Tugas 1

Analisis Algoritma

Disusun oleh :

Afifah Kho'eriah (140810160008)

Baby Cattleya Gustina Permatagama (140810160048)

Muhammad Islam Taufikurahman (140810160062)

S-1 Teknik Informatika

Fakultas Matematika & Ilmu Pengetahuan Alam

Universitas Padjadjaran

Jalan Raya Bandung - Sumedang Km. 21 Jatinangor 45363

A. Membandingkan Algoritma Perpangkatan dengan Cara Iterasi dan Rekursif

a. Cara Iterasi

Algoritma :

```
step 1 : deklasikan x dan y , kemudian masukkan bilangan yang akan di
pangkatkan ke x dan jumlah pangkat ke y
step 2 : hasil = x
         ulang step dari i=1 sampai i<y
             lakukan proses penghitungan dimana hasil = hasil*x
hasil= hasil*x
```

Program :

```
/*
   Nama Program : Program Menghitung Pangkat dengan looping while
   Oleh    : Baby, Afifah, Islam
   Dibuat : Senin, 25 Maret 2018
*/


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

int main()
{
    int exponent;
    float basis, hasil = 1;

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

    cout << "Masukkan pangka:  ";
    cin >> exponent;

    cout << basis << "^" << exponent << " = ";

    auto start = chrono::steady_clock::now();

    while (exponent != 0) {
        hasil *= basis;
        --exponent;
    }

    auto end = chrono::steady_clock::now();
    auto diff = end - start;

    cout << hasil << endl;
```

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

    return 0;
}
```

b. **Cara Rekursif**

Algoritma :

Function pangkat(input x,y:integer):integer
  if y = 0 then
        pangkat ß 1
  else
        pangkat ß pangkat(x,y-1)*x
  endif

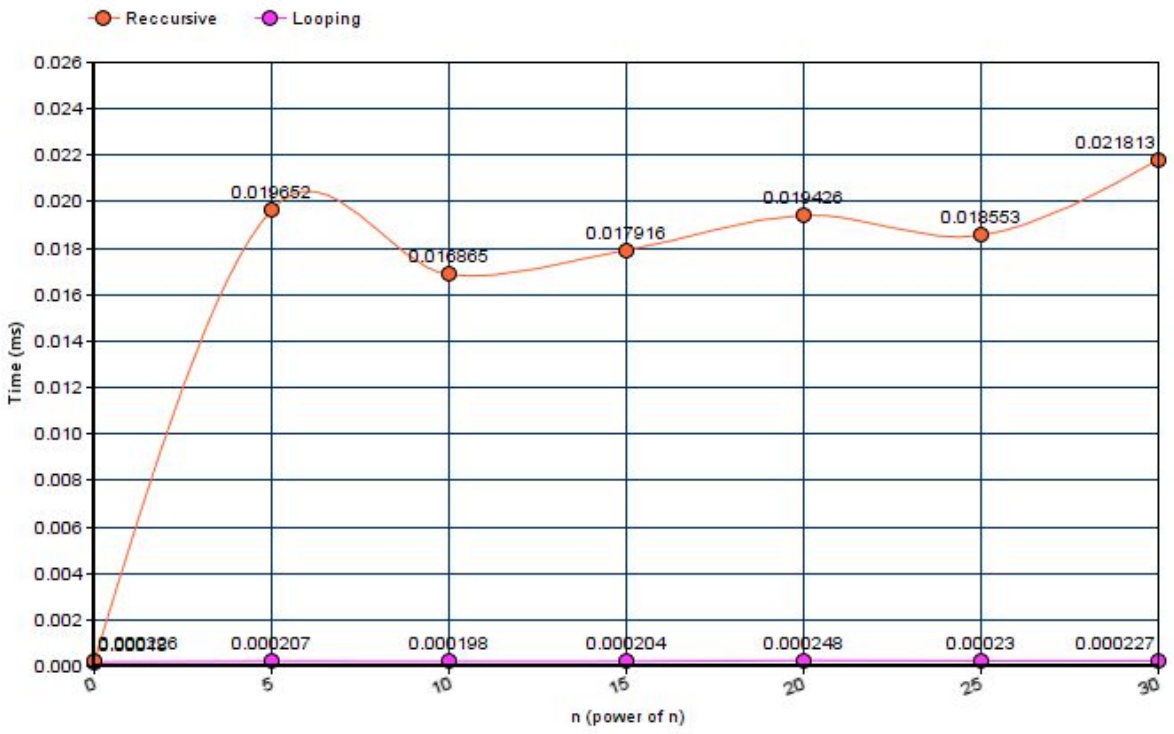
endfunction

Program :

