Exercise Sheet — 2D Maxima with Plane Sweep Algorithm

Learning Objective

Implement and analyze an O(n log n) plane-sweep algorithm to compute all maximal points in a 2D set. A point p dominates q if xp ≥ xq and yp ≥ yq with at least one strict; a point is maximal if none dominates it.

Part A — Manual Example

Given P = {(2,5), (4,1), (1,4), (3,3), (5,2), (6,6), (7,4), (5,5)}:

- Plot the points on the Cartesian plane.
- By inspection, mark the maximal points.
- Formally verify your answer using the dominance definition.

Part B — Algorithmic Design

Describe the sweep-line strategy and write pseudocode.

```
Algorithm Maxima2D_Sweep(P):
    sort P in decreasing order of x
    maxY ← -∞
    maxima ← ∅
    for each (x, y) in P:
        if y ≥ maxY:
            maxima ← maxima ∪ {(x, y)}
            maxY ← y
    return maxima
```

Explain why sorting by descending x ensures correctness, and analyze time/space complexity.

Part C — Implementation (Python/C++/Java)

Write a program that reads n followed by n lines of x y pairs and prints the maxima set.

```
Input
8
2 5
4 1
1 4
3 3
5 2
6 6
7 4
5 5

Output
Maximal points: (7,4), (6,6), (5,5)
```

Part D — Empirical Comparison

Implement a brute-force $O(n^2)$ verifier. Generate random datasets for $n \in$ {10^3, 10■, 10■}. Measure runtime of sweep vs brute force, plot runtime (log-scale) vs n, and discuss empirical trends.

Part E — Discussion Questions

- Why is the sweep algorithm $O(n \log n)$?
- How would the answer change if dominance required strictness in both coordinates?
- What breaks when extending to 3D?
- Geometric meaning of the maxima frontier and its relation to Pareto optimality.

Optional Extension

Implement an online variant using a balanced BST/Fenwick tree for y-values to support insertions and dominance queries.

Starter Code Snippets (Python)

```
# Sweep-line maxima (Python)
def maxima_sweep(points):
    pts = sorted(points, key=lambda p: (-p[0], -p[1]))  # sort by x desc, y desc
    maxima = []
    maxY = float('-inf')
    for x, y in pts:
        if y >= maxY:
            maxima.append((x, y))
            maxY = y
    return maxima

# Brute-force verifier
def dominates(a, b):
    return (a[0] >= b[0]) and (a[1] >= b[1]) and (a != b)

def maxima_bruteforce(points):
    res = []
    for p in points:
        if not any(dominates(q, p) for q in points):
            res.append(p)
    return res
```