**TUGAS MODUL PRAKTIKUM 2**

**KOMPLEKSITAS WAKTU DARI ALGORITMA**

**Untuk Memenuhi Tugas**

**Mata Kuliah Praktikum Analisis Algoritma**

**Disusun oleh :**

**Felia Sri Indriyani 140810170018**

**TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

**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 .

2. Kompleksitas waktu, (), adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input .

3. Kompleksitas ruang, (), adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input .

KOMPLEKSITAS WAKTU

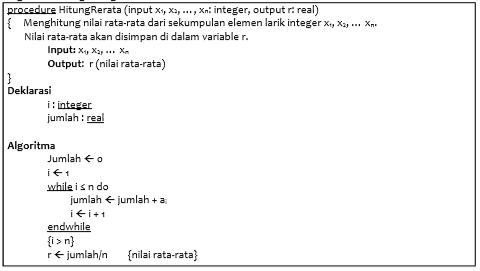
Kompleksitas waktu sebuah algoritma dapat dihitung dengan langkah-langkah sebagai berikut:

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**



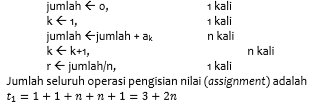
**Menghitung Kompleksitas Waktu dari Algoritma Menghitung Nilai Rata-rata**

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.

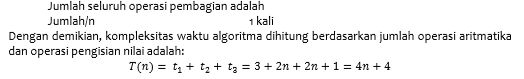
1. Operasi pengisian nilai (assignment)



1. Operasi penjumlahan



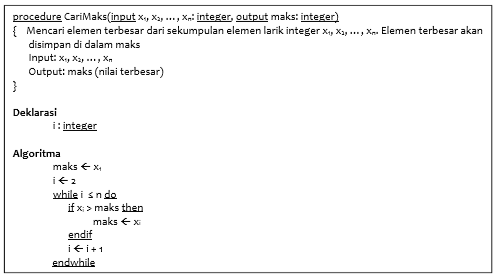
1. Operasi pembagian



**Studi Kasus 1: Pencarian Nilai Maksimal**

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

**Algoritma Pencarian Nilai Maksimal**



Jawaban Studi Kasus 1

T(n) = 2(n – 2) + (n – 2) + 2

= 3 n - 4

**Source code :**

**/\*\*\*\***

**Nama Program : Mencari nilai maksimal**

**Nama : Felia Sri Indriyani**

**NPM : 140810170018**

**\*\*\*\*/**

**#include<iostream>**

**using namespace std;**

**main()**

**{**

**int nilai[100], n,max,min;**

**cout <<"=====================================";**

**cout<<"\nMasukkan Banyak Inputan : "; cin>>n;**

**cout<<"\n";**

**max=-1000;**

**min=1000;**

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

**{**

**cout<<" Nilai Ke- "<<i+1<<" : ";**

**cin>>nilai[i];**

**}**

**cout << "\n-------------------------------------" << endl;**

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

**{**

**if(nilai[i]>max)**

**max=nilai[i];**

**}**

**cout<<" Nilai Maksimum nya = "<<max;**

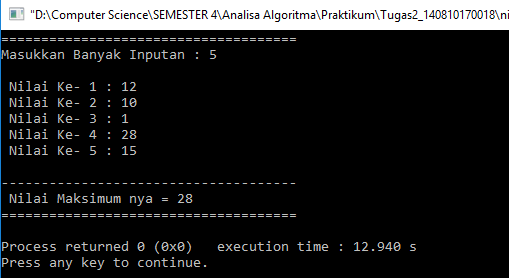
**cout << "\n=====================================";**

**cout<<"\n";**

**return 0;**

**}**

**Screenshoot :**



**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:

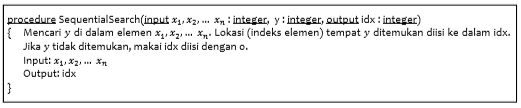
(1) () : kompleksitas waktu untuk kasus terbaik (best case) merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari .

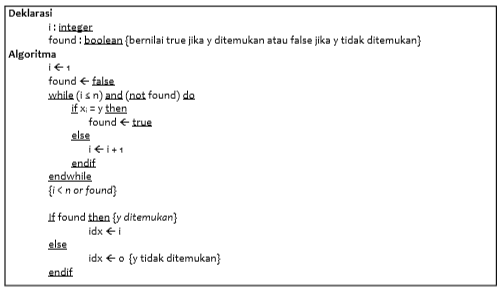
(2) () : kompleksitas waktu untuk kasus rata-rata (average case) merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari . Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus searching diandaikan data yang dicari mempunyai peluang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

(3) () : 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 ratarata 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.





Jawaban Studi Kasus 2

1. Kasus terbaik: ini terjadi bila a1 = x.

Tmin(n) = 1

1. Kasus terburuk: bila an = x atau x tidak ditemukan.

Tmax(n) = n

1. *Kasus rata-rata*: Jika *x* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = x) akan dieksekusi sebanyak *j*kali.