```
/*
   Nama Program : Program Menghitung Pangkat dengan rekursif
   Oleh    : Baby, Afifah, Islam
   Dibuat : Senin, 25 Maret 2018
*/

#include <iostream>
#include <math.h>
#include <chrono>
using namespace std;

//function declaration
double Power(double base, int exponent);

int main()
{
    double base, power;
    int exponent;

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

    auto start = chrono::steady_clock::now();

    power = Power(base, exponent);

    auto end = chrono::steady_clock::now();

    auto diff = end - start;
```

```
    cout<<base<< "^"<<exponent<<" = "<<power<<endl;
    cout << chrono::duration <double, milli> (diff).count() << " ms"
<< endl;
    return 0;
}

double Power(double base, int exponent)
{
    if(exponent == 0)
        return 1;
    else if(exponent > 0)
        return base * pow(base, exponent - 1);
    else
        return 1 / pow(base, - exponent);
}
```

Hasil Perbandingan *running time* (perhitungan n pangkat n)



Analisis

Di sini rekursif lebih lambat dibandingkan iterasi, karena harus membuat *multiple stack* sebelum melakukan kalkulasi. Selain itu dari segi kompleksitas pun memiliki kompleksitas berbeda. Fungsi pangkat dengan iterasi akan selalu diselesaikan dalam y langkah. Dengan kata lain, kecepatan atau efisiensi dari fungsi pangkat bergantung kepada y.

Untuk rekursif, kompleksitas waktu diukur dari jumlah operasi perkalian (*)

- Jika y = 0 maka pangkat sama dengan 1 (tidak ada operasi perkalian) {basis}
- Jika y > 0 maka: pangkat(x,y) = x * pangkat(x,y-1) atau $x^y = x * x^{y-1}$ (ada operasi perkalian) {rekurens}

Yang mempengaruhi jumlah pemanggilan rekursif adalah nilai y atau (y-1) atau T(n-1)

B. Membandingkan Algoritma Pencarian dengan *Binary Search* dan *Linear Search*

a. *Linear Search*

Algoritma :

```
Step 0: Mulai.
Step 1: Deklarasi kontanta max dengan nilai = besarnya array
Step 2: Deklarasi array beserta nilai nilai setiap komponen array.
Step 3: deklarasi variabel target untuk menampung nilai yg dicari.
Step 4: Minta input angka yg dicari/target dari user dan masukkan ke
variable target.
Step 5: Deklarasi integer result yg nilainya merupakan hasil
pemanggilan fungsi linierSearch() dan parsing array, max, dan target
sebagai parameter.

Step 6: 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.

Step 7:      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
Step 8: Stop
```

Program :

```
/*
   Nama Program : Program Linear Search
   Oleh    : Baby, Afifah, Islam
   Dibuat : Senin, 25 Maret 2018
*/

#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;

}
```

*b. **Binary Search***

Algoritma :

```
Step 0: Mulai.
Step 1: Deklarasi kontanta max dengan nilai = besarnya array
Step 2: Deklarasi array beserta nilai nilai setiap komponen array.
Step 3: deklarasi variabel target untuk menampung nilai yg dicari.
Step 4: Minta input angka yg dicari/target dari user dan masukkan ke
variable target.
Step 5: Deklarasi intejer result yg nilainya merupakan hasil
pemanggilan fungsi binarySearch() dan parsing array, target, index
pertama (0), dan index terakhir (max-1) sebagai parameter.
Step 6: 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.

