Export to Golang

a. Experiment1

```go
package main

import (
    "fmt"
    "time"
)

func findCompany(array []string) {
    tx := time.Now()
    for i := 0; i < len(array); i++ {
        if array[i] == "telkom" {
            fmt.Println("Company Found!")
        }
    }
    ty := time.Now()
    fmt.Printf("The performance is %v ms", (ty.Sub(tx).Seconds()))
}

func main() {
    company := "telkom"
    var largeCompanyName []string
    for i := 0; i < 10; i++ {
        largeCompanyName = append(largeCompanyName, company)
    }
    findCompany(largeCompanyName)
}
```

Performance result = 0.0059 ms

- The code defines a function named **findCompany** that takes an array of strings as its argument.
- The function uses a loop to iterate through the array and checks if any of the strings in the array match the string "telkom".
- The function also measures the time it takes to perform the search using the **time.Now()** function, which returns the current time, and calculates the time difference using **the Sub()** method of the time.Duration type.
- The **main** function initializes an empty array named **largeCompanyName** and populates it with the string "telkom" using a loop that runs **ten** times.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment1.go
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
The performance is 0.0059464 ms
```

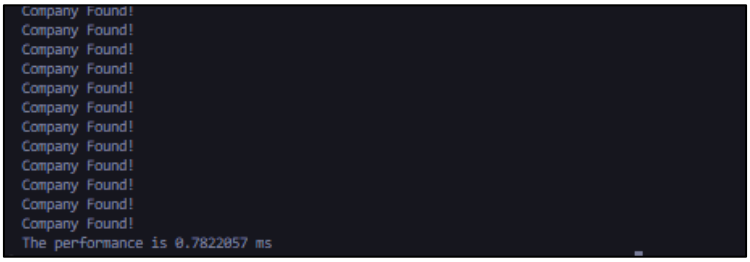
b. Experiment 2

```go
package main

import (
    "fmt"
    "time"
)

func findCompany(array []string) {
    tx := time.Now()
    for i := 0; i < len(array); i++ {
        if array[i] == "telkom" {
            fmt.Println("Company Found!")
        }
    }
    ty := time.Now()
    fmt.Printf("The performance is %v ms", (ty.Sub(tx).Seconds()))
}

func main() {
    company := "telkom"
    var largeCompanyName []string
    for i := 0; i < 1000; i++ {
        largeCompanyName = append(largeCompanyName, company)
    }
    findCompany(largeCompanyName)
}
```

- Same as experiment 1, only for the loop in the main function, it iterates 1000 times.

Performance result = 0.78 ms	

Execution result :



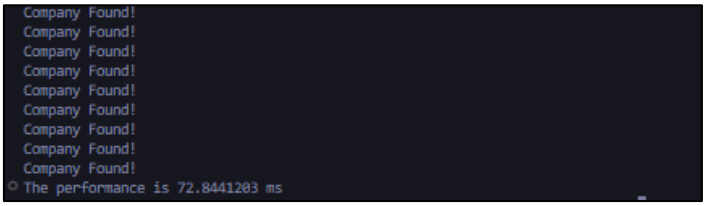
c. Experiment 3

```go
package main

import (
    "fmt"
    "time"
)

func findCompany(array []string) {
    tx := time.Now()
    for i := 0; i < len(array); i++ {
        if array[i] == "telkom" {
            fmt.Println("Company Found!")
        }
    }
    ty := time.Now()
    fmt.Printf("The performance is %v ms", (ty.Sub(tx).Seconds()))
}

func main() {
    company := "telkom"
    var largeCompanyName []string
    for i := 0; i < 100000; i++ {
        largeCompanyName = append(largeCompanyName, company)
    }
    findCompany(largeCompanyName)
}
```

Performance result = 72.844 ms

- Same as experiment 1, only for the loop in the main function, it iterates 100000 times.
- Of the three experiments, it was **concluded** that the **more iterations** were carried out, the **longer** it took for program execution.
- In experiments 1, 2, and 3 the complexity is O(n)

Execution result :



d. Experiment 4

```go
package main

import (
    "fmt"
    "time"
)

func findAddress(addresses []string) {
    tx := time.Now()
    fmt.Println("The Default Address is", addresses[0])
    ty := time.Now()
    fmt.Println("The performance is", ty.Sub(tx).Seconds(), "ms")
}

func main() {
    address := "DKI Jakarta"
    var addresses []string
    for i := 0; i < 10; i++ {
        addresses = append(addresses, address)
    }
    findAddress(addresses)
}
```

Performance result = 0.0005117 ms

- The code defines a function named **findAddress** that takes a slice of strings as its argument.
- The function also measures the time it takes to perform the search using the **time.Now()** function, which returns the current time, and calculates the time difference using **the Sub()** method of the time.Duration type.
- A slice of strings named **addresses** is created by appending the **address** variable to itself 10 times

	using a **for** loop and the **append** function.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment4.go
The Default Address is DKI Jakarta
The performance is 0.0005117 ms
```

e. Experiment 5

```go
package main

import (
    "fmt"
    "time"
)

func findAddress(addresses []string) {
    tx := time.Now()
    fmt.Println("The Default Address is", addresses[0])
    ty := time.Now()
    fmt.Println("The performance is", ty.Sub(tx).Seconds(), "ms")
}

func main() {
    address := "DKI Jakarta"
    var addresses []string
    for i := 0; i < 1000; i++ {
        addresses = append(addresses, address)
    }
    findAddress(addresses)
}
```

Performance result = 0.0005117 ms

- Same as experiment 4, only for the loop in the main function, it iterates 1000 times.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment5.go
The Default Address is DKI Jakarta
The performance is 0.0005117 ms
```

f. Experiment 6

```go
package main

import (
    "fmt"
    "time"
)

func findAddress(addresses []string) {
    tx := time.Now()
    fmt.Println("The Default Address is", addresses[0])
    ty := time.Now()
    fmt.Println("The performance is", ty.Sub(tx).Seconds(), "ms")
}

func main() {
    address := "DKI Jakarta"
    var addresses []string
    for i := 0; i < 100000; i++ {
        addresses = append(addresses, address)
    }
    findAddress(addresses)
}
```

Performance result = 0.0005065 ms

- Same as experiment 4, only for the loop in the main function, it iterates 100000 times.
- The increase in the amount of data has no effect on the performance / program execution time. The time remains at 0.0005 ms.
- Experiment 4, 5, and 6 have O(1) complexity.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment6.go
The Default Address is DKI Jakarta
The performance is 0.0005065 ms
```

g. Experiment 7

```go
package main

import "fmt"

func logPairs(array1 []string, array2 []string, words string) {
    counter := 0
    for i := 0; i < len(array1); i++ {
        for j := 0; j < len(array2); j++ {
            counter++
            fmt.Println(words, counter, array1[i], "dan", array2[j])
        }
    }
}

func main() {
    foods := []string{"Sate", "Bakso", "Dimsum", "Rames"}
    drinks := []string{"Jeruk", "Teh", "Kelapa", "Cendol"}
    logPairs(foods, drinks, "Menu")
}
```

- The code defines a function called **logPairs** that takes in two string arrays and a string as arguments.
- The function prints out a message with the given string parameter, and then loops through the elements of the two arrays and prints out pairs of elements with an incrementing counter.
- The number of foods is 4 and number of drinks is 4. And the results obtained are 16 menus.
- It can be concluded that the complexity of experiment 7 is O(n2).

