**LAPORAN UNTUK ANALISIS REKURSIF**

DAFTAR ISI

[Analisis Kompleksitas Algoritma Binary Search 2](#_Toc185833876)

[A. Parameter yang merupakan ukuran input: 2](#_Toc185833877)

[Operasi dasar algoritma: 2](#_Toc185833878)

[A. Operasi Perbandingan Utama 2](#_Toc185833879)

[B. Operasi Pembagian Range 2](#_Toc185833880)

[Kasus Terbaik, Terburuk, dan Rata-Rata 3](#_Toc185833881)

[A. Kasus terbaik: 3](#_Toc185833882)

[A. Kasus terburuk: 3](#_Toc185833883)

[B. Keunggulan menggunakan Kompleksitas O(log n): 4](#_Toc185833884)

[C. Kasus Rata Rata 4](#_Toc185833885)

[Relasi rekuren dan kondisi awal: 6](#_Toc185833886)

[Penyelesaian Persamaan Rekurensi 6](#_Toc185833887)

[A. Ekspansi Rekursif 6](#_Toc185833888)

[B. Pola Rekursi 6](#_Toc185833889)

[C. Perhitungan Total Operasi 7](#_Toc185833890)

[D. Perhitungan Total Operasi 7](#_Toc185833891)

[Kesimpulan 7](#_Toc185833892)

# Analisis Kompleksitas Algoritma Binary Search

## Parameter yang merupakan ukuran input:

Parameter utama yang menentukan ukuran input (n) dalam algoritma binary search adalah jumlah elemen dalam array yang akan dicari.

Input size (n) = high - low + 1

Andaikan kita punya data array: [10, 20, 30, 40, 50]

low = 0 (first index)

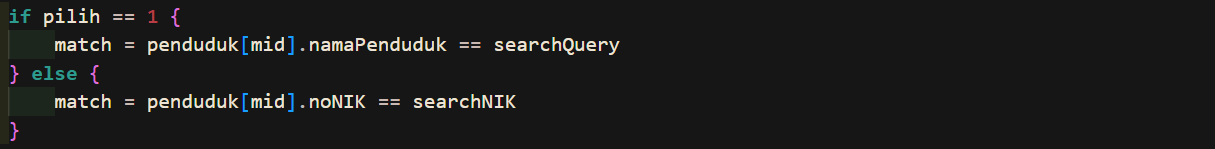
high = 4 (last index)

Input size (n) = high - low + 1 = 4 - 0 + 1 = 5 elemen

* Range pencarian: [low...high]
* Setiap iterasi membagi range pencarian menjadi 2 bagian

# Operasi dasar algoritma:

## Operasi Perbandingan Utama



-> Operasi ini membutuhkan waktu konstan O(1)

## B. Operasi Pembagian Range



# Kasus Terbaik, Terburuk, dan Rata-Rata

## Kasus terbaik:

* Data ditemukan di tengah array pada pencarian pertama.
* Kompleksitas waktu: O(1) atau jumlah operasi = 1 kali
* Contoh:

A screen shot of a computer

Description automatically generated

## Kasus terburuk:

- Data tidak ditemukan, dan element berada di posisi paling kiri/kanan.  
- Kompleksitas: O(log n) dan jumlah operasi = log₂(n)

**Contoh:**

A screenshot of a computer

Description automatically generated

**Penjelasan:**

Setiap level melakukan:

1. Perhitungan mid - O(1)
2. Perbandingan nilai - O(1)
3. Update low atau high - O(1)

Total per level = O(1) Total level = log₂(n) Total kompleksitas = O(1) × log₂(n) = O(log n)

## Keunggulan menggunakan Kompleksitas O(log n):

1. Scalability yang sangat baik:

* Double ukuran input → hanya +1 operasi
* 10x ukuran input → +3.32 operasi
* 100x ukuran input → +6.64 operasi

2. Efisiensi:

* Untuk n = 1 miliar
* Linear Search: 1 miliar operasi
* Binary Search: ≈ 30 operasi

Grafik Pertumbuhan Kompleksitas:

A screenshot of a computer

Description automatically generated

## Kasus Rata Rata

* Rata-rata jumlah perbandingan = log₂(n)
* Kompleksitas = O(log n)
* Contoh:

A screen shot of a computer

Description automatically generated

Mencari NIK 1001:

Level 1: check 1004 (mid=3)

Level 2: check 1002 (mid=1)

Level 3: check 1001 (mid=0)

Total: 3 perbandingan

Mencari NIK 1004:

Level 1: check 1004 (mid=3)

Total: 1 perbandingan

Mencari NIK 1008:

Level 1: check 1004 (mid=3)

Level 2: check 1006 (mid=5)

Level 3: check 1008 (mid=7)

Total: 3 perbandingan

Untuk n = 8 elemen:

* + Best case: 1 perbandingan
  + Worst case: 3 perbandingan (log₂(8) = 3)
  + Average case:

Jumlah total perbandingan untuk semua kemungkinan dibagi jumlah elemen.

Misalnya:

NIK 1001: 3 perbandingan

NIK 1002: 3 perbandingan

NIK 1003: 2 perbandingan

NIK 1004: 1 perbandingan

NIK 1005: 2 perbandingan

NIK 1006: 3 perbandingan

NIK 1007: 3 perbandingan

NIK 1008: 3 perbandingan

Rata-rata = (3+3+2+1+2+3+3+3)/8 = 2.5 atau log₂(8)

Kesimpulan nya. rata-rata jumlah perbandingan ≈ log₂(n) Karena:

* + Setiap level membagi data menjadi 2
  + Jumlah level = log₂(n)
  + Di setiap level ada 1 perbandingan

# Relasi rekuren dan kondisi awal:

**A. Kondisi Dasar**

Relasi rekuren untuk jumlah operasi dasar:  
Kondisi awal:

***C(1) = O(1) atau T(1) = 1.***

**B. Relasi Rekurensi**

***T(n) = T(n/2) + 1, untuk n > 1***

Dimana:

* T(n/2): Waktu untuk menyelesaikan subproblem setengah ukuran
* 1: Waktu untuk operasi perbandingan di setiap level rekursi

# Penyelesaian Persamaan Rekurensi

## A. Ekspansi Rekursif

T(n) = T(n/2) + 1

= [T(n/4) + 1] + 1

= [T(n/8) + 1] + 1 + 1

= [T(n/16) + 1] + 1 + 1 + 1

## B. Pola Rekursi

Misalkan untuk n = 16:

Level 0: n = 16 → 1 operasi

Level 1: n = 8 → 1 operasi

Level 2: n = 4 → 1 operasi

Level 3: n = 2 → 1 operasi

Level 4: n = 1 → 1 operasi

Total level = log₂(16) = 4

## C. Perhitungan Total Operasi

***T(n) = T(1) + jumlah level = 1 + log₂(n)***

## D. Perhitungan Total Operasi

***T(n) = 1 + log₂(n)***

**Penjelasan:**

A. Untuk setiap level k:

* Ukuran problem = n/2ᵏ
* Jumlah operasi per level = 1

B. Total level sampai mencapai base case:

* n/2ᵏ = 1
* n = 2ᵏ
* k = log₂(n)

# Kesimpulan

* Kompleksitas waktu: O(log n)
* Space complexity: O(log n) karena menggunakan rekursi
* Algoritma ini sangat efisien untuk pencarian pada data terurut
* Jauh lebih baik dibanding linear search yang memiliki kompleksitas O(n)