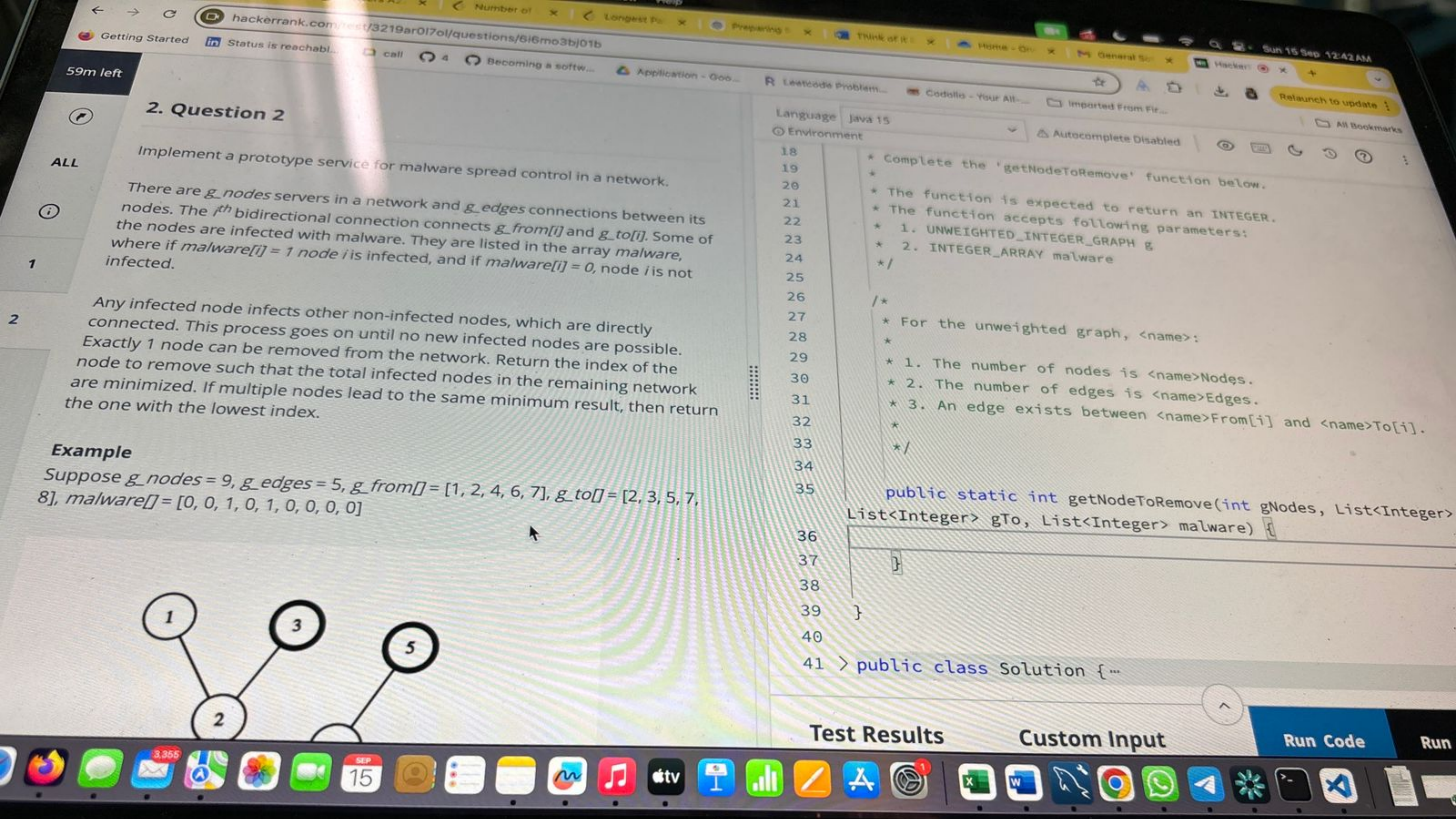


hackerrank.com/test/3219ar0l7ol/questions/6i6mo3bj01b

59m left

ALL

2. Question 2

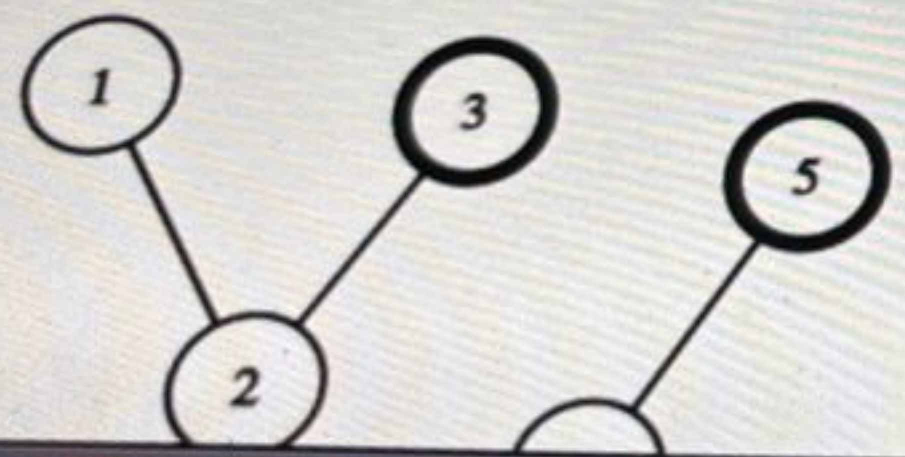
Implement a prototype service for malware spread control in a network.

There are *g_nodes* servers in a network and *g_edges* connections between its nodes. The *i*th bidirectional connection connects *g_from[i]* and *g_to[i]*. Some of the nodes are infected with malware. They are listed in the array *malware*, where if *malware[i] = 1 node i* is infected, and if *malware[i] = 0,* node *i* is not infected.

Any infected node infects other non-infected nodes, which are directly connected. This process goes on until no new infected nodes are possible. Exactly 1 node can be removed from the network. Return the index of the node to remove such that the total infected nodes in the remaining network are minimized. If multiple nodes lead to the same minimum result, then return the one with the lowest index.

