If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view, and the command palette.

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

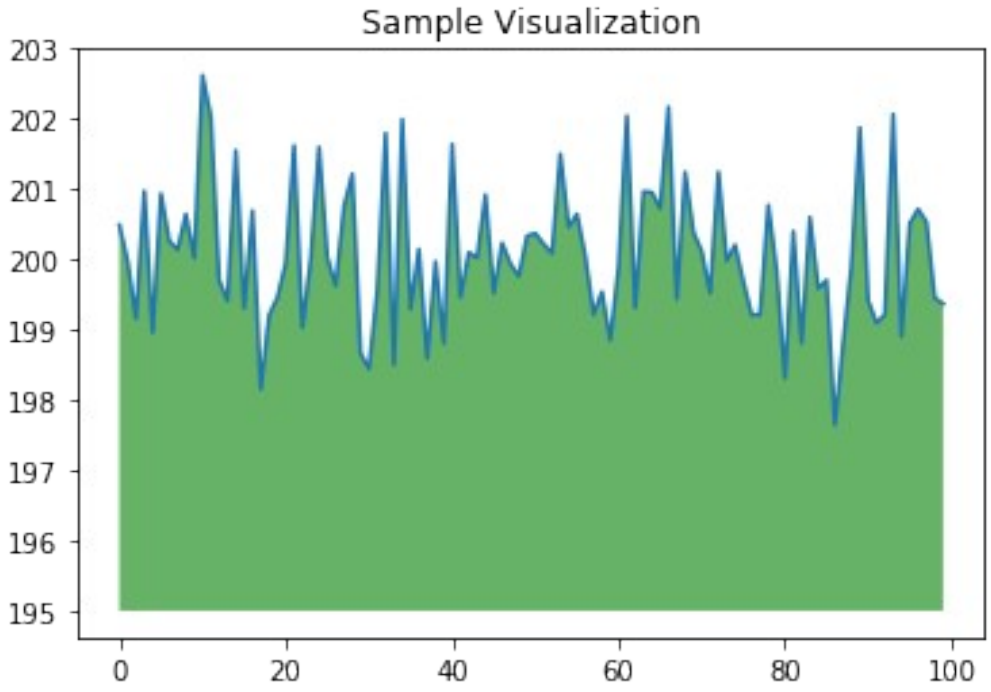
With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt
```

```
ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g',
alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

- Overview of Colaboratory
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets
- TensorFlow 2 in Colab
- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning Crash Course

These are a few of the notebooks from Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Linear regression with tf.keras using synthetic data
- TensorFlow with GPUs
- TensorFlow with TPUs
- NeMo Voice Swap: Use Nvidia's NeMo conversational AI Toolkit to swap a voice in an audio fragment with a computer generated one.
- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

DIA DICKSY ADITYA

210411100079

STRUKTUR DATA 2C

```
        TUGAS AKHIR PRAKTIKUM STRUKDAT 2C
```

Modul 1

Teori Stack

Stacks adalah satu struktur data dimana penambahan dan penghapusan data, hanya dapat dilakukan pada satu ujung yang sama, atau yang biasa dikenal dengan istilah top. Semakin data jauh berada dari posisi top, maka data tersebut diindikasikan berada di stack lebih lama dibandingkan dengan data yang berada dekat pada data di posisi top. Jika terdapat data baru yang ditambahkan di stack, maka data ini pulalah yang akan dihapus ketika terdapat proses penghapusan data. Konsep ini dikenal dengan nama LIFO-Last In First Out.

Algoritma Stack

- Membuat wadah untuk penampungan setiap item
- Menambahkan suatu item
- Menambahkan suatu item dibagian selanjutnya
- Penambahan item dilakukan secara bertumpuk
- Item yang masuk paling awal akan berada pada bagian bawah dan item yang terakhir masuk akan berada pada bagian atas

Code

```
def stack():
    return []
def push(s, data):
    s.append(data)
def pop(s):
    return s.pop()
def peek(s):
    return s[-1]
def isEmpty(s):
    return s == []
def size(s):
    return len(s)
```

Modul 2

#Teori Queue & Dequeue

Queue pada Struktur Data atau antrian adalah sekumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung disebut dengan sisibelakang(rear), dan penghapusan(pengambilan elemen) dilakukan lewat ujung lain (disebut dengan sisi depan

atau front). Queue atau antrian prinsip yang digunakan adalah “Masuk Pertama Keluar Pertama” atau FIFO (First In First Out).

