LAPORAN 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

DISUSUN OLEH : AHMAD FAAIZ A (140810180023)

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
MARET 2019

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++

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace std::chrono;

void merge(int arr[], int l, int mid, int r)
{
    int sizeL = mid - l + 1; //left-array's length, +1 since index starts from 0
    int sizeR = r - mid;     //right-array's length

    int arr_left[sizeL], arr_right[sizeR]; //temp data

    for (int i = 0; i < sizeL; i++) //copy data to temp arr_left
    {
        arr_left[i] = arr[l + i];
    }

    for (int i = 0; i < sizeR; i++) //copy data to temp arr_right
    {
        arr_right[i] = arr[mid + 1 + i]; //mid+1 since arr_right starts from the next of mi
d element
    }

    int i = 0; //initial index of left subarray
    int j = 0; //initil index of right subaray
    int k = l; //initial index of merged subarray

    while (i < sizeL && j < sizeR)
    {
        if (arr_left[i] < arr_right[j]) //if left element is smaller than right element, co
py element of left array
        {
            arr[k] = arr_left[i];
            i++;
        }
        else
        {
            arr[k] = arr_right[j];
            j++;
        }
        k++;
    }

    //copy remainnig elements
    while (i < sizeL)
    {
        arr[k] = arr_left[i];
        i++;
        k++;
    }

    while (j < sizeR)
    {
        arr[k] = arr_right[j];
        j++;
        k++;
    }
```

```
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r) //to stop the reccursion when there's only 1 element in the array
    {
        int mid = (l + r) / 2;

        mergeSort(arr, l, mid);     //merge left subarray
        mergeSort(arr, mid + 1, r); //merge right subarray

        merge(arr, l, mid, r);
    }
}

void print(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}

int main()
{
    int arr[] = {5, 6, 9, 8, 12, 15, 12, 0, -1, 5};

    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "INPUT:" << endl;
    print(arr, arr_size);

    auto start = high_resolution_clock::now();
    mergeSort(arr, 0, arr_size - 1);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

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

    cout << "OUTPUT:" << endl;
    print(arr, arr_size);
}
```

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

   Jawab:

```
INPUT:
5 6 9 8 12 15 12 0 -1 5
Time taken by function: 998 microseconds
OUTPUT:
-1 0 5 5 6 8 9 12 12 15
```

   Kecepatan berdasarkan perhitungan:
   Karena kompleksitas waktunya adalah O(n lg n), maka bila diinputkan n = 20 didapat:

$$T(20) = 20 \log_2 20 = 20 \times 4.32 = 86.43$$

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:

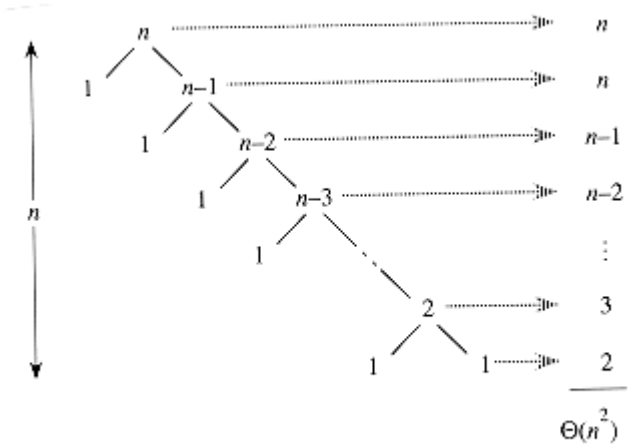
$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Jawab:

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

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

Jawab:



Cost untuk keseluruhan tree adalah: $\theta(n^2)$

Jadi, Big-O, Big-Ω, dan Big-Θ masing-masing adalah n^2.

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

Jawab:

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

void selectionSort(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
```

```
        int min_index = i; //index of minimum element in subarray
        for (int j = i + 1; j < size; j++)
        {
            if (arr[min_index] > arr[j])
            {
                min_index = j;
            }
            swap(arr[min_index], arr[i]);
        }
    }
}

void print(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}

int main()
{
    int arr[] = {5, 6, 8, 7, 4, 5, 12, -1, 0, 65};
    int size_arr = sizeof(arr) / sizeof(arr[0]);

    cout << "INPUT" << endl;
    print(arr, size_arr);

    selectionSort(arr, size_arr);

    cout << "OUTPUT" << endl;
    print(arr, size_arr);
}
```

