MODUL PRAKTIKUM 4
REKURENSI DAN PARADIGMA ALGORITMA DIVIDE & CONQUER

MATA KULIAH
ANALISIS ALGORITMA
D10G.4205 & D10K.0400601

PENGAJAR : (1) MIRA SURYANI, S.Pd., M.Kom
(2) INO SURYANA, Drs., M.Kom
(3) R. SUDRAJAT, Drs., M.Si
FAKULTAS : MIPA
SEMESTER : IV dan VI

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

MARET 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 T(n) dari algoritma merge-sort dapat dideskripsikan dengan perulangan:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$
$$\text{with solution } T(n) = \Theta(n \lg n).$$

BEDAH ALGORITMA MERGE-SORT

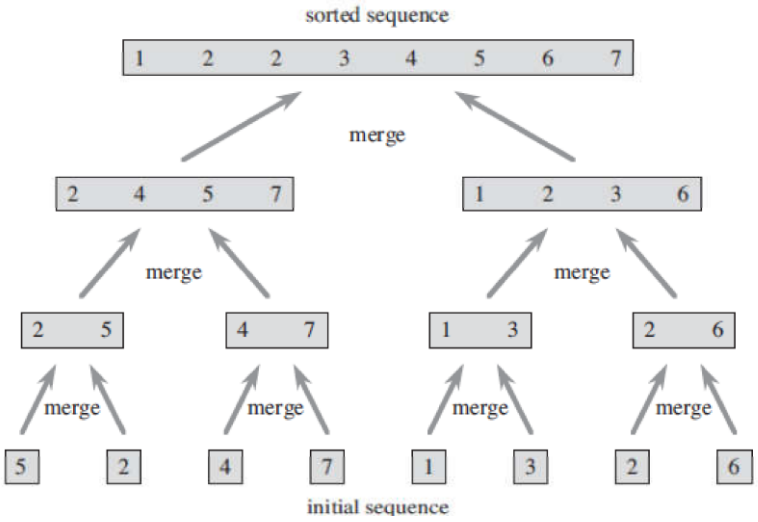
- Merupakan algoritma sorting dengan paradigma divide & conquer
- Running time worst case-nya mempunyai laju pertumbuhan yang lebih rendah dibandingkan insertion sort
- Karena kita berhadapan dengan banyak subproblem, kita notasikan setiap subproblem sebagai sorting sebuah subarray A[p..r]
- Inisialisasi,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

```
MERGE-SORT(A, p, r)
//sorts the elements in the subarray A[p..r]
1   if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```



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

