Nama : Fauzan Akmal Hariz
NPM : 140810180005
Tugas-4

PRAKTIKUM ANALISIS ALGORITMA

REKURENSI DAN PARADIGMA ALGORITMA DIVIDE & CONQUER

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.

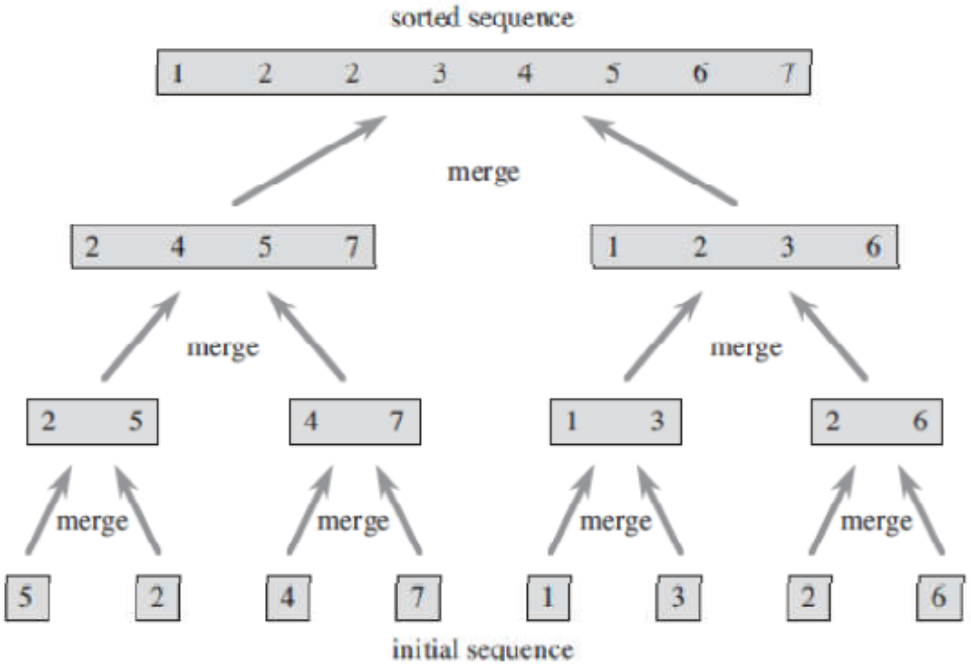
Untuk mengurutkan A[p..r]:

- **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 $\Theta(n)$, 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

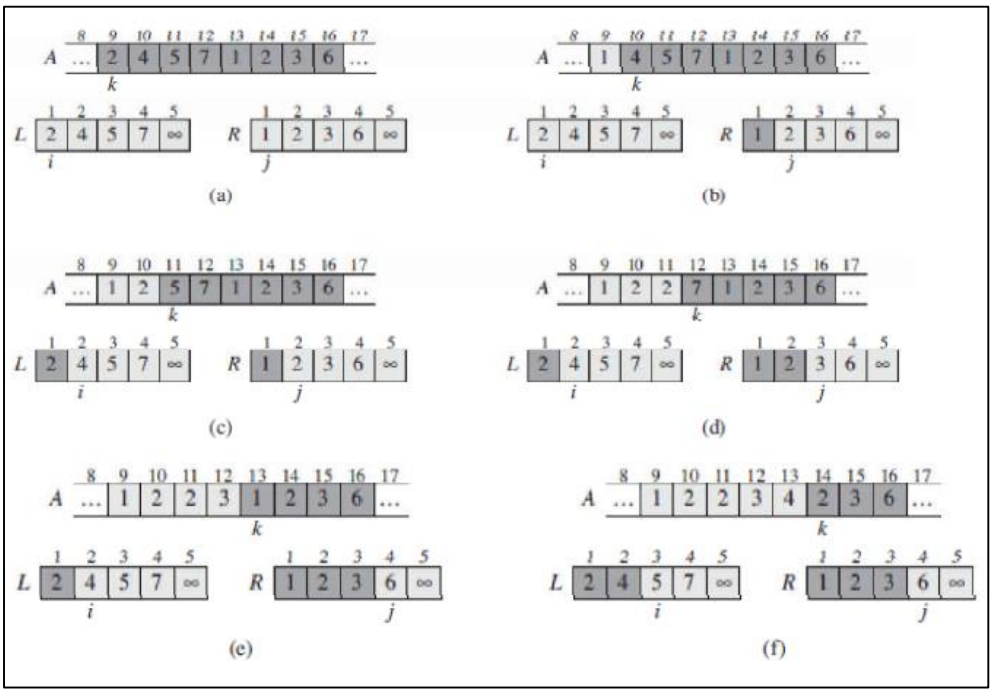
Untuk melihat running time prosedur MERGE berjalan di Θ(n), dimana n-p+1, perhatikan perulangan for pada baris ke 3 dan 4,

