**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  Muhammad Fauzan Ahsani  NIM. 2310817310009 | Mengetahui,  Dosen Penanggung Jawab Praktikum  Muti’a Maulida, S.Kom., M.TI.  NIP. 198810272019032013 |

# DAFTAR ISI

[LEMBAR PENGESAHAN i](#_Toc199324363)

[DAFTAR ISI ii](#_Toc199324364)

[DAFTAR TABEL iii](#_Toc199324365)

[DAFTAR GAMBAR iv](#_Toc199324366)

[SOAL 1 1](#_Toc199324367)

[A Output Program 3](#_Toc199324368)

[B Output Program 12](#_Toc199324369)

[C Pembahasan 14](#_Toc199324370)

[TAUTAN GITHUB 21](#_Toc199324371)

# DAFTAR TABEL

# DAFTAR GAMBAR

[Gambar 1 Contoh Source Code Sequential Searching 1](#_Toc199933678)

[Gambar 2 Contoh Source Code Binary Searching 2](#_Toc199933679)

[Gambar 3 Tampilan Menu Searching 12](#_Toc199933680)

[Gambar 4 Tampilan Sequential Searching Apabila Angka Ditemukan 12](#_Toc199933681)

[Gambar 5 Tampilan Sequential Searching Apabila Angka Tidak Ditemukan 12](#_Toc199933682)

[Gambar 6 Tampilan Binary Searching Apabila Angka Ditemukan 13](#_Toc199933683)

[Gambar 7 Tampilan Binary Searching Apabila Angka Tidak Ditemukan 13](#_Toc199933684)

[Gambar 8 Tampilan Penjelasan Perbedaan Dua Metode Search 13](#_Toc199933685)

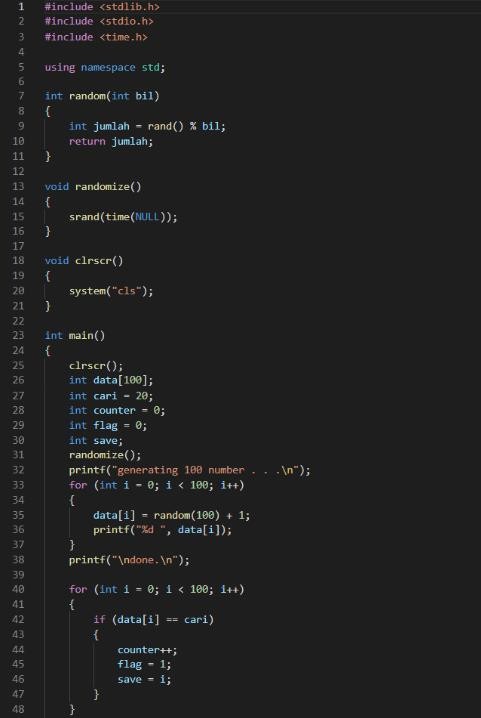
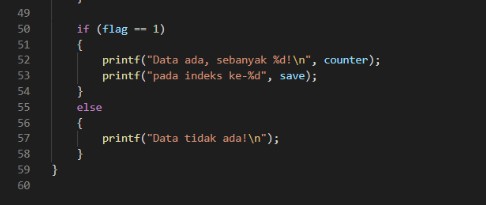
[Gambar 9 Ilustrasi Sequential Search 16](#_Toc199933686)

[Gambar 10 Ilustrasi Binary Search 18](#_Toc199933687)

# SOAL 1

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

* Sequential Searching



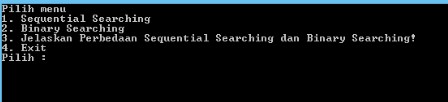
Gambar Contoh Source Code Sequential Searching