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment7.go
Menu 1 Sate dan Jeruk
Menu 2 Sate dan Teh
Menu 3 Sate dan Kelapa
Menu 4 Sate dan Cendol
Menu 5 Bakso dan Jeruk
Menu 6 Bakso dan Teh
Menu 7 Bakso dan Kelapa
Menu 8 Bakso dan Cendol
Menu 9 Dimsum dan Jeruk
Menu 10 Dimsum dan Teh
Menu 11 Dimsum dan Kelapa
Menu 12 Dimsum dan Cendol
Menu 13 Rames dan Jeruk
Menu 14 Rames dan Teh
Menu 15 Rames dan Kelapa
Menu 16 Rames dan Cendol
```

h. Experiment 8

```go
package main

import "fmt"

var results [][]string

func arrange(array []string, memory []string) [][]string {
    if memory == nil {
        memory = make([]string, 0)
    }
    for i := 0; i < len(array); i++ {
        current := []string{array[i]}
        if len(array) == 1 {
            results = append(results, append(memory, current...))
        } else {
            subarray := make([]string, len(array)-1)
            copy(subarray, array[:i])
            copy(subarray[i:], array[i+1:])
            newMemory := append(memory, current...)
            arrange(subarray, newMemory)
        }
    }
    return results
}

func main() {
    candidates := []string{"Baswedan", "Subianto", "Maharani"}
    chairs := arrange(candidates, nil)
    for _, chair := range chairs {
        fmt.Println(chair)
    }
}
```

- The program defines a function called arrange which takes an array of strings and an optional memory slice of strings and returns the results slice.
- It loops through the array of strings and creates a new slice containing the current string.
- It then recursively calls the arrange function with the new subarray and memory slice.
- In the main function, a loop is used to iterate through the chairs slice, and each chair is printed to the console.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment8.go
[Baswedan Subianto Maharani]
[Baswedan Maharani Subianto]
[Subianto Baswedan Maharani]
[Subianto Maharani Baswedan]
[Maharani Baswedan Subianto]
[Maharani Subianto Baswedan]
```

i. Experiment 9

```go
package main

import "fmt"

var results [][]string

func arrange(array []string, memory []string) [][]string {
    if memory == nil {
        memory = make([]string, 0)
    }
    for i := 0; i < len(array); i++ {
        current := []string{array[i]}
        if len(array) == 1 {
            results = append(results, append(memory, current...))
        } else {
            subarray := make([]string, len(array)-1)
            copy(subarray, array[:i])
            copy(subarray[i:], array[i+1:])
            newMemory := append(memory, current...)
            arrange(subarray, newMemory)
        }
    }
    return results
}

func main() {
    candidates := []string{"Baswedan", "Subianto", "Maharani", "Ganjar
"}
    chairs := arrange(candidates, nil)
    for _, chair := range chairs {
        fmt.Println(chair)
    }
}
```

- Same as experiment 8, but this time the contents of the candidates were 4. And more combinations were obtained.
- In experiment 8, there is 3 data, and the results are 3! = 6
- In experiment 9, there are 4 data, and the results are 4! = 24
- It can be concluded that with a recursive approach, experiments 8 and 9, the complexity is O(n!).

Execution result :

```
[Maharani Ganjar Baswedan Subianto]
[Maharani Ganjar Subianto Baswedan]
[Ganjar Baswedan Subianto Maharani]
[Ganjar Baswedan Maharani Subianto]
[Ganjar Subianto Baswedan Maharani]
[Ganjar Subianto Maharani Baswedan]
[Ganjar Maharani Baswedan Subianto]
[Ganjar Maharani Subianto Baswedan]
```

j. Experiment 10

```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func findCompany(array []string, input int) {
    for i := 0; i < len(array); i++ {
        if i == input {
            fmt.Printf("Company Found: %s\n", array[input])
            break
        }
        fmt.Printf("Searching Company... %d\n", i+1)
    }
}

func main() {
    rand.Seed(time.Now().UnixNano())
    telco := []string{"Telkom", "Indosat", "XL", "Verizon", "AT&T", "
Nippon", "Vodafone", "Orange", "KDDI", "Telefonica", "T-Mobile"}
    company := rand.Intn(11)
    findCompany(telco, company)
}
```

- The program defines a function named "findCompany" that takes two parameters: an array of strings and an integer.
- The function iterates through the array using a for loop, checking if the current index equals the given integer.
- The main function generates a random integer between 0 and 10 using the time as a seed.
- It then calls the findCompany function with the randomly generated integer and an array of company names.

Execution result :

```
PS F:\00_KULIAH\TALENT NURTURING TELKOM\Teaching Factory\Task3\source_code> go run experiment10.go
Searching Company... 1
Searching Company... 2
Searching Company... 3
Searching Company... 4
Searching Company... 5
Company Found: Nippon
```

k. Experiment 11

```go
package main

import "fmt"

func generateData() []string {
    number := "0821234567"
    var customers []string
    for i := 0; i < 100; i++ {
        var mobileNumber string
        if i < 10 {
            mobileNumber = number + "0" + fmt.Sprintf("%d", i)
        } else {
            mobileNumber = number + fmt.Sprintf("%d", i)
        }
        customers = append(customers, mobileNumber)
    }
    return customers
}

func sendPromoDiscount(array []string) {
    for i := 0; i < len(array); i++ {
        fmt.Printf("Sending Promo to %s\n", array[i])
    }
    for i := 0; i < 10; i++ {
        fmt.Printf("Sending Discount to Chosen Customer %d\n", i+1)
    }
}

func main() {
    customers := generateData()
    sendPromoDiscount(customers)
}
```

- This program generates a list of 100 mobile numbers as a string using the function generateData().
- The function sendPromoDiscount() is called and passed the list of generated mobile numbers.
- The sendPromoDiscount() function iterates through the list of mobile numbers and prints a message for each number indicating that a promo is being sent to that number.
- After iterating through all mobile numbers, the function then sends a discount to 10 chosen customers by printing a message for each chosen customer.

