**LAPORAN PRAKTIKUM**

**ALGORITMA & STRUKTUR DATA**

**MODUL 6**

****

**SEARCHING**

**Oleh:**

**Indra Suryadilaga NIM. 2410817310014**

**PROGRAM STUDI TEKNOLOGI INFORMASI**

**FAKULTAS TEKNIK**

**UNIVERSITAS LAMBUNG MANGKURAT**

**JUNI 2025**

# LEMBAR PENGESAHAN

**LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA**

**MODUL 5**

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

Nama Praktikan : Indra Suryadilaga

NIM : 2410817310014

|  |  |
| --- | --- |
| Menyetujui,  Asisten Praktikum  Zulfa Auliya Akbar  NIM. 2210817210010 | Mengetahui,  Dosen Penanggung Jawab Praktikum  Andreyan Rizky Baskara, S.Kom., M.Kom  NIP. 199307032019031011 |

# DAFTAR ISI

[LEMBAR PENGESAHAN 2](#_Toc199973002)

[DAFTAR ISI 3](#_Toc199973003)

[DAFTAR GAMBAR 4](#_Toc199973004)

[SOAL 1 5](#_Toc199973005)

[A. Source Code 8](#_Toc199973006)

[B. Output Program 13](#_Toc199973007)

[C. Pembahasan 14](#_Toc199973008)

[GITHUB 19](#_Toc199973009)

# DAFTAR GAMBAR

[Gambar 1. Soal No. 1 Source Code Sequential Searching 6](#_Toc200567039)

[Gambar 2. Soal No. 1 Source Code Binary Searching 7](#_Toc200567040)

[Gambar 3. Output Sequential Searching 13](#_Toc200567041)

[Gambar 4. Output Binary Searching 13](#_Toc200567042)

[Gambar 5. Output Penjelasan Perbedaan Sequential dan Binary Searching 14](#_Toc200567043)

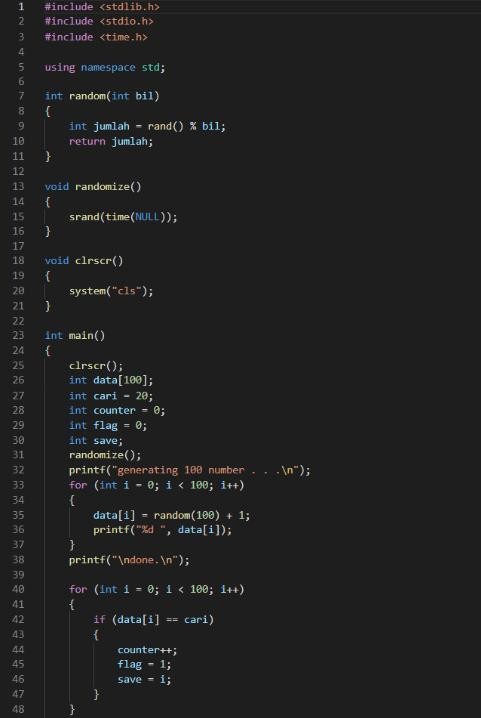
# DAFTAR TABEL

[Table 1. Source Code Algoritma Searcing 8](#_Toc200567071)

# SOAL 1

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

* Sequential Searching


Gambar 1. Soal No. 1 Source Code Sequential Searching

* Binary Searching





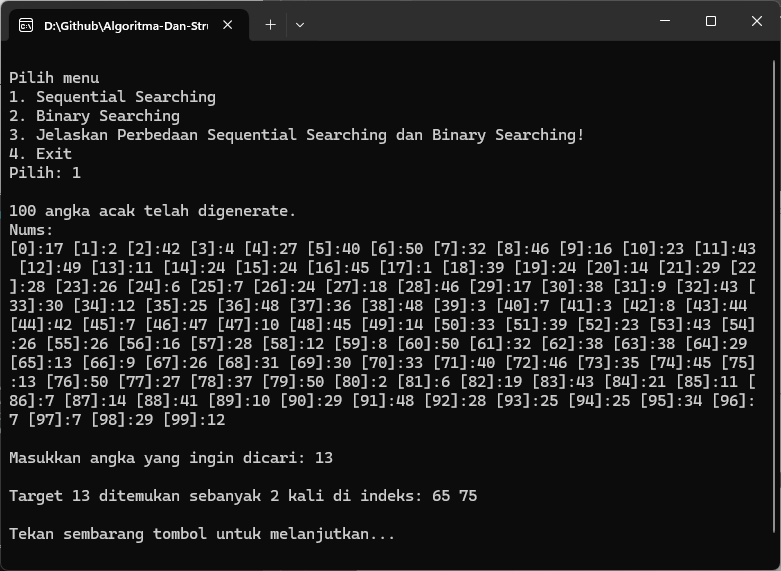
Gambar 2. Soal No. 1 Source Code Binary Searching