Dequeue sama halnya dengan queue hanya saja terdapat beberapa fungsi yang di tambahkan seperti fungsi addRear, addFront, removeRear, removeFront dimana fungsi tersebut dapat digunakan untuk mengahapus atau menambahkan item pada bagian rear(belakang) atau front(depan) pada sebuah list

#Algoritma

- Membuat sebuah wadah untuk penampungan item
- Menambahkan sebuah item
- Item yang baru ditambahkan berada pada bagian paling akhir
- Item pertama yang ditambahkan akan keluar pertama kali juga
- Item terakhir yang ditamabahkan akan keluar paling akhir juga

#Code

```
### Queue ###
def createQueue():
    q = []
    return (q)
def enqueue(q, data):
    q.insert(0, data)
    return(q)
def dequeue(q):
    data = q.pop()
    return(data)
def isEmpty(q):
    return (q == [])
def size(q):
    return (len(q))

### Dequeue ###
def createDeque():
    return []
def addFront(deque, item):
    deque.insert(0, item)
    return deque
def addRear(deque, item):
    deque.append(item)
    return deque
def removeFront(deque):
    return deque.pop(0)
def removeRear(deque):
    return deque.pop()
def isEmpty(deque):
    return len(deque) == 0
def size(deque):
    return len(deque)
```

Modul 3

Teori Linked-List

Pemorgraman python yang telah menyediakan type data yang dinamis, Yaitu List. ukuran dan variabel yang bertipe data list dapat diatur sesuai dengan keinginan user selama program dijalankan, dan tidak harus mempunyai ukuran tetap diawal. Tipe data ini pun menyediakan metode tambahan seperti menambah data pada saat diperlukan, sehingga tipe data bersifat dinamis. Linked list merupakan struktur data yang bersifat dinamis

Algoritma Linked List

- Dalam penambahkan node baru terdapat dua tahapan:
  a. Pertama, menautkan node baru ke node awal dari linkedlist.
  b. kedua, melakukan modifikasi head dari linked list agar menunjuk pada node baru.
- Untuk mengetahui size dari linked list tersebut dapat menggunakan traversal linked list dengan menelusuri setiap node terdapat tiga tahapan:
  a. Pertama, membuat pointer bantuan current yang berada pada posisi node pertama atau *head*
  b. Kedua, Pointer current bergerak dengan perintah `current = current.getNext()` yang berfungsi sebagai increment dari variabel count yang digunakan untuk menghitung jumlah node.
  c. Ketiga, selama hasil dari `current = current.getNext()` bukan `none` pointer current akan terus berjalan menelusuri setiap node. `none` sendiri merepresentasikan bahwa setelah node tersebut tidak ada data lagi yang dikaitkan.

#Code

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def getData(self):
        return self.data
    def getNext(self):
        return self.next
    def setData(self, newData):
        self.data = newData
    def setNext(self, newNext):
        self.next = newNext

class linkedList:
    def __init__(self):
```

```
        self.head = None

    def isEmpty(self):
        return self.head == None

    def add(self, item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count += 1
            current = current.getNext()
        return count

    def display(self):
        current = self.head
        array = []
        while current != None:
            print(current.data)
            array.append(current.data)
            current = current.getNext()
        return array

    def search(self, item):
        current = self.head
        find = False
        while current != None:
            if current.data == item:
                find = True
                break
            else:
                current = current.getNext()
        return find

    def insertPrevious(self, item, newItem):
        current = self.head
        previous = None
        found = False
        if self.search(item):
            while not found:
                if current.data == item:
                    found = True
                else:
                    previous = current
                    current = current.getNext()
            if previous == None:
```