```
MERGE(A, p, q, r)
1. n1 ← q - p + 1;    n2 ← r - q
2. //create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
3. for i ← 1 to n1  do  L[i] ← A[p + i - 1]
4. for j ← 1 to n2  do  R[j] ← A[q + j]
5. L[n1 + 1] ← ∞; R[n2 + 1] ← ∞
6. i ← 1; j ← 1
7. for k ← p to r
8.     do  if L[i] ≤ R[j]
9.             then  A[k] ← L[i]
10.                  i ← i + 1
11.            else  A[k] ← R[j]
12.                  j ← j + 1
```

RUNNING TIME MERGE

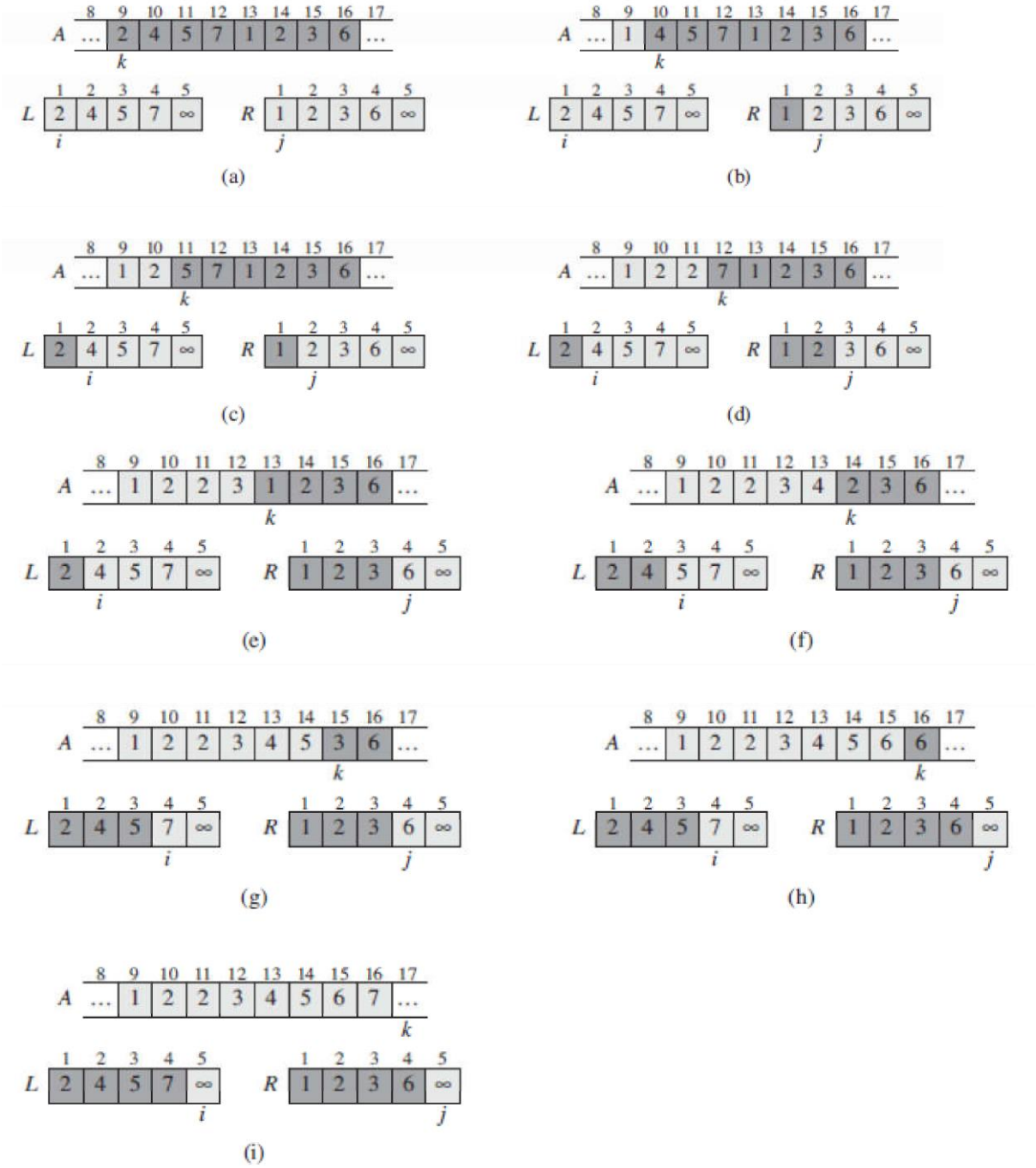
Untuk melihat running time prosedur MERGE berjalan di Θ(n), dimana n-p+1, perhatikan perulangan for pada baris ke 3 dan 4,

$$\Theta(n1 + n2) = \Theta(n)$$

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 $n \leq c$, untuk nilai c konstan), kita mempunyai *best case*. Solusi brute-force membutuhkan waktu konstan $\Theta(1)$
- Sebailknya, kita membagi input ke dalam sejumlah a subproblem, setiap (1/b) dari ukuran

original problem (Pada merge sort $a = b = 2$)

- Misalkan waktu yang dibutuhkan untuk membagi ke dalam n-ukuran problem adalah $D(n)$
- Ada sebanyak a subproblem yang harus diselesaikan, setiap subproblem (n/b) → setiap subproblem membutuhkan waktu T(n/b) sehingga kita menghabiskan $aT(n/b)$
- Waktu untuk **combine** solusi kita misalkan $C(n)$
- Maka persamaan **rekurensinya untuk divide & conquer** adalah:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

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

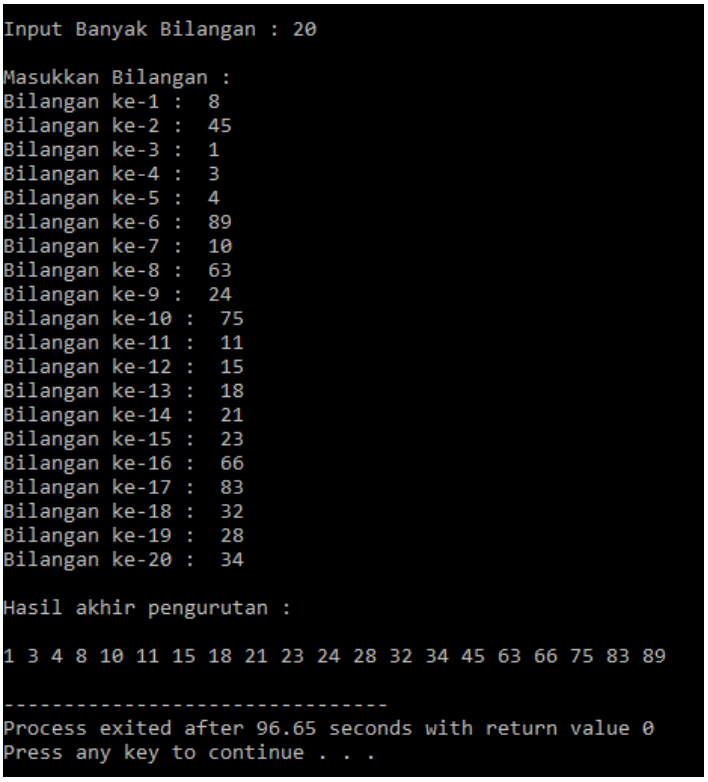
Studi Kasus 1: MERGE SORT

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?

