**Tugas Praktikum Analisis Algoritma**

**Modul 2 (Kompleksitas Waktu Algoritma)**



Disusun oleh :

Muhammad Fahmi Irfananda

(140810160028)

S-1 Teknik Informatika   
Fakultas Matematika & Ilmu Pengetahuan Alam   
Universitas Padjadjaran  
Jalan Raya Bandung - Sumedang Km. 21 Jatinangor 45363

**Studi Kasus 1 : Pencarian Nilai Maksimal**

**Algoritma :**

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

**Program :**

|  |
| --- |
| */\*\**  *\* Mencari nilai maximum dari suatu array*  *\** **@param**{number[]} *grades*  *\*/*  module.exports = (grades) **=>** {  **let** max = grades[0];  for (**let** i = 1; i < grades.length; i++) {  if (grades[i] > max) {  max = grades[i];  }  }  return max;  }; |

|  |
| --- |
| *#!/usr/bin/env node*  'use strict';  **const** maxGrade = require('./max-grade');  **function** main() {  **const** max = maxGrade([1, 0, 3, 5, 10, 30, 7]);  console.log(eval(max === 30));  }  main(); |

|  |
| --- |
| >>> find max in 10 numbers.  [7,6,2,9,10,10,10,10,8,9]  maxGrade: 0.333ms  max: 10  >>> find max in 20 numbers.  [0,10,5,15,19,6,13,12,13,17,16,15,18,13,19,17,19,19,20,19]  maxGrade: 0.030ms  max: 20  >>> find max in 50 numbers.  [44,44,25,32,6,22,18,27,41,26,45,22,38,35,23,45,16,28,49,42,30,26,22,38,37,29,27,49,29,47,37,44,40,49,48,44,38,50,41,46,46,45,45,48,44,45,47,49,50,50]  maxGrade: 0.017ms  max: 50  >>> find max in 100 numbers.  [6,61,94,52,55,91,25,47,15,63,20,49,31,62,26,69,36,30,67,52,71,61,33,89,94,52,37,81,83,62,83,71,57,93,87,84,97,98,55,88,50,54,47,70,55,46,48,68,59,75,93,81,68,80,86,60,57,78,84,90,83,78,89,74,100,78,70,100,91,92,71,89,73,95,92,91,92,97,81,94,82,95,98,84,100,86,98,90,96,89,96,98,92,97,98,98,96,98,98,100]  maxGrade: 0.022ms  max: 100 |

**Perhitungan Kompleksitas :**

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (A[i] > maks).

Kompleksitas waktu CariElemenTerbesar : T(n) = n – 1.

**Studi Kasus 2 : *Sequential Search***

**Algoritma :**

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 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

**Program :**

|  |
| --- |
| */\*\**  *\* Mencari index bilangan suatu array*  *\** **@param**{number[]} *nums*  *\** **@param**{number} *key number to search*  *\** **@returns**{number} *index of found num. if not found it return -1*  *\*/*  module.exports = (nums, key) **=>** {  **let** foundIdx = -1;  for (**let** i = 0; i < nums.length; i++) {  if (key === nums[i]) {  foundIdx = i;  }  }  return foundIdx;  } |

|  |
| --- |
| *#!/usr/bin/env node*  'use strict';  **const** linearSearch = require('./linear-search');  **const** random = require('../utils/random');  **function** main() {  **const** lable = 'linear search';  **let** maxArr = [10, 20, 50, 100];  maxArr.forEach((m) **=>** {  **let** nums = [];  for (**let** i = 0; i < m; i++) {  nums.push(random.getInt(0, m));  }    **let** key = random.getInt(nums[0], nums[nums.length - 1]);  console.log(`\n>>> search in ${m} numbers.`);  console.log(`[${nums}]`);  console.time(lable);  **const** foundIdx = linearSearch(nums, key);  console.timeEnd(lable);  console.log(`found ${key} in index ${foundIdx}`);  });  }  main(); |