```
                self.add(newItem)
            else:
                temp = Node(newItem)
                temp.setNext(current)
                previous.setNext(temp)

    def insertNext(self, item, newItem):
        current = self.head
        previous = None
        found = False
        if self.search(item):
            while not found:
                if current.data == item:
                    found = True
                    previous = current
                    current = current.getNext()
                else:
                    current = current.getNext()
            temp = Node(newItem)
            if current == None:
                previous.setNext(temp)
            else:
                temp.setNext(current)
                previous.setNext(temp)

    def remove(self, item):
        current = self.head
        previous = None
        found = False
        if self.search(item):
            while not found:
                if current.getData() == item:
                    found = True
                else:
                    previous = current
                    current = current.getNext()
            if previous == None:
                self.head = current.getNext()
            else:
                previous.setNext(current.getNext())


myNode = linkedList()

myNode.add(20)
myNode.add(27)
myNode.add(18)
myNode.add(25)
myNode.remove(19)
myNode.display()
```

```
print("")
myNode.insertNext(19, 72)
myNode.insertPrevious(19, 90)
myNode.insertPrevious(24, 100)
myNode.remove(25)
myNode.display()
```

```
25
18
27
20

18
27
20

[18, 27, 20]
```

Modul 4

Teori Sorting

Proses menyusun kembali data yang sebelumnya telah disusun dengan suatu pola tertentu, sehingga tersusun secara teratur menurut aturan tertentu dengan tujuan agar data mudah untuk diolah. dan pengurutan data ini dilakukan dengan membandingkan setiap data dengan cara :

- Bubble Sort : mebutuhkan waktu yang lama untuk mengurutkan sebuah data dan algoritma ini adalah algoritma paling sederhana.
- Selection Sort : emiliki waktu lebih cepat daripada algoritma bubble sort karena pada selection sort terjadi reduksi waktu pada saat pertukaran data.
- Insertion Sort : mengasumsiskan bahwa data sudah terurut pada posisi awal, sehingga data berikutnya yang akan dibandingkan dan disisipkan pada bagian data yang sudah terurut tersebut.

Algoritma

- Bubble Sort :
  - Data dibaca mulai dari paling kiri atau mulai dari indeks 0 dari suatu list data
  - Bandingkan data antara 2 buah data yang berdekatan
  - Pindah salah satu posisi

- Lakukan perpindahan lagi seperti contoh sebelumnya dan pindah satu posisi kembali seperti langkah ke 3
- Lakukan secara terus menerus

- Selection Sort :
  - Mengasumsikan bahwa data ke-1 sampai dengan (2/n) sudah dalam keadaan tersebut
  - Menyisipkan data ke (n/2)+1 atau key ke dalam bagian data yang sudah dalam keadaan terurut dengan cara:
    - Lakukan perbandingan data antar key dengan data pada indeks sebelumnya, jika key bernilai lebih kecil, maka lakukan pergerseran posisi data.
    - Lakukan terus perbandingan data key tersebut dengan data pada indeks sebelumnya, sampai data key tidak lebih kecil lagi atau sampai dengan data pada posisi pertama.

Code

```
def ganjilGenapSort(data):
    print(f'Data= {data}')
    for x in range(len(data)):
        if x % 2 == 0:
            for j in range(0, len(data), 2):
                if data[j] > data[j+1]:
                    data[j], data[j+1] = data[j+1], data[j]
        else:
            for j in range(1, len(data)-1, 2):
                if data[j] > data[j+1]:
                    data[j], data[j+1] = data[j+1], data[j]
        print(f'Genap-Ganjil Sorting \n{data}')

    return data

def modifiedSelection(data):
    for x in range(len(data)//2):
        maxIndex = x
        minIndex = x
        iter = x+1

        print(f'Iterasi ke- {x+1}')
        for j in range(x, len(data)):
            if data[minIndex] > data[j]:
                minIndex = j

        data[x], data[minIndex] = data[minIndex], data[x]
        print(f'urut data minimal \t: {data}')
```

```
        for y in range(len(data)-iter, x, -1):
            if data[maxIndex] < data[y]:
                maxIndex = y

        data[len(data)-iter], data[maxIndex] = data[maxIndex],
data[len(data)-iter]
        print(f'urut data maksimal \t: {data}')

    return data

a = [13, 12, 10, 8, 7, 5, 11, 2]
b = [10, 2, 5, 8, 1, 20, 7, 12, 4]
print(ganjilGenapSort(a))
print(f'Data Urut= {modifiedSelection(b)}')
```

