

**LAPORAN PRAKTIKUM  
ALGORITMA & STRUKTUR DATA  
MODUL 6**



**Searching**

**Oleh:**

**Rizki Adhitiya Maulana**

**NIM. 2410817110014**

**PROGRAM STUDI TEKNOLOGI INFORMASI  
FAKULTAS TEKNIK  
UNIVERSITAS LAMBUNG MANGKURAT  
JUNI 2025**

**LEMBAR PENGESAHAN**  
**LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA**  
**MODUL 6**

Laporan Praktikum Algoritma & Struktur Data Modul 6 : Searching ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Rizki Adhitiya Maulana  
NIM : 2410817110014

Menyetujui,  
Asisten Praktikum

Mengetahui,  
Dosen Penanggung Jawab Praktikum

Muhammad Fauzan Ahsani  
NIM. 2310817310009

Muti'a Maulida, S.Kom., M.TI.  
NIP. 198810272019032013

## DAFTAR ISI

LEMBAR PENGESAHAN .....	i
DAFTAR ISI.....	ii
DAFTAR TABEL .....	iii
DAFTAR GAMBAR .....	iv
SOAL 1 .....	1
A    Output Program.....	3
B    Output Program.....	12
C    Pembahasan.....	14
TAUTAN GITHUB .....	21

## DAFTAR TABEL

## DAFTAR GAMBAR

Gambar 1 Contoh Source Code Sequential Searching.....	1
Gambar 2 Contoh Source Code Binary Searching.....	2
Gambar 3 Tampilan Menu Searching .....	12
Gambar 4 Tampilan Sequential Searching Apabila Angka Ditemukan.....	12
Gambar 5 Tampilan Sequential Searching Apabila Angka Tidak Ditemukan .....	12
Gambar 6 Tampilan Binary Searching Apabila Angka Ditemukan.....	13
Gambar 7 Tampilan Binary Searching Apabila Angka Tidak Ditemukan .....	13
Gambar 8 Tampilan Penjelasan Perbedaan Dua Metode Search .....	13
Gambar 9 Ilustrasi Sequential Search .....	16
Gambar 10 Ilustrasi Binary Search .....	18

## SOAL 1

Ketikkan source code berikut pada program IDE bahasa pemrograman C++  
(Gabungkan 2 code berikut menjadi 1 file (Menu):

- Sequential Searching

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4
5  using namespace std;
6
7  int random(int bil)
8  {
9      int jumlah = rand() % bil;
10     return jumlah;
11 }
12
13 void randomize()
14 {
15     srand(time(NULL));
16 }
17
18 void clrscr()
19 {
20     system("cls");
21 }
22
23 int main()
24 {
25     clrscr();
26     int data[100];
27     int cari = 20;
28     int counter = 0;
29     int flag = 0;
30     int save;
31     randomize();
32     printf("generating 100 number . . .\n");
33     for (int i = 0; i < 100; i++)
34     {
35         data[i] = random(100) + 1;
36         printf("%d ", data[i]);
37     }
38     printf("\ndone.\n");
39
40     for (int i = 0; i < 100; i++)
41     {
42         if (data[i] == cari)
43         {
44             counter++;
45             flag = 1;
46             save = i;
47         }
48     },
49
50     if (flag == 1)
51     {
52         printf("Data ada, sebanyak %d!\n", counter);
53         printf("pada indeks ke-%d", save);
54     }
55     else
56     {
57         printf("Data tidak ada!\n");
58     }
59 }
60
```

Gambar 1 Contoh Source Code Sequential Searching

- Binary Searching

```

1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int n, kiri, kanan, tengah, temp, key;
7      bool ketemu = false;
8
9      cout << "Masukan jumlah data? ";
10     cin >> n;
11     int angka[n];
12     for (int i = 0; i < n; i++)
13     {
14         cout << "Angka ke - [" << i << "] : ";
15         cin >> angka[i];
16     }
17
18     for (int i = 0; i < n; i++)
19     {
20         for (int j = 0; j < n - 1; j++)
21         {
22             if (angka[j] > angka[j + 1])
23             {
24                 temp = angka[j];
25                 angka[j] = angka[j + 1];
26                 angka[j + 1] = temp;
27             }
28         }
29     }
30     cout << "-----\n";
31     cout << "Data yang telah diurutkan adalah:\n";
32     for (int i = 0; i < n; i++)
33     {
34         cout << angka[i] << " ";
35     }
36     cout << "\n-----\n";
37     cout << "Masukan angka yang dicari: ";
38     cin >> key;
39
40     kiri = 0;
41     kanan = n - 1;
42     while (kiri <= kanan)
43     {
44         tengah = (kiri + kanan) / 2;
45         if (key == angka[tengah])
46         {
47             ketemu = true;
48             break;
49         }
50         else if (key < angka[tengah])
51         {
52             kanan = tengah - 1;
53         }
54         else
55         {
56             kiri = tengah + 1;
57         }
58     }
59     if (ketemu == true)
60     {
61         cout << "Angka ditemukan! ";
62     }
63     else
64         cout << "Angka tidak ditemukan!";
65     return 0;
66 }
67

```

Gambar 2 Contoh Source Code Binary Searching

- Tampilan Menu Program