### Source Code

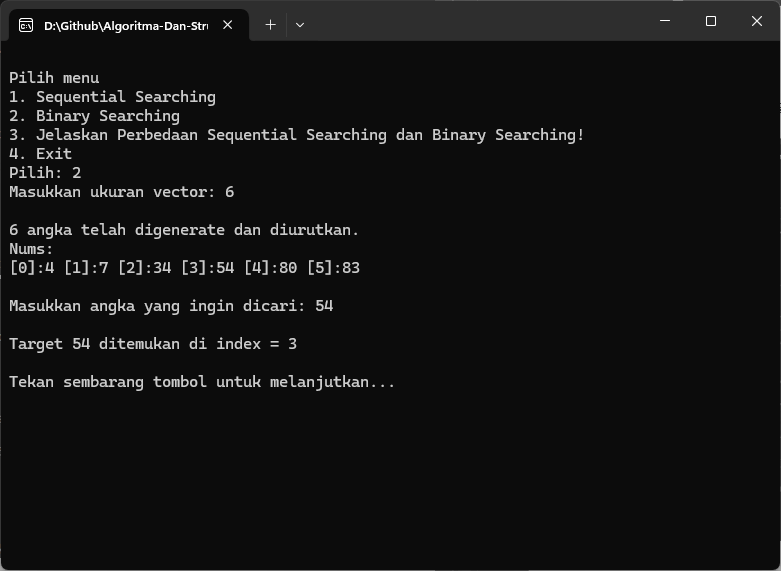
Table 1. Source Code Algoritma Searcing

|  |  |
| --- | --- |
|  | #include <iostream> |
|  | #include <conio.h> |
|  | #include <random> |
|  | #include <vector> |
|  | #include <algorithm> |
|  | #include <iomanip> |
|  | using namespace std; |
|  |  |
|  | // Sequential Search |
|  | void sequentialSearch(const vector<int>& nums, int target) { |
|  | vector<int> foundIndices; |
|  |  |
|  | cout << "\n100 angka acak telah digenerate.\nNums:\n"; |
|  | for (int i = 0; i < nums.size(); i++) { |
|  | cout << "[" << i << "]:" << nums[i] << " "; |
|  | } |
|  | cout << "\n\nMasukkan angka yang ingin dicari: "; |
|  | cin >> target; |
|  |  |
|  | for (int i = 0; i < nums.size(); i++) { |
|  | if (nums[i] == target) { |
|  | foundIndices.push\_back(i); |
|  | } |
|  | } |
|  |  |
|  | if (!foundIndices.empty()) { |
|  | cout << "\nTarget " << target << " ditemukan sebanyak " << foundIndices.size() << " kali di indeks: "; |
|  | for (int i : foundIndices) { |
|  | cout << i << " "; |
|  | } |
|  | cout << "\n"; |
|  | } else { |
|  | cout << "Target " << target << " tidak ditemukan dalam data.\n"; |
|  | } |
|  | } |
|  |  |
|  | // Binary Search |
|  | void binarySearch(const vector<int>& numsUnsorted, int target) { |
|  | vector<int> nums = numsUnsorted; |
|  |  |
|  | sort(nums.begin(), nums.end()); |
|  |  |
|  | cout << "\n" << nums.size() << " angka telah digenerate dan diurutkan.\nNums:\n"; |
|  | for (int i = 0; i < nums.size(); i++) { |
|  | cout << "[" << i << "]:" << nums[i] << " "; |
|  | } |
|  |  |
|  | cout << "\n\nMasukkan angka yang ingin dicari: "; |
|  | cin >> target; |
|  |  |
|  | int kiri = 0, kanan = nums.size() - 1; |
|  | int posisi = -1; |
|  |  |
|  | while (kiri <= kanan) { |
|  | int tengah = (kiri + kanan) / 2; |
|  | if (nums[tengah] == target) { |
|  | posisi = tengah; |
|  | break; |
|  | } else if (target < nums[tengah]) { |
|  | kanan = tengah - 1; |
|  | } else { |
|  | kiri = tengah + 1; |
|  | } |
|  | } |
|  |  |
|  | if (posisi != -1) |
|  | cout << "\nTarget " << target << " ditemukan di index = " << posisi << "\n"; |
|  | else |
