**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

**MODUL KE-5**

**SEARCHING PADA PYTHON**

****

**Disusun Oleh:**

**Nama :** Oktario Mufti Yudha

**NPM :** 2320506044

**Kelas :** 04 (Empat)

**Program Studi S1 Teknologi Informasi**

**Fakultas Teknik, Universitas Tidar**

**Genap 2023/2024**

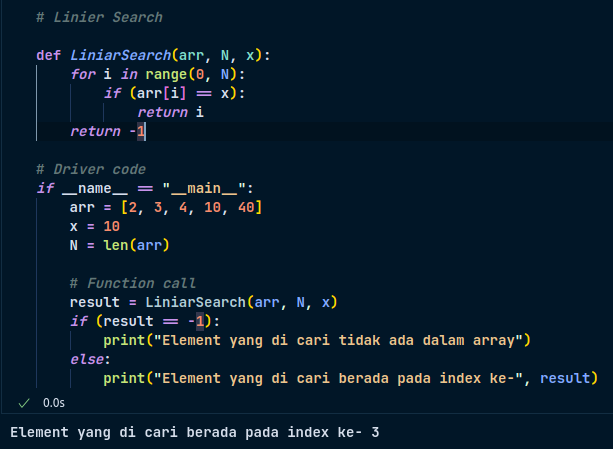
1. **Tujuan Praktikum**
2. Mahasiswa mampu memahami search pada python
3. Mahasiswa mampu menerapkan search pada program python
4. **Dasar Teori**

Python menawarkan berbagai metode pencarian kepada para pemrogram untuk mengambil informasi dengan efisien dari berbagai jenis struktur data. Ada dua jenis Algoritma Pencarian yang umum digunakan:

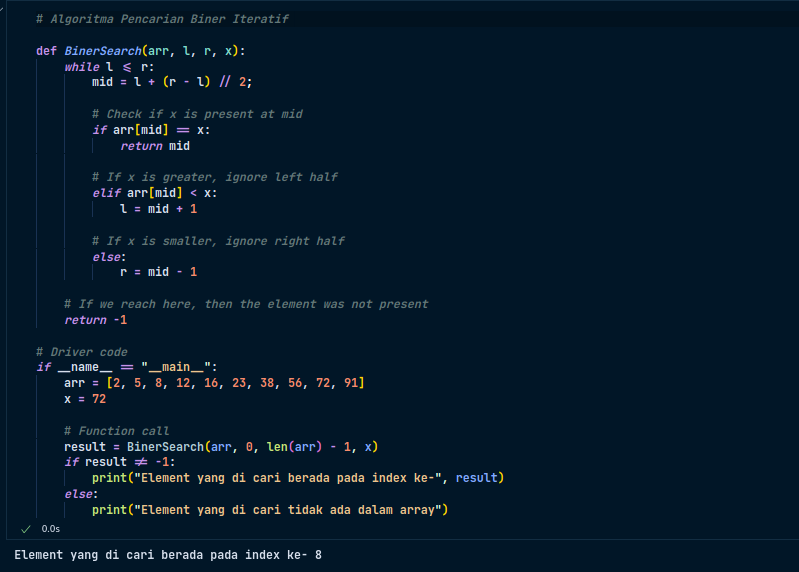
* Pencarian Sekuensial: Di sini, kita secara berurutan menelusuri semua elemen dan memeriksa masing-masing elemen.
* Pencarian Interval: Pada jenis pencarian ini, array dibagi menjadi 2 atau 3 interval pada setiap langkah. Setelah pembagian array menjadi interval, kita menentukan interval di mana kemungkinan elemen yang dicari berada, kemudian kita bagi interval tersebut menjadi 2 atau 3 sub-interval dan terus mengulangi proses ini sampai panjang subinterval menjadi 0.

Contoh dari Pencarian Sekuensial adalah Pencarian Linier, sementara contoh dari Pencarian Interval termasuk Pencarian Biner dan Pencarian Ternary.