```
Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih :
```

Jelaskan perbedaan Sequential Searching dan Binary Searching beserta kelebihan dan kekurangan masing-masing?

### A Output Program

```
1 #include <iostream>
2 #include <conio.h>
3 #include <random>
4 #include <vector>
5 #include <algorithm>
6
7 using namespace std;
8
9 void sequentialSearch(const vector<int> &nums,
10 int target)
11 {
12     vector<int> indices;
13
14     for (auto i = 0; i < nums.size(); i++)
15     {
16         if (nums[i] == target)
17             indices.push_back(i);
18     }
19
20     if (indices.empty()) cout << "angka " << target
21 << " tidak ditemukan pada array" << endl;
```



```
18     else
19     {
20         cout << "Angka " << target << " terdapat
pada array sebanyak " << indices.size() << " Kali"
<< endl;
21         cout << "Angka " << target << " ditemukan
pada indeks: ";
22         for (auto i = 0; i < indices.size(); i++)
23         {
24             cout << indices[i];
25             if (i != indices.size() - 1) cout <<
", ";
26         }
27         cout << "." << endl;
28     }
29 }
30
31 void binarySearch(const vector<int> &nums, int
target)
32 {
33     int high = nums.size() - 1;
34     int low = 0;
35     int index = -1;
36
37     while (low <= high)
38     {
39         int mid = low + (high - low) / 2;
40
41         if (nums[mid] == target)
42         {
```

```

43         index = mid;
44         break;
45     }
46     if (nums[mid] < target) low = mid + 1;
47     else high = mid - 1;
48 }
49
50     if (index == -1) cout << "angka " << target <<
" tidak ditemukan pada array" << endl;
51     else cout << "angka " << target << " ditemukan
pada indeks ke " << index << endl;
52 }
53
54 void clearScreen()
55 {
56     system("cls");
57 }
58
59 void explain()
60 {
61     cout << "\nPERBEDAAN SEQUENTIAL/LINEAR SEARCH
DENGAN BINARY SEARCH" << endl;
62     cout << "SEQUENTIAL SEARCH: " << endl;
63     cout << "> Melakukan pengecekan pada array
melalui traversal indeks." << endl;
64     cout << "> Jika elemen pada array yang dicari
sama dengan elemen target, maka cetak nilai
indeks." << endl;

```

```
65     cout << "> Kompleksitas Waktu:  $O(n)$ , karena
penggunaan fungsi loop for untuk pencarian target
secara traversal." << endl;
66     cout << "> Kompleksitas Ruang:  $O(1)$ , karena
penggunaan memori konstan." << endl;
67     cout << "LINEAR SEARCH tidak memiliki syarat
tertentu." << endl;
68     cout << "LINEAR SEARCH dapat diterapkan pada:
" << endl;
69     cout << "1. Data yang Tidak Terurut" << endl;
70     cout << "2. Data berukuran kecil" << endl;
71     cout << "3. Pencarian Node Pada Linked List"
<< endl;
72     cout << endl;
73     cout << "BINARY SEARCH: " << endl;
74     cout << "> Membagi array menjadi dua bagian
melalui indeks tengah mid." << endl;
75     cout << "> Bandingkan elemen tengah dengan
elemen target." << endl;
76     cout << "> Jika elemen tengah sama dengan
elemen target, elemen pada array sudah ditemukan."
<< endl;
77     cout << "> Jika elemen tengah kurang dari
elemen target, cari di bagian kanan array." <<
endl;
78     cout << "> Jika elemen tengah lebih dari elemen
target, cari di bagian kiri array." << endl;
79     cout << "> Ulangi kedua tahap di atas sehingga
elemen target ditemukan." << endl;
```

```

80     cout << "> Kompleksitas Waktu: O(log n), karena
    pembagian interval waktu pencarian." << endl;
81     cout << "> Kompleksitas Ruang: O(1), karena
    penggunaan memori konstan." << endl;
82     cout << "SYARAT BINARY SEARCH: array harus
    tersortir terlebih dahulu." << endl;
83     cout << "BINARY SEARCH dapat diterapkan pada:
    " << endl;
84     cout << "1. Machine Learning" << endl;
85     cout << "2. Computer Graphics (algoritma untuk
    ray tracing atau texture mapping)" << endl;
86     cout << "3. Pencarian data pada dataset besar"
    << endl;
87 }
88
89 int main()
90 {
91     int opt, target;
92     do
93     {
94         cout << "Pilih menu" << endl;
95         cout << "1. Sequential Searching" << endl;
96         cout << "2. Binary Searching" << endl;
97         cout << "3. Jelaskan Perbedaan Sequential
    Searching dan Binary Searching!" << endl;
98         cout << "4. Exit" << endl;
99         cout << "Pilih: ";
100        cin >> opt;
101
102        switch (opt)

```