|  |
| --- |
| >>> search in 10 numbers.  [7,2,1,6,6,6,3,10,4,7]  linear search: 0.343ms  found 7 in index 9  >>> search in 20 numbers.  [8,12,20,13,10,20,10,12,16,19,6,13,18,2,9,18,9,12,14,5]  linear search: 0.029ms  found 7 in index -1  >>> search in 50 numbers.  [43,37,1,0,22,14,37,43,50,27,26,46,35,40,29,44,28,49,35,48,12,7,15,35,13,44,29,20,22,22,29,1,18,30,42,50,34,4,32,18,24,21,42,15,15,50,2,42,18,45]  linear search: 0.018ms  found 43 in index 7  >>> search in 100 numbers.  [54,83,16,26,80,37,10,74,88,66,10,49,49,84,99,17,64,98,50,75,18,70,50,27,38,0,74,94,57,50,6,91,61,51,35,40,83,10,96,6,38,29,4,16,61,71,22,20,20,99,19,4,35,74,28,22,52,25,99,56,63,11,67,34,74,19,5,85,37,79,80,77,45,78,73,20,41,39,99,96,88,86,7,73,17,84,4,28,52,50,23,66,97,75,15,98,76,35,31,96]  linear search: 0.023ms  found 85 in index 67 |

**Perhitungan Kompleksitas :**

Jumlah operasi perbandingan elemen tabel:

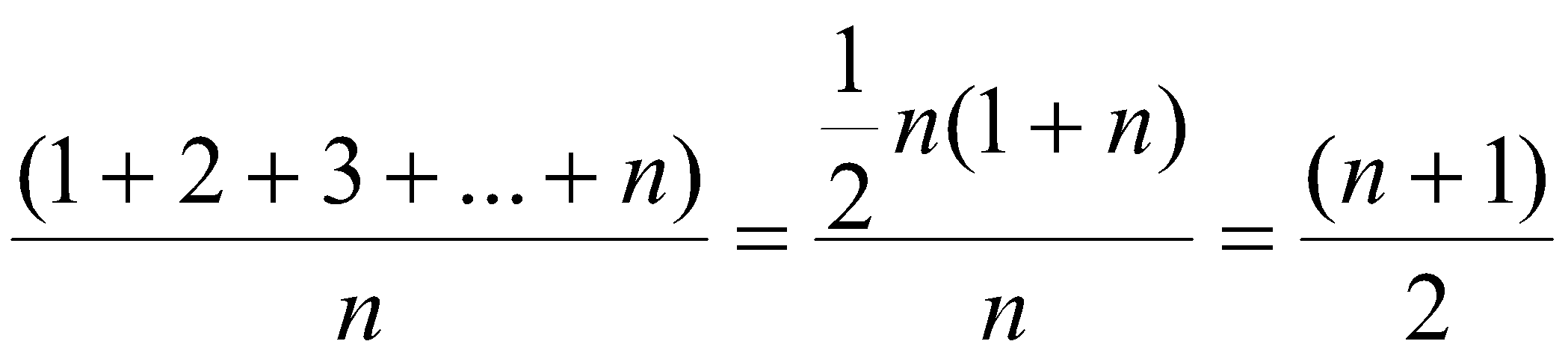
1*. Kasus terbaik*: ini terjadi bila *a*1 = *x.*

*T*min(*n*) =1

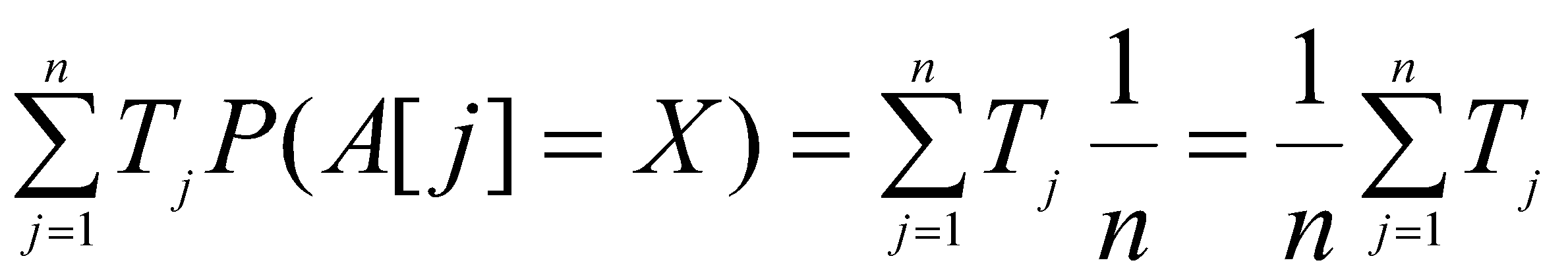
2. *Kasus terburuk*: bila *an* = *x* atau *x* tidak ditemukan.

