QUESTION FORMAT: (ADD PLAG DECLAR & STAT OF COMPL. NO IMAGES OF CODE.)

**Question:**

**Source Code:**

**Sample Screen Dumps:**

**How this was tested:**

**Question: 1**

**Source Code:**

import random

arrA = []

sizeA = 300

arrB = []

sizeB = 400

for size in range(sizeA):

    arrA.append(random.randint(0,1024))

for size in range(sizeB):

    arrB.append(random.randint(0,1024))

temp = 0

print(f"Unsorted Array A: {arrA}")

print(f"Unsorted Array B: {arrB}")

# Shell Sort

gap = sizeA // 2

flag = True

while gap >= 1 and flag == True:

    flag = False

    for i in range(sizeA - gap):

        if arrA[i] > arrA[i + gap]:

            temp = arrA[i]

            arrA[i] = arrA[i + gap]

            arrA[i + gap] = temp

            flag = True

    if gap > 1:

        gap = gap // 2

# Quick Sort

def qSort(arr):

    quick(arr, 0, len(arr) - 1)

def quick(arr, first, last):

    if first < last:

        pivotPos = partition(arr, first, last)

        quick(arr, first, pivotPos - 1)

        quick(arr, pivotPos + 1, last)

def partition(arr, first, last):

    pivot = arr[first]

    u = first

    d = last

    while True:

        while u < last and arr[u] <= pivot:

            u += 1

        while arr[d] > pivot:

            d -= 1

        if u < d:

            temp = arr[u]

            arr[u] = arr[d]

            arr[d] = temp

        else:

            break

    temp = arr[first]

    arr[first] = arr[d]

    arr[d] = temp

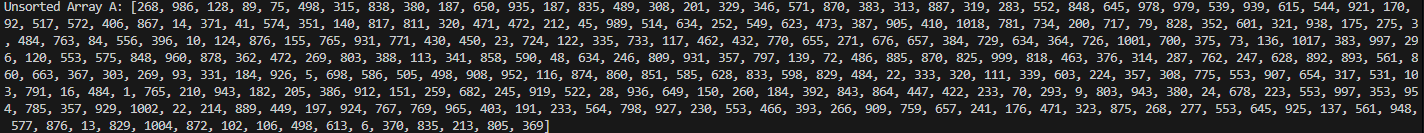
    return d

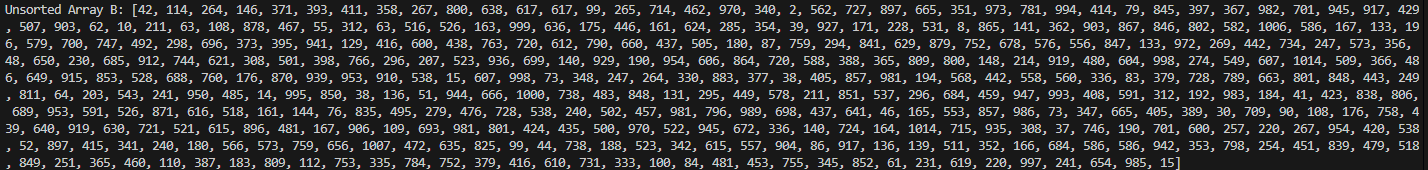
qSort(arrB)

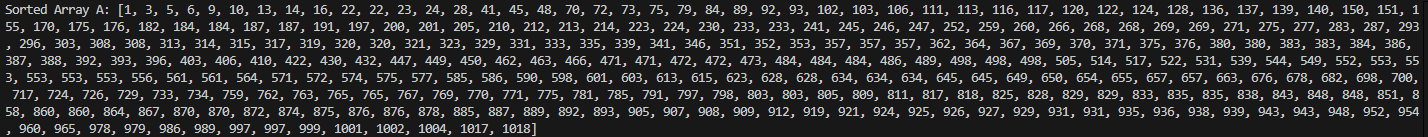
print(f"Sorted Array A: {arrA}")

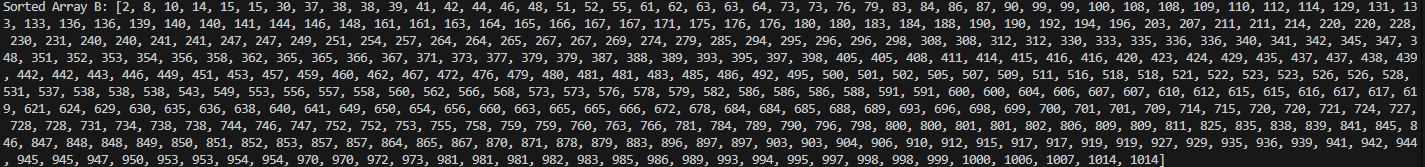
print(f"Sorted Array B: {arrB}")

**Sample Screen Dumps:**









**How this was tested:** The sizes of both arrays were initially set to smaller values to check the functionality of the code. Arrays of various sizes, including both odd and even, were tested to confirm that the program handled them properly. Additionally, both sorting algorithms were provided with an already sorted array to verify their correctness.

