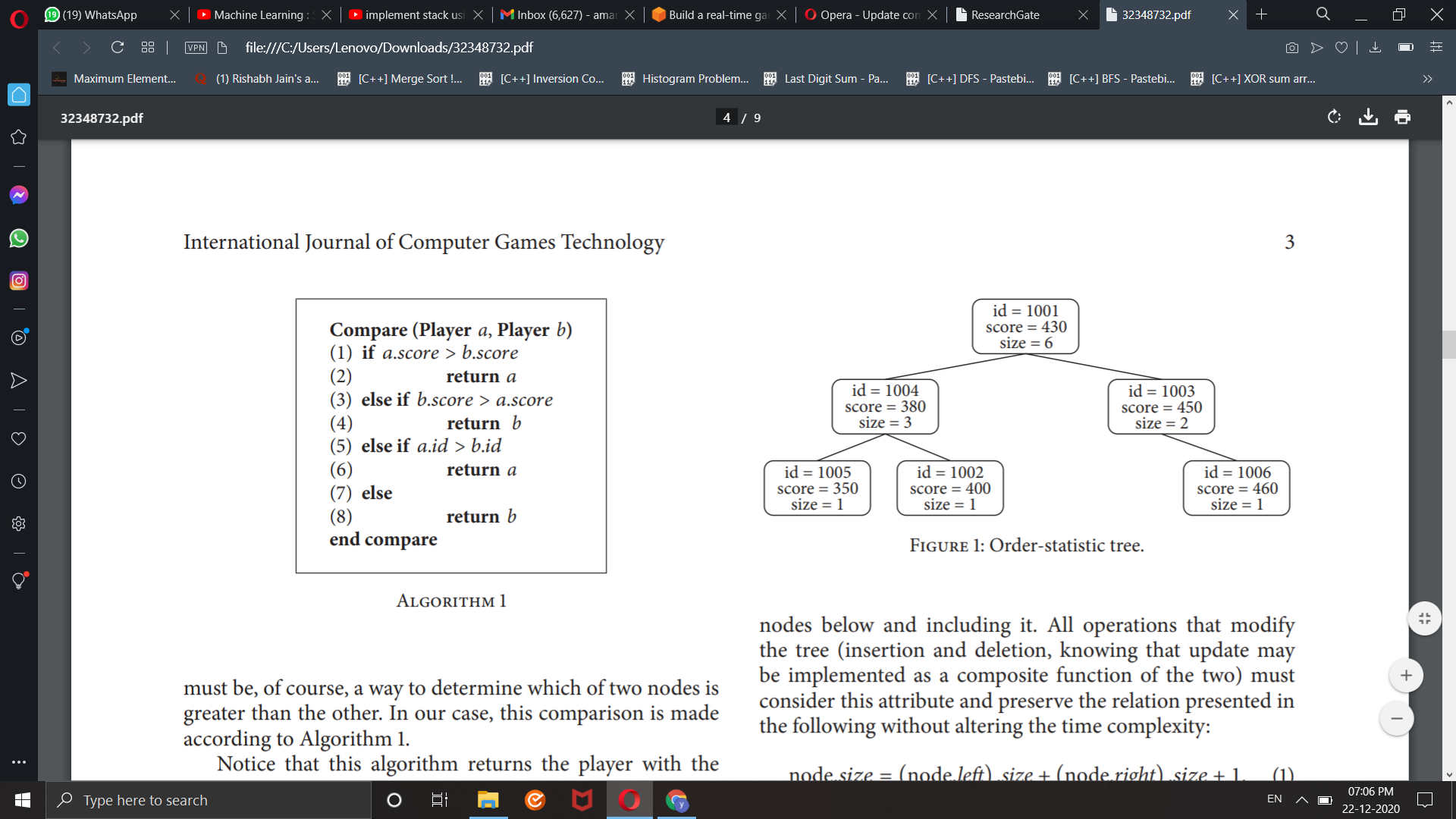