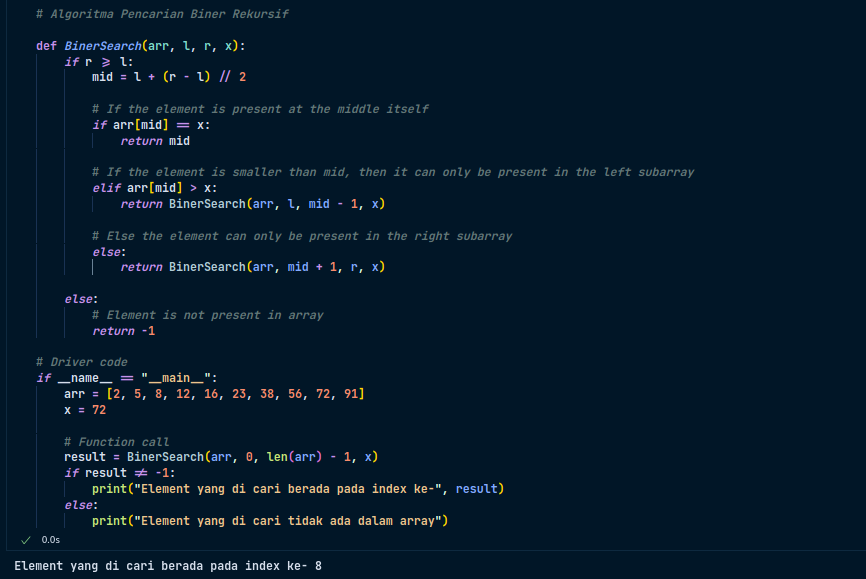
1. **Hasil dan Pembahasan**
   1. Linier Search

 *Gambar 3.1: Linier Search*

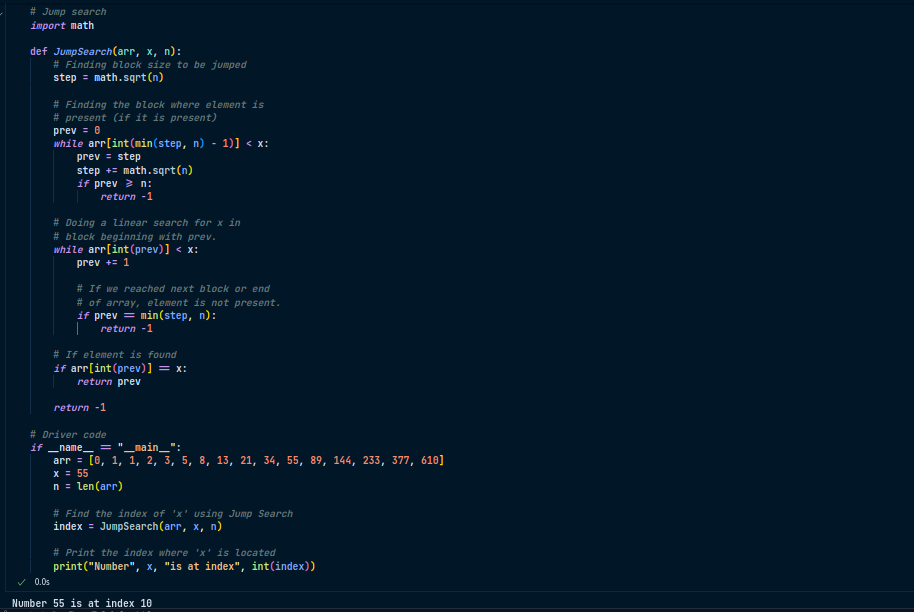
1. def LiniarSearch(arr, N, x): : Ini adalah definisi dari sebuah fungsi bernama LiniarSearch. Fungsi ini memiliki tiga parameter, yaitu arr (array atau daftar elemen yang akan dicari), N (jumlah elemen dalam array), dan x (elemen yang akan dicari).
2. for i in range(0, N): : Perulangan for ini akan melakukan iterasi dari i = 0 hingga N-1. Ini berarti kita akan memeriksa setiap elemen dalam array.
3. if (arr[i] == x): : Pernyataan if ini memeriksa apakah elemen ke-i dalam array sama dengan elemen yang dicari, yaitu x.
4. return i : Jika elemen yang dicari ditemukan, fungsi akan mengembalikan indeks elemen tersebut.
5. return -1 : Jika setelah iterasi selesai, elemen yang dicari tidak ditemukan, fungsi akan mengembalikan nilai -1.
6. if \_\_name\_\_ == "\_\_main\_\_": : Ini adalah sebuah pernyataan yang memeriksa apakah skrip Python sedang dijalankan sebagai program utama atau diimpor sebagai modul. Jika dijalankan sebagai program utama, blok kode di dalamnya akan dieksekusi.
7. arr = [2, 3, 4, 10, 40] : Inisialisasi array arr yang berisi beberapa elemen.
8. x = 10 : Inisialisasi variabel x dengan nilai yang akan dicari dalam array.
9. N = len(arr) : Inisialisasi variabel N dengan panjang (jumlah elemen) dari array arr.
10. result = LiniarSearch(arr, N, x) : Memanggil fungsi LiniarSearch dengan memberikan array arr, panjang array N, dan nilai yang akan dicari x sebagai argumen.
11. if (result == -1): : Memeriksa apakah hasil pencarian result sama dengan -1, yang menunjukkan bahwa elemen tidak ditemukan dalam array.
12. print("Element yang di cari tidak ada dalam array") : Mencetak pesan jika elemen tidak ditemukan dalam array.
13. else: : Jika hasil pencarian tidak sama dengan -1, berarti elemen ditemukan.
14. print("Element yang di cari berada pada index ke-", result) : Mencetak pesan yang menyatakan bahwa elemen ditemukan dan menampilkan indeksnya.
15. Output dari program ini akan bergantung pada apakah elemen yang dicari (x) ada dalam array (arr) atau tidak.
16. Algoritma Pencarian Biner Iteratif

