Online Assessment of Ivy_Homes

1. **The working code** is present in the Ivy_Homes folder present in this Repo.
Also, this folder consist 3 sub-folders named v1, v2 and v3, which contains code with their respective endpoints.
Each sub-folder contains 4 files-
--> vX.js : where the main logic is written
--> count.js : this is a logic to extract data from both the json files and return count of names and totalRequest made
--> wordvX.json : where all the extracted names are saved
--> reqvX.json : where the count of totalRequest made to the server is stored

***(if you look into wordvX and reqX, both are array that data with respect to the first character in query).
*** X: 1, 2, or 3

2. Coming to my approach-

a) first I tried to figure out the response coming from the endpoints. The following are my observations-

i) the response of api contains results arrays which holds the actual names to be extracted and count that holds the length of results.

ii) the maximum count of names coming in a response is limited, that is 10, 12 and 15 for endpoints v1, v2 and v3.

iii) names extracted from endpoint -
~ v1 contains only small case characters
~ v2 contains small case as well as numeric characters
~ v3 contains smallcase, numeric as well as some special characters.

b) Firstly, I made an array of characters based on the above operation. And took ans empty string curr = "";

c) Then, I append a character from the array to curr and hit the corresponding endpoint. Let's take an example, assume I append character 'a' to curr and hit the request to the v1 endpoint(http://35.200.185.69:8000/v1/autocomplete?query=${curr}), further I got a response, there may be two situation-

i) count < 10 (maximum limit of v1): then its fine I'll just store all the names.

ii) count == 10 : then I'll back to the step (c) and process again till count == 10. (that means I'll search for 'aa', 'ab',....).

d) The moments I'll not get any count == 10(maximum limit) it will simply return and store the result.

***This approach is way too optimised as the code runs only in that direction where there is chance of finding result.

3. During the assessment I created-

a) a recursive function(solve) that optimally search for the result
b) two functions readExistingData && readReqData that store my result to a permanent file.

4. Constraints(Rate Limit)-

I observed that each version of endpoints had some constraints on making the number of requests per minute. Like-
- v1 has a limit of 100 req/min.
- v2 has a limit of 50 req/min.
- v3 has a limit of 80 req/min.

In order tackle these I limited the number of request to-
- 1 req/600ms for v1.
- 1 req/1300ms for v2.
- 1 req/750ms for v3.

5. Total number of requests needed to make to the API-
- 32284 for v1
- 7738 for v2
- 3268 for v3

6. Total number of records obtain from the API-
- 19375 from v1
- 14165 from v2
- 11283 from v3