Step 7:     Cek nilai result >= 0
            ya: cetak bahwa target ditemukan di index ke i di array
            tidak: cetak bahwa target tidak ditemukan di array
Step 8: Stop
```

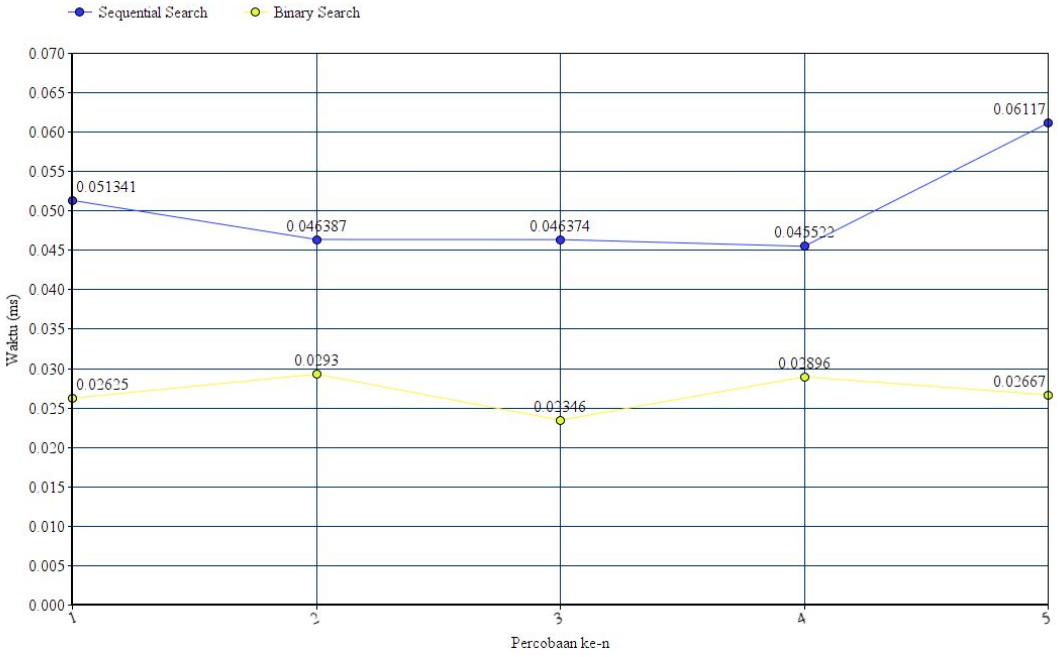
Program :

```
/*
  Nama Program : Program Binary Seacrh
  Oleh   : Baby, Afifah, Islam
  Dibuat : Senin, 25 Maret 2018
*/

#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;

