Objective	The objective is to take a group of N people who have stated their position on T topics, and then group them into (N/2) pairs so that each pair has at least one position on which they disagree, so they can argue about it. Let's define "disagreement" as a difference of 4 or more on a seven-point scale.
Some difficulty	the problem here is very different from standard matching algorithms, which typically exploit similarity. Instead, here we are exploiting difference. Take, for example, someone who has weak preference on most issues (3,4, or 5 on a scale from 1-7). The are easy to match in a similarity space, because they're in the middle of everything. But that same property makes them hard to match in a difference space.
One solution (R code is attached - if you want to see it you're welcome to look)	do this by creating a three-dimensional matrix of size N x N x T. Each cell is binary [0 or 1] - does this pairing disagree on this topic? Let's call each sufficiently disagreeing person-person-topic combination a "branch". The hard cases will have few branches, the easy cases will have many. The matrix is essential, and that structure makes the rest of the work very easy. You then need to follow a simple loop.
	{
	1. grab the person with the fewest remaining eligible pairs (lowest branch count).
	2. among their eligible partners, grab the person who themselves has the lowest branch count.
	3. among the topics on which they disagree, assign them the topic on which their disagreement is strongest.
	4. update the matrix by pruning branches that are now ineligible by this most recent pairing.
	}
	"matcher_2.R" is a two-round design - everyone needs two different partners, one in each round. This has the same underlying mechanism as what I described, but there's extra checks to make sure people aren't assigned the same partner in each round, or have both of their pairings in the same round, or have the same topic in both rounds. These checks are all handled in step 4 above, by pruning branches after each pairing is made.
Your Solution	Please describe your algorithm and implement it in javascript
Your Tests	What ways can you verify that your algorithm worked