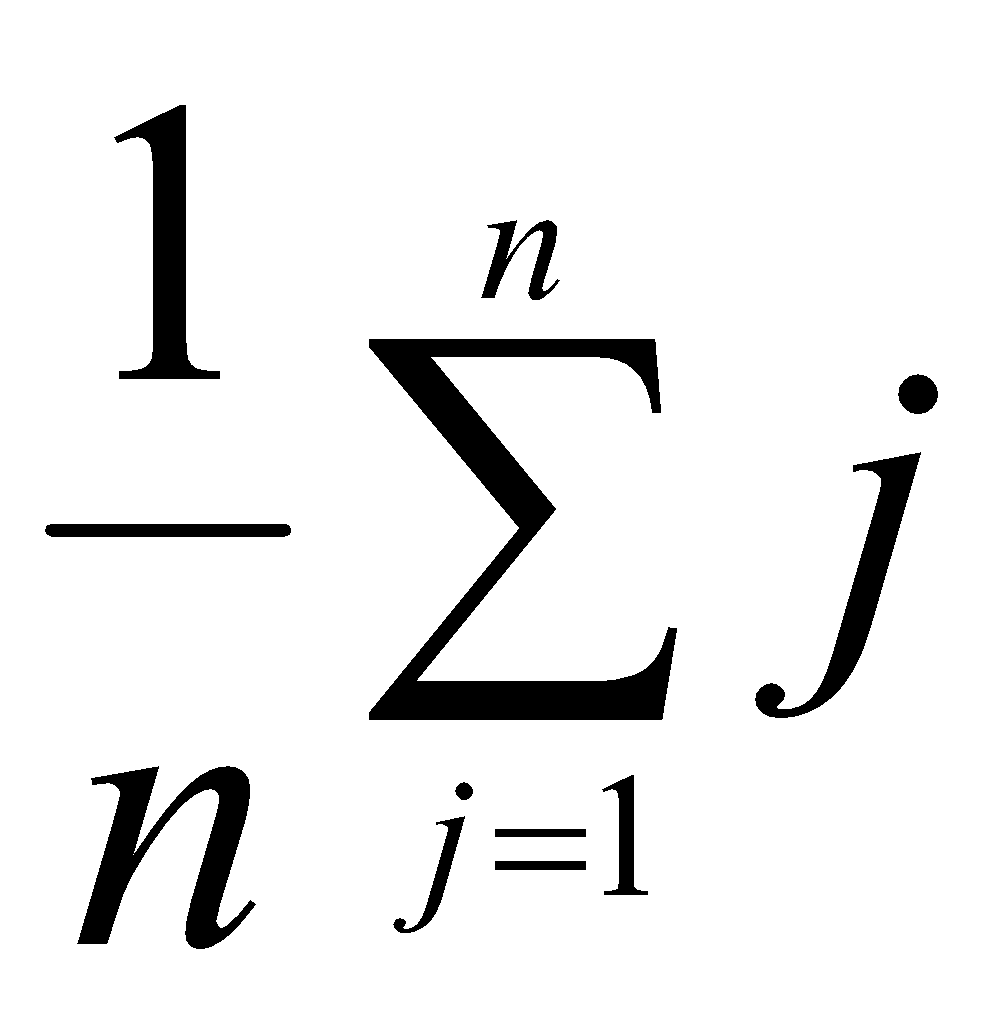
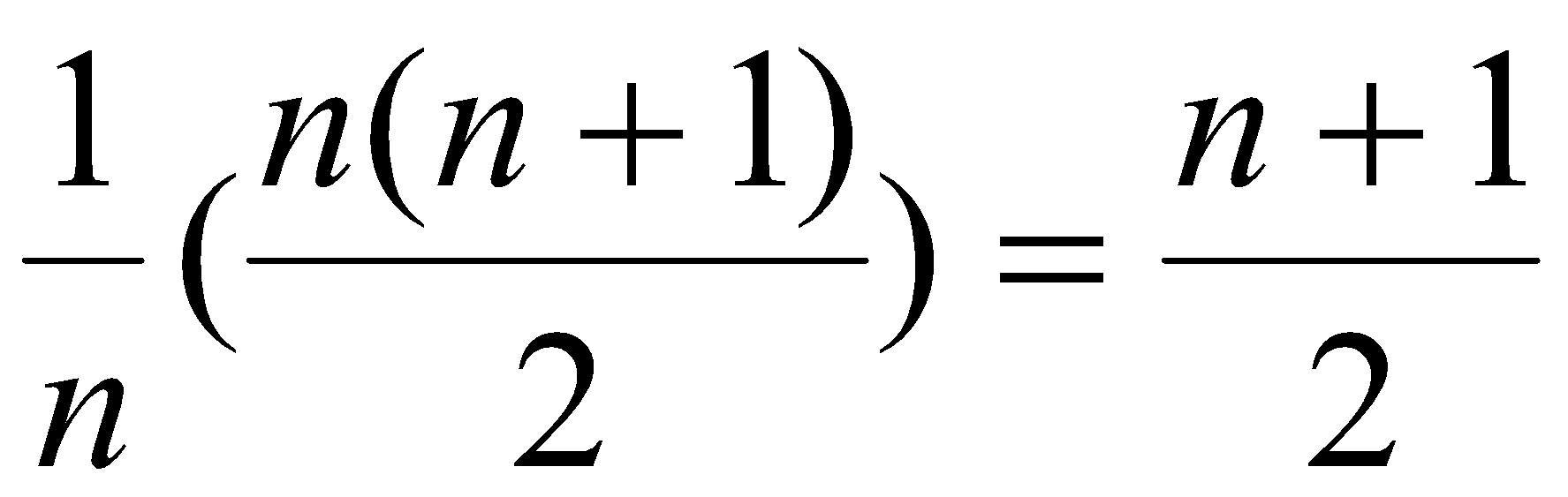
*T*max(*n*) = *n*

