**MODUL PRAKTIKUM 2**

**ANALISIS ALGORITMA**



**Disusun Oleh:**

Nama : Kevin Akbar Adhiguna

NPM : 140810170055

Kelas : A

**PROGRAM STUDI S-1 TEKNIK INFORMATIKA**

**DEPARTEMEN ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**MARET 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 .

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 :

**Kompleksitas Waktu**

**Jika if tidak dihitung :**

Operasi pengisian nilai (*assignment*)

maks 🡨 x1 1 kali

i 🡨 2 1 kali

maks 🡨 xi i n-1 kali

i 🡨 i + 1 n-1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

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

**Jika if dihitung :**

Operasi pengisian nilai (*assignment*)

maks 🡨 x1 1 kali

i 🡨 2 1 kali

maks 🡨 xi i n-1 kali

i 🡨 i + 1 n-1 kali

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

Operasi perbandingan

if xi > maks n-1 kali

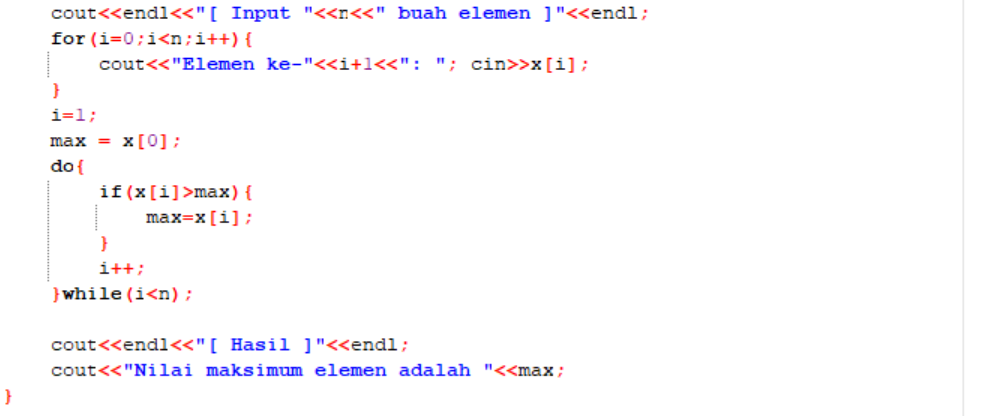
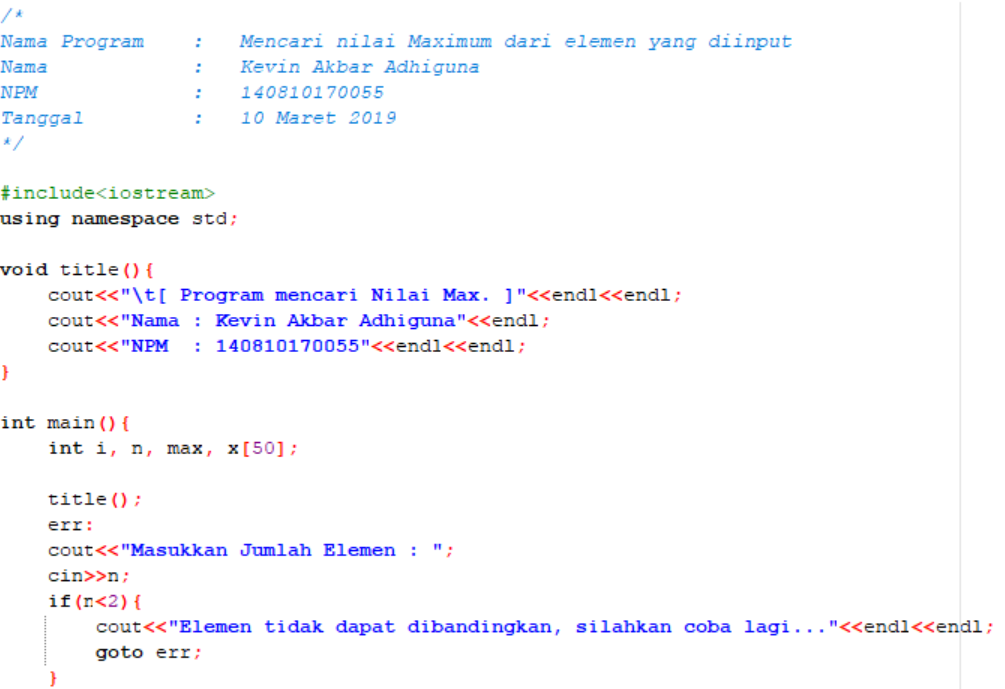
Jumlah seluruh operasi perbandingan adalah

