**LAPORAN PRAKTIKUM 2**

**ANALISIS ALGORITMA**

****

**RIZKY ANUGERAH**

**140810180049**

**KELAS A**

Fakultas Matematika dan Ilmu Pengetahuan Alam

Program Studi Teknik Informatika

Universitas Padjadjaran

2020

**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, n.

Contoh algoritma pengurutan elemen-elemen larik, 􀀁 adalah jumlah elemen larik.

Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks n x n.

2. Kompleksitas waktu, T(n), adalah jumlah operasi yang dilakukan untuk melaksanakan

algoritma sebagai fungsi dari input n.

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

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 dengan cara menghitung

masing-masing jumlah operasi. Jika operasi tersebut berada di sebuah loop, maka jumlah

operasinya bergantung berapa kali loop tersebut diulangi.

(i) 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

t1 = 1 + 1 + n + n + 1 = 3 + 2n

(ii) Operasi penjumlahan

Jumlah + ak, n kali

k+1, n kali

Jumlah seluruh operasi penjumlahan adalah

t2 = n + n = 2n

(iii) 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:

T(n) = t1 + t2 + t3 = 3 + 2n + 2n + 1 = 4n + 4

**LATIHAN**

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

**Jawab**

*/\**

*Nama    : Rizky Anugerah*

*NPM     : 140810180049*

*Kelas   : A*

*Program : Pencarian Nilai Maksimal*

*\*/*

#include <iostream>

using namespace std;

main()

{

    int x[5] = {82,20,12,91,45};

    int n = sizeof(x) / sizeof(x[0]);

*// Deklarasi*

    int maks = x[0];

    int i = 2;

*// Algoritma*

    while (i <= n)

    {

        if (x[i] > maks)

        {

            maks = x[i];

        }

        i = i + 1;

    }

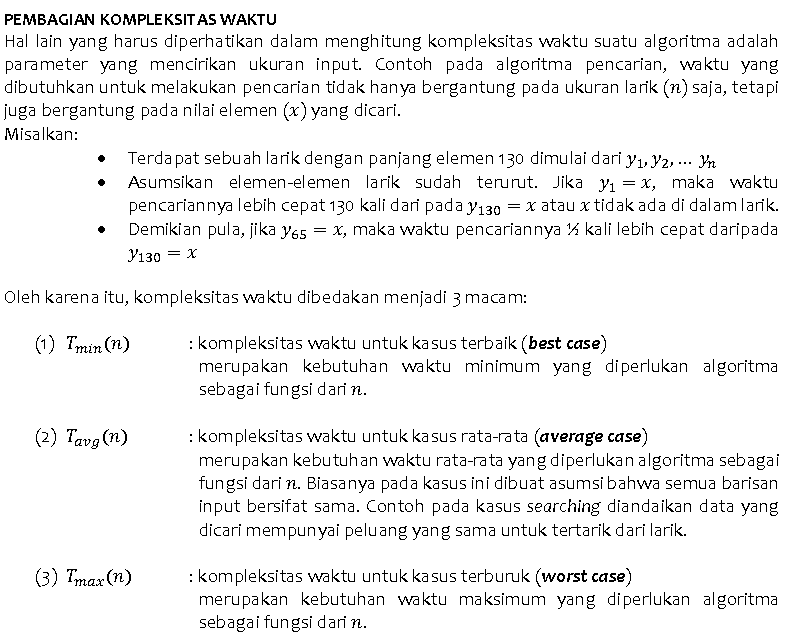
    cout << "Nilai terbesar dari larik tersebut : " << maks;

