoritma:**

Menentukan T(n) :

Oleh karena itu :

Karena

# III. Studi Kasus 3: Insertion Sort

## Source code:

#include <math.h>

#include <stdio.h>

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 printArray(int arr[], int n){

int i;

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

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

printf("\n");}

int main(){

int arr[] = { 21, 32, 18, 65, 3 };

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

printf("Masukkan Array \t\t: ");

printArray(arr, n);

insertionSort(arr, n);

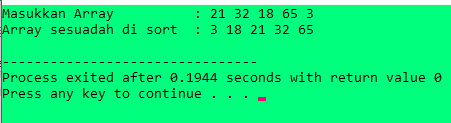
printf("Array sesuadah di sort\t: ");

printArray(arr, n);

return 0;

}

## Hasil Program:



**Kompleksitas Algoritma:**

Menentukan T(n) :

T(n) =

# IV. Studi Kasus 4: Bubble Sort

## Source code:

#include <stdio.h>

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

int temp = \*xp;

\*xp = \*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 printArray(int arr[], int size){

int i;

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

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

printf("\n");

}

int main(){

int arr[] = {43, 47, 8, 10, 99, 3, 25};

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

printf("Masukkan Array \t\t : ");

printArray(arr, n);

bubbleSort(arr, n);

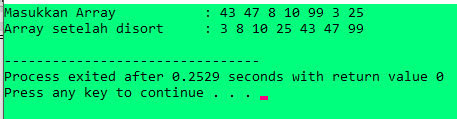
printf("Array setelah disort\t : ");

printArray(arr, n);

return 0;

}

## Hasil Program:



**Kompleksitas Algoritma:**

Menentukan T(n) :