Export to Golang

1. **Experiment1**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  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. |

Hasil eksekusi:

**Text

Description automatically generated**

1. **Experiment 2**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  Performance result = 0.78 ms | * Same as experiment 1, only for the loop in the main function, it iterates 1000 times. |

Hasil eksekusi:

**A picture containing graphical user interface

Description automatically generated**

1. **Experiment 3**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  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) |

Hasil eksekusi:

**Graphical user interface

Description automatically generated with low confidence**

1. **Experiment 4**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  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. |

Hasil eksekusi:

****

1. **Experiment 5**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  Performance result = 0.0005117 ms | * Same as experiment 4, only for the loop in the main function, it iterates 1000 times. |

Hasil eksekusi:

****

1. **Experiment 6**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  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. |

Hasil eksekusi:

****

1. **Experiment 7**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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). |

Hasil eksekusi:

**Text

Description automatically generated**

1. **Experiment 8**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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. |

Hasil eksekusi:

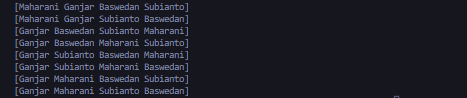
**Text

Description automatically generated**

1. **Experiment 9**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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!). |

Hasil eksekusi:

****

1. **Experiment 10**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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. |

Hasil eksekusi:

**Text

Description automatically generated**

1. **Experiment 11**

|  |  |
| --- | --- |
|  | * 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. |

Hasil eksekusi:

**Text

Description automatically generated**

1. **Experiment 12**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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** |

Hasil eksekusi:

**Text

Description automatically generated**

1. **Experiment 13**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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. |

Hasil eksekusi:

Karena data terlalu besar, saya tidak bisa mendapatkan hasil dari program menggunakan vs code. Jadi tidak ada screenshoot berhasil karena program berhenti di tengah-tengah karena warning data yang terlalu banyak untuk diproses.

1. **Big-O Calculation Example**

|  |  |
| --- | --- |
| **Text  Description automatically generated** | * 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. |

1. **Hash Table Implementation**

|  |  |
| --- | --- |
| **Text  Description automatically generated**  The result : | * 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. |

1. **Dynamic Programming**

|  |  |
| --- | --- |
| Text  Description automatically generated | * 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. |