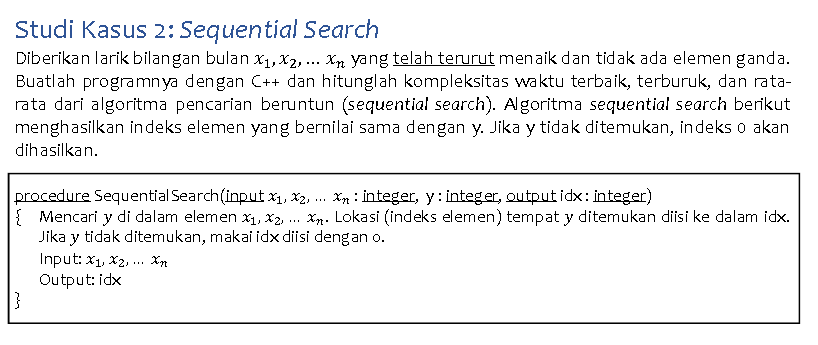
}

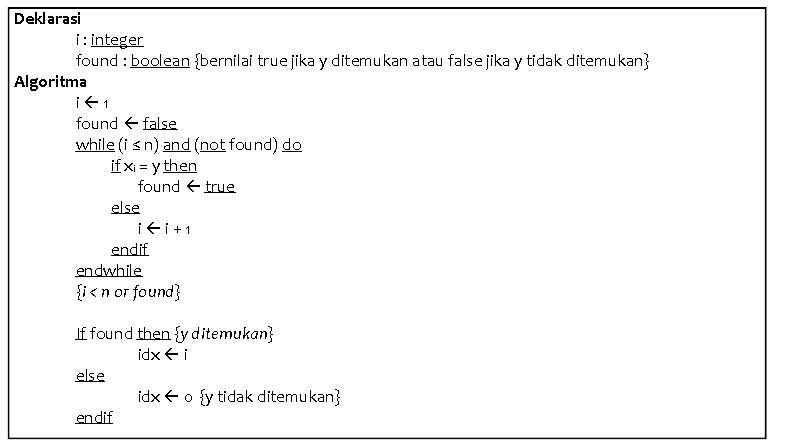
**Kompleksitas waktu:**

T(n) = 2 (n – 2) + (n – 2) + 2

= 3n - 4







**Jawab**

*/\**

*Nama    : Rizky Anugerah*

*NPM     : 140810180049*

*Kelas   : A*

*Program : Sequential Search*

*\*/*

#include <iostream>

using namespace std;

main()

{

    int x[5] = {1,2,3,4,5};

    int y = 2;*// integer yang akan dicari*

    int n = sizeof(x) / sizeof(x[0]);

*// Deklrasi*

    int i = 1;

    int idx;

    bool found = false;

*// Algoritma*

    while (i <= n && !found)

    {

        if (x[i] == y)

        {

            found = true;

        }

        else

        {

            i = i + 1;

        }

    }

    if (found == true)

    {

        idx = i;*// Jika y ditemukan*

    }

    else

    {

        idx = 0;*// Jika y tidak ditemukan*

    }

    cout << "y berada pada index : " << idx;

