DESIGN AND ANALYSIS OF ALGORITHM

TEAM MEMBERS:

22k-4333: Alishba Hassan

22k-4617: Nimil Zubair

22k-4526: Waniya Badar

Question No:1

A :

Part 1:

def Find\_min\_max(Array,low,high):

  if(low==high):

    return(Array[low],Array[low])

  if((high-low)==1):

    if(Array[low]<Array[high]):

      return (Array[low],Array[high])

    else:

      return(Array[high],Array[low])

  mid=(low+high)//2

  min1,max1=Find\_min\_max(Array,low,mid)

  min2,max2=Find\_min\_max(Array,mid+1,high)

  min\_value=min(min1,min2)

  max\_value=max(max1,max2)

  return(min\_value,max\_value)

n=int(input("Enter Number of Elements in an Array : "))

A=[]

for i in range (n):

  element=int(input(f"Number {i+1}: "))

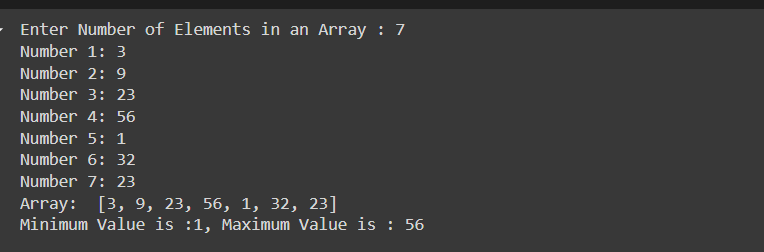
  A.append(element)

print("Array: ",A)

Min,Max=Find\_min\_max(A,0,len(A)-1)

print(f"Minimum Value is :{Min}, Maximum Value is : {Max}")

Example Output:



Part 2:

* Defining Recurrence Relation:

1. Recursive Comparison: Function call itself twice for each half of the array that requires time complexity: T(n/2).
2. Additional Comparison: There are two additional comparisons to find minimum and maximum value .One comparison to find minimum value between min1 and min2Other comparison to find maximum value between max1 and max2.

Hence Recurrence relation is:

T(n)=2T(n/2)+2

Base case: T(2)=1,as only one comparison will be performed.

* Solving Recurrence Relation:

T(n)=2T(n/2)+2 ---(i)

T(n/2)=2T(n/4)+2 ---(ii)

T(n/4)=2T(n/4)+2 ---(iii)

Substituting eq (ii) in eq (i):

T(n)=2(2T(n/4)+2)+2

T(n)=4T(n/4)+4+2 ---(iv)

Substituting eq (iii) in eq (iv):

T(n)=4(2T(n/8)+2)+4+2

T(n)=8T(n/8)+8+4+2

T(n)=2kT(n/2k )+2(2k-1)

Since 2k =T(1):

T(n)=n+2(n-1)

T(n)=3(n/2) -2

Part 3:

Brute Force approach:

1. Scanning array once to find minimum , requiring n-1 comparisons
2. Scanning array once to find maximum , requiring n-1 comparisons

The Algorithm using divide and conquer performs approximately 3(n/2) -2 comparisons, which is more efficient than the brute force approach which requires 2n-2 comparisons. Hence this algorithm performance is more efficient than brute force.

B:

Part 1:

def Power(a,n):

  if n==0:

    return 1

  if n % 2==0:  # if n is even

    c=Power(a,n/2)

    return c\*c

  elif n % 2==1:  # if n is odd

    c=Power(a,(n-1)/2)

    return a\*c\*c

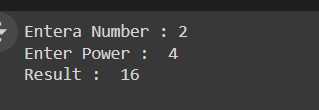
a=int(input("Entera Number : "))

n=int(input("Enter Power :  "))

P=Power(a,n)

print("Result : ",P)

Example Output:



Part 2:

* If n ==0 no multiplication is performed T(0)=1
* Multiplication is performed in two cases:

If n is even while function call the problem is reduce to n/2 and multiplication is T(n)=T(n/2)+1.

If n is odd while function call the problem is reduce to (n-1)/2 and multiplied with T(n)=T(n-1)/2.

* Solving Recurrence Relation:

T(n)=T(n/2)+1 ---(i)

T(n/2)=T(n/4)+1 ----(ii)

T(n/4)=T(n/8)+1 ---(Iii)

Substituting eq (ii) in eq (i):

T(n)=T(n/4)+2 ---(iv)

Substituting eq (iiI) in eq (iv):

T(n)=T(n/8)+3

T(n)=T(n/2k)+k

Since k=log(n)

T(n)=log(n)

Part 3:

The Brute force approach for a^n would require n-1 multiplications as it directly multiply a by itself n-1 times. So Multiplications will be O(n) for brute force.