```

103         {
104             case 1:
105                 {
106                     vector<int> nums (100);
107                     mt19937_64 rng(random_device{}());
108                     uniform_int_distribution<int>
109                     dist(1, 50);
110                     for (auto &val: nums)
111                     {
112                         val = dist(rng);
113                     }
114
115                     cout << "Generating 100 numbers..."
116                     << endl;
117                     for (auto i = 0; i < nums.size();
118                     i++)
119                     {
120                         cout << nums[i] << "[" << i <<
121                         "]" << " ";
122                     }
123                     cout << endl;
124                     cout << "Masukkan angka yang ingin
125                     dicari: "; cin >> target;
126
127                     sequentialSearch(nums, target);
128                     break;
129                 }
130             case 2:

```

```

128         {
129             int size;
130             cout << "Masukkan ukuran vector:
";
131             cin >> size;
132             if (size < 1)
133             {
134                 cout << "Error: Mohon masukkan
bilangan di atas 0" << endl;
135                 break;
136             }
137
138             vector<int> nums(size);
139             mt19937_64 rng(random_device{}());
140             uniform_int_distribution<int>
dist(1, 100);
141
142             for (auto &val: nums)
143             {
144                 val = (dist(rng));
145             }
146
147             sort(nums.begin(), nums.end());
148
149             cout << "Generating " << size <<
" numbers..." << endl;
150             for (auto i = 0; i < nums.size();
i++)
151             {

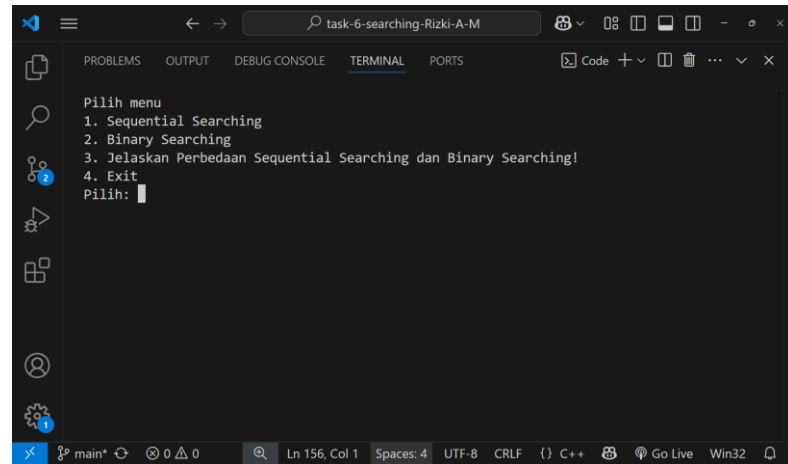
```

```
152         cout << nums[i] << "[" << i <<
    "]" << " ";
153     }
154     cout << endl;
155     cout << "Masukkan angka yang ingin
    dicari: "; cin >> target;
156
157     binarySearch(nums, target);
158     break;
159 }
160
161     case 3:
162         explain();
163         break;
164
165     case 4:
166         cout << "\nTERIMA KASIH\n";
167         cout << "Programme was made by
    Rizki Adhitiya Maulana (2410817110014)" << endl;
168         break;
169
170     default:
171         cout << "Opsi tidak terdefinisi,
    mohon masukkan ulang opsi" << endl;
172         break;
173 }
174
175     if (opt != 4)
176     {
```

177	cout << "\nTekan sembarang tombol untuk melanjutkan...";
178	getch();
179	clearScreen();
180	}
181	
182	}
183	while (opt != 4);
184	
185	return 0;
186	}



## B Output Program

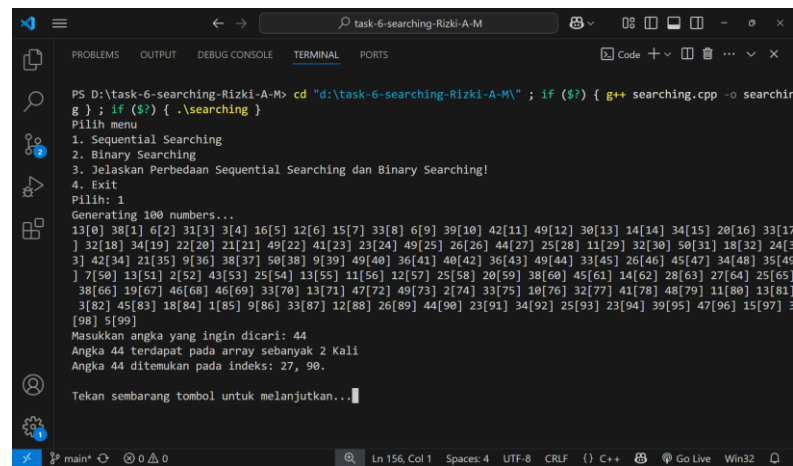