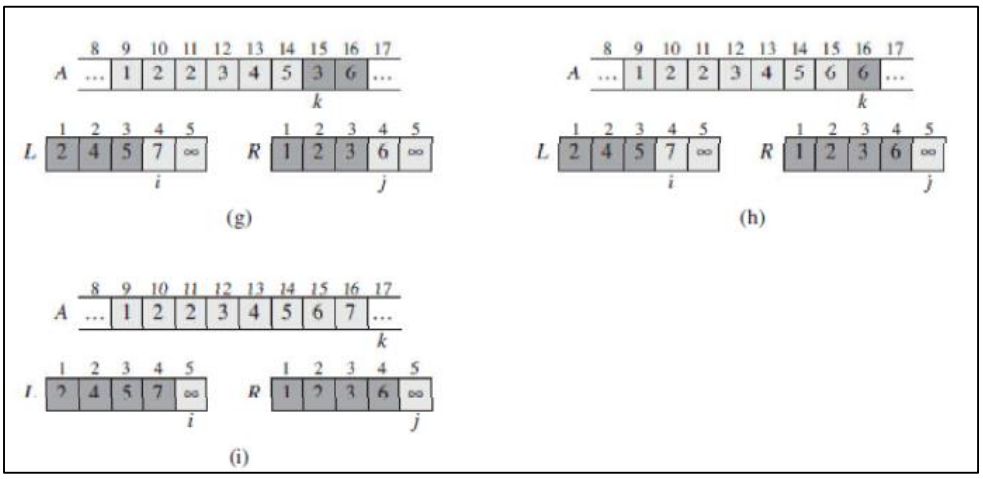
$$\Theta(n1 + n2) = \Theta(n)$$

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 $\Theta(1)$
- Sebaliknya, 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 $(n/b) \rightarrow$ 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 PPT praktikum analisis algortima (pertemuan 4).

Studi Kasus

Studi Kasus 1: MERGE SORT

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

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

  Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   23 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Merge Sort
Deskripsi       :   Studi Kasus 1 - Merge Sort
*/

#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

void merge (int *a, int low, int high, int mid){
    int i;
    int j;
    int k;
    int 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;
    int i;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    cout << "\n==================\n";
    cout << "Program Merge Sort\n";
    cout << "==================\n";

    cout << "\nSilahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan:
"; cin >> n; cout << "\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 << "\nArray yang Telah Diurutkan: ";
    for (i=0; i<n; i++){
        cout << " " << arr[i];
    }
```

```
    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(t2 - t1).count();
    cout << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

Screenshot Program:

```
==================
Program Merge Sort
==================

Silahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan: 20

Masukkan Elemen ke-1: 5
Masukkan Elemen ke-2: 17
Masukkan Elemen ke-3: 9
Masukkan Elemen ke-4: 11
Masukkan Elemen ke-5: 3
Masukkan Elemen ke-6: 15
Masukkan Elemen ke-7: 7
Masukkan Elemen ke-8: 19
Masukkan Elemen ke-9: 1
Masukkan Elemen ke-10: 13
Masukkan Elemen ke-11: 16
Masukkan Elemen ke-12: 8
Masukkan Elemen ke-13: 20
Masukkan Elemen ke-14: 2
Masukkan Elemen ke-15: 4
Masukkan Elemen ke-16: 6
Masukkan Elemen ke-17: 18
Masukkan Elemen ke-18: 10
Masukkan Elemen ke-19: 12
Masukkan Elemen ke-20: 14

Array yang Telah Diurutkan:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 16 17 18 19 20

Durasi Waktu: 32475534 microseconds
```

- 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?
  Durasi waktu yang dibutuhkan untuk 20 input:
  - Berdasarkan program hasilnya:
    32475534 microsecond = 32,475534 second
  - Berdasarkan kompleksitas algoritma merge sort adalah O(n lg n):
    $T(20 \log_{10} 20) = 26$

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.

```
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
```