* Binary Searching



Gambar Contoh Source Code Binary Searching

* Tampilan Menu Program

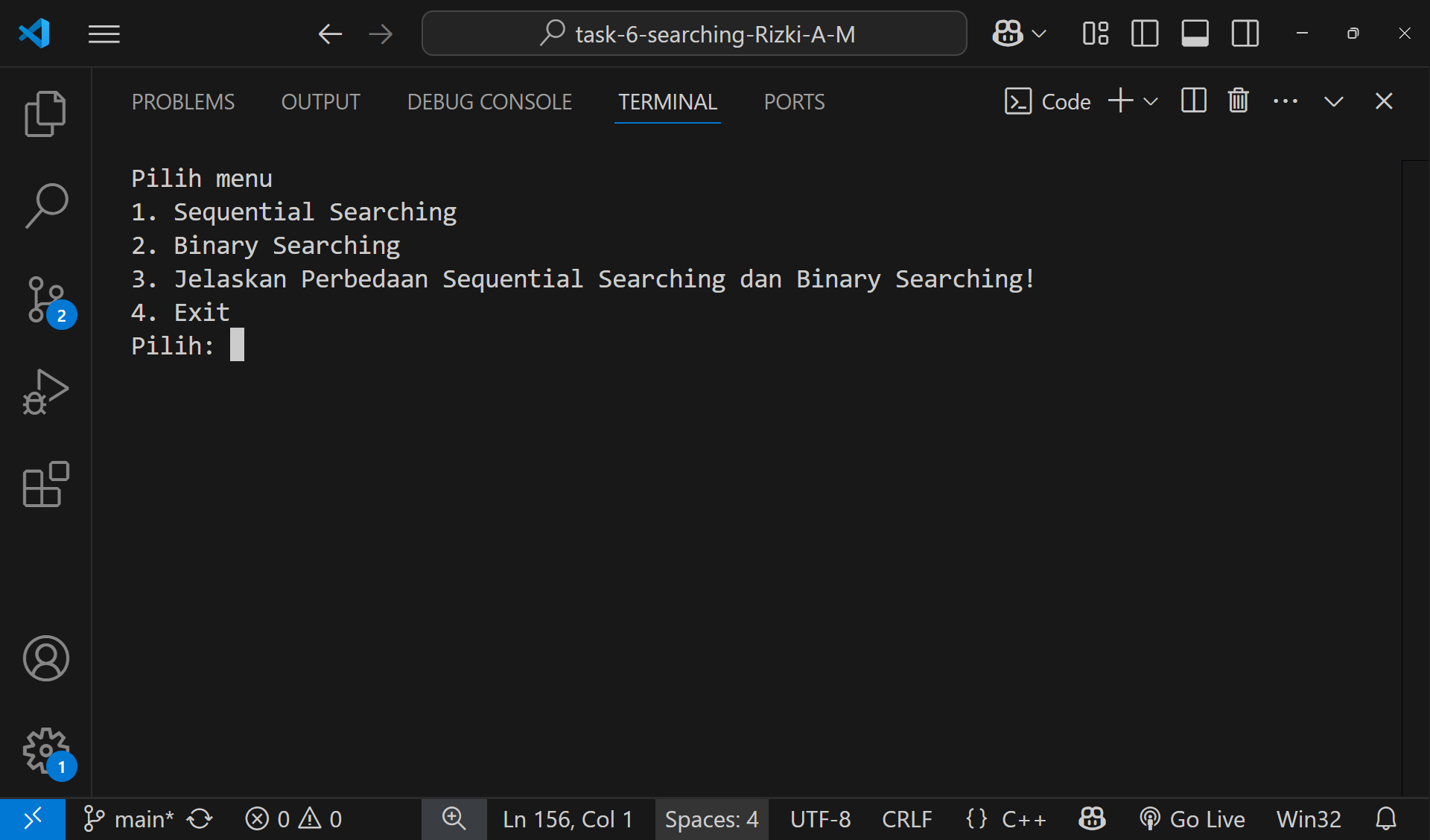


Jelaskan perbedaan Sequential Searching dan Binary Searching beserta kelebihan dan kekurangan masing-masing?