```

task-6-searching-Rizki-A-M
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 

```

Gambar 3 Tampilan Menu Searching

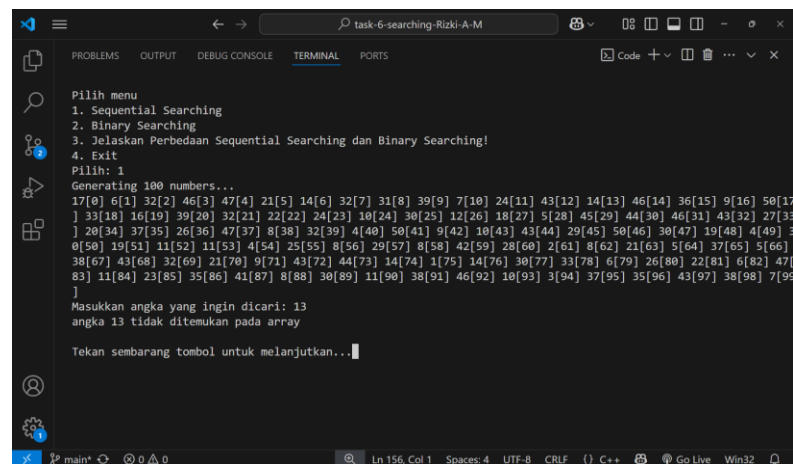


```

task-6-searching-Rizki-A-M
PS D:\task-6-searching-Rizki-A-M> cd "d:\task-6-searching-Rizki-A-M\" ; if ($?) { g++ searching.cpp -o searching
g } ; if ($?) { .\searching }
Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 1
Generating 100 numbers...
13[0] 38[1] 6[2] 31[3] 3[4] 16[5] 12[6] 15[7] 33[8] 6[9] 39[10] 42[11] 49[12] 30[13] 14[14] 34[15] 20[16] 33[17]
] 32[18] 34[19] 22[20] 21[21] 49[22] 41[23] 23[24] 49[25] 26[26] 44[27] 25[28] 11[29] 32[30] 50[31] 18[32] 24[33]
] 42[34] 21[35] 9[36] 38[37] 50[38] 9[39] 49[40] 36[41] 40[42] 36[43] 49[44] 33[45] 26[46] 45[47] 34[48] 35[49]
] 7[50] 13[51] 2[52] 43[53] 25[54] 13[55] 11[56] 12[57] 25[58] 20[59] 38[60] 45[61] 14[62] 28[63] 27[64] 25[65]
] 38[66] 19[67] 46[68] 46[69] 33[70] 13[71] 47[72] 49[73] 2[74] 33[75] 10[76] 32[77] 41[78] 48[79] 11[80] 13[81]
] 3[82] 45[83] 18[84] 1[85] 9[86] 33[87] 12[88] 26[89] 44[90] 23[91] 34[92] 25[93] 23[94] 39[95] 47[96] 15[97] 3
] 98[98] 5[99]
Masukkan angka yang ingin dicari: 44
Angka 44 terdapat pada array sebanyak 2 Kali
Angka 44 ditemukan pada indeks: 27, 90.
Tekan sembarang tombol untuk melanjutkan...

```

Gambar 4 Tampilan Sequential Searching Apabila Angka Ditemukan

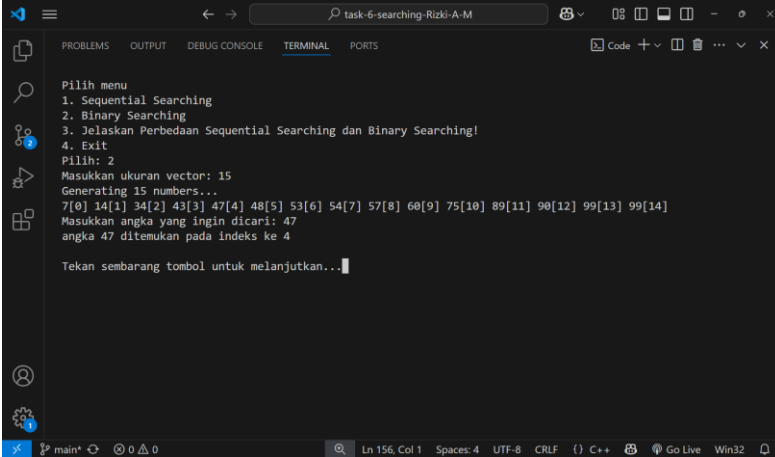


