Nama : Rizky Anugerah
NPM : 140810180049
Tugas 4

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

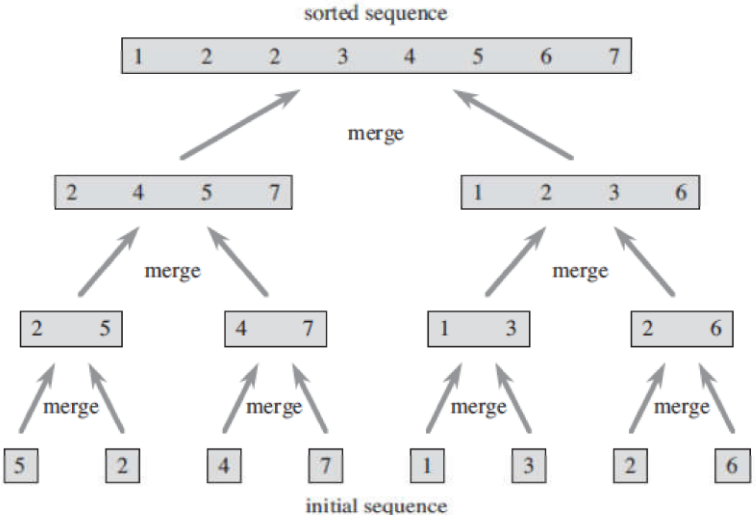
Untuk mengurutkan A[p..r]:

- **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

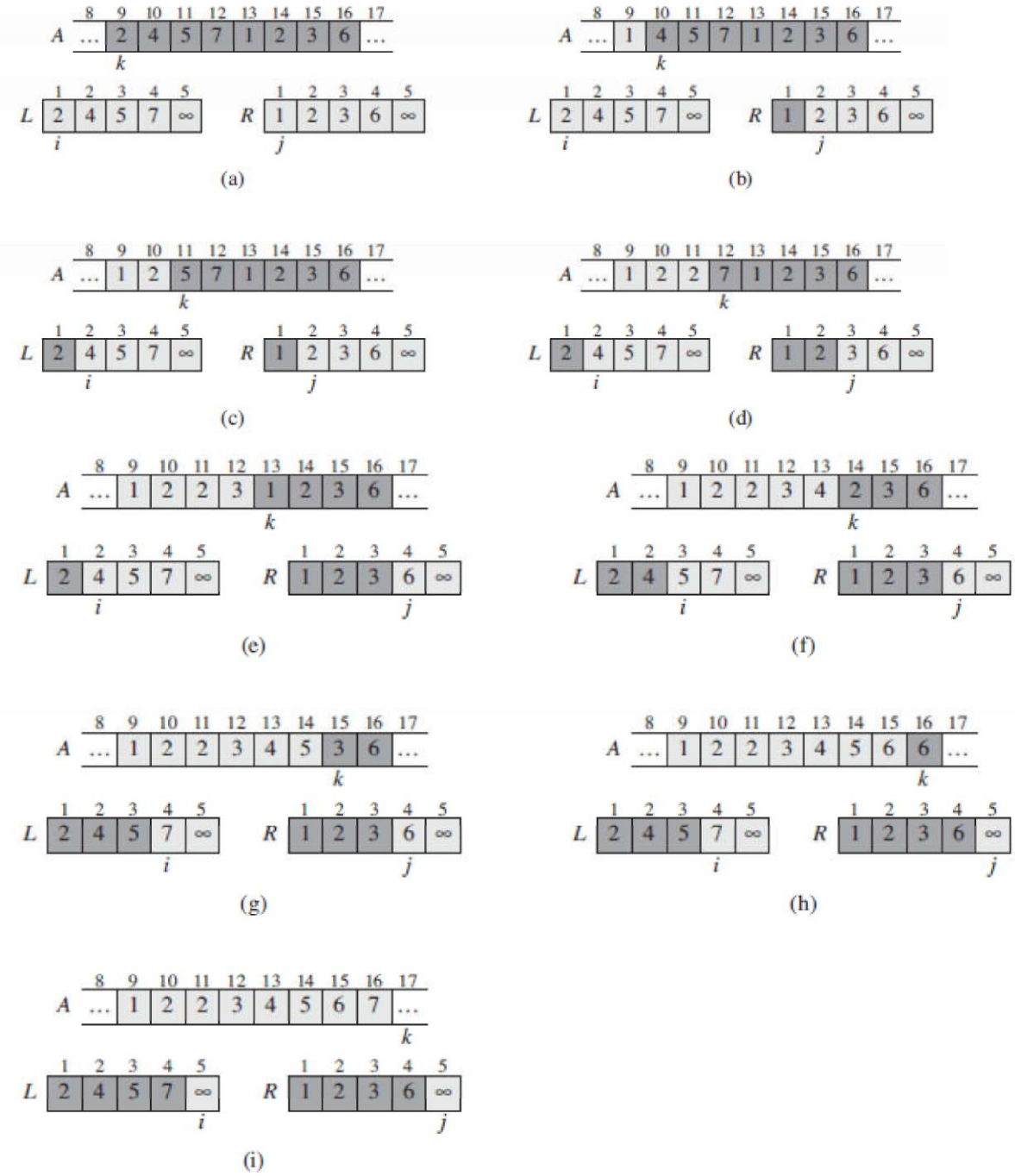
perulangan for pada baris ke 3 dan 4,

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

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 (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++

   Jawab:

```
/*
Nama    : Rizky Anugerah
NPM     : 140810180049
Kelas   : A
Program : Merge Sort
*/

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

int data[100];

void mergeSort(int first, int mid, int last)
{
    cout << endl;

    int temp[100], tempFirst = first, tempMid = mid, i = 0;
    while(tempFirst < mid && tempMid < last)
    {
        if(data[tempFirst] < data[tempMid])
            temp[i] = data[tempFirst], tempFirst++;
        else
            temp[i] = data[tempMid], tempMid++;
        i++;
    }
    while(tempFirst < mid)                                  // jika masih
bersisa
        temp[i] = data[tempFirst], tempFirst++, i++;
    while(tempMid < last)
        temp[i] = data[tempMid], tempMid++, i++;
    for(int j=0, k=first; j<i, k<last; j++, k++)            //mengembalika
n ke array semula
    {
        cout << data[k] << ' ' << temp[j] << endl;
        data[k] = temp[j];                                  //sudah teruru
t
    }
}
```

```
void merge(int first, int last)                                    // membagi dat
a secara rekursif
{
    if(last-first != 1)
    {
        int mid = (first+last)/2;
        merge(first, mid);
        merge(mid, last);
        mergeSort(first, mid, last);
    }
}

int main()
{
    int n;

    cout << "Masukan jumlah data : "; cin >> n;

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

    // start perhitungan running time
    auto start = chrono::steady_clock::now();
    merge(0,n);
    auto end = chrono::steady_clock::now();

    cout << "\nData terurut : ";
    for(int i=0; i<n; i++)
    {
        cout << data[i] << ' ';
    }

    cout << endl;

    // running time
    cout << "Elapsed time in nanoseconds : "  << chrono::duration_cast<chrono:
:nanoseconds>(end - start).count()  << " ns" << endl;
    cout << "Elapsed time in microseconds : " << chrono::duration_cast<chrono:
:microseconds>(end - start).count() << " µs" << endl;
    cout << "Elapsed time in milliseconds : " << chrono::duration_cast<chrono:
:milliseconds>(end - start).count() << " ms" << endl;
    cout << "Elapsed time in seconds : " << chrono::duration_cast<chrono::seco
nds>(end - start).count() << " sec";

    return 0;
```

```
}
```

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:

```
Data terurut : 11 12 23 29 33 34 36 41 42 43 49 56 65 68 69 71 77 82 89 90 Elapsed time in nanoseconds : 226114 ns
Elapsed time in microseconds : 226 µs
Elapsed time in milliseconds : 0 ms
Elapsed time in seconds : 0 sec
```

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

  Selection sort adalah algoritma pengurutan yang sederhana namun sangat efisien dalam penggunaanya, juga memiliki penulisan code yang simpel. Algoritma ini menggabungkan metode searching dan sorting. Dalam selection sort ada dua cara pengurutan yaitu secara Ascending (mengurutkan dari kecil ke besar), index nilai paling kecil disimpan lalu ditukar dengan index pertama. Sedangkan secara Descending (mengurutkan dari besar ke kecil/kebalikan dari ascending), index dengan nilai paling besar disimpan lalu ditukar. Contoh ilustrasi cara kerja selection sort adalah sebagai berikut:

  1. Pertama jika menggunakan cara Ascending maka, program akan mencari index dengan nilai paling kecil dari semua data yang acak lalu ditukar dengan index paling depan.
  2. Setelah itu program mencari lagi index dengan nilai kecil kedua, disimpan pada variabel temporary lalu ditukar dengan index kedua.
  3. Begitu seterusnya sampai semua data terurut.