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

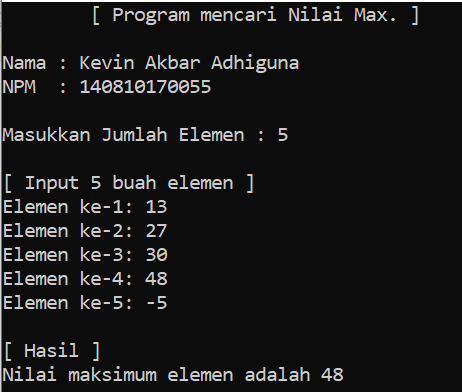
Simpulan :

Kompleksitas Waktu tetap , meskipun if dihitung atau tidak. Mengacu perkataan pa Ino Suryana, M.Kom. , if boleh dihitung dan boleh juga tidak. Meskipun terkadang hal tersebut tergantung buku yang menjadi acuan. Dalam buku : *Munir, Rinaldi, 2008, Matematika Diskrit* , kondisi if tidak dihitung.

**Program**

****

**Hasil Compile & Run program**

****

**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 :

**Kompleksitas Waktu**

Operasi :

i 🡨 1 [ Best : 1 Worst : 1 Average : (1+1)/2 = 1 ]

found 🡨 false [ Best : 1 Worst : 1 Average : (1+1)/2 = 1 ]

found 🡨 true [ Best : 1 Worst : 1 Average : (1+1)/2=1 ]

i 🡨 i + 1 [ Best : 0 Worst : n-1 Average : (0+(n-1))/2 = (n-1)2 ]

