TUGAS 4 PRAKTIKUM
REKURENSI DAN PARADIGMA ALGORITMA DIVIDE & CONQUER

MATA KULIAH ANALISIS ALGORITMA
D10G.4205 & D10K.0400601

Muhammad Ariq Farhansyah Mutyara
140810170053

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
JATINANGOR
2018

Nomor 1

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:

- Program

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

using namespace std;
using namespace std::chrono;

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;
      high_resolution_clock::time_point  t1  =  high_resolution_
clock::now();
      cout<<"Masukkan jumlah elemen data yang ingin diurutkan: ";
      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<<"\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<<endl<<duration<<" microseconds" <<endl;
}
```

- Output (input n = 20):

```
Masukkan jumlah elemen data yang ingin diurutkan: 20
Masukkan elemen ke-1: 1
Masukkan elemen ke-2: 3
Masukkan elemen ke-3: 45
Masukkan elemen ke-4: 67
Masukkan elemen ke-5: 23
Masukkan elemen ke-6: 90
Masukkan elemen ke-7: 22
Masukkan elemen ke-8: 34
Masukkan elemen ke-9: 56
Masukkan elemen ke-10: 11
Masukkan elemen ke-11: 2
Masukkan elemen ke-12: 6
Masukkan elemen ke-13: 9
Masukkan elemen ke-14: 34
Masukkan elemen ke-15: 77
Masukkan elemen ke-16: 90
Masukkan elemen ke-17: 12
Masukkan elemen ke-18: 78
Masukkan elemen ke-19: 33
Masukkan elemen ke-20: 8

Array yang telah diurutkan:  1 2 3 6 8 9 11 12 22 23 33 34 34 45 56 67 77 78 90 90
34654760 microseconds
```

- Kompleksitas waktu:

  Durasi waktu yang dibutuhkan untuk 20 input: 34654760 ms = 34.65476 s

  Big-O = Big-Ω = Big-θ = n * log n

Nomor 2

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:

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

- 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++

Jawab:

- Program

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

using namespace std;
using namespace std::chrono;

void selectionSort (int arr[], int n){
      int i, 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, i;
    high_resolution_clock::time_point   t1   =   high_resolution_
clock::now();
```

```
	cout<<"Masukkan jumlah elemen data yang ingin diurutkan: ";
	cin>>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<<endl<<duration<<" microseconds" <<endl;
}
```

- Output (input n = 20):

```
Masukkan jumlah elemen data yang ingin diurutkan: 20
Masukkan elemen ke-1: 1
Masukkan elemen ke-2: 3
Masukkan elemen ke-3: 45
Masukkan elemen ke-4: 67
Masukkan elemen ke-5: 23
Masukkan elemen ke-6: 90
Masukkan elemen ke-7: 22
Masukkan elemen ke-8: 34
Masukkan elemen ke-9: 56
Masukkan elemen ke-10: 11
Masukkan elemen ke-11: 2
Masukkan elemen ke-12: 6
Masukkan elemen ke-13: 9
Masukkan elemen ke-14: 34
Masukkan elemen ke-15: 77
Masukkan elemen ke-16: 90
Masukkan elemen ke-17: 12
Masukkan elemen ke-18: 78
Masukkan elemen ke-19: 33
Masukkan elemen ke-20: 8

Array yang telah diurutkan:  1 2 3 6 8 9 11 12 22 23 33 34 34 45 56 67 77 78 90 90
57061898 microseconds
```

- Kompleksitas waktu:

  Durasi waktu yang dibutuhkan untuk 20 input: 57061898 ms = 57.061898 s

  Big-O = Big-Ω = Big-θ = n^2

Nomor 3

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:

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

- 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++

Jawab:

- Program

```
#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, i, num;
      list *head = new list;
      head = NULL;

      high_resolution_clock::time_point  t1  =  high_resolution_
clock::now();
      cout<<"Masukkan jumlah elemen data yang ingin diurutkan: ";
      cin>>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<<endl<<duration<<" microseconds" <<endl;
}
```

- Output (input n = 20):

```
Masukkan jumlah elemen data yang ingin diurutkan: 20
Masukkan elemen ke-1: 1
Masukkan elemen ke-2: 3
Masukkan elemen ke-3: 45
Masukkan elemen ke-4: 67
Masukkan elemen ke-5: 23
Masukkan elemen ke-6: 90
Masukkan elemen ke-7: 22
Masukkan elemen ke-8: 34
Masukkan elemen ke-9: 56
Masukkan elemen ke-10: 11
Masukkan elemen ke-11: 2
Masukkan elemen ke-12: 6
Masukkan elemen ke-13: 9
Masukkan elemen ke-14: 34
Masukkan elemen ke-15: 77
Masukkan elemen ke-16: 90
Masukkan elemen ke-17: 12
Masukkan elemen ke-18: 78
Masukkan elemen ke-19: 33
Masukkan elemen ke-20: 8

Array yang telah diurutkan:  1 2 3 6 8 9 11 12 22 23 33 34 34 45 56 67 77 78 90 90
60977387 microseconds
```

- Kompleksitas waktu:

  Durasi waktu yang dibutuhkan untuk 20 input: 60977387 ms = 60.977387 s

  Big-O = n

  Big-Ω = Big-θ = n^2

Nomor 4

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:

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

- 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++

Jawab:

- Program

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

using namespace std;
using namespace std::chrono;

void bubbleSort (int arr[], int n){
	int i, 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, i;
	high_resolution_clock::time_point t1 = high_resolution_
clock::now();
	cout<<"Masukkan jumlah elemen data yang ingin diurutkan: ";
```

```
	cin>>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<<endl<<duration<<" microseconds" <<endl;
}
```

- Output (input n = 20):

```
Masukkan jumlah elemen data yang ingin diurutkan: 20
Masukkan elemen ke-1: 1
Masukkan elemen ke-2: 3
Masukkan elemen ke-3: 45
Masukkan elemen ke-4: 67
Masukkan elemen ke-5: 23
Masukkan elemen ke-6: 90
Masukkan elemen ke-7: 22
Masukkan elemen ke-8: 34
Masukkan elemen ke-9: 56
Masukkan elemen ke-10: 11
Masukkan elemen ke-11: 2
Masukkan elemen ke-12: 6
Masukkan elemen ke-13: 9
Masukkan elemen ke-14: 34
Masukkan elemen ke-15: 77
Masukkan elemen ke-16: 90
Masukkan elemen ke-17: 12
Masukkan elemen ke-18: 78
Masukkan elemen ke-19: 33
Masukkan elemen ke-20: 8

Array yang telah diurutkan:  1 2 3 6 8 9 11 12 22 23 33 34 34 45 56 67 77 78 90 90
54063391 microseconds
```

- Kompleksitas waktu:

  Durasi waktu yang dibutuhkan untuk 20 input: 54063391 ms = 54.063391 s

  Big-O = n

  Big-Ω = Big-θ = n^2