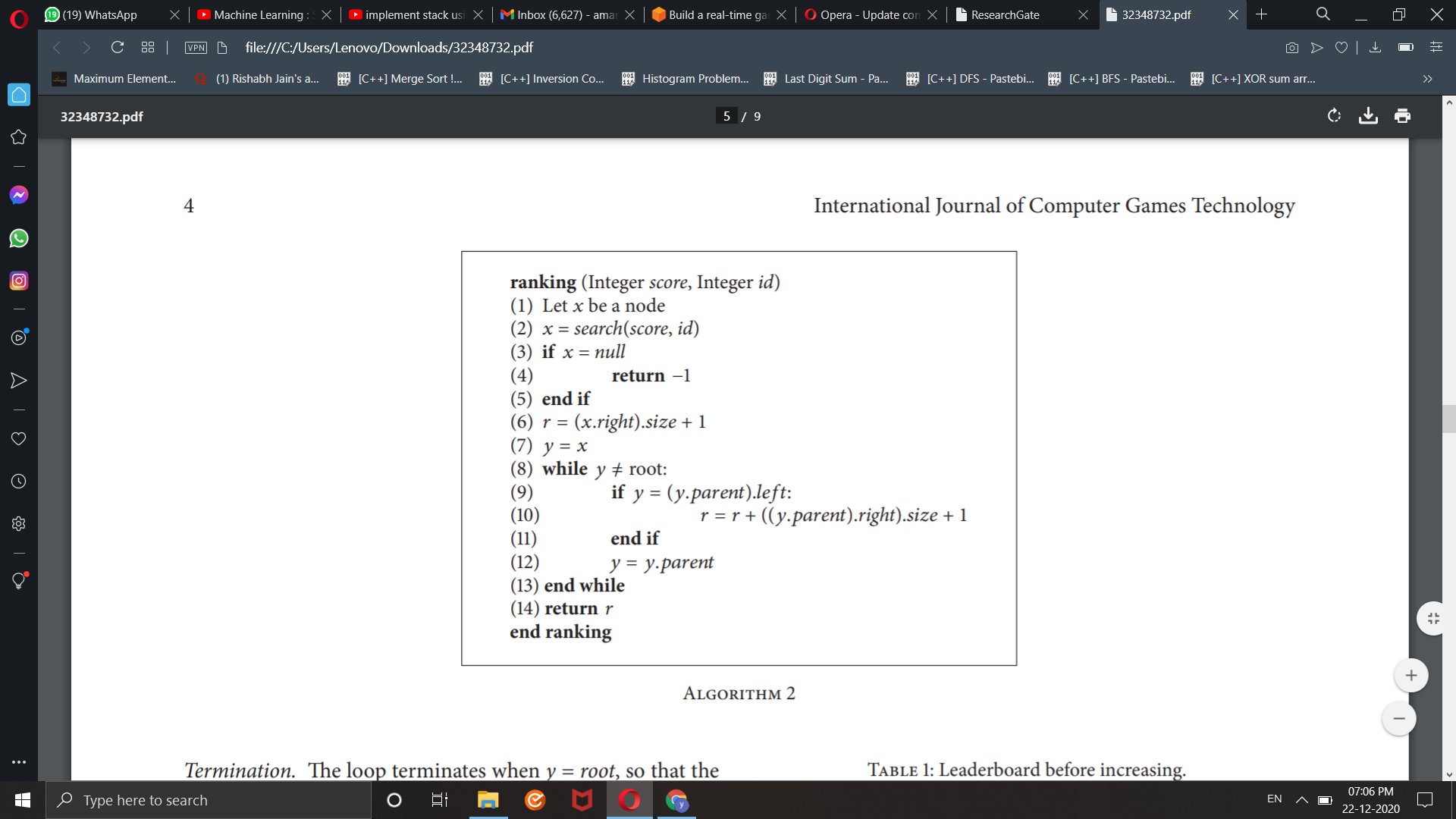
**Deliverables:**