```

task-6-searching-Rizki-A-M
Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 1
Generating 100 numbers...
17[0] 6[1] 32[2] 46[3] 47[4] 21[5] 14[6] 32[7] 31[8] 39[9] 7[10] 24[11] 43[12] 14[13] 46[14] 36[15] 9[16] 50[17]
] 33[18] 16[19] 39[20] 32[21] 22[22] 24[23] 10[24] 30[25] 12[26] 18[27] 5[28] 45[29] 44[30] 46[31] 43[32] 27[33]
] 20[34] 37[35] 26[36] 47[37] 8[38] 32[39] 4[40] 50[41] 9[42] 10[43] 43[44] 29[45] 50[46] 30[47] 19[48] 4[49] 3
] 0[50] 19[51] 11[52] 11[53] 4[54] 25[55] 8[56] 29[57] 8[58] 42[59] 28[60] 2[61] 8[62] 21[63] 5[64] 37[65] 5[66]
] 38[67] 43[68] 32[69] 21[70] 9[71] 43[72] 44[73] 14[74] 1[75] 14[76] 30[77] 33[78] 6[79] 26[80] 22[81] 6[82] 47[
] 83[83] 11[84] 23[85] 35[86] 41[87] 8[88] 30[89] 11[90] 38[91] 46[92] 10[93] 3[94] 37[95] 35[96] 43[97] 38[98] 7[99]
]
Masukkan angka yang ingin dicari: 13
angka 13 tidak ditemukan pada array
Tekan sembarang tombol untuk melanjutkan...

```

Gambar 5 Tampilan Sequential Searching Apabila Angka Tidak Ditemukan



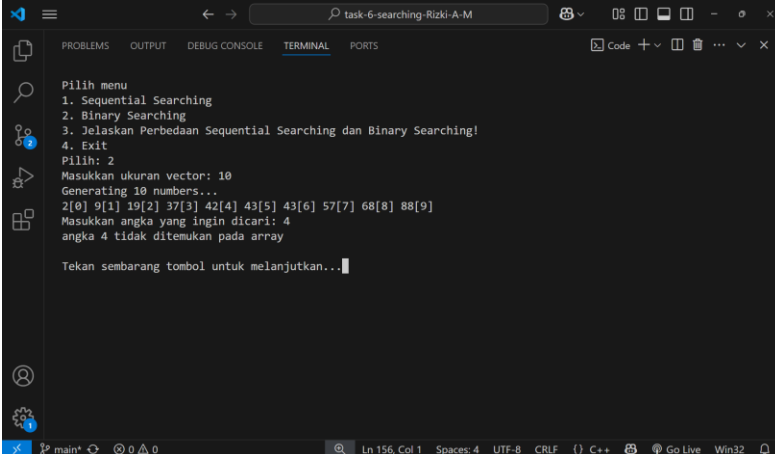
```

Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 2
Masukkan ukuran vector: 15
Generating 15 numbers...
7[0] 14[1] 34[2] 43[3] 47[4] 48[5] 53[6] 54[7] 57[8] 60[9] 75[10] 89[11] 90[12] 99[13] 99[14]
Masukkan angka yang ingin dicari: 47
angka 47 ditemukan pada indeks ke 4

Tekan sembarang tombol untuk melanjutkan...

```

Gambar 6 Tampilan Binary Searching Apabila Angka Ditemukan



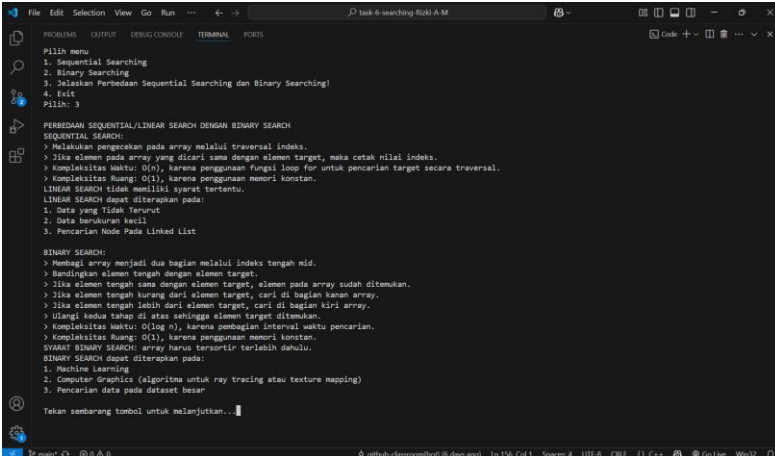
```

Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 2
Masukkan ukuran vector: 10
Generating 10 numbers...
2[0] 9[1] 19[2] 37[3] 42[4] 43[5] 57[7] 68[8] 88[9]
Masukkan angka yang ingin dicari: 4
angka 4 tidak ditemukan pada array

Tekan sembarang tombol untuk melanjutkan...

```

Gambar 7 Tampilan Binary Searching Apabila Angka Tidak Ditemukan



```

Pilih menu
1. Sequential Searching
2. Binary Searching
3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!
4. Exit
Pilih: 3

PERBEDAAN SEQUENTIAL/LINEAR SEARCH DENGAN BINARY SEARCH

SEQUENTIAL SEARCH:
> Melakukan pengecekan pada array melalui traversal indeks.
> Jika elemen pada array yang dicari sama dengan elemen target, maka cetak nilai indeks.
> Kompleksitas waktu: O(n), karena penggunaan fungsi loop for untuk pencarian target secara traversal.
> Kompleksitas ruang: O(1), karena penggunaan memori konstan.
LINEAR SEARCH tidak memiliki syarat tertentu.
LINEAR SEARCH dapat diterapkan pada:
1. Data yang tidak terurut
2. Data berukuran kecil
3. Pencarian Node Pada Linked List