Execution result :

```
Sending Promo to 082123456798
Sending Promo to 082123456799
Sending Discount to Chosen Customer 1
Sending Discount to Chosen Customer 2
Sending Discount to Chosen Customer 3
Sending Discount to Chosen Customer 4
Sending Discount to Chosen Customer 5
Sending Discount to Chosen Customer 6
Sending Discount to Chosen Customer 7
Sending Discount to Chosen Customer 8
Sending Discount to Chosen Customer 9
Sending Discount to Chosen Customer 10
```

l. Experiment 12

```go
package main

import "fmt"
import "strconv"

func generateData(n int) []string {
    var baseNumber string = "082"
    customers := make([]string, n)
    for i := 0; i < n; i++ {
        mobileNumber := baseNumber + strconv.Itoa(i+1)
        customers[i] = mobileNumber
    }
    return customers
}

func sendPromoDiscount(customers []string) {
    for i := 0; i < len(customers); i++ {
        fmt.Printf("Sending Promo to %s\n", customers[i])
    }
    fmt.Printf("It's finished sending Promo to %d customers\n", len(
customers))

    for i := 0; i < len(customers); i++ {
        fmt.Printf("Sending Discount to %s\n", customers[i])
    }
    fmt.Printf("It's finished sending Discount to %d customers\n", len
(customers))
}

func main() {
    data := generateData(1000)
    sendPromoDiscount(data)
}
```

- Function ‘generateData’ takes an integer n as input and returns the ‘customers’ slice.
- It will loop through **n** and sets each element of **customers** to a string consisting of **baseNumber** and the current number plus one, converted to a string using **strconv.Itoa**

Execution result :

```
Sending Discount to 082994
Sending Discount to 082995
Sending Discount to 082996
Sending Discount to 082997
Sending Discount to 082998
Sending Discount to 082999
Sending Discount to 0821000
It's finished sending Discount to 1000 customers
```

m. Experiment 13

```go
package main

import (
    "fmt"
    "strconv"
)

func generateData(n int) []string {
    baseNumber := "082"
    customers := make([]string, n)
    for i := 0; i < n; i++ {
        mobileNumber := baseNumber + strconv.Itoa(i+1)
        customers[i] = mobileNumber
    }
    return customers
}

func sendPromoDiscount(data1 []string, data2 []string) {
    for _, customer := range data1 {
        fmt.Printf("Sending Promo to %v\n", customer)
    }
    fmt.Printf("It's finished sending Promo to %v customers\n", len(
data1))

    for _, customer := range data2 {
        fmt.Printf("Sending Discount to %v\n", customer)
    }
    fmt.Printf("It's finished sending Discount to %v customers\n", len
(data2))
}

func main() {
    dataPromo := generateData(100000000)
    dataDiscount := generateData(1000)
    sendPromoDiscount(dataPromo, dataDiscount)
}
```

- Same as the previous experiment, but this time, the program will generate 100 million phone number data and store them in the dataPromo slice.
- Then, the program generates 1000 numbers and stores them in the dataDiscount slice.
- SendPromoDiscount function, will receive 2 parameters, slice from dataPromo and slice from dataDiscount.
- Each parameter will be iterated using a loop.

Execution result :

Because the data is too big, I can't get the result of program using vs code. So there is no successful screenshot because the program stops in the middle due to a warning that the data is too much to process.

n. Big-O Calculation Example

```go
func calculateBigO(input []int) bool {
    l := 5
    p := 8
    k := (x * y)

    for i := 0; i < len(input); i++ {
        addInput()
        calculateNewNumber()
    }
    return true
}
```

- The function is called “calculateBigO”, takes an input array of integers and returns a Boolean value.
- The function loops through the input array and calls two functions “addInput” and “calculateNewNumber” for each element.

o. Hash Table Implementation

```go
package main

import (
    "fmt"
)

type HashTable struct {
    data [][]string
}

func NewHashTable(size int) *HashTable {
    return &HashTable{make([][]string, size)}
}

func (h *HashTable) _hash(key string) int {
    hash := 0
    for i := 0; i < len(key); i++ {
        hash = (hash + int(key[i])*i) % len(h.data)
    }
    return hash
}

func (h *HashTable) Set(key, value string) {
    address := h._hash(key)
    if h.data[address] == nil {
        h.data[address] = []string{}
    }
    h.data[address] = append(h.data[address], key, value)
}

func (h *HashTable) Get(key string) string {
    address := h._hash(key)
    currentBucket := h.data[address]
    if currentBucket != nil {
        for i := 0; i < len(currentBucket); i += 2 {
            if currentBucket[i] == key {
                return currentBucket[i+1]
            }
        }
    }
    return ""
}

func main() {
    myHashTable := NewHashTable(100)
    myHashTable.Set("082124606606", "Rony Setyawan")
    fmt.Println(myHashTable.Get("082124606606"))
}
```

The result :

```
go run /tmp/AzzWmGQAHg.go
Rony Setyawan
```

- This program creates a hash table data structure with “data” field as a 2D slice of string slices.
- NewHashTable() function takes an integer as input and returns a pointer to a new HashTable with an empty data slice with the specified size.
- _hash() function takes a string as input and returns an integer that is the hashed value of the string.
- Set() function takes two string inputs (key and value), it gets the hash address of the key by calling _hash() function and then stores the key and value in the data slice at the hashed address.
- Get() function takes a string key as input, it gets the hash address of the key by calling _hash() function and then retrieves the value associated with the key from the data slice at the hashed address. It returns an empty string if the key is not found.

p. Dynamic Programming

```go
package main

import (
    "fmt"
)

var counterPlainRecursive = 0
var counterDynamicProgramming = 0

func fibonacciPlainRecursive(n int) int {
    counterPlainRecursive++
    if n < 2 {
        return n
    }
    return fibonacciPlainRecursive(n-1) + fibonacciPlainRecursive(n-2)
}

func fibonacciDynamicProgramming() func(int) int {
    cache := make(map[int]int)
    var fib func(int) int
    fib = func(n int) int {
        counterDynamicProgrammin ++
g       if val, ok := cache[n]; ok {
            return val
        }
        if n < 2 {
            return n
        }
        result := fib(n-1) + fib(n-2)
        cache[n] = result
        return result
    }
    return fib
}

func main() {
    fibonacciPlainRecursive(20)
    fasterFibonacci := fibonacciDynamicProgramming()
    fasterFibonacci(20)
    fmt.Println("we did", counterPlainRecursive, "calculations for Plain Recursiv  )
    fmt.Println("we did", counterDynamicProgrammin
"calculations for Dynamic Programmin  )
"
```

- The **Plain Recursive** function uses a recursive algorithm to calculate the nth number in the sequence. If n is less than 2, it returns n. Otherwise, it recursively calls itself with n-1 and n-2 as arguments.
- The **Dynamic Programming** function creates a **map** called **cache** to **store the results of previous calculations**. If the result for n is **already** in the **cache**, it **returns that value**. Otherwise, it calculates the result using a recursive call to itself, stores the result in the cache, and returns the result.