Example

Suppose *g_nodes* = 9, *g_edges* = 5, *g_from[]* = [1, 2, 4, 6, 7], *g_to[]* = [2, 3, 5, 7, 8], *malware[]* = [0, 0, 1, 0, 1, 0, 0, 0, 0]



Language Java 15

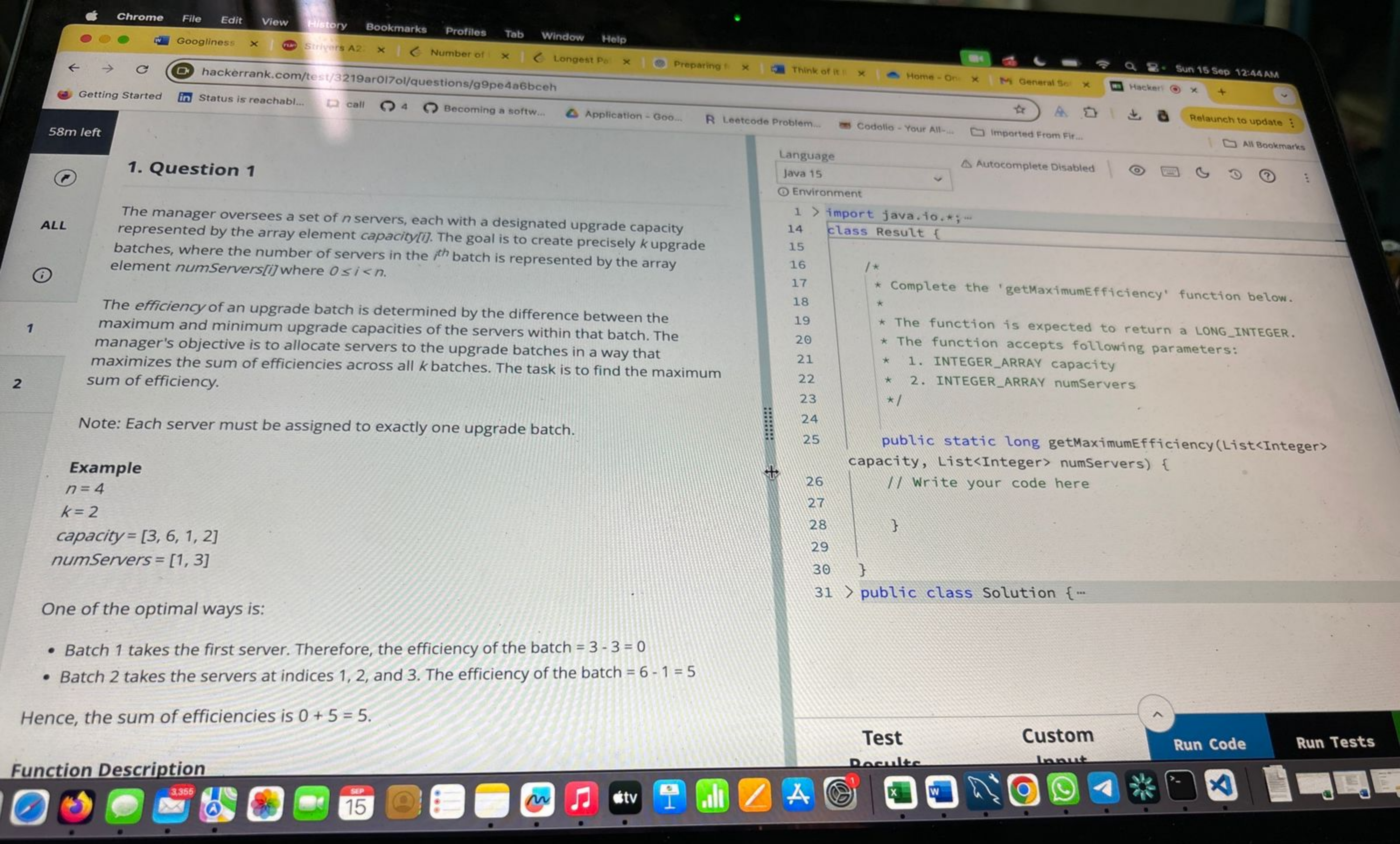
Environment

Autocomplete Disabled

```
     * Complete the 'getNodeToRemove' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. UNWEIGHTED_INTEGER_GRAPH g
     *  2. INTEGER_ARRAY malware
     */

    /*
     * For the unweighted graph, <name>:
     *
     * 1. The number of nodes is <name>Nodes.
     * 2. The number of edges is <name>Edges.
     * 3. An edge exists between <name>From[i] and <name>To[i].
     *
     */

    public static int getNodeToRemove(int gNodes, List<Integer>
List<Integer> gTo, List<Integer> malware) {

    }

}

public class Solution {…
```

Test Results   Custom Input   Run Code   Run

MacBook Air



58m left

1. Question 1

The manager oversees a set of *n* servers, each with a designated upgrade capacity represented by the array element *capacity[i]*. The goal is to create precisely *k* upgrade batches, where the number of servers in the i^{th} batch is represented by the array element *numServers[i]* where $0 \le i < n$.

The *efficiency* of an upgrade batch is determined by the difference between the maximum and minimum upgrade capacities of the servers within that batch. The manager's objective is to allocate servers to the upgrade batches in a way that maximizes the sum of efficiencies across all *k* batches. The task is to find the maximum sum of efficiency.

Note: Each server must be assigned to exactly one upgrade batch.

Example

$n = 4$

$k = 2$

capacity = [3, 6, 1, 2]

numServers = [1, 3]