```
/*
Nama : Sina Mustopa
NPM : 140810180017
Kelas : A
*/

#include <iostream>
using namespace std;

int a[100];
void merge(int,int,int);
void merge_sort(int low,int high)
        {       int mid;
                        if(low<high)
                {
                  mid=(low+high)/2;
                  merge_sort(low,mid);
                  merge_sort(mid+1,high);
                  merge(low,mid,high);
                 }
}

void merge(int low,int mid,int high){
          int h,i,j,b[50],k;
          h=low;
          i=low;
          j=mid+1;
          while((h<=mid)&&(j<=high))
          {
```

```
            if(a[h]<=a[j]){
                        b[i]=a[h]; h++;
                }
            else{
                        b[i]=a[j]; j++;
                } i++;
            }
            if(h>mid){
                for(k=j;k<=high;k++)
                        {
                          b[i]=a[k]; i++;
                                }
                        }
            else{
                        for(k=h;k<=mid;k++)
                                { b[i]=a[k]; i++;
                        }
                }
            for(k=low;k<=high;k++)
             a[k]=b[k];
             }

main(){
 int num,i;

 cout<<"--------------------------"<<endl;
 cout<<"    Merge Sort Program              "<<endl;
 cout<<"--------------------------"<<endl;
 cout<<endl;
 cout<<"Input Banyak Bilangan : ";cin>>num;
 cout<<endl;
 cout<<"Masukkan Bilangan :"<<endl;

 for(i=1;i<=num;i++){
  cout<<"Bilangan ke-"<<i<<" :  ";cin>>a[i] ;
 }
 merge_sort(1,num);
 cout<<endl<<"Hasil akhir pengurutan :"<<endl<<endl;
 for(i=1;i<=num;i++)
  cout<<a[i]<<" ";
 cout<<endl;
 }
```



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?

- Menggunakan perhitungan Big O -> $T(20 \log_{10} 20) = 26$

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:

$$T(n) = \begin{cases} ©(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

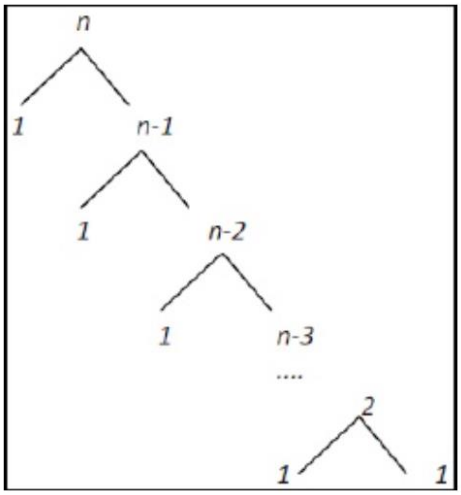
- Selesaikan persamaan rekurensi T(n) dengan **metode recursion-tree** untuk mendapatkan kompleksitas waktu asimptotiknya dalam Big-O, Big-Ω, dan Big-Θ
- Lakukan implementasi koding program untuk algoritma selection sort dengan menggunakan bahasa C++

Pengerjaan :