There can be 4 possible algorithms:

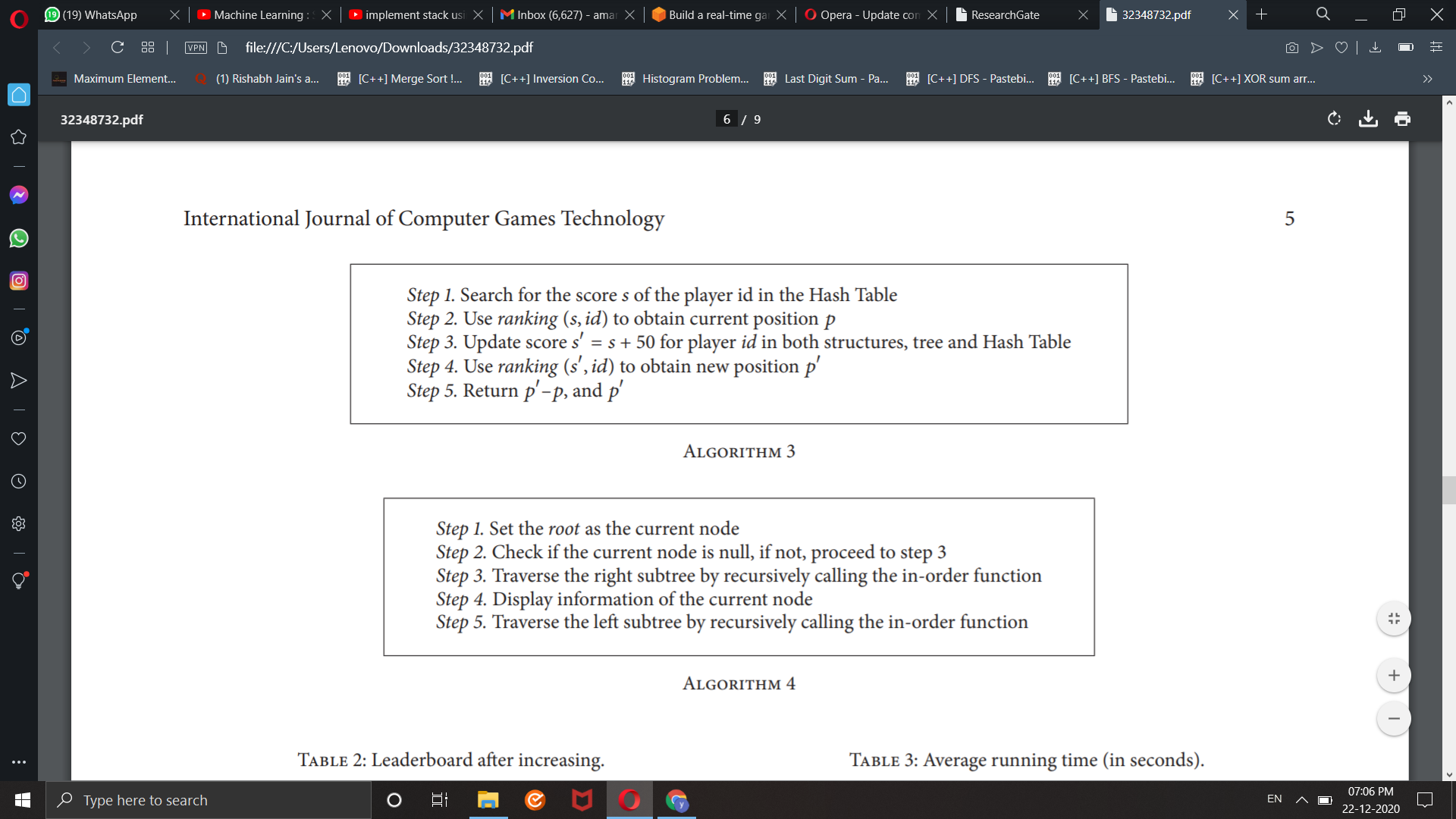
1.)



