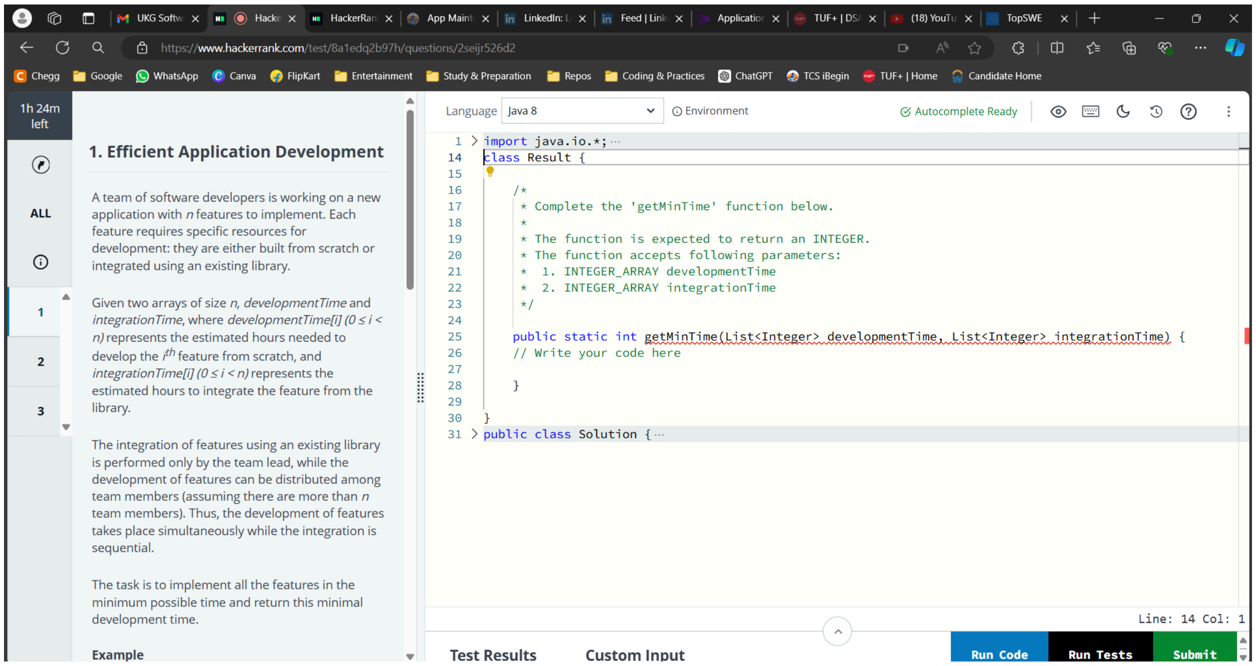
UKG Coding Assessment

Question 1:



1. Efficient Application Development

A team of software developers is working on a new application with *n* features to implement. Each feature requires specific resources for development: they are either built from scratch or integrated using an existing library.

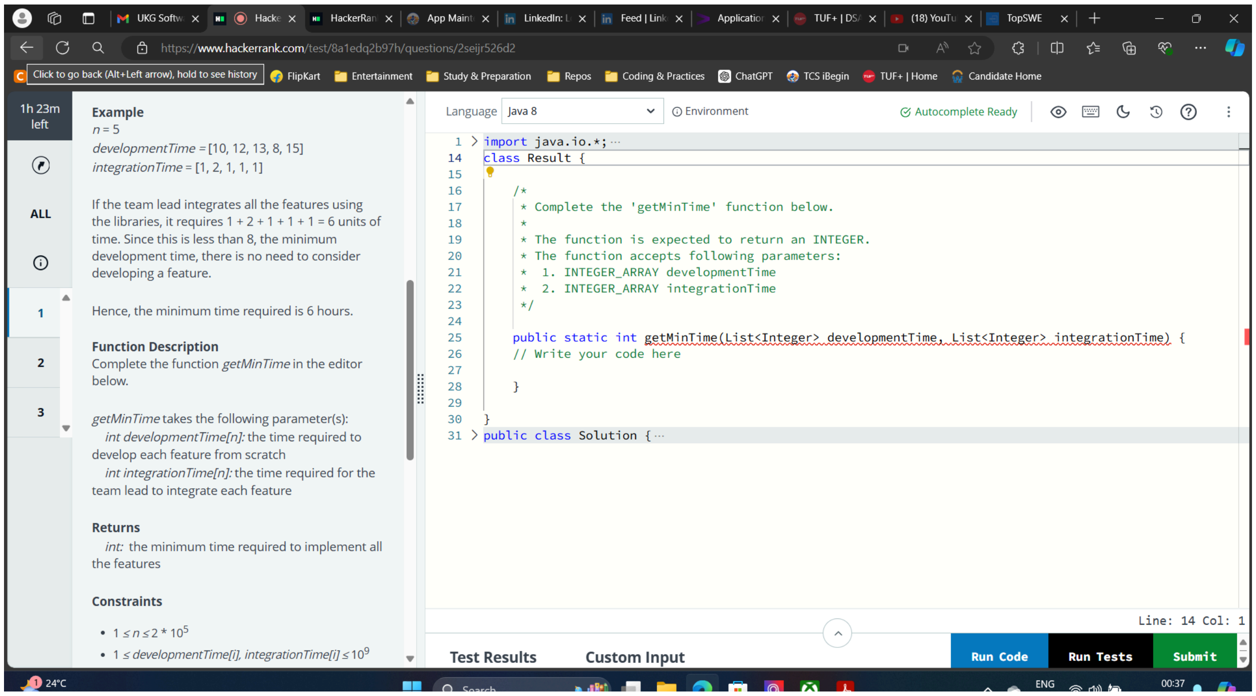
Given two arrays of size *n*, *developmentTime* and *integrationTime*, where *developmentTime[i]* $(0 \le i < n)$ represents the estimated hours needed to develop the i^{th} feature from scratch, and *integrationTime[i]* $(0 \le i < n)$ represents the estimated hours to integrate the feature from the library.