```
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
```

- Subproblem = 1
- Masalah setiap subproblem = n-1
- Waktu proses pembagian = n
- Waktu proses penggabungan = n

- $T(n) = \{\Theta(1)\, T(n-1) + \Theta(n)$



1. T(n) = cn + cn-c +cn-2c + ... + 2c +cn
 = c((n-1)(n-2)/2) + cn
 = c((n^2-3n+2)/2) + cn
 = c(n^2/2)-(3n/2)+1 + cn
 =O(n^2)

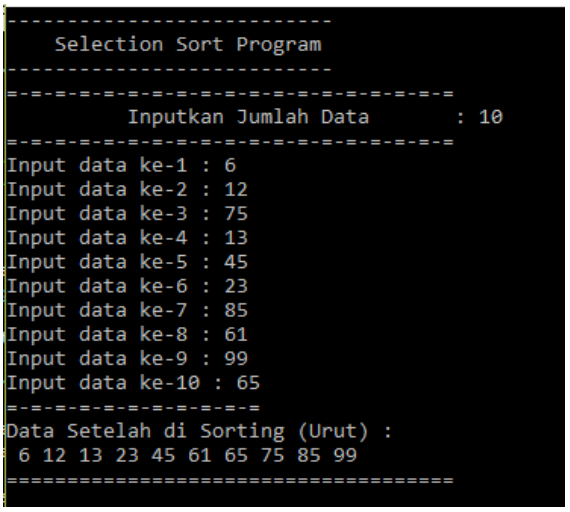
2. T(n) = cn + cn-c +cn-2c + ... + 2c +cn
 = c((n-1)(n-2)/2) + cn
 = c((n^2-3n+2)/2) + cn
 = c(n^2/2)-(3n/2)+1 + cn
 = Ω (n^2)

3. T(n) = cn^2
 = Θ(n^2)

Source Code :

```
/*
Nama : Sina Mustopa
NPM : 140810180017
Kelas : A
*/
#include <iostream>
#include<conio.h>

using namespace std;

int data[50],data2[50];
int n;

void tukar(int a, int b)
{
        int t;
        t = data[b];
        data[b] = data[a];
        data[a] = t;
}
void selection_sort()
{
        int pos,i,j;
        for(i=1;i<=n-1;i++)
        {
           pos = i;
           for(j = i+1;j<=n;j++)
           {
          if(data[j] < data[pos]) pos = j;
        }
    if(pos != i) tukar(pos,i);
  }
}

int main(){
 cout<<"---------------------------"<<endl;
 cout<<"    Selection Sort Program          "<<endl;
 cout<<"---------------------------"<<endl;
        cout << "=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-="<<endl;
        cout<<"        Inputkan Jumlah Data      : ";cin>>n;
        cout << "=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-="<< endl;
        for(int i=1;i<=n;i++)
        {
        cout<<"Input data ke-"<<i<<" : ";
        cin>>data[i];
        data2[i]=data[i];
        }

        selection_sort();
        cout << "=-=-=-=-=-=-=-=-=-=-=-=" << endl;
        cout<<"Data Setelah di Sorting (Urut) : "<<endl;
        for(int i=1; i<=n; i++)
        {
                cout<<" "<<data[i];
```

```
    }

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



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:

$$T(n) = \begin{cases} \text{©}(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

- 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++

Pengerjaan :

```
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
```

Subproblem = 1
Masalah setiap subproblem = n-1
Waktu proses penggabungan = n
Waktu proses pembagian = n

$T(n) = \{\Theta(1)\ T(n-1) + \Theta(n)$

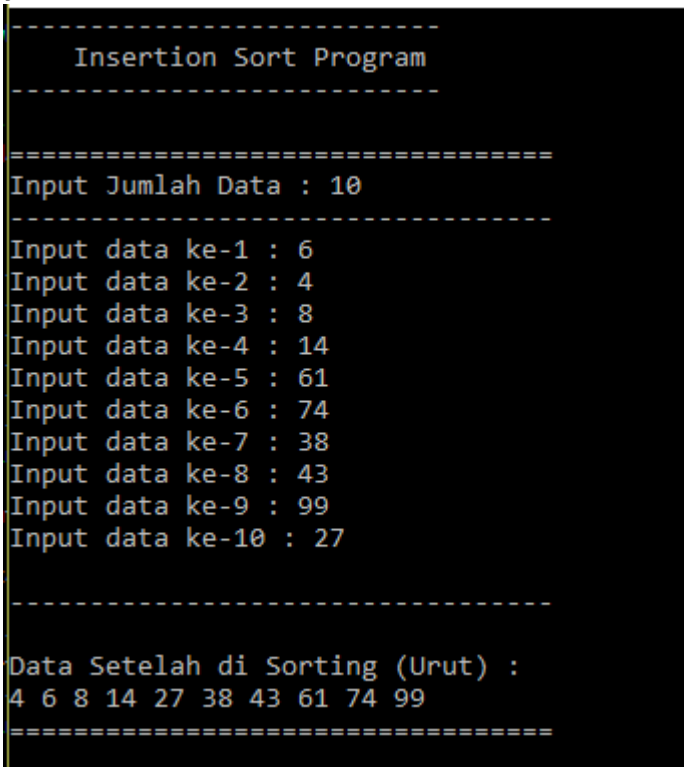
a. T(n) = cn + cn-c +cn-2c + ... + 2c +cn <= $2cn^2 + cn^2$
= c((n-1)(n-2)/2) + cn<= $2cn^2 + cn^2$
= c((n^2-3n+2)/2) + cn<= $2cn^2 + cn^2$
= c(n^2/2)-c(3n/2)+c+cn <= $2cn^2 + cn^2$
=O(n^2)