## Output Program

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186 | #include <iostream>  #include <conio.h>  #include <random>  #include <vector>  #include <algorithm>  using namespace std;  void sequentialSearch(const vector<int> &nums, int target)  {      vector<int> indices;      for (auto i = 0; i < nums.size(); i++)      {          if (nums[i] == target) indices.push\_back(i);      }      if (indices.empty()) cout << "angka " << target << " tidak ditemukan pada array" << endl;      else      {          cout << "Angka " << target << " terdapat pada array sebanyak " << indices.size() << " Kali" << endl;          cout << "Angka " << target << " ditemukan pada indeks: ";          for (auto i = 0; i < indices.size(); i++)          {              cout << indices[i];              if (i != indices.size() - 1) cout << ", ";          }          cout << "." << endl;      }  }  void binarySearch(const vector<int> &nums, int target)  {      int high = nums.size() - 1;      int low = 0;      int index = -1;      while (low <= high)      {          int mid = low + (high - low) / 2;          if (nums[mid] == target)          {              index = mid;              break;          }          if (nums[mid] < target) low = mid + 1;          else high = mid - 1;      }      if (index == -1) cout << "angka " << target << " tidak ditemukan pada array" << endl;      else cout << "angka " << target << " ditemukan pada indeks ke " << index << endl;  }  void clearScreen()  {      system("cls");  }  void explain()  {      cout << "\nPERBEDAAN SEQUENTIAL/LINEAR SEARCH DENGAN BINARY SEARCH" << endl;      cout << "SEQUENTIAL SEARCH: " << endl;      cout << "> Melakukan pengecekan pada array melalui traversal indeks." << endl;      cout << "> Jika elemen pada array yang dicari sama dengan elemen target, maka cetak nilai indeks." << endl;      cout << "> Kompleksitas Waktu: O(n), karena penggunaan fungsi loop for untuk pencarian target secara traversal." << endl;      cout << "> Kompleksitas Ruang: O(1), karena penggunaan memori konstan." << endl;      cout << "LINEAR SEARCH tidak memiliki syarat tertentu." << endl;      cout << "LINEAR SEARCH dapat diterapkan pada: " << endl;      cout << "1. Data yang Tidak Terurut" << endl;      cout << "2. Data berukuran kecil" << endl;      cout << "3. Pencarian Node Pada Linked List" << endl;      cout << endl;      cout << "BINARY SEARCH: " << endl;      cout << "> Membagi array menjadi dua bagian melalui indeks tengah mid." << endl;      cout << "> Bandingkan elemen tengah dengan elemen target." << endl;      cout << "> Jika elemen tengah sama dengan elemen target, elemen pada array sudah ditemukan." << endl;      cout << "> Jika elemen tengah kurang dari elemen target, cari di bagian kanan array." << endl;      cout << "> Jika elemen tengah lebih dari elemen target, cari di bagian kiri array." << endl;      cout << "> Ulangi kedua tahap di atas sehingga elemen target ditemukan." << endl;      cout << "> Kompleksitas Waktu: O(log n), karena pembagian interval waktu pencarian." << endl;      cout << "> Kompleksitas Ruang: O(1), karena penggunaan memori konstan." << endl;      cout << "SYARAT BINARY SEARCH: array harus tersortir terlebih dahulu." << endl;      cout << "BINARY SEARCH dapat diterapkan pada: " << endl;      cout << "1. Machine Learning" << endl;      cout << "2. Computer Graphics (algoritma untuk ray tracing atau texture mapping)" << endl;      cout << "3. Pencarian data pada dataset besar" << endl;  }  int main()  {      int opt, target;      do      {          cout << "Pilih menu" << endl;          cout << "1. Sequential Searching" << endl;          cout << "2. Binary Searching" << endl;          cout << "3. Jelaskan Perbedaan Sequential Searching dan Binary Searching!" << endl;          cout << "4. Exit" << endl;          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);                  }                  cout << "Generating 100 numbers..." << endl;                  for (auto i = 0; i < nums.size(); i++)                  {                      cout << nums[i] << "[" << i << "]" << " ";                  }                  cout << endl;                  cout << "Masukkan angka yang ingin dicari: "; cin >> target;                  sequentialSearch(nums, target);                  break;              }              case 2:              {                  int size;                  cout << "Masukkan ukuran vector: ";                  cin >> size;                  if (size < 1)                  {                      cout << "Error: Mohon masukkan bilangan di atas 0" << endl;                      break;                  }                  vector<int> nums(size);                  mt19937\_64 rng(random\_device{}());                  uniform\_int\_distribution<int> dist(1, 100);                  for (auto &val: nums)                  {                      val = (dist(rng));                  }                  sort(nums.begin(), nums.end());                  cout << "Generating " << size << " numbers..." << endl;                  for (auto i = 0; i < nums.size(); i++)                  {                      cout << nums[i] << "[" << i << "]" << " ";                  }                  cout << endl;                  cout << "Masukkan angka yang ingin dicari: "; cin >> target;                  binarySearch(nums, target);                  break;              }              case 3:                  explain();                  break;              case 4:                  cout << "\nTERIMA KASIH\n";                  cout << "Programme was made by Rizki Adhitiya Maulana (2410817110014)" << endl;                  break;              default:                  cout << "Opsi tidak terdefinisi, mohon masukkan ulang opsi" << endl;                  break;          }          if (opt != 4)          {              cout << "\nTekan sembarang tombol untuk melanjutkan...";              getch();              clearScreen();          }      }      while (opt != 4);      return 0;  } |