The integration of features using an existing library is performed only by the team lead, while the development of features can be distributed among team members (assuming there are more than *n* team members). Thus, the development of features takes place simultaneously while the integration is sequential.

The task is to implement all the features in the minimum possible time and return this minimal development time.

Example

```java
import java.io.*;···
class Result {

    /*
     * Complete the 'getMinTime' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY developmentTime
     *  2. INTEGER_ARRAY integrationTime
     */

    public static int getMinTime(List<Integer> developmentTime, List<Integer> integrationTime) {
    // Write your code here

    }

}
public class Solution {···
```



Example
$n = 5$
developmentTime = [10, 12, 13, 8, 15]
integrationTime = [1, 2, 1, 1, 1]

If the team lead integrates all the features using the libraries, it requires 1 + 2 + 1 + 1 + 1 = 6 units of time. Since this is less than 8, the minimum development time, there is no need to consider developing a feature.

Hence, the minimum time required is 6 hours.

Function Description
Complete the function *getMinTime* in the editor below.

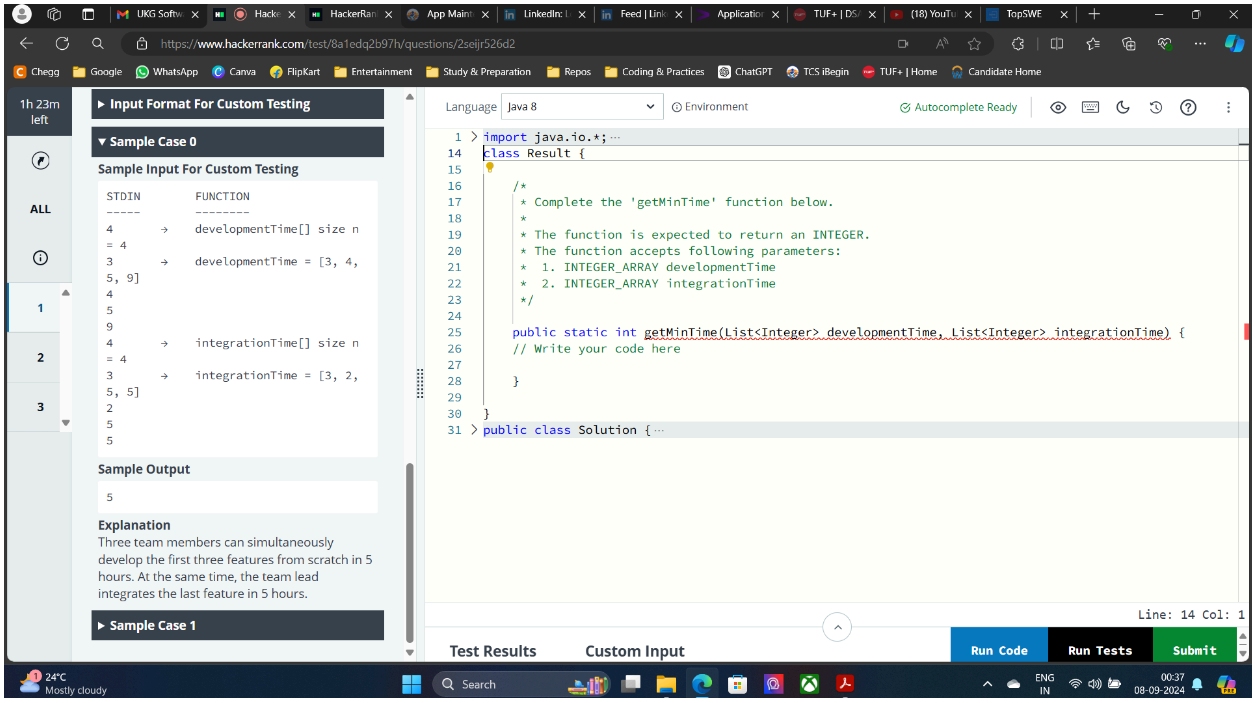
getMinTime takes the following parameter(s):
int developmentTime[n]: the time required to develop each feature from scratch
int integrationTime[n]: the time required for the team lead to integrate each feature

Returns
int: the minimum time required to implement all the features

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le developmentTime[i], integrationTime[i] \le 10^9$

```java
import java.io.*;···
class Result {

    /*
     * Complete the 'getMinTime' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY developmentTime
     *  2. INTEGER_ARRAY integrationTime
     */

    public static int getMinTime(List<Integer> developmentTime, List<Integer> integrationTime) {
    // Write your code here

    }

}
public class Solution {···
```



▸ Input Format For Custom Testing

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
4      →    developmentTime[] size n
= 4
3      →    developmentTime = [3, 4,
5, 9]
4
5
9
4      →    integrationTime[] size n
= 4
3      →    integrationTime = [3, 2,
5, 5]
2
5
5
```

Sample Output