|  | cout << "\nTarget tidak ditemukan dalam data.\n"; |
|  | } |
|  |  |
|  | void clearScreen() { |
|  | system("cls"); |
|  | } |
|  |  |
|  | // Penjelasan |
|  | void explain() { |
|  | cout << "\n=== PERBEDAAN SEQUENTIAL DAN BINARY SEARCHING ===\n"; |
|  | cout << "Sequential Search:\n"; |
|  | cout << " - Tidak perlu data terurut.\n"; |
|  | cout << " - Mencari satu-satu dari awal hingga akhir.\n"; |
|  | cout << " - Waktu pencarian: O(n).\n\n"; |
|  | cout << "Binary Search:\n"; |
|  | cout << " - Hanya bisa dilakukan pada data yang sudah terurut.\n"; |
|  | cout << " - Membagi dua bagian data hingga ketemu.\n"; |
|  | cout << " - Waktu pencarian: O(log n).\n"; |
|  | } |
|  |  |
|  | int main() { |
|  | int opt, target; |
|  |  |
|  | do { |
|  | cout << "\nPilih menu\n"; |
|  | cout << "1. Sequential Searching\n"; |
|  | cout << "2. Binary Searching\n"; |
|  | cout << "3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!\n"; |
|  | cout << "4. Exit\n"; |
|  | cout << "Pilih: "; |
|  | cin >> opt; |
|  |  |
|  | switch (opt) { |
|  | case 1: { |
|  | vector<int> nums(100); |
|  | mt19937\_64 rng(random\_device{}()); |
|  | uniform\_int\_distribution<int> dist(1, 50); |
|  |  |
|  | for (auto& val : nums) { |
|  | val = dist(rng); |
|  | } |
|  |  |
|  | sequentialSearch(nums, target); |
|  | break; |
|  | } |
|  |  |
|  | case 2: { |
|  | int size; |
|  | cout << "Masukkan ukuran vector: "; |
|  | cin >> size; |
|  |  |
|  | vector<int> nums(size); |
|  | mt19937\_64 rng(random\_device{}()); |
|  | uniform\_int\_distribution<int> dist(1, 100); |
|  |  |
|  | for (auto& val : nums) { |
|  | val = dist(rng); |
|  | } |
|  |  |
|  | binarySearch(nums, target); |
|  | break; |
|  | } |
|  |  |
|  | case 3: |
|  | explain(); |
|  | break; |
|  |  |
|  | case 4: |
|  | cout << "\nTERIMA KASIH\n"; |
|  | cout << "Program was made by Daniel Noprianto(2410817110010)\n"; |
|  | break; |
|  |  |
|  | default: |
|  | cout << "Opsi tidak valid. Coba lagi.\n"; |
|  | } |
|  |  |
|  | if (opt != 4) { |
|  | cout << "\nTekan sembarang tombol untuk melanjutkan..."; |
|  | getch(); |
|  | clearScreen(); |
|  | } |
|  |  |
|  | } while (opt != 4); |
|  |  |
|  | return 0; |
|  | } |