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

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

Jawab:

```
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
```

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

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

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

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

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

$= \mathbf{\Omega(n^2)}$

$T(n) = cn^2$

$= \mathbf{\Theta(n^2)}$

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

  Jawab:

  Jadi kompleksitas waktu masing masing Big-O, Big-Ω, dan Big-Θ nya yaitu $\mathbf{n^2}$

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

  Jawab:

```cpp
#include <iostream>
using namespace std;

int n;
int data[100], data2[100];

void swap(int a, int b)
{
    int t;

    t = data[b];
    data[b] = data[a];
    data[a] = t;
}

void output()
{
    for (int i = 1; i <= n; i++)
    {
        cout << data[i] << ' ';
    }
    cout << endl;
}

void selectionSort()
{
    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)
        {
            swap(pos, i);
        }
        output();
    }
}

int 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];
    }

    output();
    selectionSort();

    cout << "\nData terurut : " << endl;
    output();

    return 0;
}
```

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 merupakan algoritma yang efisien untuk mengurutkan angka yang mempunyai jumlah elemen sedikit. Metode penyisipan (Insertion sort) bertujuan untuk menjadikan bagian sisi kiri array terurutkan sampai dengan seluruh array berhasil diurutkan. Metode ini mengurutkan bilangan-bilangan yang telah dibaca; dan berikutnya secara berulang akan menyisipkan bilangan-bilangan dalam array yang belum terbaca ke sisi kiri array yang telah terurut.

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

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

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

  Jawab:

```
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
```

Best case

Best case atau kondisi terbaik yaitu dimana semua data telah terurut. Untuk setiap j = 2,3,.... n, kita dapat menemukan A[i] kurang dari atau sama dengan key dimana i memiliki nilai inisial (j – 1). Dengan kata lain, ketika i = j – 1, akan selalu didapatkan key A[i] pada waktu perluangan *while* berjalan.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n}(1) + c_6 \sum_{j=2}^{n}(1-1) + c_7 \sum_{j=2}^{n}(1-1) + c_8(n-1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n + (c_1 + c_2 + c_4 + c_5 + c_8)$$

Running time pada *best case* ini dapat ditunjukkan dengan an + b dimana konstanta a dan b bergantung pada *statement* c_i. Sehingga T(n) adalah fungsi liniear dari n.

$$T(n) = an + b = O(n) \qquad \ldots\ldots(4)$$

Worst Case

Worst case atau kondisi terburuk terjadi jika *array* diurutkan dalam urutan terbalik yaitu, dalam urutan menurun (besar ke kecil). Dalam urutan terbalik, selalu ditemukan A[i] lebih besar dari key pada perulangan *loop*. Sehingga, harus membandingkan setiap elemen A[j] dengan seluruh urutan elemen sub-*array* A[1.....j-1] dan juga tj = j untuk j = 2,3,.... n. Secara ekuivalen, saat perulangan while keluar disebabkan i telah mencapai indeks 0. Oleh karena itu, tj = j untuk j = 2,3,...,n dan *running time worst case* dapat dihitung menggunakan persamaan sebagai berikut :

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n}(1) + c_6 \sum_{j=2}^{n}(1-1) + c_7 \sum_{j=2}^{n}(1-1) + c_8(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n}\left[\frac{n(n+1)}{2} + 1\right] + c_6 \sum_{j=2}^{n}\left[\frac{n(n-1)}{2}\right] + c_7 \sum_{j=2}^{n}\left[\frac{n(n-1)}{2}\right] + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

Running time pada *worst case* ini dapat dinyatakan sebagai $an^2 + b_n + n$, untuk konstanta a, b dan c bergantung pada *statement* c_i. Oleh karena itu, T(n) adalah fungsi kuadrat dari n.

$$T(n) = an^2 + bn + c = O(n^2) \quad \ldots\ldots (8)$$

Average Case

Average case terjadi apabila data yang diurutkan acak. Key dalam A[i] adalah kurang dari setengah elemen dalam A[1.....j-1] dan lebih besar dari setengah lainnya. Hal ini berarti pada kondisi *average*, perulangan *while* harus melalui setengah jalan melalui *subarray* A diurutkan [1...j-1] untuk memutuskan dimana memposisikan key. Ini berarti tj = j/2.