```
5
```

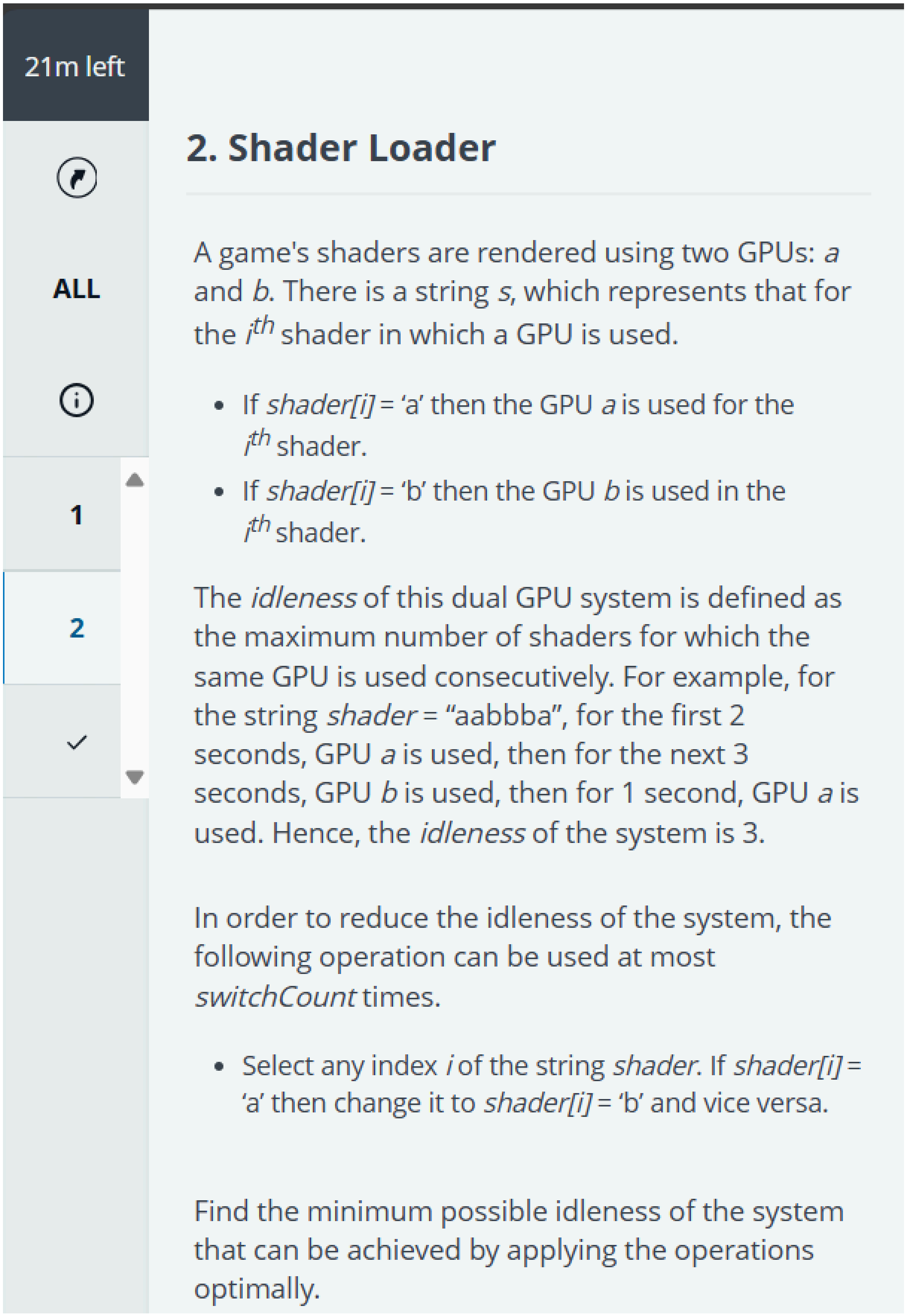
Explanation

Three team members can simultaneously develop the first three features from scratch in 5 hours. At the same time, the team lead integrates the last feature in 5 hours.

▸ Sample Case 1

```java
import java.io.*;…
class Result {

    /*
     * Complete the 'getMinTime' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY developmentTime
     *  2. INTEGER_ARRAY integrationTime
     */

    public static int getMinTime(List<Integer> developmentTime, List<Integer> integrationTime) {
    // Write your code here

