LAB-8:

Finding the maximum and minimum

CODE:

def find\_max\_min(arr, low, high):

if low == high:

return (arr[low], arr[low])

if high == low + 1:

if arr[low] < arr[high]:

return (arr[low], arr[high])

else:

return (arr[high], arr[low])

mid = (low + high) // 2

left\_min, left\_max = find\_max\_min(arr, low, mid)

right\_min, right\_max = find\_max\_min(arr, mid + 1, high)

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

overall\_max = max(left\_max, right\_max)

return (overall\_min, overall\_max)

arr = [3, 5, 1, 2, 4, 8]

low = 0

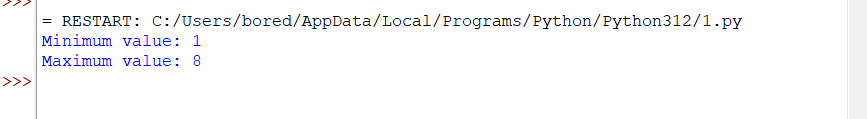
high = len(arr) - 1

min\_val, max\_val = find\_max\_min(arr, low, high)

print(f"Minimum value: {min\_val}")

print(f"Maximum value: {max\_val}")

OUTPUT:



Merge sort

CODE:

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

L = arr[:mid]

R = arr[mid:]

merge\_sort(L)

merge\_sort(R)

i = j = k = 0

while i < len(L) and j < len(R):

if L[i] < R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

while i < len(L):

arr[k] = L[i]

i += 1

k += 1

while j < len(R):

arr[k] = R[j]

j += 1

k += 1

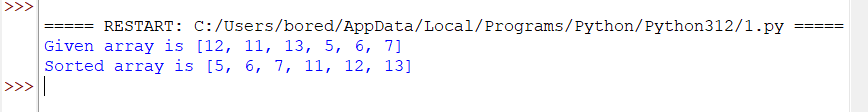
arr = [12, 11, 13, 5, 6, 7]

print("Given array is", arr)

merge\_sort(arr)

print("Sorted array is", arr)

OUTPUT:



Quick sort

CODE:

def partition(array, low, high):

pivot = array[high]

i = low - 1

for j in range(low, high):

if array[j] <= pivot:

i = i + 1

(array[i], array[j]) = (array[j], array[i])

(array[i + 1], array[high]) = (array[high], array[i + 1])

return i + 1

def quickSort(array, low, high):

if low < high:

pi = partition(array, low, high)

quickSort(array, low, pi - 1)

quickSort(array, pi + 1, high)

data = [1, 7, 4, 1, 10, 9, -2]

print("Unsorted Array")

print(data)

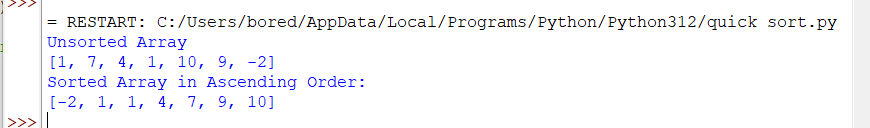
size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')

print(data)

OUTPUT:



Binary search

CODE:

def binary\_search(arr, low, high, x):

if high >= low:

mid = (high + low) // 2

if arr[mid] == x:

return mid

elif arr[mid] > x:

return binary\_search(arr, low, mid - 1, x)

else:

return binary\_search(arr, mid + 1, high, x)

else:

return -1

arr = [ 2, 3, 4, 10, 40 ]

x = 10

result = binary\_search(arr, 0, len(arr)-1, x)

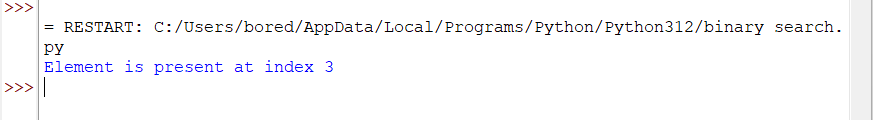
if result != -1:

print("Element is present at index", str(result))

else:

print("Element is not present in array")

OUTPUT:



Strassens matrix multiplication

CODE:

import numpy as np

def split\_matrix(matrix):

"""

Split a matrix into four quadrants.

