LAPORAN PRAKTIKUM 4
Analysis Algorithm

Rekurensi dan Paradigma Algoritma Divide and Conquer

Disusun oleh :

Mohammad Dhikri
140810180075

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

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:

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

BEDAH ALGORITMA MERGE-SORT

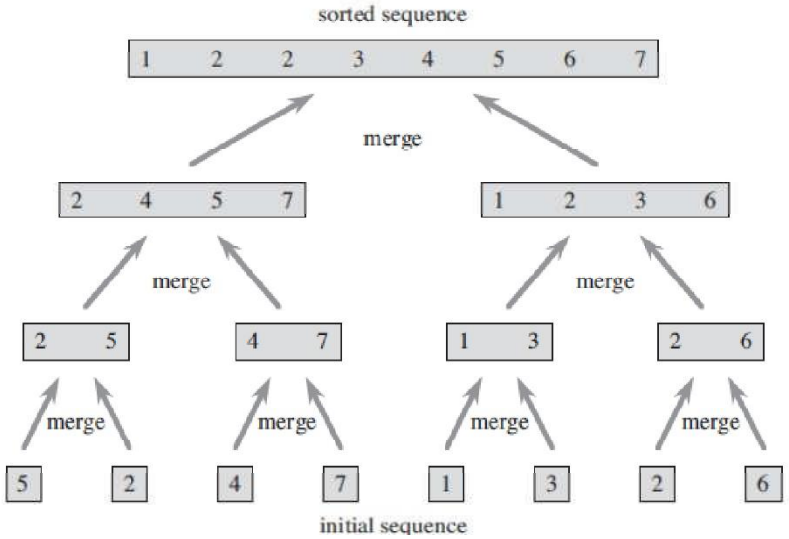
- Merupakan algoritma sorting dengan paradigma divide & conquer
- Running time worst case-nya mempunyai laju pertumbuhan yang lebih rendah dibandingkan insertion sort
- Karena kita berhadapan dengan banyak subproblem, kita notasikan setiap subproblem sebagai sorting sebuah subarray A[p..r]
- Inisialisasi,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

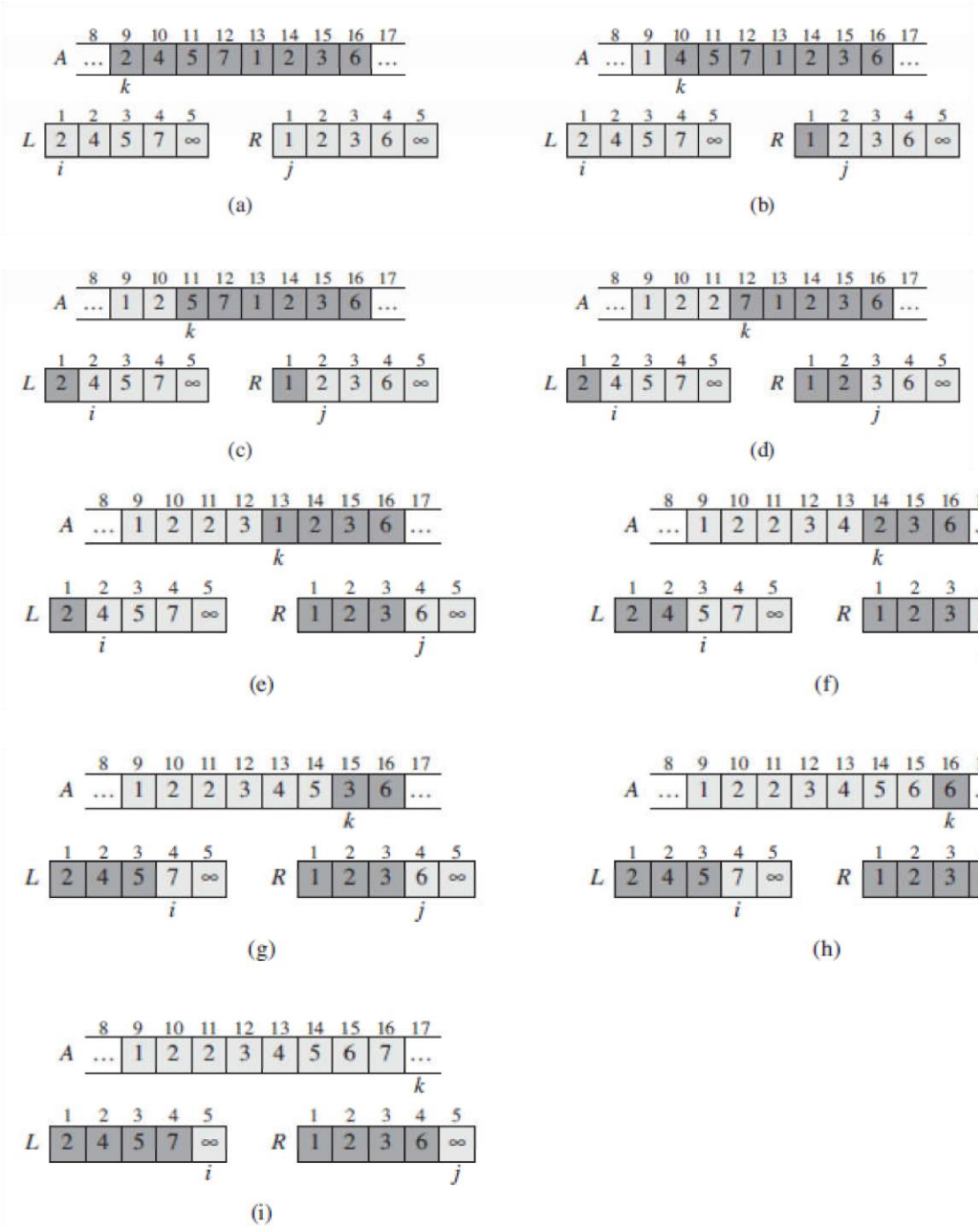
Untuk melihat running time prosedur MERGE berjalan di Θ(n), dimana n-p+1, perhatikan perulangan for pada baris ke 3 dan 4,