    }

}
public class Solution {…
```

Question 2:



2. Shader Loader

A game's shaders are rendered using two GPUs: *a* and *b*. There is a string *s*, which represents that for the *i*th shader in which a GPU is used.

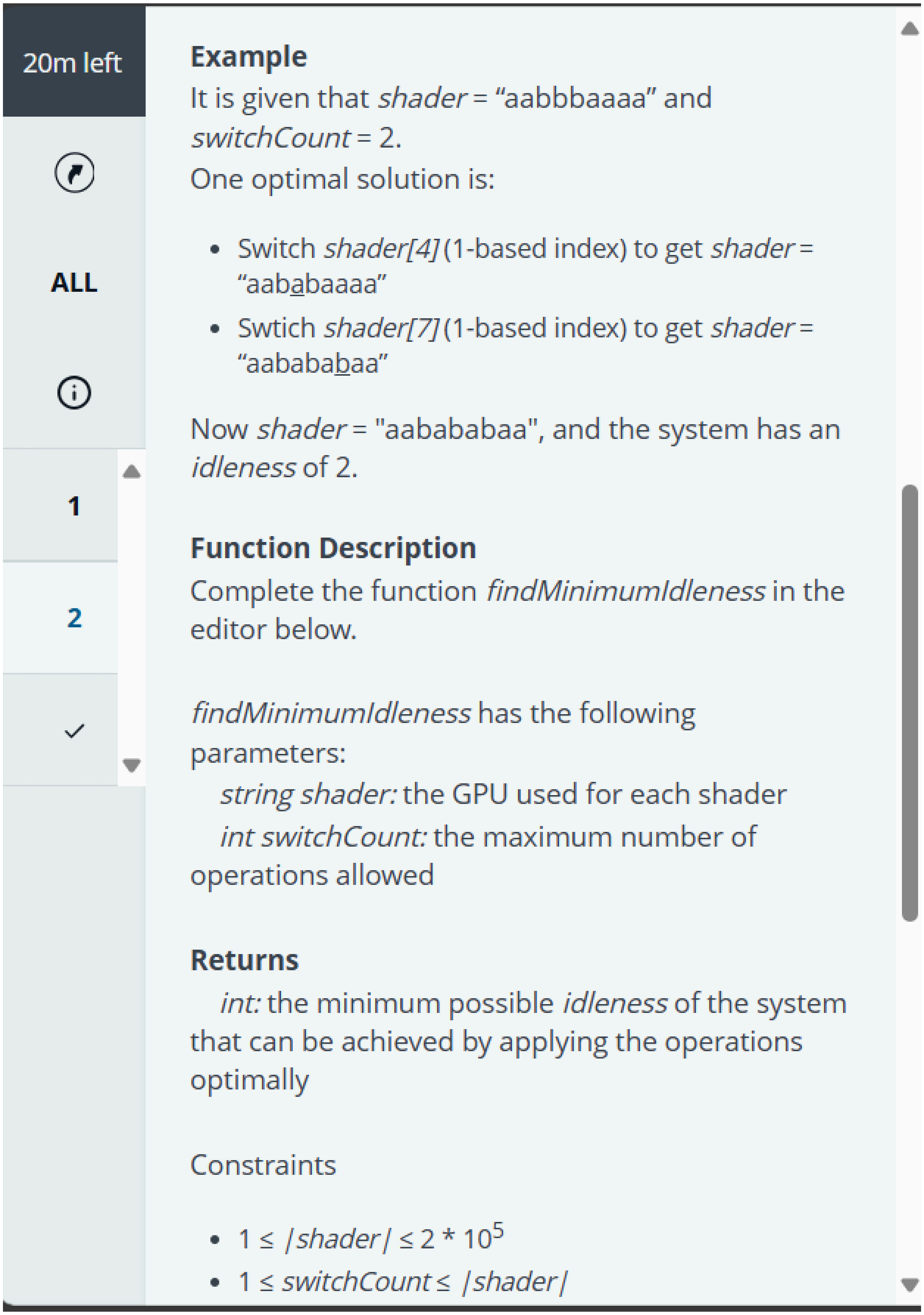
- If *shader[i]* = 'a' then the GPU *a* is used for the *i*th shader.
- If *shader[i]* = 'b' then the GPU *b* is used in the *i*th shader.

The *idleness* of this dual GPU system is defined as the maximum number of shaders for which the same GPU is used consecutively. For example, for the string *shader* = "aabbba", for the first 2 seconds, GPU *a* is used, then for the next 3 seconds, GPU *b* is used, then for 1 second, GPU *a* is used. Hence, the *idleness* of the system is 3.

In order to reduce the idleness of the system, the following operation can be used at most *switchCount* times.

- Select any index *i* of the string *shader*. If *shader[i]* = 'a' then change it to *shader[i]* = 'b' and vice versa.

Find the minimum possible idleness of the system that can be achieved by applying the operations optimally.