```
Data= [13, 12, 10, 8, 7, 5, 11, 2]
Genap-Ganjil Sorting
[12, 13, 8, 10, 5, 7, 2, 11]
Genap-Ganjil Sorting
[12, 8, 13, 5, 10, 2, 7, 11]
Genap-Ganjil Sorting
[8, 12, 5, 13, 2, 10, 7, 11]
Genap-Ganjil Sorting
[8, 5, 12, 2, 13, 7, 10, 11]
Genap-Ganjil Sorting
[5, 8, 2, 12, 7, 13, 10, 11]
Genap-Ganjil Sorting
[5, 2, 8, 7, 12, 10, 13, 11]
Genap-Ganjil Sorting
[2, 5, 7, 8, 10, 12, 11, 13]
Genap-Ganjil Sorting
[2, 5, 7, 8, 10, 11, 12, 13]
[2, 5, 7, 8, 10, 11, 12, 13]
Iterasi ke- 1
urut data minimal     : [1, 2, 5, 8, 10, 20, 7, 12, 4]
urut data maksimal    : [1, 2, 5, 8, 10, 4, 7, 12, 20]
Iterasi ke- 2
urut data minimal     : [1, 2, 5, 8, 10, 4, 7, 12, 20]
urut data maksimal    : [1, 2, 5, 8, 10, 4, 7, 12, 20]
Iterasi ke- 3
urut data minimal     : [1, 2, 4, 8, 10, 5, 7, 12, 20]
urut data maksimal    : [1, 2, 4, 8, 7, 5, 10, 12, 20]
Iterasi ke- 4
urut data minimal     : [1, 2, 4, 5, 7, 8, 10, 12, 20]
urut data maksimal    : [1, 2, 4, 5, 7, 8, 10, 12, 20]
Data Urut= [1, 2, 4, 5, 7, 8, 10, 12, 20]
```

Modul 5

Teori Recursive

Dikenal dengan perulangan adalah suatu materi didalam struktur data, yang dimana rekursif ini melibatkan beberapa pemanggilan nama fungsi dari rekursif itu sendiri, namun menggunakan parameter yang lebih kecil dari parameter awal.

Algoritma

Seperti contoh permasalahan bilangan faktorial. yang dimana dimisalkan kita ingin mencari bilangan faktorial. pastinya bilangan itu akan dikaitkan dengan bilangan itu -1, dan seterusnya sampai dikalikan dengan 1, dengan rumus n(n-1)...1

Code

```
def display(early, goals):
    if  early== 'A' and goals == 'B' :
        early = A
        goals = B
    elif early == 'A' and goals == 'C' :
        early = A
        goals = C
    elif early == 'B' and goals == 'A' :
        early = B
        goals = A
    elif early == 'B' and goals == 'C' :
        early = B
        goals = C
    elif early == 'C' and goals == 'A' :
        early = C
        goals = A
    elif early == 'C' and goals == 'B' :
        early = C
        goals = B
    goals.insert(0, early.pop(0))
    print('A: ')
    for i in A:
        print(f'|{i}|')
    print('B: ')
    for i in B:
        print(f'|{i}|')
    print('C: ')
    for i in C:
        print(f'|{i}|')
```

```python
def towers(n, early, help, goals):
    if n == 1:
        print(f'Lempengan - 1 dari {early} ke {goals}')
        display(early, goals)
    else:
        towers(n-1, early, goals, help)
        print(f'Lempengan - {n} dari {early} ke {goals}')
        display(early, goals)
        towers(n-1, help, early, goals)

A = [1, 2, 3, 4]
B = []
C = []
for i in A:
    print('|{i}|')
towers(4, "A", "B", "C")
```