The divide and conquer method would require Multiplications O(log n) making it more efficient than brute force

C:

def inversions(Array):

  def Inv\_counts(Array,temp\_Array,low,mid,high):

    i=low

    j=mid+1

    k=low

    counts=0

    while i<=mid and j<=high:

      if Array[i]<= Array[j]:

        temp\_Array[k]=Array[i]

        i+=1

      else:

        temp\_Array[k]=Array[j]

        counts+=(mid-i+1)

        j+=1

      k+=1

    while i<=mid:

      temp\_Array[k]=Array[i]

      i+=1

      k+=1

    while j<=high:

      temp\_Array[k]=Array[j]

      j+=1

      k+=1

    for i in range(low,high+1):

      Array[i]=temp\_Array[i]

    return counts

  def countMergeSort(Array,temp\_Array,low,high):

    counts=0;

    if low>=high:

      return 0

    mid=(low+high)//2

    counts+=countMergeSort(Array,temp\_Array,low,mid)

    counts+=countMergeSort(Array,temp\_Array,mid+1,high)

    counts+=Inv\_counts(Array,temp\_Array,low,mid,high)

    return counts

  temp\_Array=[0]\*len(Array)

  return countMergeSort(Array,temp\_Array,0,len(Array)-1)

n=int(input("Enter Number of Elements in an Array : "))

A=[]

for i in range (n):

  element=int(input(f"Number {i+1}: "))

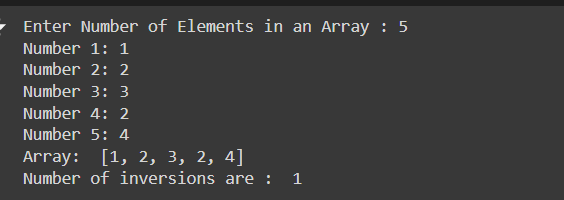
  A.append(element)

print("Array: ",A)

inversions\_count=inversions(A)

print("Number of inversions are : ",inversions\_count)

Example Output:



* Inv\_counts functions counts inversions between the left and right halves of the array
* countMergeSort recursively divides the array into two halves ,counts inversions and then merges them.
* Time complexity O(nlogn) due to recursively splitting and merging steps.
* Brute force would have required O(n2) by comparing each pair (i,j) where i<j and counting inversions
* Divide and Conquer requires only O(nlogn) which is more efficient than brute force.

D:

1. Best Case Complexity:

This happens whenever the pivot selected splits the array in hales that are almost equal at every level of recursive process. In this case, the height of the recursion tree is logarithmic (approx. (log n) ,and each recursive level takes linear time to split the given array into partitions.

Best case will yield Complexity :)O(n log n).

For example:

Input Array:

8, 3, 7, 1, 9, 6, 2, 5, 48, 3, 7, 1, 9, 6, 2, 5, 48, 3, 7, 1, 9, 6, 2, 5, 4

1. Choose Pivot (5):  
   Partition: 3, 1, 2, 43, 1, 2, 43, 1, 2, 4 | 5 | 8, 7, 9, 68, 7, 9, 68, 7, 9, 6
2. Left Partition (3, 1, 2, 43, 1, 2, 43, 1, 2, 4):  
   Pivot (3): 1, 21, 21, 2 | 3 | 444
3. Right Partition (8, 7, 9, 68, 7, 9, 68, 7, 9, 6):  
   Pivot (8): 7, 67, 67, 6 | 8 | 999
4. Further Breakdown:
   * 1,21, 21,2 → Sorted as 1,21, 21,2
   * 7,67, 67,6 → Sorted as 6,76, 76,7
5. Combine Results:  
   Left: 1,2,3,41, 2, 3, 41,2,3,4, Pivot: 555, Right: 6,7,8,96, 7, 8, 96,7,8,9

Final Output:

1, 2, 3, 4, 5, 6, 7, 8, 91, 2, 3, 4, 5, 6, 7, 8, 91, 2, 3, 4, 5, 6, 7, 8, 9

1. Worst Case Complexity:

This happens when the pivot is always the smallest or largest element in the array .In these cases, one subarray will have n-1 elements while the other subarray has 0 elements. This leads to skew recursion tree. The height of recursion tree will be linear, equal to n and each level needs linear work to partition, and hence the total work done will be : n+(n-1)+(n-2)+….1=O(n2).

