In fact we want to match jugglers to circuits such that no juggler could switch to a circuit that they prefer more than the one they are assigned to and be a better fit for that circuit than one of the other jugglers assigned to it.

Solution 1:

1. Figure out #J’s per C
   1. 12 / 3 = 4
2. Find Js’ ranks for each preferred C
3. Sort all firsts among Js
4. Start with highest ranks and fill until a C is full
5. Remove all ranks for that circuit
6. Collapse all preference columns
7. Go back to 3

Solution 1 big-theta:

1. 1st Calc: J \* X
2. 1st Sort: JlogJ ?
   1. Whatever the browser’s native search
3. Other sorts: ½\*C\*JlogJ
   1. J Gets smaller after each assignment
4. Circuit close: J\*X\*1/3X = 1/3JX^2
   1. There are J\*X juggler-circuit matches that are all removed at closure
   2. Closing also tells the juggler to remove a pref; uses linear loop, which gets smaller each time, so avgs to 1/3X
5. Final: JX + JlogJ + ½CJlogJ + 1/3JX^2 ~ O(CJlogJ + JX^2)

Solution 1b big-theta: (modified the sort in the middle to use binary search)

1. 1st Calc: J \* X
2. 1st Sort: JlogJ ?
   1. Whatever the browser’s native search
3. Other sorts: ½\*2\*C\*logJ = ClogJ
   1. Gets smaller, so ½,
   2. Doesn’t resort entire list, just removes and reinserts with new rank
   3. Binary search is logJ; 2 of them (remove and place)
4. Circuit close: J\*X\*1/3X = 1/3JX^2
   1. There are J\*X juggler-circuit matches that are all removed at closure
   2. Closing also tells the juggler to remove a pref; uses linear loop, which gets smaller each time, so avgs to 1/3X
5. Final: JX + JlogJ + ClogJ + 1/3JX^2 ~ O(ClogJ + JX^2)

Solution 1 pseudo code:

JModel:

* H, E, P
* Id
* PrefsToss[]
  + C#
  + Rank
  + (C points to this)
* PrefsKeep[]
  + C#
  + Rank

CModel:

* H, E, P
* Id
* NumJs
* JRanks[]
  + Pointer to the J’s rank
* assignedJs[]
  + Pointer to J

Algorithm

1. Sort J’s by first avail pref
   1. If no pref then put at end
2. Iterate through Js, Fill C’s
3. If a C gets full
   1. Remove all its J ranks
   2. Go back to 1

Thoughts:

* Ideal world: all J’s get their #1 C; but if start here then there will be an uneven distro
* There might also be a pool of those that don’t fit into any of their preferred Circuits
  + Only can happen if number of circuits, C, is greater than the number of preferences per juggler, X
  + This is not the case in the provided sample; 3 circuits, 3 prefs per juggler. There will be no pool
* Can we somehow modify the rank to reflect preference?
  + Can calculate all of the adjusted ranks and put them all into a single pool, sorted by rank.
  + Assignments will happen in descending order, if a rank’s juggler or circuit has already been assigned then toss and move on
* Perhaps have preference be calculated in with a base according to the largest dot product possible? So rank in 1st col will always outrank 2nd col, etc
  + No, because ranks are compared 1:1 once preference is thrown out
  + Preference only means which C is filled first
* Can we count the number of preferred Js for each circuit and use that to somehow make it more efficient?
  + i.e. if Jc (number of Js in a C) >= T (team size) then all highest ranks are put in
  + so, perhaps we could use the number of Js in a C to help pad the rank?
  + But there are some Cs with Jc >= T that DON’T get them all, so preference DOES matter…
* Can’t calculate all ranks and then just go through and assign the top of all 1st prefs because what if those 1st prefs change because some of the 2nd prefs lost out on their first prefs, so now this IS their first pref.. this is why you have to sort the jugglers
* Let’s say I have been placed into my second preference circuit. My first preference had only a rank of 75. However, if that circuit ws already filled with jugglers that all had better ranks than 75 then I would not get placed in it.
* You can’t pick the highest juggler of all your prefs (in a circuit) because that juggler could also be the highest rank if another circuit so only the circuit that has that juggler at a higher preference gets him/her;
  + therefore you have to pick the highest rank in the highest preference that is available to your circuit.
  + Which is why we can get away with the preliminary step of: as we add highest-ranking jugglers to the circuits, those that fill up the first preference first close the circuit

Other optimization techniques:

* Change firstPref to a prop and not a getter, so the sort does less work
* Remove jugglers from all their preferred circuits’ preferredJugglers lists when a juggler is assigned
* Do a sorted remove for the smarter sort, but will have to use the first ranks BEFORE the circuit is closed and they’re rmoved, otherwise it won’t be sorted, anymore, on firstRank