```
|{i}|
|{i}|
|{i}|
|{i}|
Lempengan - 1 dari A ke B
A:
|2|
|3|
|4|
B:
|1|
C:
Lempengan - 2 dari A ke C
A:
|3|
|4|
B:
|1|
C:
|2|
Lempengan - 1 dari B ke C
A:
|3|
|4|
B:
C:
|1|
|2|
Lempengan - 3 dari A ke B
A:
|4|
B:
|3|
```

C:
|1|
|2|
Lempengan - 1 dari C ke A
A:
|1|
|4|
B:
|3|
C:
|2|
Lempengan - 2 dari C ke B
A:
|1|
|4|
B:
|2|
|3|
C:
Lempengan - 1 dari A ke B
A:
|4|
B:
|1|
|2|
|3|
C:
Lempengan - 4 dari A ke C
A:
B:
|1|
|2|
|3|
C:
|4|
Lempengan - 1 dari B ke C
A:
B:
|2|
|3|
C:
|1|
|4|
Lempengan - 2 dari B ke A
A:
|2|
B:
|3|
C:
|1|

|4|
Lempengan - 1 dari C ke A
A:
|1|
|2|
B:
|3|
C:
|4|
Lempengan - 3 dari B ke C
A:
|1|
|2|
B:
C:
|3|
|4|
Lempengan - 1 dari A ke B
A:
|2|
B:
|1|
C:
|3|
|4|
Lempengan - 2 dari A ke C
A:
B:
|1|
C:
|2|
|3|
|4|
Lempengan - 1 dari B ke C
A:
B:
C:
|1|
|2|
|3|
|4|

Modul 6

Teori Searching

Suatu metode untuk melakukan suatu pencarian agar dapat diketahui posisinya terhadap kumpulan data yang lainnya sehingga dapat digunakan untuk proses berikutnya

Algoritma

- Sequential Search : membandingkan data yang dicari dengan seluruh data yang terdapat pada kumpulan data secara satu persatu.
- Binary Search : Tidak dilakukan secara satu persatu namun memanfaatkan kelebihan pencarian di data yang sudah terurut. dilakukan pencarian mulai dari dari indeks list yg berada ditengah.

Code

Sequential Search

```
def sequentialSearch(data, item):
    indexFind = []
    x = 0
    y = 0
    while x < len(data):
        if data[x] == item:
            indexFind.append(x)
            x += 1
        else:
            x += 1
        y += 1
    if len(indexFind) == 0:
        return ["Data tidak ada", y]

    return [indexFind, y]

a = [1, 5, 9, 8, 1, 5, 10, 26, 5, 12]
[hasil, jumlahIterasi] = sequentialSearch(a, 0)
print(f'Posisi data= {hasil}' )
print(f'Jumlah Iterasi= {jumlahIterasi}')
```

```
Posisi data= Data tidak ada
Jumlah Iterasi= 10
```

Binary Search

```
def binarySearch(data, item):
    pertama = 0
    terakhir = len(data)-1
    # found = Salah
    listFound = []
    iter = 0

    while pertama <= terakhir:
        midpoint = (pertama + terakhir) // 2
        if data[midpoint] == item:
            # found = Benar
            listFound.append(midpoint)
            if item < data[midpoint+1]:
                terakhir = midpoint - 1
            else:
                pertama = midpoint + 1
        else:
            if item < data[midpoint]:
                terakhir = midpoint -1
            else:
                pertama = midpoint + 1
        iter += 1

    if len(listFound) == 0:
        return ["Data tidak ada", iter]

    return [listFound, iter]


a = [1, 1, 5, 5, 5, 8, 9, 10, 12, 26]
[hasil, iterasi]=binarySearch(a, 10)
# print(binSearch(a, 1))
print(f'Posisi data= {hasil}')
print(f'Jumlah Iterasi= {iterasi}')
```

```
Posisi data= [7]
Jumlah Iterasi= 4
```

Modul 7

Teori Hashing

Termasuk pencarian juga yang dimana hashing sendiri lebih cepat jika semua data pada list terletak tepat beradda di tempatnya masing- masing. sehingga hanya melakukan satu kali proses perbandingan saja.

Ada fungsi hashing, yaitu :

- remainderFunction() : untuk memoduluskan setiap data sesuai dengan jumlah hashTable yang diinginkan.
- createHashTable() : untuk pembuatan tabel hash table, yang awalnya hanya berisi list kosong atau bisa diisi dengan string "None"
- putData () : Untuk menyusun data ke dalam hash table, berdasarkan nilai hash yang dihasilkan.
- searchHash() : Untuk mencari data yang diinginkan, dan menghasilkan output/nilai balik berupa benar atau salah.

Algoritma

Setiap data dimoduluskan sesuai dengan jumlah kolom hash table yang diinginkan disetiap sisa bagi itu. maka itulah tempat bagi data tersebut ditempatkan, dan apabila terdapat modulus sama akan dilakukan chaining

Code