One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

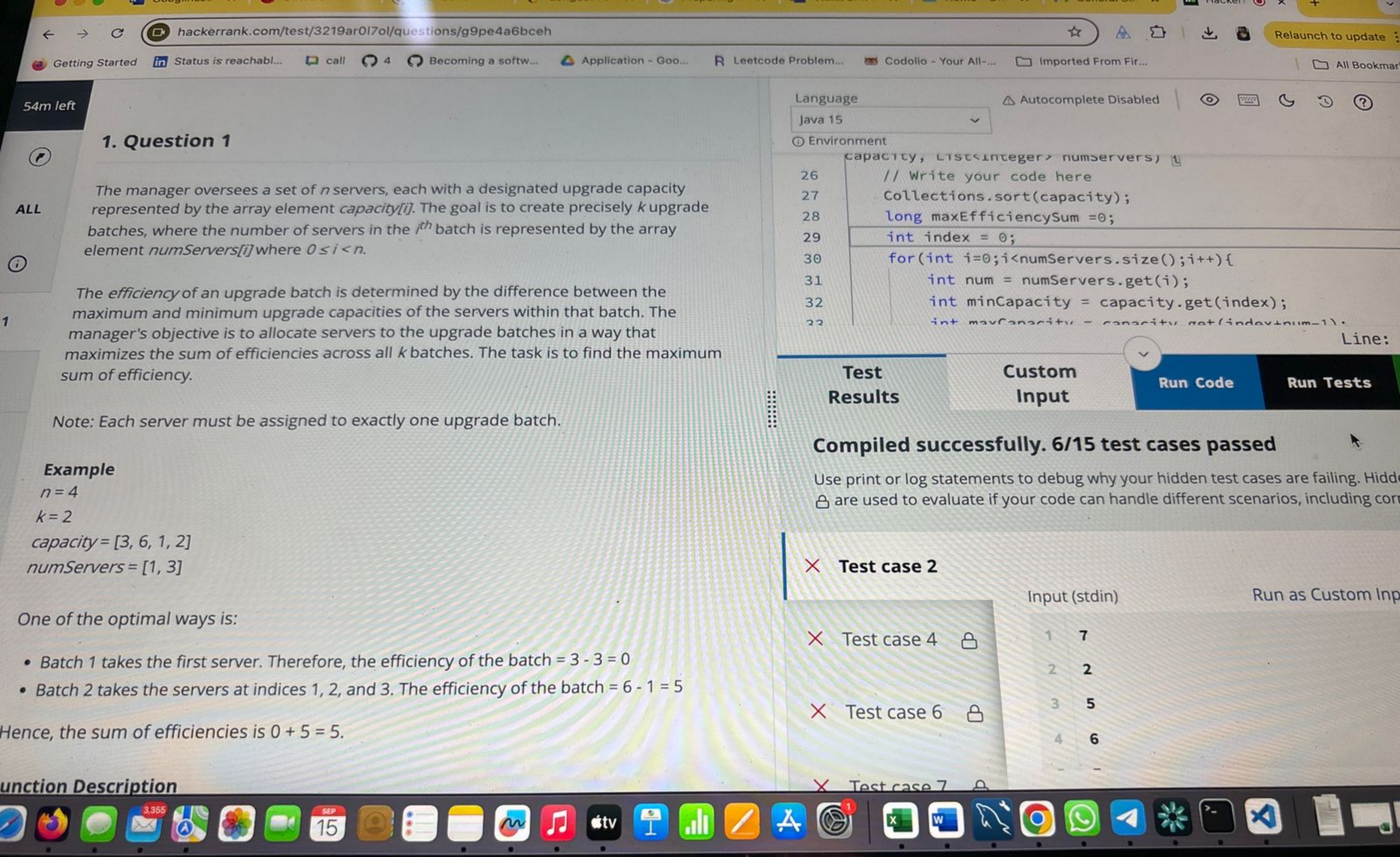
Hence, the sum of efficiencies is 0 + 5 = 5.

Function Description

Language: Java 15

Autocomplete Disabled

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

    /*
     * Complete the 'getMaximumEfficiency' function below.
     *
     * The function is expected to return a LONG_INTEGER.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY capacity
     *  2. INTEGER_ARRAY numServers
     */

    public static long getMaximumEfficiency(List<Integer> capacity, List<Integer> numServers) {
    // Write your code here

    }

}
> public class Solution {…
```



hackerrank.com/test/3219ar0l7ol/questions/g9pe4a6bceh

54m left

1. Question 1

The manager oversees a set of *n* servers, each with a designated upgrade capacity represented by the array element *capacity[i]*. The goal is to create precisely *k* upgrade batches, where the number of servers in the i^{th} batch is represented by the array element *numServers[i]* where $0 \le i < n$.

The *efficiency* of an upgrade batch is determined by the difference between the maximum and minimum upgrade capacities of the servers within that batch. The manager's objective is to allocate servers to the upgrade batches in a way that maximizes the sum of efficiencies across all *k* batches. The task is to find the maximum sum of efficiency.

Note: Each server must be assigned to exactly one upgrade batch.

Example

$n = 4$

$k = 2$

capacity = [3, 6, 1, 2]

numServers = [1, 3]

One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

Hence, the sum of efficiencies is 0 + 5 = 5.

unction Description

Language: Java 15

Autocomplete Disabled