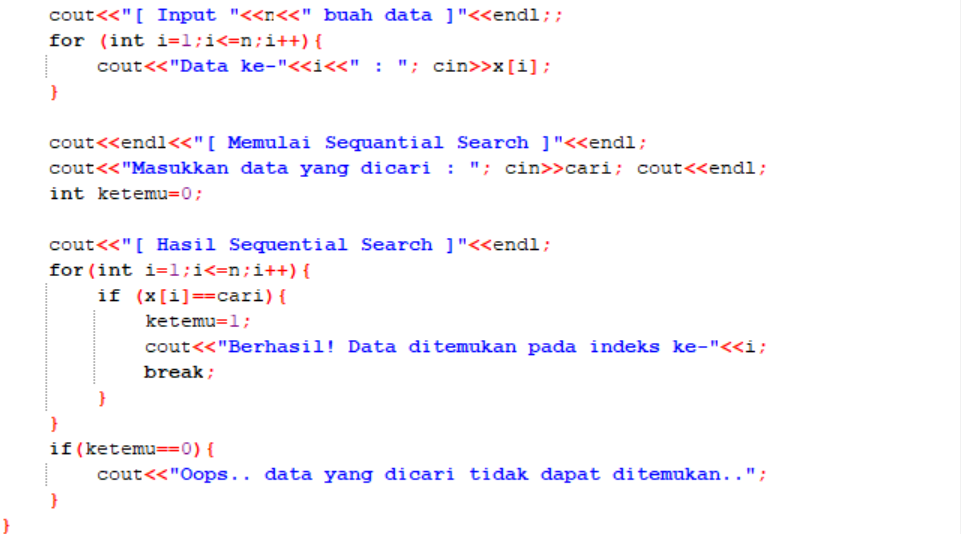
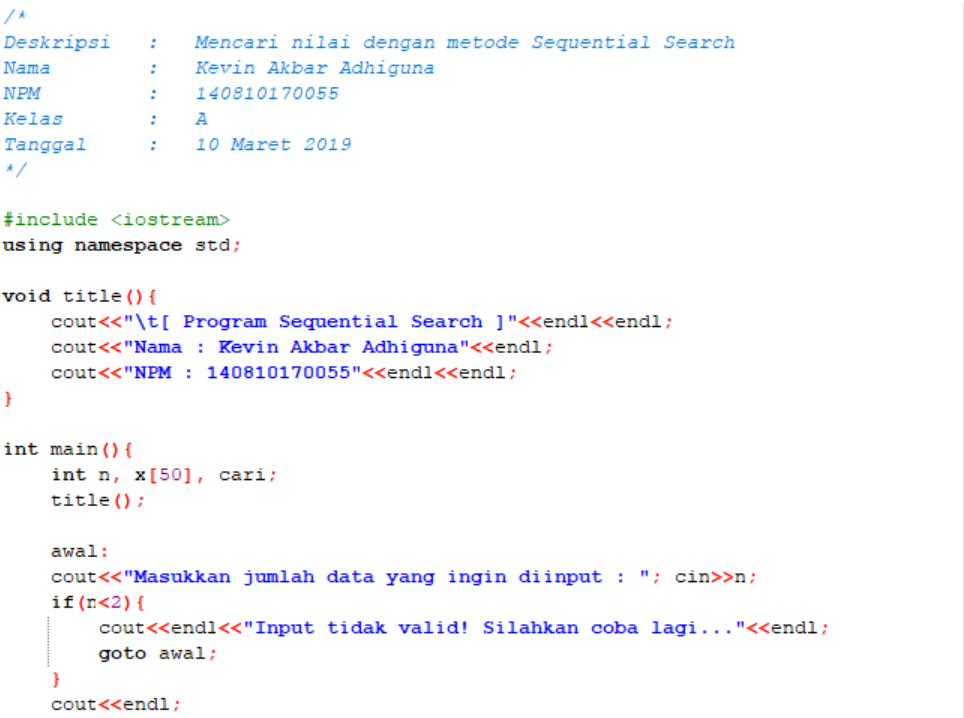
idx 🡨 i atau idx 🡨 0 [ Best : 1 Worst : 1 Average : (1+1)/2 = 1 ]

Best Case :

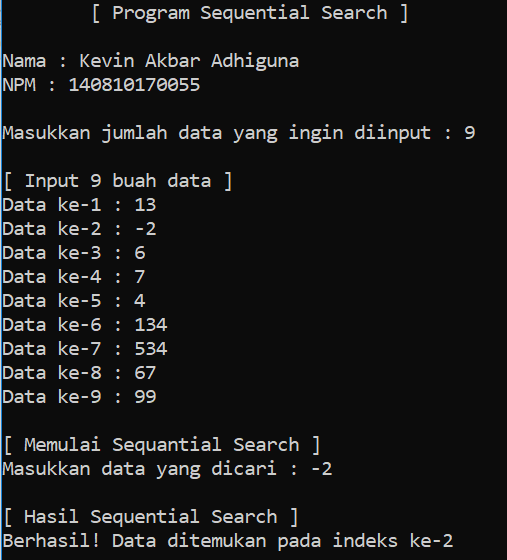
Worst Case :

Average :

**Program**

****

**Hasil Compile & Run program**

****

# 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 :

**Kompleksitas Waktu**

1. 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

Oleh karena itu, diperoleh

1. Worst Case

Pada kasus terburuk, elemen ditemukan ketika ukuran larik = 1. Pada kasus terburuk ini, ukuran larik setiap kali memasuki looping while-do adalah :

(sebanyak 2 kali)

Ini berarti, looping while-do dikerjakan sebanyak 2 kali.