Worst case will yield Complexity : O(n2).

For example:

### Input Array:

9,8,7,6,5,4,3,2,19, 8, 7, 6, 5, 4, 3, 2, 19,8,7,6,5,4,3,2,1 (sorted in descending order)

1. **Choose Pivot (9):**  
   Partition: | 9 | 8,7,6,5,4,3,2,18, 7, 6, 5, 4, 3, 2, 18,7,6,5,4,3,2,1
2. **Right Partition (**8,7,6,5,4,3,2,18, 7, 6, 5, 4, 3, 2, 18,7,6,5,4,3,2,1**):**  
   Pivot (8): | 8 | 7,6,5,4,3,2,17, 6, 5, 4, 3, 2, 17,6,5,4,3,2,1
3. **Next Right Partition (**7,6,5,4,3,2,17, 6, 5, 4, 3, 2, 17,6,5,4,3,2,1**):**  
   Pivot (7): | 7 | 6,5,4,3,2,16, 5, 4, 3, 2, 16,5,4,3,2,1
4. **Continue Similarly:**  
   Each step selects the largest element as the pivot, leaving the rest of the array in the right partition.
   * Pivot (6): | 6 | 5,4,3,2,15, 4, 3, 2, 15,4,3,2,1
   * Pivot (5): | 5 | 4,3,2,14, 3, 2, 14,3,2,1
   * Pivot (4): | 4 | 3,2,13, 2, 13,2,1
   * Pivot (3): | 3 | 2,12, 12,1
   * Pivot (2): | 2 | 111
   * Pivot (1): | 1 |

### Final Output:

1, 2, 3, 4, 5, 6, 7, 8, 91, 2, 3, 4, 5, 6, 7, 8, 91, 2, 3, 4, 5, 6, 7, 8, 9

E:

Part 1:

def closest\_pair(Array):

  #Sorting array in nlogn

  Array.sort()

  def closest\_inarray(Array):

    min\_diff=float('inf')

    Pair=(None,None)

    for i in range(len(Array)-1):

      difference=Array[i+1]-Array[i]

      if(difference<min\_diff):

        min\_diff=difference

        Pair=(Array[i],Array[i+1])

    return Pair,min\_diff

  closestPair,min\_difference=closest\_inarray(Array)

  return closestPair,min\_difference

a=int(input("Enter Number of Elements in an Array : "))

A=[]

for i in range (a):

  element=int(input(f"Number {i+1}: "))

  A.append(element)

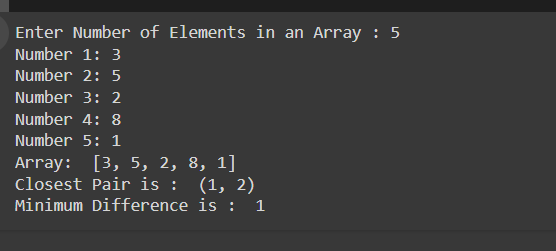
print("Array: ",A)

closestPair,min\_difference=closest\_pair(A)

print("Closest Pair is : ",closestPair)

print("Minimum Difference is : ",min\_difference)

Example Output:



Part 2:

Yes, this is an efficient algorithm for the one-dimensional closest-pair problem:

* Optimal Efficiency: In one-dimensional case: O(n log n) because the problem naturally include a basic operation that yields finding the closest pair.
* Simplicity and Directness : The problem then merely reduces to a simple scan of the consecutive elements, so the solution is elegant and simple

In conclusion, divide and conquer based algorithm is efficient and well suited.

F:

def FindPeak(Array,high,low):

  if(high==low):

      return low

  mid=(low+high)//2

  if(Array[mid]>Array[mid+1] and Array[mid]>Array[mid-1]):

      return mid,Array[mid]

  elif(Array[mid]<Array[mid+1]):

      return FindPeak(Array,high,mid+1)

  else:

      return FindPeak(Array,mid-1,low)

#input number of elements in an array

n=int(input("Enter Number of Elements in an Array : "))

A=[]

for i in range (n):

  element=int(input(f"Number {i+1}: "))

