**LAPORAN PRAKTIKUM**

**MODUL 2 – KOMPLEKSITAS WAKTU DARI ALGORITMA**

**PRAKTIKUM MATA KULIAH ANALISIS ALGORITMA**



Disusun oleh:

Nama : Muhammad Fadillah Arsa

NPM : 140810170005

Kelas : A

Program Studi : S1 Teknik Informatika

Fakultas : Matematika dan Ilmu Pengetahuan Alam

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

**LAPORAN PRAKTIKUM**

**MODUL 2 – KOMPLEKSITAS WAKTU DARI ALGORITMA**

**PRAKTIKUM MATA KULIAH ANALISIS ALGORITMA**

# Pendahuluan

Dalam memecahkan suatu masalah dengan komputer seringkali kita dihadapkan pada pilihan berikut:

1. Menggunakan algoritma yang waktu eksekusinya cepat dengan komputer standar
2. Menggunakan algoritma yang waktu eksekusinya tidak terlalu cepat dengan komputer yang cepat

Dikarenakan keterbatasan sumber daya, pola pemecahan masalah beralih ke pertimbangan menggunakan algoritma. Oleh karena itu diperlukan algoritma yang efektif dan efisien atau lebih tepatnya Algoritma yang mangkus.

Algoritma yang mangkus diukur dari berapa **jumlah waktu dan ruang (space) memori** yang dibutuhkan untuk menjalankannya. Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang. Penentuan kemangkusan algoritma adakah dengan melakukan pengukuran kompleksitas algoritma.

Kompleksitas algoritma terdiri dari kompleksitas waktu dan ruang. Terminologi yang diperlukan dalam membahas kompleksitas waktu dan ruang adalah:

1. Ukuran input data untuk suatu algoritma, .

Contoh algoritma pengurutan elemen-elemen larik, adalah jumlah elemen larik. Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks .

1. Kompleksitas waktu, adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input .
2. Kompleksitas ruang, , adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input .

**KOMPLEKSITAS WAKTU**

Kompleksitas waktu sebuah algoritma dapat dihitung dengan langkah-langkah sebagai berikut:

1. Menetapkan ukuran input
2. Menghitung banyaknya operasi yang dilakukan oleh algoritma.

Dalam sebuah algoritma terdapat banyak jenis operasi seperti operasi penjumlahan, pengurangan, perbandingan, pembagian, pembacaan, pemanggilan prosedur, dsb.

**CONTOH**

**Algoritma Menghitung Nilai Rata-rata**

procedure HitungRerata (input x1, x2, …, xn: integer, output r: real)

{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer x1, x2, … xn.

Nilai rata-rata akan disimpan di dalam variable r.

**Input:** x1, x2, … xn

**Output**: r (nilai rata-rata)

}

**Deklarasi**

i : integer

jumlah : real

**Algoritma**

Jumlah 🡨 0

i 🡨 1

while i ≤ n do

jumlah 🡨 jumlah + ai

i 🡨 i + 1

endwhile

{i > n}

r 🡨 jumlah/n {nilai rata-rata}

**Menghitung Kompleksitas Waktu dari Algoritma Menghitung Nilai Rata-rata**

Jenis-jenis operasi yang terdapat di dalam Algoritma HitungRerata adalah:

* Operasi pengisian nilai/*assignment* (dengan operator “🡨”)
* Operasi penjumlahan (dengan operator “+”)
* Operasi pembagian (dengan operator “/”)

Cara menghitung kompleksitas waktu dari algoritma tersebut adalah sengan cra menghitung masing-masing jumlah operasi. Jika operasi tersebut berada di sebuah loop, maka jumlah operasinya bergantung berapa kali loop tersebut diulangi.

1. Operasi pengisian nilai (*assignment*)

jumlah 🡨 0, 1 kali

k 🡨 1, 1 kali

jumlah 🡨jumlah + ak n kali

k 🡨 k+1, n kali

r 🡨 jumlah/n, 1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

1. Operasi penjumlahan

Jumlah + ak, n kali

k+1, n kali

Jumlah seluruh operasi penjumlahan adalah

1. Operasi pembagian

Jumlah seluruh operasi pembagian adalah

Jumlah/n 1 kali

Dengan demikian, kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi aritmatika dan operasi pengisian nilai adalah:

# Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

**Algoritma Pencarian Nilai Maksimal**

procedure CariMaks(input x1, x2, …, xn: integer, output maks: integer)

{ Mencari elemen terbesar dari sekumpulan elemen larik integer x1, x2, …, xn. Elemen terbesar akan disimpan di dalam maks

Input: x1, x2, …, xn

Output: maks (nilai terbesar)

}

**Deklarasi**

i : integer

**Algoritma**

maks 🡨 x1

i 🡨 2

while i ≤ n do

if xi > maks then

maks 🡨 xi

endif

i 🡨 i + 1

endwhile

{i > n}

Jawaban Studi Kasus 1

PROGRAM:

#include <iostream>

using namespace std;

main(){

//- - - - - - - - - DELKARASI

int n, maks, i, x[99];

//- - - - - - - - - INPUT

cout<<"------------------------------------------"<<endl;

cout<<"- - - Menghitung Nilai Data Terbesar - - -"<<endl;

cout<<"------------------------------------------"<<endl;

for(;;){

cout<<"1. Masukkan jumlah data: ";

cin>>n;

if(n<2){

cout<<"Maaf, data minimal 2."<<endl;

continue;

}

break;

}

cout<<"2. Masukkan data:"<<endl;

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

cout<<"ke-"<<i+1<<": ";

cin>>x[i];

}

//- - - - - - - - - ALGORITMA UTAMA

i=1;

maks=x[0];

do {

if(x[i]>maks){

maks=x[i];

}

i=i+1;

}

while(i<n);

cout<<"Nilai terbesar dari data input adalah: "<<maks<<endl;

}

ANALISIS ALGORITMA:

* **MENURUT BUKU MATEMATIKA DISKRIT RINALDI MUNIR:**

Operasi perbandingan elemen larik yang dimaksudkan di dalam algoritma adalah A[i] >maks. Operasi ini terdapat di dalam looping for. Jumlah operasi perbandingan elemen ditentukam oleh berapa kali looping for dieksekusi, yaitu n-1 kali. Dengan demikian, kompleksitas waktu algoritma ini adalah T(n)=n-1.

* **MENURUT HITUNGAN BIASA:**

maks 🡨 x1 1

i 🡨 2 1

while i ≤ n do //proses perbandingan tidak saya hitung

if xi > maks then //proses perbandingan tidak saya hitung

maks 🡨 xi n

endif

i 🡨 i + 1 n

endwhile

1. Best Case: 1+1+n+n = 2+ 2n

**PEMBAGIAN KOMPLEKSITAS WAKTU**

Hal lain yang harus diperhatikan dalam menghitung kompleksitas waktu suatu algoritma adalah parameter yang mencirikan ukuran input. Contoh pada algoritma pencarian, waktu yang dibutuhkan untuk melakukan pencarian tidak hanya bergantung pada ukuran larik () saja, tetapi juga bergantung pada nilai elemen () yang dicari.

Misalkan:

* Terdapat sebuah larik dengan panjang elemen 130 dimulai dari
* Asumsikan elemen-elemen larik sudah terurut. Jika , maka waktu pencariannya lebih cepat 130 kali dari pada atau tidak ada di dalam larik.
* Demikian pula, jika , maka waktu pencariannya ½ kali lebih cepat daripada

Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

1. : kompleksitas waktu untuk kasus terbaik (***best case***)

merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari .

1. : kompleksitas waktu untuk kasus rata-rata (***average case***)

merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari . Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus *searching* diandaikan data yang dicari mempunyai peluang yang sama untuk tertarik dari larik.

1. : kompleksitas waktu untuk kasus terburuk (***worst case***)

merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari .

# Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian beruntun (*sequential search*). Algoritma *sequential search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

procedure SequentialSearch(input : integer, y : integer, output idx : integer)

{ Mencari di dalam elemen . Lokasi (indeks elemen) tempat ditemukan diisi ke dalam idx. Jika tidak ditemukan, makai idx diisi dengan 0.

Input:

Output: idx

}

**Deklarasi**

i : integer

found : boolean {bernilai true jika y ditemukan atau false jika y tidak ditemukan}

**Algoritma**

i 🡨 1

found 🡨 false

while (i ≤ n) and (not found) do

if xi = y then

found 🡨 true

else

i 🡨 i + 1

endif

endwhile

{*i < n or found*}

If found then {*y ditemukan*}

idx 🡨 i

else

idx 🡨 0 {y tidak ditemukan}

endif

Jawaban Studi Kasus 2

PROGRAM:

#include<iostream>

using namespace std;

void PencarianBeruntun(int x[], int tanya, int n){

bool ketemu=false;

int i=0;

while(i<n && !ketemu){

if(x[i] == tanya){

ketemu=true;

}else{

i++;

}

}

int lokasi;

if(ketemu){

lokasi = i+1;

} else{

lokasi = 0;

}

cout<<"Elemen yang dicari berada di indeks: "<< lokasi;

}

main(){

int n, tanya;

cout<<"Masukkan banyak elemen array: ";

cin>>n;

int x[n];

cout<<"Masukan elemen array berupa angka:"<<endl;

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

cout<<"elemen["<<i+1<<"] = ";

cin>>x[i];

}

cout<<endl<<"Masukan elemen array yang ingin dicari: ";

cin>>tanya;

PencarianBeruntun(x, tanya, n);

}

ANALISIS ALGORITMA:

* **BERDASARKAN BUKU MATEMATIKA DISKRIT RINALDI MUNIR:**

Algoritma ini membandingkan setiap elemen larik dengan x, mulai dari elemen pertama sampai x ditemukan atau sampai elemen terakhir. Jika x ditemukan, maka proses pencarian dihentikan. Kita akan menghitung jumlah operasi perbandingan elemen larik yang terjadi selama pencarian (ai=x). Operasi perbandingan yang lain, seperti i<=n tidak akan dihitung. Operasi perbandungan elemen-elemen larik adalah operasi abstrak yang mendasari algoritma pencarian.