```
def remainderFunction(a , b):
    return a % b

def createHashTable(a = 11):
    return [[None] for i in range(a)]

def chainning(data, hashTable):
    for i in data:
        index = remainderFunction(i, len(hashTable))
        if hashTable[index] == [None]:
            hashTable[index] = [i]
        else:
            hashTable[index].append(i)

def searchHash(a, hashTable):
    findIndex = remainderFunction(a, len(hashTable))
    for i in range(len(hashTable[findIndex])):
        if a == hashTable[findIndex][i]:
            print(f'Data berada di slot ke- {findIndex}, dan indeks
ke-{i}')
            return

    print(False)


a = [54, 26, 93, 17, 77, 31, 44, 55, 20]
```

```
makeHashTable = createHashTable()
chainning(a, makeHashTable)
print(makeHashTable)
searchHash(66, makeHashTable)
searchHash(54, makeHashTable)
searchHash(20, makeHashTable)
searchHash(55, makeHashTable)
searchHash(100, makeHashTable)
```

```
[[77, 44, 55], [None], [None], [None], [26], [93], [17], [None],
[None], [31, 20], [54]]
False
Data berada di slot ke- 10, dan indeks ke-0
Data berada di slot ke- 9, dan indeks ke-1
Data berada di slot ke- 0, dan indeks ke-2
False
```

Modul 8

Teori Merge Sort dan Quick Sort

Merge Sort adalah pegurutan dengan cara membagi data menjadi 2 kumpulan data dari masing-masing kumpulan yang telah diurutkan namun kemudian setelah terurut dilakukan penggabungan dua kumpulan data tersebut dalam keadaan terurut

Quick Sort adalah metode partisi yang dimana mempunyai efektifitas tinggi dengan teknik menukarkan 2 elemen dengan jarak yang cukup besar.

Algoritma

Hampir sama seperti pencarian pada pengurutan banyak data sehingga kita bisa membagi dua dan membandingkannya antar data.

Code

```
iter  = 0
banding = 0
geser = 0
def partitionD(array, start, end):
    global iter,banding,geser
    pivot = array[start]
    left = start + 1
    right = end
    while True:
        while left <= right and array[left] >= pivot:
            left = left + 1
```

```
            banding +=1
        while left <= right and array[right] <= pivot:
            right = right - 1
            banding +=1
        if right<left:
            break
        else:
            array[left], array[right] = array[right], array[left]
            geser +=1
    array[start], array[right] = array[right], array[start]
    geser +=1
    iter +=1
    print(f'Iterasi ke - {iter}\n{array} Pivot -> {pivot}')
    return right

def partitionA(array, start, end):
    global iter,banding,geser
    pivot = array[start]
    low = start + 1
    high = end

    while True:
        while low <= high and array[high] >= pivot:
            high = high - 1
            banding +=1
        while low <= high and array[low] <= pivot:
            low = low + 1
            banding +=1
        if low <= high:
            array[low], array[high] = array[high], array[low]
            geser +=1
        else:
            break
    array[start], array[high] = array[high], array[start]
    geser +=1
    iter +=1
    print(f'Iterasi ke - {iter}\n{array} Pivot -> {pivot}')
    return high

def quick_sortA(array, start, end):
    if start < end:
        p = partitionA(array, start, end)
        quick_sortA(array, start, p-1)
        quick_sortA(array, p+1, end)
    return banding,geser
def quick_sortD(array, start, end):
    if start < end:
        p = partitionD(array, start, end)
        quick_sortD(array, start, p-1)
        quick_sortD(array, p+1, end)
```

