QUICK SORT DOCUMENTATION

Description for QuickSort using C programming:

This C program is designed to manage and sort a list of customer orders. It utilizes a custom data structure called CustomerOrder to represent each order, storing information such as order ID, customer name, order date, and total amount. The program incorporates dynamic memory allocation and the QuickSort algorithm to efficiently sort the list of orders based on their order dates.

1.Data Structure (CustomerOrder):

The CustomerOrder structure encapsulates the essential details of a customer order.Fields include orderID (identification number), customerName, orderDate (using the struct tm structure for date representation), and totalAmount (the cost of the order).

2. Functions:

inputDate: A function to prompt the user for input in the "YYYY-MM-DD" format and store the date in the struct tm format.

partition: Partitions the array for the QuickSort algorithm. It dynamically allocates memory for temporary variables and effectively reorganizes the array.

quickSort: Implements the QuickSort algorithm to sort an array of CustomerOrder structures. It uses a comparison function (compareOrderByDate) to determine the order of elements based on their order dates.

compareOrderByDate: A comparison function used by quickSort to compare two CustomerOrder structures based on their order dates. It converts dates to time_t values and uses difftime for the comparison.

3. Main Function:

Prompts the user to input the number of customer orders (n).

Dynamically allocates memory for an array of CustomerOrder structures based on the user input.

Collects details for each customer order, including order ID, customer name, order date, and total amount.

Prints the original list of customer orders, displaying order details such as ID, customer name, date, and amount.

Calls the quickSort function to sort the array of orders based on the order date.

Prints the sorted list of customer orders.

Frees the dynamically allocated memory to prevent memory leaks.

Note:

The program assumes that the user inputs the date in the correct format ("YYYY-MM-DD").

The comparison function involves converting the order dates to time_t values, allowing for accurate comparisons using difftime.

Description for QuickSort using Python:

This Python script manages customer orders through a command-line interface. It includes a CustomerOrder class representing individual orders and utilizes a quicksort algorithm to sort the orders based on their order dates. The main features of the script include:

1. CustomerOrder Class:

The CustomerOrder class is designed to store information about a customer order.

Attributes include order_id (unique identifier), customer_name, order_date (date of the order), and total_amount (the cost of the order).

2. Date Input Function:

The script includes a function named input_date() to get date input from the user.

It prompts the user to enter a date in the format "YYYY-MM-DD" and uses datetime.strptime to parse the input into a datetime object.

3. Quicksort Implementation:

The script contains a custom quicksort implementation (quicksort) to sort an array of customer orders.

The sorting is based on a user-defined comparison function (compare_order_by_date), which compares customer orders by their order dates.

4. User Interaction:

The main() function serves as the main entry point for the script.

It prompts the user to input the number of customer orders and details for each order, including order ID, customer name, order date, and total amount.

5. Displaying Orders:

The script displays the original list of customer orders and then the sorted list based on order dates.

The information includes order ID, customer name, order date (formatted as "YYYY-MM-DD"), and total amount.

6. Error Handling:

Some basic error handling is implemented, such as handling invalid date formats and catching value errors during input.

7. Main Execution:

The script checks if it is being run as the main program (__name__ == "__main__") and calls the main() function accordingly.

Compilers used:

c program:

I use Visual Studio Code to run c program.

To run the program, type the following in the terminal: ./your_program_name

Python:

I used Jupyter Notebook to run python code.

Jupyter Notebook doesn't use a traditional compiler or interpreter in the same way that standalone Python scripts might. Instead, it relies on a concept called the Jupyter kernel.

Kernel: The Jupyter Notebook runs a kernel, which is a separate process responsible for executing the code.

When you run a cell, the code in that cell is sent to the kernel, which executes it and sends the results back to the notebook.

Difference between C programming and Python in Quick Sort:

In C (Procedural Paradigm):

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

// Struct representing a customer order

typedef struct {

   int orderID;

   char customerName[50];

   struct tm orderDate;  // Using struct tm for date (from time.h)

   double totalAmount;

   // Add other fields as needed for your application
```

```c
} CustomerOrder;


// Function to parse date input and store it in struct tm
void inputDate(struct tm *date) {
    printf("Enter date in the format YYYY-MM-DD: ");
    scanf("%d-%d-%d", &date->tm_year, &date->tm_mon, &date->tm_mday);
    // Adjustments for struct tm
    date->tm_year -= 1900;  // Years since 1900
    date->tm_mon -= 1;      // Months (0-11)
}
// Function to perform QuickSort
void quickSort(CustomerOrder *arr, int low, int high) {
    // ...
}
// Example comparison function for customer orders by order date
int compareOrderByDate(const void *a, const void *b) {
    time_t timeA = mktime(&(((CustomerOrder *)a)->orderDate));
    time_t timeB = mktime(&(((CustomerOrder *)b)->orderDate));
    return difftime(timeA, timeB);
}
int main() {
    // ...
}
```

In Python (Object-Oriented Paradigm):

```python
from datetime import datetime
# Class representing a customer order
class CustomerOrder:
    def __init__(self, order_id, customer_name, order_date, total_amount):
        self.order_id = order_id
        self.customer_name = customer_name
        self.order_date = order_date
        self.total_amount = total_amount
        # Add other fields as needed for your application
    @staticmethod
    def input_date():
        date_str = input("Enter date in the format YYYY-MM-DD: ")
        return datetime.strptime(date_str, "%Y-%m-%d")
    @staticmethod
    def compare_order_by_date(order_a, order_b):
        time_a = order_a.order_date.timestamp()
        time_b = order_b.order_date.timestamp()
        return int(time_a - time_b)
    @classmethod
    def quicksort(cls, arr):
        # ...
        # Use cls.compare_order_by_date instead of compare_order_by_date
        # ...
def main():
    n = int(input("Enter the number of customer orders: "))
    orders = []
    print("Enter details for each customer order:")
    for i in range(n):
        # ...
        order_date = CustomerOrder.input_date()
        # ...
        order = CustomerOrder(order_id, customer_name, order_date, total_amount)
        orders.append(order)
    # ...
    orders_sorted = CustomerOrder.quicksort(orders)
    # ...
```

Key Differences:

Class and Object Concept:

In Python, we encapsulate the data and related functions within a class (CustomerOrder), promoting the concept of objects.

In C, we use structs to group related data, but the struct itself doesn't contain methods (functions) directly.

Static and Class Methods:

In Python, we use the @staticmethod and @classmethod decorators to define static and class methods, respectively.

C doesn't have the concept of static or class methods. Functions are defined globally or within structs.

Object-Oriented Approach:

Python leverages an object-oriented paradigm, where data and functions are bundled together within classes and objects.

C, while supporting structs, focuses more on procedural programming, where functions operate on data structures but aren't inherently tied to them.