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

*T*avg(*n*) = 

Cara lain: asumsikan bahwa *P*(*aj* = *x*) = 1/*n*. Jika *aj* = *x* maka *Tj* yang dibutuhkan adalah *Tj = j.* Jumlah perbandingan elemen larik rata-rata:

*T*avg(*n*) = 

= = 

**Studi Kasus 3 : *Binary Search***

**Algoritma :**

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

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

**Program :**

|  |
| --- |
| */\*\**  *\* Perform Binary search*  *\* Worst case time complexity is O(log n)*  *\* The given array need to be sorted first*  *\**  *\** **@param**{number[]} *nums array of number*  *\** **@param**{number[]} *key key to search*  *\*/*  **const** binarySearch = (nums, key) **=>** {  **let** low = 0;  **let** high = nums.length;  **let** itemIdx = -1;  while(low <= high) {  **let** mid = parseInt((high + low)/2, 10);  if (key === nums[mid]) {  itemIdx = mid;  break;  }  if (nums[mid] < key) {  low = mid + 1;  } else {  high = mid - 1;  }  }  return itemIdx;  }  module.exports = binarySearch; |

\

|  |
| --- |
| *#!/usr/bin/env node*  **const** binarySearch = require('./binary-search');  **const** random = require('../utils/random');  **function** main() {  **const** bsLable = 'binarySearch';  **let** maxArr = [10, 20, 50, 100];  maxArr.forEach((m, idx) **=>** {  **let** nums = [];  for (**let** i = 0; i < m; i++) {  nums.push(random.getInt(0, m));  }    *// key is random between range*  **let** randIdx = random.getInt(0, nums.length);  **let** key = nums[randIdx];  console.log(`\n>>> search ${key} in ${m} numbers.`);  console.log(`[${nums}]`);  console.time(bsLable);  **const** foundIdx = binarySearch(nums, key);  console.timeEnd(bsLable);    console.log(`found ${key} in index ${foundIdx}`);    });  }  main(); |

**Perhitungan Kompleksitas :**

1*. Kasus terbaik*

*T*min(*n*) = 1

2. *Kasus terburuk*:

*T*max (*n*) = 2log *n*

**Studi Kasus 4 : *Insertion Sort***

**Algoritma :**

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

**Program :**

|  |
| --- |
| */\*\**  *\* Insertion sort*  *\** **@param**{number[]} *nums*  *\** **@returns**{number[]} *ascending sorted nums*  *\*/*  module.exports = (nums) **=>** {  for (**let** i = 0; i < nums.length; i++) {  for (**let** j = i; j > 0; j--) {  if (nums[j-1] > nums[j]) {  *// swap nums value*  **let** temp = nums[j];  nums[j] = nums[j-1];  nums[j-1] = temp;  }  }  }  return nums;  }; |

|  |
| --- |
| *#!/usr/bin/env node*  'use strict';  **const** insertionSort = require('./insertion-sort');  **const** random = require('../utils/random');  **function** main() {  **const** lable = 'insertion search';  **let** maxArr = [10, 20, 50, 100];  maxArr.forEach((m) **=>** {  **let** nums = [];  for (**let** i = 0; i < m; i++) {  nums.push(random.getInt(0+i, m));  }    console.log(`\n>>> sort ${m} numbers.`);  console.log(`[${nums}]`);  console.time(lable);  **const** sorted = insertionSort(nums);  console.timeEnd(lable);  console.log(`[${sorted}]`);  });  }  main(); |

|  |
| --- |
| >>> sort 10 numbers.  [5,9,10,5,10,9,8,10,10,9]  insertion search: 0.427ms  [5,5,8,9,9,9,10,10,10,10]  >>> sort 20 numbers.  [17,7,14,14,18,8,12,13,18,9,16,19,15,13,16,17,19,20,18,19]  insertion search: 0.062ms  [7,8,9,12,13,13,14,14,15,16,16,17,17,18,18,18,19,19,19,20]  >>> sort 50 numbers.  [31,28,41,35,45,6,20,14,9,49,43,50,12,44,45,16,29,39,34,42,46,27,24,27,32,25,40,28,34,49,37,49,41,41,39,36,46,48,41,47,50,41,43,44,47,48,48,47,49,49]  insertion search: 0.240ms  [6,9,12,14,16,20,24,25,27,27,28,28,29,31,32,34,34,35,36,37,39,39,40,41,41,41,41,41,42,43,43,44,44,45,45,46,46,47,47,47,48,48,48,49,49,49,49,49,50,50]  >>> sort 100 numbers.  [48,44,21,18,19,87,90,53,39,33,73,76,34,67,94,44,89,69,28,20,41,60,35,24,83,81,84,90,89,67,64,80,64,82,46,63,100,75,55,52,62,74,65,80,74,75,78,78,73,62,71,53,79,66,69,60,94,57,94,59,67,66,71,76,87,98,93,79,84,69,92,89,72,100,100,92,91,87,80,92,100,88,90,85,84,92,91,96,100,95,91,93,100,96,100,100,97,99,100,99]  insertion search: 0.936ms  [18,19,20,21,24,28,33,34,35,39,41,44,44,46,48,52,53,53,55,57,59,60,60,62,62,63,64,64,65,66,66,67,67,67,69,69,69,71,71,72,73,73,74,74,75,75,76,76,78,78,79,79,80,80,80,81,82,83,84,84,84,85,87,87,87,88,89,89,89,90,90,90,91,91,91,92,92,92,92,93,93,94,94,94,95,96,96,97,98,99,99,100,100,100,100,100,100,100,100,100] |

**Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma.**

**Perhitungan Kompleksitas :**

* Best Case
* Worst Case
* Average Case

**Studi Kasus 5 : *Selection Sort***

**Algoritma :**

procedure SelectionSort(input/outputx1, 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 xj > ximaks then