"""

row, col = matrix.shape

row2, col2 = row // 2, col // 2

return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:, col2:]

def strassen\_multiply(matrix1, matrix2):

"""

Perform matrix multiplication using Strassen's algorithm.

"""

# Base case: if the matrices are small enough, just multiply conventionally

if len(matrix1) <= 2:

return np.dot(matrix1, matrix2)

# Split matrices into quadrants

A, B, C, D = split\_matrix(matrix1)

E, F, G, H = split\_matrix(matrix2)

# Calculate the products needed for Strassen's algorithm

P1 = strassen\_multiply(A, F - H)

P2 = strassen\_multiply(A + B, H)

P3 = strassen\_multiply(C + D, E)

P4 = strassen\_multiply(D, G - E)

P5 = strassen\_multiply(A + D, E + H)

P6 = strassen\_multiply(B - D, G + H)

P7 = strassen\_multiply(A - C, E + F)

# Calculate the quadrants of the result matrix

result\_top\_left = P5 + P4 - P2 + P6

result\_top\_right = P1 + P2

result\_bottom\_left = P3 + P4

result\_bottom\_right = P1 + P5 - P3 - P7

# Combine the quadrants into the result matrix

top\_half = np.hstack((result\_top\_left, result\_top\_right))

bottom\_half = np.hstack((result\_bottom\_left, result\_bottom\_right))

return np.vstack((top\_half, bottom\_half))

# Example usage:

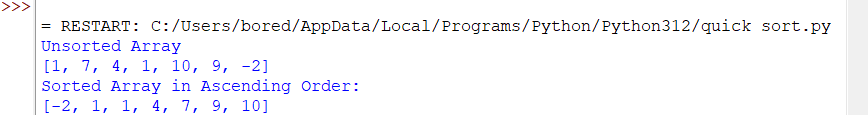
matrix1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])

matrix2 = np.array([[17, 18, 19, 20], [21, 22, 23, 24], [25, 26, 27, 28], [29, 30, 31, 32]])

result = strassen\_multiply(matrix1, matrix2)

print(result)

OUTPUT:



Karatsuba algorithm for multiplication

CODE:

import re

def findSum(str1, str2):

if len(str1) > len(str2):

str1, str2 = str2, str1

result = ""

n1, n2 = len(str1), len(str2)

str1, str2 = str1.zfill(n2), str2.zfill(n2)

carry = 0

for i in range(n2 - 1, -1, -1):

sum\_val = (int(str1[i]) - 0) + (int(str2[i]) - 0) + carry

result = str(sum\_val % 10 + 0) + result

carry = sum\_val // 10

if carry:

result = str(carry + 0) + result

return result

def findDiff(str1, str2):

result = ""

n1, n2 = len(str1), len(str2)

str1, str2 = str1.zfill(n2), str2.zfill(n2)

carry = 0

for i in range(n2 - 1, -1, -1):

sub = (int(str1[i]) - 0) - (int(str2[i]) - 0) - carry

if sub < 0:

sub += 10

carry = 1

else:

carry = 0

result = str(sub + 0) + result

return result

def removeLeadingZeros(s):

pattern = "^0+(?!$)"

s = re.sub(pattern, "", s)

return s

def multiply(A, B):

if len(A) < 10 or len(B) < 10:

return str(int(A) \* int(B))

n = max(len(A), len(B))

n2 = n // 2

A = A.zfill(n)

B = B.zfill(n)

Al, Ar = A[:n2], A[n2:]

Bl, Br = B[:n2], B[n2:]

p = multiply(Al, Bl)

q = multiply(Ar, Br)

r = multiply(findSum(Al, Ar), findSum(Bl, Br))

r = findDiff(r, findSum(p, q))

return removeLeadingZeros(findSum(findSum(p + '0' \* n, r + '0' \* n2), q))

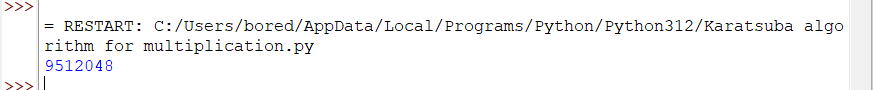
if \_\_name\_\_ == "\_\_main\_\_":

A = "1456"

B = "6533"

print(multiply(A, B))

OUTPUT:



Closest pair of points using divide and conquer

CODE:

import math

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def compareX(a, b):

p1 = a

p2 = b

return (p1.x - p2.x)

def compareY(a, b):

p1 = a

p2 = b

return (p1.y - p2.y)

def dist(p1, p2):

return math.sqrt((p1.x - p2.x)\*(p1.x - p2.x) + (p1.y - p2.y)\*(p1.y - p2.y))

def bruteForce(P, n):

min\_dist = float("inf")

for i in range(n):

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

if dist(P[i], P[j]) < min\_dist:

min\_dist = dist(P[i], P[j])

return min\_dist

def min(x, y):

return x if x < y else y

def stripClosest(strip, size, d):

min\_dist = d

strip = sorted(strip, key=lambda point: point.y)

for i in range(size):

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

if (strip[j].y - strip[i].y) >= min\_dist:

break

if dist(strip[i], strip[j]) < min\_dist:

min\_dist = dist(strip[i], strip[j])

return min\_dist

def closestUtil(P, n):

if n <= 3: return bruteForce(P, n)

mid = n//2

midPoint = P[mid]

dl = closestUtil(P, mid)

dr = closestUtil(P[mid:], n - mid)

d = min(dl, dr)

strip = []

for i in range(n):

if abs(P[i].x - midPoint.x) < d:

strip.append(P[i])

return min(d, stripClosest(strip, len(strip), d))

def closest(P, n):

P = sorted(P, key=lambda point: point.x)

return closestUtil(P, n)

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

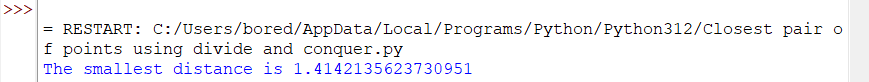
P = [Point(x=2, y=3), Point(x=12, y=30),

Point(x=40, y=50), Point(x=5, y=1), Point(x=12, y=10), Point(x=3, y=4)]

n = len(P)

print("The smallest distance is", closest(P, n))

OUTPUT:



Median of medians

CODE:

def median\_of\_medians(arr):

if len(arr) <= 5:

return sorted(arr)[len(arr) // 2]

chunks = [arr[i:i+5] for i in range(0, len(arr), 5)]

medians = [sorted(chunk)[len(chunk) // 2] for chunk in chunks]

pivot = median\_of\_medians(medians)

lesser = [x for x in arr if x < pivot]

equal = [x for x in arr if x == pivot]

greater = [x for x in arr if x > pivot]

if len(lesser) == k:

return pivot

elif len(lesser) < k:

return median\_of\_medians(greater)

else:

return median\_of\_medians(lesser)

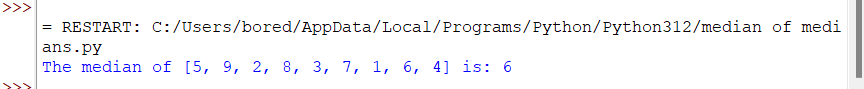
arr = [5, 9, 2, 8, 3, 7, 1, 6, 4]

k = 5

result = median\_of\_medians(arr)

print(f"The median of {arr} is: {result}")

OUTPUT:



Meet in middle technique

CODE:

from typing import List

import bisect

X = [0] \* 2000005

Y = [0] \* 2000005

def calcsubarray(a: List[int], x: List[int], n: int, c: int) -> None:

for i in range((1 << n)):

s = 0

for j in range(n):

if (i & (1 << j)):

s += a[j + c]

x[i] = s

def solveSubsetSum(a: List[int], n: int, S: int) -> int:

global Y

calcsubarray(a, X, n // 2, 0)

calcsubarray(a, Y, n - n // 2, n // 2)

size\_X = 1 << (n // 2)

size\_Y = 1 << (n - n // 2)

YY = Y[:size\_Y]

YY.sort()

Y = YY

maxx = 0

for i in range(size\_X):

if (X[i] <= S):

p = bisect.bisect\_left(Y, S - X[i])

if (p == size\_Y or (p < size\_Y and Y[p] != (S - X[i]))):

p -= 1

if ((Y[p] + X[i]) > maxx):

maxx = Y[p] + X[i]

return maxx

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

a = [3, 34, 4, 12, 5, 2]

n = len(a)

S = 10

print("Largest value smaller than or equal to given sum is {}".format(

solveSubsetSum(a, n, S)))

OUTPUT:

