Furkan Cemal Çalışkan

1810206046

1. **Implement python code for below data list with 5 famous Sort algorithms:**

*friends = [*

*{'name': 'Rachel' , 'year': '1969'} ,*

*{'name': 'Ross' , 'year': '1966'} ,*

*{'name': 'joey' , 'year': '1967'} ,*

*{'name': 'Monica' , 'year': '1964'} ]*

**Answer:**

#Furkan Cemal Çalışkan

#1810206046

import numpy as np

from random import randint

friends = [

{'name': 'Rachel' , 'year': '1969'} ,

{'name': 'Ross' , 'year': '1966'} ,

{'name': 'joey' , 'year': '1967'} ,

{'name': 'Monica' , 'year': '1964'} ]

friendsArray = []

for i in range(4):

result = friends[i].items()

data = list(result)

friendsArray.append(data)

#Selection sort

def selectionSort(array):

length = len(array)

for i in range(length-1):

minIndex = i

for j in range(i+1, length):

if array[j]<array[minIndex]:

minIndex = j

array[i], array[minIndex] = array[minIndex], array[i]

return array

#Bubble sort

def bubbleSort(array):

n = len(array)

for i in range(n):

# Create a flag that will allow the function to

# terminate early if there's nothing left to sort

already\_sorted = True

# Start looking at each item of the list one by one,

# comparing it with its adjacent value. With each

# iteration, the portion of the array that you look at

# shrinks because the remaining items have already been

# sorted.

for j in range(n - i - 1):

if array[j] > array[j + 1]:

# If the item you're looking at is greater than its

# adjacent value, then swap them

array[j], array[j + 1] = array[j + 1], array[j]

# Since you had to swap two elements,

# set the `already\_sorted` flag to `False` so the

# algorithm doesn't finish prematurely

already\_sorted = False

# If there were no swaps during the last iteration,

# the array is already sorted, and you can terminate

if already\_sorted:

break

return array

#Insertion sort

def insertionSort(array):

# Loop from the second element of the array until

# the last element

for i in range(1, len(array)):

# This is the element we want to position in its

# correct place

key\_item = array[i]

# Initialize the variable that will be used to

# find the correct position of the element referenced

# by `key\_item`

j = i - 1

# Run through the list of items (the left

# portion of the array) and find the correct position

# of the element referenced by `key\_item`. Do this only

# if `key\_item` is smaller than its adjacent values.

while j >= 0 and array[j] > key\_item:

# Shift the value one position to the left

# and reposition j to point to the next element

# (from right to left)

array[j + 1] = array[j]

j -= 1

# When you finish shifting the elements, you can position

# `key\_item` in its correct location

array[j + 1] = key\_item

return array

#Merge sort

def mergeSort(array):

if len(array) > 1:

mid = len(array) // 2

left = array[:mid]

right = array[mid:]

# Recursive call on each half

mergeSort(left)

mergeSort(right)

# Two iterators for traversing the two halves

i = 0

j = 0

# Iterator for the main list

k = 0

while i < len(left) and j < len(right):

if left[i] <= right[j]:

# The value from the left half has been used

array[k] = left[i]

# Move the iterator forward

i += 1

else:

array[k] = right[j]

j += 1

# Move to the next slot

k += 1

# For all the remaining values

while i < len(left):

array[k] = left[i]

i += 1

k += 1

while j < len(right):

array[k]=right[j]

j += 1

k += 1

return array

#Quicksort

def quickSort(array):

# If the input array contains fewer than two elements,

# then return it as the result of the function

if len(array) < 2:

return array

low, same, high = [], [], []

# Select your `pivot` element randomly

pivot = array[randint(0, len(array) - 1)]

for item in array:

# Elements that are smaller than the `pivot` go to

# the `low` list. Elements that are larger than

# `pivot` go to the `high` list. Elements that are

# equal to `pivot` go to the `same` list.

if item < pivot:

low.append(item)

elif item == pivot:

same.append(item)

elif item > pivot:

high.append(item)

# The final result combines the sorted `low` list

# with the `same` list and the sorted `high` list

return quickSort(low) + same + quickSort(high)

select = selectionSort(friendsArray)

bubble = bubbleSort(friendsArray)

insert = insertionSort(friendsArray)

merge = mergeSort(friendsArray)

quick = quickSort(friendsArray)

print(f"SelectionSort = {select}\n")

print(f"BubbleSort = {bubble}\n")

print(f"InsertionSort = {insert}\n")

print(f"MergeSort = {merge}\n")

print(f"QuickSort = {quick}\n")

print("Furkan Cemal Caliskan")

print("1810206046")

**Output:**

**Text, letter

Description automatically generated**

1. **Select your team leader name and project subject.**

I chose Semih Şahin, the subject of Human Emotion Recognition, as the team leader and added my name into excel file.