b. T(n) = cn <= cn
= Ω (n)

c. T(n) = (cn + cn^2)/n
= Θ(n)

```
/*
Nama : Sina Mustopa
NPM : 140810180017
Kelas : A
*/

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

using namespace std;

int data[50],data2[50],n;

void insertion_sort()
{
        int temp,i,j;
        for(i=1;i<=n;i++){
           temp = data[i];
                j = i -1;
           while(data[j]>temp && j>=0){
                        data[j+1] = data[j];
                   j--;
           }
           data[j+1] = temp;
        }
}
int main(){
         cout<<"--------------------------"<<endl;
         cout<<"    Insertion Sort Program        "<<endl;
         cout<<"--------------------------"<<endl;
        cout << "\n================================"<<endl;
        cout<<"Input Jumlah Data : "; cin>>n;
        cout << "---------------------------------" << endl;
        for(int i=1;i<=n;i++)
        {
          cout<<"Input data ke-"<<i<<" : ";
          cin>>data[i];
          data2[i]=data[i];
```

```
        }
        cout << "\n---------------------------------" << endl;
        insertion_sort();
        cout<<"\nData Setelah di Sorting (Urut) : "<<endl;
        for(int i=1; i<=n; i++)
        {
          cout<<data[i]<<" ";
        }
        cout << "\n=================================="<<endl;
        getch();
}
```



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:

$$T(n) = \begin{cases} ©(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

- 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++.

Pengerjaan :

Subproblem = 1
Masalah setiap subproblem = n-1
Waktu proses pembagian = n
Waktu proses penggabungan = n

$$T(n) = \{\Theta(1)\ T(n-1) + \Theta(n)$$

a. T(n) = cn + cn-c +cn-2c + ... + 2c +c <= $2cn^2 + cn^2$
   = c((n-1)(n-2)/2) + c<= $2cn^2 + cn^2$
   = c((n^2-3n+2)/2) + c<= $2cn^2 + cn^2$
   = c(n^2/2)-c(3n/2)+2c <= $2cn^2 + cn^2$
   =O(n^2)

b. T(n) = cn + cn-c +cn-2c + ... + 2c +c <= $2cn^2 + cn^2$
   = c((n-1)(n-2)/2) + c<= $2cn^2 + cn^2$
   = c((n^2-3n+2)/2) + c<= $2cn^2 + cn^2$
   = c(n^2/2)-c(3n/2)+2c <= $2cn^2 + cn^2$
   = Ω (n^2)

c. T(n) = $cn^2 + cn^2$
   = Θ(n^2)

```
/*
Nama : Sina Mustopa
NPM : 140810180017
Kelas : A
*/

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

using namespace std;


int main(){
        int arr[100],n,temp;

        cout<<"--------------------------"<<endl;
```

```
    cout<<"    Bubble Sort Program          "<<endl;
    cout<<"--------------------------"<<endl;
    cout<<"=============================="<<endl;
    cout<<"Banyak Elemen Input :       ";cin>>n;
    cout<<"------------------------------" <<endl;

    for(int i=0;i<n;++i){
        cout<<"Inputkan Elemen ke-"<<i+1<<" : ";cin>>arr[i];
    }

    for(int i=1;i<n;i++){
        for(int j=0;j<(n-1);j++){
            if(arr[j]>arr[j+1]){
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
    cout << "-----------------------------------------------" << endl;
    cout<<"\nHasil dari Bubble Sort (Urut) : "<<endl;
    for(int i=0;i<n;i++){
        cout<<" "<<arr[i];
    }
}
```

```
--------------------------
    Bubble Sort Program
--------------------------
==============================
Banyak Elemen Input :        10
------------------------------
Inputkan Elemen ke-1 : 9
Inputkan Elemen ke-2 : 41
Inputkan Elemen ke-3 : 66
Inputkan Elemen ke-4 : 46
Inputkan Elemen ke-5 : 82
Inputkan Elemen ke-6 : 61
Inputkan Elemen ke-7 : 31
Inputkan Elemen ke-8 : 24
Inputkan Elemen ke-9 : 11
Inputkan Elemen ke-10 : 4
-----------------------------------------------

Hasil dari Bubble Sort (Urut) :
 4 9 11 24 31 41 46 61 66 82
===============================================
```

Teknik Pengumpulan

δ. 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

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

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

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