Example

It is given that *shader* = “aabbbaaaa” and *switchCount* = 2.

One optimal solution is:

- Switch *shader[4]* (1-based index) to get *shader* = “aababaaaa”
- Swtich *shader[7]* (1-based index) to get *shader* = “aabababaa”

Now *shader* = "aabababaa", and the system has an *idleness* of 2.

Function Description

Complete the function *findMinimumIdleness* in the editor below.

findMinimumIdleness has the following parameters:

string shader: the GPU used for each shader

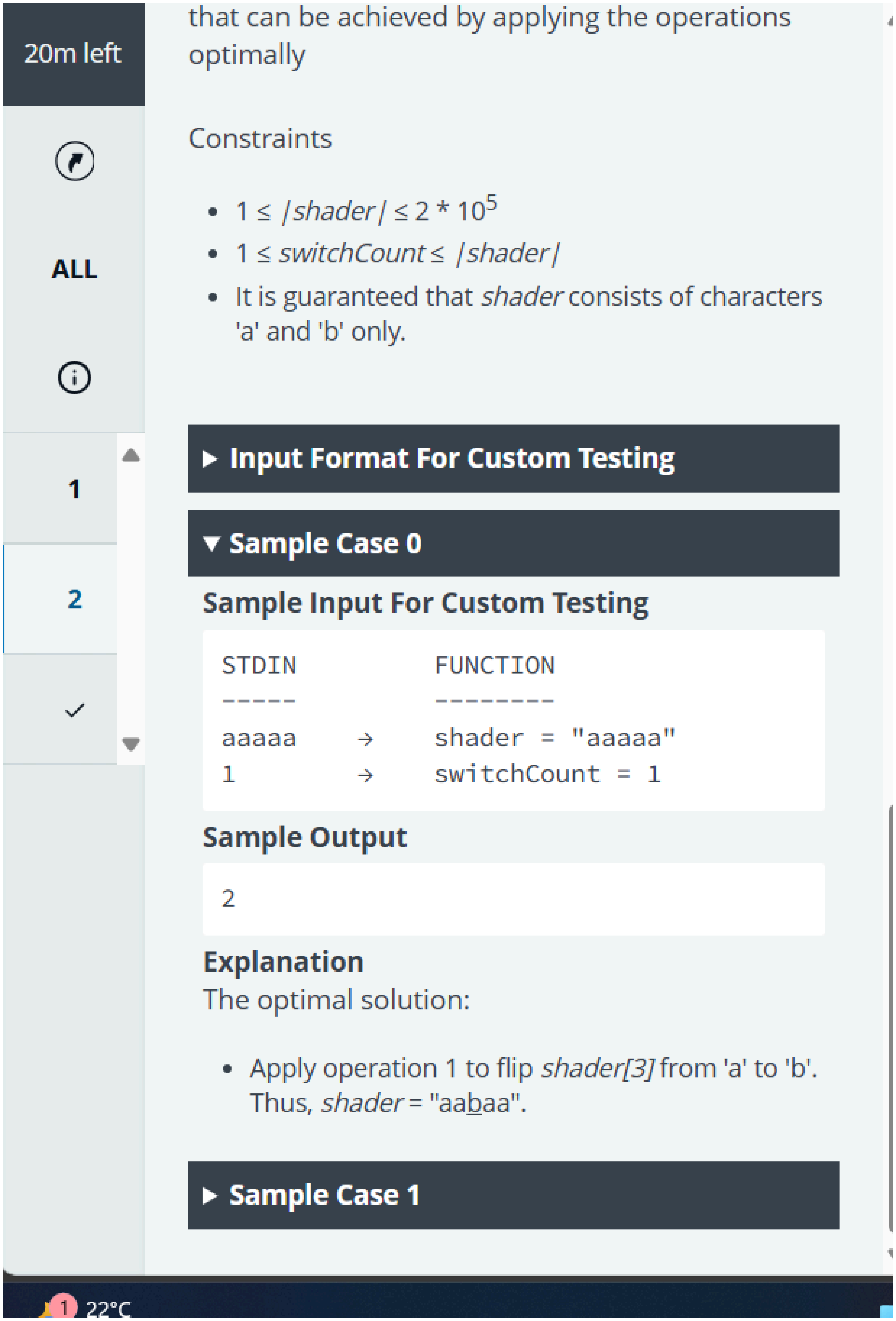
int switchCount: the maximum number of operations allowed

Returns

int: the minimum possible *idleness* of the system that can be achieved by applying the operations optimally

Constraints

- $1 \le |shader| \le 2 * 10^5$
- $1 \le switchCount \le |shader|$



that can be achieved by applying the operations optimally

Constraints

- 1 ≤ *|shader|* ≤ 2 * 10^5
- 1 ≤ *switchCount* ≤ *|shader|*
- It is guaranteed that *shader* consists of characters 'a' and 'b' only.

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input For Custom Testing