**Question:** 2

**Source Code (to be pasted after code in Question 1):**

arrC = []

ptr1 = 0

ptr2 = 0

while ptr1 < sizeA and ptr2 < sizeB:

    if arrA[ptr1] > arrB[ptr2]:

        arrC.append(arrB[ptr2])

        ptr2 += 1

    elif arrB[ptr2] > arrA[ptr1]:

        arrC.append(arrA[ptr1])

        ptr1 += 1

    else:

        arrC.append(arrA[ptr1])

        arrC.append(arrB[ptr2])

        ptr1 += 1

        ptr2 += 1

while ptr1 < sizeA:

    arrC.append(arrA[ptr1])

    ptr1 += 1

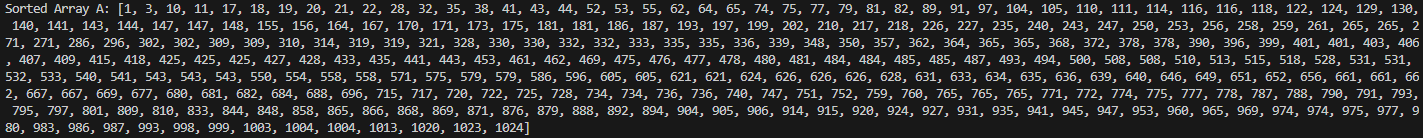
while ptr2 < sizeB:

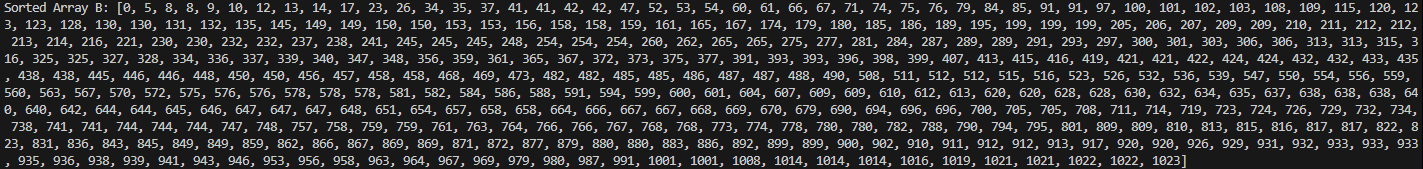
    arrC.append(arrB[ptr2])

    ptr2 += 1

print(f"Merged Array: {arrC}")

**Sample Screen Dumps:**

****

****

**A black screen with many small squares

AI-generated content may be incorrect.**

**How this was tested:** The sizes of both arrays were initially set to smaller values to check the functionality of the code. The lengths of arrays arrA, arrB and arrC were printed and as expected, the length of arrC was the combined lengths of arrays arrA and arrB.

**Question:** 3

**Source Code:**

import random

arr = []

eArr = [] # holds the extreme points

for i in range(10):

    arr.append(random.randint(0,20))

print(f"Unchecked array: {arr}")

def extremeCheck(arr):

    global epts

    epts = False

    for i in range(1, len(arr) - 1):

        if (arr[i] < arr[i-1] and arr[i] < arr[i+1]) or (arr[i] > arr[i-1] and arr[i] > arr[i+1]):

            eArr.append(arr[i])

            print(f"Value at index {i}: {arr[i]} is an extreme point")

            epts = True

extremeCheck(arr)

if epts:

    print(f"Line print of extreme points: {eArr}")

else:

    print("SORTED")

**Sample Screen Dumps:**

**A screenshot of a computer

AI-generated content may be incorrect.**

****

**How this was tested:** To test the extremeCheck function with an unsorted array, the code was simply run. As it’s highly unlikely for the random generated array to be sorted, a temporary sorted array was made and was passed to extremeCheck function.