*T*avg(*n*) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2

**Source code :**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Nama Program : Sequential Search**

**Nama : Felia Sri Indriyani**

**NPM : 140810170018**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <iostream>**

**using namespace std;**

**main()**

**{**

**int n, cari, A[100], index, jwb;**

**bool ketemu = false;**

**cout << "\n=====================================";**

**cout << "\nMasukan banyak data = "; cin >> n;**

**cout << "\n-------------------------------------" << endl;**

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

**{**

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

**cin >> A[i];**

**}**

**cout << "\nMasukan data yang akan dicari : "; cin >> cari;**

**cout << "\n-------------------------------------" << endl;**

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

**if(A[i] == cari){**

**ketemu = true;**

**index = i;**

**i = n;**

**}**

**}**

**if(ketemu == true){**

**cout << "Data telah ditemukan pada data ke-" << index+1;**

**}**

**else{**

**cout << "Maaf. Data tidak ditemukan!";**

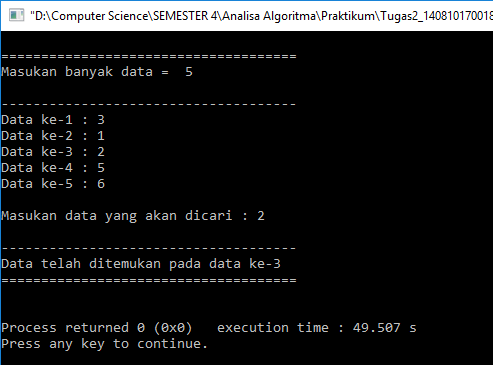
**}**

**cout << "\n=====================================" << endl;**

**cout << "\n";**

**}**

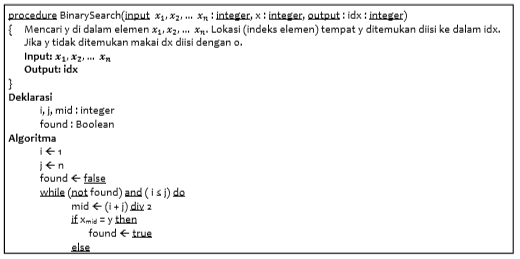
**Screenshoot :**

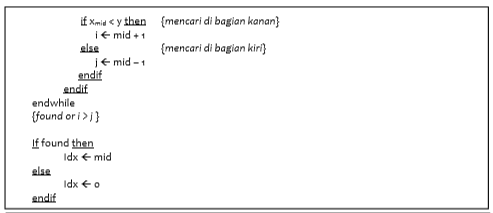


**Studi Kasus 3:**

**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 ratarata 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.





Jawaban Studi Kasus 3

1. Kasus terbaik

Tmin(n) = 1

1. Kasus terburuk

Tmax (n) = 2log n

**Source code :**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Nama Program : Binary Search**

**Nama : Felia Sri Indriyani**

**NPM : 140810170018**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**int main()**

**{**

**int n, i, arr[100], cari, awal, akhir, tengah;**

**cout << "\n=================================="<<endl;**

**cout<<"Masukkan banyak data : ";cin>>n;**

**cout << "\n----------------------------------" << endl;**

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

**{**

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

**cin>>arr[i];**

**}**

**cout<<"\nMasukkan data yang akan di cari :"; cin>>cari;**

**awal = 0;**

**akhir = n-1;**

**cout << "\n----------------------------------" << endl;**

**while (awal <= akhir)**

**{**

**tengah = (awal+akhir)/2;**

**if(arr[tengah] < cari)**

**{**

**awal = tengah + 1;**

**}**

**else if(arr[tengah] == cari)**

**{**

**cout<<cari<<" ditemukan pada data ke-"<<tengah+1<<"\n";**

**break;**

**}**

**else**

**{**

**akhir = tengah - 1;**

**}**

**tengah = (awal + akhir)/2;**

**}**

**if(awal > akhir)**

**{**

**cout<<cari<<"Tidak Ditemukan! Karena ";**

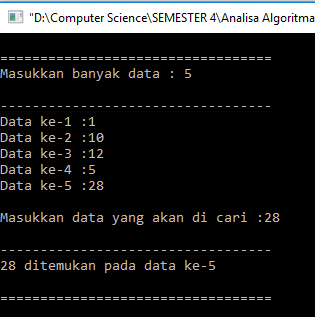
**}**

**cout << "\n=================================="<<endl;**

**getch();**

**}**

**Screenshoot :**

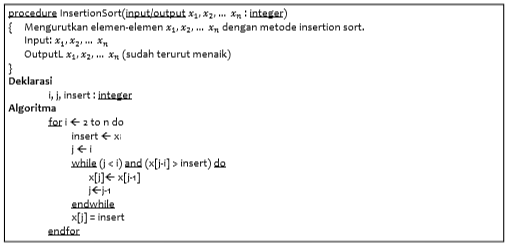


**Studi Kasus 4: Insertion Sort**

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.



Jawaban Studi Kasus 4

Loop sementara dijalankan hanya jika i> j dan arr [i] <arr [j]. Jumlah total iterasi loop sementara (Untuk semua nilai i) sama dengan jumlah inversi.

