BAB II

MATERI PENUNJANG

Dalam bab ini dijabarkan tentang beberapa teori yang dapat menunjang pembahasan pada bab III. Teori yang dijabarkan merupakan konsep dasar yang meliputi matriks, operasi perkalian matriks, algoritma, notasi big Oh, dan running time.

2.1. Matriks dan Operasi Perkalian Matriks

2.1.1. Matriks

Definisi 2.1.1.1.

Sebuah matriks adalah suatu susunan segi empat siku-siku dari bilangan-bilangan. Bilangan-bilangan dalam susunan tersebut dinamakan entri dalam matriks.

Contoh 2.1.1

Susunan berikut ini merupakan matriks.

$$\begin{bmatrix} 3 & 5 \\ 1 & 4 \\ 7 & 11 \end{bmatrix}, \quad \begin{bmatrix} 0 & 3 & -2 & 8 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 2 \\ 5 & 5 & 5 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 5 \end{bmatrix}$$

Seperti yang ditunjukkan pada contoh 2.1.1, maka ukuran matriks dapat bermacam-macam. Ukuran matriks itu sendiri dijelaskan dengan menyatakan banyaknya baris dan banyaknya kolom pada matriks tersebut. Sebagai contoh yaitu matriks pertama pada contoh 2.1.1 mempunyai 3 baris dan 2 kolom

sehingga ukuran dari matriks tersebut adalah 3 kali 2 yang biasanya ditulis dengan 3 x 2.

Dalam penulisannya, huruf besar digunakan untuk menyatakan matriks-matriks, sedangkan huruf kecil digunakan untuk menyatakan entri dalam matriks. Jika A adalah sebuah matriks, maka a_{ij} digunakan untuk menyatakan entri yang terdapat pada baris ke- i dan kolom ke- j dari A. Sehingga suatu matriks berukuran m x n secara umum dapat ditulis

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

sedangkan baris ke-i dari matriks A adalah matriks berukuran 1 x n dan untuk i=1,2,…,m, baris ke-i tersebut dapat ditulis

$$A^{(i)} = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix}$$

dan untuk j= 1,2,…,n kolom ke-j dari matriks A dapat ditulis

$$A_{(j)} = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

2.1.2. Operasi Perkalian Matriks

Definisi 2.1.2.1

Jika A adalah suatu matriks dan c adalah suatu skalar, maka hasil kali cA adalah matriks yang diperoleh dengan mengalikan masing-masing entri dari A oleh c.

Contoh 2.1.2

Jika A adalah matriks

$$A = \begin{bmatrix} -1 & 3 \\ 4 & 0 \\ 1 & 2 \end{bmatrix}$$

maka

$$3A = \begin{bmatrix} -3 & 9 \\ 12 & 0 \\ 3 & 6 \end{bmatrix} \quad \text{dan} \quad (-2A) = \begin{bmatrix} 2 & -6 \\ -8 & 0 \\ -2 & -4 \end{bmatrix}$$

Definisi 2.1.2.2.

Jika A adalah matriks berukuran m x r dan B adalah matriks berukuran r x n, maka hasil kali AB adalah matriks berukuran m x n yang entri-entrinya ditentukan sebagai berikut. Untuk mencari entri pada baris ke- i dan kolom ke- j dari matriks AB, dipilih baris ke- i dari matriks A dan kolom ke- j dari matriks B. Entri-entri yang bersesuaian dari baris dan kolom tersebut dikalikan bersama-sama dan kemudian hasil kali yang diperoleh dijumlahkan.

Misalkan C = AB dan c_{ij} adalah entri dari matriks C, maka c_{ij} berada pada baris ke- i dan kolom ke- j dari AB. Misalkan $A^{(i)}$ sebagai notasi dari baris ke- i matriks A dan $B_{(j)}$ sebagai kolom ke- j dari matriks B, maka dapat ditulis

$$C_{(j)}^{(i)} = A^{(i)}B_{(j)} \qquad (2.1.2.1)$$

Persamaan 2.1.2.1 dapat digambarkan pada hasil kali antara sembarang matriks A dan sembarang matriks B yang memenuhi syarat untuk operasi perkalian matriks.