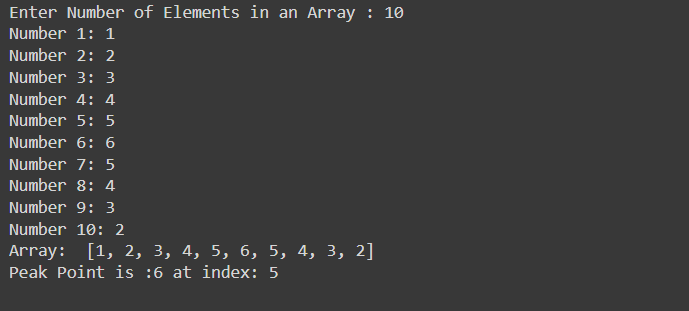
  A.append(element)

print("Array: ",A)

Peak\_index,Peak=FindPeak(A,len(A)-1,0)

print(f"Peak Point is :{Peak} at index: {Peak\_index}")

Example Output:



Time Complexity: O(logn)

As we have solved the problem by breaking Array into two halves like binary search and recursively have solved the problem to achieve time complexity O(logn)

The recurrence relation is:

T(n) <= T(n/2)+c

When n>2,

T(2) <=c

In this problem we divided the Array into two halves and Peak point is where:

(Array[mid]>Array[mid+1] and Array[mid]>Array[mid-1])

If (Array[mid]<Array[mid+1]) then we have checked the left part of the Array

Else the peak point will lie in right part of the array

For example: {1,2,3,4,5,4,2}

1. First call:

Mid=3, value=4,not peak. Move Right

1. Second call:

Mid=5 ,value =4 , not peak. Move left

1. Third call:

Mid=4, value=5 , which is peak point

G:

def Max\_profit(price,low,high,bday,sday,profit):

  if(high==low):

    bday=sday=low

    profit=0

    return(0,low,high)# No profit

  mid=(low+high)//2

  #recursive call to fing max profit in left and right halves of the array

  l\_prof,l\_buy,l\_sell=Max\_profit(price,low,mid,bday,sday,profit) # checking in left Half

  r\_prof,r\_buy,r\_sell=Max\_profit(price,mid+1,high,bday,sday,profit)# Checking in right half

  # Min price in left half:

  min\_lprice=min(price[low:mid+1])

  # MIn price in right half

  max\_rprice=max(price[mid+1:high+1])

  check\_profit=max\_rprice-min\_lprice

  if(l\_prof>=r\_prof and l\_prof>= check\_profit):

    bday=l\_buy

    sday=l\_sell

    profit=l\_prof

    return(profit,bday,sday)

  elif(r\_prof>=l\_prof and r\_prof>=check\_profit):

    bday=r\_buy

    sday=r\_sell

    profit=r\_prof

    return(profit,bday,sday)

  else:

    bday=price.index(min\_lprice,low,mid+1)

    sday=price.index(max\_rprice,mid+1,high+1)

    return(check\_profit,bday,sday)

n=int(input("Enter Number of Days : "))

price=[]

for i in range (n):

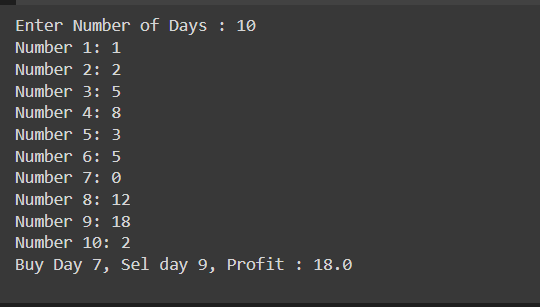
  element=float(input(f"Number {i+1}: "))

  price.append(element)

profit,buyday,sellday=Max\_profit(price,0,len(price)-1,0,0,0)

print(f"Buy Day {buyday+1}, Sel day {sellday+1}, Profit : {profit}")

Example Output:



H:

Part 2:

def inversions(Array):

  def SigInv\_counts(Array,temp\_Array,low,mid,high):

    counts=0

    for i in range (low,mid+1):

      j=mid+1

      while j<=high and Array[i]>2\*Array[j]:

        j+=1

      counts+=(j-(mid+1))

    k=low

    j=mid+1

    i=low

    while i<=mid and j<=high:

      if Array[i]<= Array[j]:

        temp\_Array[k]=Array[i]

        i+=1

      else:

        temp\_Array[k]=Array[j]

        j+=1

      k+=1

    while i<=mid:

      temp\_Array[k]=Array[i]

      i+=1

      k+=1

    while j<=high:

      temp\_Array[k]=Array[j]

      j+=1

      k+=1

    for i in range(low,high+1):