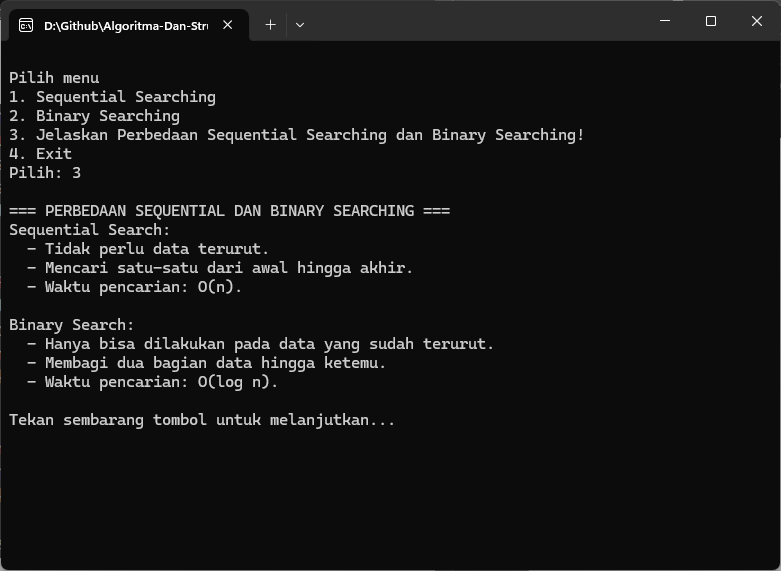
### Output Program



Gambar 3. Output Sequential Searching



Gambar 4. Output Binary Searching



Gambar 5. Output Penjelasan Perbedaan Sequential dan Binary Searching

### Pembahasan

**Alur Program Sorting**

Ketika program dijalankan, sistem akan menampilkan menu utama dengan 4 pilihan yang tersedia menggunakan do-while loop yang terus berjalan hingga user memilih opsi 4 (Exit). Setiap kali user memilih opsi 1-3, program akan melakukan proses sesuai dengan pilihan yang dipilih.

Untuk opsi 1 (Sequential Searching), program akan menggenerate 100 angka acak dengan rentang 1-50 menggunakan random number generator *mt19937\_64* dan *uniform\_int\_distribution*. Data yang dihasilkan disimpan ke vector dan ditampilkan beserta indeksnya. User kemudian diminta untuk memasukkan angka yang ingin dicari, dan program akan mencari semua angka tersebut dalam data.

Untuk opsi 2 (Binary Searching), program akan meminta user untuk memasukkan ukuran vector terlebih dahulu, kemudian menggenerate angka acak sesuai ukuran yang ditentukan dengan rentang 1-100. Data yang dihasilkan akan diurutkan terlebih dahulu menggunakan fungsi *sort()* dari *STL*, kemudian ditampilkan. Setelah itu, user diminta memasukkan angka yang ingin dicari, dan program akan melakukan pencarian menggunakan algoritma binary search.

Untuk opsi 3, program akan menampilkan penjelasan mengenai perbedaan antara Sequential Search dan Binary Search, termasuk karakteristik dan kompleksitas waktu masing-masing algoritma.

Pada akhir setiap operasi (kecuali exit), program akan menampilkan pesan "*Tekan sembarang tombol untuk melanjutkan...*" dan menunggu input menggunakan getch(). Setelah user menekan tombol apapun, layar akan dibersihkan menggunakan *system("cls")* dan menu utama akan ditampilkan kembali. Loop ini akan terus berlanjut hingga user memilih opsi 4 untuk keluar dari program.

**Analisis Algoritma Sequential Search**

Algoritma Sequential Search (Linear Search) bekerja dengan cara memeriksa setiap elemen dalam array secara berurutan dari awal hingga akhir untuk mencari target yang diinginkan. Implementasi ini menggunakan vector foundIndices untuk menyimpan semua indeks di mana target ditemukan, sehingga dapat menangani kasus di mana target muncul lebih dari sekali.

Proses searching dimulai dengan loop for yang iterasi dari indeks 0 hingga ukuran vector dikurangi satu. Pada setiap iterasi, program membandingkan elemen pada posisi i dengan target yang dicari. Jika elemen tersebut sama dengan target, indeks i akan disimpan dalam vector foundIndices. Proses ini berlanjut hingga seluruh elemen dalam array telah diperiksa.

Setelah proses pencarian selesai, program akan memeriksa apakah vector foundIndices kosong atau tidak. Jika tidak kosong, program akan menampilkan informasi bahwa target ditemukan beserta jumlah kemunculan dan semua indeks di mana target tersebut ditemukan. Jika kosong, program akan menampilkan pesan bahwa target tidak ditemukan dalam data.