$$AB = \begin{bmatrix} * & * & \cdots & * \\ a_{21} & a_{22} & \cdots & a_{2r} \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * \end{bmatrix} \begin{bmatrix} b_{11} & * & \cdots & * \\ b_{21} & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots \\ b_{r1} & * & \cdots & * \end{bmatrix} = \begin{bmatrix} * & * & \cdots & * \\ c_{21} & * & \cdots & * \\ \vdots & \vdots & \vdots & \vdots \\ * & * & \cdots & * \end{bmatrix}$$

Terlihat bahwa $c_{21} = a_{2*}\, b_{*1}$

$$= a_{21}.b_{11} + a_{22}.b_{21} + \ldots + a_{2r}.b_{r1}$$

Sehingga secara umum persamaan 2.1.2.1 dapat ditulis lengkap sebagai

$$c_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + \ldots + a_{ir}.b_{rj} \qquad (2.1.2.2)$$

Untuk memperjelas definisi perkalian dua buah matriks disajikan contoh 2.1.3

Contoh 2.1.3

Cari hasil kali AB dari

$$A = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 2 & 1 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & -1 \\ 1 & 1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix}$$

Penyelesaian

Karena A adalah matriks berukuran 2 x 4 dan B adalah matriks berukuran 4 x 2, maka hasil kali AB adalah matriks berukuran 2 x 2. Entri pada baris ke- 2 dan kolom ke- 2 dari AB dihitung sebagai berikut.

$$\begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & 1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} \square & \square \\ \square & 8 \end{bmatrix}$$

$(0.(-1)) + (2.1) + (1.0) + (3.2) = 8$

Perhitungan untuk hasil selebihnya adalah

$(1.0) + (0.1) + ((-1).2) + (2.1) = 0$

$$(1.(-1)) + (0.1) + ((-1).0) + (2.2) = 3$$

$$(0.0) + (2.1) + (1.2) + (3.1) = 7$$

$$AB = \begin{bmatrix} 0 & 3 \\ 7 & 8 \end{bmatrix}$$

2.1.3. Sifat Assosiatif dari Perkalian Matriks

Meskipun sifat komutatif untuk perkalian bilangan riil tidak berlaku pada operasi perkalian matriks, namun banyak sifat-sifat operasi pada bilangan riil yang berlaku pada matriks. Salah satu di antara sifat yang paling penting adalah sifat assosiatif pada operasi perkalian.

Theorema 2.1.3.1

Jika matriks A bersesuaian dengan matriks B dalam operasi perkalian matriks dan matriks B bersesuaian dengan matriks C, maka matriks A akan bersesuaian dengan matriks BC dan matriks AB akan bersesuaian dengan matriks C sehingga

$$A(BC) = (AB)C \qquad (2.1.3.5)$$

Bukti

Misalkan matriks A berukuran m x n, matriks B berukuran n x p, dan matriks C berukuran p x s maka matriks (AB) berukuran m x p dan matriks (BC) berukuran n x s. Baris ke-i dari matriks (AB) untuk i=1,2,…,m adalah

$$(AB)^{(i)} = (A^{(i)}B_{(1)} \quad A^{(i)}B_{(2)} \quad \dots \quad A^{(i)}B_{(p)})$$

dan kolom ke-j dari matriks (BC) untuk j= 1,2,…,s adalah

$$(BC)_{(j)} = \begin{pmatrix} B^{(1)}C_{(j)} \\ B^{(2)}C_{(j)} \\ \vdots \\ B^{(n)}C_{(j)} \end{pmatrix}$$

sehinga untuk sembarang posisi i dan j berlaku

$$((AB)C)^{(i)}_{(j)} = (AB)^{(i)}C_{(j)}$$

$$= (\mathrm{A}^{(i)}\mathrm{B}_{(1)}).\mathrm{c}_{1j} + (\mathrm{A}^{(i)}\mathrm{B}_{(2)}).\mathrm{c}_{2j} + \ldots + (\mathrm{A}^{(i)}\mathrm{B}_{(p)}).\mathrm{c}_{pj}$$

$$= (a_{i1}b_{11} + a_{i2}b_{21} + \ldots + a_{in}b_{n1}).c_{1j} + (a_{i1}b_{12} + a_{i2}b_{22} + \ldots + a_{in}b_{n2}).c_{2j}$$

$$+ \ldots + (a_{i1}b_{1p} + a_{12}b_{2p} + \ldots + a_{in}b_{np}).c_{pj} \qquad \text{(i)}$$

