COSC6320 Programming Assignment 2

Out: Oct 17, Thursday
Due: Nov. 1, Friday, 11.59pm
Submissions will not be accepted after the deadline.

Instructions:
Read the Academic Dishonesty policy posted in Blackboard which is repeated here: All submitted work should be your own. Copying or using other people's work (including from the Web) will result in $-MAX$ points, where MAX is the maximum possible number of points for that assignment. Repeat offense will result in getting a failure grade in the course and reporting to the Chair. If you have any questions regarding any assignment, please contact me or the TA (earlier the better).

By submitting this assignment, you affirm that you followed the Academic Dishonest Policy.

Justify your answers. Show appropriate work.
Please write legibly and clearly (if you are writing by hand).

Please start to work on the problem early, as it might generally require some time.

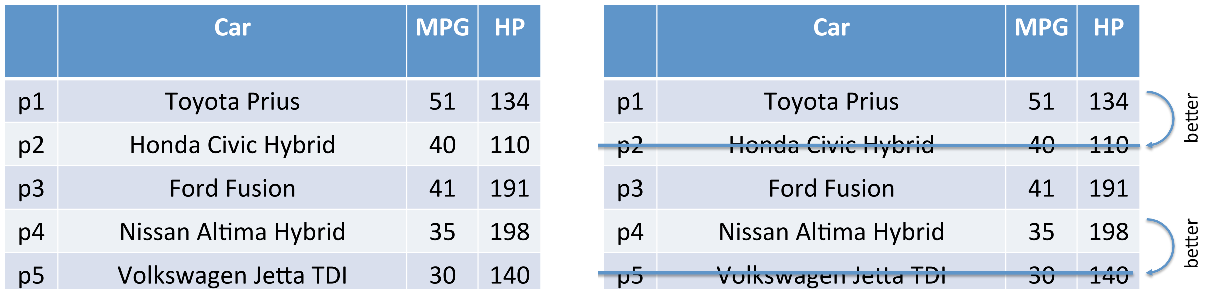
1 Problem Motivation

Your friend is looking for a car that has a high miles per gallon (MPG) and a high horse power (HP). There are so many cars in the market, but some of them are obviously "uninteresting." For example, consider five cars[1] in Figure 1a. The second car (Honda Civic Hybrid) is worse than the first car (Toyota Prius) in terms of both MPG and HP (since the numbers are smaller). So, why should your friend consider the second car? In this case, your friend should remove the second car from the list (see Figure 1b). Similarly, the fifth car is worse than the fourth car; so it should be removed as well. Now, if you look at the rest three cars, none of them are better than the other in terms of both HP and MPG. For example, the first car is better than the third car in terms of MPG but worse in terms of HP. We call these cars, *interesting cars*. Your friend wants to find all interesting cars from his/her list but there are too many of them! **Your task:** Write a program to help your friend find a list of all interesting cars as fast as possible.

2 The Problem

Mathematically, you can represent each car as a 2-dimensional point $p \in \mathbb{R}^2$; this means that each point p has two coordinates, $p[0]$ and $p[1]$, where the values in each coordinate is a non-negative real.

[1]Disclaimer: The numbers in the figures are made up. They do not reflect the real qualities of the mentioned cars.



	Car	MPG	HP
p1	Toyota Prius	51	134
p2	Honda Civic Hybrid	40	110
p3	Ford Fusion	41	191
p4	Nissan Altima Hybrid	35	198
p5	Volkswagen Jetta TDI	30	140

(a) All cars

	Car	MPG	HP
p1	Toyota Prius	51	134
~~p2~~	~~Honda Civic Hybrid~~	~~40~~	~~110~~
p3	Ford Fusion	41	191
p4	Nissan Altima Hybrid	35	198
~~p5~~	~~Volkswagen Jetta TDI~~	~~30~~	~~140~~

better
better

(b) Interesting cars

Figure 1: Car example.

For example, five cars in Figure 1a can be represented by five points: $p_1 = (51, 134)$, $p_2 = (40, 110)$, $p_3 = (41, 191)$, $p_4 = (35, 198)$, and $p_5 = (30, 140)$. In fact, we will consider a more general situation where there are d quality measures of a car to consider, for any integer $d \geq 1$; thus, cars are represented as d-dimensional points (in $\mathbb{R}^d$). The example above is the case where $d = 2$.