imaks <- j

endif

endfor

{pertukarkan ximaks dengan xi}

temp <- xi

xi <- ximaks

ximaks <- temp

endfor

**Program :**

|  |
| --- |
| */\*\**  *\** **@param**{number[]} *nums*  *\** **@returns**{number[]} *ascending sorted nums*  *\*/*  module.exports = (nums) **=>** {  **let** size = nums.length;  for (**let** i = 0; i < size - 1; i++) {  *// find min number*  **let** minIdx = i;  for (**let** j = i+1; j < size; j++) {  if (nums[j] < nums[minIdx]) {  minIdx = j;  }  }  if (minIdx !== i) {  **let** temp = nums[i];  nums[i] = nums[minIdx];  nums[minIdx] = temp;  }  }  return nums;  } |
| *#!/usr/bin/env node*  'use strict';  **const** selectSort = require('./selection-sort');  **const** random = require('../utils/random');  **function** main() {  **let** res = selectSort([1, 0, 3, 5, 10, 30, 7]);  console.log(res);  res = selectSort([100, 45, 28, 3, 2, 1, 0]);  console.log(res);  res = selectSort([7, 100, 10, 20, 50, 120, 0]);  console.log(res);  **const** lable = 'insertion search';  **let** maxArr = [10, 20, 50, 100];  maxArr.forEach((m) **=>** {  **let** nums = [];  for (**let** i = 0; i < m; i++) {  nums.push(random.getInt(0+i, m));  }    console.log(`\n>>> sort ${m} numbers.`);  console.log(`[${nums}]`);  console.time(lable);  **const** sorted = selectSort(nums);  console.timeEnd(lable);  console.log(`[${sorted}]`);  });  }  main(); |
| [ 0, 1, 3, 5, 7, 10, 30 ]  [ 0, 1, 2, 3, 28, 45, 100 ]  [ 0, 7, 10, 20, 50, 100, 120 ]  >>> sort 10 numbers.  [8,7,7,7,10,8,10,10,8,10]  insertion search: 0.263ms  [7,7,7,8,8,8,10,10,10,10]  >>> sort 20 numbers.  [16,11,18,13,12,20,9,15,15,13,20,14,15,20,16,18,19,18,18,20]  insertion search: 0.057ms  [9,11,12,13,13,14,15,15,15,16,16,18,18,18,18,19,20,20,20,20]  >>> sort 50 numbers.  [48,13,23,5,9,17,42,27,49,14,45,47,25,24,31,16,29,38,20,49,45,41,49,32,38,49,40,30,47,30,40,50,50,39,45,46,47,49,47,42,45,45,48,46,44,48,50,48,48,50]  insertion search: 0.169ms  [5,9,13,14,16,17,20,23,24,25,27,29,30,30,31,32,38,38,39,40,40,41,42,42,44,45,45,45,45,45,46,46,47,47,47,47,48,48,48,48,48,49,49,49,49,49,50,50,50,50]  >>> sort 100 numbers.  [47,56,12,19,8,14,84,34,66,27,81,60,38,84,42,65,90,20,69,81,94,35,91,53,59,70,61,80,44,32,85,72,81,60,78,96,80,73,66,84,86,59,47,77,46,82,90,77,68,63,55,63,68,57,97,60,81,67,72,91,60,61,65,98,69,89,96,94,70,91,77,77,100,85,83,91,90,82,87,83,81,86,84,83,99,95,90,99,94,91,96,96,94,98,100,98,100,99,98,99]  insertion search: 0.624ms  [8,12,14,19,20,27,32,34,35,38,42,44,46,47,47,53,55,56,57,59,59,60,60,60,60,61,61,63,63,65,65,66,66,67,68,68,69,69,70,70,72,72,73,77,77,77,77,78,80,80,81,81,81,81,81,82,82,83,83,83,84,84,84,84,85,85,86,86,87,89,90,90,90,90,91,91,91,91,91,94,94,94,94,95,96,96,96,96,97,98,98,98,98,99,99,99,99,100,100,100] |

