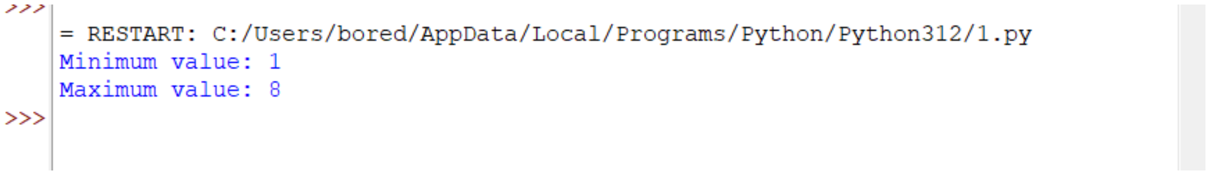
LAB-8:

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:



```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/1.py
    Minimum value: 1
    Maximum value: 8
>>>
```

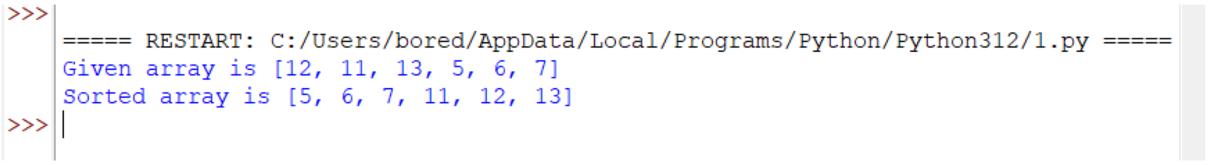
Merge sort

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:



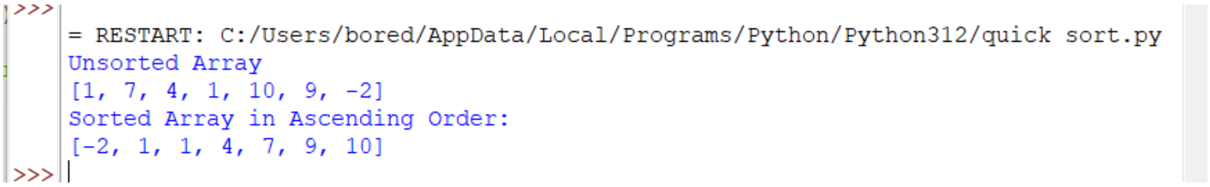
```
>>>
    ===== RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/1.py =====
    Given array is [12, 11, 13, 5, 6, 7]
    Sorted array is [5, 6, 7, 11, 12, 13]
>>> |
```

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:



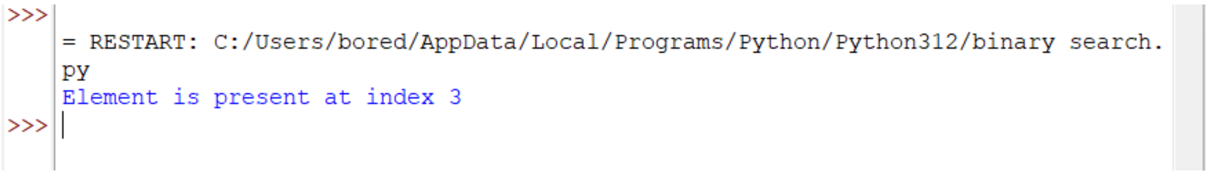
```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/quick sort.py
Unsorted Array
[1, 7, 4, 1, 10, 9, -2]
Sorted Array in Ascending Order:
[-2, 1, 1, 4, 7, 9, 10]
>>> |
```

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:



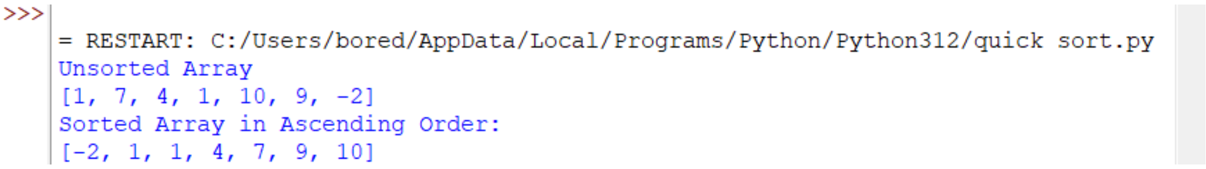
```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/binary search.
py
Element is present at index 3
>>> |
```

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:



```
>>> 
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/quick sort.py
Unsorted Array
[1, 7, 4, 1, 10, 9, -2]
Sorted Array in Ascending Order:
[-2, 1, 1, 4, 7, 9, 10]
```

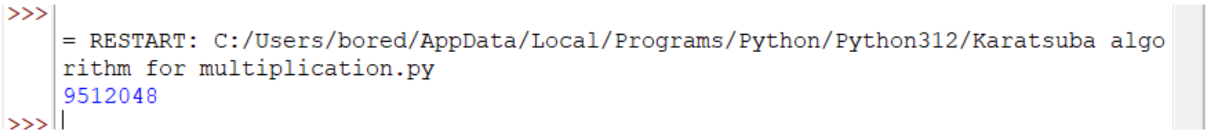
Karatsuba algorithm for multiplication

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:



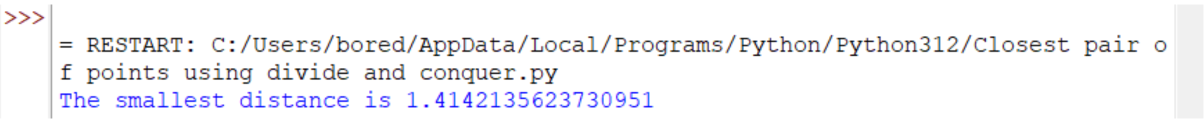
```
>>>
    = RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/Karatsuba algo
    rithm for multiplication.py
    9512048
>>>
```

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:



```
>>> 
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/Closest pair o
f points using divide and conquer.py
The smallest distance is 1.4142135623730951
```

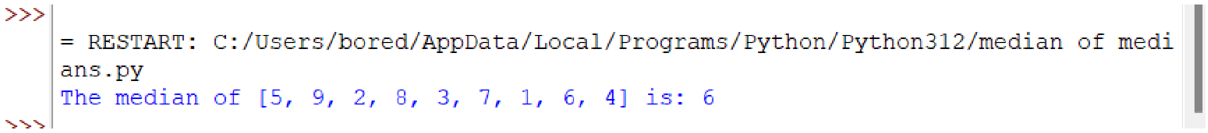
Median of medians

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:



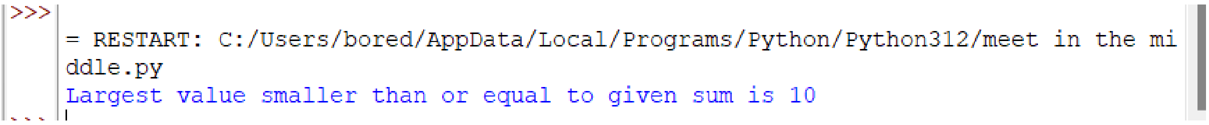
```
>>> 
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/median of medians.py
The median of [5, 9, 2, 8, 3, 7, 1, 6, 4] is: 6
>>> 
```

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:



```
>>>
= RESTART: C:/Users/bored/AppData/Local/Programs/Python/Python312/meet in the mi
ddle.py
Largest value smaller than or equal to given sum is 10
```