Kompleksitas waktu keseluruhan dari jenis penyisipan adalah O (n + f (n)) di mana f (n) adalah jumlah inversi.Jika jumlah inversi adalah O(n), maka kompleksitas waktu dari jenis penyisipan adalah O(n).

Dalam kasus terburuk, bisa ada inversi n \* (n-1) / 2. Kasus terburuk terjadi ketika array diurutkan dalam urutan terbalik. Jadi kompleksitas waktu kasus terburuk dari jenis penyisipan adalah O (n2).

**Source Code :**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Nama Program : Insertion sort**

**Nama : Felia Sri Indriyani**

**NPM : 140810170018**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <iostream>**

**#include <conio.h>**

**using namespace std;**

**int data[100],data2[100],n;**

**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;**

**}**

**}**

**int main()**

**{**

**cout << "\n=================================="<<endl;**

**cout<<"Masukkan Jumlah Data : "; cin>>n;**

**cout<<endl;**

**cout << "\n----------------------------------" << endl;**

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

**{**

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

**cin>>data[i];**

**data2[i]=data[i];**

**}**

**cout << "\n----------------------------------" << endl;**

**insertion\_sort();**

**cout<<"\nData Setelah di Sort : "<<endl;**

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

**{**

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

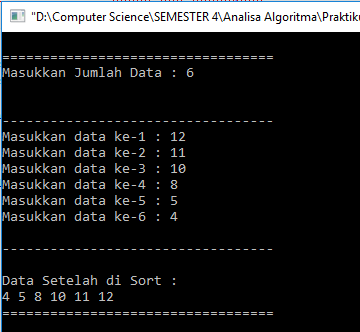
**}**

**cout << "\n=================================="<<endl;**

**getch();**

**}**

**Screenshoot :**

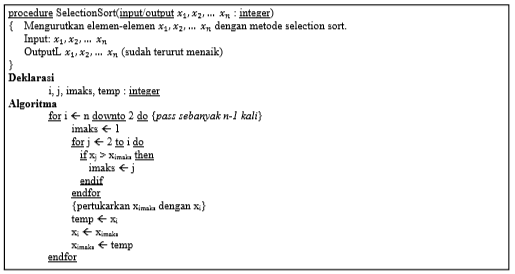


**Studi Kasus 5: Selection Sort**

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.



Jawaban Studi Kasus 5

1. Jumlah operasi perbandingan element. Untuk setiap *pass* ke-*i*,

*i*= 1 –>  jumlah perbandingan  = *n* – 1

*i* = 2 –>  jumlah perbandingan = *n* – 2

*i*= 3  –> jumlah perbandingan = *n* – 3

:

*i* = *k* –>  jumlah perbandingan = *n* – *k*

:

*i*= *n* – 1  –> jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah    *T*(*n*) = (*n* – 1) + (*n* – 2) + … + 1

Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada  batasan apakah data masukannya sudah terurut atau acak.

1. Jumlah operasi pertukaran

Untuk setiap*i* dari 1 sampai *n* – 1, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya   adalah  *T*(*n*) = *n* – 1.

Jadi, algoritma pengurutan maksimum membutuhkan *n*(*n* – 1 )/2 buah operasi perbandingan elemen dan *n* – 1  buah operasi pertukaran.

**Source Code :**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Nama Program : Selection sort**

**Nama : Felia Sri Indriyani**

**NPM : 140810170018**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**#include <iostream>**

**#include<conio.h>**

**using namespace std;**

**int data[100],data2[100];**

**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);**

**}**

**}**

**int main()**

**{**

**cout << "\n=====================================";**

**cout<<"\nMasukkan Jumlah Data : ";cin>>n;**

**cout << "\n-------------------------------------" << endl;**

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

**{**

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

**cin>>data[i];**

**data2[i]=data[i];**

**}**

**selection\_sort();**

**cout << "\n-------------------------------------" << endl;**

**cout<<"Data Setelah di Sort : "<<endl;**

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

**{**

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

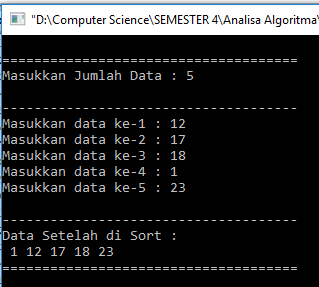
**}**

**cout << "\n=====================================\n";**

**getch();**

**}**

**Screenshoot :**



Teknik Pengumpulan

* Lakukan push ke github/gitlab untuk semua program dan laporan hasil analisa yang berisi jawaban dari pertanyaan-pertanyaan yang diajukan. Silahkan sepakati dengan asisten praktikum.

Penutup

* Ingat, berdasarkan Peraturan Rektor No 46 Tahun 2016 tentang Penyelenggaraan Pendidikan, mahasiswa wajib mengikuti praktikum 100%
* Apabila tidak hadir pada salah satu kegiatan praktikum segeralah minta tugas pengganti ke asisten praktikum
* Kurangnya kehadiran Anda di praktikum, memungkinkan nilai praktikum Anda tidak akan dimasukkan ke nilai mata kuliah.