}
```

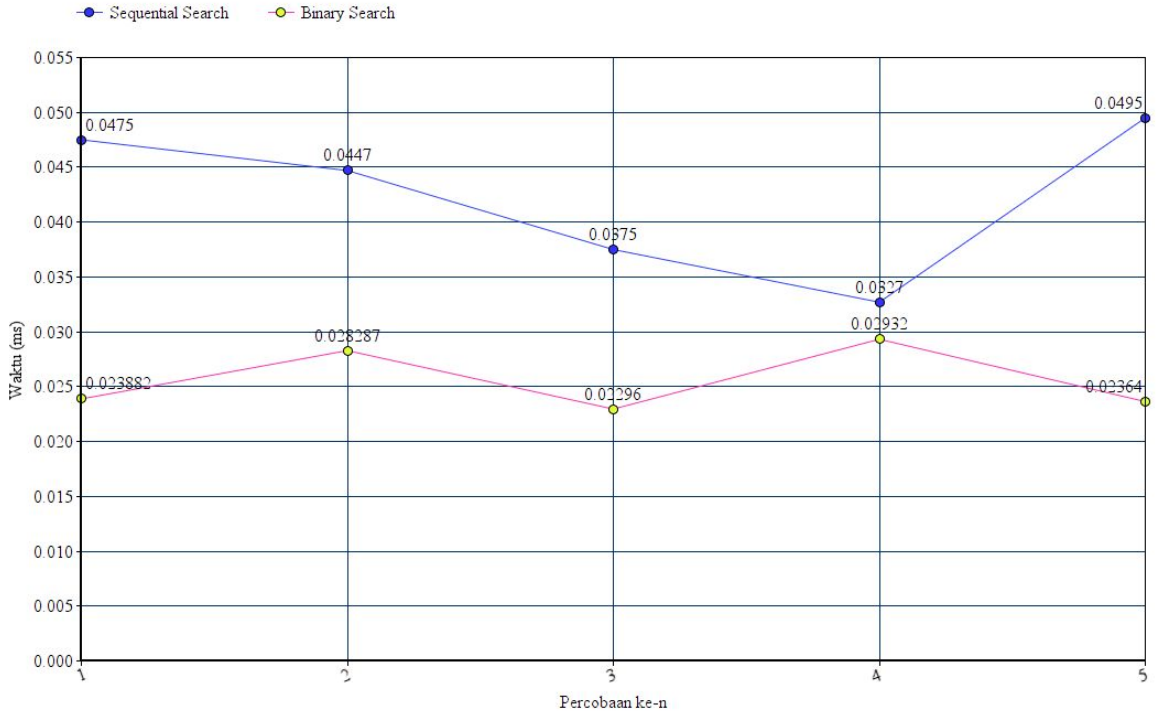
Hasil Perbandingan *running time*

Worst Case (data tidak ditemukan) :



Best Case :

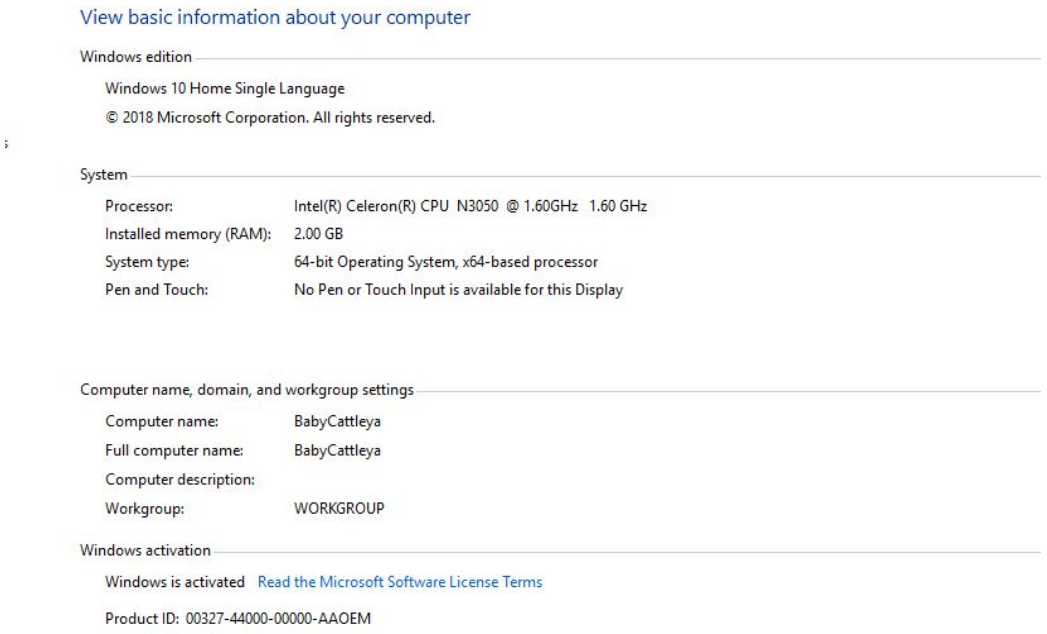
Data ada di tengah (*binary search)* & data di awal (*linear search)*



Analisis

Untuk mengukur lama waktu *sequential search* kami melakukan pengujian sebanyak lima kali. Yaitu menguji dalam keadaan *worst case*, dan *best case*. Untuk mengukur lama waktu *binary search* juga kami melakukan pengujian sebanyak lima kali. Yaitu menguji dalam keadaan *worst case*, dan *best case*. Dari percobaan dapat disimpulkan algoritma sequential Search memiliki kompleksitas waktu lebih besar dibanding dengan *binary search.*

C. Spesifikasi Komputer



D. Kesimpulan

Dari percobaan dengan menggunakan bahasa c++, dapat disimpulkan algoritma sequential Search memiliki kompleksitas waktu lebih besar dibanding dengan *binary search.* Serta algoritma perpangkatan dengan iterasi lebih cepat dibandingkan rekursif.