Contoh :

n=128 maka 128 -> 64 -> 32 -> 16 -> 8 -> 4 -> 2 -> 1 (sebanyak 2 = 7 kali pembagian)

Sehingga jumlah Operasi perbandingan elemen (amid = x) adalah :

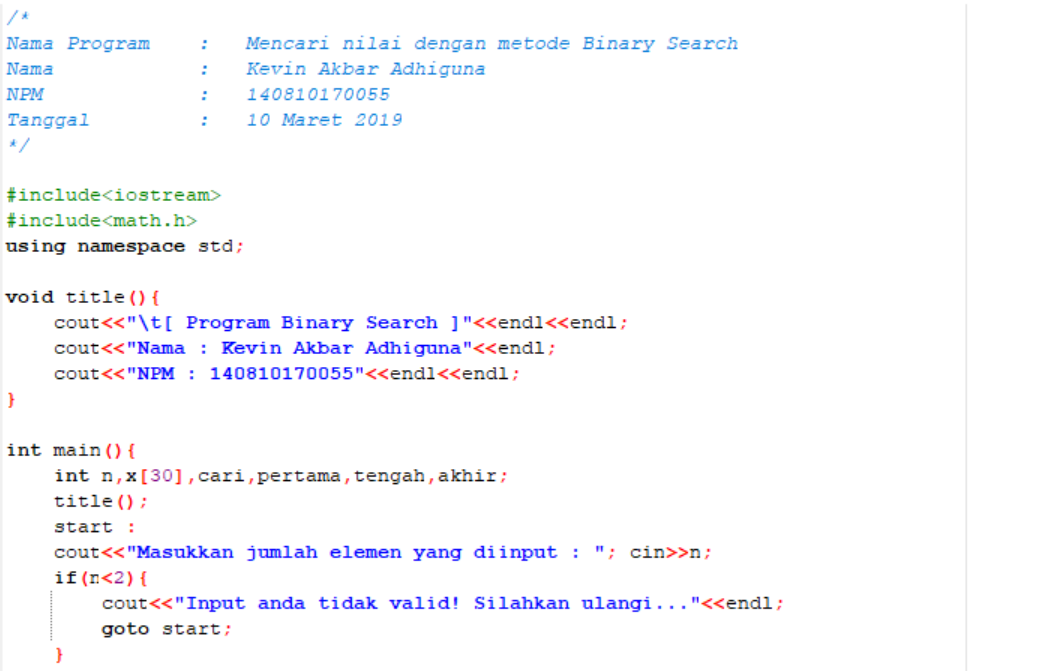
2

3. Average Case

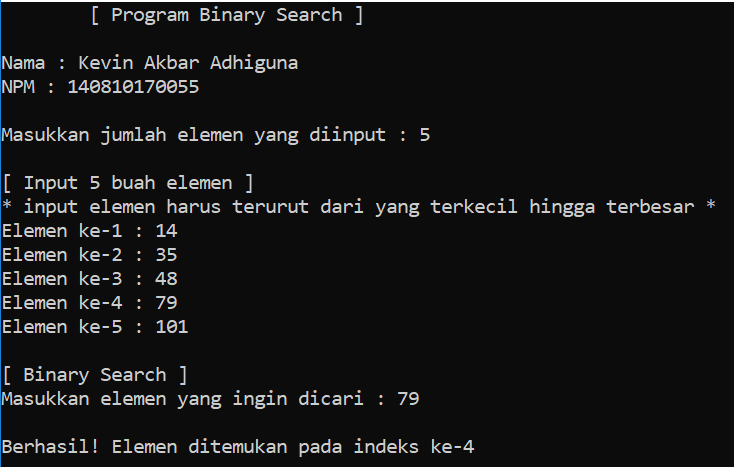
Average Case dapat dihitung dengan cara : (Best Case + Worst Case) / 2, sehingga :

2/2

**Program**

****

**Hasil Compile & Run program**

****

# 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 :

**Kompleksitas Waktu**

1. Best Case

insert 🡨 xi 1 kali

j 🡨 i 1 kali

x[j]🡨 x[j-1] 1 kali

j🡨j-1 1 kali

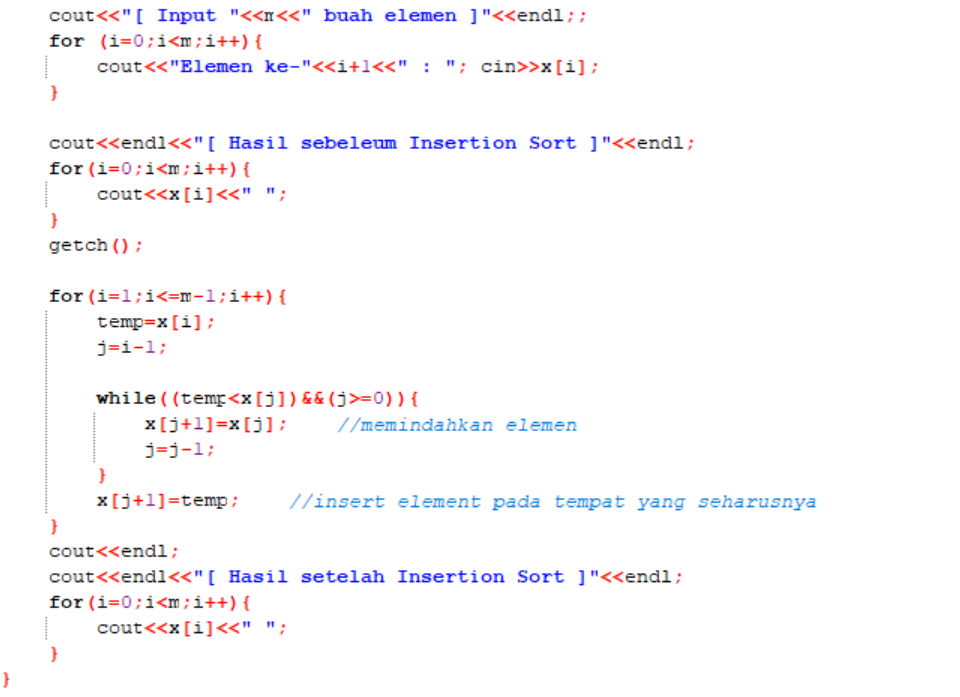
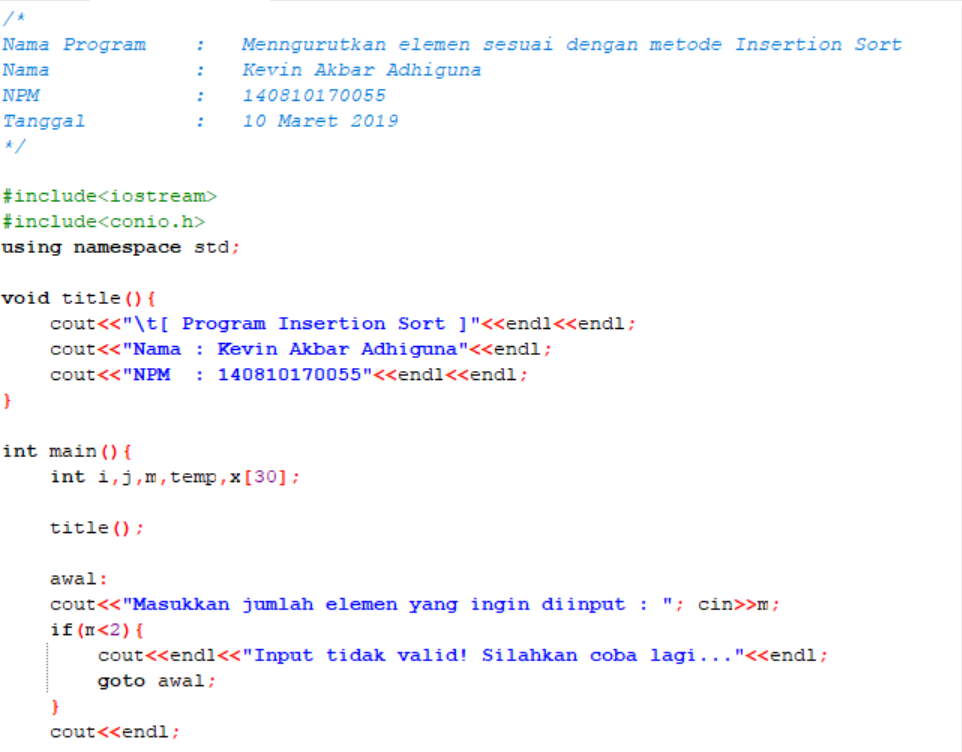
Oleh karena itu,