 *Gambar 3.2: Algoritma Pencarian Biner Iteratif*

1. def BinerSearch(arr, l, r, x):: Mendefinisikan fungsi BinerSearch dengan empat parameter: arr (array yang akan dicari), l (indeks kiri), r (indeks kanan), dan x (elemen yang akan dicari).
2. while l <= r:: Melakukan loop while selama indeks kiri (l) kurang dari atau sama dengan indeks kanan (r).
3. mid = l + (r - l) // 2;: Menghitung indeks tengah dari rentang pencarian menggunakan operasi pembagian bulat (floor division).
4. if arr[mid] == x:: Memeriksa apakah elemen tengah sama dengan elemen yang dicari (x). Jika ya, mengembalikan indeks tengah.
5. elif arr[mid] < x:: Jika elemen tengah lebih kecil dari elemen yang dicari (x), mengabaikan setengah kiri dari array dengan menetapkan l ke mid + 1.
6. else:: Jika elemen tengah lebih besar dari elemen yang dicari (x), mengabaikan setengah kanan dari array dengan menetapkan r ke mid - 1.
7. return -1: Jika elemen tidak ditemukan dalam array, mengembalikan nilai -1.
8. if \_\_name\_\_ == "\_\_main\_\_":: Memeriksa apakah skrip Python sedang dijalankan sebagai program utama atau diimpor sebagai modul.
9. arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]: Inisialisasi array arr yang telah diurutkan.
10. x = 72: Inisialisasi nilai yang akan dicari dalam array.
11. result = BinerSearch(arr, 0, len(arr) - 1, x): Memanggil fungsi BinerSearch dengan memberikan array arr, indeks kiri (0), indeks kanan (len(arr) - 1), dan nilai yang akan dicari x sebagai argumen.
12. if result != -1:: Memeriksa apakah hasil pencarian result tidak sama dengan -1, yang menunjukkan bahwa elemen ditemukan dalam array.
13. print("Element yang di cari berada pada index ke-", result): Mencetak pesan yang menyatakan bahwa elemen ditemukan dan menampilkan indeksnya.
14. else:: Jika hasil pencarian sama dengan -1, berarti elemen tidak ditemukan dalam array.
15. print("Element yang di cari tidak ada dalam array"): Mencetak pesan yang menyatakan bahwa elemen tidak ditemukan.
16. Output dari program ini akan tergantung pada apakah elemen yang dicari (x) ditemukan dalam array yang telah diurutkan (arr) atau tidak.
17. Algoritma Pencarian Biner Rekursif

 *Gambar 3.3: Algoritma Pencarian Biner Rekursif*

1. def BinerSearch(arr, l, r, x): : Mendefinisikan fungsi BinerSearch dengan empat parameter: arr (array yang akan dicari), l (indeks kiri), r (indeks kanan), dan x (elemen yang akan dicari).
2. if r >= l: : Memeriksa apakah indeks kanan lebih besar atau sama dengan indeks kiri. Ini merupakan kondisi dasar untuk berhenti dari rekursi.
3. mid = l + (r - l) // 2 : Menghitung indeks tengah dari rentang pencarian menggunakan operasi pembagian bulat (floor division).
4. if arr[mid] == x: : Memeriksa apakah elemen tengah sama dengan elemen yang dicari (x). Jika ya, mengembalikan indeks tengah.
5. elif arr[mid] > x: : Jika elemen tengah lebih besar dari elemen yang dicari (x), melakukan pencarian pada setengah kiri dari array dengan memanggil fungsi BinerSearch secara rekursif dengan mengatur r ke mid - 1.
6. else: : Jika elemen tengah lebih kecil dari elemen yang dicari (x), melakukan pencarian pada setengah kanan dari array dengan memanggil fungsi BinerSearch secara rekursif dengan mengatur l ke mid + 1.
7. else: : Menangani kasus ketika r kurang dari l, yang menunjukkan bahwa elemen tidak ditemukan dalam array.
8. if \_\_name\_\_ == "\_\_main\_\_": : Memeriksa apakah skrip Python sedang dijalankan sebagai program utama atau diimpor sebagai modul.
9. arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91] : Inisialisasi array arr yang telah diurutkan.
10. x = 72 : Inisialisasi nilai yang akan dicari dalam array.
11. result = BinerSearch(arr, 0, len(arr) - 1, x) : Memanggil fungsi BinerSearch dengan memberikan array arr, indeks kiri (0), indeks kanan (len(arr) - 1), dan nilai yang akan dicari x sebagai argumen.
12. if result != -1: : Memeriksa apakah hasil pencarian result tidak sama dengan -1, yang menunjukkan bahwa elemen ditemukan dalam array.
13. print("Element yang di cari berada pada index ke-", result) : Mencetak pesan yang menyatakan bahwa elemen ditemukan dan menampilkan indeksnya.
14. else: : Jika hasil pencarian sama dengan -1, berarti elemen tidak ditemukan dalam array.
15. print("Element yang di cari tidak ada dalam array") : Mencetak pesan yang menyatakan bahwa elemen tidak ditemukan.
16. Jump Search

 *Gambr 3.4: Jump Search*

1. import math: Mengimpor modul math untuk menggunakan fungsi-fungsi matematika seperti sqrt (akar kuadrat).
2. def JumpSearch(arr, x, n): : Mendefinisikan fungsi JumpSearch dengan tiga parameter: arr (array yang akan dicari), x (elemen yang akan dicari), dan n (jumlah elemen dalam array).
3. step = math.sqrt(n): Menghitung ukuran blok yang akan dilewati pada setiap lompatan. Ini dihitung sebagai akar kuadrat dari jumlah elemen dalam array.
4. prev = 0: Variabel prev digunakan untuk menyimpan indeks awal blok sebelumnya.
5. while arr[int(min(step, n) - 1)] < x: : Melakukan pencarian di setiap blok hingga nilai elemen terakhir dalam blok tidak kurang dari nilai yang dicari (x). Jika nilai tersebut kurang dari x, maka pergeseran blok dilakukan dengan menambahkan nilai dari akar kuadrat dari jumlah elemen dalam array (step).
6. while arr[int(prev)] < x: : Melakukan pencarian linier di dalam blok dimulai dari indeks prev hingga ditemukan nilai yang sama dengan x atau nilai yang melebihi x.
7. if arr[int(prev)] == x: : Jika nilai yang ditemukan sama dengan x, mengembalikan indeks elemen tersebut.
8. return -1: Jika nilai x tidak ditemukan dalam array, mengembalikan -1.
9. if \_\_name\_\_ == "\_\_main\_\_": : Memeriksa apakah skrip Python sedang dijalankan sebagai program utama atau diimpor sebagai modul.
10. arr = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]: Inisialisasi array arr yang telah diurutkan.
11. x = 55: Inisialisasi nilai yang akan dicari dalam array.
12. n = len(arr): Menyimpan jumlah elemen dalam array.
13. index = JumpSearch(arr, x, n): Memanggil fungsi JumpSearch untuk mencari indeks dari nilai x dalam array.
14. print("Number", x, "is at index", int(index)): Mencetak indeks di mana nilai x ditemukan dalam array. Jika nilai x tidak ditemukan, pesan "Number x is at index -1" akan dicetak.
15. Output dari program ini adalah indeks di mana elemen yang dicari (x) ditemukan dalam array (arr). Jika elemen tersebut ditemukan, program akan mencetak pesan "Number x is at index index", di mana x adalah nilai yang dicari, dan index adalah indeks di mana elemen tersebut ditemukan dalam array. Jika elemen tidak ditemukan dalam array, program akan mencetak pesan "Number x is at index -1".
16. **Tugas**