}

**Kompleksitas waktu:**

* Best Case: jika x1 = y

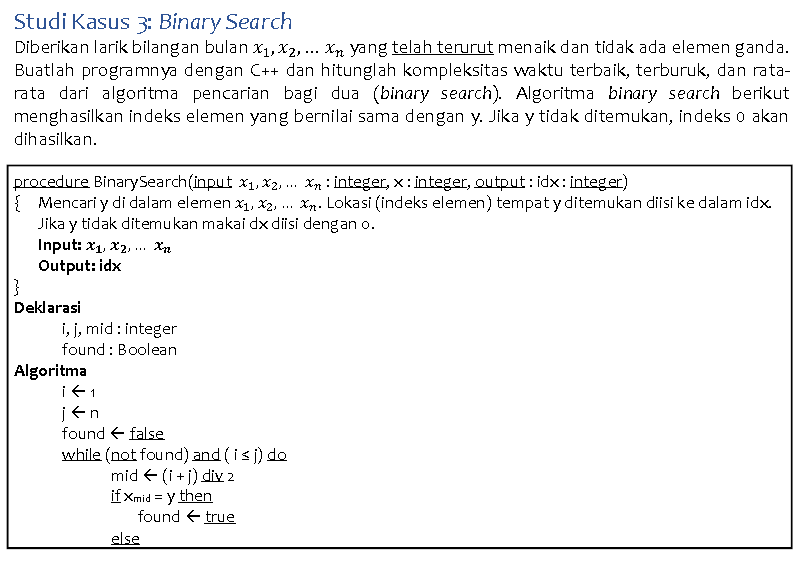
Tmin(n) = 1

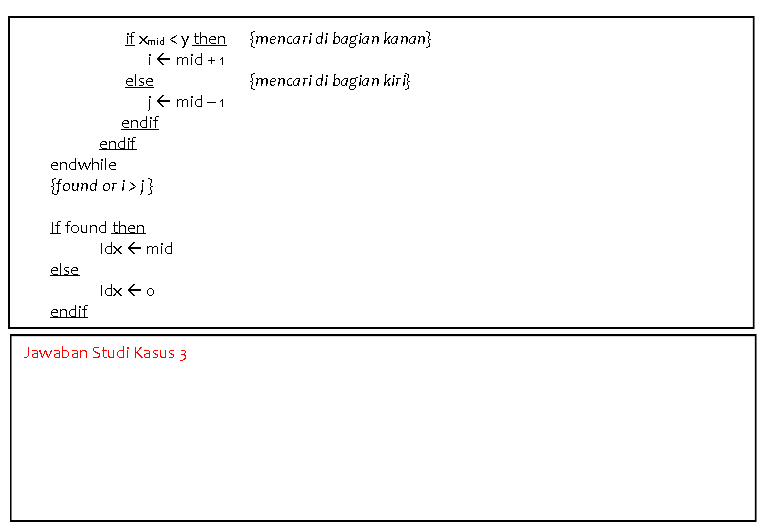
* Worst Case: jika xn = y atau y tidak ditemukan.

Tmax(n) = n

* Average Case: jika *y* ditemukan pada posisi ke-*j*, maka operasi perbandingan (ak = y) akan dieksekusi sebanyak *j* kali.

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





**Jawab**

*/\**

*Nama    : Rizky Anugerah*

*NPM     : 140810180049*

*Kelas   : A*

*Program : Binary Search*

*\*/*

#include <iostream>

using namespace std;

main()

{

    int x[5] = {1, 2, 3, 4, 5};*// Input*

    int y = 3;*// integer yang akan dicari*

    int idx;*// Output*

    int n = sizeof(x) / sizeof(x[0]);

*// Deklarasi*

    int i, j, mid;

    bool found;

*// Algoritma*

    i = 1;

    j = n;

    found = false;

    while (!found && i <= j)

    {

        mid = (i + j) / 2;

        if (x[mid] == y)

        {

            found = true;

        }

        else if (x[mid] < y)*// mencari di bagian kanan*

        {

            i = mid + 1;

        }

        else*// mencari di bagian kiri*

        {

            j = mid - 1;

        }

    }

    if (found == true)

    {

        idx = mid;

    }

    else

    {

        idx = 0;

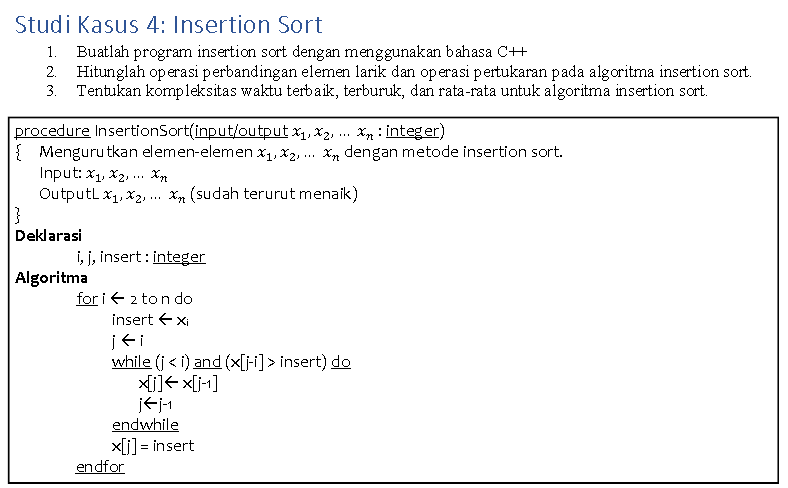
    }

    cout << "y berada pada index : " << idx;

}

**Kompleksitas waktu:**

* Best case: Jika ditemukan pada array[mid] atau indeks di tengah yaitu Tmin(n) = 1
* Average case: Jika ditemukan pada indeks di awal atau di akhir
* Worst case: Jika tidak ditemukan sama sekali yaitu Tmax (n) = 2log n



**Jawab**

*/\**

*Nama    : Rizky Anugerah*

*NPM     : 140810180049*

*Kelas   : A*

*Program : Insertion Sort*

*\*/*

#include <iostream>

using namespace std;

main()