1. Worst Case

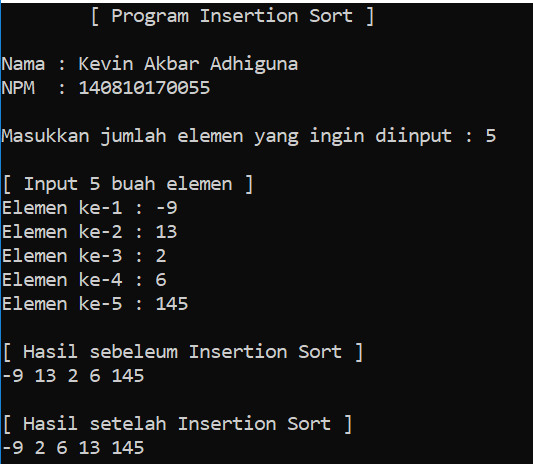
|  |  |  |  |
| --- | --- | --- | --- |
| J | Perbandingan | Pertukaran | Total Operasi |
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 4 |
| 4 | 3 | 3 | 6 |
| 5 | 4 | 4 | 8 |
| N | (n-1) | (n-1) | 2(n-1) |

1. Average Case

**Program**

****

**Hasil Compile & Run program**

****

# 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 :

**Kompleksitas Waktu**

1. Jumlah operasi perbandingan elemen

Untuk setiap pass ke-I, i=n,n-1, … ,2, operasi perbandingan elemen yang dilakukan adalah sebagai berikut :

--> Jumlah operasi perbandingan elemen = n - 1

--> Jumlah operasi perbandingan elemen = n - 2

--> Jumlah operasi perbandingan elemen = n - 3

.

.

.

--> Jumlah operasi perbandingan elemen = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah

Yang mana, **ini merupakan kompleksitas waktu untuk kasus terbaik dan kasus terburuk** karena algoritma selection sort tidak bergantung pada terurut atau acaknya data yang diinput.