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

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 (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?

Jawab :

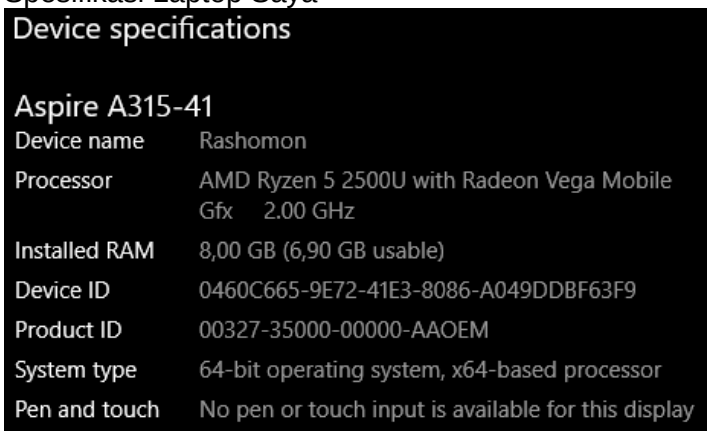
```
#include <iostream>
using namespace std;
void Merge(int *a, int low, int high, int mid) {
        int i, j, k, temp[high-low+1];
        i = low;
        k = 0;
        j = mid + 1;
        while (i <= mid && j <= high){
                if (a[i] < a[j]){
                        temp[k] = a[i];
                        k++;
                        i++;
                }
                else{
                        temp[k] = a[j];
                        k++;
                        j++;
                }
        }
```

```
        while (i <= mid){
                temp[k] = a[i];
                k++;
                i++;
        }
        while (j <= high){
                temp[k] = a[j];
                k++;
                j++;
        }
        for (i = low; i <= high; i++){
                a[i] = temp[i-low];
        }
}
void MergeSort(int *a, int low, int high){
        int mid;
        if (low < high){
                mid=(low+high)/2;
                MergeSort(a, low, mid);
                MergeSort(a, mid+1, high);
                Merge(a, low, high, mid);
        }
}

int main(){
        int n, i;
        cout<<"\nMasukkan jumlah data yang akan di urutkan: ";
        cin>>n;
        int arr[n];
        for(i = 0; i < n; i++){
                cout<<"Masukkan elemen ke-"<<i+1<<": ";
                cin>>arr[i];
        }

        MergeSort(arr, 0, n-1);
        cout<<"\nData Terurut: ";
        for (i = 0; i < n; i++)
      cout<<"->"<<arr[i];
        return 0;
}
```

