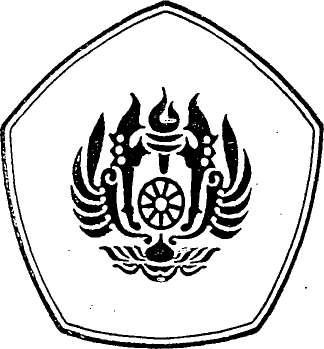
**LAPORAN MODUL PRAKTIKUM 2**

**KOMPLEKSITAS WAKTU DARI ALGORITMA**

**MATA KULIAH**

**ANALISIS ALGORITMA**

**D10G.4205 & D10K.0400601**



**DISUSUN OLEH:**

**MUHAMMAD SIRADJ AL FAUZI**

**140810170021**

**KELAS A**

**PROGRAM STUDI S-1 TEKNIK INFORMATIKA**

**DEPARTEMEN ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN FEBRUARI 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, 𝑛.

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

1. Kompleksitas waktu, 𝑇(𝑛), adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input 𝑛.
2. 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 sengan cra 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

# 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 | | |
| Jawaban Studi Kasus 1    maks  x1 1 kali i  2 1 kali maks  xi n kali i  i + 1 n kali    𝑇(𝑛) = 1 + 1 + 𝑛 + 𝑛 = 2𝑛 + 2 |
| Program Studi Kasus 1   #include<iostream>using namespace std;main(){int angka, n, max;cout << "Masukan angka : "; cin >> n;cout << endl;int i = 1;while (i <= n){cout << "Masukan Angka ke-"<< i << " : "; cin >> angka;if (i == 1){max = angka;}else if (max < angka){max = angka;}i++;}cout << endl;cout << "Nilai maksimum: " << max; } |

## **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 (𝑛) saja, tetapi juga bergantung pada nilai elemen (𝑥) yang dicari.

Misalkan:

* Terdapat sebuah larik dengan panjang elemen 130 dimulai dari 𝑦1, 𝑦2, … 𝑦𝑛
* Asumsikan elemen-elemen larik sudah terurut. Jika 𝑦1 = 𝑥, maka waktu pencariannya lebih cepat 130 kali dari pada 𝑦130 = 𝑥 atau 𝑥 tidak ada di dalam larik.
* Demikian pula, jika 𝑦65 = 𝑥, maka waktu pencariannya ½ kali lebih cepat daripada

## 𝑦130 = 𝑥

Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

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

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

merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari 𝑛. 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. 𝑇𝑚𝑎𝑥(𝑛) : kompleksitas waktu untuk kasus terburuk (***worst case***) merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari 𝑛.

# Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan 𝑥1,𝑥2,… 𝑥𝑛 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.

|  |  |
| --- | --- |
| procedure SequentialSearch(input 𝑥1, 𝑥2, … 𝑥𝑛 : integer, y : integer, output idx : integer)  { Mencari 𝑦 di dalam elemen 𝑥1, 𝑥2, … 𝑥𝑛. Lokasi (indeks elemen) tempat 𝑦 ditemukan diisi ke dalam idx. Jika 𝑦 tidak ditemukan, makai idx diisi dengan 0.  Input: 𝑥1, 𝑥2, … 𝑥𝑛  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    Best Case :  i  1 1 kali found  false 1 kali found  true 1 kali idx  i 1 kali  𝑇𝑚𝑖𝑛(𝑛) = 1 + 1 + 1 + 1 = 4  Average Case :  i  1 1 kali found  false 1 kali i  i + 1 ½ n kali  found  true 1 kali idx  i 1 kali  𝑇𝑎𝑣𝑔(𝑛) = 1 + 1 +𝑛 + 1 + 1 = 𝑛 + 4    Worst Case :  i  1 1 kali found  false 1 kali i  i + 1 n kali found  true 1 kali idx  i 1 kali  𝑇𝑚𝑎𝑥(𝑛) = 1 + 1 + 𝑛 + 1 + 1 = 𝑛 + 4 |
| Program Studi Kasus 2    #include <iostream> using namespace std;  int main()  {  int n; int x[10];  cout << "Masukkan Jumlah Data : "; cin >> n;  for (int i = 0; i < n; i++)  {  cout << "Masukkan Data ke - " << i+1 << " : ";  cin >> x[i];  }    int y;  cout << "Masukkan yang dicari : "; cin >> y;    int i = 0; bool found = false; int idx;  while ((i < n) && (!found))  {  if (x[i] == y) found = true; else i++;  }  if (found)  idx = i+1; else idx = 0;    cout << "Yang dicari berada di urutan : " << idx << endl;    return 0;  } |

## Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan 𝑥1,𝑥2,… 𝑥𝑛 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.

|  |
| --- |
| procedure BinarySearch(input 𝑥1, 𝑥2, … 𝑥𝑛 : integer, x : integer, output : idx : integer)  { Mencari y di dalam elemen 𝑥1, 𝑥2, … 𝑥𝑛. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx. Jika y tidak ditemukan makai dx diisi dengan 0.  **Input:** 𝒙𝟏, 𝒙𝟐, … 𝒙𝒏  **Output: idx**  }  **Deklarasi**  i, j, mid : integer found : Boolean  **Algoritma**   1.  1 2.  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*}   * 1.  mid + 1   else {*mencari di bagian kiri*}   * 1.  mid – 1   endif  endif  endwhile  {*found or i > j* }    If found then  Idx  mid else  Idx  0 endif |

|  |
| --- |
| Jawaban Studi Kasus 3  Best Case :  i  1 1 kali j  n 1 kali found  false 1 kali mid  (i + j) div 2 1 kali found  true 1 kali Idx  mid 1 kali  𝑇𝑚𝑖𝑛(𝑛) = 1 + 1 + 1 + 1 + 1 + 1 = 6    Average Case :  i  1 1 kali j  n 1 kali  fomid und  (i + j)  falsediv 2 1½ n + 1 kali kali  i  mid + 1 or j  mid – 1 ½ n kali  found  true 1 kali Idx  mid 1 kali    𝑇𝑎𝑣𝑔(𝑛) = 1 + 1 + 1 +𝑛 + 1 +𝑛 + 1 + 1 = 𝑛 + 6    Worst Case :  i j  1n 11 kali kali found  false 1 kali  mid  (i + j) div 2 n + 1 kali i  mid + 1 or j  mid – 1 n kali  found  true 1 kali Idx  mid 1 kali  𝑇𝑚𝑎𝑥(𝑛) = 1 + 1 + 1 + 𝑛 + 1 + 𝑛 + 1 + 1 = 2𝑛 + 6 |
| Program Studi Kasus 3    #include <iostream> using namespace std;  int main()  { int n; int x[10];  cout << "Masukkan Jumlah Data : "; cin >> n;  for (int i = 0; i < n; i++)  {  cout << "Masukkan Data ke - " << i+1 << " : "; cin >> x[i];  }  int y;  cout << "Masukkan yang dicari : "; cin >> y;  int i = 0;  int j = n-1;  bool found = false; int idx; int mid;  while ((i <= j) && (!found))  {  mid = (i + j)/2; if (x[mid] == y)  found = true; else  {  if (x[mid] < y) i = mid + 1; else  j = mid - 1;  }    }    if (found) idx = mid+1; else  idx = 0;    cout << "Yang dicari berada di urutan : " << idx << endl;    return 0;  } |