**Do you agree that an array has no extreme points if and only if it is sorted? Explain your answer**

It is true that array has no extreme points if and only if it is sorted. If the array is sorted in ascending order, all elements (excluding first and last) will be larger than the previous element but smaller than the next element. Similarly, if the array is sorted in descending order, all elements (excluding first and last) will be smaller than the previous element but larger than the next element. Such conditions A[i-1] < A[i] < A[i+1] or A[i-1] > A[i] > A[i+1] does not comply with the stated conditions A[i− 1] < A[i] > A[i + 1] or A[i − 1] > A[i] < A[i + 1]

**Question:** 4

**Source Code:**

import random

arr = []

N = 50

for i in range(N):

    arr.append(random.randint(1,1024))

combinations = {}

# Storing ALL UNIQUE COMBINATIONS as dictionary with array values

for i in range(len(arr)):

    for j in range(i+1, len(arr)):

        a = arr[i]

        b = arr[j]

        product = a\*b

        if product not in combinations.keys():

            combinations[product] = []

        if a>b: # Storing as a sorted tuple to avoid duplicates, easier to work with also.

            a,b = b,a

        if (a,b) not in combinations[product]:

            combinations[product].append((a,b))

found = False

for key in combinations.keys():

    ptr = combinations.get(key)

    if len(ptr) == 2:

        found = True

        print(f"Product {key} has combinations: {ptr}")

    elif len(ptr) > 2:

        found = True

        print(f"Product {key} has the following combinations")

        for i in range(len(ptr)):

            for j in range(i+1, len(ptr)):

                a = ptr[i]

                b = ptr[j]

                print(f"{a} and {b}")

if not found:

    print("There are no 2-pairs of integers having the same product for this run. Try increasing the value of N or re-run the code!")

""" rows = [f"Key {k} has value {v}" for k, v in combinations.items()]

max\_len = max(len(row) for row in rows)

count = 0

print("FULL DICTIONARY:")

for k,v in combinations.items():

    text = f"Key {k} has value {v}"

    print(f"{text:<{max\_len}}", end="\t")

    count += 1

    if (count == 5):

        count = 0

        print() """

**Sample Screen Dumps:**

****

**A screen shot of a computer program

AI-generated content may be incorrect.**

**How this was tested:** Different values of N were fed into the random generator so as to increase the likelihood of having more 2-pairs of integers.

As an extra, I felt it would be good practice to have the code print the full dictionary at the end but this is commented out by default to not spam the output.

In order to present the dictionary in evenly spaced columns, I consulted ChatGPT with the following prompt:

**How can I not use a fixed width to determine the maximum length needed for this code: (insert code). I don’t want to have very big spaces between columns**

This resulted in calculating the max length of the longest row so as to then feed this to the spacing system in the print statement.

**Question:** 5

**Source Code:**

class Stack:

    def \_\_init\_\_(self):

        self.items = []

    def \_\_str\_\_(self):

        return f"Current stack contents: {self.items}"

    def push(self,item):

        self.items.append(item)

    def pop(self):

        if self.is\_empty():

            print("Cannot pop an empty stack. Program will now terminate.")

            exit(1)

        else:

            return self.items.pop()

    def peek(self):

        if self.is\_empty():

            print("There are no elements in the stack")

        else:

            return self.items[-1]

    def is\_empty(self):

        if self.size() == 0:

            return True

    def size(self):

        return len(self.items)

def eval(expression):

    contents = expression.split()

    if not contents:

        print("You did not input anything. Program will now terminate. ")

        exit(1)

    stack = Stack()

    operators = ['+', '-', 'x', '/']

    for i in contents:

        print(f"Evaluating {i}")

        try:

            i = float(i)

            stack.push(i)

        except ValueError:

            if i not in operators:

                print(f"Operator {i} is invalid. Program will now terminate")

                exit(1)

            elif i in operators:

                if stack.size() < 2:

                    print(f"Not enough numbers in the stack for operator {i}. Program will now terminate.")

                    exit(1)

                else:

                    b = stack.pop()

                    a = stack.pop()

                    if i == '+':

                        stack.push(a + b)

                    elif i == '-':

                        stack.push(a - b)

                    elif i == 'x':

                        stack.push(a \* b)

                    elif i == '/':

                        if b == 0:

                            print("Cannot divide by zero. Program will now terminate.")

