ASSIGNMENT (25.06.24)

Median of Medians

1. To Implement the Median of Medians algorithm ensures that you handle the worst-case time complexity efficiently while finding the k-th smallest element in an unsorted array.

arr = [12, 3, 5, 7, 19] k = 2 Expected Output:5

arr = [12, 3, 5, 7, 4, 19, 26] k = 3 Expected Output:5

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6 Expected Output:6

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

def select_pivot(arr, low, high):
    if high - low + 1 <= 5:
        sublist = arr[low:high+1]
        sublist.sort()
        return low + len(sublist) // 2
    medians = []
    for i in range(low, high + 1, 5):
        sub_high = i + 4 if i + 4 <= high else high
        sublist = arr[i:sub_high + 1]
        sublist.sort()
        medians.append(sublist[len(sublist) // 2])
    return select_pivot(medians, 0, len(medians) - 1)

def median_of_medians(arr, low, high, k):
```

```
    if low == high:
        return arr[low]
    pivot_index = select_pivot(arr, low, high)
    pivot_index = partition(arr, low, high, pivot_index)
    if k == pivot_index:
        return arr[k]
    elif k < pivot_index:
        return median_of_medians(arr, low, pivot_index - 1, k)
    else:
        return median_of_medians(arr, pivot_index + 1, high, k)

def find_kth_smallest(arr, k):
    return median_of_medians(arr, 0, len(arr) - 1, k - 1)
arr1 = [12, 3, 5, 7, 19]
k1 = 2
print(find_kth_smallest(arr1, k1))
Output: 5
arr2 = [12, 3, 5, 7, 4, 19, 26]
k2 = 3
print(find_kth_smallest(arr2, k2))
Output: 5
arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k3 = 6
print(find_kth_smallest(arr3, k3))
Output: 6
```

2. To Implement a function median_of_medians(arr, k) that takes an unsorted array arr and an integer k, and returns the k-th smallest element in the array.

arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] k = 6

arr = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27] k = 5

Output: An integer representing the k-th smallest element in the array.

```
def partition(arr, low, high, pivot_index):
    pivot_value = arr[pivot_index]
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    store_index = low
```

```
    for i in range(low, high):
        if arr[i] < pivot_value:
            arr[store_index], arr[i] = arr[i], arr[store_index]
            store_index += 1
    arr[store_index], arr[high] = arr[high], arr[store_index]
    return store_index

def select_pivot(arr, low, high):
    if high - low + 1 <= 5:
        sublist = arr[low:high+1]
        sublist.sort()
        return low + len(sublist) // 2
    medians = []
    for i in range(low, high + 1, 5):
        sub_high = min(i + 4, high)
        sublist = arr[i:sub_high + 1]
        sublist.sort()
        medians.append(sublist[len(sublist) // 2])
    return select_pivot(medians, 0, len(medians) - 1)

def median_of_medians(arr, low, high, k):
    if low == high:
        return arr[low]
    pivot_index = select_pivot(arr, low, high)
    pivot_index = partition(arr, low, high, pivot_index)
    if k == pivot_index:
        return arr[k]
    elif k < pivot_index:
        return median_of_medians(arr, low, pivot_index - 1, k)
    else:
        return median_of_medians(arr, pivot_index + 1, high, k)

def find_kth_smallest(arr, k):
    return median_of_medians(arr, 0, len(arr) - 1, k - 1)
```

```
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k1 = 6
print(find_kth_smallest(arr1, k1))
Output: 6
arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]
k2 = 5
print(find_kth_smallest(arr2, k2))
Output: 21
```

Closest Pair of Points(Divide and Conquer)

1. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).

(i) Input : points = [[1,3],[-2,2],[5,8],[0,1]],k=2
Output:[[-2, 2], [0, 1]]

(ii) Input: points = [[1, 3], [-2, 2]], k = 1
Output: [[-2, 2]]

(iii) Input: points = [[3, 3], [5, -1], [-2, 4]], k = 2
Output: [[3, 3], [-2, 4]]

```
import heapq
import math

def k_closest(points, k):
    def distance(point):
        return math.sqrt(point[0]**2 + point[1]**2)

    min_heap = []
    for point in points:
        dist = distance(point)
        heapq.heappush(min_heap, (dist, point))

    result = []
    for _ in range(k):
        result.append(heapq.heappop(min_heap)[1])
```

```
    return result
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print(k_closest(points1, k1))
Output: [[-2, 2], [0, 1]]
points2 = [[1, 3], [-2, 2]]
k2 = 1
print(k_closest(points2, k2))
Output: [[-2, 2]]
points3 = [[3, 3], [5, -1], [-2, 4]]
k3 = 2
print(k_closest(points3, k3))
Output: [[3, 3], [-2, 4]]
```

2. Given four lists A, B, C, D of integer values, Write a program to compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

(i) Input: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2]
Output: 2

(ii) Input: A = [0], B = [0], C = [0], D = [0]
Output: 1

```
def four_sum_count(A, B, C, D):
    AB_sum = {}
    for a in A:
        for b in B:
            if a + b in AB_sum:
                AB_sum[a + b] += 1
            else:
                AB_sum[a + b] = 1

    count = 0
    for c in C:
        for d in D:
            if -(c + d) in AB_sum:
```

```
                count += AB_sum[-(c + d)]

    return count
A1 = [1, 2]
B1 = [-2, -1]
C1 = [-1, 2]
D1 = [0, 2]
print(four_sum_count(A1, B1, C1, D1))
```

Output: 2

```
A2 = [0]
B2 = [0]
C2 = [0]
D2 = [0]
print(four_sum_count(A2, B2, C2, D2))
```

Output: 1