**Tugas Analisis Algoritma**



Disusun oleh :

Dzakia Rayhana (140810160015)

Fidriyanto Rizkillah (140810160043)

Vega Savera Yuana (140810160053)

S-1 Teknik Informatika   
Fakultas Matematika & Ilmu Pengetahuan Alam   
Universitas Padjadjaran  
Jalan Raya Bandung - Sumedang Km. 21 Jatinangor 45363

1. **Membandingkan Algoritma Perpangkatan dengan Cara Iterasi dan Rekursif**
2. **Iterasi**

Algoritma:

1. Deklarasikan a dan x

2. Input bilangan yang akan dipangkatkan ke a dan jumlah pangkat ke x

3. Bila x != 0

Lakukan perhitungan hasil = hasil \* a

Lakukan penguranan –x

Ulang operasi hingga x = 0

Program:

#include <iostream>

#include <chrono>

using namespace std;

int main()

{

int x;

float a, hasil = 1;

cout << "Masukkan a: "; cin >> a;

cout << "Masukkan pangkat: "; cin >> x;

cout << a << "^" << x << "= ";

auto start = chrono::steady\_clock::now();

while (x != 0) {

hasil \*= a;

--x;

}

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << hasil << endl;

cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;

return 0;

}

1. **Rekursif**

Algoritma

Program

#include <iostream>

#include <math.h>

#include <chrono>

using namespace std;

//Deklarasi fungsi

double Power(double base, int eksponen);

int main()

{

double base, power;

int eksponen;

cout << "Masukkan Basis: "; cin >> base;

cout << "Masukkan Pangkat: "; cin >> eksponen;

auto start = chrono::steady\_clock::now();

power = Power(base, eksponen);

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << base << "^" << eksponen << " = " << power << endl;

cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;

return 0;

}

double Power(double base, int eksponen)

{

if(eksponen == 0)

return 1;

else if(eksponen > 0)

return base \* pow(base, eksponen - 1);

else

return 1 / pow(base, - eksponen);

}

1. **Perbandingan *running time* (perhitungan n pangkat n)**

**Analisis**

Rekursif memiliki running time yang lebih lama dibanding iterasi. Hal ini disebabkan karena membuat *multiple stack* terlebih dahulu sebelum melakukan perhitungan. Kompleksitas yang dimiliki kedua cara pun berbeda. Fungsi iterasi akan selalu diselesaikan dalam *y* langkah yang artinya kecepatan atau efiseinsi dari fungsi akan bergantung kepada y.

Sedangkan untuk rekursif, kompleksitas waktu diukur dari jumlah operasi perkalian dengan beberapa kemungkinan sebagai berikut:

* y = 0, maka pangkat sama dengan 1 (tidak ada operasi perkalian)
* y > 0, maka pangkat(x,y) = x \* pangkat(x, y-1) atau xy = x \* xy-1 (ada operasi perkalian).

Jumlah pemanggilan rekursif akan dipengaruhi oleh nilay y atau (y-1) atau T(n-1).

1. **Membandingkan Algoritma Pencarian dengan *Binary Search* dan *Linear Search***
2. **Linear Search**

Algoritma

1. Mulai.

2. Deklarasi konstanta max dengan nilai = besarnya array

3. Deklarasi array beserta nilai nilai setiap komponen array.

4. Deklarasi variabel target untuk menampung nilai yg dicari.

5. Minta input angka yg dicari/target dari user dan masukkan ke variable target.

6. Deklarasi integer result yg nilainya merupakan hasil pemanggilan fungsi linierSearch() dan parsing array, max, dan target sebagai parameter.

7. Proses di fungsi linierSearch()

Ulangi step dari i=0 sampai i<max

cek nilai array ke i = target

jika ya: return nilai i ke variabel result.

return nilai -1 (Penanda tidak ditemukan) ke variabel result.

8. Cek nilai result >= 0

jika ya: cetak bahwa target ditemukan di index ke i di array

jika tidak: cetak bahwa target tidak ditemukan di array

9. Stop

Program

#include<iostream>

#include <chrono>

using namespace std;

typedef int larik [];

void linearSearch(larik a, int n, int kunci, int& found, int& lokasi){

found = lokasi = 0;

while (!found && lokasi < n) {

if (a[lokasi] == kunci){

found = 1;

}

else {

lokasi=lokasi+1;

}

}

}

main() {

larik x = {0,1,2,3,4,5,6,7,8,9};

int n, kunci, found, lokasi;

cout << "Kunci Pencarian data : " ; cin >> kunci;

auto start = chrono::steady\_clock::now();

linearSearch(x, 10, kunci, found, lokasi);

if (found)

cout << "Ditemukan di posisi : " << lokasi+1 <<endl ;

else

cout << "Tidak ditemukan";

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;

}

1. **Binary Search**

Algoritma

1. Mulai.

2. Deklarasi kontanta max dengan nilai = besarnya array

3. Deklarasi array beserta nilai nilai setiap komponen array.

4. deklarasi variabel target untuk menampung nilai yg dicari.