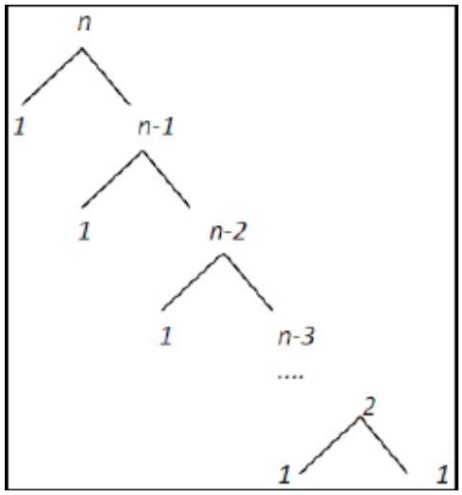
Subproblem = 1
Masalah setiap subproblem = n-1
Waktu proses pembagian = n
Waktu proses penggabungan = n

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

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

$$T(n) = \begin{cases} \Theta(1) \\ T(n-1) + \Theta(n) \end{cases}$$

- 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
= Ω(n^2)

T(n) = cn^2
= Θ(n^2)

- Lakukan implementasi koding program untuk algoritma selection sort dengan menggunakan bahasa C++.
Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   23 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Selection Sort
Deskripsi       :   Studi Kasus 2 - Selection Sort
*/

#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

void selectionSort (int arr[], int n){
    int i;
    int j;
    for (i=0; i<n; i++){
        for (j=i+1; j<n; j++){
            if (arr[i] > arr[j]){
                arr[i]=arr[i]+arr[j];
                arr[j]=arr[i]-arr[j];
                arr[i]=arr[i]-arr[j];
            }
        }
    }
```

```
}

int main(){
    int n;
    int i;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    cout << "\n======================\n";
    cout << "Program Selection Sort\n";
    cout << "======================\n";

    cout << "\nSilahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan:
"; cin >> n; cout << "\n";

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

    selectionSort(arr, n);

    cout<<"\nArray yang Telah Diurutkan: ";
    for (i=0; i<n; i++){
        cout<<" "<<arr[i];
    }

    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>( t2 - t1 ).count();
    cout << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

Screenshot Program:

```
====================
Program Selection Sort
====================

Silahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan: 20

Masukkan Elemen ke-1: 5
Masukkan Elemen ke-2: 17
Masukkan Elemen ke-3: 9
Masukkan Elemen ke-4: 11
Masukkan Elemen ke-5: 3
Masukkan Elemen ke-6: 15
Masukkan Elemen ke-7: 7
Masukkan Elemen ke-8: 19
Masukkan Elemen ke-9: 1
Masukkan Elemen ke-10: 13
Masukkan Elemen ke-11: 16
Masukkan Elemen ke-12: 8
Masukkan Elemen ke-13: 20
Masukkan Elemen ke-14: 2
Masukkan Elemen ke-15: 4
Masukkan Elemen ke-16: 6
Masukkan Elemen ke-17: 18
Masukkan Elemen ke-18: 10
Masukkan Elemen ke-19: 12
Masukkan Elemen ke-20: 14

Array yang Telah Diurutkan:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 16 17 18 19 20

Durasi Waktu: 31210146 microseconds
```

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.

```
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

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

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

$$T(n) = \begin{cases} \Theta(1) \\ T(n-1) + \Theta(n) \end{cases}$$

- 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 <= $2cn^2 + cn^2$
= 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)

T(n) = cn <= cn
= Ω (n)

T(n) = (cn + cn^2)/n
= Θ(n)

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

Source Code:

```cpp
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   23 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Insertion Sort
Deskripsi       :   Studi Kasus 3 - Insertion Sort
*/

#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

struct list {
    int data;
    list *next;
};

list* InsertinList(list *head, int n){
    list *newnode = new list;
    list *temp = new list;

    newnode->data = n;
    newnode->next = NULL;

    if(head == NULL){
        head = newnode;
        return head;
    }
    else {
        temp = head;

        if(newnode->data < head->data){
            newnode->next = head;
            head = newnode;
            return head;
        }

        while(temp->next != NULL){
            if(newnode->data < (temp->next)->data) {
                break;
            }
            temp=temp->next;
        }
```

