LAPORAN PRAKTIKUM

MODUL VIII
SEARCH

Disusun oleh:
Syafanida Khakiki
NIM: 2311102005

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

1. Menunjukkan beberapa algoritma dalam Pencarian.
2. Menunjukkan bahwa pencarian merupakan suatu persoalan yang bisa diselesaikan dengan beberapa algoritma yang berbeda.
3. Dapat memilih algoritma yang paling sesuai untuk menyelesaikan suatu permasalahan pemrograman.

BAB II

DASAR TEORI

a. Pengertian Searching

Pencarian (Searching) yaitu proses menemukan suatu nilai tertentu pada kumpulan data. Hasil pencarian adalah salah satu dari tiga keadaan ini: data ditemukan, data ditemukan lebih dari satu, atau data tidak ditemukan. Searching juga dapat dianggap sebagai proses pencarian suatu data di dalam sebuah array dengan cara mengecek satu persatu pada setiap index baris atau setiap index kolomnya dengan menggunakan teknik perulangan untuk melakukan pencarian data. Terdapat 2 metode pada algoritma Searching, yaitu:

a. Sequential Search

Sequential Search merupakan salah satu algoritma pencarian data yang biasa digunakan untuk data yang berpola acak atau belum terurut. Sequential search juga merupakan teknik pencarian data dari array yang paling mudah, dimana data dalam array dibaca satu demi satu dan diurutkan dari index terkecil ke index terbesar, maupun sebaliknya. Konsep Sequential Search yaitu:

- Membandingkan setiap elemen pada array satu per satu secara berurut.
- Proses pencarian dimulai dari indeks pertama hingga indeks terakhir.
- Proses pencarian akan berhenti apabila data ditemukan. Jika hingga akhir array data masih juga tidak ditemukan, maka proses pencarian tetap akan dihentikan.
- Proses perulangan pada pencarian akan terjadi sebanyak jumlah N elemen pada array.

Algoritma pencarian berurutan dapat dituliskan sebagai berikut :

1) i ← 0
2) ketemu ← false
3) Selama (tidak ketemu) dan (i <= N) kerjakan baris 4
4) Jika (Data[i] = x) maka ketemu ← true, jika tidak i ← i + 1
5) Jika (ketemu) maka i adalah indeks dari data yang dicari, jika tidak data tidak ditemukan.

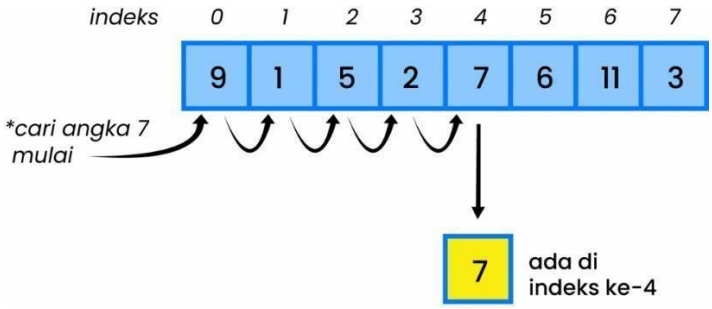
Di bawah ini merupakan fungsi untuk mencari data menggunakan pencarian sekuensial.

```
int SequentialSearch (int x)
{
    int i = 0;
    bool ketemu = false;
    while ((!ketemu) && (i < Max)){
        if (Data[i] == x)
            ketemu = true;
       else
            i++;
    }
        if   (ketemu)
            return i;
        else
            return -1;
    }
```

Fungsi diatas akan mengembalikan indeks dari data yang dicari. Apabila data tidak ditemukan maka fungsi diatas akan mengembalikan nilai –1.

