Analysis of 2D Maxima Algorithm

Class 3

Analysis of 2D Maxima

- In the previous lecture we developed the mathematical model of the problem.
- The mathematical description of the algorithm was:
  - Input: given a set of points $P = \{ p_1, p_2, p_3, \ldots, p_n \}$ in 2D space.
  - Output: the set of maximal points of P,
  - i.e., those points p_i such that p_i is not dominated by any other point of P.
- Now we will convert this mathematical model into the algorithm.

Brute-Force Algorithm:

- No intension to develop the optimal or efficient algorithm yet.
- Let $P = \{ p_1, \ p_2, \ p_3, \ \ldots, \ p_n \}$ be the initial set of points.
- For each point p_i, test it against all other points p_j.
- If p_i is not dominated by any other point, then output it.

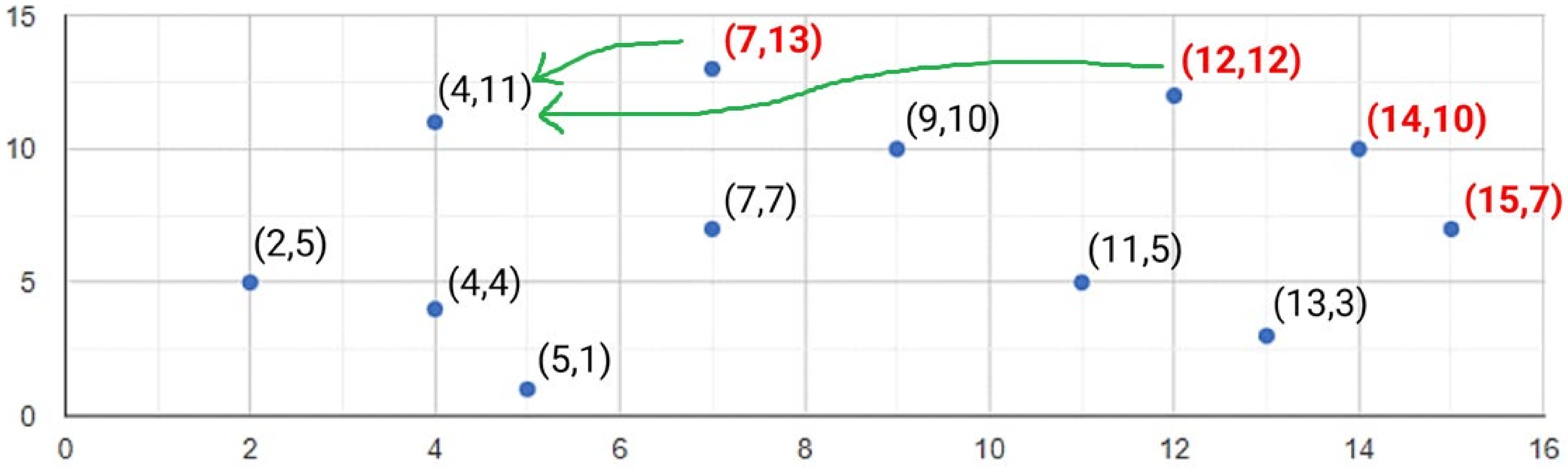
Pseudocode

- It is the pseudo language.
- Algorithm steps in plain English language.
- No technical or programming language concepts.
- Its main goal is to represent the algorithm in readable form for everyone having no programming background.
- Not compiled or executed on computer system.