                            exit(1)

                        stack.push(a / b)

        print(stack)

    if stack.size() != 1:

            print("There are still additional contents in the stack and not enough operators. Invalid RPN entry. Program will now terminate")

            exit(1)

    else:

        print(f"The answer is {stack.peek()}")

entry = input("Enter a valid RPN expression, separated by a space to denote the next item in the stack \n")

eval(entry)

**Sample Screen Dumps:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A black background with white text

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**How this was tested:** Several test cases were fed as input expressions to test all the error messages; valid expressions, empty input, invalid operators, insufficient numbers for an operator, dividing by zero and stack not being of size 1 at end of evaluation. The Stack class includes two unused error messages which were kept to preserve its core functionality and structure.

**Question:** 6

**Source Code:**

def isPrime(num):

    if num <= 1:

        return False

    if num == 2:

        return True

    for i in range(2, num):

        if num % i == 0:

            return False

        else:

            return True

def sieve(num):

    prime = [True for i in range(num+1)]

    prime[0] = prime[1] = False

    for i in range(2, num + 1):

        if prime[i]:

            for j in range(i \* 2, num + 1, i):

                prime[j] = False

    for i in range(2, num+1):

        if prime[i]:

            print(i, end= " ")

try:

    num = int(input("Enter a whole number to check if it's prime \n"))

    print(isPrime(num))

except ValueError:

    print("You did not input a whole number. Program will now terminate.")

    exit(1)

try:

    num = int(input("Enter a whole number to perform Sieve of Eratosthenes on. \n"))

    sieve(num)

except ValueError:

    print("You did not input a whole number. Program will now terminate.")

    exit(1)

**Sample Screen Dumps:**

**A black screen with many small squares

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**How this was tested:** Different numbers were fed as input to both functions to test their functionality including invalid numbers to test the functionality of the try except blocks.

**Question:** 7

**Source Code:**

import csv

def collatzSeq(num):

    seq = [num]

    while num != 1:

        if num % 2 == 0:

            num = num // 2

        else:

            num = (3 \* num) + 1

        seq.append(num)

    return seq

with open("collatz.csv", "w", newline='') as file:

    w = csv.writer(file)

    for i in range(2, 513):

        collatz = collatzSeq(i)

        # w.writerow(collatz) # Pure CSVs

        w.writerow([f"{i}: {','.join(map(str, collatz))}"]) # Prettier version

    print("collatz.csv was made in the same working directory as this project")

**Sample Screen Dumps:**

****

**A number in a box

AI-generated content may be incorrect.**

**A number in a row

AI-generated content may be incorrect.**

(only from 2 to 25 shown in screenshot to not spam images)

**How this was tested:** The code was run to check that the code prints the Collatz sequence from 2 up to 512 (included). Furthermore, both the pure and pretty format were tested by uncommenting one and commenting the other.

In order to present the data in the pretty format, I consulted Grok with the following prompt:

**Make the output look like in this format ‘512: 512,256,128... etc’**

**Question:** 8

**Source Code:**

def nrmethod(num, tolerance, maxIterations):

    if num < 0:

        print("Cannot compute square root of a negative number")

    if num == 0:

        return 0

    x\_0 = num / 2

    for i in range(maxIterations):

            x\_1 = (x\_0 + num / x\_0) / 2

            if abs(x\_0 \* x\_0 - num) < tolerance:

                return x\_1

            x\_0 = x\_1

    print("MAX ITERATIONS REACHED")

    return x\_0

num = float(input("Enter a number "))

tolerance = float(input("Enter tolerance "))

maxIterations = int(input("Enter max iterations "))

result = nrmethod(num, tolerance, maxIterations)

print(f"Approximate square root is {result:.6f}")

**Sample Screen Dumps:**

**A black screen with white text

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**How this was tested:** Valid inputs were given and the result was found to be a close approximation. It was noted that the result was very inaccurate with low max iterations.

**Question:** 9

**Source Code:**

def findRepeated(arr):

freq = {}

nums = []

for i in arr:

if (freq.get(i) is not None):

freq[i] = freq[i] + 1

else:

freq[i] = 1

for i in freq.keys():

if freq.get(i) > 1:

nums.append(i)

return nums

arr = [1,2,1,5,6,5,8,9,56,4,2,5,7,3,2,1,4]

result = findRepeated(arr)

if len(result) != 0:

print(f"Numbers repeated more than once: {result}")

else:

print("No numbers are repeated more than once")

**Sample Screen Dumps:**





**How this was tested:** Different combinations of values in arr were fed to the function and it outputted as expected.

**Question:** 10

**Source Code:**

**Sample Screen Dumps:**

**How this was tested:**