**Perhitungan Kompleksitas :**

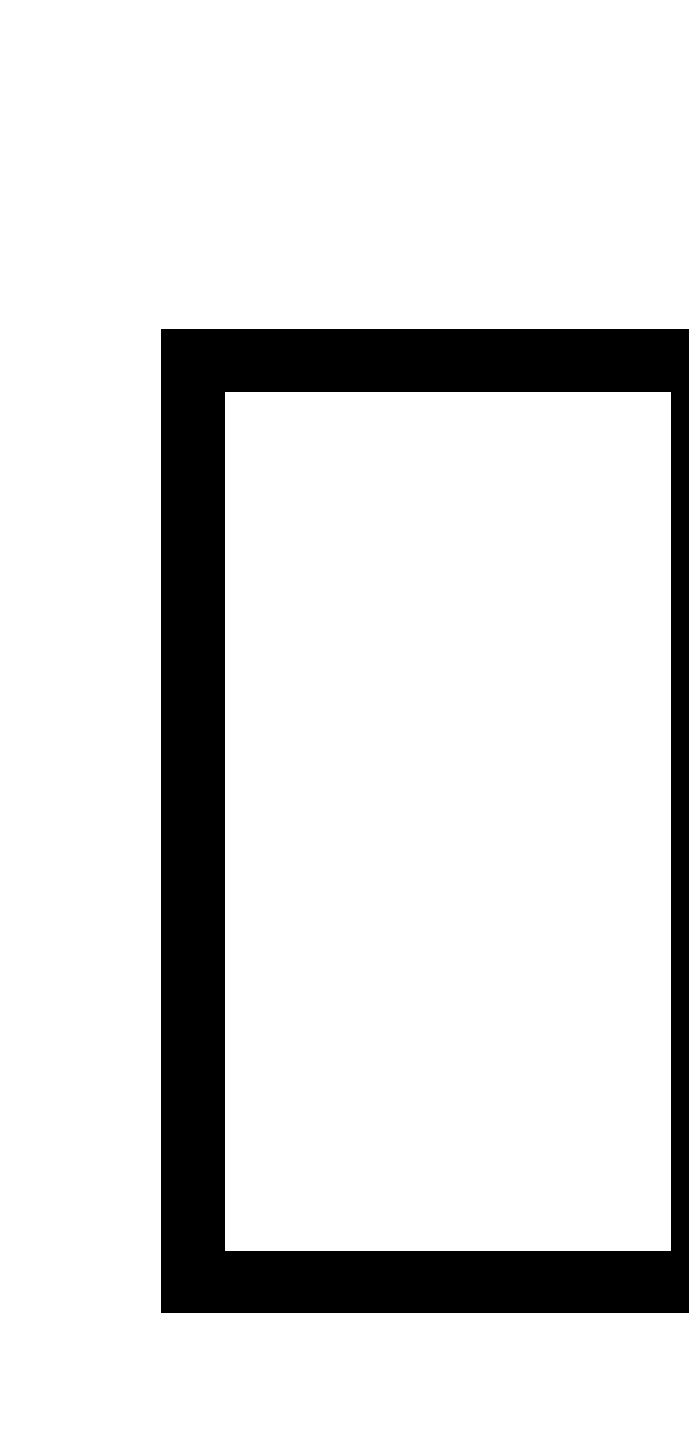
(i) Jumlah operasi perbandingan elemen

Untuk setiap *pass* ke-*i*,

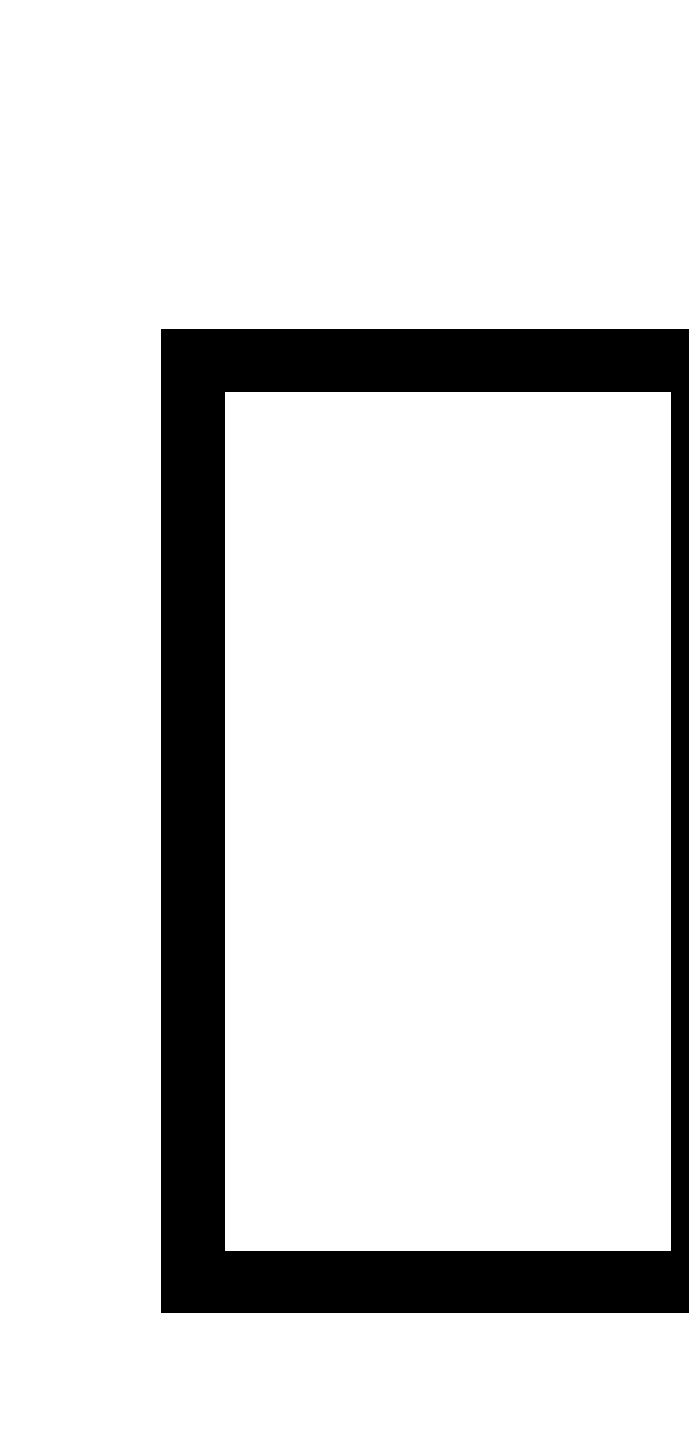
*i* = 1 → jumlah perbandingan = *n* – 1

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