Helsa Nesta – Naily Khairiya
Institut Teknologi Sepuluh Nopember

CODE CONVERSION JAVASCRIPT TO JAVA

A. EXPERIMENT 1

```java
import java.util.Arrays;

class experiment1 {
    public static void main(String[] args) {
        String company = "telkom";
        String[] companies = new String[10];
        Arrays.fill(companies, company);
        findCompany(companies);
    }

    public static void findCompany(String[] array) {
        long tx = System.nanoTime();
        for (int i = 0; i < array.length; i++) {
            if (array[i] == "telkom") {
                System.out.println("Company Found!");
            }
        }
        long ty = System.nanoTime();
        double duration = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + duration + "
ms")}
}
```

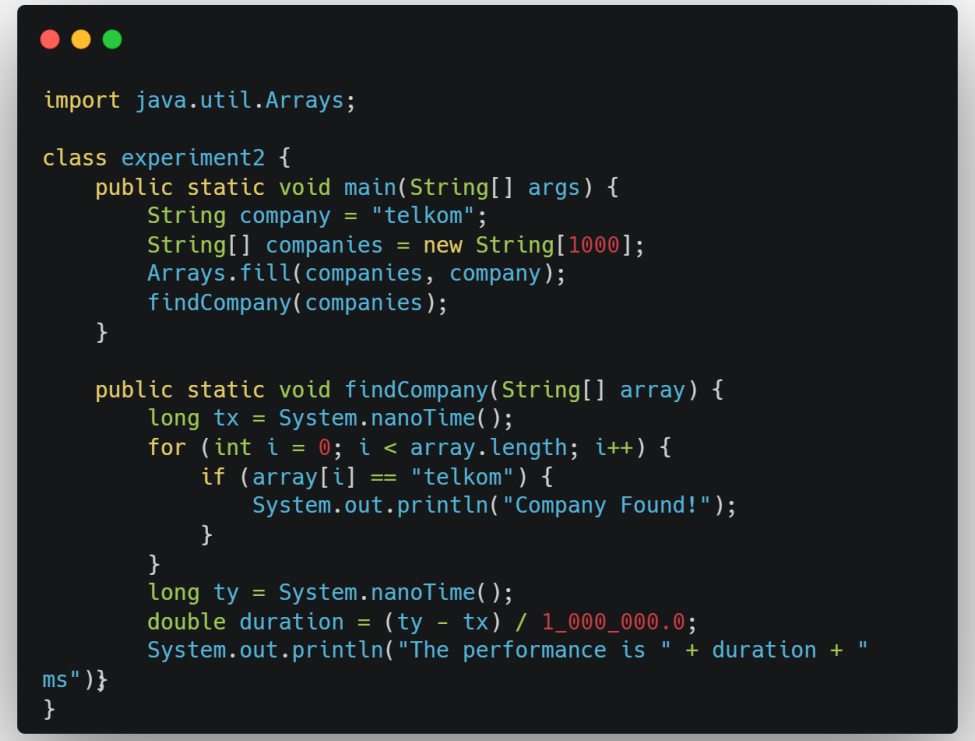
Explanation :

- The program above is to find a company within an array of companies.
- For this case, the company being searched for is "Telkom".
- Initially, an array of length 10 will be filled with "company" using "Arrays.fill".
- Then, the search is performed by calling the function findCompany().
- The findCompany() function searches through the array using a "for" loop.
- If the array element is equal to "Telkom", the system will print "Company found!"
- Since the array has already been filled with "Telkom", the system will continue to print "Company found!" throughout the entire length of the array.
- Within the findCompany() function, performance will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.
- Complexity findCompany :
  T(findCompany) = T(1) + T(1) * array.length + T(1) + T(1) + T(1)
  = T(1) * n
  = O(n)

Execution Result :

```
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
The performance is 0.884379 ms
```

B. EXPERIMENT 2



```java
import java.util.Arrays;

class experiment2 {
    public static void main(String[] args) {
        String company = "telkom";
        String[] companies = new String[1000];
        Arrays.fill(companies, company);
        findCompany(companies);
    }

    public static void findCompany(String[] array) {
        long tx = System.nanoTime();
        for (int i = 0; i < array.length; i++) {
            if (array[i] == "telkom") {
                System.out.println("Company Found!");
            }
        }
        long ty = System.nanoTime();
        double duration = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + duration + "
ms")}
}
```

Explanation :

- Same as experiment 1, but the length of the array is 1000.
- The program above is to find a company within an array of companies.
- For this case, the company being searched for is "Telkom".
- Initially, an array of length 10 will be filled with "company" using "Arrays.fill".
- Then, the search is performed by calling the function findCompany().
- The findCompany() function searches through the array using a "for" loop.
- If the array element is equal to "Telkom", the system will print "Company found!"

- Since the array has already been filled with "Telkom", the system will continue to print "Company found!" throughout the entire length of the array.
- Within the findCompany() function, performance will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.
- The performance is longer than experiment 1
- Complexity findCompany :
  T(findCompany) = T(1) + T(1) * array.length + T(1) + T(1) + T(1)
  = T(1) * n
  = O(n)

Execution result :

```
...
...
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
The performance is 50.84236 ms
```

C. EXPERIMENT 3

```java
import java.util.Arrays;

class experiment3 {
    public static void main(String[] args) {
        String company = "telkom";
        String[] companies = new String[100000];
        Arrays.fill(companies, company);
        findCompany(companies);
    }

    public static void findCompany(String[] array) {
        long tx = System.nanoTime();
        for (int i = 0; i < array.length; i++) {
            if (array[i] == "telkom") {
                System.out.println("Company Found!");
            }
        }
        long ty = System.nanoTime();
        double duration = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + duration + "
ms")}
}
```

Explanation :

- Same as experiment 1 and also experiment 2, but the length of the array is 100000.
- The program above is to find a company within an array of companies.
- For this case, the company being searched for is "Telkom".
- Initially, an array of length 10 will be filled with "company" using "Arrays.fill".
- Then, the search is performed by calling the function findCompany().
- The findCompany() function searches through the array using a "for" loop.
- If the array element is equal to "Telkom", the system will print "Company found!"
- Since the array has already been filled with "Telkom", the system will continue to print "Company found!" throughout the entire length of the array.
- Within the findCompany() function, performance will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.
- The performance is longer than experiment 1 and longer than experiment 2
- Complexity findCompany :
  T(findCompany) = T(1) + T(1) * array.length + T(1) + T(1) + T(1)
  = T(1) * n
  = O(n)

Execution result :

...

