**Median of Medians**

**Code with output:**

def partition(arr, low, high, pivot):

pivot\_value = arr[pivot]

arr[pivot], arr[high] = arr[high], arr[pivot]

store\_index = low

for i in range(low, high):

if arr[i] < pivot\_value:

arr[i], arr[store\_index] = arr[store\_index], arr[i]

store\_index += 1

arr[store\_index], arr[high] = arr[high], arr[store\_index]

return store\_index

def median\_of\_medians(arr, low, high):

if high - low + 1 <= 5:

sublist = sorted(arr[low:high+1])

return sublist[len(sublist) // 2]

medians = []

for i in range(low, high + 1, 5):

sublist = arr[i:min(i+5, high+1)]

medians.append(sorted(sublist)[len(sublist) // 2])

return median\_of\_medians(medians, 0, len(medians) - 1)

def select(arr, low, high, k):

if low == high:

return arr[low]

pivot\_index = median\_of\_medians(arr, low, high)

pivot\_index = partition(arr, low, high, arr.index(pivot\_index))

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

return select(arr, low, pivot\_index - 1, k)

else:

return select(arr, pivot\_index + 1, high, k)

def find\_kth\_smallest(arr, k):

return select(arr, 0, len(arr) - 1, k - 1)

# Test cases

arr1 = [12, 3, 5, 7, 19]

k1 = 2

print(f"The {k1}-th smallest element is {find\_kth\_smallest(arr1, k1)}")

arr2 = [12, 3, 5, 7, 4, 19, 26]

k2 = 3

print(f"The {k2}-th smallest element is {find\_kth\_smallest(arr2, k2)}")

arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k3 = 6

print(f"The {k3}-th smallest element is {find\_kth\_smallest(arr3, k3)}")

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.

Code with output:

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[i], arr[store\_index] = arr[store\_index], arr[i]

store\_index += 1

arr[store\_index], arr[high] = arr[high], arr[store\_index]

return store\_index

def mu(arr, low, high):

n = high - low + 1

if n <= 5:

sublist = sorted(arr[low:high+1])

return sublist[n // 2]

medians = []

for i in range(low, high + 1, 5):

sublist = sorted(arr[i:min(i+5, high+1)])

medians.append(sublist[len(sublist) // 2])

return mu(medians, 0, len(medians) - 1)

def select(arr, low, high, k):

if low == high:

return arr[low]

pivot\_value = median\_of\_medians\_util(arr, low, high)

pivot\_index = partition(arr, low, high, arr.index(pivot\_value))

if k == pivot\_index:

return arr[k]

elif k < pivot\_index:

return select(arr, low, pivot\_index - 1, k)

else:

return select(arr, pivot\_index + 1, high, k)

def m(arr, k):

if k < 1 or k > len(arr):

return None # k is out of bounds

return select(arr, 0, len(arr) - 1, k - 1)

arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

k1 = 6

print(f"The {k1}-th smallest element is(m(arr1, k1)}") # Expected Output: 6

arr2 = [23, 17, 31, 44, 55, 21, 20, 18, 19, 27]

k2 = 5

print(f"The {k2}-th smallest element is {m(arr2, k2)}") # Expected Output: 20

3. 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).

Code with output:

def k (points, k):

points.sort(key=lambda p: p[0]\*2 + p[1]\*2)

return points[:k]

points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]

k1 = 2

print(k (points1, k1)) # Output: [[-2, 2], [0, 1]]

points2 = [[1, 3], [-2, 2]]

k2 = 1

print(k(points2, k2)) # Output: [[-2, 2]]

points3 = [[3, 3], [5, -1], [-2, 4]]

k3 = 2

print(k (points3, k3)) # Output: [[3, 3], [-2, 4]]

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.

Code with output:

def s(A, B, C, D):

from collections import defaultdict

sum\_ab = defaultdict(int)

for a in A:

for b in B:

sum\_ab[a + b] += 1

count = 0

for c in C:

for d in D:

target = -(c + d)

if target in sum\_ab:

count += sum\_ab[target]

return count

A1 = [1, 2]

B1 = [-2, -1]

C1 = [-1, 2]

D1 = [0, 2]

print(sA1, B1, C1, D1)) # Output: 2

A2 = [0]

B2 = [0]

C2 = [0]

D2 = [0]

print(s(A2, B2, C2, D2)) # Output: 1