{

    int x[5] = {3, 71, 42, 23, 96};

    int n = sizeof(x) / sizeof(x[0]);

*// Deklarasi*

    int i, j, insert;

*// Algoritma*

    for (i = 1; i < n; i++)

    {

        insert = x[i];

        j = i - 1;

        while (j >= 0 && x[j] > insert)

        {

            x[j + 1] = x[j];

            j = j - 1;

        }

        x[j + 1] = insert;

    }

    cout << "Hasil Insertion Sort : ";

    for (j = 0; j < n; j++)

    {

        cout << x[j] << " ";

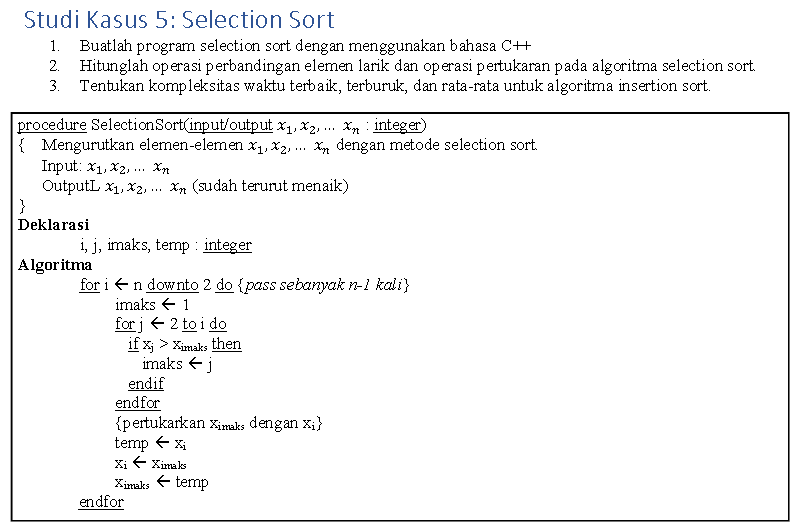
    }

}

**Kompleksitas waktu:**

* Best case: Jika array sudah terurut sehingga loop while tidak dijalankan
* Average case: Jika sebagian elemen array sudah terurut
* Worst case: Jika array harus diurutkan sebanyak n kali, 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).

|  |  |  |  |
| --- | --- | --- | --- |
| J | Perbandingan | Perpindahan | Total operasi |
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 4 |
| 4 | 3 | 3 | 6 |
| n | (n-1) | (n-1) | **2n-2 = 2(n-1)** |



**Jawab**

*/\**

*Nama    : Rizky Anugerah*

*NPM     : 140810180049*

*Kelas   : A*

*Program : Selection Sort*

*\*/*

#include <iostream>

using namespace std;

main()

{

    int x[5] = {1, 3, 5, 2, 4};

    int n = sizeof(x) / sizeof(x[0]);

*// Deklarasi*

    int i, j, imaks, temp;

*// Algoritma*

    for (i = 2; i < n; i++)

    {

        imaks = 1;

        for (j = 2; j < i; j++)

        {

            if (x[j] > x[imaks])

            {

                imaks = j;

            }

        }

        temp = x[i];

        x[i] = x[imaks];

        x[imaks] = temp;

    }

    cout << "Hasil Selection Sort : ";

    for (int i = 0; i < n; i++)

    {

        cout << x[i] << " ";

    }

}

**Kompleksitas waktu:**

* Jumlah operasi perbandingan elemen

Untuk setiap loop ke-i,

i = 1 🡪 jumlah perbandingan = n-1

i = 2 🡪 jumlah perbandingan = n-2

i = k 🡪 jumlah perbandingan = n-k

i = n-1 🡪 jumlah perbandingan = 1

sehingga T(n) = (n-1) + (n-2) + … + 1 = n(n-1)/2 dimana kompleksitas waktu ini berlaku menjadi yang terbaik, rata-rata maupun yang terburuk karena algoritma ini tidak melihat apakah arraynya sudah urut atau tidak terlebih dahulu.

* Jumlah operasi pertukaran

Untuk setiap loop ke-1 sampai n-1 terjadi satu kali pertukaran elemen sehingga T(n) = n-1.