...

```
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
Company Found!
The performance is 222.140282 ms
```

D. EXPERIMENT 4

```java
import java.util.Arrays;

class experiment4 {
    public static void main(String[] args) {
        String address = "DKI Jakarta";
        String[] addresses = new String[10];
        Arrays.fill(addresses, address);
        findAddress(addresses);
    }

    public static void findAddress(String[] addresses) {
        long tx = System.nanoTime();
        System.out.println("The Default Address is " + (addresses[0]));
        long ty = System.nanoTime();
        double performance = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + performance + "
ms")}
}
```

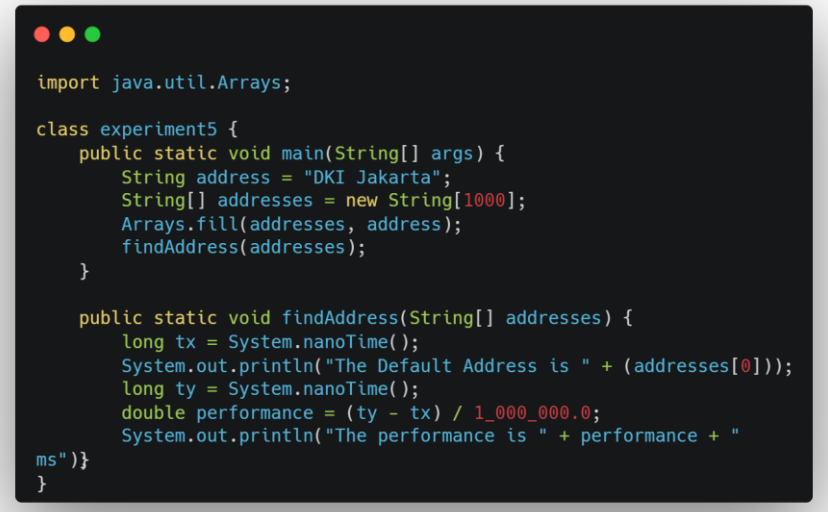
Explanation :

- The program code is to search for an address within an array.
- After filling the array, it performs a search using the findAddress function.
- Within the findAddress function, there is no iteration for searching, but it directly prints the address[0].
- Therefore, the length of the array does not affect it.
- The complexity of findAddress will always be O(1).
- Performance within the findAddress() function will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.

Execution result :

```
The Default Address is DKI Jakarta
The performance is 44.638599 ms
```

E. EXPERIMENT 5



```java
import java.util.Arrays;

class experiment5 {
    public static void main(String[] args) {
        String address = "DKI Jakarta";
        String[] addresses = new String[1000];
        Arrays.fill(addresses, address);
        findAddress(addresses);
    }

    public static void findAddress(String[] addresses) {
        long tx = System.nanoTime();
        System.out.println("The Default Address is " + (addresses[0]));
        long ty = System.nanoTime();
        double performance = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + performance + "
ms")}
}
```

Explanation :

- The program code is to search for an address within an array.
- After filling the array, it performs a search using the findAddress function.
- Within the findAddress function, there is no iteration for searching, but it directly prints the address[0].
- Therefore, the length of the array does not affect it.
- The complexity of findAddress will always be O(1).
- Performance within the findAddress() function will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.

Execution result :

```
The Default Address is DKI Jakarta
The performance is 39.273195 ms
```

F. EXPERIMENT 6

```java
import java.util.Arrays;

class experiment6 {
    public static void main(String[] args) {
        String address = "DKI Jakarta";
        String[] addresses = new String[100000];
        Arrays.fill(addresses, address);
        findAddress(addresses);
    }

    public static void findAddress(String[] addresses) {
        long tx = System.nanoTime();
        System.out.println("The Default Address is " + (addresses[0]));
        long ty = System.nanoTime();
        double performance = (ty - tx) / 1_000_000.0;
        System.out.println("The performance is " + performance + "
ms")}
}
```

Explanation :

- The program code is to search for an address within an array.
- After filling the array, it performs a search using the findAddress function.
- Within the findAddress function, there is no iteration for searching, but it directly prints the address[0].
- Therefore, the length of the array does not affect it.
- The complexity of findAddress will always be O(1).
- Performance within the findAddress() function will be checked using nanoTime() at the beginning and end. Performance is obtained by taking the time difference and dividing it by 1,000,000 to get the value in milliseconds.

Execution Result :

```
The Default Address is DKI Jakarta
The performance is 40.308514 ms
```

G. EXPERIMENT 7

```java
class experiment7 {
    public static void main(String[] args) {
        String[] foods = { "Sate", "Bakso", "Dimsum", "Rames" };
        String[] drinks = { "Jeruk", "Teh", "Kelapa", "Cendol" };
        logPairs(foods, drinks, "Menu");
    }

    public static void logPairs(String[] array1, String[] array2, String words) {
        int counter = 0;
        for (int i = 0; i < array1.length; i++) {
            for (int j = 0; j < array2.length; j++) {
                counter++;
                System.out.println(words + ' ' + counter + ' ' + array1[i] + " dan " +
array2[j]); }
        }
    }
}
```

Explanation :

- The main method initializes two arrays of Strings named "foods" and "drinks" with several food and drink options respectively.
- After that, the method "logPairs" is called with the "foods" and "drinks" arrays, and the string "Menu" as its parameters. The purpose of this method is to print out all possible pairs of elements from the two arrays.
- The method accomplishes this by using two nested for loops that iterate over each element of both arrays. A counter is used to keep track of the number of pairs that are printed out.
- For each pair, the method prints out the string "Menu", the current value of the counter, and the two elements from the arrays that make up the pair.
- The output of running this program would be a list of all possible combinations of food and drink items, each labeled with a counter value and the string "Menu".
- Complexity : T(1) * length_array1 * T(1) * length_array2 = O(n*m) = O(nm)

Execution result :

```
Menu 1 Sate dan Jeruk
Menu 2 Sate dan Teh
Menu 3 Sate dan Kelapa
Menu 4 Sate dan Cendol
Menu 5 Bakso dan Jeruk
Menu 6 Bakso dan Teh
Menu 7 Bakso dan Kelapa
Menu 8 Bakso dan Cendol
Menu 9 Dimsum dan Jeruk
Menu 10 Dimsum dan Teh
Menu 11 Dimsum dan Kelapa
Menu 12 Dimsum dan Cendol
Menu 13 Rames dan Jeruk
Menu 14 Rames dan Teh
Menu 15 Rames dan Kelapa
Menu 16 Rames dan Cendol
```

