CSE 106: Offline Assignment

Emergency Communication Network

(Using Divide & Conquer)

Bangladesh University of Engineering and Technology, BUET
Department of CSE

1. Introduction

This assignment introduces you to the **Divide and Conquer** paradigm through a practical problem: building a minimum-cost communication network connecting cities.

You will:

- Spatially partition cities using a divide and conquer strategy.
- Connect cities within small groups (base case).
- Merge subproblems by connecting the two halves.
- Analyze the time complexity of your approach.

2. Problem Description

A country needs to build an emergency communication network connecting N cities. Each city has a 2D coordinate (x, y), and the cost of connecting two cities is simply the **Euclidean distance** between them:

$$Cost(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Your goal is to connect **all** N cities into a single connected network while keeping the **total cost** (sum of all edge costs) small, using a **Divide and Conquer** approach.

Constraints

- The entire network must be **connected** (every city can reach every other city).
- The total cost is kept small using a divide and conquer approach.
- $4 \leq N \leq 10^5$

Your algorithm must run in strictly better than $O(N^2)$ time.

3. Input Specification

```
N M
<ID> <x> <y>
<ID> <x> <y>
...
```

3.1 Parameter Explanation

1. N — Number of Cities

- Total number of cities in the network.
- Cities are labeled from 1 to N.

2. M — Base Case Threshold

- When a subproblem has $\leq M$ cities, solve it directly (base case).
- In the base case, connect the cities using the cheapest edges that form a connected subgraph.
- Typical values: $2 \leq M \leq 50$.

3. City Lines

Each subsequent line contains:

- **City ID** — integer from 1 to N
- x **coordinate** — integer or decimal
- y **coordinate** — integer or decimal

4. Algorithmic Approach

Your solution must follow a **Divide and Conquer** strategy:

4.1 Step 1: Divide

- Sort or partition the cities by their x-coordinate.
- Split them into two roughly equal halves at the **median** x-coordinate.

4.2 Step 2: Conquer (Base Case)

- When a group has $\leq M$ cities, connect them directly.
- Since M is small, you can consider all $O(M^2)$ pairs and pick the cheapest edges that connect all cities in the group.
- A simple approach: sort edges by cost and greedily add them if they connect two previously disconnected components (think of this as checking connectivity — you can use a simple visited/BFS/DFS check).

4.3 Step 3: Merge (Combine)

- After solving the left and right halves independently, you have two separate connected networks.
- To merge them into one, you need to add at least one edge connecting a city from the left half to a city from the right half.
- You do **not** need to check all pairs across the two halves. Since the cities are sorted by x-coordinate, the cheapest cross-edge will almost always involve **boundary cities** — the cities closest to the dividing line.
- Specifically, compare only the **rightmost** M **cities** of the left half with the **leftmost** M **cities** of the right half (or fewer, if a half has fewer than M cities). This gives at most $O(M^2)$ comparisons per merge, which is $O(1)$ since M is a small constant.
- Pick the cheapest edge among these boundary pairs and add it.

Since each half is already connected internally, adding even one cross-edge makes the entire network connected. By comparing boundary cities (those near the dividing x-coordinate), we find a good cross-edge efficiently without scanning all $O(N^2)$ pairs.

Focus on understanding the following:

- Your **recurrence relation**.
- Your **overall time complexity** and how you derived it.
- How your **merge step** works and why it produces a valid connected network.

5. Output Format

```
Total Cost: <value>
Edges:
u1 v1
u2 v2
...
```

- `Total Cost` is the sum of Euclidean distances of all edges, rounded to 2 decimal places.
- Each edge line lists the IDs of the two connected cities.
- Edge order does not matter.
- The output must represent a **connected** network.

6. Sample Input and Output

6.1 Sample 1: Simple Line of Cities

Input

```
5 3
1 0 0
2 1 0
3 2 0
4 3 0
5 4 0
```

Explanation

Five cities along the x-axis. Optimal solution connects each consecutive pair.

Output

```
Total Cost: 4.00
Edges:
1 2
2 3
3 4
4 5
```

6.2 Sample 2: Square of Cities

Input

```
4 4
1 0 0
2 0 1
3 1 1
4 1 0
```

Explanation

Four cities at corners of a unit square. Since $M = 4$, all cities fit in the base case. The cheapest connected network uses 3 edges of length 1 each (a spanning tree of the square).

Output

```
Total Cost: 3.00
Edges:
1 2
2 3
3 4
```

6.3 Sample 3: Two Clusters Far Apart

Input

```
6 3
1 0 0
2 1 0
3 2 0
4 100 0
5 101 0
6 102 0
```

Explanation

The divide step splits into two clusters: $\{1, 2, 3\}$ and $\{4, 5, 6\}$. Each cluster is solved in the base case. The merge step must connect the closest boundary nodes (city 3 and city 4).

Output

```
Total Cost: 102.00
Edges:
1 2
2 3
4 5
5 6
3 4
```

6.4 Sample 4: 2D Spread

Input

```
6 3
1 0 0
2 3 4
3 1 2
```

```
4 6 1
5 8 3
6 7 5
```

Explanation

Cities are spread in 2D. Sorted by x: $\{1, 3, 2\}$ (left) and $\{4, 6, 5\}$ (right). Each half is connected in the base case, then merged by the cheapest cross edge.

Output (One Valid Solution)

```
Total Cost: 14.25
Edges:
1 3
3 2
6 5
4 5
2 6
```

Important Notes

- Outputs shown are only **one valid solution**. Multiple valid solutions may exist.
- Edge order does not matter.
- Your solution will be evaluated primarily on:
  - **Connectivity** — all cities must be reachable.
  - **Use of Divide & Conquer** — brute force $O(N^2)$ solutions receive no credit.
  - **Code clarity** — well-structured code, and you should be able to explain the divide, conquer, and merge steps.

7. Hints

1. **Merge is simple**: You only need to compare boundary cities (last M from left, first M from right). No need to check all cross-pairs.
2. **Base Case**: Keep it simple. For small groups ($\leq M$ cities), brute force is fine — try all pairs, sort by distance, and greedily pick edges that connect new components.
3. **Connectivity Check**: To check if adding an edge connects two components, you can use BFS/DFS from any node.

4. **Don't Overthink**: You are **not** expected to find the theoretically optimal minimum spanning tree. A good divide and conquer solution that produces a reasonably low-cost connected network is sufficient.

8. Evaluation Criteria

Criteria	**Marks**
Correct use of Divide & Conquer	40
Correctness (valid connected network)	20
Time Complexity Analysis	15
Explanation of Merge Step	15
Code Clarity & Efficiency	10
Total	**100**

Note: Solutions that do not use Divide & Conquer or run in $O(N^2)$ time will not receive credit for the algorithmic component.