```
    return banding,geser

# main
arr = [56,26,93,17,31,44]
n = len(arr)
[banding,geser]=quick_sortA(arr,0,n-1)
print(f"\nHasil Akhir = {arr}")
print(f"Total Perbandingan : {banding}")
print(f"Total Pergeseran : {geser}")

iterasi,banding,geser = 0,0,0
def merge_sortA(a_list):
    global iterasi,banding,geser
    if len(a_list) > 1:
        mid = len(a_list) // 2
        left_half = a_list[:mid]
        right_half = a_list[mid:]
        merge_sortA(left_half)
        merge_sortA(right_half)
        i,j,k = 0,0,0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                a_list[k] = left_half[i]
                i += 1
                geser +=1
            else:
                a_list[k] = right_half[j]
                j += 1
            k += 1
            banding +=1
        while i < len(left_half):
            a_list[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            a_list[k] = right_half[j]
            j += 1
            k += 1
    if len(a_list) >1:
        iterasi +=1
        print(f"iterasi ke - {iterasi}")
        print("Merging ", a_list)
    return geser,banding

def merge_sortD(a_list):
    global iterasi,banding,geser
    if len(a_list) > 1:
        mid = len(a_list) // 2
        left_half = a_list[:mid]
        right_half = a_list[mid:]
```

```
        merge_sortD(left_half)
        merge_sortD(right_half)
        i,j,k = 0,0,0
        while i < len(left_half) and j < len(right_half):
            if left_half[i] > right_half[j]:
                a_list[k] = left_half[i]
                i += 1
                geser +=1
            else:
                a_list[k] = right_half[j]
                j += 1
            k += 1
            banding +=1
        while i < len(left_half):
            a_list[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            a_list[k] = right_half[j]
            j += 1
            k += 1
    if len(a_list) >1:
        iterasi +=1
        print(f"iterasi ke - {iterasi}")
        print("Merging ", a_list)
    return geser,banding


array = [56,26,93,17,31,44]
while True:
    inpt = int(input("Pilih Metode Pengurutan:\n1.Merge Sort\n2.Quick
Sort\nPilih: "))
    if inpt==1:
        inpt2 =int(input("Pengurutan Secara:\n1. Ascending\n2.
Descending\nPilih: "))
        if inpt2 == 1:
            [total_geser,total_banding] = merge_sortA(array)
            print(f'\nBanyaknya Perbandingan - {total_banding}')
            print(f'Banyaknya Pergeseran - {total_geser}')
        elif inpt2 ==2:
            [total_geser,total_banding] = merge_sortD(array)
            print(f'\nBanyaknya Perbandingan - {total_banding}')
            print(f'Banyaknya Pergeseran - {total_geser}')
        else:
            print("Maaf Inputan Tidak Valid!")
    elif inpt ==2:
        inpt2 =int(input("Pengurutan Secara:\n1. Ascending\n2.
Descending\nPilih: "))
        if inpt2 ==1:
            n = len(array)
```

```
            [banding,geser]=quick_sortA(array,0,n-1)
            print(f"\nHasil Akhir = {array}")
            print(f"Total Perbandingan : {banding}")
            print(f"Total Pergeseran : {geser}")
        elif inpt2 == 2:
            n = len(array)
            [banding,geser]=quick_sortD(array,0,n-1)
            print(f"\nHasil Akhir = {array}")
            print(f"Total Perbandingan : {banding}")
            print(f"Total Pergeseran : {geser}")
        else:
            print("Maaf Inputan Tidak Valid!")
    else:
        print("Maaf Inputan Tidak Valid!")

    akhir = int(input("\nPilih\n1.Lagi\n2.Keluar\nPilih: "))
    if akhir ==1:
        pass
    else:
        break
```

```
Iterasi ke - 1
[31, 26, 44, 17, 56, 93] Pivot -> 56
Iterasi ke - 2
[17, 26, 31, 44, 56, 93] Pivot -> 31
Iterasi ke - 3
[17, 26, 31, 44, 56, 93] Pivot -> 17

Hasil Akhir = [17, 26, 31, 44, 56, 93]
Total Perbandingan : 9
Total Pergeseran : 5
Pilih Metode Pengurutan:
1.Merge Sort
2.Quick Sort
Pilih: 1
Pengurutan Secara:
1. Ascending
2. Descending
Pilih: 1
iterasi ke - 1
Merging  [26, 93]
iterasi ke - 2
Merging  [26, 56, 93]
iterasi ke - 3
Merging  [31, 44]
iterasi ke - 4
Merging  [17, 31, 44]
iterasi ke - 5
Merging  [17, 26, 31, 44, 56, 93]
```

```
Banyaknya Perbandingan - 9
Banyaknya Pergeseran - 5

Pilih
1.Lagi
2.Keluar
Pilih: 2
```