# 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 𝑥1, 𝑥2, … 𝑥𝑛 : integer)  { Mengurutkan elemen-elemen 𝑥1, 𝑥2, … 𝑥𝑛 dengan metode insertion sort.  Input: 𝑥1, 𝑥2, … 𝑥𝑛  OutputL 𝑥1, 𝑥2, … 𝑥𝑛 (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    Best Case :   2 to n do 1 kali for i    insert  xi  n kali j  i n kali x[j] = insert n kali    𝑇𝑚𝑖𝑛(𝑛) = 1 + 𝑛 + 𝑛 + 𝑛 = 3𝑛 + 1    Average Case :  for i  2 to n do 1 kali  insert  xi  n kali j  i n kali  x[j] x[j-1] n \* ½ n kali jj-1 n \* ½ n kali x[j] = insert n kali    (𝑛) = 1 + 𝑛 + 𝑛 + 1 𝑛2 + 1 𝑛2 + 𝑛 = 𝑛2 + 3𝑛 + 1  𝑇𝑎𝑣𝑔 2 2    Worst Case : for i  2 to n do 1 kali    insert  xi  n kali j  i n kali  x[j] x[j-1] n \* n kali jj-1 n \* n kali  x[j] = insert n kali    𝑇𝑚𝑎𝑥(𝑛) = 1 + 𝑛 + 𝑛 + 𝑛2 + 𝑛2 + 𝑛 = 2𝑛2 + 3𝑛 + 1 |
| Program Studi Kasus 4    #include <iostream> using namespace std;  int main()  {  int n;  int x[10];  cout << "Masukkan Jumlah Data : "; cin >> n;  for (int i = 0; i < n; i++)  {  cout << "Masukkan Data ke - " << i+1 << " : ";  cin >> x[i];  }  cout << "Data Sebelum di Sorting : ";  for (int i = 0; i < n; i++) cout << x[i] << " ";  cout << endl;    int insert;  int j;  for (int i = 1; i < n; i++)  {  insert = x[i];  j = i-1;  while ((j >= 0) && (x[j] > insert))  {  x[j+1] = x[j];  j--; }  x[j+1] = insert;  }  cout << "Data setelah di Sorting : "; for (int i = 0; i < n; i++) cout << x[i] << " ";  return 0;  } |

# 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 𝑥1, 𝑥2, … 𝑥𝑛 : integer)  { Mengurutkan elemen-elemen 𝑥1, 𝑥2, … 𝑥𝑛 dengan metode selection sort.  Input: 𝑥1, 𝑥2, … 𝑥𝑛  OutputL 𝑥1, 𝑥2, … 𝑥𝑛 (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 x j > ximaks then imaks  j  endif endfor  {pertukarkan ximaks dengan xi} temp  xi xi  ximaks ximaks  temp endfor |

|  |
| --- |
| Jawaban Studi Kasus 5      Best Case :  for i  n downto 2 do 1 kali  imaks  1 n kali  for j  2 to i do n kali  imaks  j n\*1 kali  temp xi  ximaks xi n kalin kali ximaks  temp n kali    𝑇𝑚𝑖𝑛(𝑛) = 1 + 𝑛 + 𝑛 + 𝑛 ∗ 1 + 𝑛 + 𝑛 + 𝑛 = 6𝑛 + 1      Average Case :  for i  n downto 2 do 1 kali  imaks  1 n kali  for j  2 to i do n kali  imaks  j n \* ½ n kali temp  xi  n kali xi  ximaks  n kali ximaks  temp n kali    1 2 + 𝑛 + 𝑛 + 𝑛 = 1 𝑛2 + 5𝑛 + 1  𝑇𝑎𝑣𝑔(𝑛) = 1 + 𝑛 + 𝑛 + 2 𝑛 2    Worst Case :  for i  n downto 2 do 1 kali  imaks  1 n kali  for j  2 to i do n kali  imaks  j n \* n kali temp  xi  n kali    xi  ximaks  n kali ximaks  temp n kali  2 + 𝑛 + 𝑛 + 𝑛 = 𝑛2 + 5𝑛 + 1  𝑇𝑚𝑎𝑥(𝑛) = 1 + 𝑛 + 𝑛 + 𝑛 |
| Program Studi Kasus 5    #include <iostream> using namespace std;  int main()  {  int n;  int x[10];  cout << "Masukkan Jumlah Data : "; cin >> n;  for (int i = 0; i < n; i++)  {  cout << "Masukkan Data ke - " << i+1 << " : "; cin >> x[i];  }  cout << "Data Sebelum di Sorting : "; for (int i = 0; i < n; i++) cout << x[i] << " ";  cout << endl;    int imaks;  int temp;  for (int i = n-1; i >= 1; i--)  {  imaks = 0;  for (int j = 1; j <= i; j++)  {  if (x[j] > x[imaks])  imaks = j;  }  temp = x[i]; x[i] = x[imaks]; x[imaks] = temp;  }    cout << "Data setelah di Sorting : "; for (int i = 0; i < n; i++) cout << x[i] << " ";    return 0;  } |

## 