Contoh dari Sequential Search, yaitu:

Int A[8] = {9,1,5,2,7,6,11,3}



Gambar 1. Ilustrasi Sequential Search

Misalkan, dari data di atas angka yang akan dicari adalah angka 7 dalam array A, maka proses yang akan terjadi yaitu:

- Pencarian dimulai pada index ke-0 yaitu angka 9, kemudian dicocokkan dengan angka yang akan dicari, jika tidak sama maka pencarian akan dilanjutkan ke index selanjutnya.
- Pada index ke-1, yaitu angka 1, juga bukan angka yang dicari, maka pencarian akan dilanjutkan pada index selanjutnya.
- Pada index ke-2 dan index ke-3 yaitu angka 5 dan 2, juga bukan angka yang dicari, sehingga pencarian dilanjutkan pada index selanjutnya.
- Pada index ke-4 yaitu angka 7 dan ternyata angka 7 merupakan angka yang dicari, sehingga pencarian akan dihentikan dan proses selesai.

b. Binary Search

Binary Search termasuk ke dalam interval search, dimana algoritma ini merupakan algoritma pencarian pada array/list dengan elemen terurut. Pada metode ini, data harus diurutkan terlebih dahulu dengan cara data dibagi menjadi dua bagian (secara logika), untuk setiap tahap pencarian. Dalam penerapannya algoritma ini sering digabungkan dengan algoritma sorting karena data yang akan digunakan harus sudah terurut terlebih dahulu. Konsep Binary Search:

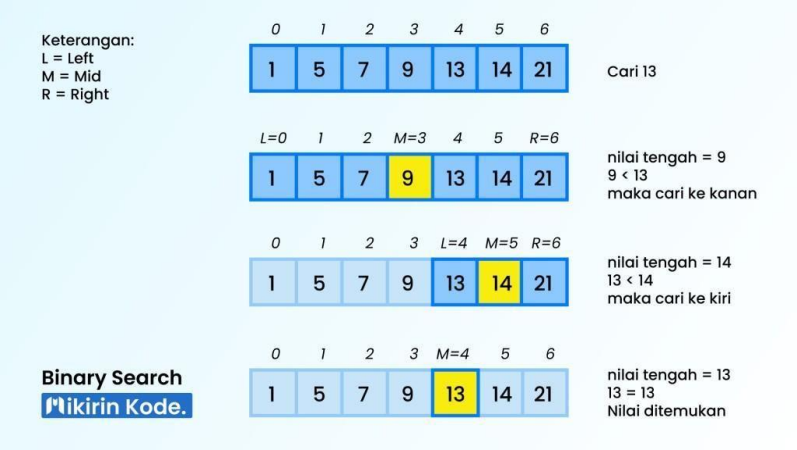
- Data diambil dari posisi 1 sampai posisi akhir N.
- Kemudian data akan dibagi menjadi dua untuk mendapatkan posisi data tengah.
- Selanjutnya data yang dicari akan dibandingkan dengan data yang berada di posisi tengah, apakah lebih besar atau lebih kecil.
- Apabila data yang dicari lebih besar dari data tengah, maka dapat dipastikan bahwa data yang dicari kemungkinan berada di sebelah kanan dari data tengah. Proses pencarian selanjutnya akan dilakukan pembagian data menjadi dua bagian pada bagian kanan dengan acuan posisi data tengah akan menjadi posisi awal untuk pembagian tersebut.
- Apabila data yang dicari lebih kecil dari data tengah, maka dapat dipastikan bahwa data yang dicari kemungkinan berada di sebelah kiri dari data tengah. Proses pencarian selanjutnya akan dilakukan pembagian data menjadi dua bagian pada bagian kiri. Dengan acuan posisi data tengah akan menjadi posisi akhir untuk pembagian selanjutnya.
- Apabila data belum ditemukan, maka pencarian akan dilanjutkan dengan kembali membagi data menjadi dua.
- Namun apabila data bernilai sama, maka data yang dicari langsung ditemukan dan pencarian dihentikan.

Algoritma pencarian biner dapat dituliskan sebagai berikut :

1) L □ 0
2) R □ N – 1
3) ketemu □ false
4) Selama (L <= R) dan (tidak ketemu) kerjakan baris 5 sampai dengan 8
5) m □ (L + R) / 2
6) Jika (Data[m] = x) maka ketemu □ true
7) Jika (x < Data[m]) maka R □ m – 1

8) Jika (x > Data[m]) maka L □ m + 1
9) Jika (ketemu) maka m adalah indeks dari data yang dicari, jika tidak data tidak ditemukan

Contoh dari Binary Search, yaitu:



Gambar 2. Ilustrasi Binary Search

- Terdapat sebuah array yang menampung 7 elemen seperti ilustrasi di atas. Nilai yang akan dicari pada array tersebut adalah 13.
- Jadi karena konsep dari binary search ini adalah membagi array menjadi dua bagian, maka pertama tama kita cari nilai tengahnya dulu, total elemen dibagi 2 yaitu 7/2 = 4.5 dan kita bulatkan jadi 4.
- Maka elemen ke empat pada array adalah nilai tengahnya, yaitu angka 9 pada indeks ke 3.
- Kemudian kita cek apakah 13 > 9 atau 13 < 9?
- 13 lebih besar dari 9, maka kemungkinan besar angka 13 berada setelah 9 atau di sebelah kanan. Selanjutnya kita cari ke kanan dan kita dapat mengabaikan elemen yang ada di kiri.
- Setelah itu kita cari lagi nilai tengahnya, didapatlah angka 14 sebagai nilai tengah. Lalu, kita bandingkan apakah 13 > 14 atau 13 < 14?
- Ternyata 13 lebih kecil dari 14, maka selanjutnya kita cari ke kiri.
- Karna tersisa 1 elemen saja, maka elemen tersebut adalah nilai tengahnya. Setelah dicek ternyata elemen pada indeks ke-4 adalah elemen yang dicari, maka telah selesai proses pencariannya.

BAB III
GUIDED

1. **Guided 1**

Source code :

```
#include <iostream>
using namespace std;
int main()
{
    int arr[10] = {9, 4, 1, 7, 5, 12, 4, 13, 4, 10};
    int cari = 10;
    bool ketemu = false;
    int i;
    // algoritma Sequential Search
    for (i = 0; i < 10; i++)
    {
        if (arr[i] == cari)
        {
            ketemu = true;
            break;
        }
    }
    cout << "\t Program Sequential Search Sederhana\n " << endl;
    cout << " data: {9, 4, 1, 7, 5, 12, 4, 13, 4, 10}" << endl;

    if (ketemu)
    {
        cout << "\n angka " << cari << " ditemukan pada indeks ke - " << i <<
endl;
    }
    else
    {
        cout << cari << " tidak dapat ditemukan pada data." << endl;
    }
    return 0;
}
```

Output :

```
        Program Sequential Search Sederhana

 data: {9, 4, 1, 7, 5, 12, 4, 13, 4, 10}

 angka 10 ditemukan pada indeks ke - 9
```

Deskripsi program :

Program ini mendemonstrasikan algoritma pencarian sekuensial (`Sequential Search`) untuk menemukan nilai tertentu dalam sebuah array.

Langkah-langkah Algoritma:

1. **Deklarasi Variabel:**
    - `arr`: Array yang akan dicari (berisi 10 elemen integer).
    - `cari`: Nilai yang ingin dicari dalam array.
    - `ketemu`: Boolean flag untuk menandakan apakah nilai ditemukan (diinisialisasi `false`).
    - `i`: Variabel iterator untuk menjelajahi array.
2. **Perulangan Pencarian:**
    - Melakukan perulangan `for` sebanyak 10 kali (sesuai jumlah elemen array).
    - Di dalam perulangan:
        - Membandingkan nilai elemen array pada indeks `i` dengan nilai `cari`.
        - Jika sama, ubah flag `ketemu` menjadi `true` dan keluar dari perulangan menggunakan `break`.
3. **Menampilkan Hasil:**
    - Mencetak pesan "Program Sequential Search Sederhana".
    - Menampilkan isi array `data`.
    - Jika `ketemu` bernilai `true`:
        - Mencetak pesan bahwa nilai `cari` ditemukan pada indeks ke-`i`.
    - Jika `ketemu` bernilai `false`:
        - Mencetak pesan bahwa nilai `cari` tidak ditemukan dalam array.

2. Guided 2

Source code :

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

int data[7] = {1, 8, 2, 5, 4, 9, 7};
int cari;

void selection_sort() {
    int temp, min, i, j;
    for (i = 0; i < 7; i++) {
        min = i;
        for (j = i + 1; j < 7; j++) {
            if (data[j] < data[min]) {
                min = j;
```