H. EXPERIMENT 8

```java
import java.util.ArrayList;
import java.util.Arrays;

class experiment8 {
    public static ArrayList<ArrayList<String>> results = new ArrayList<>();
    public static String[] candidates = { "Baswedan", "Subianto", "Maharani" };

    public static void main(String[] args) {
        ArrayList<ArrayList<String>> chairs = arrange(candidates);
        System.out.println(Arrays.deepToString(chairs.toArray()));
    }

    public static ArrayList<ArrayList<String>> arrange(String[] array) {
        return arrange(array, new ArrayList<>());
    }

    public static ArrayList<ArrayList<String>> arrange(String[] array, ArrayList<String> memory)
{       String current;
        ArrayList<String> newMemory;
        if (memory == null) {
            newMemory = new ArrayList<>();
        } else {
            newMemory = new ArrayList<>(memory);
        }
        for (int i = 0; i < array.length; i++) {
            current = array[i];
            ArrayList<String> newArray = new ArrayList<>(Arrays.asList(array));
            newArray.remove(i);
            if (array.length == 1) {
                newMemory.add(current);
                results.add(new ArrayList<>(newMemory));
                return results;
            }
            newMemory.add(current);
            arrange(newArray.toArray(new String[0]), newMemory);
            newArray.add(i, current);
            newMemory.remove(newMemory.size() - 1);
        }
        return results;
    }
}
```

Explanation :

- At the beginning of the class, there are two static variables declared - "results" and "candidates". "results" is an ArrayList that will be used to store the final result, while "candidates" is an array of Strings that contains the candidates to be arranged.
- The main method initializes an ArrayList named "chairs" by calling the "arrange" method with the "candidates" array as its parameter. It then prints out the result of the method using the "Arrays.deepToString()" method.
- The "arrange" method is called with an array of Strings and an ArrayList of Strings named "memory". If "memory" is null, a new ArrayList is created, otherwise a copy of the "memory" ArrayList is made.
- The method then iterates over each element of the input array using a for loop. For each element, it creates a new ArrayList named "newArray" that contains all elements of the input array except for the current one.
- Otherwise, the current element is added to the "newMemory" ArrayList and the "arrange" method is called recursively with the "newArray" and "newMemory" parameters.

- After the recursive call, the current element is added back to the "newArray" ArrayList and the current element is removed from the "newMemory" ArrayList. The method then moves on to the next element of the input array.
- Overall, this program generates all possible permutations of the elements in the "candidates" array using a recursive approach. The resulting permutations are stored in the "results" ArrayList and printed out in the main method.
- In the worst-case scenario, where all candidates are distinct, the number of permutations is equal to n factorial (n!). This means that the program must perform n! recursive calls to generate all possible permutations.
- Each recursive call iterates over n elements, and for each element, it creates a new ArrayList and calls the "arrange" method recursively again. The time complexity of each recursive call is O(n).
- Therefore, the total time complexity of the program is O(n! * n) which is simplified to O(n!).

Execution result :

```
[[Baswedan, Subianto, Maharani], [Baswedan, Maharani, Subianto], [Subianto, Baswedan, Maharani], [Subian
to, Maharani, Baswedan], [Maharani, Baswedan, Subianto], [Maharani, Subianto, Baswedan]]
```

I. EXPERIMENT 9

```java
import java.util.ArrayList;
import java.util.Arrays;

class experiment9 {
    public static ArrayList<ArrayList<String>> results = new ArrayList<>();
    public static String[] candidates = { "Baswedan", "Subianto", "Maharani", "Ganjar" };

    public static void main(String[] args) {
        ArrayList<ArrayList<String>> chairs = arrange(candidates);
        System.out.println(Arrays.deepToString(chairs.toArray()));
    }

    public static ArrayList<ArrayList<String>> arrange(String[] array) {
        return arrange(array, new ArrayList<>());
    }

    public static ArrayList<ArrayList<String>> arrange(String[] array, ArrayList<String> memory)
{       String current;
        ArrayList<String> newMemory;
        if (memory == null) {
            newMemory = new ArrayList<>();
        } else {
            newMemory = new ArrayList<>(memory);
        }
        for (int i = 0; i < array.length; i++) {
            current = array[i];
            ArrayList<String> newArray = new ArrayList<>(Arrays.asList(array));
            newArray.remove(i);
            if (array.length == 1) {
                newMemory.add(current);
                results.add(new ArrayList<>(newMemory));
                return results;
            }
            newMemory.add(current);
            arrange(newArray.toArray(new String[0]), newMemory);
            newArray.add(i, current);
            newMemory.remove(newMemory.size() - 1);
        }
        return results;
    }
}
```

Explanation :

- At the beginning of the class, there are two static variables declared - "results" and "candidates". "results" is an ArrayList that will be used to store the final result, while "candidates" is an array of Strings that contains the candidates to be arranged.
- The main method initializes an ArrayList named "chairs" by calling the "arrange" method with the "candidates" array as its parameter. It then prints out the result of the method using the "Arrays.deepToString()" method.
- The "arrange" method is called with an array of Strings and an ArrayList of Strings named "memory". If "memory" is null, a new ArrayList is created, otherwise a copy of the "memory" ArrayList is made.
- The method then iterates over each element of the input array using a for loop. For each element, it creates a new ArrayList named "newArray" that contains all elements of the input array except for the current one.
- Otherwise, the current element is added to the "newMemory" ArrayList and the "arrange" method is called recursively with the "newArray" and "newMemory" parameters.
- After the recursive call, the current element is added back to the "newArray" ArrayList and the current element is removed from the "newMemory" ArrayList. The method then moves on to the next element of the input array.
- Overall, this program generates all possible permutations of the elements in the "candidates" array using a recursive approach. The resulting permutations are stored in the "results" ArrayList and printed out in the main method.
- In the worst-case scenario, where all candidates are distinct, the number of permutations is equal to n factorial (n!). This means that the program must perform n! recursive calls to generate all possible permutations.
- Each recursive call iterates over n elements, and for each element, it creates a new ArrayList and calls the "arrange" method recursively again. The time complexity of each recursive call is O(n).

Therefore, the total time complexity of the program is O(n! * n) which is simplified to O(n!).

