ANALISIS ALGORITMA

Laporan Praktikum 2

Dibuat oleh:

Prayudha Adhitia Libramawan

(NPM: 140810180008)

KELAS B

Jurusan Teknik Informatika

Fakultas Matematika & Ilmu Pengetahuan Alam

Universitas Padjadjaran

Sumedang

2020

Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

Algoritma Pencarian Nilai Maksimal

procedure CariMaks(input $x_1, x_2, \ldots, x_n$: integer, output maks: integer)
{ Mencari elemen terbesar dari sekumpulan elemen larik integer $x_1, x_2, \ldots, x_n$. Elemen terbesar akan disimpan di dalam maks
Input: $x_1, x_2, \ldots, x_n$
Output: maks (nilai terbesar)
}

Deklarasi

i : integer

Algoritma

maks ← x_1
i ← 2
while i ≤ n do
if x_i > maks then
maks ← x_i
endif
i ← i + 1
endwhile

Jawaban Studi Kasus 1:

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

typedef int angka[5];

void cariMaks(angka &x, int &maks, int &n);

main()
{
    angka x;
    int n;
    int maks;
    cout << "Masukkan banyak angka : ";
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        cin >> x[i];
    }

    cariMaks(x, maks, n);
}

void cariMaks(angka &x, int &maks, int &n)
{
    maks = x[1];
    int i = 2;
    while (i <= n)
    {
        if (x[i] > maks)
        {
            maks = x[i];
```

```
        }
        i++;
    }
    cout << "Nilai maks : " << maks;
}
```

Kompleksitas waktu:

T(n) = 2(n-2) + (n-2) + 2

= 3n-4

Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots x_n$ yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian beruntun (*sequential search*). Algoritma *sequential search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```
procedure SequentialSearch(input x1, x2, ... xn : integer, y : integer, output idx : integer)
{   Mencari y di dalam elemen x1, x2, ... xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan, makai idx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
```

```
Deklarasi
        i : integer
        found : boolean {bernilai true jika y ditemukan atau false jika y tidak ditemukan}
Algoritma
        i ← 1
        found ← false
        while (i ≤ n) and (not found) do
            if xi = y then
                found ← true
            else
                i ← i + 1
            endif
        endwhile
        {i < n or found}

        If found then {y ditemukan}
                idx ← i
        else
                idx ← 0 {y tidak ditemukan}
        endif
```

Jawaban Studi Kasus 2:

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

main()
{
    int jumlah, search, A[100], index, jwb;
    bool found = false;
    cout << "SEQUENTIAL SEARCH";
    cout << "\nMasukan banyak data =  ";
    cin >> jumlah;

    for (int i = 0; i < jumlah; i++)
    {
        cout << "Data ke-" << i + 1 << " : ";
        cin >> A[i];
    }

    cout << "\nMasukan data yang dicari : ";
    cin >> search;
```

```
    cout << "\n====================================" << endl;

    for (int i = 0; i < jumlah; i++)
    {
        if (A[i] == search)
        {
            found = true;
            index = i;
            i = jumlah;
        }
    }
    if (found == true)
    {
        cout << "Data ditemukan pada data ke-" << index + 1;
    }
    else
    {
        cout << "Data tidak Ada!";
    }
    cout << "\n====================================" << endl;
    cout << "\n";
    system("pause");
}
```

1. Best Case
   Jika a1=x. $T_{min}(n) = 1$

2. Average
   Jika x ditemukan pada posisi ke-j, maka operasi perbandingan ($a_k = x$) akan dieksekusi sebanyak j kali.
   $T_{avg}(n) = \frac{(1+2+3+\cdots+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$

3. Worst Case
   Jika a_n = x atau x tidak ditemukan

Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots x_n$ yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian bagi dua (*binary search*). Algoritma *binary search* berikut menghasilkan indeks elemen yang bernilai sama dengan y. Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```
procedure BinarySearch(input x1, x2, ... xn : integer, x : integer, output : idx : integer)
{   Mencari y di dalam elemen x1, x2, ... xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan makai dx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
Deklarasi
        i, j, mid : integer
        found : Boolean
Algoritma
        i ← 1
        j ← n
        found ← false
        while (not found) and ( i ≤ j) do
                mid ← (i + j) div 2
                if xmid = y then
                    found ← true
                else
                    if xmid < y then
                        i ← mid + 1 else
                        j ← mid – 1 endif
        endif endwhile
        {found or i > j }
                If found then
                    Idx <- mid
                else
                    Idx <- o
                endif
```

Jawaban Studi Kasus 3:

```
using namespace std;
int main()
{
    int jumlah, i, array[100], search, first, last, mid;
    cout << "Binary Search"  << endl;
    cout << "Masukkan banyak data : ";
    cin >> jumlah;

    for (i = 0; i < jumlah; i++)
    {
        cout << "Data ke-" << i + 1 << " :";
        cin >> array[i];
    }
    cout << "\nMasukkan data yang di cari :";
    cin >> search;
    first = 0;
    last = jumlah - 1;
    cout << "\n==================================" << endl;
```

```
    while (first <= last)
    {
        mid = (first + last) / 2;
        if (array[mid] < search)
        {
            first = mid + 1;
        }
        else if (array[mid] == search)
        {
            cout << search << " ditemukan pada data ke-" << mid + 1 << "\n";
            break;
        }
        else
        {
            last = mid - 1;
        }
        mid = (first + last) / 2;
    }

    if (first > last)
    {
        cout << search << " Tidak Ditemukan!";
    }
    cout << "\n==================================" << endl;
    getch();
}
```

1. Best Case
   $T_{min}(n) = 1$

2. Average
   Jika terdapatnya index di awal atau akhir elemen

3. Worst Case
   $T_{max}(n) = {}^{2}\log n$

Studi Kasus 4: Insertion Sort

1. Buatlah program insertion sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure InsertionSort(input/output x1, x2, … xn : integer)
{   Mengurutkan elemen-elemen x1, x2, … xn dengan metode insertion sort.
    Input: x1, x2, … xn
    OutputL x1, x2, … xn (sudah terurut menaik)
}
Deklarasi
        i, j, insert : integer
Algoritma
        for i ← 2 to n do
            insert ← xi
            j ← i
            while (j < i) and (x[j-i] > insert) do
                x[j]← x[j-1]
                j←j-1
            endwhile
            x[j] = insert
        endfor
```

Jawaban Studi Kasus 4:

```
#include <iostream>
#include <conio.h>

using namespace std;

int data1[100], data2[100], jumlah;

void Insert_sort();

int main()
{
    cout << "Insertion Short" << endl;
    cout << "Masukkan Jumlah Data : ";
    cin >> jumlah;
    cout << endl;

    for (int i = 1; i <= jumlah; i++)
    {
        cout << "Masukkan data ke-" << i << " : ";
        cin >> data1[i];
        data2[i] = data1[i];
    }
    cout << "\n================================" << endl;
    Insert_sort();
    cout << "\nData Setelah di Urutkan : " << endl;

    for (int i = 1; i <= jumlah; i++)
    {
        cout << data1[i] << " ";
    }
    cout << "\n================================" << endl;
    getch();
}
```

```
void Insert_sort()
{
    int temp, i, j;
    for (i = 1; i <= jumlah; i++)
    {
        temp = data1[i];
        j = i - 1;
        while (data1[j] > temp && j >= 0)
        {
            data1[j + 1] = data1[j];
            j--;
        }
        data1[j + 1] = temp;
    }
}
```

1. Best Case
   Jika tidak ada looping dan array sudah terurut dengan benar

2. Average
   Jika saat array terutut setengahnya. Jumlah total iterasi loop sementara sama dengan jumlah inversi yang dimana jenis penyisipannya adalah O(n+f(n)) dan f(n) adalah jumlah inversi. Jadi jumlah inversi = jumlah penyisipan.

3. Worst Case
   Jika saat array terbalik urutannya. Dikarenakan bisa ada inversi n*(n-1)/2 . Jadi penyisipannya O(n2)

Studi Kasus 5: Selection Sort

1. Buatlah program selection sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma selection sort.

```
procedure SelectionSort(input/output x1, x2, … xn : integer)
{ Mengurutkan elemen-elemen x1, x2, … xn dengan metode selection sort.
    Input: x1, x2, … xn
    OutputL x1, x2, … xn (sudah terurut menaik)
}
Deklarasi
        i, j, imaks, temp : integer
Algoritma
        for i ← n downto 2 do {pass sebanyak n-1 kali}
            imaks ← 1
            for j ← 2 to i do
              if xj > ximaks then
                imaks ← j
              endif
            endfor
            {pertukarkan ximaks dengan xi}
            temp ← xi
            xi ← ximaks
            ximaks ← temp
        endfor
```

Jawaban Studi Kasus 5:

```cpp
#include <iostream>
#include <conio.h>

using namespace std;

int data1[100], data2[100];
int jumlah;

void trade(int s, int r);
void SelectionSort();

int main()
{
    cout << "Selection Short";
    cout << "\nMasukkan Jumlah Data Yang akan diinput : ";
    cin >> jumlah;
    for (int i = 1; i <= jumlah; i++)
    {
        cout << "Masukkan data ke-" << i << " : ";
        cin >> data1[i];
        data2[i] = data1[i];
    }

    SelectionSort();
    cout << "\n====================================" << endl;
    cout << "Data Setelah di Urutkan : " << endl;
    for (int i = 1; i <= jumlah; i++)
    {
        cout << " " << data1[i];
```

```
    }

    cout << "\n=====================================\n";
    getch();
}

void trade(int s, int r)
{
    int t;
    t = data1[r];
    data1[r] = data1[s];
    data1[s] = t;
}
void SelectionSort()
{
    int pos, i, j;
    for (i = 1; i <= jumlah - 1; i++)
    {
        pos = i;
        for (j = i + 1; j <= jumlah; j++)
        {
            if (data1[j] < data1[pos])
                pos = j;
        }
        if (pos != i)
            trade(pos, i);
    }
}
```

Jumlah operasi perbandingan element. Untuk setiap pass ke-i,
i = 1 -> jumlah perbandingan = n-1
i = 2 -> jumlah perbandingan = n-2
i = 3 -> jumlah perbandingan = n-3
i = k -> jumlah perbandingan = n-k
i = n-1 -> jumlah perbandingan = 1

Jumlah keseluruhan
T(n) = (n-1) + (n-2) + … + 1 = $\sum_{i=1}^{n-1} n - k = \frac{n(n-1)}{2}$
Dikarenakan algoritma sorting tidak bergantung pada batasan yang data inputnya sudah terurut atau acak. Maka diatas merupakan kompleksitas waktu untuk worst case dan best case.

Jumlah operasi pertukaran
Untuk setiap I dari 1 sampai n-1, terjadi satu kali pertukarana elemen. Sehingga jumlah operasi pertukaran keseluruhan yaitu T(n) = n-1

Jadi, algoritma pada case ini membutuhkan n(n-1)/2 operasi perbandinga nelemen dan n-1 operasi pertukaran.