1. Best Case, apabila ai=x

Operasi perbandungan elemen (ai=x) hanya dilakukan satu kali maka

Tmin(n)=1

1. Worst Case, apabila an=x atau x tidak ditemukan.

Seluruh elemen larik dibandingkan, maka jumlah perbandingan elemen larik (ai=x) adalah

Tmax(n)=n

1. Average Case, jika x ditemuka pada posisi ke-j, maka operasi perbandungan (ai=x) dilakukan sebanyak j kali. Jadi, kebutuhan waktu rata-rata algoritma ini adalah:

Tavg(n)= (1+2+3+…+n)/n = ½ n(1+n) / n = (n+1)/2

# Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian bagi dua (*binary search*). Algoritma *binary search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

procedure BinarySearch(input : integer, x : integer, output : idx : integer)

{ Mencari y di dalam elemen . Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx. Jika y tidak ditemukan makai dx diisi dengan 0.

**Input:**

**Output: idx**

}

**Deklarasi**

i, j, mid : integer

found : Boolean

**Algoritma**

i 🡨 1

j 🡨 n

found 🡨 false

while (not found) and ( i ≤ j) do

mid 🡨 (i + j) div 2

if xmid = y then

found 🡨 true

else

if xmid < y then {*mencari di bagian kanan*}

i 🡨 mid + 1

else {*mencari di bagian kiri*}

j 🡨 mid – 1

endif

endif

endwhile

{*found or i > j* }

If found then

Idx 🡨 mid

else

Idx 🡨 0

Endif

Jawaban Studi Kasus 3

PROGRAM:

#include<iostream>

using namespace std;

void PencarianBiner(int x[],int tanya,int n){

int i=0, j=n-1, tengah;

bool ketemu=false;

while((!ketemu) && i <= j){

tengah = (i+j)/2;

if(x[tengah]==tanya){

ketemu=true;

}

else{

if(x[tengah]<tanya){

i=tengah+1;

}else{

j=tengah-1;

}

}

}

int lokasi;

if(ketemu){

lokasi = i+1;

} else{

lokasi = 0;

}

cout<<"Elemen yang dicari berada di indeks: "<< lokasi;

}

main(){

int n, tanya;

cout<<"Masukkan banyak elemen array: ";

cin>>n;

int x[n];

cout<<"Masukan elemen array berupa angka:"<<endl;

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

cout<<"elemen["<<i+1<<"] = ";

cin>>x[i];

}

cout<<endl<<"Masukan elemen array yang ingin dicari: ";

cin>>tanya;

PencarianBiner(x, tanya, n);

}

ANALISIS ALGORITMA:

* **BERDASARKAN BUKU MATEMATIKA DISKRIT RINALDI MUNIR:**

Algoritma ini membagi larik di pertengahan menjadi dua bagian yang berukuran sama (n/2) bagian, bagian kiri dan baguan kanan. Jika elemen pertengahan tidak sama dengan x, keputusan dibuat untuk melakukan pencarian pada bagian kiri atau bagian kanan. Proses bagi-dua dilakukan lagi pada bagian yang dipilih. Perhatikanlah bahwa setiap kali memasuki looping while-do maka ukuran larik yang ditelusuri berkurang menjadi setengah kali ukuran semua: n, n/2, n/4, …

Kita akan menghitung jumlah operasi perbandingan elemen dengan x yang terjadi selama pencarian (amid = x). Operasi perbandingan yang lain, seperti i<=j dan amid < x tidak akan dihitung. Untuk penyederhanaan, asumsikan ukuran larik adalah perangkatan dari 2 yaitu n=2k.

1. Best Case, apabila x ditemukan pada elemen pertengahan amid, dan operasi perbandingan elemen (amid=x) yang dilakukan hanya satu kali. Pada kasus ini

Tmid(n)=1

1. Worst Case, apabila elemen x ditemukan ketika ukuran larik =1. Pada kasusu ini, ukuran larik setiap kali memasuki looping while-do adalah n, n/2, n/4, n/8, …, 1 (sebanyak 2log n kali)

Jumlah operasi perbandungan elemen (amid = x) adalah:

Tmax(n)=2log n

1. Average Case, sulit ditentukan

# Studi Kasus 4: Insertion Sort

1. Buatlah program insertion sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

procedure InsertionSort(input/output : integer)

{ Mengurutkan elemen-elemen dengan metode insertion sort.