$$(A(BC))^{(i)}_{(j)} = A^{(i)}(BC)_{(j)}$$

$$= a_{i1}.(B^{(1)}C_{(j)}) + a_{i2}.(B^{(2)}C_{(j)}) + \ldots + a_{in}.(B^{(n)}C_{(j)})$$

$$= a_{i1}(b_{11}c_{1j} + b_{12}c_{2j} + \ldots + b_{1p}c_{pj}) + a_{i2}(b_{21}c_{1j} + b_{22}c_{2j} + \ldots + b_{2p}c_{pj})$$

$$+ \ldots + a_{in}(b_{n1}c_{1j} + b_{n2}c_{2j} + \ldots + b_{np}c_{pj}) \qquad \text{(ii)}$$

karena i dan j adalah sembarang maka pembandingan antara persamaan (i) dan (ii) menunjukkan bahwa (AB)C = A(BC)

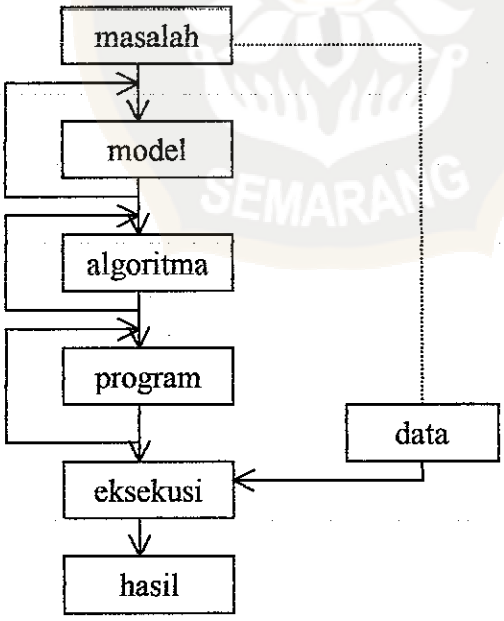
2.2. Algoritma

Terbentuknya model matematika yang sesuai pada suatu pemecahan masalah hanya merupakan bagian dari pemecahan masalah tersebut. Oleh karena itu untuk melengkapi proses pemecahan masalah tersebut diperlukan suatu metode atau langkah-langkah yang akan digunakan untuk membawa model matematika

tersebut ke arah penyelesaiannya. Langkah-langkah tersebut biasa disebut dengan algoritma.

Algoritma pertama kali diperkenalkan oleh seorang ahli matematika yaitu Abu Ja'far Muhammad Ibnu Musa Al Khawarizmi, dimana secara bahasa algoritma mempunyai arti suatu metode khusus untuk menyelesaikan suatu masalah yang nyata. Di dalam bidang pemrograman, algoritma didefinisikan sebagai suatu metode khusus yang tepat dan terdiri dari serangkaian langkah yang terstruktur dan ditulis secara matematis yang akan dikerjakan untuk menyelesaikan masalah dengan bantuan komputer.

Di dalam prosesnya, hubungan antara masalah, algoritma, dan hasil dapat digambarkan sebagai berikut :



Gambar 2.2.1. Diagram hubungan masalah, algoritma, dan hasil

Suatu algoritma yang baik mempunyai beberapa kriteria sebagai berikut :

1. Adanya input

   Suatu algoritma mempunyai input yang diperoleh dari himpunan objek yang telah ditetapkan.

2. Adanya output

   Dalam menyelesaikan suatu masalah, algoritma harus menghasilkan output yang merupakan solusi atau hasil dari masalah yang sedang diselesaikan.

3. Efektif dan efisien

   Suatu algoritma dikatakan efektif jika algortima tersebut dapat menghasilkan suatu solusi yang sesuai dengan masalah yang sedang diselesaikan. Sedangkan algortima yang efisien adalah algoritma yang mempunyai waktu proses yang singkat dan penggunaan memorinya sedikit.

4. Jumlah langkahnya berhingga

   Banyaknya langkah yang disusun harus berhingga, karena jika tidak demikian maka proses yang dilakukan akan memerlukan waktu yang relatif lebih lama.

5. Berakhir

   Proses yang dilakukan dalam mencari suatu penyelesaian masalah harus berhenti atau berakhir dengan hasil akhir berupa solusi atau berupa informasi tentang tidak diketemukannya solusi. Dengan perkataan lain, baik dalam kondisi solusinya ada maupun tidak maka proses tetap harus mempunyai akhir.