```
STDIN          FUNCTION
-----          --------
aaaaa     →    shader = "aaaaa"
1         →    switchCount = 1
```

Sample Output

```
2
```

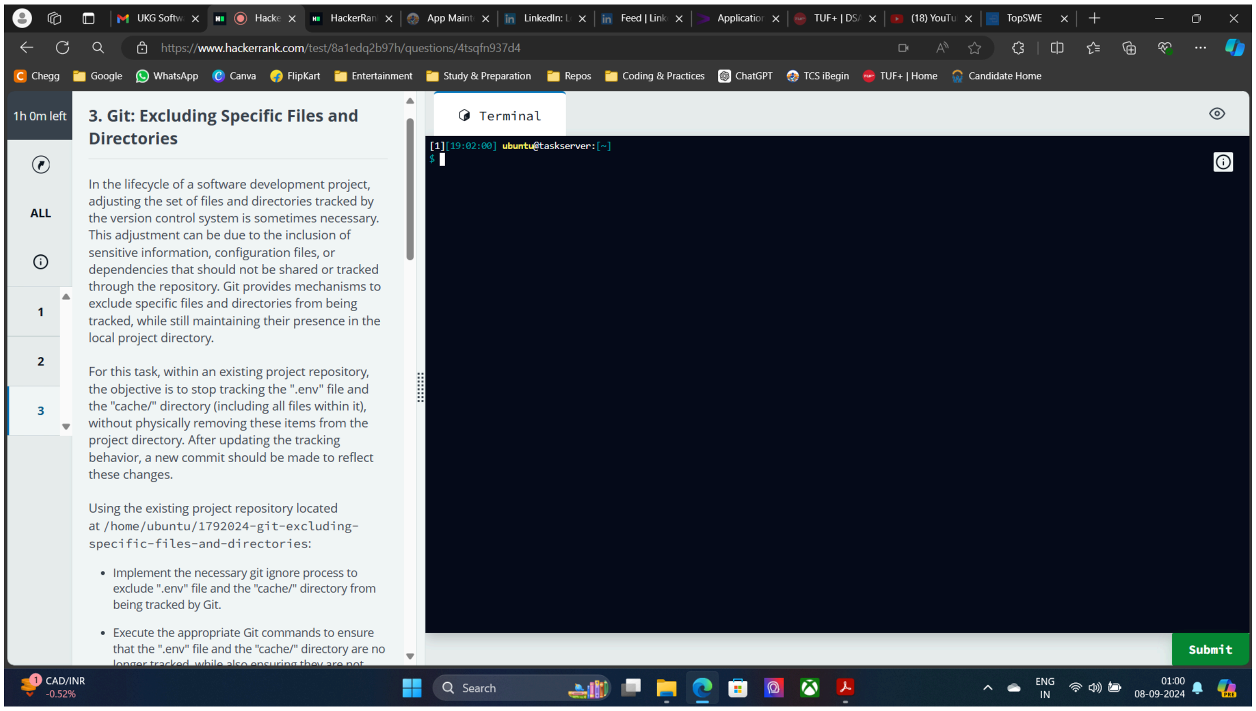
Explanation

The optimal solution:

- Apply operation 1 to flip *shader[3]* from 'a' to 'b'. Thus, *shader* = "aa<u>b</u>aa".

▶ **Sample Case 1**

Question 3:



https://www.hackerrank.com/test/8a1edq2b97h/questions/4tsqfn937d4
1h 0m left
ALL
1
2
3
Terminal
Submit

3. Git: Excluding Specific Files and Directories

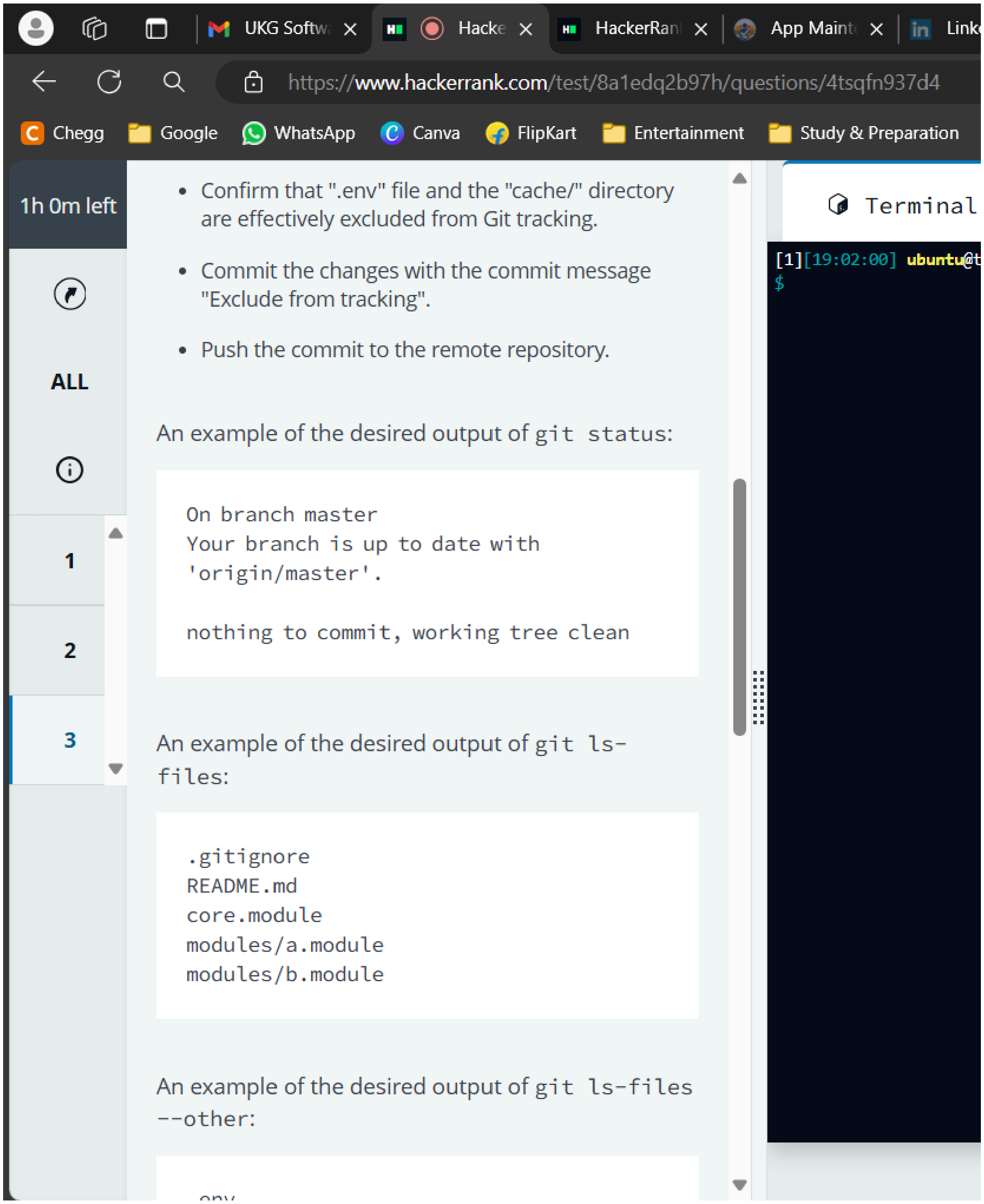
In the lifecycle of a software development project, adjusting the set of files and directories tracked by the version control system is sometimes necessary. This adjustment can be due to the inclusion of sensitive information, configuration files, or dependencies that should not be shared or tracked through the repository. Git provides mechanisms to exclude specific files and directories from being tracked, while still maintaining their presence in the local project directory.

For this task, within an existing project repository, the objective is to stop tracking the ".env" file and the "cache/" directory (including all files within it), without physically removing these items from the project directory. After updating the tracking behavior, a new commit should be made to reflect these changes.

Using the existing project repository located at `/home/ubuntu/1792024-git-excluding-specific-files-and-directories`:

- Implement the necessary git ignore process to exclude ".env" file and the "cache/" directory from being tracked by Git.
- Execute the appropriate Git commands to ensure that the ".env" file and "cache/" directory are no longer tracked, while also ensuring they are not

