

# **LAPORAN PRAKTIKUM**

## **MODUL 8 ALGORITMA SEARCHING**



**Disusun oleh:  
Nofita Fitriyani  
NIM: 2311102001**

**Dosen Pengampu:**  
Wahyu Andi Saputra, S.Pd., M.Eng

**PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO  
PURWOKERTO  
2024**

# **BAB I**

## **TUJUAN PRAKTIKUM**

- a. Menunjukkan beberapa algoritma dalam Pencarian.
- b. Menunjukkan bahwa pencarian merupakan suatu persoalan yang bisa diselesaikan dengan beberapa algoritma yang berbeda.
- c. Dapat memilih algoritma yang paling sesuai untuk menyelesaikan suatu permasalahan pemrograman.

## **BAB II**

### **DASAR TEORI**

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)  $i \leftarrow 0$
- 2)  $ketemu \leftarrow false$
- 3) Selama (tidak  $ketemu$ ) dan  $(i \leq N)$  kerjakan baris 4
- 4) Jika  $(Data[i] = x)$  maka  $ketemu \leftarrow true$ , jika tidak  $i \leftarrow i + 1$
- 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.

```

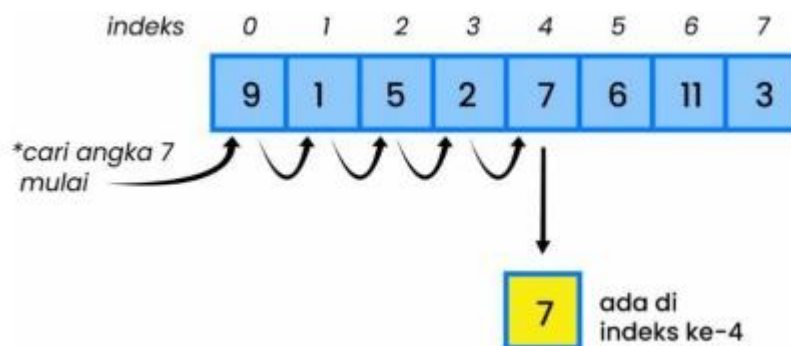
int SequentialSearch (int x)
{
    int i = 0;
    bool ketemu = false;
    while ((!ketemu) && (i < Max)){
        if (Data[i] == x)
            ketemu = true;
        else
            i++;
    }
    if (ketemu)
        return i;
    else
        return -1;
}

```

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 ( $L \leq R$ ) 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 n = 10;
    int data[n] = {9,4,1,7,5,12,4,13,4,10};
    int cari = 10;
    bool ketemu = false;
    int i;

    for (i = 0; i < n; i++){
        if(data[i] == cari){
            ketemu = true;
            break;
        }
    }

    cout << "Program Sequential Search" << endl;
    cout << "data : {9,4,1,7,5,12,4,13,4,10}" << endl;

    if (ketemu){
        cout << "\nAngka " << cari << " ditemukan pada indeks ke-" << i << endl;
    }else{
        cout << "data tidak ditemukan" << endl;
    }

    return 0;
```

```
}
```

### Screenshoot program

```
PS C:\Praktikum Struktur Data\Modul 8> cd "c:\Pra  
ed1 }  
Program Sequential Search  
data : {9,4,1,7,5,12,4,13,4,10}  
  
Angka 10 ditemukan pada indeks ke-9  
PS C:\Praktikum Struktur Data\Modul 8>
```

### Deskripsi program

Program tersebut adalah implementasi pencarian sekuensial (sequential search) dalam bahasa C++. Program ini mencari sebuah angka tertentu dalam sebuah array. Pertama, array data berisi 10 elemen diinisialisasi dengan nilai-nilai tertentu. Program kemudian mencari nilai cari (dalam hal ini 10) di dalam array tersebut. Variabel ketemu digunakan untuk menandai apakah nilai yang dicari ditemukan atau tidak. Dalam loop for, program memeriksa setiap elemen array hingga menemukan elemen yang sesuai dengan nilai yang dicari, kemudian menetapkan ketemu ke true dan keluar dari loop. Setelah pencarian selesai, program mencetak hasil pencarian, baik nilai ditemukan beserta indeksnya, atau pesan bahwa data tidak ditemukan.

## 2. Guided 2

```
#include<iostream>  
#include<conio.h>  
#include<iomanip>  
  
using namespace std;  
  
int dataArray[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(dataArray[j] < dataArray[min]){
                min = j;
            }
        }
        temp = dataArray[i];
        dataArray[i] = dataArray[min];
        dataArray[min] = temp;
    }
}

void BinarySearch(){
    int awal, akhir, tengah;
    bool b_flag = false;
    awal = 0;
    akhir = 6;
    while(!b_flag && awal <= akhir){
        tengah = (awal + akhir)/2;
        if(dataArray[tengah] == cari){
            b_flag = true;
        } else if(dataArray[tengah] < cari){
            awal = tengah + 1;
        } else {
            akhir = tengah - 1;
        }
    }
    if(b_flag){
        cout << "\nData ditemukan pada index ke-" << tengah <<
endl;
    } else {

```

```

        cout << "\nData tidak ditemukan" << endl;
    }
}

int main(){
    cout << "BINARY SEARCH" << endl;
    cout << "\nData : ";
    for(int x = 0; x < 7; x++){
        cout << setw(3) << dataArray[x];
    }
    cout << endl;

    cout << "Masukkan data yang ingin dicari : ";
    cin >> cari;

    cout << "\nData diurutkan : ";
    Selection_Sort();

    for(int x = 0; x < 7; x++){
        cout << setw(3) << dataArray[x];
    }
    cout << endl;
    BinarySearch();
    _getche();
    return 0;
}

```

### Screenshoot program

```

BINARY SEARCH

Data :   1  8  2  5  4  9  7
Masukkan data yang ingin dicari : 5

Data diurutkan :   1  2  4  5  7  8  9

Data ditemukan pada index ke-3

```

**Deskripsi program**

Program tersebut mengimplementasikan algoritma pencarian biner (binary search) pada sebuah array yang telah diurutkan menggunakan algoritma selection sort. Program ini dimulai dengan mendefinisikan sebuah array berisi tujuh elemen. Setelah itu, program meminta pengguna untuk memasukkan nilai yang ingin dicari. Program kemudian mengurutkan array menggunakan fungsi Selection\_Sort, yang menempatkan elemen terkecil di posisi awal array secara berurutan. Setelah array diurutkan, program menggunakan fungsi BinarySearch untuk mencari nilai yang dimasukkan pengguna. Algoritma binary search ini membagi array menjadi dua bagian, secara berulang-ulang mempersempit pencarian hingga nilai ditemukan atau jangkauan pencarian habis. Hasil pencarian akan dicetak ke layar, menunjukkan apakah nilai ditemukan beserta indeksnya atau menyatakan bahwa data tidak ditemukan.

## LATIHAN KELAS - 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 <cstring>

using namespace std;

void Selection_Sort(char arr[], int n) {
    int i, j, min_idx;
    char temp;
    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }
        temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
    }
}

bool BinarySearch(char arr[], int n, char cari, int &index) {
    int awal = 0, akhir = n - 1, tengah;
    while (awal <= akhir) {
        tengah = (awal + akhir) / 2;
        if (arr[tengah] == cari) {
            index = tengah;
        }
    }
}
```

```

        return true;
    } else if (arr[tengah] < cari) {
        awal = tengah + 1;
    } else {
        akhir = tengah - 1;
    }
}
return false;
}

int main() {
    string kalimat;
    char cari;

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

    cout << "Masukkan huruf yang ingin dicari: ";
    cin >> cari;

    int n = kalimat.length();
    char arr[n + 1];
    strcpy(arr, kalimat.c_str());

    Selection_Sort(arr, n);

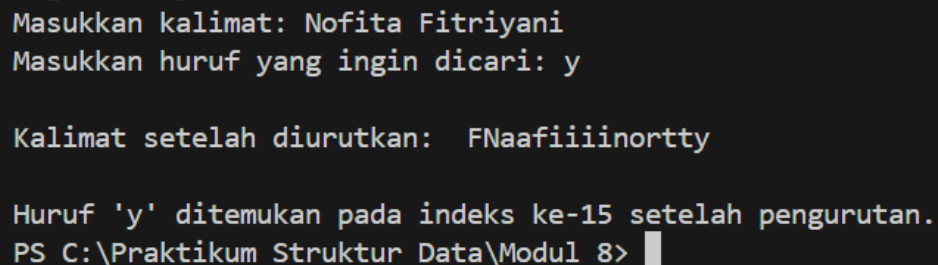
    cout << "\nKalimat setelah diurutkan: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i];
    }
    cout << endl;

    int index;
    if (BinarySearch(arr, n, cari, index)) {

```

```
        cout << "\nHuruf '" << cari << "' ditemukan pada indeks  
ke-" << index << " setelah pengurutan." << endl;  
    } else {  
        cout << "\nHuruf '" << cari << "' tidak ditemukan." <<  
endl;  
    }  
  
    return 0;  
}
```

### Screenshoot program



```
Masukkan kalimat: Nofita Fitriyani  
Masukkan huruf yang ingin dicari: y  
  
Kalimat setelah diurutkan: FNaafiiiinortty  
  
Huruf 'y' ditemukan pada indeks ke-15 setelah pengurutan.  
PS C:\Praktikum Struktur Data\Modul 8>
```

### Deskripsi program

Program di atas adalah implementasi dari algoritma pencarian biner pada sebuah kalimat yang telah diinput oleh pengguna, setelah diurutkan menggunakan algoritma selection sort. Program dimulai dengan meminta pengguna untuk memasukkan sebuah kalimat dan sebuah huruf yang ingin dicari. Kalimat tersebut kemudian dikonversi menjadi array karakter dan diurutkan dengan selection sort, yang mengurutkan elemen-elemen array berdasarkan urutan leksikografis. Setelah pengurutan, program mencetak kalimat yang telah diurutkan. Selanjutnya, program menggunakan algoritma binary search untuk mencari huruf yang diinput oleh pengguna dalam array yang sudah diurutkan. Jika huruf ditemukan, program mencetak indeks di mana huruf tersebut ditemukan; jika tidak, program mencetak pesan bahwa huruf tidak ditemukan. Output dari program mencakup kalimat yang diurutkan dan hasil pencarian huruf beserta indeksnya jika ditemukan.

## 2. Unguided 2

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

### Source code

```
#include <iostream>
#include <cctype>

using namespace std;

// Fungsi untuk memeriksa apakah karakter adalah huruf vokal
bool cekVokal(char c) {
    c = tolower(c);
    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');
}

// Fungsi untuk menghitung jumlah huruf vokal dalam string
int hitungVokal(const string& str) {
    int count = 0;
    for (char c : str) {
        if (cekVokal(c)) {
            count++;
        }
    }
    return count;
}

int main() {
    string kalimat;

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

    // Hitung jumlah huruf vokal dalam kalimat
```

```
int jumlahVokal = hitungVokal(kalimat);

// Tampilkan jumlah huruf vokal
cout << "Jumlah huruf vokal dalam kalimat: " << jumlahVokal
<< endl;

return 0;
}
```

## Screenshoot program

```
PS C:\Praktikum Struktur Data\Modul 8> cd "c:\Praktikum Struktur Data\Modul 8\Program Unguided2"
PS C:\Praktikum Struktur Data\Modul 8> .\Program Unguided2.exe
Masukkan sebuah kalimat: Nofita Fitriyani
Jumlah huruf vokal dalam kalimat: 7
PS C:\Praktikum Struktur Data\Modul 8>
```

## Deskripsi program

Program di atas adalah implementasi sederhana untuk menghitung jumlah huruf vokal dalam sebuah kalimat yang diinput oleh pengguna. Program dimulai dengan meminta pengguna untuk memasukkan sebuah kalimat. Fungsi cekVokal digunakan untuk memeriksa apakah sebuah karakter adalah huruf vokal (baik huruf besar maupun huruf kecil), dengan mengonversi karakter menjadi huruf kecil dan memeriksa apakah karakter tersebut adalah salah satu dari 'a', 'e', 'i', 'o', atau 'u'. Fungsi hitungVokal iterasi melalui setiap karakter dalam string yang diberikan dan menggunakan fungsi cekVokal untuk menghitung berapa kali huruf vokal muncul dalam kalimat tersebut. Hasil jumlah huruf vokal kemudian dicetak ke layar. Program ini efektif dalam menentukan jumlah huruf vokal dalam sebuah kalimat yang diinput oleh pengguna, dan memberikan hasil yang sesuai dengan kalimat yang dimasukkan.

### 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;

int hitungAngka(int data[], int n, int cari) {
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (data[i] == cari) {
            count++;
        }
    }
    return count;
}

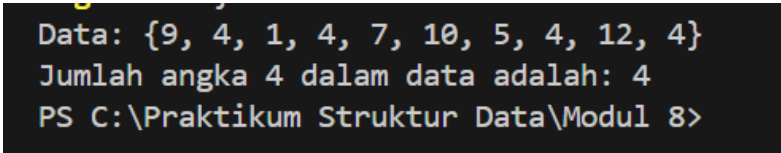
int main() {
    int data[] = {9, 4, 1, 4, 7, 10, 5, 4, 12, 4};
    int n = sizeof(data) / sizeof(data[0]);
    int cari = 4;

    int jumlah = hitungAngka(data, n, cari);

    cout << "Data: {9, 4, 1, 4, 7, 10, 5, 4, 12, 4}" << endl;
    cout << "Jumlah angka " << cari << " dalam data adalah: " <<
    jumlah << endl;

    return 0;
}
```

## Screenshoot program

A screenshot of a terminal window showing the output of the C++ program. The output consists of three lines: the first line displays the array data as {9, 4, 1, 4, 7, 10, 5, 4, 12, 4}; the second line shows the count of the number 4 in the array as 4; and the third line shows the command prompt path PS C:\Praktikum Struktur Data\Modul 8>.

```
Data: {9, 4, 1, 4, 7, 10, 5, 4, 12, 4}
Jumlah angka 4 dalam data adalah: 4
PS C:\Praktikum Struktur Data\Modul 8>
```

**Deskripsi program**

Program di atas adalah implementasi sederhana dari pencarian sekuensial (Sequential Search) untuk menghitung berapa banyak kemunculan angka tertentu dalam sebuah array. Program dimulai dengan mendefinisikan sebuah array data yang berisi sejumlah nilai. Fungsi `hitungAngka` menerima array, ukuran array, dan nilai yang akan dicari sebagai parameter. Fungsi ini kemudian iterasi melalui setiap elemen dalam array dan menghitung berapa kali nilai yang dicari muncul. Dalam fungsi `main`, ukuran array dihitung, dan nilai yang ingin dicari (`cari`) diatur ke 4. Fungsi `hitungAngka` dipanggil untuk menghitung jumlah kemunculan angka 4 dalam array data. Hasil perhitungan kemudian dicetak ke layar, menampilkan jumlah kemunculan angka 4 dalam array yang telah ditentukan. Program ini sederhana namun efektif dalam mencari dan menghitung kemunculan nilai tertentu dalam sebuah array.

## **BAB IV**

### **KESIMPULAN**

Sequential cocok digunakan untuk data yang tidak terurut atau hanya terdiri dari sejumlah kecil data. Binary Search cocok digunakan untuk data yang sudah terurut dan memiliki jumlah data yang cukup besar, karena memiliki kompleksitas waktu yang lebih baik dibandingkan Sequential Search. Pemilihan metode pencarian yang tepat akan sangat bergantung pada sifat data yang akan dicari, baik dari segi ukuran data, apakah sudah terurut atau tidak, serta kebutuhan efisiensi waktu dalam proses pencarian.

Kelemahan Sequential Search terletak pada efisiensi waktu yang kurang baik untuk data yang besar karena proses pencarian harus dilakukan secara linear, dengan jumlah iterasi sebanyak  $N$  elemen pada array. Kelemahan Binary Search terletak pada persyaratan data yang harus diurutkan terlebih dahulu dan kompleksitas algoritmanya untuk mengimplementasikan pembagian data menjadi dua.