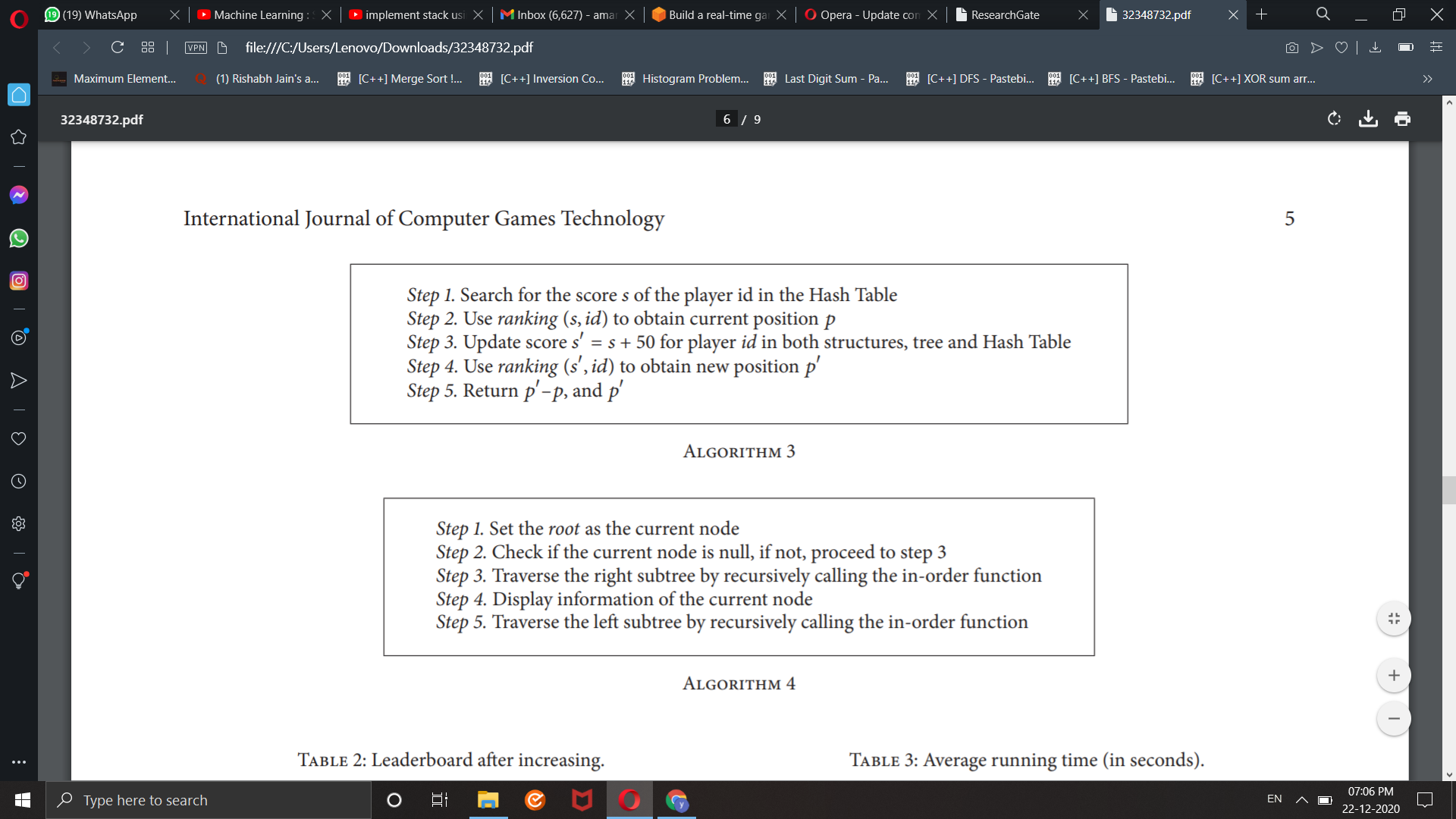
2.)



