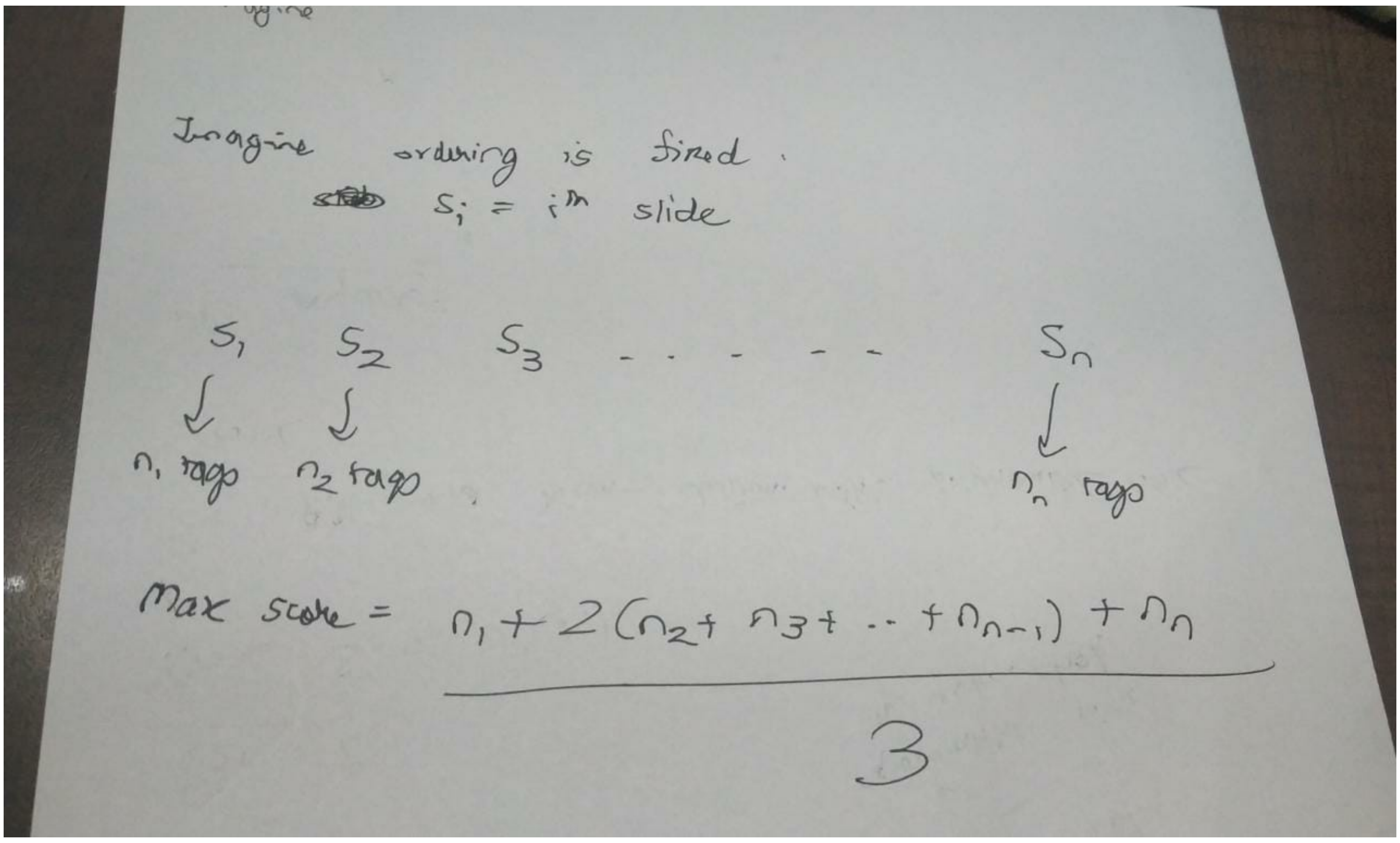
HashCode 2019 Approaches

Rough observations



Imagine ordering is fixed.

[illegible] S_i = i^{th} slide

$$\begin{tikzcd} S_1 \arrow[d] & S_2 \arrow[d] & S_3 & - - - - - & S_n \arrow[d] \\ n_1 \text{ tags} & n_2 \text{ tags} & & & n_n \text{ tags} \end{tikzcd}$$

$$\text{Max score} = \frac{n_1 + 2(n_2 + n_3 + \cdots + n_{n-1}) + n_n}{3}$$

b. Has no vertical (total - 80000)

C. is small in size (1000 photos) - 500 V + 500 H

D. 60K V + 30K H

E. has no horizontal

Number of Tags:

B: 840000

Each tag has 1-2 photos.

240000 have 1 photo 600000 have 2

C: 2166

<No. of occurrences of tag, No. of such tags>

1	871
2	439
3	435
9	10
10	184
17	1
18	4
19	37
20	185

D: 220

3-4k 49

17k 36

11 1

99 3

100 45

199 5

200 32

990-1k 50

E: 500

All tags 2900-3200 occurrences

Analysis of Edges

<BothV, BothH, VH> i.e. edges between 'pictures' [non-0-score].

C:

12990 11990 25026

Anmol

E.cpp → Iterate LS with random 100 neighbours allowed

Anmol's approach

Firstly, made observations to calculate a very high upper bound for the solution (didn't help us) .I first observed the similarities of the problem to non-metric TSP and began exploring possible heuristics to solve this. Upon not getting much progress, I coded local search by implementing basic 2-swap heuristic based on swap of which photos would lead to the maximum increase in objective function. $O(N^2)$ seemed costly and hence, later reduced to choosing of photos in a consecutive fashion. Results were not good enough.

Shashwat's approach

Upon checking which for each tag how many photos it appears in across files, the following useful property was noticed for file B:

6e5 tags appear in 2 photos, 2.4e5 appear in just 1 photo. Now since due to the large number of photos we were having trouble making a $O(N^2)$ adjacency matrix, and in any arrangement we want adjacent photos to have at least one tag in common, I basically just iterated over these 6e5 edges and added an edge between their 2 photos on an adjacency list. Then I ran a naive DFS and that give a decent score. I also tried to apply a heuristic of sorting the adjacency list for each node by it's score when put together with the adjacent node, but this approach only led to an increase of 1-2k in the score on the file.

Nikhil's Approach

My approach was primarily for test files D and E where the number of distinct tags were pretty low (< 500) so I had used *bitsets* to optimize the set intersection and difference functions. I implemented a greedy approach of finding the for each vertical the best match and a creating a slide for the pair. Once the slides are fixed, I implemented local search to swap slides if the score improves. The initial greedy took a lot of time to run since when N was large, $O(N^2)$ took 10mins to run on my PC after which local search kept improving for further 10 mins or so. The solution was okayish but there was a lot of scope for improvement.