Day 6 programs

1.Maximum and minimum:

```
numbers = [3, 7, 1, 9, 4, 6, 8, 2, 5]
max_num = max(numbers)
min_num = min(numbers)
print("Maximum number:", max_num)
print("Minimum number:", min_num)
```

2.merge sort:

```
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
    return arr
arr = [12, 11, 13, 5, 6, 7]
sorted_arr = merge_sort(arr)
print("Sorted array:", sorted_arr)
```

3.Quick sort:

```
def quick_sort_inplace(arr, low, high):
    if low < high:
        pivot_index = partition(arr, low, high)
        quick_sort_inplace(arr, low, pivot_index - 1)
        quick_sort_inplace(arr, pivot_index + 1, high)
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
arr = [29, 10, 14, 37, 13]
quick_sort_inplace(arr, 0, len(arr) - 1)
print(arr)
```

4.Binary search:

```
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return –1
```

5.Strassens matrix multiplication:

```
def strassen_matrix_multiply(A, B):
    n = len(A)
    if n == 1:
        return [[A[0][0] * B[0][0]]]

    new_size = n // 2
    A11 = [row[:new_size] for row in A[:new_size]]
    A12 = [row[new_size:] for row in A[:new_size]]
    A21 = [row[:new_size] for row in A[new_size:]]
    A22 = [row[new_size:] for row in A[new_size:]]
    B11 = [row[:new_size] for row in B[:new_size]]
    B12 = [row[new_size:] for row in B[:new_size]]
    B21 = [row[:new_size] for row in B[new_size:]]
    B22 = [row[new_size:] for row in B[new_size:]]
    S1 = [[B12[i][j] - B22[i][j] for j in range(new_size)] for i in range(new_size)]
    S2 = [[A11[i][j] + A12[i][j] for j in range(new_size)] for i in range(new_size)]
    S3 = [[A21[i][j] + A22[i][j] for j in range(new_size)] for i in range(new_size)]
    S4 = [[B21[i][j] - B11[i][j] for j in range(new_size)] for i in range(new_size)]
    S5 = [[A11[i][j] + A22[i][j] for j in range(new_size)] for i in range(new_size)]
    S6 = [[B11[i][j] + B22[i][j] for j in range(new_size)] for i in range(new_size)]
    S7 = [[A12[i][j] - A22[i][j] for j in range(new_size)] for i in range(new_size)]
    S8 = [[B21[i][j] + B22[i][j] for j in range(new_size)] for i in range(new_size)]
    S9 = [[A11[i][j] - A21[i][j] for j in range(new_size)] for i in range(new_size)]
    S10 = [[B11[i][j] + B12[i][j] for j in range(new_size)] for i in range(new_size)]
    P1 = strassen_matrix_multiply(A11, S1)
    P2 = strassen_matrix_multiply(S2, B22)
    P3 = strassen_matrix_multiply(S3, B11)
    P4 = strassen_matrix_multiply(A22, S4)
    P5 = strassen_matrix_multiply(S5, S6)
    P6 = strassen_matrix_multiply(S7, S8)
    P7 = strassen_matrix_multiply(S9, S10)
    C11 = [[P5[i][j] + P4[i][j] - P2[i][j] + P6[i][j] for j in range(new_size)] for i in range(new_size)]
    C12 = [[P1[i][j] + P2[i][j] for j in range(new_size)] for i in range(new_size)]
    C21 = [[P3[i][j] + P4[i][j] for j in range(new_size)] for i in range(new_size)]
    C22 = [[P5[i][j] + P1[i][j] - P3[i][j] - P7[i][j] for j in range(new_size)] for i in range(new_size)]
    result = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(new_size):
        for j in range(new_size):
            result[i][j] = C11[i][j]
            result[i][j + new_size] = C12[i][j]
            result[i + new_size][j] = C21[i][j]
            result[i + new_size][j + new_size] = C22[i][j]
    return result
```

6. Karatsuba algorithm for multiplication

```
def karatsuba(x, y):
    if x < 10 or y < 10:
        return x * y
    m = max(len(str(x)), len(str(y)))
    m2 = m // 2
    high1, low1 = divmod(x, 10**m2)
    high2, low2 = divmod(y, 10**m2)
    z0 = karatsuba(low1, low2)
    z1 = karatsuba((low1 + high1), (low2 + high2))
    z2 = karatsuba(high1, high2)
    return z2 * 10**(2*m2) + (z1 - z2 - z0) * 10**m2 + z0
result = karatsuba(1234, 5678)
print(result)
```

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(points, n):
    min_dist = float('inf')
    for i in range(n):
        for j in range(i + 1, n):
            if dist(points[i], points[j]) < min_dist:
                min_dist = dist(points[i], points[j])
    return min_dist
 def closest_pair(points):
    points.sort(key=lambda x: x[0])
    return closest_pair_util(points, len(points))
 def closest_pair_util(points, n):
    if n <= 3:
        return brute_force(points, n)
    mid = n // 2
    mid_point = points[mid]
```

```
        dl = closest_pair_util(points[:mid], mid)
    dr = closest_pair_util(points[mid:], n - mid)
        d = min(dl, dr)
        strip = []
    for point in points:
        if abs(point[0] - mid_point[0]) < d:
            strip.append(point)
        strip.sort(key=lambda x: x[1])
        min_strip = d
    for i in range(len(strip)):
        j = i + 1
        while j < len(strip) and (strip[j][1] - strip[i][1]) < min_strip:
            min_strip = dist(strip[i], strip[j])
            j += 1
        return min(d, min_strip)
 points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
print("The smallest distance is", closest_pair(points))
```

8.Median of medians:

```
import statistics
def median_of_medians(arr):
    sublists = [arr[x:x+5] for x in range(0, len(arr), 5)]
    medians = [statistics.median(sublist) for sublist in sublists]
    if len(medians) <= 5:
        pivot = statistics.median(medians)
    else:
        pivot = median_of_medians(medians)
    lower = [x for x in arr if x < pivot]
    upper = [x for x in arr if x > pivot]
    if len(lower) == 5:
        return pivot
    elif len(lower) > 5:
        return median_of_medians(lower)
    else:
        return median_of_medians(upper)
arr = [3, 8, 2, 10, 5, 1, 7, 4, 6, 9]
result = median_of_medians(arr)
print("Median of the list:", result)
```

9.Meet in middle technique:

```
def meet_in_middle(arr, target):
    n = len(arr)
    result = []
    for i in range(1 << n):
        subset = [arr[j] for j in range(n) if (i & (1 << j))]
        if sum(subset) == target:
            result.append(subset)
    return result
arr = [3, 1, 7, 5, 9, 2]
target = 10
print(meet_in_middle(arr, target))
```