```
        newnode->next = temp->next;
        temp->next = newnode;
        return head;
    }
}

int main(){
    int n;
    int i;
    int num;
    list *head = new list;
    head = NULL;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    cout << "\n======================\n";
    cout << "Program Insertion Sort\n";
    cout << "======================\n";

    cout << "\nSilahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan:
"; cin >> n; cout << "\n";

    for(i=0; i<n; i++){
        cout<<"Masukkan Elemen ke-" << i+1 << ": "; cin>>num;
        head = InsertinList(head, num);
    }

    cout<<"\nArray yang Telah Diurutkan: ";
    while(head != NULL){
        cout << " " << head->data;
        head = head->next;
    }

    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>( t2 - t1 ).count();
    cout << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

Screenshot Program:

```
======================
Program Insertion Sort
======================

Silahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan: 20

Masukkan Elemen ke-1: 5
Masukkan Elemen ke-2: 17
Masukkan Elemen ke-3: 9
Masukkan Elemen ke-4: 11
Masukkan Elemen ke-5: 3
Masukkan Elemen ke-6: 15
Masukkan Elemen ke-7: 7
Masukkan Elemen ke-8: 19
Masukkan Elemen ke-9: 1
Masukkan Elemen ke-10: 13
Masukkan Elemen ke-11: 16
Masukkan Elemen ke-12: 8
Masukkan Elemen ke-13: 20
Masukkan Elemen ke-14: 2
Masukkan Elemen ke-15: 4
Masukkan Elemen ke-16: 6
Masukkan Elemen ke-17: 18
Masukkan Elemen ke-18: 10
Masukkan Elemen ke-19: 12
Masukkan Elemen ke-20: 14

Array yang Telah Diurutkan:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 16 17 18 19 20

Durasi Waktu: 35206989 microseconds
```

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.

```
Algoritma:
for i ← 1 to n - 1 do
    for j ← n downto i + 1 do
        if data[j] < data[j-1] then
            { lakukan pertukaran }
            temp ← data[j]
            data[j] ← s[j-1]
            data[j-1] ← temp
        endif
    endfor
endfor
```

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

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

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

$$T(n) = \begin{cases} \Theta(1) \\ T(n-1) + \Theta(n) \end{cases}$$

- 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-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)

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)

T(n) = $cn^2 + cn^2$
= Θ(n^2)

- Lakukan implementasi koding program untuk algoritma bubble sort dengan menggunakan bahasa C++.
  Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   23 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Bubble Sort
Deskripsi       :   Studi Kasus 4 - Bubble Sort
*/

#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

void bubbleSort (int arr[], int n){
    int i;
    int j;
    for (i=0; i<n; i++){
        for (j=0; j<n-i-1; j++){
            if (arr[j] > arr[j+1]){
                arr[j]=arr[j]+arr[j+1];
                arr[j+1]=arr[j]-arr[j + 1];
                arr[j]=arr[j]-arr[j + 1];
            }
        }
    }
}

int main(){
    int n;
    int i;
    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    cout << "\n====================\n";
    cout << "Program Bubble Sort\n";
    cout << "====================\n";

    cout << "\nSilahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan:
"; cin >> n; cout << "\n";

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

```
    }

    bubbleSort(arr, n);

    cout << "\nArray yang Telah Diurutkan: ";
    for (i=0; i<n; i++){
        cout << " " << arr[i];
    }

    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>( t2 - t1 ).count();
    cout << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

Screenshot Program:

```
==================
Program Bubble Sort
==================

Silahkan Masukkan Jumlah Elemen Data yang Ingin Diurutkan: 20

Masukkan Elemen ke-1: 5
Masukkan Elemen ke-2: 17
Masukkan Elemen ke-3: 9
Masukkan Elemen ke-4: 11
Masukkan Elemen ke-5: 3
Masukkan Elemen ke-6: 15
Masukkan Elemen ke-7: 7
Masukkan Elemen ke-8: 19
Masukkan Elemen ke-9: 1
Masukkan Elemen ke-10: 13
Masukkan Elemen ke-11: 16
Masukkan Elemen ke-12: 8
Masukkan Elemen ke-13: 20
Masukkan Elemen ke-14: 2
Masukkan Elemen ke-15: 4
Masukkan Elemen ke-16: 6
Masukkan Elemen ke-17: 18
Masukkan Elemen ke-18: 10
Masukkan Elemen ke-19: 12
Masukkan Elemen ke-20: 14

Array yang Telah Diurutkan:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
 16 17 18 19 20

Durasi Waktu: 28921395 microseconds
```

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.

REFERENSI

PPT Praktikum Analisis Algortima (Pertemuan 4)

Modul Praktikum 4 Analisis Algoritma