Problem. For any two distinct points p_i and $p_{i'}$, we say that $p_{i'}$ is *worse* than p_i if, for all $0 \leq j \leq d-1$, $p_i[j] \geq p_{i'}[j]$, and there exists some $0 \leq j' \leq d-1$ such that $p_i[j'] > p_{i'}[j']$. We say that a point p_i is *interesting* if it is not worse than any other point. Our problem is to find *the number of interesting points.*

Input. You are given an integer $n \leq 1{,}000{,}000$ representing the number of points (cars), integer $1 \leq d \leq 10$, and n points $p_0, p_1, \ldots, p_{n-1} \in \mathbb{R}^d$.

The input function is already provided in the file example.cpp. You can simply use ReadInput(n, d, points) or ReadInput("1.in", n, d, points) (use the first function if you want to read from cin and use the second one if you want to read from file "1.in"). You can then use point[i][j] to refer to the j^{th} dimension of point p_i. You can use all these without understanding the input file. However, if you want to write your own input function (especially, if you are not coding in C++), then the input format is as follows.

Input format: The first line of the input consists of the values of n and d. In the next n lines, you will find d real numbers in each line, indicating values in d coordinates of the points. For example, for the data in Figure 1a, you will get the following input.

```
5 2
51 134
40 110
41 191
35 198
30 140
```

This input can be found in file 1.in. There are also other test files and answer files given, as mentioned in Section 3.

Output. For the purpose of evaluation, your algorithm has to output the correct number of interesting points.

	1.in	2.in	3.in	4.in	5.in	6.in	7.in	8.in
SimpleAlgo1	0.000074	242.745	212.673	246.747	542.595	too long	too long	too long
SimpleAlgo2	0.00004	74.9864	74.9502	75.207	100.039	too long	too long	393.752
New Algorithm (not given)	0.000068	0.086386	0.395291	0.075483	2.80277	2.04786	8.53655	6.50607

Table 1: Time of algorithms in seconds. Note that the running time of algorithms varies from machine to machine, but their relative time usually does not change.

3 Guideline and Evaluation

You will have to implement algorithms for the problem above. You have to turn in the following.

- A source file where you implement your algorithms and measure their running times. You are allowed to implement up to two (different) algorithms. *It is very important that your algorithm(s) is (are) correct.* We will test your algorithms against many test sets that we have.

- A document describing your algorithm(s). Your document should present correctness and running time analysis of your algorithms. It should also present the running time of our algorithms when you experimented on your computers.

We will measure the running time of your algorithm(s). See example.cpp for how to do so. Your algorithm(s) have to be better than two simple algorithms provided in example.cpp. Also, we will compare how fast your algorithms run on different data sets of different sizes, i.e., how well it scales to large sizes. Your score will (partly) depend on how good (i.e., fast) your algorithms are and how well you can explain your algorithms in your documents. We encourage you to try all different techniques you learned in the class. Techniques that are most relevant for this problem are: *divide and conquer, sorting, randomization, and hashing.*

To show you how much we expect your algorithm to be better than the algorithms we gave, we have implemented another algorithm. Its running time compared to the given algorithms is shown in Table 1. This new algorithm is still not the best algorithm. So, it is still possible that your algorithms are better that this!

Accompanying Files. To help you get started, we have provided the following files.

- **example.cpp**: This file provides input functions and a way to measure the running time of your algorithm.

- **1.in, 2.in, ..., 8.in**: These files contain sample inputs. You should try to be much better than our algorithms at least on these inputs. (We will use more input files to test your programs on the evaluation date. So, be prepared.)

- **1.out, 2.out, ..., 8.out**: These files contain the solution for the input files provided above.

Although your program can be in any one of these programming languages — C/C++/Python/Java — we prefer it is in C++ (for fair comparison).[2] If that's not the case, then you should write your own input/output and running time calculation as shown in the example.cpp. Your program should be well annotated (these carry points as well).

Submission:

[2]We will also compare how fast your algorithm scales to large sizes; so even if you don't use C++, the relative scaling of your algorithm will matter and we will take that into account besides the absolute run times in our testing.

You should do your work in github (github.com) and allow access to your private repository by the TA by the deadline.