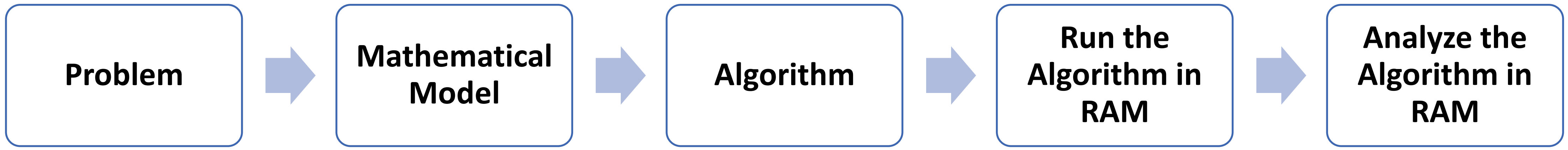
Model of Computation
and
2D Maxima

Class 2

Random Access Machine (RAM)

- A RAM is an idealized machine with an infinitely large random-access memory.
- It is our mathematical model of computation.
- Instructions are executed one-by-one (there is no parallelism).
- One instruction per clock cycle.
- Random Access vs Sequential Access.

- Two operands with basic arithmetic operations (e.g., $a + b$).
- Each instruction involves performing some basic operation on two values in the machine's memory.
- We don’t discuss the hardware, OS or the brand of the computer but just consider a general computer.
- The algorithms will use some finite memory at the end when they are translated to computer programs.
- But for now, we will consider infinite memory allocated to the algorithm.


Problem
Mathematical Model
Algorithm
Run the Algorithm in RAM
Analyze the Algorithm in RAM

Basic RAM Operations

- Assigning a value to a variable.
- Computing any basic arithmetic operation (+, -, x and division) on integer values of any size.
- Performing any comparison or Boolean operations.
- Accessing an element of an array (e.g., A[10]).
- It is like a normal Computer so nothing special...!!!
- We will consider integers only because of simplicity.

RAM Model

- We assume that each basic operation takes the same constant time to execute.
- The RAM model does a good job of describing the computational power of most modern (non-parallel) machines.
- For example, Instructions of Assembly language take CPU clock cycles.
- These operation (Arithmetic, Comparison, Assignment, Boolean, Indexing, etc.) take constant time in Random Access Machine.

- These operations are involved in our Algorithm Analysis phase.
- We don’t say that the algorithm execution takes same time based on Input and Output data. So, these two parameters are not included in analysis of the algorithm.
- Basic operations in C++ that are assumed to take up same amount of CPU time.

- $x = y;$
- $z = a + b;$
- $z = a - b;$
- $z = a * b;$
- $z = a / b;$
- $z = w[10];$
- $x <= y;$

- For example, $\boldsymbol{x = y}$ takes 1 microsecond then $\boldsymbol{z = a + b}$ will take 2 microseconds because it involved addition then assignment operation.
- But in future we will not use the unit seconds or microseconds.
- We will simply say that the operation will take a single unit time.

Example: 2D Maxima

- Suppose you want to buy a car.
- You want to pick the fastest car.
- But fast cars are expensive; you want the cheapest.
- You cannot decide which is more important: speed or price.
- Definitely, we do NOT want a car if there is another that is both faster and cheaper.