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

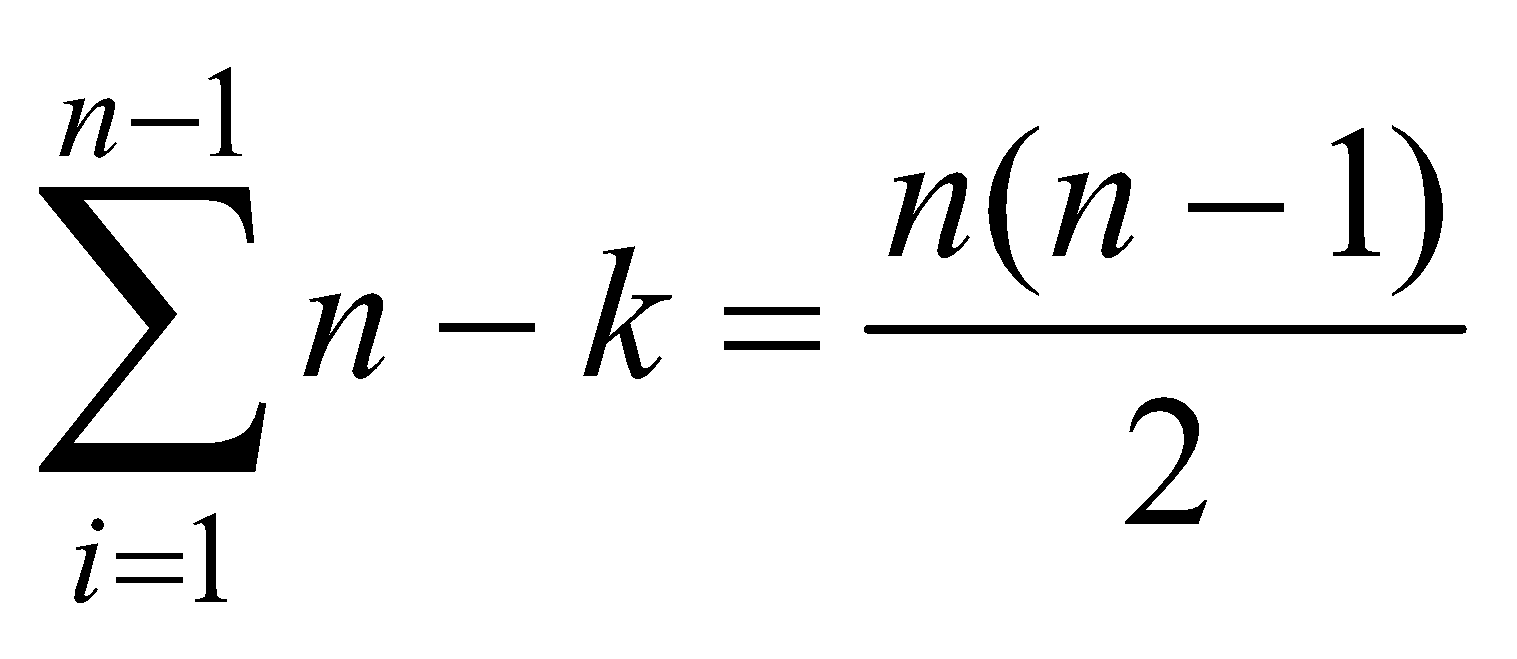


*i* = *k* → jumlah perbandingan = *n* – *k*



*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.

(ii) 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.