```
capacity, List<Integer> numServers) {
    // Write your code here
    Collections.sort(capacity);
    long maxEfficiencySum =0;
    int index = 0;
    for(int i=0;i<numServers.size();i++){
        int num = numServers.get(i);
        int minCapacity = capacity.get(index);
        int maxCapacity = capacity.get(index+num-1);
```

Test Results | Custom Input | Run Code | Run Tests

Compiled successfully. 6/15 test cases passed

Use print or log statements to debug why your hidden test cases are failing. Hidd
are used to evaluate if your code can handle different scenarios, including cor

✗ **Test case 2**

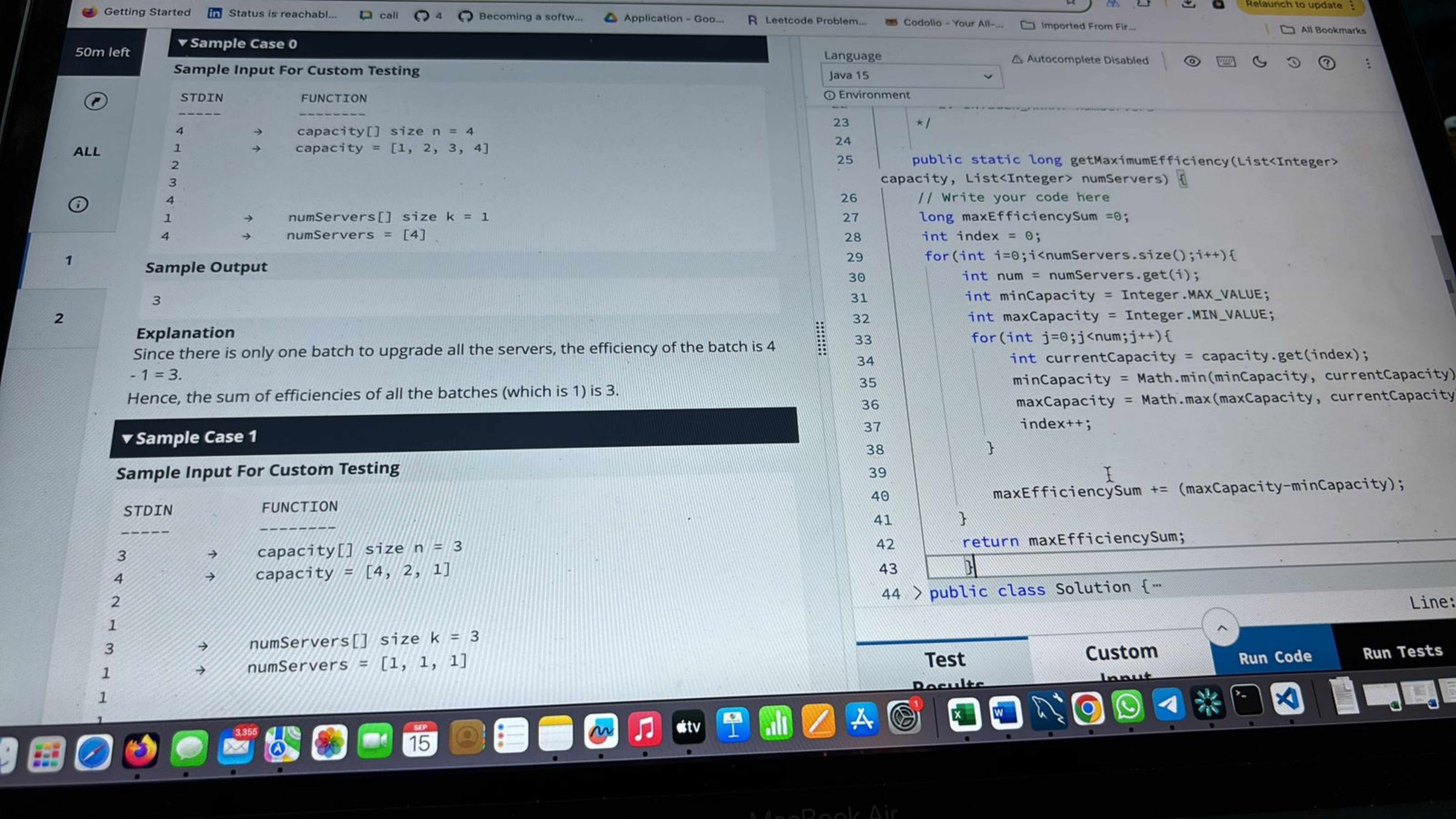
✗ Test case 4

✗ Test case 6

✗ Test case 7

Input (stdin) | Run as Custom Inp

```
7
2
5
6
```



▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
4     →     capacity[] size n = 4
1     →     capacity = [1, 2, 3, 4]
2
3
4
1     →     numServers[] size k = 1
4     →     numServers = [4]
```

Sample Output

```
3
```

Explanation

Since there is only one batch to upgrade all the servers, the efficiency of the batch is 4 - 1 = 3.

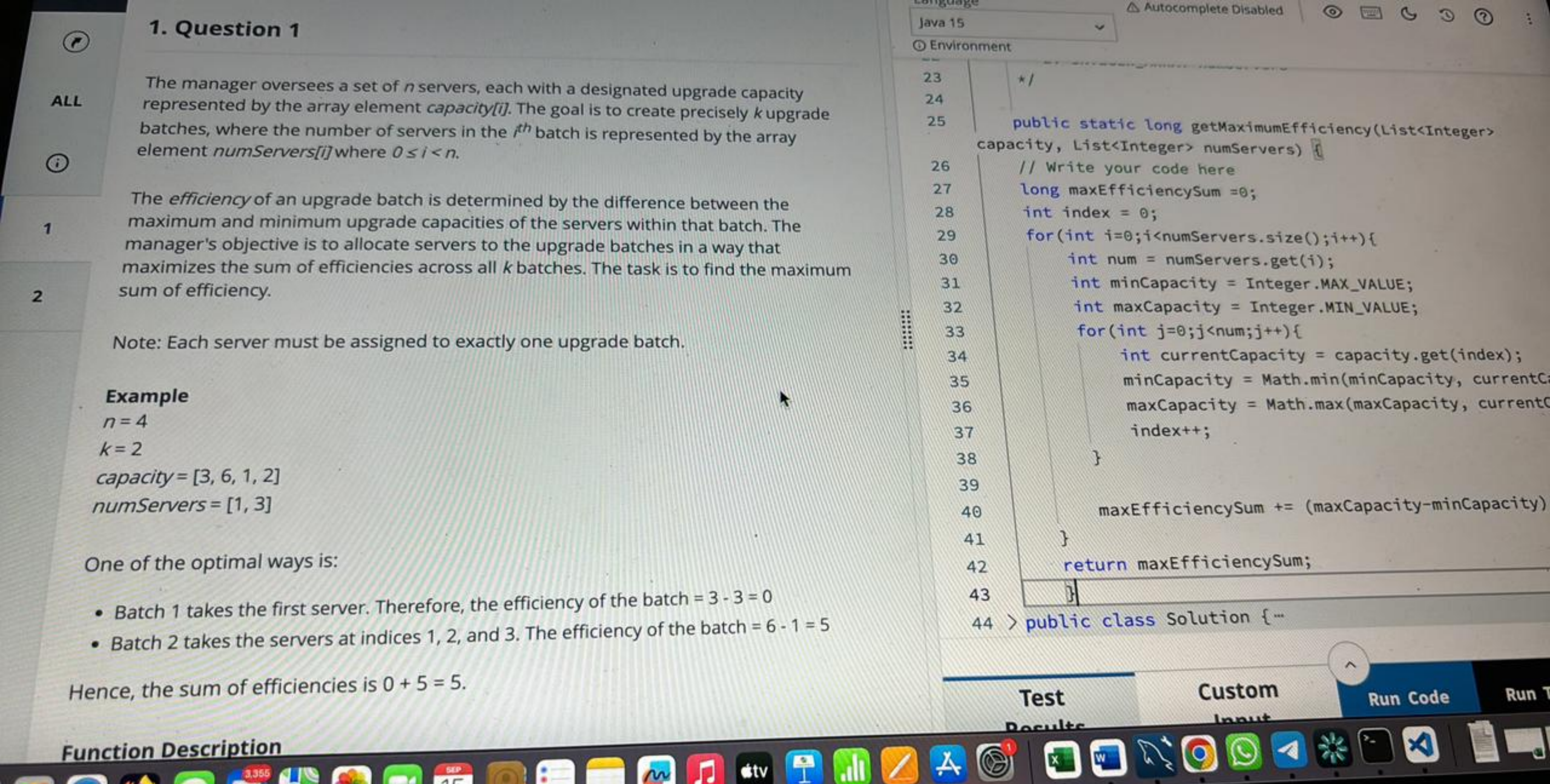
Hence, the sum of efficiencies of all the batches (which is 1) is 3.

▾ Sample Case 1

Sample Input For Custom Testing

```
STDIN       FUNCTION
-----       --------
3     →     capacity[] size n = 3
4     →     capacity = [4, 2, 1]
2
1
3     →     numServers[] size k = 3
1     →     numServers = [1, 1, 1]
1
1
```

```java
     */

    public static long getMaximumEfficiency(List<Integer> capacity, List<Integer> numServers) {
    // Write your code here
    long maxEfficiencySum =0;
    int index = 0;
    for(int i=0;i<numServers.size();i++){
        int num = numServers.get(i);
        int minCapacity = Integer.MAX_VALUE;
        int maxCapacity = Integer.MIN_VALUE;
        for(int j=0;j<num;j++){
            int currentCapacity = capacity.get(index);
            minCapacity = Math.min(minCapacity, currentCapacity)
            maxCapacity = Math.max(maxCapacity, currentCapacity
            index++;
        }

        maxEfficiencySum += (maxCapacity-minCapacity);
    }
    return maxEfficiencySum;
    }
> public class Solution {…
```



1. Question 1

The manager oversees a set of *n* servers, each with a designated upgrade capacity represented by the array element *capacity[i]*. The goal is to create precisely *k* upgrade batches, where the number of servers in the i^{th} batch is represented by the array element *numServers[i]* where $0 \le i < n$.

The *efficiency* of an upgrade batch is determined by the difference between the maximum and minimum upgrade capacities of the servers within that batch. The manager's objective is to allocate servers to the upgrade batches in a way that maximizes the sum of efficiencies across all *k* batches. The task is to find the maximum sum of efficiency.

Note: Each server must be assigned to exactly one upgrade batch.

Example

$n = 4$

$k = 2$

capacity = [3, 6, 1, 2]

numServers = [1, 3]

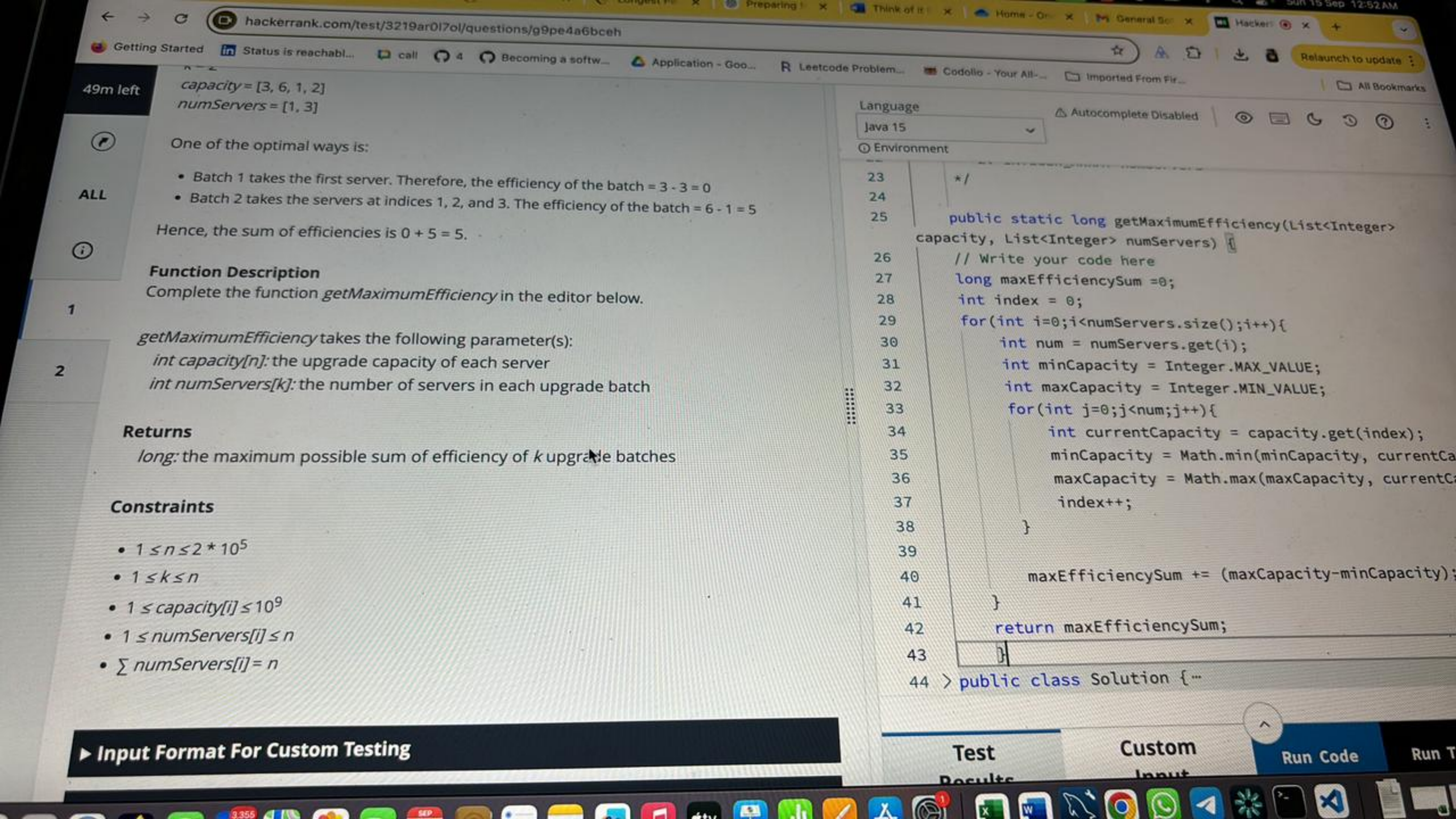
One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

Hence, the sum of efficiencies is 0 + 5 = 5.

Function Description

```
    */
    public static long getMaximumEfficiency(List<Integer>
capacity, List<Integer> numServers) {
        // Write your code here
        long maxEfficiencySum =0;
        int index = 0;
        for(int i=0;i<numServers.size();i++){
            int num = numServers.get(i);
            int minCapacity = Integer.MAX_VALUE;
            int maxCapacity = Integer.MIN_VALUE;
            for(int j=0;j<num;j++){
                int currentCapacity = capacity.get(index);
                minCapacity = Math.min(minCapacity, currentC
                maxCapacity = Math.max(maxCapacity, currentC
                index++;
            }

            maxEfficiencySum += (maxCapacity-minCapacity)
        }
        return maxEfficiencySum;
    }
public class Solution {⋯
```



capacity = [3, 6, 1, 2]
numServers = [1, 3]

One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

Hence, the sum of efficiencies is 0 + 5 = 5.

Function Description

Complete the function *getMaximumEfficiency* in the editor below.

getMaximumEfficiency takes the following parameter(s):
int capacity[n]: the upgrade capacity of each server
int numServers[k]: the number of servers in each upgrade batch

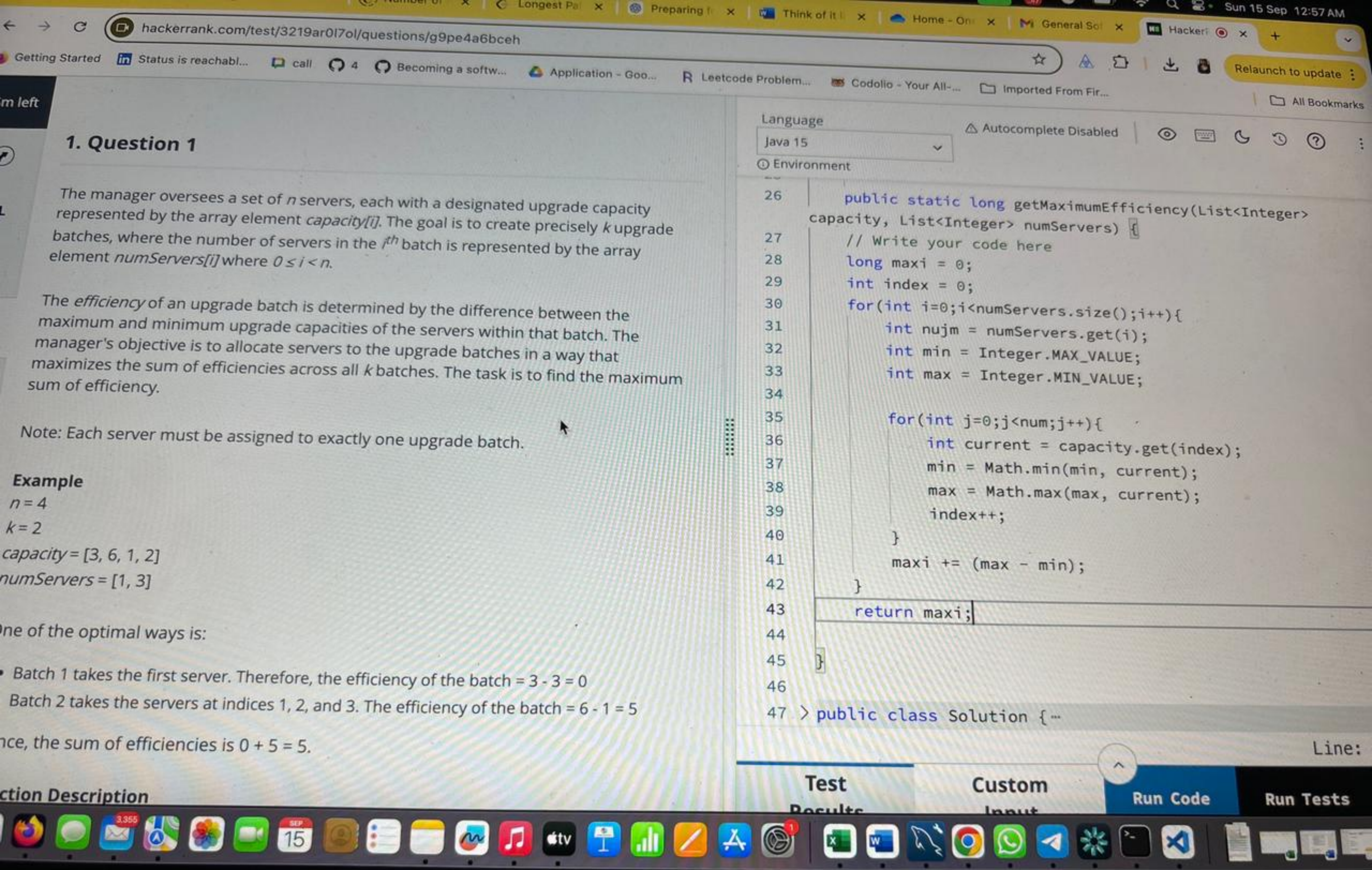
Returns

long: the maximum possible sum of efficiency of *k* upgrade batches

Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le k \le n$
- $1 \le capacity[i] \le 10^9$
- $1 \le numServers[i] \le n$
- $\sum numServers[i] = n$

```
    */

    public static long getMaximumEfficiency(List<Integer>
capacity, List<Integer> numServers) {
        // Write your code here
        long maxEfficiencySum =0;
        int index = 0;
        for(int i=0;i<numServers.size();i++){
            int num = numServers.get(i);
            int minCapacity = Integer.MAX_VALUE;
            int maxCapacity = Integer.MIN_VALUE;
            for(int j=0;j<num;j++){
                int currentCapacity = capacity.get(index);
                minCapacity = Math.min(minCapacity, currentCa
                maxCapacity = Math.max(maxCapacity, currentC
                index++;
            }

            maxEfficiencySum += (maxCapacity-minCapacity);
        }
        return maxEfficiencySum;
    }
public class Solution {⋯
```



1. Question 1

The manager oversees a set of *n* servers, each with a designated upgrade capacity represented by the array element *capacity[i]*. The goal is to create precisely *k* upgrade batches, where the number of servers in the i^{th} batch is represented by the array element *numServers[i]* where $0 \le i < n$.