```
[1][19:02:00] ubuntu@taskserver:[~]
$ 
```



- Confirm that ".env" file and the "cache/" directory are effectively excluded from Git tracking.
- Commit the changes with the commit message "Exclude from tracking".
- Push the commit to the remote repository.

An example of the desired output of `git status`:

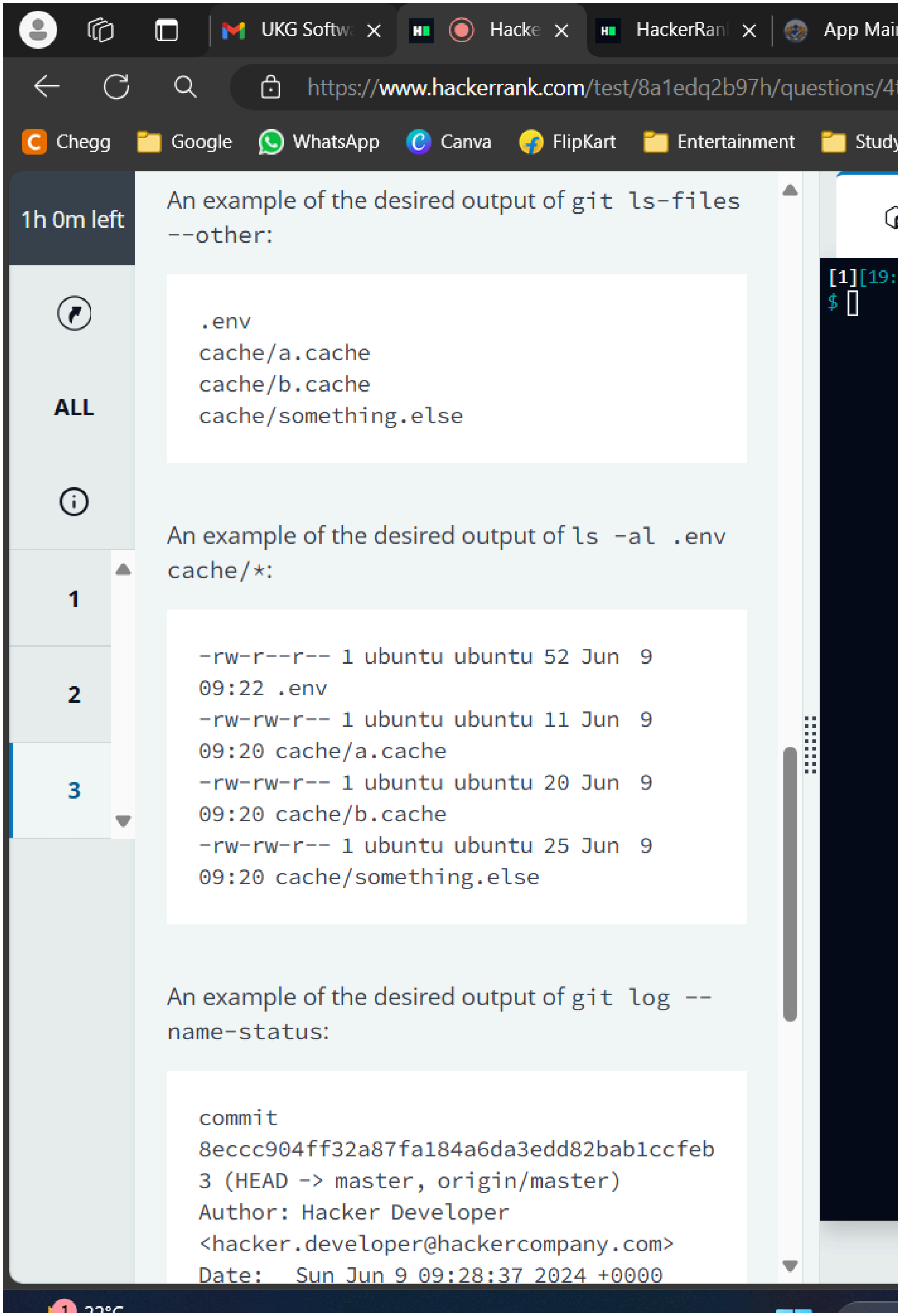
```
On branch master
Your branch is up to date with
'origin/master'.

nothing to commit, working tree clean
```

An example of the desired output of `git ls-files`:

```
.gitignore
README.md
core.module
modules/a.module
modules/b.module
```

An example of the desired output of `git ls-files --other`:

```
[1][19:02:00] ubuntu@t
$
```



An example of the desired output of `git ls-files --other`:

```
.env
cache/a.cache
cache/b.cache
cache/something.else
```

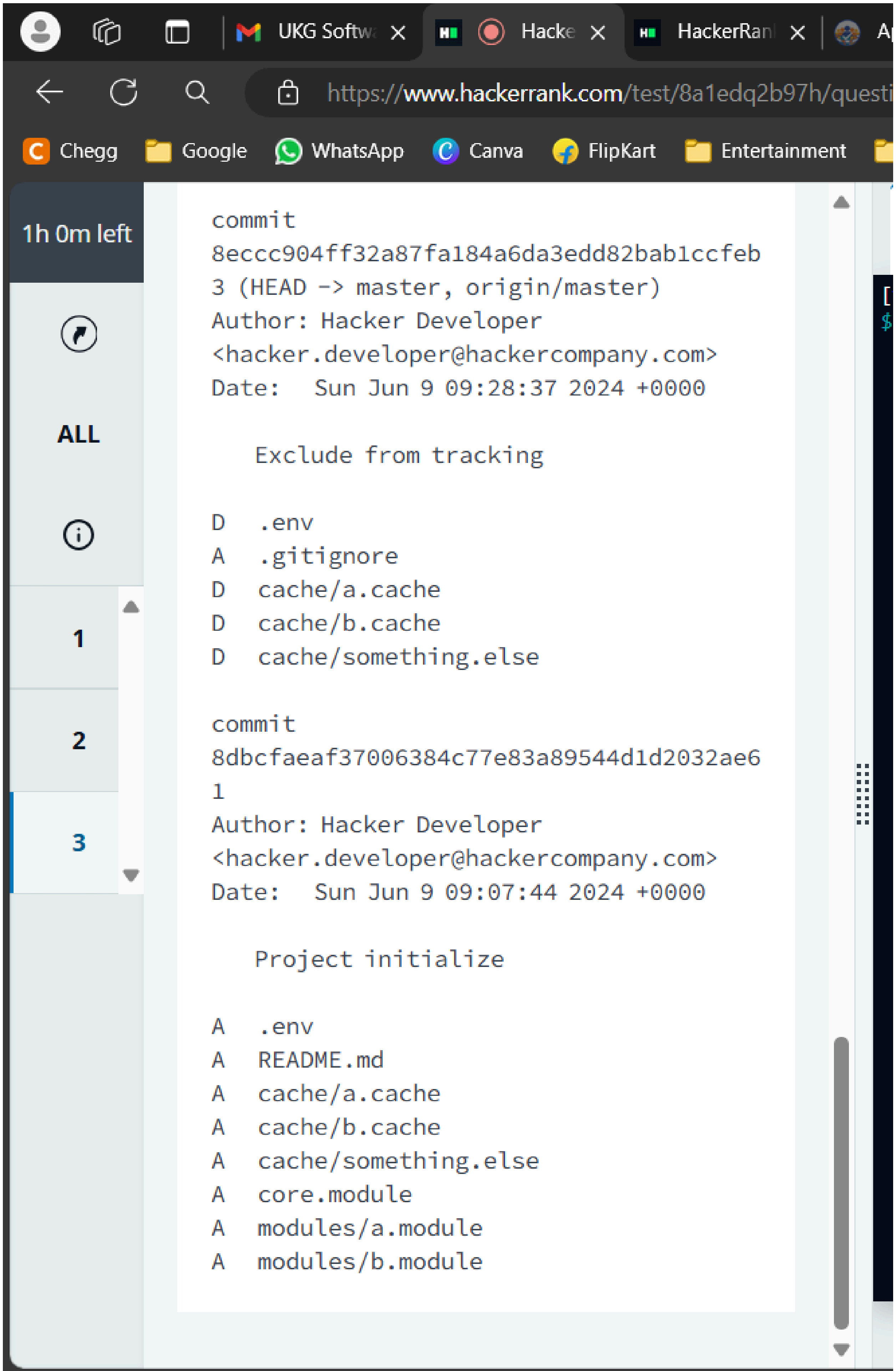
An example of the desired output of `ls -al .env cache/*`:

```
-rw-r--r-- 1 ubuntu ubuntu 52 Jun  9
09:22 .env
-rw-rw-r-- 1 ubuntu ubuntu 11 Jun  9
09:20 cache/a.cache
-rw-rw-r-- 1 ubuntu ubuntu 20 Jun  9
09:20 cache/b.cache
-rw-rw-r-- 1 ubuntu ubuntu 25 Jun  9
09:20 cache/something.else
```

An example of the desired output of `git log --name-status`:

```
commit
8eccc904ff32a87fa184a6da3edd82bab1ccfeb
3 (HEAD -> master, origin/master)
Author: Hacker Developer
<hacker.developer@hackercompany.com>
Date:   Sun Jun 9 09:28:37 2024 +0000
```

```
[1][19:
$ 
```



```
commit
8eccc904ff32a87fa184a6da3edd82bab1ccfeb
3 (HEAD -> master, origin/master)
Author: Hacker Developer
<hacker.developer@hackercompany.com>
Date:   Sun Jun 9 09:28:37 2024 +0000

    Exclude from tracking

D   .env
A   .gitignore
D   cache/a.cache
D   cache/b.cache
D   cache/something.else

commit
8dbcfaeaf37006384c77e83a89544d1d2032ae6
1
Author: Hacker Developer
<hacker.developer@hackercompany.com>
Date:   Sun Jun 9 09:07:44 2024 +0000

    Project initialize

A   .env
A   README.md
A   cache/a.cache
A   cache/b.cache
A   cache/something.else
A   core.module
A   modules/a.module
A   modules/b.module
```