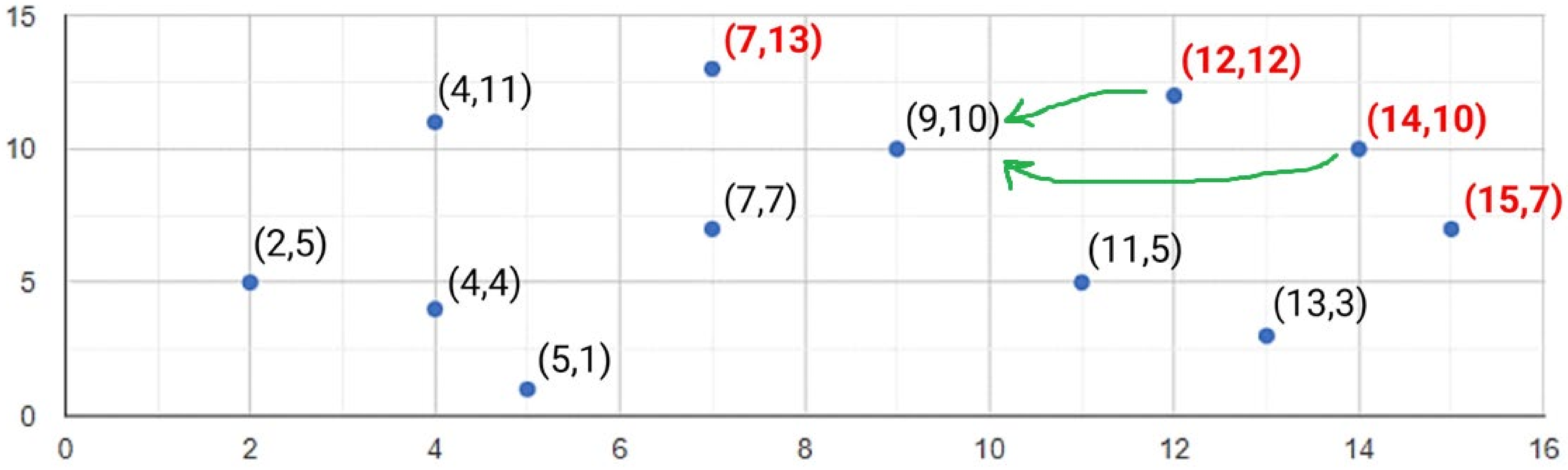
```
MAXIMA (int n, Point P[1 … n])
For i ← 1 to n
Do maximal ← true
     For j ← 1 to n
     Do
          If (i ≠ j) and (P[i].x ≤ P[j].x) and (P[i].y ≤ P[j].y)
             Then maximal ← false
             Break
     If (maximal = true)
          Then output P[i].x and P[i].y
```

- This algorithm will take 2 inputs:
  - Array P containing the points of the 2D space
  - Length of the array n
- *Maximal* is the boolean variable to check if a point is maximal or not

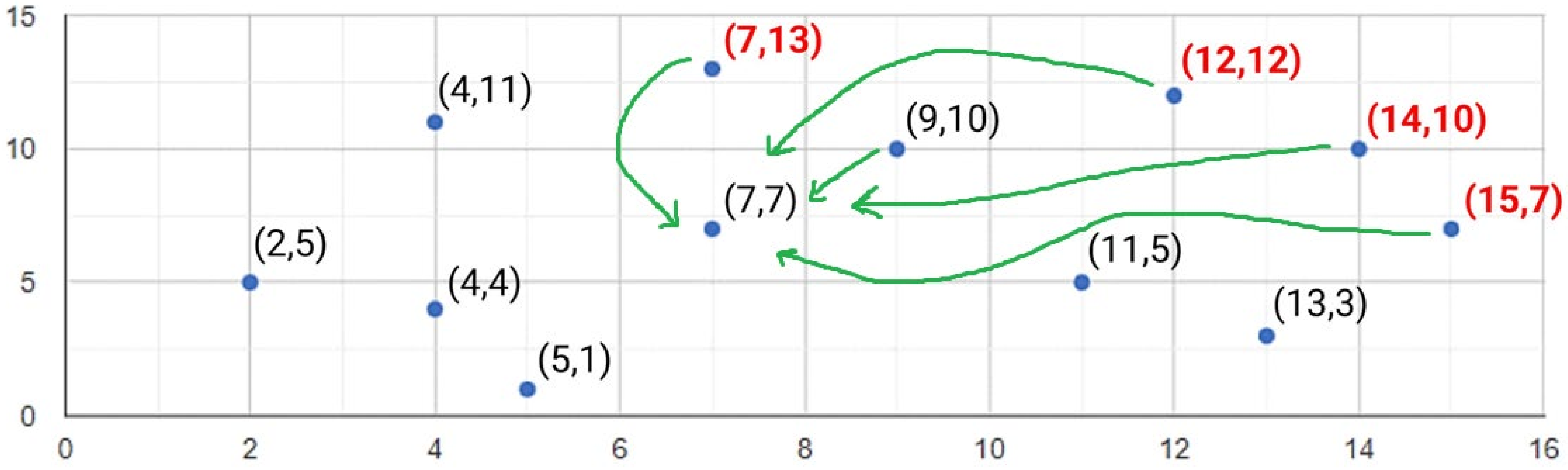
Points that dominate (4, 11)