```
            }
        }
        temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}

void binarysearch() {
    // searching
    int awal, akhir, tengah, b_flag = 0;
    awal = 0;
    akhir = 6; // Update the last index to 6 (0-based index)
    while (b_flag == 0 && awal <= akhir) {
        tengah = (awal + akhir) / 2;
        if (data[tengah] == cari) {
            b_flag = 1;
            break;
        } else if (data[tengah] < cari)
            awal = tengah + 1;
        else
            akhir = tengah - 1;
    }
    if (b_flag == 1)
        cout << "\n Data ditemukan pada index ke- " << tengah << endl;
    else
        cout << "\n Data tidak ditemukan\n";
}

int main() {
    cout << "\t BINARY SEARCH " << endl;
    cout << "\n Data : ";
    // tampilkan data awal
    for (int x = 0; x < 7; x++)
        cout << setw(3) << data[x];
    cout << endl;

    cout << "\n Masukkan data yang ingin Anda cari :";
    cin >> cari;

    cout << "\n Data diurutkan : ";
    // urutkan data dengan selection sort
    selection_sort();

    // tampilkan data setelah diurutkan
```

```
    for (int x = 0; x < 7; x++)
        cout << setw(3) << data[x];
    cout << endl;

    binarysearch();

    // Menunggu input user sebelum program berakhir
    cin.get();
    cin.get();

    return EXIT_SUCCESS;
}
```

Output :

```
        BINARY SEARCH

Data :   1  8  2  5  4  9  7

Masukkan data yang ingin Anda cari :5

Data diurutkan :   1  2  4  5  7  8  9

Data ditemukan pada index ke- 3
```

Deskripsi program :

Program ini mengimplementasikan algoritma **Binary Search** untuk mencari data dalam array yang sudah diurutkan sebelumnya menggunakan algoritma **Selection Sort**.

Berikut adalah langkah-langkah program:

1. **Memuat data:**
   - Program memuat 7 data integer ke dalam array `data`.
   - Data awal ditampilkan ke layar.
2. **Meminta data yang ingin dicari:**
   - Pengguna diminta untuk memasukkan data yang ingin dicari (`cari`).
3. **Mengurutkan data dengan Selection Sort:**
   - Algoritma Selection Sort digunakan untuk mengurutkan data dalam array `data`.
   - Data yang diurutkan ditampilkan ke layar.
4. **Melakukan Binary Search:**
   - Algoritma Binary Search digunakan untuk mencari data `cari` dalam array `data` yang sudah diurutkan.
   - Jika data ditemukan, indeksnya ditampilkan ke layar.
   - Jika data tidak ditemukan, pesan "Data tidak ditemukan" ditampilkan ke layar.
5. **Menunggu input user:**
   - Program menunggu input user sebelum program berakhir.

Penjelasan Algoritma:

- **Selection Sort:**
  - Algoritma ini bekerja dengan cara mengulang seluruh elemen dalam array dan menemukan elemen terkecil pada setiap iterasi.
  - Elemen terkecil kemudian ditukar dengan elemen pada posisi awal iterasi.
  - Proses ini diulang hingga seluruh elemen dalam array terurut.
- **Binary Search:**
  - Algoritma ini bekerja dengan cara membagi array menjadi dua bagian secara berulang.
  - Pada setiap iterasi, program membandingkan data yang dicari dengan elemen di tengah array.
  - Jika data yang dicari lebih kecil dari elemen tengah, program membagi bagian kiri array menjadi dua bagian.
  - Jika data yang dicari lebih besar dari elemen tengah, program membagi bagian kanan array menjadi dua bagian.
  - Proses ini diulang hingga data yang dicari ditemukan atau seluruh array telah dibagi menjadi elemen tunggal.

BAB IV
UNGUIDED

1. Unguided 1

Buatlah sebuah program untuk mencari sebuah huruf pada sebuah kalimat yang sudah di input dengan menggunakan Binary Search!

Source code :

```
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

// Fungsi binary search untuk mencari huruf dalam string yang sudah diurutkan
int binarySearch(const string &str, char target) {
    int left = 0;
    int right = str.length() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (str[mid] == target) {
            return mid; // huruf ditemukan
        } else if (str[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // huruf tidak ditemukan
}

int main() {
    string kalimat;
    char huruf;

    // Input kalimat
    cout << "Masukkan kalimat: ";
    getline(cin, kalimat);

    // Input huruf yang dicari
    cout << "Masukkan huruf yang ingin dicari: ";
    cin >> huruf;
```

```
    // Membuat salinan dari kalimat untuk diurutkan
    string sortedKalimat = kalimat;