## Output Program

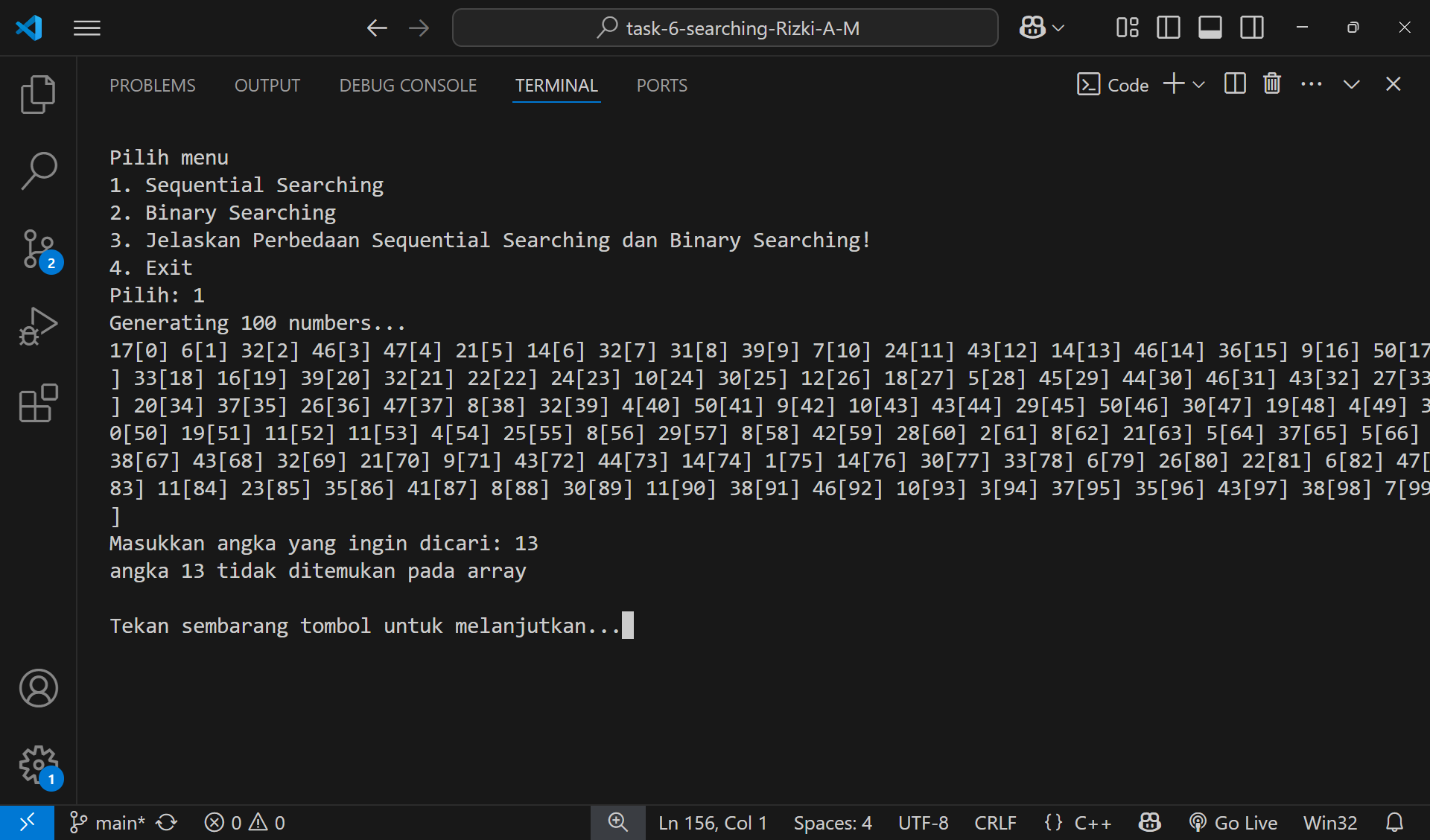


Gambar Tampilan Menu Searching

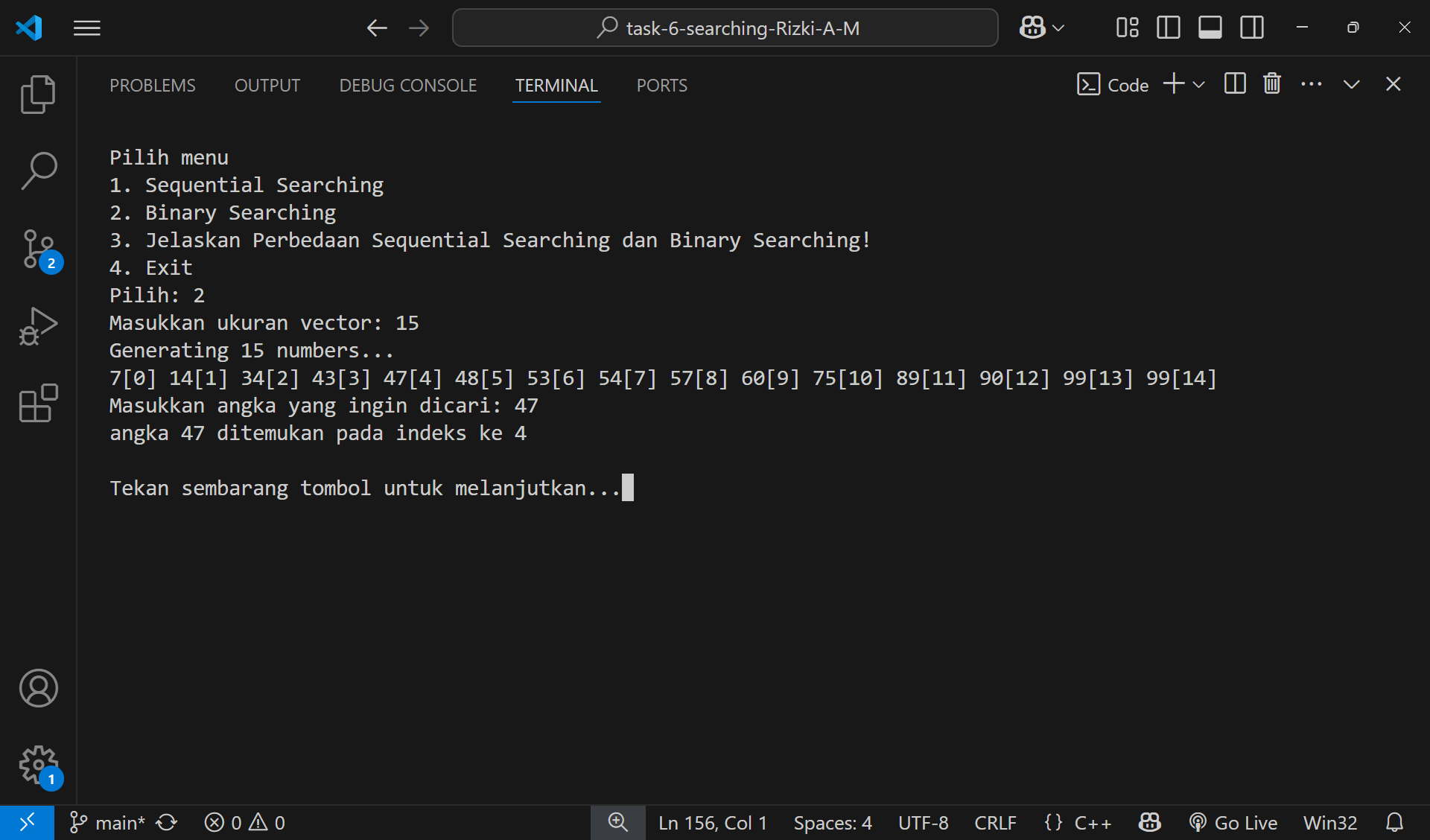
Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Sequential Searching Apabila Angka Ditemukan