BINARY SEARCH:
> Membagi array menjadi dua bagian melalui indeks tengah mid.
> Bandingkan elemen tengah dengan elemen target.
> Jika elemen tengah sama dengan elemen target, elemen pada array sudah ditemukan.
> Jika elemen tengah kurang dari elemen target, cari di bagian kanan array.
> Jika elemen tengah lebih dari elemen target, cari di bagian kiri array.
> Ulangi kedua tahap di atas sehingga elemen target ditemukan.
> Kompleksitas waktu: O(log n), karena pembagian interval waktu pencarian.
> Kompleksitas ruang: O(1), karena penggunaan memori konstan.
SYARAT BINARY SEARCH: array harus terurut terlebih dahulu.
BINARY SEARCH dapat diterapkan pada:
1. Machine Learning
2. Computer Graphics (algoritma untuk ray tracing atau texture mapping)
3. Pencarian data pada dataset besar

Tekan sembarang tombol untuk melanjutkan...

```

Gambar 8 Tampilan Penjelasan Perbedaan Dua Metode Search

## C Pembahasan

### • Alur Program

Program Search apabila dimulai, akan muncul menu utama yang akan menampilkan 4 pilihan. Pilihan pertama untuk melakukan Sequential Search, pilihan kedua untuk melakukan Binary Search, pilihan ketiga menampilkan penjelasan perbedaan dari Sequential Search dan Binary Search, dan pilihan keempat untuk keluar dari program yang ada. Program yang ada akan terus muncul atau jalan karena adanya loop dan akan berhenti apabila pilihan keempat atau exit dipilih. Setelah setiap pilihan yang ada selesai melakukan apa yang diinginkan oleh user, tampilan yang ada akan otomatis dibersihkan dengan menekan tombol apapun yang ada agar tampilan terminal tetap rapi dan tidak menumpuk.

Apabila pilihan pertama yaitu Sequential Search yang dipilih, program yang ada akan secara otomatis membuat 100 angka acak dari nilai 1 sampai 50 dan menampilkannya dengan indeks. Kemudian akan muncul pilihan angka berapa yang akan di cari dari 100 angka acak yang ada. Setelah menentukan angka yang ingin dicari, maka fungsi *sequentialSearch()* akan bekerja dengan memindai setiap elemen dalam *vector* secara berurutan. Dan akhirnya hasil dari fungsi tersebut akan memberi tahu user kalo angka yang dicari ditemukan sebanyak sekian dan di indeks berapa saja angka tersebut ada, apabila angka yang ingin dicari tidak ada maka akan muncul pesan angka yang ingin dicari tidak ditemukan.

Apabila pilihan kedua yaitu Binary Search yang dipilih, program akan meminta user yang ada untuk menentukan ukuran dari *vektor* yang ingin dibuat. User harus memasukkan nilai lebih dari 0 agar program dapat lanjut ke tahap berikutnya, apabila user memasukkan besar *vektor* sama dengan 0 program akan menampilkan pesan Error dan kembali ke menu utama. Lanjut setelah selesai menentukan besar *vektor*, program akan secara otomatis membuat angka dari 1 sampai 100 sesuai dengan besar dari *vektor* awal dan ditampilkan dalam indeks. Kemudian user diminta untuk memasukkan angka berapa yang ingin

dicari, setelahnya fungsi *binarySearch()* akan dijalankan. Fungsi *binary search* bekerja dengan membagi *vector* menjadi dua bagian secara berulang, sehingga lebih efisien untuk data yang sudah terurut. Dan akhirnya akan menampilkan angka yang dicari ada di indeks berapa dan apabila angka yang dicari tidak ada akan muncul pesan kalo angka yang dicari tidak ditemukan.

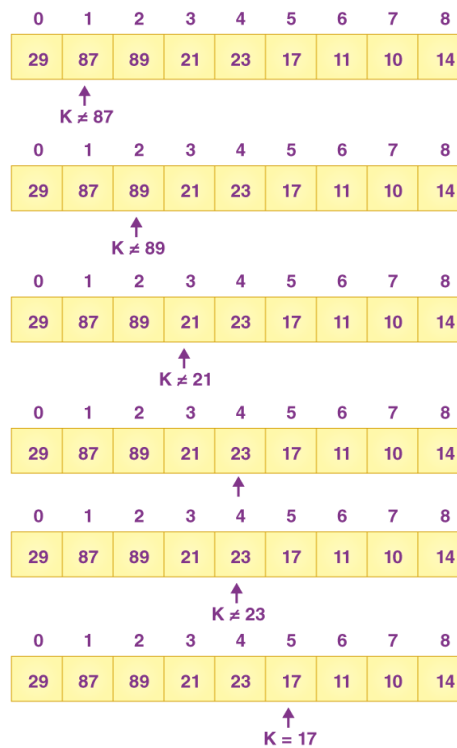
Apabila pilihan ketiga yang dipilih, akan muncul penjelasan dari perbedaan Sequential Search dan Binary Search. Penjelasan yang ada akan membahas cara kerja masing-masing, analisis kompleksitas waktu dan ruang serta kondisi yang cocok untuk penerapan algoritma yang ada.

Terus yang terakhir pilihan keempat yaitu exit, akan menampilkan pesan penutup untuk mengakhiri program yang ada.

- **Sequential Search**

Sequential Search, adalah metode yang bekerja dengan memeriksa setiap elemen dalam sebuah daftar secara berurutan hingga menemukan nilai yang dicari atau mencapai akhir daftar. Dalam implementasi ini, ketika elemen yang cocok dengan nilai target ditemukan, indeks (posisi) dari elemen tersebut akan disimpan ke dalam sebuah *vector* bernama *indices*.

Untuk menentukan apakah nilai target ada dalam daftar, program cukup memeriksa apakah *vector indices* kosong. Apabila *indices* kosong, ini menandakan bahwa nilai target tidak ditemukan sama sekali. Namun, apabila *indices* tidak kosong, program akan menampilkan berapa kali nilai target ditemukan (menggunakan fungsi *size()* dari *vector indices*), dan kemudian akan mencetak semua indeks tempat nilai target tersebut berada. Proses pencetakan indeks ini dilakukan dengan mengiterasi melalui setiap elemen di dalam *vector indices* menggunakan *loop for*.



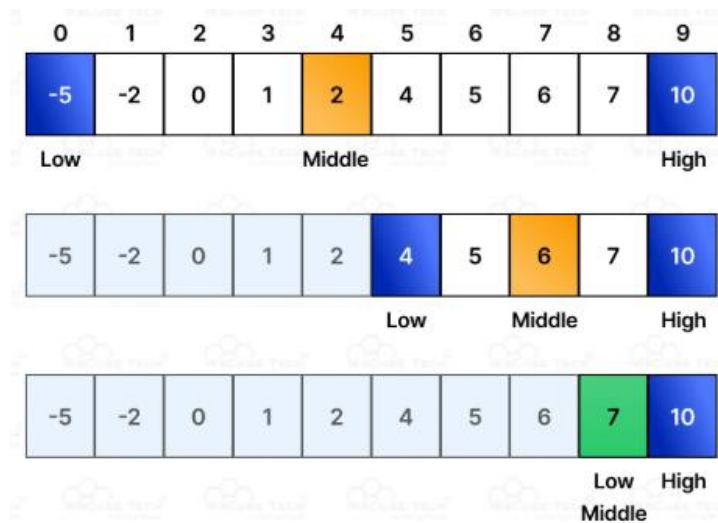
Gambar 9 Ilustrasi Sequential Search

Sequential Search memiliki kompleksitas waktu  $O(n)$ , dimana fungsi ini melakukan iterasi melalui setiap elemen dalam *vector* *nums* menggunakan *loop* *for*. Dalam skenario terburuk (misalnya, elemen target berada di akhir *vector* atau tidak ada sama sekali), fungsi ini harus memeriksa semua  $n$  elemen. Oleh karena itu, Semakin banyak jumlah elemen dalam vektor, maka waktu yang dibutuhkan juga semakin lama. Artinya, waktu prosesnya tumbuh sebanding dengan jumlah elemen, atau disebut juga  $O(n)$  (linear). Sementara itu, Sequential Search memiliki kompleksitas ruang  $O(n)$ , dimana fungsi ini menggunakan *vector* *indices* untuk menyimpan semua indeks tempat angka target ditemukan. Dalam skenario terburuk, apabila semua elemen dalam *vector* *nums* adalah angka target yang dicari, maka *vector* *indices* akan menyimpan  $n$  indeks. Jadi, semakin banyak elemen dalam vektor input, maka semakin besar juga memori yang dibutuhkan. Ini berarti penggunaan memorinya sebanding dengan jumlah elemen, atau disebut  $O(n)$  (linear).

- **Binary Search**

Binary Search adalah algoritma pencarian yang sangat efisien, dirancang khusus untuk menemukan elemen dalam sebuah daftar atau *array* yang sudah dalam kondisi terurut. Cara kerjanya adalah dengan terus-menerus mengurangi interval pencarian elemen target secara iteratif. Dalam kode ini, implementasi Binary Search menggunakan pendekatan iteratif melalui *loop while* untuk secara sistematis membagi ruang pencarian dan membandingkannya dengan elemen target.

Proses dimulai dengan mendefinisikan batas-batas pencarian, *high* diinisialisasi sebagai indeks paling kanan (*ukuran array - 1*), dan *low* sebagai indeks paling kiri (*0*). Sebuah variabel *index* juga disiapkan dengan nilai *-1* sebagai penanda awal apabila target belum ditemukan. *Loop while* akan terus berjalan selama *low* tidak melebihi *high*. Di setiap iterasi, program menghitung *mid* (indeks tengah) dari interval pencarian saat ini. Nilai elemen di *mid* kemudian dibandingkan dengan elemen target. Apabila **sama**, pencarian berhasil dan *index* diperbarui. Apabila elemen di *mid* **lebih kecil dari** target, pencarian dilanjutkan di paruh kanan *array* dengan menggeser *low* ke *mid + 1*, dan apabila elemen di *mid* **lebih besar dari** target, pencarian dilanjutkan di paruh kiri *array* dengan menggeser *high* ke *mid - 1*. Proses pembagian dan perbandingan ini berulang hingga target ditemukan atau interval pencarian habis, menandakan target tidak ada di dalam *array*.



Gambar 10 Ilustrasi Binary Search

Binary Search memiliki kompleksitas waktu  $O(\log n)$ , dimana fungsi ini bekerja dengan membagi ruang pencarian menjadi dua pada setiap langkah iterasi (melalui variabel *mid*). Ini berarti jumlah langkah yang dibutuhkan untuk menemukan elemen target (atau menentukan bahwa elemen tidak ada) berkurang secara eksponensial dengan setiap langkah. Misalnya, ada 1024 elemen, dibutuhkan maksimal sekitar 10 langkah ( $\log_2 1024 = 10$ ). Jadi, semakin banyak elemen, waktu yang dibutuhkan memang bertambah, tapi tidak terlalu cepat atau hanya naik sedikit setiap kali jumlah elemen digandakan. Ini disebut  $O(\log n)$  (logaritmik). Sementara itu, Binary Search memiliki kompleksitas ruang  $O(1)$ , dimana fungsi ini hanya menggunakan sejumlah kecil variabel tetap seperti *high*, *low*, *mid*, dan *index*, terlepas dari ukuran *vector* masukan. Ruang memori yang digunakan tidak bertambah seiring dengan peningkatan *n*, sehingga kompleksitas ruangnya adalah  $O(1)$  (konstan).

#### • Perbandingan Sequential Search Dengan Binary Search

Pada dasarnya, perbedaan fundamental antara Sequential Search dan Binary Search terletak pada caranya untuk melakukan pencarian. Sequential Search bekerja dengan metode *traversal* yang sederhana, dimana ia memeriksa setiap elemen dalam *array* satu per satu, bergerak dari indeks paling kiri hingga paling kanan, hingga elemen target ditemukan. Di sisi lain, Binary Search

menggunakan pendekatan yang jauh lebih efisien dengan membagi *array* menjadi dua bagian berdasarkan indeks tengah (*mid*). Setelah itu, ia membandingkan elemen tengah dengan target. Apabila sama, maka target ditemukan. Apabila elemen tengah kurang dari target, pencarian dilanjutkan di paruh kanan *array*, dan apabila elemen tengah lebih dari target, pencarian difokuskan pada paruh kiri *array*. Proses pembagian ini berulang hingga target ditemukan atau tidak ada lagi ruang pencarian.

Perbedaan cara kerja ini secara langsung memengaruhi kompleksitas waktu kedua algoritma. Sequential Search memiliki kompleksitas waktu  $O(n)$ , yang berarti waktu pencarian akan meningkat secara linear seiring dengan bertambahnya jumlah elemen ( $n$ ) dalam *array*, karena ia mungkin harus memeriksa setiap elemen. Hal ini disebabkan oleh iterasi *for loop* yang menyeluruh. Sebaliknya, Binary Search menunjukkan kompleksitas waktu yang jauh lebih efisien, yaitu  $O(\log n)$ . Efisiensi ini didapatkan karena pada setiap langkah, Binary Search secara efektif mengurangi ruang pencarian menjadi setengahnya, sehingga jumlah operasi yang diperlukan bertambah jauh lebih lambat dibandingkan ukuran *array*. Sementara itu, dalam hal kompleksitas ruang, baik Sequential Search maupun Binary Search umumnya dianggap memiliki kompleksitas  $O(1)$  untuk variabel utama mereka, karena penggunaan memori tambahan bersifat konstan dan tidak bergantung pada ukuran *input* (meskipun `sequentialSearch` dalam kode Anda memiliki `vector<int> indices` yang bisa berukuran  $O(n)$  di *worst case* apabila semua elemen adalah target).

Perbedaan mendasar lainnya adalah syarat yang harus dipenuhi oleh kedua algoritma. Sequential Search tidak memiliki syarat khusus, artinya ia dapat berfungsi dengan baik pada data yang tidak terurut, menjadikannya pilihan yang fleksibel untuk berbagai kondisi. Namun, fleksibilitas ini datang dengan konsekuensi karena kompleksitas waktunya  $O(n)$ , Sequential Search tidak cocok untuk *dataset* berukuran besar. Penerapannya lebih sesuai untuk data



yang tidak terurut, *array* berukuran kecil, atau pencarian *node* spesifik pada *linked list*.

Di sisi lain, Binary Search memiliki syarat ketat, ia hanya dapat bekerja pada *array* yang sudah tersortir. Jika data belum terurut, proses pengurutan awal (misalnya dengan Merge Sort atau Quick Sort yang memiliki kompleksitas waktu  $O(n \log n)$ ) harus dilakukan terlebih dahulu, yang menambah biaya komputasi. Meskipun demikian, karena efisiensinya yang tinggi  $O(\log n)$ , Binary Search sangat relevan dan dapat diterapkan pada bidang-bidang seperti Machine Learning, Computer Graphics (untuk algoritma *ray tracing* atau *texture mapping*), serta pencarian data pada *dataset* besar yang sudah terurut.

### **TAUTAN GITHUB**

[https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM\\_ALGORITMA\\_DAN\\_STRUKTUR\\_DATA.git](https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM_ALGORITMA_DAN_STRUKTUR_DATA.git)