Execution result :

```
[[Baswedan, Subianto, Maharani, Ganjar], [Baswedan, Subianto, Ganjar, Maharani], [Baswedan, Maharani, Subianto, Ganjar], [Baswedan, Maharani, Ganjar, Subianto], [Baswedan, Ganjar, Subianto, Maharani], [Baswedan, Ganjar, Maharani, Subianto], [Subianto, Baswedan, Maharani, Ganjar], [Subianto, Baswedan, Ganjar, Maharani], [Subianto, Maharani, Baswedan, Ganjar], [Subianto, Maharani, Ganjar, Baswedan], [Subianto, Ganjar, Baswedan, Maharani], [Subianto, Ganjar, Maharani, Baswedan], [Maharani, Baswedan, Subianto, Ganjar], [Maharani, Baswedan, Ganjar, Subianto], [Maharani, Subianto, Baswedan, Ganjar], [Maharani, Subianto, Ganjar, Baswedan], [Maharani, Ganjar, Baswedan, Subianto], [Maharani, Ganjar, Subianto, Baswedan], [Ganjar, Baswedan, Subianto, Maharani], [Ganjar, Baswedan, Maharani, Subianto], [Ganjar, Subianto, Baswedan, Maharani], [Ganjar, Subianto, Maharani, Baswedan], [Ganjar, Maharani, Baswedan, Subianto], [Ganjar, Maharani, Subianto, Baswedan]]
```

J. EXPERIMENT 10

```java
class experiment10 {
    public static String[] telco = { "Telkom", "Indosat", "XL", "Verizon", "AT&T", "Nippon", "Vodafone",
"Orange",
            "KDDI", "Telefonica", "T-Mobile" };

    public static void main(String[] args) {
        int company = (int) Math.round(Math.random( ) * 10);
        findCompany(telco, company);
    }

    public static void findCompany(String[] array, int input) {
        for (int i = 0; i < array.length; i++) {
            if (array[i] == array[input]) {
                System.out.println("Company Found " + array[input]);
                break;
            }
            System.out.println("Searching Company...");
        }
    }

}
```

Explanation :

- The code is searching for a company in a string array.
- The company being searched for is randomly selected using math.random.
- The search is performed using a for loop.
- If the searched company is found, the system prints "company found" and breaks out of the loop.
- Otherwise, "Searching company" is printed at each iteration.
- The complexity of the program is O(N). However, in the best case scenario where the company being searched for is the first element in the array "companies", the complexity can be O(1).

Execution result:

(1)

```
Searching Company...
Company Found Indosat
```

(2)

```
Searching Company...
Searching Company...
Searching Company...
Searching Company...
Searching Company...
Searching Company...
Searching Company...
Company Found Orange
```

K. EXPERIMENT 11

```java
class experiment11 {
    public static String[] generateData() {
        String number = "0821234567";
        String[] customers = new String[100];
        String mobileNumber;

        for (int i = 0; i < 100; i++) {
            if (i < 10) {
                mobileNumber = number + "0" + i;
            } else {
                mobileNumber = number + i;
            }
            customers[i] = mobileNumber;
        }

        return customers;
    }

    public static void sendPromoDiscount(String[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.println("Sending Promo to " + array[i]);
        }
        for (int i = 0; i < 10; i++) {
            System.out.println("Sending Discount to Chosen Customer " + (i +
1));    }
    }

    public static void main(String[] args) {
        String[] customers = generateData();
        sendPromoDiscount(customers);
    }
}
```

Explanation:

- The code above aims to generate a string and send a promotion.
- In the generatedata function, the string 0821234567 will then be degenerated with an iteration for 100 times based on the iteration number.
- For iterations less than 0, a 0 will be added before the iteration value.
- After all of them have been generated, the values will be returned in the form of an array of strings.
- In the sendPromoDiscount function, it will print "sending promo to" for all customers, but there will be an iteration from i=0 to i<10 for the chosen customers.

Execution result :

```
Sending Promo to 082123456782
Sending Promo to 082123456783
Sending Promo to 082123456784
Sending Promo to 082123456785
Sending Promo to 082123456786
Sending Promo to 082123456787
Sending Promo to 082123456788
Sending Promo to 082123456789
Sending Promo to 082123456790
Sending Promo to 082123456791
Sending Promo to 082123456792
Sending Promo to 082123456793
Sending Promo to 082123456794
Sending Promo to 082123456795
Sending Promo to 082123456796
Sending Promo to 082123456797
Sending Promo to 082123456798
Sending Promo to 082123456799
Sending Discount to Chosen Customer 1
Sending Discount to Chosen Customer 2
Sending Discount to Chosen Customer 3
Sending Discount to Chosen Customer 4
Sending Discount to Chosen Customer 5
Sending Discount to Chosen Customer 6
Sending Discount to Chosen Customer 7
Sending Discount to Chosen Customer 8
Sending Discount to Chosen Customer 9
Sending Discount to Chosen Customer 10
```

L. EXPERIMENT 12

```java
import java.util.ArrayList;

class experiment12 {
    public static ArrayList<String> generateData(int n) {
        String baseNumber = "882";
        ArrayList<String> customers = new ArrayList<String>();
        String mobileNumber;

        for (int i = 0; i < n; i++) {
            mobileNumber = baseNumber + String.format("%09d", 1);
            customers.add(mobileNumber);
        }

        return customers;
    }

    public static void sendPromoDiscount(ArrayList<String> input) {
        for (int i = 0; i < input.size(); i++) {
            System.out.println("Sending Promo to " + input.get(i)); // O(n)
        }
        System.out.println("Its Finished to send Promo to " + input.size() + " Customers"); // O(1)

        for (int i = 0; i < input.size(); i++) {
            System.out.println("Sending Discount to " + input.get(i)); // O(n)
        }
        System.out.println("Its Finished to send Discount to " + input.size() + " Customers"); //
O(1)}

    public static void main(String[] args) {
        ArrayList<String> data = generateData(1000);
        sendPromoDiscount(data);
    }
}
```

Explanation :

- This program generates a list of phone numbers and sends a promo and discount to each phone number.
- The generateData method generates phone numbers with "082" as a prefix and adds them to an ArrayList.
- The sendPromoDiscount method sends a promo and discount to each phone number in the ArrayList.
- The main method calls generateData to create an ArrayList of 1000 phone numbers and then calls sendPromoDiscount to send a promo and discount to each phone number in the ArrayList.