Spesifikasi Laptop Saya



Kompleksitas waktu yang dibutuhkan untuk inputan sebanyak 20

```
Data Terurut: ->1->1->3->4->5->6->7->9->12->23->24->32->42->54->76->78->80->87->90->97
--------------------------------
Process exited after 30.6 seconds with return value 0
Press any key to continue . . .
```

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

Jawab :

Algoritma selection sort adalah algoritma yang melakukan pengurutan secara berulang-ulang hingga didapatkan hasil pengurutan yang sesuai. Algoritma selection sort akan memindai nilai terkecil dari suatu kumpulan data dan jika ada, data teresebut akandiletakkan pada urutan pertama. Begitu selanjutnya untuk urutan kedua dan seterusnya. Cara kerja dari selection sort ini dalah sebagai berikut.

- Melakukan pengecekan dimulai dari data pertama hingga data ke-n.
- Menentukan data dengan indeks minimum (jika acending) atau maksimum (jika descending) dalam sebuah data tersebut.
- Menukarkan data dengan indeks minimum (jika ascending) atau maksimum (jika descending) dengan bilangan pertama (i = 1) dari data tersebut.
- Mengulangi langkah di atas untuk sisa data bilangan berikutnya (i = i +1) sampai didapatkan urutan yang sesuai.

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

Menentukan T(n):

$$T(n) = (n-1) + (n-2) + \cdots + 1 = \sum_{k=1}^{n-1} \mathrm{n} - \mathrm{k} = \frac{\mathrm{n}(\mathrm{n}-1)}{2}$$

Oleh Karena itu:

$$T(n) = O(n^2)$$

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

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)

T(n) = cn + cn-c +cn-2c + ... + 2c +cn

= c((n-1)(n-2)/2) + cn

$= c((n^2-3n+2)/2) + cn$

$= c(n^2/2)-(3n/2)+1 + cn$

$= \Omega\ (n^2)$

$T(n) = cn^2$

$= \Theta(n^2)$

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

```
#include <iostream>
using namespace std;
int data[100],data2[100];
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);
  }
}

main(){
  cout<<"Masukkan Jumlah Data : ";
  cin>>n;
  for(int i=1;i<=n;i++){
    cout<<"Masukkan data ke "<<i<<" : ";
    cin>>data[i];
    data2[i]=data[i];
  }
  selection_sort();
  cout<<"Data Setelah di Sort : ";
  for(int i=1; i<=n; i++){
    cout<<" "<<data[i];
  }
}
```

```
Masukkan Jumlah Data : 7
Masukkan data ke 1 : 2
Masukkan data ke 2 : 1
Masukkan data ke 3 : 6
Masukkan data ke 4 : 0
Masukkan data ke 5 : 9
Masukkan data ke 6 : 8
Masukkan data ke 7 : 4
Data Setelah di Sort :  0 1 2 4 6 8 9
--------------------------------
Process exited after 9.879 seconds with return value 0
Press any key to continue . . .
```

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

Jawab:

Insertion sort adalah sebuah algoritma pengurutan yang membandingkan dua elemen data pertama, mengurutkannya, kemudian mengecek elemen data berikutnya satu persatu dan membandingkannya dengan elemen data yang telah diurutkan. Karena algoritma ini bekerja dengan membandingkan elemen-elemen data yang akan diurutkan, algoritma ini termasuk pula dalam comparison-based sort.

Ide dasar dari algoritma Insertion Sort ini adalah mencari tempat yang “tepat” untuk setiap elemen array, dengan cara sequential search. Proses ini kemudian menyisipkan sebuah elemen array yang diproses ke tempatnya yang seharusnya. Proses dilakukan sebanyak N-1 tahapan (dalam sorting disebut sebagai “pass“), dengan indeks dimulai dari 0.

Proses pengurutan dengan menggunakan algoritma Insertion Sort dilakukan dengan cara membandingkan data ke-i (dimana i dimulai dari data ke-2 sampai dengan data terakhir) dengan data berikutnya. Jika ditemukan data yang lebih kecil maka data tersebut disisipkan ke depan sesuai dengan posisi yang seharusnya.