Also compare points (12,12) and (14,10) with the point (9,10)



Many points dominate the point (7,7)



- So, after the execution of the algorithm, we got these 4 point as maximal points:
  - (7, 13)
  - (12, 12)
  - (14, 10)
  - (15, 7)
- First value is the car speed
- Second value is the money saved
- So, we can now easily decide which car is best suited and according to our requirements.

Analysis of this Algorithm

- The last stage of the process is the Analysis of our algorithm:
  - Runtime Analysis
  - Space (Memory) Utilization of the Algorithm
- But we will only consider the execution time and ignore the memory utilization.
- As we have *Random Access Machine* having infinite space, so we don’t have issue of the memory utilization.
- So, we will only consider the running time for now.

Running Time Analysis

- The main purpose of our mathematical analysis will be measuring the execution time.
- We can run this algorithm using C language then find the CPU execution time using Debugging tools.
- But we will not find the running time in this way.
- We have to use mathematical tools and estimate the running time mathematically.

Analysis of 2D Maxima

- To measure the running time of the brute-force 2D maxima algorithm, we could:
- Count the number of steps of the pseudo code that are executed.
- Or count the number of times an element of P is accessed.
- Or the number of comparisons that are performed.

- The running time depends upon the input size, e.g., n
- Different inputs of the same size may result in different running time.
- For example, breaking out of the inner loop in the brute-force algorithm depends not only on the input size of P but also the structure of the input.

- So, we will use some criteria to calculate the execution time without doing in depth of the type and size of the data.
- Worst Case Time
- Average Case Time
- Best Case Time

Worst Case Time

- Worst-case time is the maximum running time over all (legal) inputs of size n.
- It is mathematically defined as:
  - Let I denote an input instance (set of values given to the algorithm),
  - let $|I|$ denote its length, and
  - let $T(I)$ denote the running time of the algorithm on input I.
  - Then T is:

$$T_{worst(n)} = \max_{|I|=n} T(I)$$

Average (Expected) Case Time

- Average-case time is the average running time over all inputs of size n.
- It is mathematically defined as:
  - Let $p(I)$ denote the probability of seeing this input.
  - The average-case time is the weighted sum of running times with weights being the probabilities:

$$T_{avg}(n) = \sum_{|I|=n} p(I)T(I)$$

Which one we use?

- We will almost always work with worst-case time.
- Average-case time is more difficult to compute; it is difficult to specify probability distribution on inputs.
- Difficult not based on formula but difficult based on randomness of input data.
- Worst-case time will specify an upper limit on the running time.
- Whatever the input will be, the maximum running time will not exceed the upper limit.

Worst Case Time of brute-force maxima Algorithm

- Input size is n
- Particularly for this algorithm,
- we will count number of times that any element of P is accessed.

```
MAXIMA (int n, Point P[1 … n])
For i ← 1 to n                                     n times
Do maximal ← true
     For j ← 1 to n                                n times
     Do
         If (i ≠ j) and (P[i].x ≤ P[j].x) and (P[i].y ≤ P[j].y) 4 memorry accesses
             Then maximal ← false
             Break
     If (maximal = true)
         Then output P[i].x,P[i].y                 2 memorry accesses
```

- The outer loop runs n times
- For each i iteration, the inner loop runs n time.
- P is accessed four times in the *if* statement.
- The output statement accesses P two times.
- In the worst case, every point is maximal, so every point is output.
  - e.g., all the points are same.