    // Mengurutkan salinan kalimat
    sort(sortedKalimat.begin(), sortedKalimat.end());

    // Melakukan binary search pada kalimat yang sudah diurutkan
    int posisi = binarySearch(sortedKalimat, huruf);

    // Mengecek apakah huruf ditemukan atau tidak
    if (posisi != -1) {
        // Temukan posisi huruf di string asli
        int asliPosisi = kalimat.find(huruf);
        cout << "Huruf \"" << huruf << "\" ditemukan pada posisi " <<
asliPosisi + 1 << " dari kalimat." << endl;
    } else {
        cout << "Huruf \"" << huruf << "\" tidak ditemukan dalam kalimat." <<
endl;
    }

    return 0;
}
```

Output :

```
Masukkan kalimat: syafanida khakiki
Masukkan huruf yang ingin dicari: f
Huruf "f" ditemukan pada posisi 4 dari kalimat.
```

Deskripsi Program :

Program ini mengimplementasikan algoritma **Binary Search** untuk mencari huruf dalam string yang sudah diurutkan.

Berikut adalah langkah-langkah program:

1. **Meminta input:**
    - Pengguna diminta untuk memasukkan kalimat.
    - Pengguna diminta untuk memasukkan huruf yang ingin dicari.
2. **Membuat salinan kalimat:**
    - Salinan kalimat dibuat dari kalimat asli.
3. **Mengurutkan salinan kalimat:**
    - Algoritma `sort` dari STL (Standard Template Library) digunakan untuk mengurutkan salinan kalimat.
4. **Melakukan Binary Search:**

- Algoritma `binarySearch` digunakan untuk mencari huruf yang dicari dalam salinan kalimat yang sudah diurutkan.
- Algoritma ini bekerja dengan cara membagi string menjadi dua bagian secara berulang.
- Pada setiap iterasi, program membandingkan huruf yang dicari dengan huruf di tengah string.
- Jika huruf yang dicari ditemukan, program mengembalikan indeksnya.
- Jika huruf yang dicari tidak ditemukan, program mengembalikan -1.

5. **Mengecek hasil:**
   - Jika huruf yang dicari ditemukan, program menghitung posisinya dalam string asli dan menampilkannya ke layar.
   - Jika huruf yang dicari tidak ditemukan, program menampilkan pesan "Huruf tidak ditemukan" ke layar.

Penjelasan Algoritma:

- **Binary Search:**
  - Algoritma ini hanya bekerja pada data yang sudah diurutkan.
  - Algoritma ini bekerja dengan cara membagi data menjadi dua bagian secara berulang.
  - Pada setiap iterasi, program membandingkan data yang dicari dengan elemen di tengah data.
  - Jika data yang dicari lebih kecil dari elemen tengah, program membagi bagian kiri data menjadi dua bagian.
  - Jika data yang dicari lebih besar dari elemen tengah, program membagi bagian kanan data menjadi dua bagian.
  - Proses ini diulang hingga data yang dicari ditemukan atau seluruh data telah dibagi menjadi elemen tunggal.

2. Unguided 2

Buatlah sebuah program yang dapat menghitung banyaknya huruf vocal dalam sebuah kalimat!

Source Code :

```
#include <iostream>
#include <string>
#include <cctype> // Untuk fungsi isalpha dan tolower
using namespace std;
// Fungsi untuk menghitung jumlah huruf vokal dalam sebuah kalimat
int hitungVokal(const string &kalimat)
{
    int jumlahVokal = 0;
    for (char huruf : kalimat)
    {
        // Mengonversi huruf menjadi huruf kecil
        char lowerHuruf = tolower(huruf);
```