      Array[i]=temp\_Array[i]

    return counts

  def countMergeSort(Array,temp\_Array,low,high):

    counts=0

    if low>=high:

      return 0

    mid=(low+high)//2

    counts+=countMergeSort(Array,temp\_Array,low,mid)

    counts+=countMergeSort(Array,temp\_Array,mid+1,high)

    counts+=SigInv\_counts(Array,temp\_Array,low,mid,high)

    return counts

  temp\_Array=[0]\*len(Array)

  return countMergeSort(Array,temp\_Array,0,len(Array)-1)

n=int(input("Enter Number of Elements in an Array : "))

A=[]

for i in range (n):

  element=int(input(f"Number {i+1}: "))

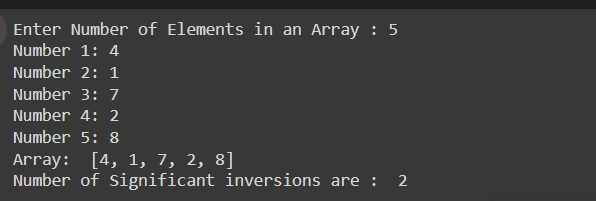
  A.append(element)

print("Array: ",A)

inversions\_count=inversions(A)

print("Number of Significant inversions are : ",inversions\_count)

Example Output:



Part 5:

class Line:

  def \_\_init\_\_(self, a, b):

    self.a = a

    self.b = b

  def y(self, x):

    return self.a \* x + self.b

class Event:

  def \_\_init\_\_(self, x, line, event\_type):

    self.x = x

    self.line = line

    self.type = event\_type

class HiddenSurfaceRemoval:

  @staticmethod

  def find\_visible\_lines(lines):

    events = []

    for line in lines:

      events.append(Event(float('-inf'), line, 1))

      events.append(Event(float('inf'), line, -1))

    events.sort(key = lambda e: e.x)

    active\_lines = []

    visible\_lines = []

    for event in events:

      if event.type == 1:

        active\_lines.append(event.line)

        if HiddenSurfaceRemoval.is\_topmost(active\_lines,event.line, event.x):

          visible\_lines.append(event.line)

      else:

        active\_lines.remove(event.line)

        if active\_lines and HiddenSurfaceRemoval.is\_topmost(active\_lines, active\_lines[-1], event.x):

          visible\_lines.append(active\_lines[-1])

    return visible\_lines

  @staticmethod

  def is\_topmost(lines, line, x):

    for other\_line in lines:

      if other\_line != line and other\_line.y(x) > line.y(x):

        return False

    return True

if \_\_name\_\_ == "\_\_main\_\_":

  lines = [

      Line(1, 2),

      Line(-2, 4),

      Line(0.5, -1),

      Line(3, 0),

      Line(-1, 1)

  ]

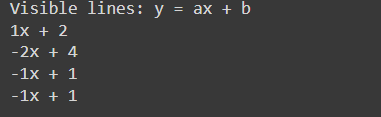
  visible\_lines = HiddenSurfaceRemoval.find\_visible\_lines(lines)

  print("Visible lines: y = ax + b")

  for line in visible\_lines:

    print(f"{line.a}x + {line.b}")

Example Out[ut:



Part 3:

from collections import defaultdict

class BankCard:

  def \_\_init\_\_(self, account\_holder\_name, account\_number):

    self.account\_holder\_name = account\_holder\_name

    self.account\_number = account\_number

  def get\_account\_holder\_name(self):

    return self.account\_holder\_name

  def get\_acount\_number(self):

    return self.account\_number

  def \_\_eq\_\_(self, other):

    if not isinstance(other, BankCard):

      return False

    return self.account\_number == other.account\_number

  def \_\_hash\_\_(self):

    return hash(self.account\_number)

class EquivalenceTestor:

  def are\_equivalent(self, card1, card2):

    return card2 == card2

class FraudDetection:

  @staticmethod

  def find\_majority\_equivalent(cards):

    if not cards:

      return False

      tester = EquivalenceTester()

      candidate = None

      count = 0

      for card in cards:

        if count == 0:

          candidate = card

          count = 1

        else:

          if tester.are\_equivalent(card, candidate):

            count += 1

          else:

            count -= 1

      count = 0

      for card in cards:

        if tester.are\_equivalent(card, candidate):

          count += 1

      return count > len(cards) / 2

if \_\_name\_\_ == "\_\_main\_\_":

  cards = []

  cards.append(BankCard("WaniyaBadar", "12345"))

  cards.append(BankCard("AlishbaHassan", "98765"))

  cards.append(BankCard("WaniyaBadar", "12345"))