a. Studi Kasus 1

 *Gambr 3.4: Studi Kasus 1*

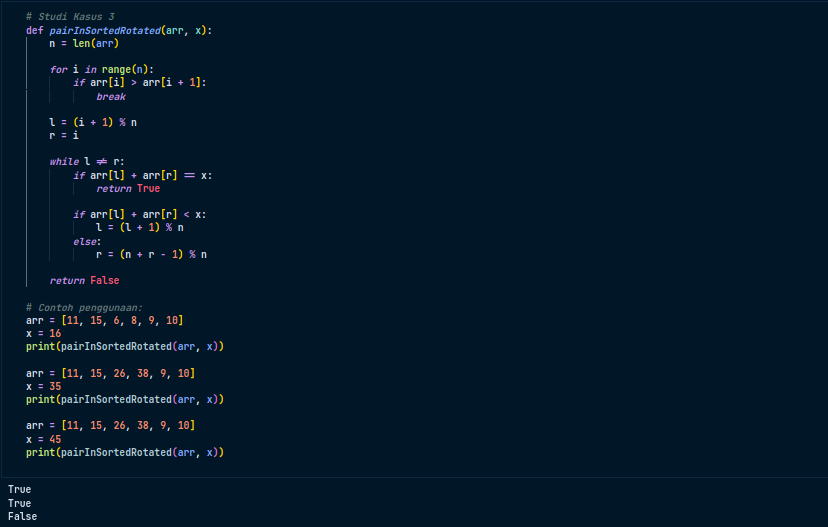
1. def findMinMax(numbers): : Mendefinisikan fungsi findMinMax yang menerima sebuah daftar angka sebagai argumen.
2. minNum = maxNum = numbers[0] : Menginisialisasi variabel minNum dan maxNum dengan elemen pertama dari daftar numbers.
3. for num in numbers[1:]: : Melakukan iterasi melalui daftar numbers dimulai dari indeks kedua hingga terakhir.
4. if num < minNum: : Memeriksa apakah nilai num lebih kecil dari minNum. Jika ya, minNum diperbarui dengan nilai num.
5. if num > maxNum: : Memeriksa apakah nilai num lebih besar dari maxNum. Jika ya, maxNum diperbarui dengan nilai num.
6. return minNum, maxNum : Mengembalikan nilai terkecil dan terbesar yang telah ditemukan.
7. scores = [90, 85, 88, 92, 78, 88, 95, 89, 84, 96, 87, 93, 82, 90, 91, 88, 85, 94, 92, 90, 86, 87, 88, 92, 89, 90, 91, 92, 93, 94, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94] : Mendefinisikan daftar scores yang berisi sejumlah nilai.
8. minScore, maxScore = findMinMax(scores) : Memanggil fungsi findMinMax dengan memberikan daftar scores sebagai argumen dan menyimpan nilai terkecil dan terbesar yang dikembalikan oleh fungsi tersebut.
9. print(f"Nilai terendah: {minScore}") : Mencetak nilai terkecil.
10. print(f"Nilai tertinggi: {maxScore}") : Mencetak nilai terbesar.
11. Output dari program ini akan menampilkan nilai terendah dan tertinggi dalam daftar scores.