```
        // Memeriksa apakah huruf merupakan huruf vokal
        if (lowerHuruf == 'a' || lowerHuruf == 'e' || lowerHuruf == 'i' ||
            lowerHuruf == 'o' || lowerHuruf == 'u')
        {
            jumlahVokal++;
        }
    }
    return jumlahVokal;
}
int main()
{
    string kalimat;
    // Input kalimat dari pengguna
    cout << "Masukkan sebuah kalimat: ";
    getline(cin, kalimat);
    // Menghitung jumlah huruf vokal dalam kalimat
    int jumlahVokal = hitungVokal(kalimat);
    // Menampilkan hasil
    cout << "Jumlah huruf vokal dalam kalimat adalah: " << jumlahVokal <<
endl;
    return 0;
}
```

Output :

Deskripsi Program :

Program ini menghitung jumlah huruf vokal dalam sebuah kalimat yang dimasukkan oleh pengguna.

Langkah-langkah program:

1. **Meminta input kalimat:**
    - Pengguna diminta untuk memasukkan sebuah kalimat melalui `getline(cin, kalimat)`.
2. **Memanggil fungsi `hitungVokal`:**
    - Fungsi `hitungVokal` didefinisikan untuk menghitung jumlah huruf vokal dalam sebuah string.
    - Fungsi ini menerima string `kalimat` sebagai parameter.
    - Fungsi ini mengembalikan nilai integer yang menunjukkan jumlah huruf vokal dalam string.
    - Fungsi ini menggunakan loop `for` untuk mengiterasi setiap karakter dalam string.
    - Di dalam loop, fungsi ini mengonversi karakter menjadi huruf kecil menggunakan `tolower(huruf)`.

    - Fungsi ini kemudian memeriksa apakah karakter tersebut merupakan huruf vokal ('a', 'e', 'i', 'o', 'u').
    - Jika ya, variabel `jumlahVokal` diincrement.
3. **Menampilkan hasil:**
    - Hasil perhitungan jumlah huruf vokal (`jumlahVokal`) ditampilkan ke layar dengan `cout << "Jumlah huruf vokal dalam kalimat adalah: " << jumlahVokal << endl;`.

Penjelasan fungsi `hitungVokal`:

- Fungsi `hitungVokal` menerima string `kalimat` sebagai parameter.
- Fungsi ini menginisialisasi variabel `jumlahVokal` dengan 0.
- Fungsi ini menggunakan loop `for` untuk mengiterasi setiap karakter dalam string `kalimat`.
    - Di dalam loop, fungsi ini mengonversi karakter menjadi huruf kecil menggunakan `tolower(huruf)`.
    - Fungsi ini kemudian menggunakan pernyataan `if` untuk memeriksa apakah karakter tersebut merupakan huruf vokal ('a', 'e', 'i', 'o', 'u').
        - Jika ya, variabel `jumlahVokal` diincrement.
- Fungsi ini mengembalikan nilai integer `jumlahVokal`.

3. Unguided 3

Diketahui data = 9, 4, 1, 4, 7, 10, 5, 4, 12, 4. Hitunglah berapa banyak angka 4 dengan menggunakan algoritma Sequential Search!

Source Code :

```
#include <iostream>
using namespace std;

// Fungsi untuk mencari jumlah kemunculan suatu angka dalam array dengan
Sequential Search
int hitungAngka(const int data[], int ukuran, int angka)
{
    int jumlah = 0;
    for (int i = 0; i < ukuran; ++i)
    {
        if (data[i] == angka)
        {
            jumlah++;
        }
    }
    return jumlah;
}

int main()
{
```

```
    const int ukuran = 10;
    int data[ukuran] = {9, 4, 1, 4, 7, 10, 5, 4, 12, 4};
    int angkaYangDicari = 4;

    // Menghitung jumlah kemunculan angka 4 dalam data menggunakan Sequential
Search
    int jumlahAngka4 = hitungAngka(data, ukuran, angkaYangDicari);

    // Menampilkan hasil
    cout << "Jumlah angka 4 dalam data adalah: " << jumlahAngka4 << endl;

    return 0;
}
```

Output :

