**TUGAS MODUL PRAKTIKUM 4**

**REKURENSI DAN PARADIGMA ALGORITMA DIVIDE & CONQUER**

**Untuk Memenuhi Tugas**

**Mata Kuliah Praktikum Analisis Algoritma**

**Disusun oleh :**

**Felia Sri Indriyani 140810170018**

**TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

# Pendahuluan

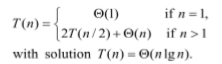
**PARADIGMA DIVIDE & CONQUER**

Divide & Conquer merupakan teknik algoritmik dengan cara memecah input menjadi beberapa bagian, memecahkan masalah di setiap bagian secara rekursif, dan kemudian menggabungkan solusi untuk subproblem ini menjadi solusi keseluruhan. Menganalisis running time dari algoritma divide & conquer umumnya melibatkan penyelesaian rekurensi yang membatasi running time secara rekursif pada instance yang lebih kecil

**PENGENALAN REKURENSI**

• Rekurensi adalah persamaan atau ketidaksetaraan yang menggambarkan fungsi terkait nilainya pada input yang lebih kecil. Ini adalah fungsi yang diekspresikan secara rekursif

• Ketika suatu algoritma berisi panggilan rekursif untuk dirinya sendiri, running time-nya sering dapat dijelaskan dengan perulangan

• Sebagai contoh, running time worst case () dari algoritma merge-sort dapat dideskripsikan dengan perulangan:

**BEDAH ALGORITMA MERGE-SORT**

 Merupakan algoritma sorting dengan paradigma divide & conquer

 Running time worst case-nya mempunyai laju pertumbuhan yang lebih rendah dibandingkan insertion sort  Karena kita berhadapan dengan banyak subproblem, kita notasikan setiap subproblem sebagai sorting sebuah subarray A[p..r]

 Inisialisasi, p=1 dan r=n, tetapi nilai ini berubah selama kita melakukan perulangan subproblem

Untuk mengurutkan A[p..r]:

• Divide dengan membagi input menjadi 2 subarray A[p..q] dan A[q+1 .. r]

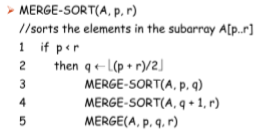
• Conquer dengan secara rekursif mengurutkan subarray A[p..q] dan A[q+1 .. r]

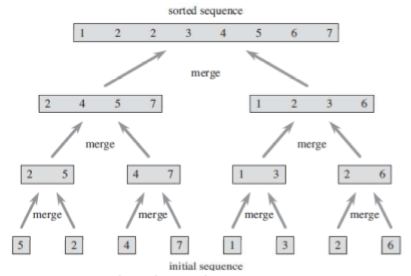
• Combine dengan menggabungkan 2 subarray terurut A[p..q] dan A[q+1 .. r] untuk menghasilkan 1 subarray terurut A[p..r]

• Untuk menyelesaikan langkah ini, kita membuat prosedur MERGE(A, p, q, r)

• Rekursi berhenti apabila subarray hanya memiliki 1 elemen (secara trivial terurut)

**PSEUDOCODE MERGE-SORT**





Gambar 1. Ilustrasi algoritma merge-sort

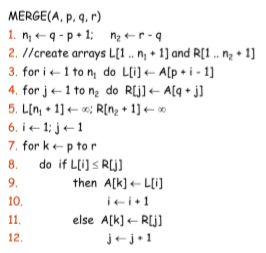
**PROSEDUR MERGE**

• Prosedur merge berikut mengasumsikan bahwa subarray A[p..q] dan A[q+1 .. r] berada pada kondisi terurut. Prosedur merge menggabungkan kedua subarray untuk membentuk 1 subarray terurut yang menggantikan array saat ini A[p..r] (input).

• Ini membutuhkan waktu Θ(n), dimana n = r-p+1 adalah jumlah yang digabungkan

• Untuk menyederhanakan code, digunakanlah elemen sentinel (dengan nilai ∞) untuk menghindari keharusan memeriksa apakah subarray kosong di setiap langkah dasar.

