1.finding maximum and minimum

def max\_min\_naive(arr):

max\_val = arr[0]

min\_val = arr[0]

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

if arr[i] > max\_val:

max\_val = arr[i]

if arr[i] < min\_val:

min\_val = arr[i]

return max\_val, min\_val 3

arr = [6, 4, 26, 14, 33, 64, 46]

max\_val, min\_val = max\_min\_naive(arr)

print("Maximum element is:", max\_val)

print("Minimum element is:", min\_val)

2.merge sort

def merge(arr,l,m,r):

n1=m-l+1

n2=r-m

L=[0]\*(n1)

R=[0]\*(n2)

for i in range(0,n1):

L[i]=arr[l+i]

for j in range(0,n2):

R[j] = arr[m+1+j]

i=0

j=0

k=l

while i<n1 and j<n2:

if L[i]<=R[j]:

arr[k]=L[i]

i+=1

else:

arr[k]=R[j]

j+=1

k+=1

while i <n1:

arr[k]=L[i]

i+=1

k+=1

while j<n2:

arr[k]=R[j]

j+=1

k+=1

def mergeSort(arr, l, r):

if l < r:

m = l+(r-l)//2

mergeSort(arr, l, m)

mergeSort(arr, m+1, r)

merge(arr, l, m, r)

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

n = len(arr)

print("Given array is")

for i in range(n):

print("%d" % arr[i],end=" ")

mergeSort(arr, 0, n-1)

print("\n\nSorted array is")

for i in range(n):

print("%d" % arr[i],end=" ")

3.quick sort

def partition(array, low, high):

i = low - 1

for j in range(low, high):

if array[j] <= pivot:

(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)

4. binary search

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")

5.matrix multipilication using strassen formula

import numpy as np

A = np.array([[12,34],[22,10]])

B = np.array([[3,4],[2,1]])

c = np.zeros((2,2))

m1,m2,m3,m4,m5,m6,m7=0,0,0,0,0,0,0

print("The first matrix is")

for i in range(2):

print()

for j in range(2):

print(A[i][j],end="\t")

print("\nThe second matrix is: ")

for i in range(2):

print()

for j in range(2):

print(B[i][j], end="\t")

m1=(A[0][0]+A[1][1])\*(B[0][0]+B[1][1])

m2=(A[1][0]+A[1][1])\*B[0][0]

m3=A[0][0]\*(B[0][1]-B[1][1])

m4 =A[1][1]\*(B[1][0]-B[0][0])

m5=(A[0][0]+A[0][1])\*B[1][1]

m6=(A[1][0]-A[0][0])\*(B[0][0]+B[0][1])

m7=(A[0][1]-A[1][1])\*(B[1][0]+B[1][1])

c[0][0]=m1+m4-m5+m7

c[0][1]=m3+m5

c[1][0]=m2+m4

c[1][1]=m1-m2+m3+m6

print("\nProduct achieved using Strassen's algorithm")

for i in range(2):

print()

for j in range(2):

print(c[i][j],end="\t")

6.karatsubha

import math

def karatsuba(X, Y):

if X < 10 and Y < 10:

return X \* Y

size = max(get\_size(X), get\_size(Y))

if size < 10:

return X \* Y

size = (size // 2) + (size % 2)

multiplier = 10 \*\* size

b = X // multiplier

a = X - (b \* multiplier)

d = Y // multiplier

c = Y - (d \* size)

u = karatsuba(a, c)

z = karatsuba(a + b, c + d)

v = karatsuba(b, d)

return u + ((z - u - v) \* multiplier) + (v \* (10 \*\* (2 \* size)))

def get\_size(value):

count = 0

while value > 0:

count += 1

value //= 10

return count

x = 145623

y = 653324

print("The final product is: ", end="")

product = karatsuba(x, y)

print(product)

7.closet pair using divide and conquer

from math import sqrt

from random import randint

arr1=[]

dist=0

p1=[]

p2=[]

min1=1000

for i in range(0, 100):

arr1.append([randint(0,100),randint(0,100)])

print(arr1)

print("\n")

def dist(a,b):

x=pow((a[0]-b[0]),2)

y=pow((a[1]-b[1]),2)

return sqrt(x+y)

for i in range(0, len(arr1)):

for j in range(i+1, len(arr1)):

dis=dist(arr1[i],arr1[j])

if(dis<min1):

min1=dis

p1=arr1[i]

p2=arr1[j+1]

print(p1,"",p2,min1)

#print (sorted(arr1))

8.median of median

def median\_of\_medians(elems):

sublists = [elems[j:j+5]

for j in range(0, len(elems), 5)]

medians = []

for sublist in sublists:

medians.append(sorted(sublist)[len(sublist)/2])

if len(medians) <= 5:

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

else:

return median\_of\_medians(medians)

9.meet in middle technique

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)))