Kelebihan dari Sequential Search adalah algoritma ini dapat bekerja pada data yang tidak terurut dan dapat menemukan semua kemunculan target dalam satu kali pencarian. Algoritma ini juga mudah diimplementasikan dan dipahami, serta memiliki space complexity yang rendah yaitu O(1) jika hanya mencari satu kemunculan. Sequential Search juga tidak memerlukan preprocessing data seperti pengurutan.

Kekurangan utama Sequential Search terletak pada time complexity yang kurang efisien untuk data berukuran besar. Pada worst case (target berada di akhir array atau tidak ada), algoritma ini memiliki time complexity O(n) karena harus memeriksa semua elemen. Average case juga memiliki time complexity O(n/2) yang masih tergolong O(n), sedangkan best case (target berada di awal array) memiliki time complexity O(1).

**Analisis Algoritma Binary Search**

Algoritma Binary Search menggunakan strategi divide and conquer dengan cara membagi array menjadi dua bagian secara berulang hingga target ditemukan atau tidak ada lagi elemen yang dapat dibagi. Algoritma ini hanya dapat bekerja pada data yang sudah terurut, sehingga implementasi ini menggunakan fungsi sort() terlebih dahulu untuk mengurutkan data.

Proses searching dimulai dengan menentukan batas kiri (kiri = 0) dan batas kanan (kanan = ukuran array - 1). Kemudian program menghitung posisi tengah menggunakan formula tengah = (kiri + kanan) / 2. Elemen pada posisi tengah akan dibandingkan dengan target yang dicari.

Jika elemen tengah sama dengan target, maka target telah ditemukan dan posisinya disimpan. Jika target lebih kecil dari elemen tengah, pencarian akan dilanjutkan pada bagian kiri array dengan mengubah kanan = tengah - 1. Jika target lebih besar dari elemen tengah, pencarian akan dilanjutkan pada bagian kanan array dengan mengubah kiri = tengah + 1. Proses ini berlanjut hingga target ditemukan atau kiri > kanan (target tidak ada).

Implementasi ini menggunakan variabel posisi untuk menyimpan indeks di mana target ditemukan. Jika posisi masih bernilai -1 setelah loop selesai, berarti target tidak ditemukan dalam data. Program kemudian akan menampilkan hasil pencarian kepada user.

Kelebihan utama Binary Search adalah time complexity yang sangat efisien yaitu O(log n) pada semua kasus (best, average, dan worst case). Algoritma ini sangat cocok untuk data berukuran besar dan memiliki performa yang predictable. Binary Search juga memiliki space complexity O(1) pada implementasi iterative dan memiliki efisiensi yang tinggi dalam praktik.

Kekurangan Binary Search terletak pada requirement bahwa data harus sudah terurut terlebih dahulu. Jika data belum terurut, diperlukan proses sorting yang memiliki time complexity minimal O(n log n), sehingga untuk pencarian sekali saja mungkin tidak efisien. Algoritma ini juga hanya dapat menemukan satu kemunculan target (biasanya yang pertama ditemukan) dan lebih kompleks untuk diimplementasikan dibanding Sequential Search.

**Perbandingan Sequential Search dan Binary Search**

Dari segi kompleksitas waktu, Binary Search memiliki keunggulan yang signifikan dengan O(log n) dibandingkan Sequential Search yang memiliki O(n). Namun, Binary Search memerlukan data yang sudah terurut, sedangkan Sequential Search dapat bekerja pada data apa adanya.

Dari segi penggunaan praktis, Sequential Search lebih cocok untuk dataset kecil atau ketika data sering berubah dan tidak selalu terurut. Binary Search lebih cocok untuk dataset besar yang sudah terurut atau ketika pencarian dilakukan berulang kali pada data yang sama.

Sequential Search juga memiliki keunggulan dalam menemukan semua kemunculan target dalam satu kali pencarian, sedangkan Binary Search biasanya hanya menemukan satu kemunculan. Untuk aplikasi yang memerlukan pencarian multiple occurrences, Sequential Search atau modifikasi Binary Search mungkin lebih sesuai.

# GITHUB

<https://github.com/IndraSuryadilaga/Algoritma-Dan-Struktur-Data>