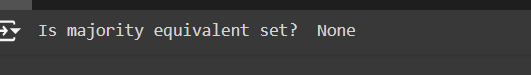
  cards.append(BankCard("WaniyaBadar", "12345"))

  cards.append(BankCard("Nimil", "09007"))

  result = FraudDetection.find\_majority\_equivalent(cards)

  print("Is majority equivalent set? ", result)

Example Output:



Question no:2 and 3

import math

import random

import os

def euclidean\_distance(p1,p2):

  return math.sqrt((p1[0]-p2[0])\*\*2+(p1[1]-p2[1])\*\*2)

def closest\_in\_strip(strip,min\_distance):

  min\_pair=None

  strip.sort(key=lambda x:x[1])

  for i in range(len(strip)):

    for j in range(i+1,len(strip)):

      if(strip[j][1]-strip[i][1]<min\_distance):

        distance=euclidean\_distance(strip[i],strip[j])

        if(distance<min\_distance):

          min\_distance=distance

          min\_pair=(strip[i],strip[j])

  return min\_distance,min\_pair

def closest\_pair\_recursive(points\_sorted\_x):

  if len(points\_sorted\_x)<=3:

    return closest\_in\_strip(points\_sorted\_x,float("inf"))

  mid=len(points\_sorted\_x)//2

  left\_points=points\_sorted\_x[:mid]

  right\_points=points\_sorted\_x[mid:]

  left\_min\_distance, left\_pair=closest\_pair\_recursive(left\_points)

  right\_min\_distance,right\_pair=closest\_pair\_recursive(right\_points)

  min\_distance=min(left\_min\_distance,right\_min\_distance)

  min\_pair=left\_pair if left\_min\_distance<right\_min\_distance else right\_pair

  mid\_x=points\_sorted\_x[mid][0]

  strip=[point for point in points\_sorted\_x if abs(point[0]-mid\_x)<min\_distance]

  strip\_distance,strip\_pair=closest\_in\_strip(strip,min\_distance)

  if strip\_distance<min\_distance:

    return strip\_distance,strip\_pair

  return min\_distance,min\_pair

def closest\_pair(points):

  if len(points)< 2:

    return None, None

  points\_sorted\_x=sorted(points,key=lambda x:x[0])

  return closest\_pair\_recursive(points\_sorted\_x)

def generate\_random\_points(num\_points=100):

  return[(random.randint(0,1000),random.randint(0,1000)) for \_ in range(num\_points)]

def save\_points\_to\_file(points,filename):

  with open(filename,'w') as file:

    for point in points:

      file.write(f"{point[0]} {point[1]}\n")

if \_\_name\_\_=="\_\_main\_\_":

  os.makedirs("closest\_pair\_samples",exist\_ok=True)

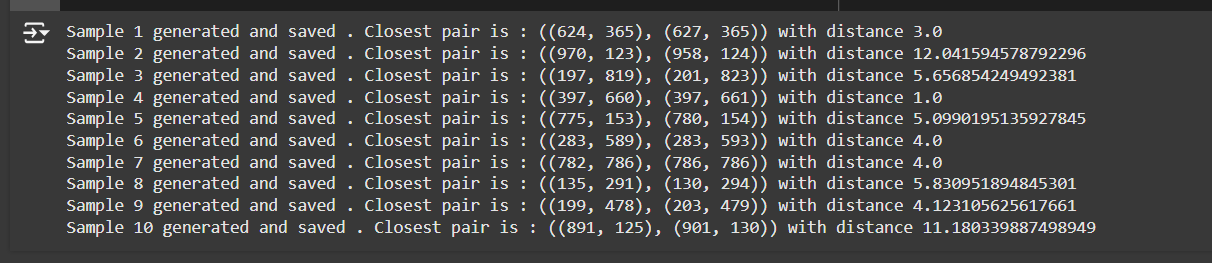
  for i in range(1,11):

    points=generate\_random\_points()

    save\_points\_to\_file(points,f"closest\_pair\_samples/sample\_{i}.txt")

    min\_distance,closest\_points=closest\_pair(points)

    print(f'Sample {i} generated and saved . Closest pair is : {closest\_points} with distance {min\_distance}')