Meskipun *running time average case* adalah setengah dari waktu *running time worst case*, *average case* masih memiliki fungsi kuadrat dari n.

$$T(n) = an^2 + bn + c = O(n^2). \quad \ldots\ldots(9)$$

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

  Jawab:

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

int n;
int data[100], data2[100];

void swap(int a, int b)
{
    int t;

    t = data[b];
    data[b] = data[a];
    data[a] = t;
}

void output()
{
    for (int i = 1; i <= n; i++)
    {
        cout << data[i] << ' ';
    }
    cout << endl;
}

void insertionSort()
{
    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 << "Masukkan jumlah data : "; cin >> n;

    for (int i=1; i<=n; i++)
    {
        cout << "Masukkan data ke- " << i << " : "; cin >> data[i];
        data2[i] = data[i];
    }

    output();
    insertionSort();

    cout << "\nData terurut : ";
    output();

    return 0;
}
```

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:

  Algoritma Bubble Sort ini merupakan proses pengurutan yang secara berangsur-angsur berpindah ke posisi yang tepat karena itulah dinamakan Bubble yang artinya gelembung. Algoritma ini akan mengurutkan data dari yang terbesar ke yang terkecil (ascending) atau sebaliknya (descending).

  Secara sederhana, bisa didefenisikan algoritma Bubble Sort adalah pengurutan dengan cara pertukaran data dengan data disebelahnya secara terus menerus sampai dalam satu iterasi tertentu tidak ada lagi perubahan.

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

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

  Jawab:

Bubble sort menggunakan dua buah *loop* yakni *inner loop* dan *outer loop*. *Outer loop* akan melakukan iterasi sebanyak n-1 kali, sedangkan *inner loop* akan melakukan penelusuran dan pertukaran nilai yang urutannya tidak benar.

1. Best case analysis

   Kondisi ini tercapai apabila *array* sudah dalam keadaan terurut. Iterasi yang bekerja hanyalah *outer loop* sebanyak n-1 kali. Oleh karena itu nilai kompleksitas algoritma yang didapat adalah O(n).

2. Worst case analysis

   Kondisi ini tercapai apabila *array* dalam keadaan terurut namun terbalik. Oleh karena itu dibutuhkan pengulangan sebanyak n-1 dan pertukaran dalam *inner loop*. Nilai T(n) yang didapat adalah

$$T(n) = (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

$$T(n) = \frac{n(n-1)}{2} \qquad (17)$$

   Dari persamaan (17) didapat nilai kompleksitas algoritmanya sebesar $O(n^2)$.

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

  Jawab:

$$T(n) = \begin{cases} a & ,n=1 \\ T(n-1)+cn & ,n>1 \end{cases}$$

Karena a=1, dan b=1. maka tidak memenuhi syarat untuk menggunakan metode master (syaratnya a>= 1 dan b>1), maka rekurensi algoritma bubble sort tidak bisa diselesaikan dengan metode master.

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

  Jawab:

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

int n;
int data[100], data2[100];

int swap(int a, int b)
{
    int t;

    t = data[b];
    data[b] = data[a];
    data[a] = t;
}

int output()
{
    for (int i = 0; i < n; i++)
    {
        cout << data[i] << ' ';
    }
    cout << endl;
}

int bubbleSort()
{
    for (int i=1; i<n; i++)
    {
        for (int j = n-1; j>=i; j--)
        {
            if (data[j] < data[j-1])
            {
                swap(j, j-1);
            }
        }
        output();
    }
```

```
    cout << endl;
}

int main()
{
    cout << "Masukan jumlah data : "; cin >> n;

    for (int i=0; i<n; i++)
    {
        cout << "Masukan data ke- " << i + 1 << " : "; cin >> data[i];
        data2[i] = data[i];
    }

    output();
    bubbleSort();

    cout << "\nData terurut : " << endl;
    output();

    return 0;
}
```