b. Studi Kasus 2

 *Gambr 3.4: Studi Kasus 2*

1. def findMissingRepeating(arr): : Mendefinisikan fungsi findMissingRepeating yang menerima sebuah array sebagai argumen.
2. n = len(arr) : Mendapatkan panjang array arr dan menyimpannya dalam variabel n.
3. temp = [0] \* n : Membuat array sementara temp yang memiliki panjang yang sama dengan array arr, diisi dengan nilai 0.
4. repeating = missing = -1 : Menginisialisasi dua variabel repeating dan missing dengan nilai -1.
5. Melakukan iterasi melalui array arr:
   * if temp[i-1] == 0: : Memeriksa apakah nilai i (elemen saat ini) dalam array arr sudah ada dalam array temp. Jika belum, set temp[i-1] menjadi 1, menandakan bahwa nilai i sudah muncul.
   * Jika nilai i sudah ada dalam array temp, maka nilai tersebut adalah nilai yang terulang.
6. Melakukan iterasi melalui array temp:
   * if temp[i] == 0: : Memeriksa apakah nilai temp[i] adalah 0. Jika ya, artinya indeks i belum muncul dalam array arr, sehingga nilai i+1 adalah nilai yang hilang.
7. return missing, repeating : Mengembalikan nilai yang hilang dan nilai yang terulang.
8. Menggunakan fungsi findMissingRepeating untuk mencari nilai yang hilang dan nilai yang terulang dalam dua contoh array.
9. print(f"Hilang = {missing}, Terulang = {repeating}") : Mencetak nilai yang hilang dan nilai yang terulang.
10. Output dari program ini akan menampilkan nilai yang hilang dan nilai yang terulang dalam setiap contoh array yang diberikan.

c. Studi Kasus 3

 *Gambr 3.4: Studi Kasus 3*

1. def pairInSortedRotated(arr, x):: Mendefinisikan fungsi bernama pairInSortedRotated yang menerima dua parameter, yaitu arr (array) dan x (nilai yang akan dicari pasangannya).
2. n = len(arr): Menghitung panjang array arr dan menyimpannya dalam variabel n.
3. Iterasi melalui array untuk menemukan titik rotasi:
   1. for i in range(n):: Iterasi dari indeks ke-0 hingga n-1.
   2. if arr[i] > arr[i + 1]:: Memeriksa apakah elemen saat ini (arr[i]) lebih besar dari elemen berikutnya (arr[i+1]). Jika ya, ini menandakan titik rotasi, lalu keluar dari loop.
4. Menginisialisasi indeks l dan r:
   1. l = (i + 1) % n: Menginisialisasi l sebagai indeks pertama setelah titik rotasi, menggunakan operasi modulo % untuk menghadirkannya kembali ke awal array jika telah mencapai akhir array.
   2. r = i: Menginisialisasi r sebagai indeks terakhir sebelum titik rotasi.
5. Melakukan loop while untuk mencari pasangan angka yang memiliki jumlah yang sama dengan x:
   1. while l != r:: Melakukan loop selama l dan r tidak bertemu.
   2. if arr[l] + arr[r] == x:: Memeriksa apakah jumlah dari elemen pada indeks l dan r sama dengan x. Jika ya, mengembalikan True.
   3. if arr[l] + arr[r] < x:: Jika jumlahnya kurang dari x, maka geser l ke kanan.
   4. else:: Jika jumlahnya lebih dari x, maka geser r ke kiri.
6. Jika tidak ada pasangan angka yang memiliki jumlah yang sama dengan x, mengembalikan False.
7. Mengeksekusi fungsi pairInSortedRotated dengan contoh array dan nilai x, lalu mencetak hasilnya.
8. Output akan menjadi True, karena terdapat pasangan angka 6 dan 10 yang jumlahnya adalah 16.
9. Output akan menjadi False, karena tidak ada pasangan angka dalam array yang jumlahnya adalah 35.
10. Output akan menjadi True, karena terdapat pasangan angka 26 dan 19 yang jumlahnya adalah 45.
11. **Kesimpulan**

Dari praktikum ini, dapat disimpulkan bahwa kita telah mempelajari berbagai teknik pencarian dalam pemrograman, seperti pencarian linier, pencarian biner, dan pencarian lompat. Setiap teknik memiliki keunggulan dan kelemahan tergantung pada sifat array yang dicari. Contoh kode yang telah dijelaskan memberikan pemahaman tentang strategi pencarian yang digunakan, penerapan algoritma, dan cara mengintegrasikan fungsi-fungsi pencarian dalam kode yang lebih besar. Dengan memahami dan menguasai teknik-teknik pencarian tersebut, kita dapat mengembangkan pemahaman yang kuat dalam pemrograman dan mampu menyelesaikan berbagai masalah secara efisien.