```
.cpp -o Unguided3_Modul8_Search } ; if (
Jumlah angka 4 dalam data adalah: 4
```

Deskripsi Program :

Program ini menghitung jumlah kemunculan suatu angka dalam array menggunakan algoritma **Sequential Search**.

Langkah-langkah program:

1. **Mendefinisikan data:**
   - Program mendefinisikan array `data` berukuran 10 dengan nilai awal `data = {9, 4, 1, 4, 7, 10, 5, 4, 12, 4}`.
   - Variabel `angkaYangDicari` diinisialisasi dengan nilai 4.
2. **Membuat fungsi `hitungAngka`:**
   - Fungsi `hitungAngka` menerima 3 parameter:
     - `data`: Array yang akan diiterasi.
     - `ukuran`: Ukuran array `data`.
     - `angka`: Angka yang ingin dicari jumlah kemunculannya.
   - Fungsi ini menginisialisasi variabel `jumlah` dengan 0.
   - Fungsi ini menggunakan loop `for` untuk mengiterasi setiap elemen dalam array `data`.
     - Di dalam loop, fungsi ini membandingkan elemen array dengan `angka`.
     - Jika elemen array sama dengan `angka`, variabel `jumlah` diincrement.
   - Fungsi ini mengembalikan nilai `jumlah`, yang menunjukkan jumlah kemunculan `angka` dalam array.
3. **Menghitung jumlah angka 4:**
   - Program memanggil fungsi `hitungAngka` dengan parameter:
     - `data`: Array `data`.

        - `ukuran`: Ukuran array `data` (10).
        - `angkaYangDicari`: Angka 4.
    - Hasil dari pemanggilan fungsi disimpan dalam variabel `jumlahAngka4`.
4. **Menampilkan hasil:**
    - Program menampilkan pesan "Jumlah angka 4 dalam data adalah: " diikuti dengan nilai `jumlahAngka4` ke layar.

Penjelasan Algoritma Sequential Search:

- Algoritma Sequential Search bekerja dengan cara mengiterasi setiap elemen dalam array dan membandingkannya dengan elemen yang dicari.
- Jika elemen yang dicari ditemukan, jumlah kemunculannya diincrement.
- Proses ini diulang hingga seluruh elemen dalam array telah diiterasi.

BAB IV

KESIMPULAN

Kesimpulan Mengenai Binary Searching dan Sequential Searching :

1. Sequential Searching

Kelebihan:

- **Sederhana dan Mudah Dipahami**: Sequential search adalah metode pencarian yang sangat sederhana, di mana setiap elemen dalam array diperiksa satu per satu sampai elemen yang dicari ditemukan atau sampai seluruh array telah diperiksa.
- **Tidak Membutuhkan Pengurutan:** Sequential search dapat digunakan pada data yang tidak terurut, sehingga tidak ada persiapan khusus yang diperlukan sebelum melakukan pencarian.
- **Fleksibilitas**: Dapat digunakan untuk berbagai tipe data dan struktur data yang berbeda.

Kekurangan:

- **Tidak Efisien untuk Dataset Besar**: Sequential search memiliki kompleksitas waktu O(n), yang berarti waktu pencarian bertambah secara linear seiring bertambahnya ukuran dataset. Ini membuat metode ini tidak efisien untuk dataset yang besar.
- **Waktu Pencarian Tidak Tetap**: Waktu yang diperlukan untuk menemukan elemen yang dicari bisa sangat bervariasi, terutama jika elemen tersebut berada di akhir array atau tidak ada di dalam array.

2. Binary Searching

Kelebihan:

- **Efisien untuk Dataset Besar**: Binary search memiliki kompleksitas waktu O(log n), yang berarti waktu pencarian bertambah secara logaritmik seiring bertambahnya ukuran dataset. Ini membuat metode ini jauh lebih efisien dibandingkan sequential search untuk dataset yang besar.
- **Waktu Pencarian Lebih Konsisten**: Waktu pencarian cenderung lebih konsisten dan dapat diprediksi karena proses pencarian selalu membagi dataset menjadi setengah pada setiap langkahnya.

Kekurangan:

- **Membutuhkan Data Terurut**: Binary search hanya bisa diterapkan pada data yang telah diurutkan. Oleh karena itu, ada langkah tambahan untuk mengurutkan data sebelum pencarian dapat dilakukan, yang bisa menambah overhead.
- **Implementasi Lebih Kompleks**: Implementasi binary search lebih kompleks dibandingkan sequential search, dan memerlukan pemahaman tentang pengindeksan dan pembagian data.

Perbandingan dan Penggunaan

Sequential Search cocok digunakan ketika:
- Data tidak terurut.
- Ukuran dataset kecil.
- Sederhana dan cepat untuk diimplementasikan.

Binary Search cocok digunakan ketika:
- Data sudah terurut atau dapat diurutkan dengan mudah.
- Ukuran dataset besar.
- Efisiensi dan kecepatan pencarian adalah prioritas utama.

DAFTAR PUSTAKA

Asisten Praktikum (2024). Search. Learning Management System.

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications.