1.

Let S_1 and E_1 be the starting and ending time of the first ending interval, S_2 and E_2 be the starting and ending time of the second ending interval, and so on.

Then we know that $E_1 \leq E_2 \leq \cdots \leq E_i$ for the first to the i-th interval.

We begin with the claim that there is an optimal solution that includes a student visit at the moment immediately prior to E_1.

We will prove the claim is correct by selecting an arbitrary optimal solution OPT.

Let x be the first student visit in OPT; and let c be the student visit at the moment immediately prior to E_1.

If x == c, then the claim is correct.

If $x \neq c$, then from the definition of c, it must be the latest possible time for a student visit that fulfill the first interval. Therefore, for OPT to fulfill the first interval with x given $x \neq c$, x must be earlier than c, in other words, $x < c$.

Let $OPT' = OPT - \{x\} + \{c\}$ We will prove the claim that OPT' is also an optimal solution.

First, the size of OPT' is the same as OPT, therefore the number of student visits is still minimum.

Then, we will prove OPT' is valid by showing that the number of intervals that c fulfills is no less than x.

For any i-th interval that is not the first, we know that $E_1 \leq E_i$. In addition, $S_1 < x < c < E_1$. We will consider the relationship between x and S_i.

Case $x > S_i$: we have $S_i < x < c < E_1 \leq E_i$, both x and c fulfill the first interval and the i-th interval. Therefore c fulfills is no less than x.

Case $x < S_i$: we have $x < S_i < E_i$ and $S_1 < x < c < E_1 \leq E_i$, both x and c fulfill the first interval. x does not fulfill the i-th interval, but c may or may not fulfill the i-th interval. In any case, the number of intervals that c fulfills is no less than x.

Here, we have shown that OPT' is valid and optimal, and since it contains c, we have proven the claim that there is an optimal solution that includes a student visit at the moment immediately prior to E_1; and that the algorithm shown in the problem statement can produce an optimal solution that minimizes the number of student visits.

2. **Algorithm:** for simplicity, sort the players from both team in increasing rating values, that is, let a_1 and s_1 denote the lowest rated player and a_n and s_n denote the highest rated player in each team.

Then, we pair each player from UCI to a player from LBSU, starting from the lowest rated player to the highest rated player, in other words, for i from 1 to n, we will pair a_i to some player s_k.

Let S denote the set of players in LBSU's team that is not paired with a UCI player.

For each a_i, if the player has a lower rating than all players in S, that is, $a_i < s_k$ for any $s_k \in S$, pair a_i with the player with the highest rating in S.

Else, let B denote the subset of S that contains the LBSU players with a lower rating than a_i, that is, $a_i > s_j$ for any $s_j \in B$, pair a_i with the player with the highest rating in B.

Repeat the process for all a_i, then we will have the maximum number of matches that UCI player has a higher rating than LBSU player.

Proof: We will prove that any other permutation will not achieve a better outcome than the one described above. In other words, the re-pair of an arbitrary inversion in any alternate ordering, compared to ours, produces an ordering that is no worse.

Using the fact from lecture that any other permutation has a pair of elements inverted compared to ours. Let X be the other permutation that does not agree with ours. Then, let A be the permutation of X with i, j swapped.

We will show that A is no worse than X. WLOG, assume that $a_i < a_j$ and $s_i < s_j$.

If $a_i < a_j < s_i < s_j$, we lose both matches in any matching orders.

If $s_i < s_j < a_i < a_j$, we win both matches in any matching orders.

If $s_i < a_i < a_j < s_j$ or $a_i < s_i < s_j < a_j$, we win only one match in any matching orders.

