# ASSIGNMENT-10

## Median of Medians

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

my name is billllllllaaaaaaaa.py - C:/Users/Admin/AppData/Local/Programs/Python/Python311/my name is billllllllaaaaaaaa.py (3.11.5)

File Edit Format Run Options Window Help

```
def find_median(arr):
    arr.sort()
    return arr[len(arr) // 2]
def partition(arr, pivot):
    low = [x for x in arr if x < pivot]
    high = [x for x in arr if x > pivot]
    pivot_count = arr.count(pivot)
    return low, pivot_count, high
def select(arr, k):
    if len(arr) <= 5:
        arr.sort()
        return arr[k]
    subgroups = [arr[i:i+5] for i in range(0, len(arr), 5)]
    medians = [find_median(subgroup) for subgroup in subgroups]
    pivot = select(medians, len(medians) // 2)
    low, pivot_count, high = partition(arr, pivot)
    if k < len(low):
        return select(low, k)
    elif k < len(low) + pivot_count:
        return pivot
    else:
        return select(high, k - len(low) - pivot_count)
arr1 = [12, 3, 5, 7, 19]
k1 = 2
print(select(arr1, k1 - 1))
arr2 = [12, 3, 5, 7, 4, 19, 26]
k2 = 3
print(select(arr2, k2 - 1))
arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k3 = 6
print(select(arr3, k3 - 1))
```

```
= RESTART: C:/Users/Admin/AppData/Local/Programs/Python/Python311/my name is billllllllaaaaa
aaa.py
5
5
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

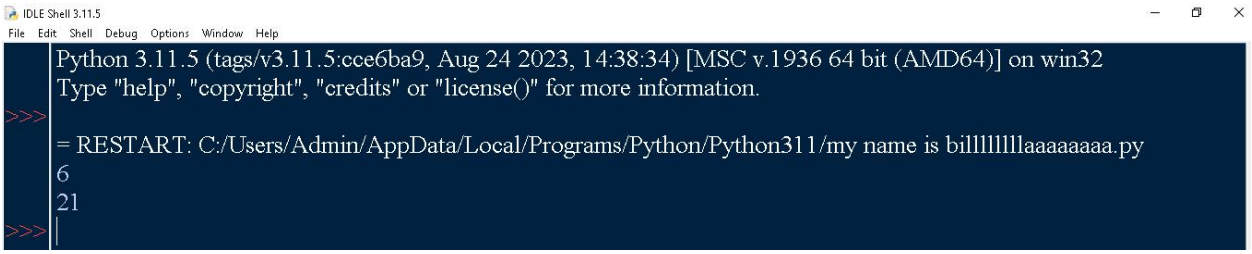
Output: An integer representing the k-th smallest element in the array.

my name is billllllllaaaaaaaa.py - C:/Users/Admin/AppData/Local/Programs/Python/Python311/my name is billllllllaaaaaaaa.py (3.11.5)

File Edit Format Run Options Window Help

```
def find_median(arr):
    arr.sort()
    return arr[len(arr) // 2]
def partition(arr, pivot):
    low = [x for x in arr if x < pivot]
    high = [x for x in arr if x > pivot]
    pivot_count = arr.count(pivot)
    return low, pivot_count, high
def select(arr, k):
    if len(arr) <= 5:
        arr.sort()
        return arr[k]
    subgroups = [arr[i:i+5] for i in range(0, len(arr), 5)]
    medians = [find_median(subgroup) for subgroup in subgroups]
    pivot = select(medians, len(medians) // 2)
    low, pivot_count, high = partition(arr, pivot)
    if k < len(low):
        return select(low, k)
    elif k < len(low) + pivot_count:
        return pivot
    else:
```

```
    else:
        return select(high, k - len(low) - pivot_count)
def median_of_medians(arr, k):
    return select(arr, k - 1)
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
k1 = 6
print(median_of_medians(arr1, k1))
arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]
k2 = 5
print(median_of_medians(arr2, k2))
```




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).

Input : points = [[1,3],[-2,2],[5,8],[0,1]],k=2

Output:[[-2, 2], [0, 1]]

Input: points = [[1, 3], [-2, 2]], k = 1

Output: [[-2, 2]]

Input: points = [[3, 3], [5, -1], [-2, 4]], k = 2

Output: [[3, 3], [-2, 4]]

```
import heapq
def euclidean_distance(point):
  return point[0] ** 2 + point[1] ** 2

def k_closest_points(points, k):
  heap = []
  for point in points:
    distance = euclidean_distance(point)
    if len(heap) < k:
      heapq.heappush(heap, (distance, point))
    else:
      if distance < heap[0][0]:
        heapq.heappop(heap)
        heapq.heappush(heap, (distance, point))
  return [point for _, point in heap]
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
result = k_closest_points(points1.copy(), 2)
print(result)
```

```
= RESTART: C:/Users/Admin/AppData/Local/Programs/Python/Python311/my name is billllllllaaaaaaaa.py
[[0, 1], [1, 3]]
```

4.) 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.

**Input**: A = [1, 2], B = [-2, -1], C = [-1, 2], D = [0, 2]

**Output**: 2

**Input**: A = [0], B = [0], C = [0], D = [0]

**Output**: 1

```
def count_quadruplets(A, B, C, D):
  ab_sums = defaultdict(int)
  for a in A:
    for b in B:
      ab_sums[a + b] += 1
  count = 0
  for c in C:
    for d in D:
      target_sum = -(c + d)
      count += ab_sums.get(target_sum, 0)
  return count
A = [1, 2]
B = [-2, -1]
C = [-1, 2]
D = [0, 2]
result = count_quadruplets(A, B, C, D)
print(result)
A = [0]
B = [0]
C = [0]
D = [0]
```

```
result = count_quadruplets(A, B, C, D)
print(result)
```

```
= RESTART: C:/Users/Admin/AppData/Local/Programs/Python/Python311/my name is billllllllaaaaaaaa.py
2
1
```