Integer Multiplication:

import random

import os

def karatsuba(x,y):

  if x<10 or y<10:

    return x\*y

  n=max(len(str(x)),len(str(y)))

  m=n//2

  high1,low1=divmod(x,10\*\*m)

  high2,low2=divmod(y,10\*\*m)

  z0=karatsuba(low1,low2)

  z1=karatsuba((low1+high1),(low2+high2))

  z2=karatsuba(high1,high2)

  return z2\*(10\*\*(2\*m))+(z1-z2-z0)\*(10\*\*m)+z0

def generate\_large\_numbers(num\_samples,digit\_length):

    samples=[]

    for \_ in range(num\_samples):

      x=random.randint(10\*\*(digit\_length-1),10\*\*digit\_length-1)

      y=random.randint(10\*\*(digit\_length-1),10\*\*digit\_length-1)

      samples.append((x,y))

    return samples

def save\_samples(samples,base\_path="integer\_multiplication\_samples"):

    os.makedirs(base\_path,exist\_ok=True)

    for i,(x,y) in enumerate(samples):

      filename=os.path.join(base\_path,f"sample\_{i+1}.txt")

      with open(filename,'w') as file:

         file.write(f"{x}\n{y}")

if \_\_name\_\_=="\_\_main\_\_":

   num\_samples=10

   digit\_length=100

   samples=generate\_large\_numbers(num\_samples,digit\_length)

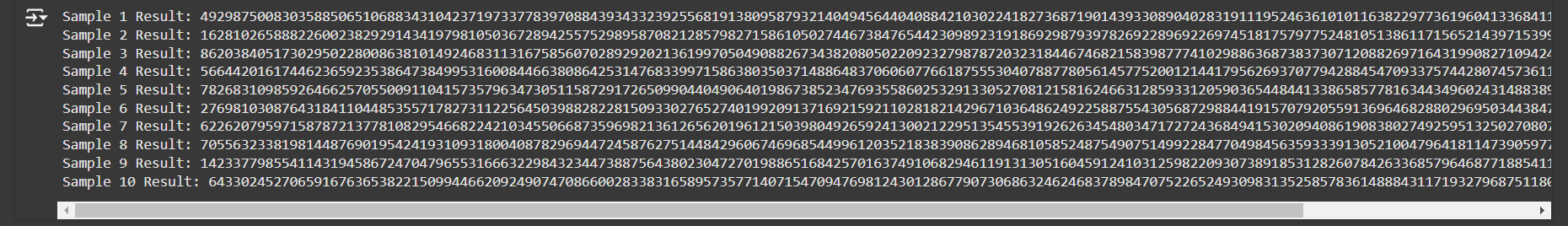
   save\_samples(samples)

   for i,(x,y) in enumerate(samples):

     result=karatsuba(x,y)

     print(f"Sample {i+1} Result: {result}")

output:



Question no:4

import tkinter as tk

from tkinter import filedialog

import math

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# Karatsuba implementation with an improved tree-like visualization

def karatsuba(x, y, depth=0):

    if x < 10 or y < 10:

        return x \* y

    n = max(len(str(x)), len(str(y)))

    m = n // 2

    high1, low1 = divmod(x, 10\*\*m)

    high2, low2 = divmod(y, 10\*\*m)

    # Recursively calculate the three products

    z0 = karatsuba(low1, low2, depth + 1)

    z1 = karatsuba((low1 + high1), (low2 + high2), depth + 1)

    z2 = karatsuba(high1, high2, depth + 1)

    # Improved visualization of recursive steps (text representation)

    tree\_text.insert(tk.END, f"{'  ' \* depth}Depth {depth}: z2={z2}, z1={z1}, z0={z0}\n")

    tree\_text.see(tk.END)

    # Return the result with the corrected formula

    return z2 \* (10\*\*(2 \* m)) + (z1 - z2 - z0) \* (10\*\*m) + z0

# Euclidean distance for Closest Pair

def euclidean\_distance(p1, p2):

    return math.sqrt((p1[0] - p2[0])\*\*2 + (p1[1] - p2[1])\*\*2)

# Closest Pair with scatter plot and highlighted pair

def closest\_pair(points):

    min\_dist = float('inf')

    pair = None

    fig, ax = plt.subplots()

    # Scatter plot of points

    x\_vals, y\_vals = zip(\*points)

    ax.scatter(x\_vals, y\_vals, color='blue', label="Points")

    # Find the closest pair

    for i in range(len(points)):

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

            dist = euclidean\_distance(points[i], points[j])

            if dist < min\_dist:

                min\_dist = dist

                pair = (points[i], points[j])

    # Draw a line and circle between the closest pair

    if pair:

        ax.plot(

            [pair[0][0], pair[1][0]],

            [pair[0][1], pair[1][1]],

            color='red',

            linestyle='--',

            linewidth=2,

            label="Closest Pair"