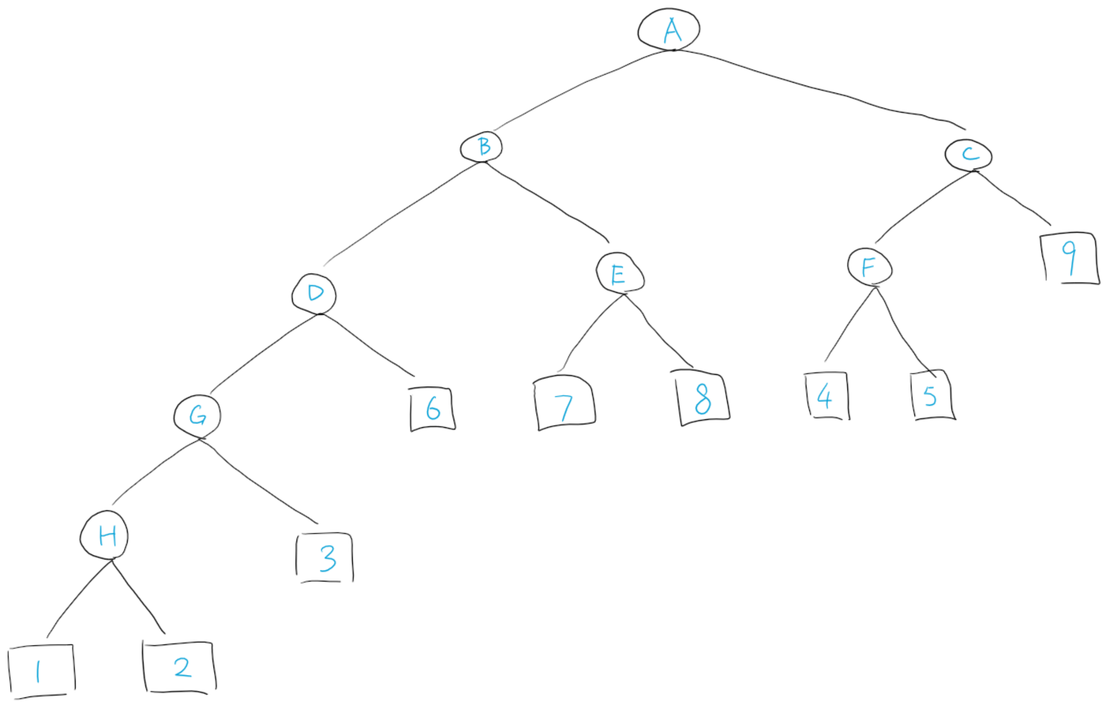
If $a_i < s_i < a_j < s_j$, by the assumption of X that it does not follow our algorithm, X will match a_i with s_i; and match a_j with s_j, losing both matches. Then, swapping the match in A will make us win one match.

If $s_i < a_i < s_j < a_j$, by the assumption of X that it does not follow our algorithm, X will match a_i with s_j; and match a_j with s_i, winning one match. Then, swapping the match in A will make us win both matches.

In cases that $a_i > a_j$ and/or $s_i > s_j$. We can follow the same logic as above, just with different orders.

Here, we have shown that the re-pair of an arbitrary inversion in any alternate ordering, compared to ours, produces an ordering that is no worse. Therefore, our algorithm is correct and can maximize the number of matches that our team has a higher rating than the opponent.

3.



The decision tree is constructed with the same method as the Huffman tree shown in lecture. We start with the two cards or nodes with the lowest count, merge them into one node with count equal to the sum of both cards/nodes, and repeat the process until we have the tree. For example, we will start with “1” and “2” since they have the lowest counts, merge them into node H with a total count of 3, and repeat.

In each non-leaf node, we will ask the question “Is the card one of {S}?” where S is the set of leaf nodes in the left subtree; and if the answer is “yes” we will go to the left child node, or if the answer is “no” we will go to the right, repeat until we get to a leaf node which is the value of the card we are looking for. For example, at node B in the tree above, we will ask “Is the card one of {1,2,3,6}?” if the answer is “yes” we will go to node D, or node E if the answer is “no”.

The tree minimizes the expected number of questions, or $\sum_{i=1}^{9} C_i D_i$, where C_i is the count of card i and D_i is the depth of node i in the tree. Because (1) all internal nodes have two children and (2) the cards with lower counts are at a higher depth, and the two cards with the minimum counts are at the maximum depth.