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:

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

Jawab:

Kita membutuhkan paling banyak $n-1$ perbandingan dan paling banyak $n-1$ pertukaran. Untuk menyisipkan elemen kedua ke elemen terakhir, kita membutuhkan paling banyak n-2n – 2 perbandingan dan paling banyak $n-2$ pertukaran, dan seterusnya. Karenanya, jumlah operasi yang diperlukan untuk melakukan penyisipan adalah: $2 \times (1 + 2 + \cdots + n - 2 + n - 1)$. Untuk menghitung hubungan perulangan untuk algoritma ini, gunakan penjumlahan berikut:

$$\sum_{q=1}^{p} q = \frac{p(p+1)}{2}.$$

Maka diperoleh:

$$\frac{2(n-1)(n-1+1)}{2} = n(n-1).$$

Sehingga :

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

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

Jawab:

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + \{c\,(n-1) + T(n-2)\} \\ &= cn + c(n-1) + \{c\,(n-2) + T(n-3)\} \\ &= cn + c\,(n-1) + c\,(n-2) + \{c(n-3) + T(n-4)\} \\ &= \ldots \\ &= cn + c\,(n-1) + c(n-2) + c(n-3) + \ldots + c2 + T(1) \\ &= c\{n + (n-1) + (n-2) + (n-3) + \ldots + 2\} + a \\ &= c\{\frac{(n-1)(n+2)}{2}\} + a \\ &= \frac{cn^2}{2} + \frac{cn}{2} + (a-c) \\ &= O(n^2) \end{aligned}$$

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

```cpp
#include <iostream>

using namespace std;

void insertionSort(int arr[], int size)

{
    for (int i = 1; i < size; i++)
    {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void print(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}

int main()
{
    int arr[] = {1, 648, 5, 8, 4, 9, 4, 6, -5, 1, 0};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "INPUT" << endl;
    print(arr, arr_size);

    insertionSort(arr, arr_size);

    cout << "OUTPUT" << endl;
    print(arr, arr_size);
}
```

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:

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

Jawab:

Ketika $i = 1$, tidak ada perbandingan yang dibuat oleh program. Ketika $i = 2$, satu perbandingan dibuat oleh program. Ketika $i = 3$, dua perbandingan dibuat, dan seterusnya. Dengan demikian, kita dapat menyimpulkan bahwa ketika $i = m, m - 1$ dibuat perbandingan. Oleh karena itu, dalam array dengan panjang n, ia memiliki $1 + 2 + 3 + 4 + \cdots + (n-2) + (n-1)$ perbandingan. Untuk menghitung hubungan perulangan untuk algoritma ini, gunakan penjumlahan berikut:

$$\sum_{q=1}^{p} q = \frac{p(p+1)}{2}.$$

Maka diperoleh:

$$\frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}.$$

Sehingga:

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

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

  Jawab:

  Tidak bisa dilakukan, karena tidak memenuhi syarat metode master dimana $a \geq 1$, dan $b \geq 2$.

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

  Jawab:

```cpp
#include <iostream>
using namespace std;
void bubbleSort(int arr[], int size)
{
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = 0; j < size - (i + 1); j++) //+1 : index of array (i) starts from 0
        {
            if (arr[j] > arr[j + 1])
```

```
            {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

void print(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << ' ';
    }
    cout << endl;
}

int main()
{
    int arr[] = {5, 4, 8, 9, 7, 2, 0, -5, 4, 1, 5};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "INPUT" << endl;
    print(arr, arr_size);

    bubbleSort(arr, arr_size);

    cout << "OUTPUT" << endl;
    print(arr, arr_size);
}
```