TSP,.....

```
import numpy as np
from scipy.spatial.distance import cdist

def branch_and_bound_tsp(dist_matrix):
    n = len(dist_matrix)
    mask = np.zeros(n, dtype=bool)
    mask[0] = True
    L = [(0, [0], mask, 0, np.inf)]
    best_tour = None
    while L:
        _, tour, mask, lb, ub = L.pop()
        if np.all(mask):
            if best_tour is None or lb < best_tour[1]:
                best_tour = (tour, lb)
        elif lb < best_tour[1]:
            for i in range(n):
                if not mask[i]:
                    new_tour = tour + [i]
                    new_mask = mask.copy()
                    new_mask[i] = True
                    new_lb = lb + dist_matrix[tour[-1], i]
                    new_ub = new_lb + np.min(cdist(dist_matrix[new_mask],
dist_matrix[~new_mask], metric='euclidean'))
                    if new_ub < best_tour[1]:
                        L.append((new_ub, new_tour, new_mask, new_lb, new_ub))
    return best_tour

def nearest_neighbor_tsp(dist_matrix):
    n = len(dist_matrix)
    mask = np.zeros(n, dtype=bool)
    mask[0] = True
    tour = [0]
    while len(tour) < n:
        i = np.argmin(np.where(~mask, dist_matrix[tour[-1]], np.inf))
        mask[i] = True
        tour.append(i)
    tour.append(0)
    return tour

# Example usage and comparison

dist_matrix = np.array([
    [0, 2, 4, 3, 5],
    [2, 0, 6, 7, 3],
    [4, 6, 0, 5, 8],
    [3, 7, 5, 0, 4],
    [5, 3, 8, 4, 0]])
```

```
opt_tour, opt_length = branch_and_bound_tsp(dist_matrix)
approx_tour = nearest_neighbor_tsp(dist_matrix)
approx_length = np.sum(dist_matrix[approx_tour[:-1], approx_tour[1:]])
approx_ratio = approx_length / opt_length

print(f"Optimal tour: {opt_tour}, length={opt_length}")
print(f"Approximation: {approx_tour}, length={approx_length}, ratio={approx_ratio:.2f}")
```

LAST

```
import random

def randomized_median_of_medians(arr, k):
    if len(arr) == 1:
        return arr[0]

    groups = [arr[i:i + 5] for i in range(0, len(arr), 5)]
    medians = [sorted(group)[len(group) // 2] for group in groups]
    pivot = randomized_median_of_medians(medians, len(medians) // 2)
    left = [x for x in arr if x < pivot]
    right = [x for x in arr if x > pivot]
    pivot_count = len(arr) - len(left) - len(right)

    if k <= len(left):
        return randomized_median_of_medians(left, k)
    elif k > len(left) + pivot_count:
        return randomized_median_of_medians(right, k - len(left) - pivot_count)
    else:
        return pivot
```