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
```

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

Jawab :

- Subproblem = 1
- Masalah setiap subproblem = n-1

- Waktu proses penggabungan = n
- Waktu proses pembagian = n

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

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

T(n) = cn + cn-c +cn-2c + ... + 2c +cn <= 2cn2 + cn2

= c((n-1)(n-2)/2) + cn<= 2cn2 + cn2

= c((n2-3n+2)/2) + cn<= 2cn2 + cn2

= c(n2/2)-c(3n/2)+c+cn <= 2cn2 + cn2

=O(n2)

T(n) = cn <= cn

= Ω (n)

T(n) = (cn + cn2)/n

= Θ(n)

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

```
#include <iostream>
using namespace std;
int data[100],data2[100];
int 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;
  }
}
main(){
  cout<<"Masukkan Jumlah Data : ";
  cin>>n;
  for(int i=1;i<=n;i++){
    cout<<"Masukkan data ke "<<i<<" : ";
    cin>>data[i];
```

```
    data2[i]=data[i];
  }
  insertion_sort();
  cout<<"Data Setelah di Sort : ";
  for(int i=1; i<=n; i++){
    cout<<" "<<data[i];
  }
}
```

```
Masukkan Jumlah Data : 7
Masukkan data ke 1 : 2
Masukkan data ke 2 : 5
Masukkan data ke 3 : 3
Masukkan data ke 4 : 4
Masukkan data ke 5 : 9
Masukkan data ke 6 : 6
Masukkan data ke 7 : 7
Data Setelah di Sort :  2 3 4 5 6 7 9
--------------------------------
Process exited after 7.986 seconds with return value 0
Press any key to continue . . .
```

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

Jawab:

1. Cara pengurutannya : bandingkan dua data kemudian swap.
2. Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.
3. Ascending : Jika elemen sekarang lebih besar dari elemen berikutnya maka kedua elemen tersebut ditukar/swap.
4. Descending : Jika elemen sekarang lebih kecil dari elemen berikutnya, maka kedua elemen tersebut ditukar/swap.
5. Algoritma :

```
swapped = false
for i ← 1 to n-1 do
        if array[i-1] > array[i] then
                swap array[i-1] and array[i]
                swapped = true
        endif
endfor
```

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

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

- Subproblem = 1
- Masalah setiap subproblem = n-1

- Waktu proses pembagian = n
- Waktu proses penggabungan = n

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

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

$T(n) = cn + cn\text{-}c + cn\text{-}2c + ... + 2c + c <= 2cn^2 + cn^2$

$= c((n-1)(n-2)/2) + c <= 2cn^2 + cn^2$

$= c((n^2\text{-}3n+2)/2) + c <= 2cn^2 + cn^2$

$= c(n^2/2)\text{-}c(3n/2)+2c <= 2cn^2 + cn^2$

$= O(n^2)$

$T(n) = cn + cn\text{-}c + cn\text{-}2c + ... + 2c + c <= 2cn^2 + cn^2$

$= c((n-1)(n-2)/2) + c <= 2cn^2 + cn^2$

$= c((n^2\text{-}3n+2)/2) + c <= 2cn^2 + cn^2$

$= c(n^2/2)\text{-}c(3n/2)+2c <= 2cn^2 + cn^2$

$= \Omega\ (n^2)$

$T(n) = cn^2 + cn^2$

$= \Theta(n^2)$

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

```
#include<iostream>
using namespace std;
main(){
	int n, i, arr[50], j, temp;
	cout<<"Masukkan total elemen yang akan diurutkan: ";
	cin>>n;
	for(i=0; i<n; i++){
		cout<<"Masukan angka ke-"<<i+1<<": ";
		cin>>arr[i];
	}
	for(i=0; i<(n-1); i++)    {
		for(j=0; j<(n-i-1); j++){
			if(arr[j]>arr[j+1]){
				temp=arr[j];
				arr[j]=arr[j+1];
```

```
                                arr[j+1]=temp;
                        }
                }
        }
        cout<<"Data terurut :\n";
        for(i=0; i<n; i++){
                cout<<arr[i]<<" ";
        }
}
```

```
Masukkan total elemen yang akan diurutkan: 8
Masukan angka ke-1: 3
Masukan angka ke-2: 4
Masukan angka ke-3: 5
Masukan angka ke-4: 1
Masukan angka ke-5: 3
Masukan angka ke-6: 7
Masukan angka ke-7: 5
Masukan angka ke-8: 9
Data terurut :
1 3 3 4 5 5 7 9
--------------------------------
Process exited after 12.21 seconds with return value 0
Press any key to continue . . .
```