6. Terstruktur

Urutan dari barisan langkah-langkah yang digunakan harus disusun sedemikian rupa agar proses penyelesaiannya tidak berbelit-belit dan memungkinkan waktu proses menjadi lebih singkat serta akan mempermudah di dalam melakukan pemeriksaan ulang.

2.3. Pengertian dan Sifat-Sifat Big Oh

Definisi 2.3.1

Diberikan S, himpunan semua fungsi yang mempunyai domain D yang mewakili himpunan bilangan riil. Dikatakan f(n) = O(g(n)), dibaca ' f(n) adalah big-Oh dari g(n)' jika ditemukan pasangan bilangan positif c dan k sedemikian hingga berlaku

$$| f(n) | \leq c.| g(n) |$$

untuk semua $n \in D$ dan $n \geq k$

Dari definisi 2.3.1 maka untuk menunjukkan f(n) = O (g(n)) hanya dibutuhkan untuk menentukan sepasang konstanta c dan k sedemikian hingga jika $n \geq k$ maka $| f(n) | \leq c.| g(n) |$. Penulisan f(n) = O(g(n)) berarti bahwa f(n) merupakan anggota dari O(g(n)) .

Contoh 2.3.1

Jika suatu fungsi $f(n) = 3n^3 + 2n^2$ merupakan fungsi dari waktu tempuh suatu algoritma maka f(n) adalah big-Oh dari n^3, yang dinotasikan $f(n) = O(n^3)$

Penyelesaian

Berdasarkan definisi 2.3.1 bahwa jika $f(n) = O(n^3)$ maka terdapat dua konstanta bulat positif c dan k sedemikian hingga berlaku

$$| 3n^3 + 2n^2 | \leq c.| n^3|$$

$$n = 0 \rightarrow f(0) = 0$$

$$n = 1 \rightarrow f(1) = 3 + 2 = 5$$

ambil $k = 0$ dan $c = 5$ sehingga berlaku $3n^3 + 2n^2 \leq 5n^3$ untuk $n \geq 0$.

Jadi terbukti bahwa $f(n) = O(n^3)$.

Teorema 2.3.1

Misalkan $f(n) = a_p n^p + a_{p-1}n^{p-1} + \ldots + a_1 n + a_0$ adalah fungsi polinomial derajat p dengan $a_p \neq 0$ dan $g(n) = n^p$, maka $f(n) = O(g(n))$.

Bukti :

Untuk membuktikan teorema 2.3.1 akan dicari sepasang konstanta c dan k sedemikian hingga $| f(n) | \leq c.| g(n) |$ untuk $n \geq k$

Dipilih $c = | a_p | + | a_{p-1} | + \ldots + | a_1 | + | a_0 |$ dan $k = 1$, maka untuk $n \geq 1$

$$| f(n) | = | a_p n^p + a_{p-1}n^{p-1} + \ldots + a_1 n + a_0 |$$

$$\leq | a_p n^p | + | a_{p-1}n^{p-1} | + \ldots + | a_1 n | + | a_0 |$$

$$\leq | a_p | | n^p | + | a_{p-1} | | n^{p-1} | + \ldots + | a_1 | | n | + | a_0 |$$

$$\leq | a_p | | n^p | + | a_{p-1} | | n^p | + \ldots + | a_1 | | n^p | + | a_0 | | n^p |$$

$= (| a_p | + | a_{p-1} | + \ldots + | a_1 | + | a_0 |) | n^p |$

$= c. | n^p |$

$= c.| g(n) |$

terlihat bahwa $| f(n) | \leq c.| g(n) |$, sehingga terbukti bahwa $f(n) = O(g(n))$.

Teorema 2.3.2

Misalkan S adalah himpunan semua fungsi dengan domain D yang merupakan himpunan bilangan riil. Fungsi-fungsi $f_1(n)$, $f_2(n)$, $g_1(n)$, dan $g_2(n)$ adalah anggota-anggota S sedemikian hingga $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka berlaku :

(a). $f_1(n) + f_2(n) = O(\max \{g_1(n),g_2(n)\})$.