Execution Result :

```
Sending Discount to 082000000984
Sending Discount to 082000000985
Sending Discount to 082000000986
Sending Discount to 082000000987
Sending Discount to 082000000988
Sending Discount to 082000000989
Sending Discount to 082000000990
Sending Discount to 082000000991
Sending Discount to 082000000992
Sending Discount to 082000000993
Sending Discount to 082000000994
Sending Discount to 082000000995
Sending Discount to 082000000996
Sending Discount to 082000000997
Sending Discount to 082000000998
Sending Discount to 082000000999
Its Finished to send Discount to 991 Customers
```

M. EXPERIMENT 13

```java
class experiment13 {
    public static String[] generateData(int n) {
        String baseNumber = "082";
        String[] customers = new String[n];
        String mobileNumber;
        for (int i = 0; i < n; i++) {
            mobileNumber = baseNumber + String.format("%09d", i + 1);
            customers[i] = mobileNumber;
        }
        return customers;
    }

    public static void sendPromoDiscount(String[] input1, String[] input2) {
        for (int i = 0; i < input1.length; i++) {
            System.out.println("Sending Promo to " + input1[i]); // O(m)
        }
        System.out.println("It's Finished to send Promo to " + input1.length + " Customers"); // O(1)
        for (int i = 0; i < input2.length; i++) {
            System.out.println("Sending Discount to " + input2[i]); // O(n)
        }
        System.out.println("It's Finished to send Discount to " + input2.length + " Customers"); //
O(1)}

    public static void main(String[] args) {
        String[] dataPromo = generateData(100000000);
        String[] dataDiscount = generateData(1000);
        sendPromoDiscount(dataPromo, dataDiscount);
    }
}
```

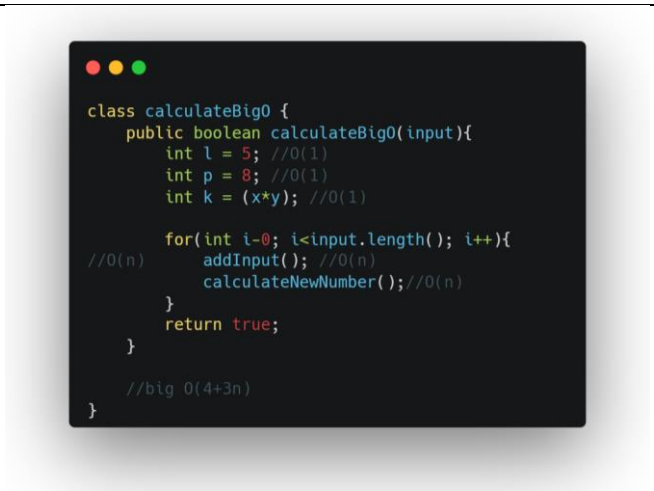
Explanation :

- generateData(n) - this method generates an array of "n" customer mobile numbers in the format of "082" followed by a 9-digit sequence starting from 1.
- sendPromoDiscount(input1, input2) - this method takes in two arrays of customer mobile numbers as input, and it sends a promotional message to all customers in the first input array and a discount message to all customers in the second input array.
- The main method calls the generateData method twice to create two arrays of customer mobile numbers with different lengths and passes them to the sendPromoDiscount method to send the messages.
- The time complexity of sending messages to customers is O(n) for both arrays.
- The result TLE because there are too many data.

Execution Result :

Time limit exceeded #stdin #stdout 5s 1002264KB

N. BIG O CALCULATION



```
class calculateBigO {
    public boolean calculateBigO(input){
        int l = 5; //O(1)
        int p = 8; //O(1)
        int k = (x*y); //O(1)

        for(int i=0; i<input.length(); i++){
//O(n)        addInput(); //O(n)
            calculateNewNumber();//O(n)
        }
        return true;
    }

    //big O(4+3n)
}
```

Explanation :

- Image above is an example for calculate big O

O. HASH TABLE IMPLEMENTATION

```
import java.util.ArrayList;

class HashTable {
    private ArrayList<String[]> data;

    public HashTable(int size) {
        this.data = new ArrayList<String[]>();
        for (int i = 0; i < size; i++) {
            this.data.add(null);
        }
    }

    private int hash(String key) {
        int hash = 0;
        for (int i = 0; i < key.length(); i++) {
            hash = (hash + key.charAt(i) * 1) % this.data.size();
        }
        return hash;
    }

    public void set(String key, String value) {
        int address = this.hash(key);
        if (this.data.get(address) == null) {
            this.data.set(address, new String[] { key, value });
        } else {
            this.data.get(address)[1] = value;
        }
    }

    public String get(String key) {
        int address = this.hash(key);
        String[] currentBucket = this.data.get(address);
        if (currentBucket != null) {
            for (int i = 0; i < currentBucket.length; i++) {
                if (currentBucket[i] != null && currentBucket[i].equals(key)) {
                    return currentBucket[i + 1];
                }
            }
        }
        return null;
    }

    public static void main(String[] args) {
        HashTable myHashTable = new HashTable(100);
        myHashTable.set("082124606606", "Rony Setyawan");
        System.out.println(myHashTable.get("082124606606"));
    }
}
```

Explanation :

- This Java code implements a basic hash table data structure using an ArrayList to store key-value pairs.
- The class has three main methods: hash which takes a key and returns a hash value, set which takes a key and a value, and stores the key-value pair in the appropriate index of the ArrayList, and get which takes a key and returns the corresponding value from the ArrayList based on the hash value of the key.
- The code also includes a main method that demonstrates the use of the hash table by creating an instance of the HashTable class, setting a key-value pair, and then retrieving the value using the key.

Execution Result :

```
Rony Setyawan
```

P. DYNAMIC PROGRAMMING IMPLEMENTATION

```java
import java.util.HashMap;
import java.util.Map;

class Fibonacci {
    private int counterPlainRecursive = 0;
    private int counterDynamicProgramming = 0;

    public int fibonacciPlainRecursive(int n) { // O(2^n)
        counterPlainRecursive++;
        if (n < 2) {
            return n;
        }
        return fibonacciPlainRecursive(n - 1) + fibonacciPlainRecursive(n - 2);
    }

    public int fibonacciDynamicProgramming(int n) { // O(n)
        Map<Integer, Integer> cache = new HashMap<>();
        return fib(n, cache);
    }

    private int fib(int n, Map<Integer, Integer> cache) {
        counterDynamicProgramming++;
        if (n < 2) {
            return n;
        } else if (cache.containsKey(n)) {
            return cache.get(n);
        } else {
            int result = fib(n - 1, cache) + fib(n - 2, cache);
            cache.put(n, result);
            return result;
        }
    }

    public static void main(String[] args) {
        Fibonacci fib = new Fibonacci();
        int n = 20;
        fib.fibonacciPlainRecursive(n);
        System.out.println("CounterPlainRecursive calculations for Plain Recursive: " +
fib.counterPlainRecursive);
        fib.fibonacciDynamicProgramming(n);
        System.out.println(
                "CounterDynamicProgramming calculations for Dynamic Programming: " +
fib.counterDynamicProgramming);
    }
}
```

Explanation :

- This Java program implements two methods for calculating the nth Fibonacci number: a plain recursive method and a dynamic programming method that uses a cache to store previously computed results.
- The plain recursive method has a time complexity of O(2^n) because it recalculates the same values multiple times, while the dynamic programming method has a time complexity of O(n) because it only calculates each value once and stores the result in a cache for later use.
- The program uses counters to track the number of calculations performed by each method for a given input value of n, and prints the results to the console.

Execution Result :

```
CounterPlainRecursive calculations for Plain Recursive: 21891
CounterDynamicProgramming calculations for Dynamic Programming: 39
```