The *efficiency* of an upgrade batch is determined by the difference between the maximum and minimum upgrade capacities of the servers within that batch. The manager's objective is to allocate servers to the upgrade batches in a way that maximizes the sum of efficiencies across all *k* batches. The task is to find the maximum sum of efficiency.

Note: Each server must be assigned to exactly one upgrade batch.

Example

n = 4
k = 2
capacity = [3, 6, 1, 2]
numServers = [1, 3]

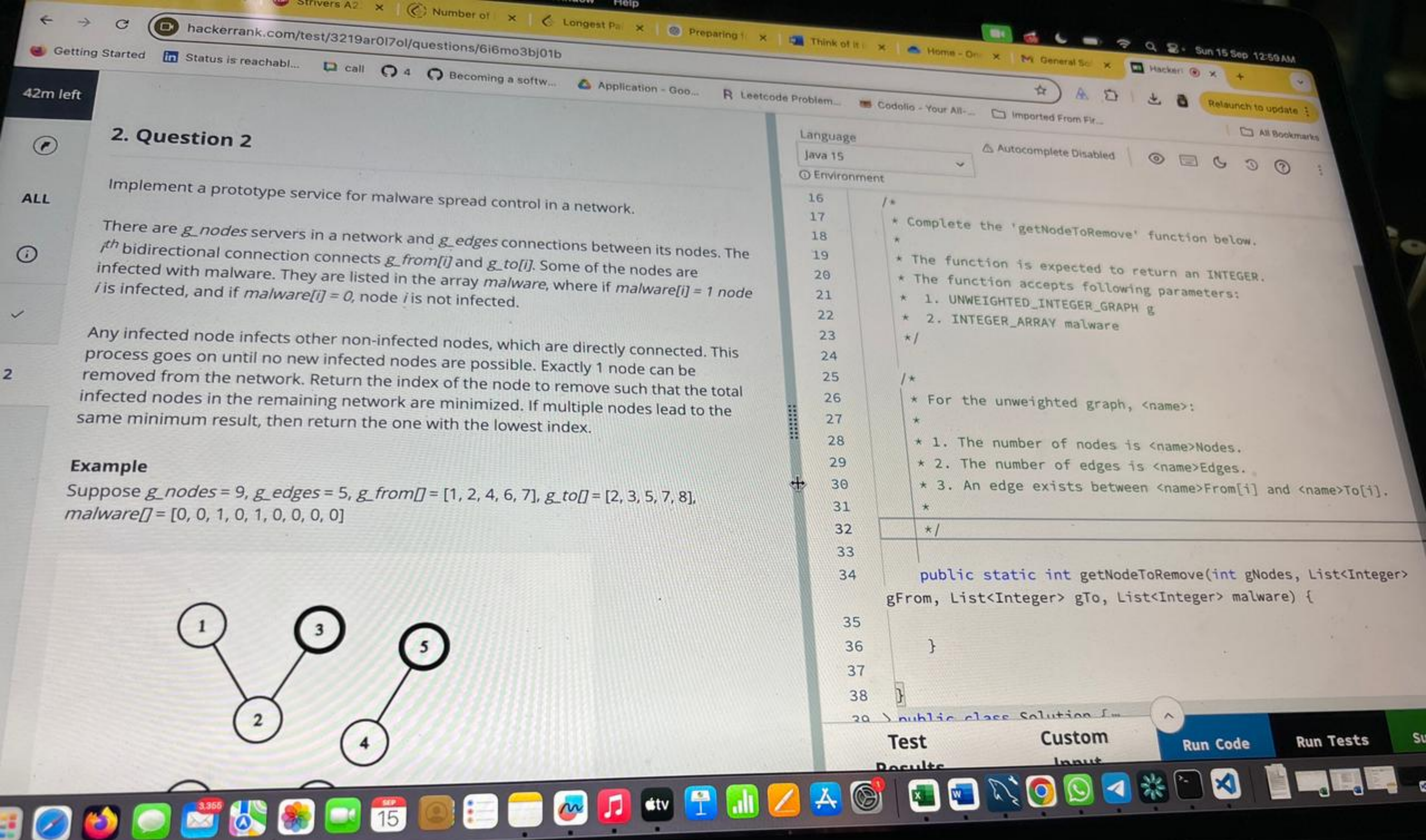
ne of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

nce, the sum of efficiencies is 0 + 5 = 5.

ction Description

```
    public static long getMaximumEfficiency(List<Integer>
capacity, List<Integer> numServers) {
        // Write your code here
        long maxi = 0;
        int index = 0;
        for(int i=0;i<numServers.size();i++){
            int nujm = numServers.get(i);
            int min = Integer.MAX_VALUE;
            int max = Integer.MIN_VALUE;

            for(int j=0;j<num;j++){
                int current = capacity.get(index);
                min = Math.min(min, current);
                max = Math.max(max, current);
                index++;
            }
            maxi += (max - min);
        }
        return maxi;

    }

> public class Solution {⋯
```



42m left

2. Question 2

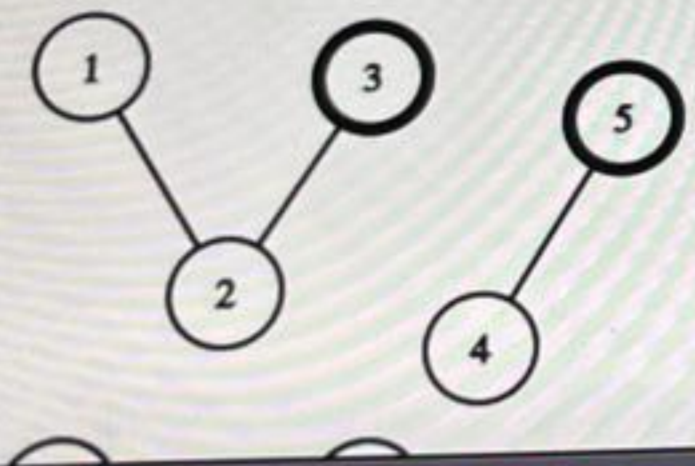
Implement a prototype service for malware spread control in a network.