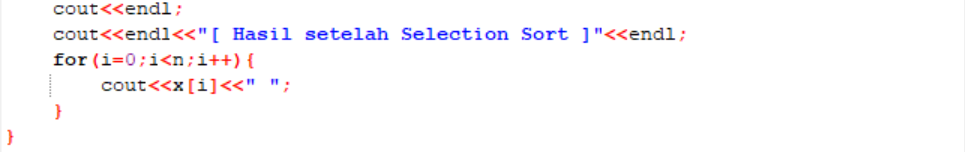
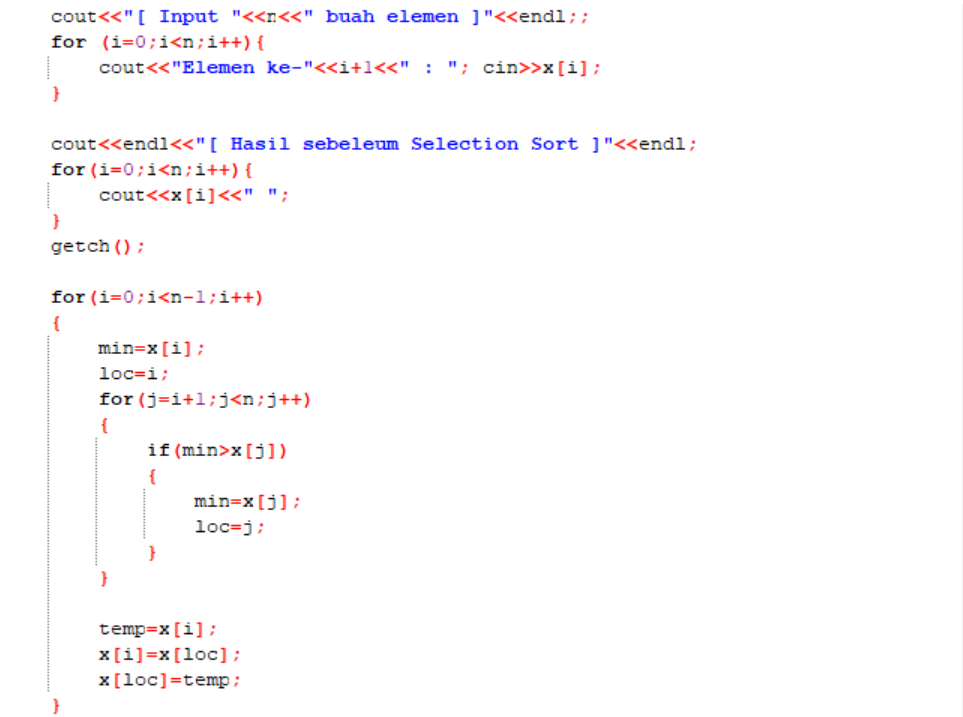
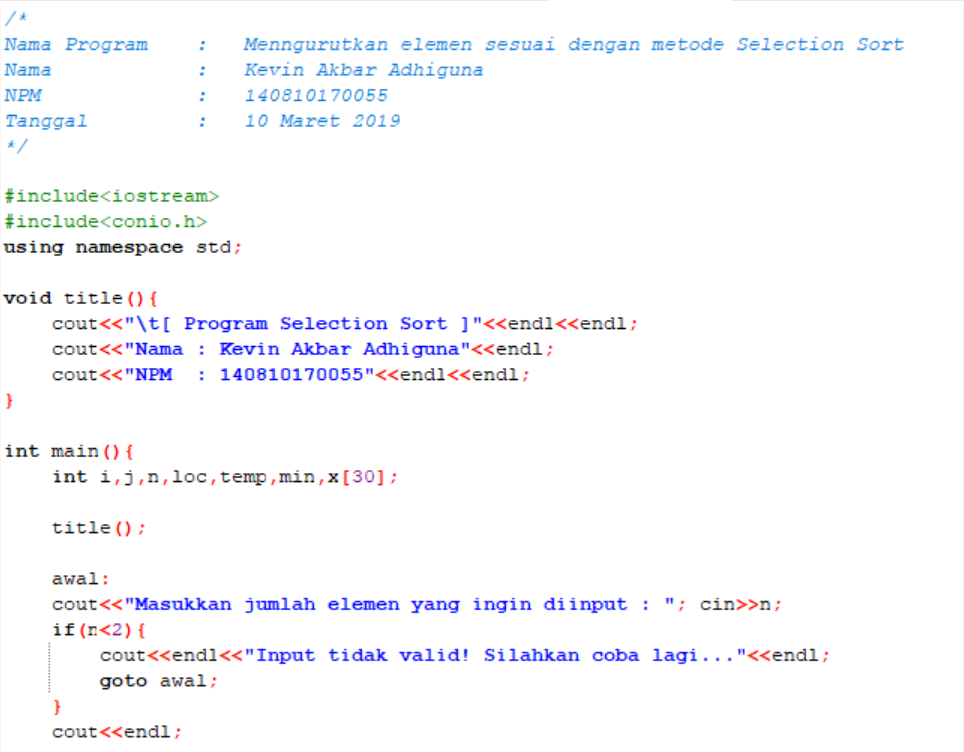
1. Jumlah operasi pertukaran

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

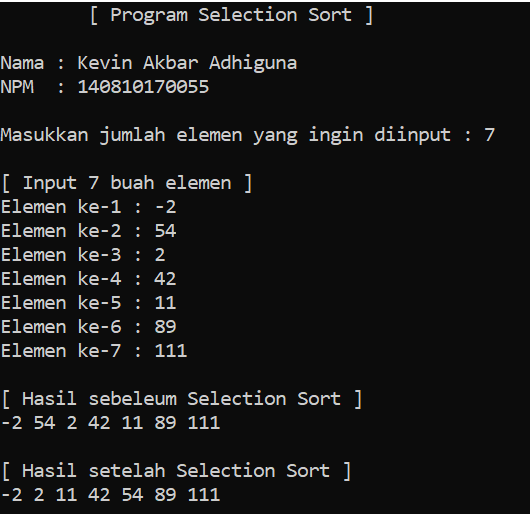
Simpulan :

Jadi, algoritma selection sort membutuhkan buah operasi perbandingan elemen dan buah operasi pertukaran.

**Program**

****

**Hasil Compile & Run program**

****