LAB TEST – 2

1.Finding the maximum and minimum

```
def find_max_min(arr):
    if not arr:
        return None, None

    max_elem = arr[0]
    min_elem = arr[0]

    for elem in arr:
        if elem > max_elem:
            max_elem = elem
        if elem < min_elem:
            min_elem = elem

    return max_elem, min_elem

arr = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
print("Max and Min:", find_max_min(arr))
```

Output:

```
=== RESTART: C:/Users
Max and Min: (9, 1)
```

2.Merge sort

```
File  Edit  Format  Run  Options  Window  Help
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]
merge_sort(arr)
print("Sorted array is:", arr)
```

Output:

```
Sorted array is: [5, 6, 7, 11, 12, 13]
```

3.Quick sort

```
def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1

    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1

def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)

arr = [10, 7, 8, 9, 1, 5]
quick_sort(arr, 0, len(arr) - 1)
print("Sorted array is:", arr)
```

Output:

```
Sorted array is: [1, 5, 7, 8, 9, 10]
```

4.Binary search

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
    return -1

arr = [2, 3, 4, 10, 40]
x = 10
result = binary_search(arr, x)
print("Element is present at index" if result != -1 else "Element is not present
```

Output:

```
Element is present at index 3
```

5. Strassen's Matrix Multiplication

```python
import numpy as np

def strassen_matrix_multiply(x, y):
    if len(x) == 1:
        return x * y

    size = len(x) // 2
    a = x[:size, :size]
    b = x[:size, size:]
    c = x[size:, :size]
    d = x[size:, size:]

    e = y[:size, :size]
    f = y[:size, size:]
    g = y[size:, :size]
    h = y[size:, size:]

    p1 = strassen_matrix_multiply(a, f - h)
    p2 = strassen_matrix_multiply(a + b, h)
    p3 = strassen_matrix_multiply(c + d, e)
    p4 = strassen_matrix_multiply(d, g - e)
    p5 = strassen_matrix_multiply(a + d, e + h)
    p6 = strassen_matrix_multiply(b - d, g + h)
    p7 = strassen_matrix_multiply(a - c, e + f)

    result = np.zeros((len(x), len(y)))
```

Output:

```
[[19. 22.]
 [43. 50.]]

=== Code Execution Successful ===
```

6. Karatsuba Algorithm for Multiplication

```
def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y

    n = max(len(str(x)), len(str(y)))
    m = n // 2

    x_high, x_low = divmod(x, 10**m)
    y_high, y_low = divmod(y, 10**m)

    z0 = karatsuba(x_low, y_low)
    z1 = karatsuba((x_low + x_high), (y_low + y_high))
    z2 = karatsuba(x_high, y_high)

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

x = 1234
y = 5678
result = karatsuba(x, y)
print("Karatsuba result:", result)
```

Output:

```
Karatsuba result: 7006652
```

7. Closest Pair of Points using Divide and Conquer

```
import math

def dist(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def brute_force(P):
    min_val = float('inf')
    n = len(P)
    for i in range(n):
        for j in range(i + 1, n):
            if dist(P[i], P[j]) < min_val:
                min_val = dist(P[i], P[j])
    return min_val

def strip_closest(strip, d):
    min_val = d
    strip.sort(key = lambda point: point[1])
    for i in range(len(strip)):
        for j in range(i+1, len(strip)):
            if (strip[j][1] - strip[i][1]) < min_val:
                min_val = min(min_val, dist(strip[i], strip[j]))
    return min_val

def closest_util(P, Q, n):
    if n <= 3:
        return brute_force(P)

    mid = n // 2
    mid_point = P[mid]

    dl = closest_util(P[:mid], Q, mid)
    dr = closest_util(P[mid:], Q, n - mid)

    d = min(dl, dr)

    strip = []
    for i in range(n):
        if abs(P[i][0] - mid_point[0]) < d:
            strip.append(P[i])
```

Output:

```
The smallest distance is: 1.4142135623730951
```

8.Median of medians

```
def partition(arr, low, high, pivot):
    i = low
    for j in range(low, high):
        if arr[j] == pivot:
            arr[j], arr[high] = arr[high], arr[j]
            break
    pivot = arr[high]
    for j in range(low, high):
        if arr[j] <= pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i

def select(arr, low, high, k):
    if low == high:
        return arr[low]

    n = high - low + 1
    medians = []
    for i in range(0, n // 5):
        medians.append(sorted(arr[low + i*5 : low + i*5 + 5])[2])
    if n % 5:
        medians.append(sorted(arr[low + (n // 5)*5 : low + (n // 5)*5 + n % 5])[

    if len(medians) <= 5:
        pivot = sorted(medians)[len(medians) // 2]
    else:
        pivot = select(medians, 0, len(medians) - 1, len(medians) // 2)

    pivot_index = partition(arr, low, high, pivot)

    if pivot_index - low == k:
        return arr[pivot_index]
    elif pivot_index - low > k:
        return select(arr, low, pivot_index - 1, k)
    else:
        return select(arr, pivot_index + 1, high, k - pivot_index + low - 1)

def median of medians(arr, k):
```

Output:

```
3rd smallest element is 7
```

9. Meet in middle technique code

```python
def subset_sum(arr, target):
    # Split the array into two halves
    n = len(arr)
    mid = n // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    # Function to calculate all possible sums of subsets
    def get_all_sums(arr):
        sums = []
        n = len(arr)
        for i in range(1 << n):
            subset_sum = 0
            for j in range(n):
                if i & (1 << j):
                    subset_sum += arr[j]
            sums.append(subset_sum)
        return sums

    # Get all possible sums for both halves
    left_sums = get_all_sums(left_half)
    right_sums = get_all_sums(right_half)

    # Create a set for the right sums for faster lookup
    right_sums_set = set(right_sums)

    # Check if there is any combination that sums to the target
    for left_sum in left_sums:
        if (target - left_sum) in right_sums_set:
            return True

    return False

arr = [3, 34, 4, 12, 5, 2]
target = 9
print(f"Is there a subset with sum {target}?", subset_sum(arr, target))
```

Output:

```
Is there a subset with sum 9? True
```