5. Minta input angka yg dicari/target dari user dan masukkan ke variable target.

6. Deklarasi intejer result yg nilainya merupakan hasil pemanggilan fungsi binarySearch() dan parsing array, target, index pertama (0), dan index terakhir (max-1) sebagai parameter.

7. Proses di fungsi binarySearch()

cek apakah target < nilai array index pertama atau target > nilai array index terakhir

jika ya: return -1 (penanda tidak ditemukan) ke result

deklarasi variabel mid yg nilainya = {index pertama (0) + index terakhir (max-1)} / 2

cek apakah nilai target = nilai array index ke mid

jika ya: return nilai mid

jika tidak :

cek apakah nilai target < nilai array index ke mid

jika ya: panggil fungsi binarySearch dengan parameter array, target, low, mid-1, dan return nilai hasilnya.

jika tidak: panggil fungsi binarySearch dengan parameter array, target, mid+1, high dan return nilai hasilnya.

8. Cek nilai result >= 0

ya: cetak bahwa target ditemukan di index ke i di array

tidak: cetak bahwa target tidak ditemukan di array

9. Stop

Program:

#include <iostream>

#include <chrono>

using namespace std;

main () {

int n, i, search, first, last, middle;

int arr[] = {0,1,2,3,4,5,6,7,8,9};

cout << endl << "Masukkan angka yang akan dicari : "; cin >> search;

int posisi;

for (int i=0; i<10-1; i++) {

posisi=i;

for (int j=i+1;j<10;j++) {

if (arr[posisi]>arr[j]) {

posisi=j;

}

}

swap(arr[i], arr[posisi]);

}

auto start = chrono::steady\_clock::now();

first = 0;

last = 10-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

cout << search << " ditemukan di indeks ke " << middle+1 << endl;

break;

}

else

{

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

cout << "Error! " << search << " tidak ditemukan dalam list";

}

auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;

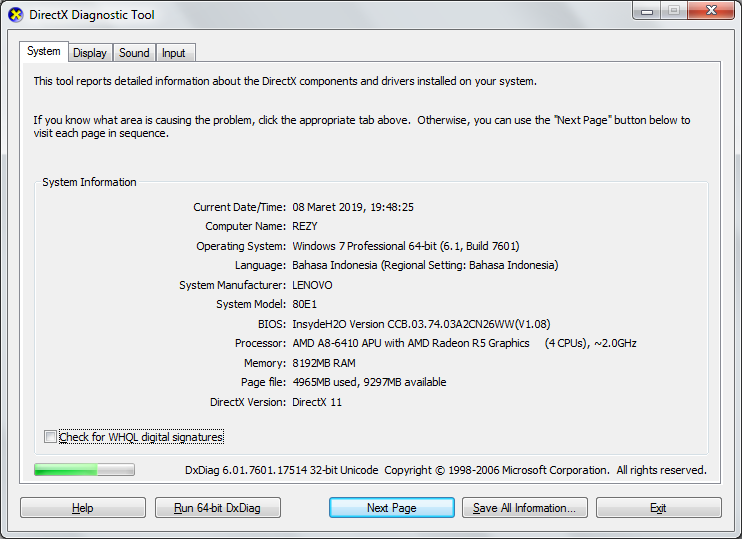
}

1. **Perbandingan *Running Time***

**Analisis**

Untuk mengukur *running time* dari kedua algoritma pencarian, kami melakukan pengujian sebanyak lima kali. Dan dari percobaan diataas, dapat disimpulkan bahwa algoritma *sequential search* memiliki kompleksitas waktu yang lebih besar dibandingkan dengan *binary search.*

1. **Spesifikasi Komputer**



1. **Kesimpulan**

Apabila kita melakukan dua operasi yang sama namun dengan algoritma yang berbeda, maka akan menghasilkan hasil yang sama dengan *running time* yang berbeda. Dapat dilihat pada percobaan diatas, algoritma *sequential search* memiliki kompleksitas waktu lebih besar dibandingkan dengan *binary search.* Serta algoritma perpangkatan dengan iterasi memiliki *running time* yang lebih cepat dibandingkan dengan algoritma rekursif.