**PSEUDOCODE PROSEDUR MERGE**



**RUNNING TIME MERGE**

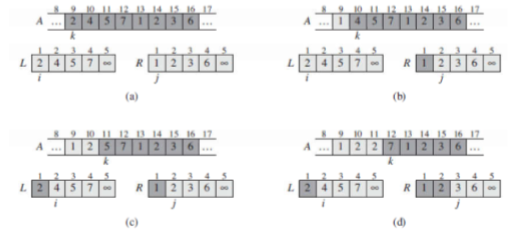
Untuk melihat running time prosedur MERGE berjalan di Θ(n), dimana n-p+1, perhatikan perulangan for pada baris ke 3 dan 4,

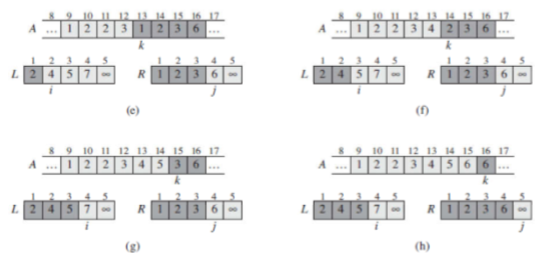


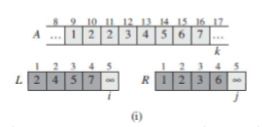
dan ada sejumlah n iterasi pada baris ke 8-12 yang membutuhkan waktu konstan.

**CONTOH SOAL MERGE-SORT**

MERGE(A, 9, 12, 16), dimana subarray A[9 .. 16] mengandung sekuen (2,4,5,7,1,2,3,6)







Algoritma merge-sort sangat mengikuti paradigma divide & conquer:

• Divide problem besar ke dalam beberapa subproblem

• Conquer subproblem dengan menyelesaikannya secara rekursif. Namun, apabila subproblem berukuran kecil, diselesaikan saja secara langsung.

• Combine solusi untuk subproblem ke dalam solusi untuk original problem

Gunakan sebuah persamaan rekurensi (umumnya sebuah perulangan) untuk mendeskripsikan running time dari algoritma berparadigma divide & conquer.

T(n) = running time dari sebuah algoritma berukuran n

• Jika ukuran problem cukup kecil (misalkan ≤, untuk nilai c konstan), kita mempunyai best case. Solusi brute-force membutuhkan waktu konstan Θ(1)

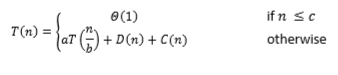
• Sebailknya, kita membagi input ke dalam sejumlah subproblem, setiap (1/) dari ukuran original problem (Pada merge sort ==2)

• Misalkan waktu yang dibutuhkan untuk membagi ke dalam n-ukuran problem adalah ()

• Ada sebanyak subproblem yang harus diselesaikan, setiap subproblem (/)  setiap subproblem membutuhkan waktu T(n/b) sehingga kita menghabiskan (/)

• Waktu untuk combine solusi kita misalkan ()

• Maka persamaan rekurensinya untuk divide & conquer adalah:



Setelah mendapatkan rekurensi dari sebuah algoritma divide & conquer, selanjutnya rekurensi harus diselesaikan untuk dapat menentukan kompleksitas waktu asimptotiknya. Penyelesaian rekurensi dapat menggunakan 3 cara yaitu, metode subtitusi, metode recursion-tree dan metode master. Ketiga metode ini dapat dilihat pada slide yang diberikan.

Studi Kasus

**Studi Kasus 1: MERGE SORT**

Setelah Anda mengetahui Algoritma Merge-Sort mengadopsi paradigma divide & conquer, lakukan Hal berikut:

1. Buat program Merge-Sort dengan bahasa C++

2. Kompleksitas waktu algoritma merge sort adalah O(n lg n). Cari tahu kecepatan komputer Anda dalam memproses program. Hitung berapa running time yang dibutuhkan apabila input untuk merge sort-nya adalah 20?

**Studi Kasus 2: SELECTION SORT**

Selection sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma selection sort, lakukan langkah-langkah berikut:

 Pelajari cara kerja algoritma selection sort

 Tentukan T(n) dari rekurensi (pengulangan) selection sort berdasarkan penentuan rekurensi divide & conquer:



 Selesaikan persamaan rekurensi T(n) dengan metode recursion-tree untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

 Lakukan implementasi koding program untuk algoritma selection sort dengan menggunakan bahasa C++

**Studi Kasus 3: INSERTION SORT**

Insertion sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma insertion sort, lakukan langkah-langkah berikut:

 Pelajari cara kerja algoritma insertion sort

 Tentukan T(n) dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:



 Selesaikan persamaan rekurensi T(n) dengan metode subtitusi untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

 Lakukan implementasi koding program untuk algoritma insertion sort dengan menggunakan bahasa C++