(b). $f_1(n) . f_2(n) = O(g_1(n) . g_2(n))$

Bukti

(a). Akan dibuktikan bahwa terdapat sepasang konstanta c dan k sedemikian hingga untuk $n \geq k$ berlaku

$$|f_1(n) + f_2(n) | = c.| \max\{g_1(n),g_2(n)\} |$$

Karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1, c_2, k_1 dan k_2 sedemikian hingga untuk $n \in D$ di mana $n \geq k_1$ dan $n \geq k_2$ berlaku :

$$| f_1(n) + f_2(n) | \leq | f_1(n) | + | f_2(n) |$$

$$\leq c_1.| g_1(n) | + c_2.| g_2(n) |$$

$$= c_1,g_1(n) + c_2.g_2(n)$$

$\leq c_1.\max\{g_1(n), g_2(n)\} + c_2.\max\{g_1(n), g_2(n)\}$

$= (c_1+c_2).\max\{g_1(n), g_2(n)\}$

$\leq c.|\max\{g_1(n), g_2(n)\}|$,untuk $c = c_1 + c_2$

maka untuk $k = \max\{k_1, k_2\}$ terbukti bahwa

$f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$.

(b). Karena $f_1(n) = O(g_1(n))$ dan $f_2(n) = O(g_2(n))$ maka terdapat c_1, c_2, k_1, dan k_2 bilangan positif sedemikian hingga untuk $n \in D$ dengan $n \geq k_1$ dan $n \geq k_2$ maka berlaku $|f_1(n)| \leq c_1.|g_1(n)|$ dan $|f_2(n)| \leq c_2.|g_2(n)|$.

Pilih $k = \max\{k_1, k_2\}$ dan $c = c_1.c_2$, maka

$|f_1(n).f_2(n)| = |f_1(n)|.|f_2(n)|$

$\leq c_1.|g_1(n)|.c_2.|g_2(n)|$

$= c_1.c_2.|g_1(n)|.|g_2(n)|$

$= c.|g_1(n).g_2(n)|$

sehingga terbukti bahwa $f_1(n).f_2(n) = O(g_1(n).g_2(n))$

2.4. Running Time

Proses suatu algoritma di dalam mencari solusi dari suatu masalah memerlukan waktu tertentu, di mana satuan waktu yang dibutuhkan diharapkan dalam waktu yang relatif singkat. Di dalam suatu analisa algoritma untuk menentukan kecepatan prosesnya digunakan istilah running time dan biasanya

dinotasikan dengan T(n). Adapun hal-hal yang mempengaruhi besar kecilnya nilai running time adalah :

a. Banyaknya langkah

Semakin banyak langkah atau instruksi yang digunakan maka semakin besar nilai running time yang dibutuhkan dalam proses tersebut.

b. Ukuran atau besar data

Ukuran atau besar data yang digunakan akan sangat berpengaruh pada proses perhitungan.

c. Jenis operasi

Jenis operasi yang digunakan juga berpengaruh pada running time. Jenis operasi tersebut meliputi operasi aritmetika, operasi nalar atau logika.

Karena terdapat beberapa hal yang dapat mempengaruhi besar kecilnya nilai running time suatu algoritma maka terdapat kemungkinan bahwa di dalam proses penyelesaian masalah suatu algoritma mempunyai kondisi yang berbeda. Oleh karena itu perhitungan running time suatu algoritma dalam proses pencarian solusi suatu masalah dibedakan dalam tiga keadaan, yaitu :

(i). Keadaan terburuk (*Worst Case*)

Merupakan suatu keadaan yang terburuk dari proses di dalam suatu algoritma. Sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang maksimum.

(ii). Keadaan rata-rata (*Average Case*)

Hal ini merupakan suatu keadaan dari running time yang ekivalen dengan nilai harapan dari fungsi running time untuk setiap input yang mungkin. Analisa untuk kondisi ini jarang dilakukan, karena adanya kesulitan dalam menentukan suatu data yang secara umum dapat mewakili keadaan rata-rata data yang digunakan.

(iii). Keadaan Terbaik (*Best Case*)

Merupakan suatu keadaan yang terbaik dari proses di dalam suatu algoritma. Sehingga waktu yang ditempuh oleh algoritma tersebut adalah waktu yang minimum.

Untuk selanjutnya dalam menentukan running time suatu algoritma diasumsikan terjadi keadaan terburuk. Hal ini dilakukan dengan beberapa alasan sebagai berikut :

(1). Keadaan terburuk dapat menerima semua batasan data masukan meskipun untuk masukan yang tidak bagus.

(2). Running time pada keadaan terburuk dari suatu algoritma merupakan batas atas dari running time atau dapat dikatakan sebagai waktu maksimal yang dibutuhkan. Sehingga dengan demikian akan memberikan suatu jaminan bahwa tidak ada running time yang lebih besar.