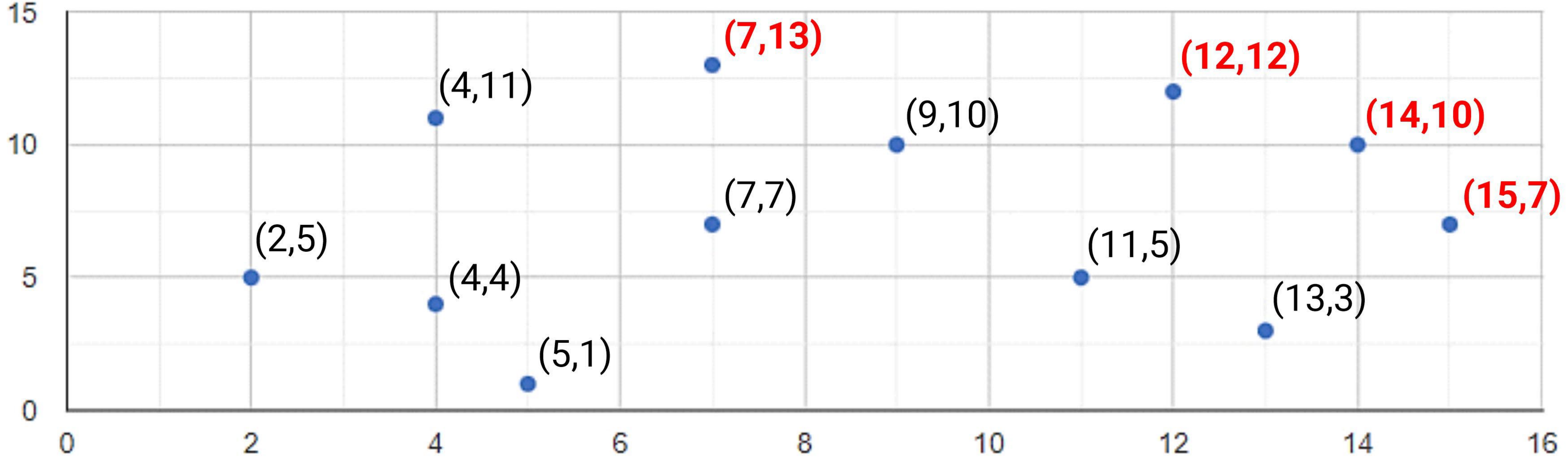
- We say that the fast, cheap car dominates the slow, expensive car relative to your selection criteria.
- So, given a collection of cars, we want to list those cars that are not dominated by any other.
- We will design an algorithm to find the list of the cars according to my requirements.
- We will make a mathematical model of the problem so that we can solve it and analyse it.

- Here is how we might model this as a formal problem mathematically:
- Let a point p in 2-dimensional space be given by its integer coordinates, $p = (p.x,\ p.y)$
- A point p is said to be dominated by point q if $p.x \ \leq \ q.x$ AND $p.y \ \leq \ q.y$
- Given a set of n points $P \ = \ \{\, p_1,\ p_2,\ p_3,\ \ldots,\ p_n \,\}$ in 2D space, a point is said to be **maximal** if it is not dominated by any other point in P.

- The car selection problem can be modelled this way:
- For each car we associate (x, y) pair where:
  - x is the speed of the car.
  - y is the negation of the price.

- Why we negate the price?
- Understand it in such a way that it is the money that will be left in your pocket after buying the car.
- So, you want to maximize the money left in your pocket after buying the car.
  - High y value means a cheap car and low y means expensive car.
  - Think of y as the money left in your pocket after you have paid for the car.
  - Maximal points correspond to the fastest and cheapest cars.

- The mathematical description of the algorithm will be:
  - 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.


15
10
5
0
0
2
4
6
8
10
12
14
16
(7,13)
(12,12)
(4,11)
(9,10)
(14,10)
(7,7)
(15,7)
(2,5)
(11,5)
(4,4)
(13,3)
(5,1)

- We will not directly develop the program to solve the problem.
- Our description of the problem is at a mathematical level.
- We have intentionally not discussed how the points are represented.
- We are not discussing that how to store the points, in array, linked list or whatever else.
- We have not discussed any input or output formats.
- We have avoided programming and other software issues.

- There could be a lot of points so we will design the algorithm to perform the task on the data of any variable length.
- Two factors will be analysed for our designed algorithm:
  - Space Resources
  - Time Resources
- If we have 1D points (a line) then how to find the maximal point?
  - We just sort the points to descending order and pick the lowest point.