        )

        # Highlight the closest pair points

        ax.scatter(pair[0][0], pair[0][1], color='green', s=100, edgecolor='black', label="Closest Point 1", zorder=5)

        ax.scatter(pair[1][0], pair[1][1], color='green', s=100, edgecolor='black', label="Closest Point 2", zorder=5)

    ax.set\_title("Closest Pair Visualization")

    ax.legend()

    # Embed the plot in the Tkinter window

    canvas = FigureCanvasTkAgg(fig, master=result\_frame)

    canvas.draw()

    canvas.get\_tk\_widget().pack()

    return min\_dist, pair

# File processing function

def process\_file():

    global tree\_text

    file\_path = filedialog.askopenfilename(filetypes=[("Text files", "\*.txt")])

    if not file\_path:

        return

    with open(file\_path, 'r') as file:

        content = file.read().strip()

        lines = content.split('\n')

    # Clear previous content in the result frame

    for widget in result\_frame.winfo\_children():

        widget.destroy()

    if len(lines) == 2:  # Integer multiplication case

        x, y = map(int, lines)

        # Create Karatsuba frame

        karatsuba\_frame = tk.Frame(result\_frame, bg="#E5E5E5", bd=2, relief="groove", padx=20, pady=20)

        karatsuba\_frame.pack(padx=20, pady=10, fill="both", expand=True)

        # Create the tree\_text widget for Karatsuba

        tree\_text = tk.Text(karatsuba\_frame, bg="#E5E5E5", fg="#000000", font=("Courier New", 12), wrap="word", height=10)

        tree\_text.pack()

        result = karatsuba(x, y)

        # Add Karatsuba result - center-aligned and wrapped

        karatsuba\_result\_label = tk.Label(

            karatsuba\_frame,

            text=f"Karatsuba Multiplication Result: {result}",

            bg="#E5E5E5",

            fg="#000000",

            font=("Courier New", 14, "bold"),

            pady=10,

            wraplength=650,  # Wrap long lines

            justify="center"  # Center the text

        )

        karatsuba\_result\_label.pack()

    else:  # Closest pair case

        points = [tuple(map(int, line.split())) for line in lines]

        # Create Closest Pair frame

        closest\_pair\_frame = tk.Frame(result\_frame, bg="#E5E5E5", bd=2, relief="groove", padx=20, pady=20)

        closest\_pair\_frame.pack(padx=20, pady=10, fill="both", expand=True)

        min\_distance, closest\_points = closest\_pair(points)

        # Add Closest Pair result

        closest\_pair\_result\_label = tk.Label(closest\_pair\_frame,

                                             text=f"Closest Pair: {closest\_points}, Distance: {min\_distance:.2f}",

                                             bg="#E5E5E5", fg="#000000", font=("Courier New", 14, "bold"), pady=10)

        closest\_pair\_result\_label.pack()

    result\_label.config(text="Processing Complete. Results displayed in the respective sections.")

# Tkinter GUI setup

root = tk.Tk()

root.title("Algorithm Runner with Visualization")

root.geometry("800x800")

root.configure(bg="#14213D")

# Header

header\_label = tk.Label(root, text="Algorithm Runner", bg="#FCA311", fg="#000000", font=("Verdana", 28, "bold"))

header\_label.pack(fill="x", pady=(0, 20))

# Instructions

instruction\_frame = tk.Frame(root, bg="#E5E5E5")

instruction\_frame.pack(padx=20, pady=10, anchor="center", fill="x")

instruction\_label = tk.Label(instruction\_frame, text="Upload a text file to process:", bg="#E5E5E5", fg="#000000", font=("Lucida Sans", 14, "bold"))

instruction\_label.pack()

# Upload Button

upload\_button = tk.Button(root, text="Upload and Process File", command=process\_file, bg="#FCA311", fg="#000000", font=("Calibri", 16, "bold"), padx=20, pady=10, bd=0, relief="flat", cursor="hand2")

upload\_button.pack(pady=20)

# Result Frame

result\_frame = tk.Frame(root, bg="#E5E5E5", bd=2, relief="groove", padx=20, pady=20)

result\_frame.pack(padx=20, pady=20, fill="both", expand=True)

# Result Label

result\_label = tk.Label(result\_frame, text="Awaiting file upload...", bg="#E5E5E5", fg="#000000", font=("Courier New", 16, "normal"), wraplength=650, justify="center")

result\_label.pack()

root.mainloop()