Gambar Tampilan Sequential Searching Apabila Angka Tidak Ditemukan



Gambar Tampilan Binary Searching Apabila Angka Ditemukan

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Binary Searching Apabila Angka Tidak Ditemukan

Sebuah gambar berisi teks, cuplikan layar, software, komputer

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Penjelasan Perbedaan Dua Metode Search

## 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 angkat 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 yng 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.

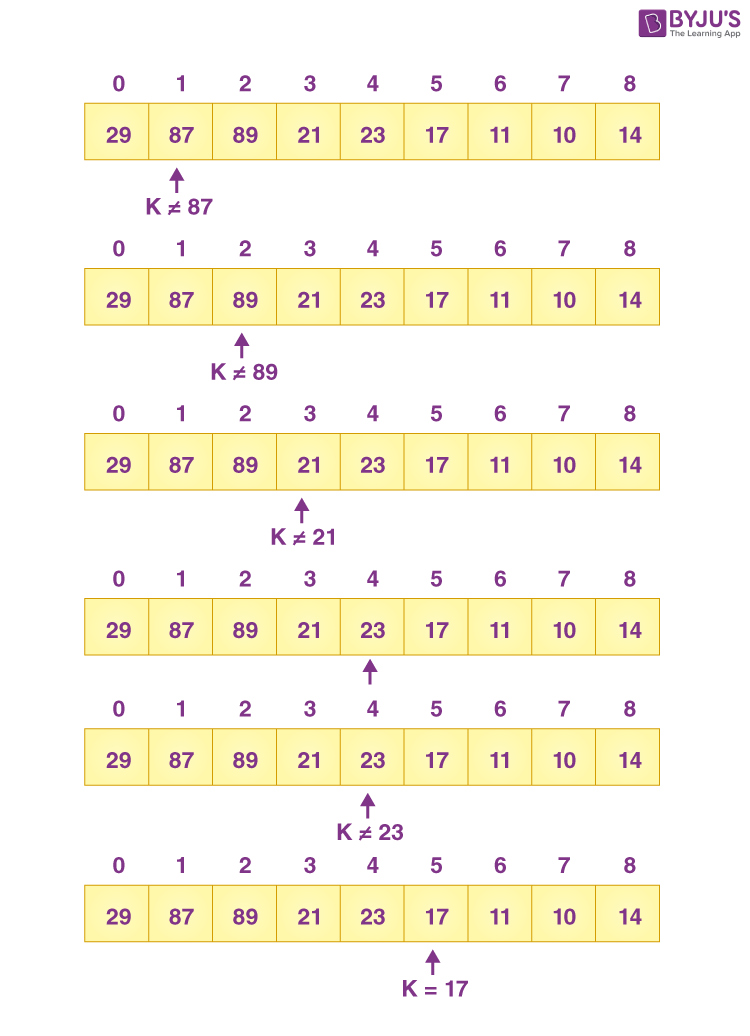
Apabila pilihan ketiga yang dipilih, akan muncul penjelasan dari perbedaan Sequential Search dan Binary Search. Penjelasan yang ada akan membahas cara kerja masing-masing, analisis kompleksitas waktu dan ruang serta kondisi yang cocok untuk penerapan algoritma yang ada.

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** vectorindices **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 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.

Sebuah gambar berisi teks, cuplikan layar, Persegi, nomor

Konten yang dihasilkan AI mungkin salah.

Gambar 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 (). 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