LeaderBoard Computational Algo

*** Ranking for a single pool
*** Scoring is out of 10

It is also assumed that the score computational function is pre-existing. We just have to produce rankings.

We'll add a certain set of conditions into the score function or module to introduce slots (for classification).

Like for example, a score of 10 will be divided into 10 slots of the following classification:
0-1 Grade J
1-2 Grade I
2-3 Grade H
3-4 Grade G
4-5 Grade F
5-6 Grade E
6-7 Grade D
7-8 Grade C
8-9 Grade B
9<=10 Grade A

These slots will be further sub-divided into Sub Slots.

0.1 - 0.2 Grade a
0.2 - 0.3 Grade b
0.3 - 0.4 Grade c
0.4 - 0.5 Grade d
0.5 - 0.6 Grade e
0.6 - 0.7 Grade f
0.7 - 0.8 Grade g
0.8 - 0.9 Grade h
0.9 - 0.00 Grade i
For example, a score of 7.90 will be classified as C.i & this slot can have thousands of individual scores of course.

We can also compute the maximum score for a pool from the score module or function. (by comparing every value from the previous while computing score (complexity will not be affected much)).

To update rankings, the score must be computed time after time
& similarly, the grading and the ranking must be done consecutively time after time.

Rankings

For a particular Grade, only subgrades should be sorted
& if sorting of different grade is done on multiple platform then this would be much more faster and efficient.
Like Grade A.a is sorted using machine 1
& Grade A.b is sorted using different machine say 2,
** machine could be virtual

Once all of the grade subdata data is sorted,
We'll get a sorted grade
Where within grade we will have entries in increasing order(sorted).
For example: if A.a has data '9.14','9.12''9.17','9.17';(Before Sorting)
After sorting 9.14 9.12 & 9.17, 9.17 we'll get
'9.12','9.14','9.17','9.17';

Now maximum score should be on the top of our rankings,
So we'll start to count from the bottom of the sorted list(or sort in decreasing order).
If multiple ranking matches we'll rank them equal.
So the ranking will be
9.17 Rank 1
9.17 Rank 1
9.14 Rank 2
9.12 Rank 3

****Assumption Aleart**
If (seems good with the company policy)
{

In order to update rankings again and again, we have to classify and sort data again and again.
So, to be more efficient

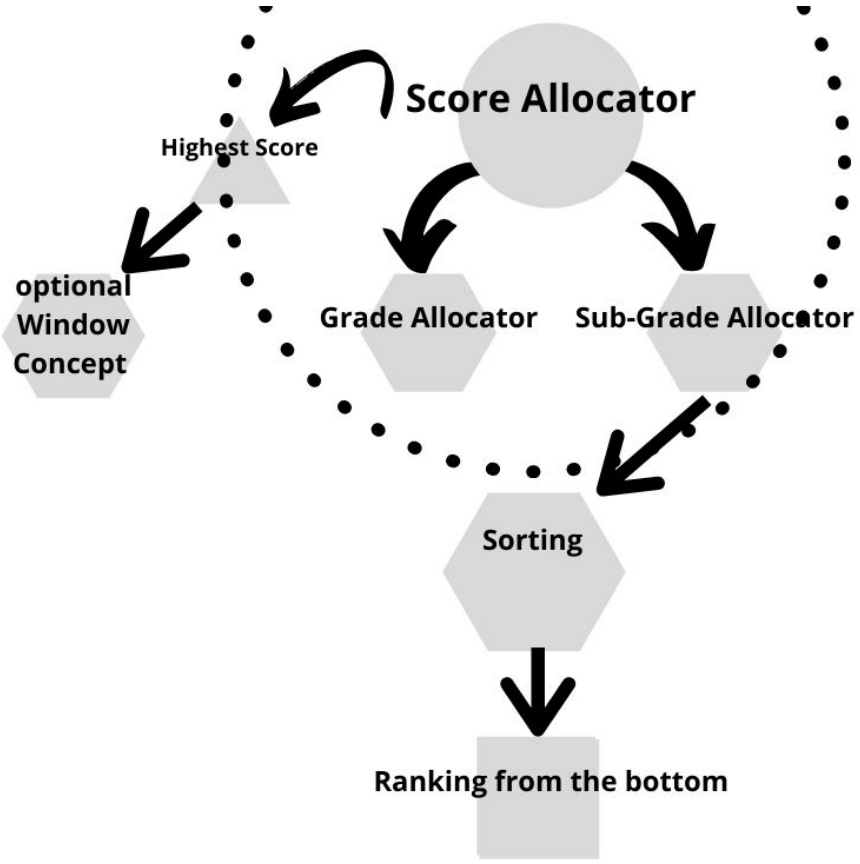
We can choose a window for ranking
And not rank the rest.

For example, if the max score comes out to be 8.2- we can choose a window of size 4 or so &
People having scores in between the range 4.2 - 8.2(window size 4) will be only ranked
For the rest, we can say
rank > n thousand.

This will be much efficient
But for the final time, we can skip this method and sort the entire dataset
And we'll only have to sort that, once.
To produce final ranking.

}

Flow Chart



Pros:

-Memory Efficient Algorithm

-Multiple machine usage for computation will make it more time efficient

Cons:

-Crux Level Algorithm,
Implementation will be different & complex.

-Only for one pool, If ranking were to be made by combining different pools algo must be modified & upgraded.

**Algo should be able to handle heavy load.