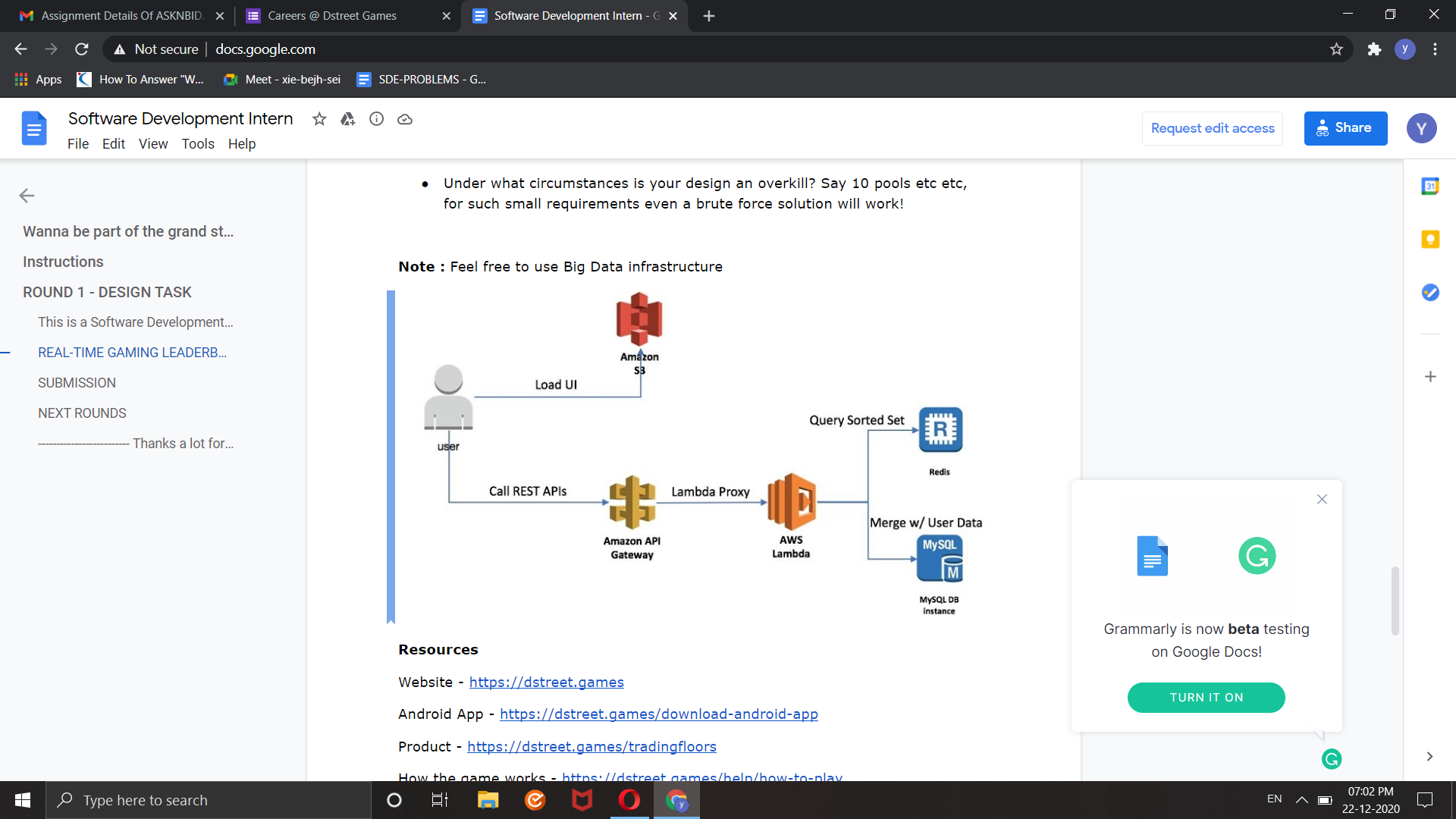
3.)



4.)



Designed flowchart:



Assumptions:

First, it runs “in-memory,” so it exploits fast data access, unlike slower HDD solutions. Although it might sound as a disadvantage as well, because the reduced space may limit how many users you can have, it is not so problematic considering that in its basic form only the user score and its identifier are needed.

For instance, if a four-byte unsigned integer is used for both attributes, a scenario with 100,000 players would require 800,000 bytes, which is less than 1 Mb. Second, it uses specific data structures, so no ordering at all is actually needed for obtaining players positions. More specifically, it uses an SBOST jointly with a hash table which allows for performing all important operations in O(log2N) time complexity. The SBOST was implemented from a Red-Black Tree, but other alternatives for Self-Balanced Binary Search Trees could be adopted as well. Third, the comparison criterion, which ultimately defines the rank of a player, may be easily modified in order to adjust to the designer needs. For instance, it could incorporate more information about the player, rather than just a single score and an identifier.

From the algorithmic point of view, such a proposal surpasses typical solutions as the ones based on databases, as well as other “in-memory,” simpler, alternatives as ordered linked lists. As presented in the experimental results section, we achieved speedups on all the scenarios we tested. In fact, the more difficult the scenario, the higher the speedup. For example, such a speedup with an input size  = 100,000 was nearly 1,000 : 1 and 15,000 : 1 compared to the other two approaches presented. With the forecast coming from a multiple linear regression with  million (actual running of such a scenario would be impractical) the corresponding speedups would be nearly as large as 10,000 : 1 and 160,000 : 1. This finding turns out to be very relevant in massive environments where dozens or even hundreds of thousands of users are common.

**Pros**

1. **Real time**predictions: It is very fast and can be used in real time.
2. S**calable**with Large datasets
3. **Insensitive to irrelevant features.**
4. **Good performance with high dimensional data**(no. of features is large)

**Cons**

1. **Independence of features does not hold:**The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome. However this condition is not met most of the times.

2. **Bad estimator:**Probability outputs are not to be taken too seriously.

3. **Training data should represent population well**