**Studi Kasus 4: BUBBLE SORT**

Bubble sort merupakan salah satu algoritma sorting yang berparadigma divide & conquer. Untuk membedah algoritma bubble sort, lakukan langkah-langkah berikut:

 Pelajari cara kerja algoritma bubble sort

 Tentukan T(n) dari rekurensi (pengulangan) insertion sort berdasarkan penentuan rekurensi divide & conquer:



 Selesaikan persamaan rekurensi T(n) dengan metode master untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ

 Lakukan implementasi koding program untuk algoritma bubble sort dengan menggunakan bahasa C++

# Teknik Pengumpulan

 Lakukan push ke github/gitlab untuk semua program dan laporan hasil analisa yang berisi jawaban dari pertanyaan-pertanyaan yang diajukan. Silahkan sepakati dengan asisten praktikum.

Penutup

 Ingat, berdasarkan Peraturan Rektor No 46 Tahun 2016 tentang Penyelenggaraan Pendidikan, mahasiswa wajib mengikuti praktikum 100%

 Apabila tidak hadir pada salah satu kegiatan praktikum segeralah minta tugas pengganti ke asisten praktikum

 Kurangnya kehadiran Anda di praktikum, memungkinkan nilai praktikum Anda tidak akan dimasukkan ke nilai mata kuliah.

Jawab :

1. Mergesort

Source code :

/\*

Nama program : Merge sort

Nama : Felia Sri Indriyani

NPM : 140810170018

\*/

#include <chrono>

#include<iostream>

using namespace std;

using namespace std::chrono;

void Merge(int A[],int p, int q,int r) /\*It merges two arrays \*/

{

int n1=q-p+1,i,j,k; /\*n1 is the size of L[]\*/

int n2=r-q; /\*n2 is the sixe of R[]\*/

int L[n1],R[n2];

for(i=0;i<n1;i++)

{

L[i]=A[p+i];

}

for(j=0;j<n2;j++)

{

R[j]=A[q+j+1];

}

i=0,j=0;

for(k=p;i<n1&&j<n2;k++)

{

if(L[i]<R[j])

{

A[k]=L[i++];

}

else

{

A[k]=R[j++];

}

}

while(i<n1) /\*If L[] has more elements than R[] then it will put all the reamining elements of L[] into A[]\*/

{

A[k++]=L[i++];

}

while(j<n2) /\*If R[] has more elements than L[] then it will put all the reamining elements of R[] into A[]\*/

{

A[k++]=R[j++];

}

}

void MergeSort(int A[],int p,int r) /\*It will will divide the array and sort them while merging them\*/

{

int q;

if(p<r)

{

q=(p+r)/2; /\*q is the middle element to divide the array\*/

MergeSort(A,p,q);

MergeSort(A,q+1,r);

Merge(A,p,q,r);

}

}

int main()

{

int A\_Size; /\*A\_Size size of A[]\*/

cout<<"Enter number of elements :";

cin>>A\_Size;

int A[A\_Size];

cout<<"Enter the array elements :";

for(int i=0;i<A\_Size;i++)

{

cin>>A[i];

}

// Use auto keyword to avoid typing long

// type definitions to get the timepoint

// at this instant use function now()

auto start = high\_resolution\_clock::now();

MergeSort(A,0,A\_Size-1);

// After function call

auto stop = high\_resolution\_clock::now();

// Get duration. Substart timepoints to

// get durarion. To cast it to proper unit

// use duration cast method

auto duration = duration\_cast<microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" << endl;

cout<<"The array elements after merge sort :";

for(int i=0;i<A\_Size;i++)

{

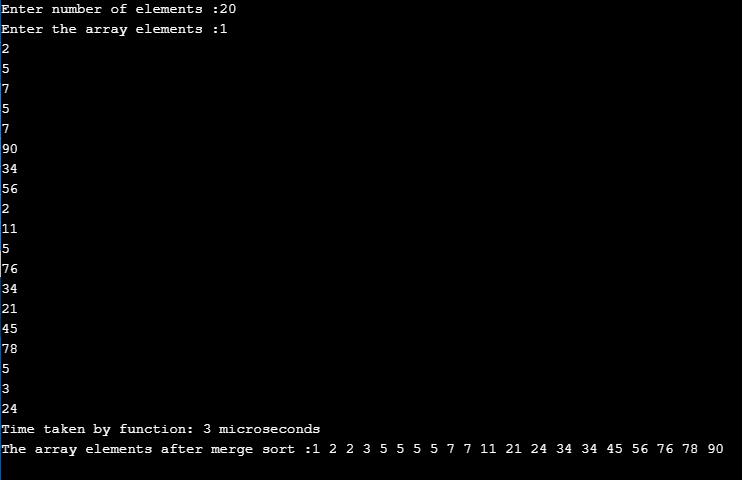
cout<<A[i]<<" ";

}

cout<<endl;

}

Screenshot :



Jika inputnya 20 makan running time nya 3 mocroseconds

Kompleksitas :

Kompleksitas Algoritma merge sort adalah O(n lg n).

1. Selection Sort

Source code :

/\*

Nama program : Bubble sort

Nama : Felia Sri Indriyani

NPM : 140810170018

\*/

#include <cstdlib>

#include <iostream>

#include <conio.h>

using namespace std;

int main ( )

{

int data [] = { 7, 3, 9, 2, 5}; //Kondisi awal:Unsorted list = data

int tmp; //Sorted list = kosong

cout<<"\tSELECTION SORT\n\n";

cout << "Data yang akan di urutkan \t:";

for( int i = 0; i < 5; i++)

cout << data[i] << " ";

cout << endl;

for (int i = 0; i < 5 -1; i++)

for (int j = i+1; j < 5; j++)

//1. Pengecekan dimulai data ke-1 sampai dengan data ke-n

if (data[i] > data[j])

{

//2. Tentukan bilangan dengan Index terkecil dari data bilangan tersebut

tmp = data[i];

//3. Tukar bilangan dengan Index terkecil tersebut dengan bilangan pertama ( I = 1 ) dari data bilangan tersebut

data[i] = data[j];

data[j] = tmp;

}

//4. Lakukan langkah 2 dan 3 untuk bilangan berikutnya ( I= I+1 ) sampai didapatkan urutan yg optimal.

cout<<"\nData yang telah di urutkan \t:";

for( int i = 0; i < 5; i++)

cout << data[i] << " ";

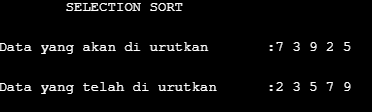
cout << endl;

getch();

return 0;

}

Screenshot:



Kompleksitas:

Menentukan T(n):

Oleh Karena itu:

Karena ,Maka )

1. Insertion Sort

Source code :

/\*

Nama program : Bubble sort

Nama : Felia Sri Indriyani

NPM : 140810170018

\*/

#include<iostream>

using namespace std;

int main()

{

int i,j,n,temp,a[30];

cout<<"Enter the number of elements:";

cin>>n;

cout<<"\nEnter the elements\n";

for(i=0;i<n;i++)

{

cin>>a[i];

}

for(i=1;i<=n-1;i++)

{

temp=a[i];

j=i-1;

while((temp<a[j])&&(j>=0))

{

a[j+1]=a[j]; //moves element forward

j=j-1;

}

a[j+1]=temp; //insert element in proper place

}

cout<<"\nSorted list is as follows by Insertion sort\n";

for(i=0;i<n;i++)

{

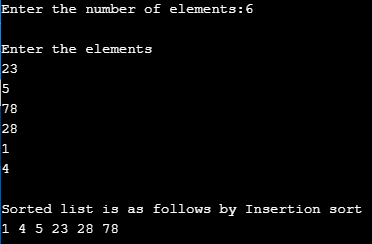
cout<<a[i]<<" ";

}

return 0;

}

Screenshot :



Kompleksitas :

Menentukan T(n):

T(n) =

1. Bubble sort

Source code :

/\*

Nama program : Bubble sort

Nama : Felia Sri Indriyani

NPM : 140810170018

\*/

#include<iostream>

using namespace std;

int main()

{

int a[50],n,i,j,temp;

cout<<"Enter the size of array: ";

cin>>n;

cout<<"Enter the array elements: ";

for(i=0;i<n;++i)

cin>>a[i];

for(i=1;i<n;++i)

{

for(j=0;j<(n-i);++j)

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

cout<<"Array after bubble sort:";

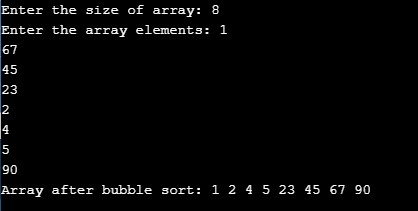
for(i=0;i<n;++i)

cout<<" "<<a[i];

return 0;

}

Screenshot :



Kompleksitas :

Menentukan T(n):