Input:

OutputL (sudah terurut menaik)

}

**Deklarasi**

i, j, insert : integer

**Algoritma**

for i 🡨 2 to n do

insert 🡨 xi

j 🡨 i

while (j < i) and (x[j-i] > insert) do

x[j]🡨 x[j-1]

j🡨j-1

endwhile

x[j] = insert

endfor

Jawaban Studi Kasus 4

PROGRAM:

#include <iostream>

#include <conio.h>

using namespace std;

int data[10],data2[10];

int n;

void tukar(int a, int b)

{

int t;

t = data[b];

data[b] = data[a];

data[a] = t;

}

void insertion\_sort()

{

int temp,i,j;

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

temp = data[i];

j = i -1;

while(data[j]>temp && j>=0)

{

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

j--;

}

data[j+1] = temp;

}

}

main()

{

cout<<"===PROGRAM INSERTION SORT==="<<endl;

cout<<"Masukkan Jumlah Data : ";

cin>>n;

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

{

cout<<"Masukkan data ke "<<i<<" : ";

cin>>data[i];

data2[i]=data[i];

}

insertion\_sort();

cout<<"Data Setelah di Sort : ";

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

{

cout<<" "<<data[i];

}

cout<<"\n\nSorting dengan insertion sort selesai";

getch();

}

# Studi Kasus 5: Selection Sort

1. Buatlah program selection sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

procedure SelectionSort(input/output : integer)

{ Mengurutkan elemen-elemen dengan metode selection sort.

Input:

OutputL (sudah terurut menaik)

}

**Deklarasi**

i, j, imaks, temp : integer

**Algoritma**

for i 🡨 n downto 2 do {*pass sebanyak n-1 kali*}

imaks 🡨 1

for j 🡨 2 to i do

if xj > ximaks then

imaks 🡨 j

endif

endfor

{pertukarkan ximaks dengan xi}

temp 🡨 xi

xi 🡨 ximaks

ximaks 🡨 temp

endfor

Jawaban Studi Kasus 5

PROGRAM:

/\*

Nama Program : Selection Sort

Nama : Muhammad Fadillah Arsa

NPM : 140810170005

Tanggal Pembuatan : 12 Maret 2019

Telah diujicoba menggunakan Dev C++

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include <iostream>

#include <conio.h>

using namespace std;

int data[10],data2[10];

int n;

void tukar(int a, int b)

{

int t;

t = data[b];

data[b] = data[a];

data[a] = t;

}

void selection\_sort()

{

int pos,i,j;

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

{

pos = i;

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

{

if(data[j] < data[pos]) pos = j;

}

if(pos != i) tukar(pos,i);

}

}

main()

{

cout<<"===PROGRAM SELECTION SORT==="<<endl;

cout<<"Masukkan Jumlah Data : ";

cin>>n;

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

{

cout<<"Masukkan data ke "<<i<<" : ";

cin>>data[i];

data2[i]=data[i];

}

selection\_sort();

cout<<"Data Setelah di Sort : ";

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

{

cout<<" "<<data[i];

}

cout<<"\n\nSorting dengan selection sort selesai";

getch();

}

ANALISIS ALGORITMA:

* **BERDASARKAN BUKU MATEMATIKA DISKRIT RINALDI MUNIR:**

Algoritma ini terdiri dari n-1 kali pass. Pada setiap pass, kita mencari elemen terbesar dari elemen-elemen a1, a2, … , an, lalu menukarkan elemen terbesar dengan an. Pass berikutnya akan mencari elemen terbesar dari sekumpulan a1, a2,… an-1, begitu seterusnya sampai larik pass terakhir sehingga tinggal satu elemen yang pasti sudah terurut. Operasi abstrak yang mendasari algoritma pengurutan adalah operasi perbandingan elemen larik (aj>aimaks) dan operasi pertukaran. Kdua operasi ini dipisahkan perhitungannya sebagai berikut:

1. Jumlah operasi perbandingan elemen

i=n -> jumlah operasi perbandingan elemen n-1

i=n-1 -> jumlah operasi perbandingan elemen n-2

i=n-2 -> jumlah operasi perbandingan elemen n-3

.

.

.

i=2 -> jumlah operasi perbandingan elemen = 1

Jumlah seluruh operasi perbandingan elemen

T(n) = n(n-1)/2

1. Jumlah operasi pertukaran

Untuk setiap i dari n sampai 2, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya adalah

T(n)=n-1

**Jadi, pengurutan seleksi membutuhkan n(n-1)2 operasi perbandungan elemen ditambah n-1 buah operasi pertukaran.**