(3). Untuk beberapa algoritma, keadaan terburuk merupakan kejadian yang sering terjadi. Sebagai contoh adalah dalam pencarian informasi pada suatu basis

data, keadaan terburuk selalu terjadi apabila informasi yang terjadi tidak berada dalam basis data.

Suatu algoritma mempunyai ciri-ciri khusus yang mungkin berbeda satu dengan yang lain, oleh karena itu dibutuhkan identifikasi algoritma secara signifikan. Untuk menganalisa running time suatu algoritma harus diketahui aturan-aturan yang dapat menyederhanakan perhitungan. Aturan-aturan yang diperlukan dalam analisa tersebut adalah :

(1). Running time setiap assigment (tugas), baca (read), dan stateman write besarnya adalah O(1).

(2). Loop

Running time dari suatu loop adalah jumlah iterasi yang dilakukan untuk menyelesaikan statemen-statemen dalam loop tersebut.

(3). Loop berkalang

Jika dalam suatu algoritma terdapat loop berkalang, yaitu loop yang berada di dalam loop, maka yang menjadi pedoman perhitungan adalah loop yang paling dalam. Running time-nya adalah hasil dari perkalian semua ukuran loop.

(4). Statemen berurutan

Apabila terdapat statemen berurutan, maka penentuan running time-nya menggunakan running time maksimal dari running time yang ada.

(5). Statemen if kondisi then statemen

Running time untuk memeriksa kondisi secara normal adalah O(1). Dan untuk statemen adalah running time pada saat statemen dieksekusi (kondisi terpenuhi).

Contoh 2.4.1

Misalkan himpunan A merupakan sebuah array yang terdiri dari n elemen, yaitu A(1), A(2), ..., A(n). Akan dilakukan pengurutan bilangan dengan urutan naik. Variabel key menunjukkan variabel kunci untuk perbandingan. Algoritma pengurutan bilangan tersebut adalah sebagai berikut

Insertion_sort(A)		Times
1	for j ← 2 to n	n
2	do key ← A[j]	n –1
3	{insert A[j] into the sorted sequence A[1..j-1]}	n –1
4	i ← j – I	n –1
5	while i > 0 and A[i] > key	$\sum_{j=2}^{n} t_j$
6	do A[i+1] ← A[i]	$\sum_{j=2}^{n} (t_j - 1)$
7	i← I – 1	$\sum_{j=2}^{n} (t_j - 1)$
8	A[i + 1] ← key	n -1

Dengan times adalah jumlah maksimal eksekusi yang dapat dilakukan untuk setiap statemen dan t_j adalah harga running time statemen pada saat j.

Penjelasan

Running time algoritma pada contoh 2.4.1 adalah jumlah running time dari setiap statemen yang dijalankan.

$T(n) = n + (n-1) + (n-1) + \sum_{j=2}^{n} t_j + \sum_{j=2}^{n}(t_j - 1) + \sum_{j=2}^{n}(t_j - 1) + (n-1)$

Keadaan terbaik terjadi jika array data yang dimasukkan dalam keadaan sudah terurut naik. Apabila terjadi keadaan terbaik, maka hanya statemen pada nomer 5 yang dikerjakan sedangkan statemen 6 dan 7 tidak perlu dikerjakan sehingga $t_j = 1$ untuk $j = 2,3,\ldots,n$ dan running time-nya adalah

$T(n) = n + (n-1) + (n-1) + (n-1) + (n-1)$

$= 5n + 4$

$= O(n)$

Jika array data tersaji dalam keadaan sebaliknya, yaitu tersaji dalam keadaan terurut menurun (urut dari besar ke kecil), maka terjadi keadaan terburuk. Dalam keadaan ini maka elemen A[j] harus dibandingkan dengan setiap elemen terurut pada subarray A[1..j-1] secara menyeluruh, sehingga $t_j = j$ untuk $j = 2,3,\ldots,n$.

Diketahui bahwa

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \quad \text{dan} \quad \sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$$

sehingga running time-nya adalah

$$T(n) = n + (n-1) + (n-1) + \left(\frac{n(n+1)}{2} - 1\right) + \left(\frac{n(n-1)}{2}\right) + \left(\frac{n(n-1)}{2}\right) + (n-1)$$

$$= \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

$= O(n^2)$