There are *g_nodes* servers in a network and *g_edges* connections between its nodes. The i^{th} bidirectional connection connects *g_from[i]* and *g_to[i]*. Some of the nodes are infected with malware. They are listed in the array *malware*, where if *malware[i] = 1* node *i* is infected, and if *malware[i] = 0*, node *i* is not infected.

Any infected node infects other non-infected nodes, which are directly connected. This process goes on until no new infected nodes are possible. Exactly 1 node can be removed from the network. Return the index of the node to remove such that the total infected nodes in the remaining network are minimized. If multiple nodes lead to the same minimum result, then return the one with the lowest index.

Example

Suppose *g_nodes* = 9, *g_edges* = 5, *g_from[]* = [1, 2, 4, 6, 7], *g_to[]* = [2, 3, 5, 7, 8], *malware[]* = [0, 0, 1, 0, 1, 0, 0, 0, 0]

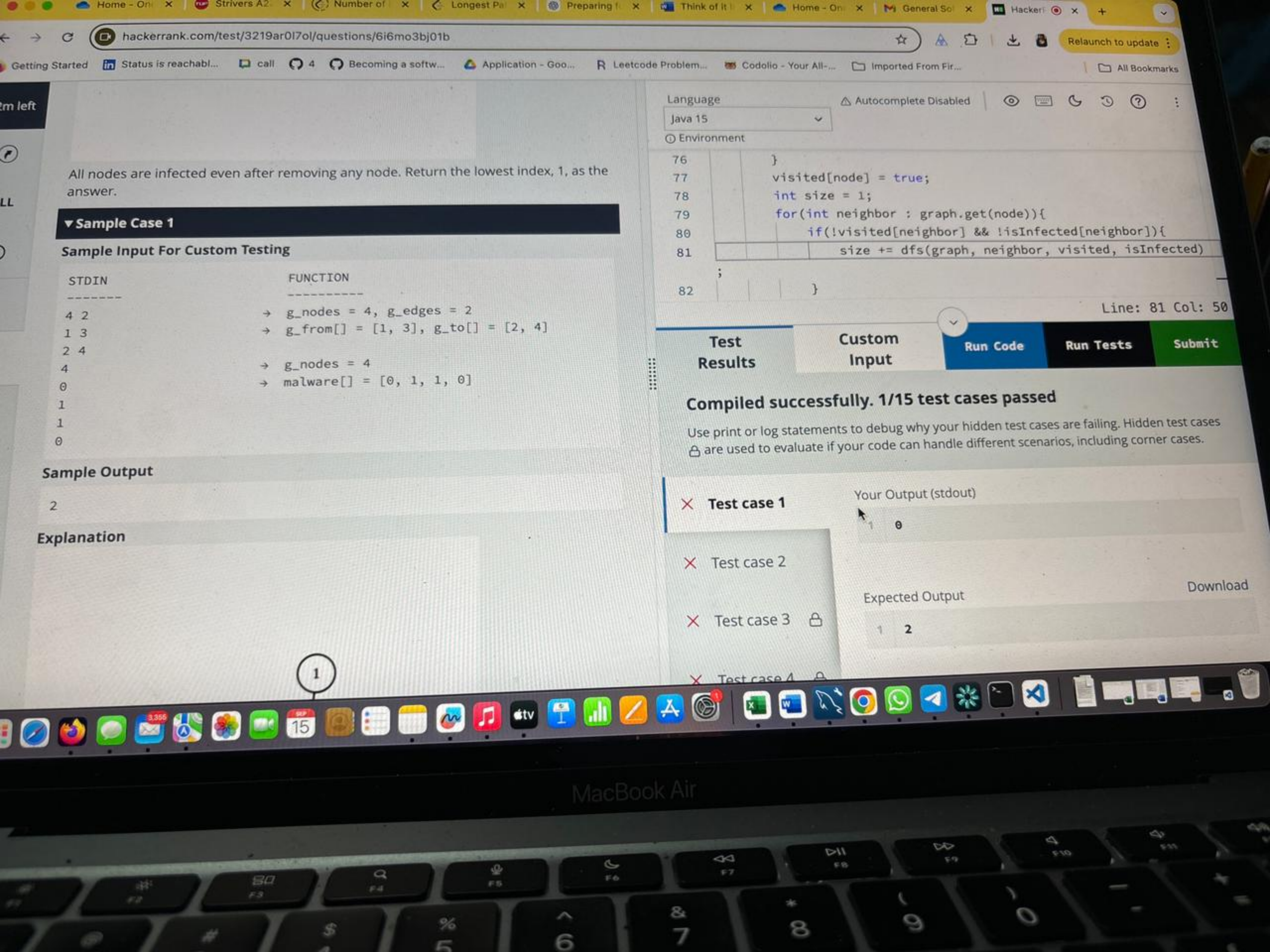


Language: Java 15

Autocomplete Disabled

```
    /*
     * Complete the 'getNodeToRemove' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts following parameters:
     *  1. UNWEIGHTED_INTEGER_GRAPH g
     *  2. INTEGER_ARRAY malware
     */

    /*
     * For the unweighted graph, <name>:
     *
     * 1. The number of nodes is <name>Nodes.
     * 2. The number of edges is <name>Edges.
     * 3. An edge exists between <name>From[i] and <name>To[i].
     *
     */

    public static int getNodeToRemove(int gNodes, List<Integer> gFrom, List<Integer> gTo, List<Integer> malware) {

    }

}
> public class Solution {...
```

Test Results | Custom Input | Run Code | Run Tests



hackerrank.com/test/3219ar0l7ol/questions/6i6mo3bj01b

All nodes are infected even after removing any node. Return the lowest index, 1, as the answer.

▾ Sample Case 1

Sample Input For Custom Testing

```
STDIN                              FUNCTION
-------                            ----------
4 2                            →   g_nodes = 4, g_edges = 2
1 3                            →   g_from[] = [1, 3], g_to[] = [2, 4]
2 4
4                              →   g_nodes = 4
0                              →   malware[] = [0, 1, 1, 0]
1
1
0
```

Sample Output

```
2
```

Explanation

Language: Java 15

Autocomplete Disabled

```java
                }
            visited[node] = true;
            int size = 1;
            for(int neighbor : graph.get(node)){
                if(!visited[neighbor] && !isInfected[neighbor]){
                    size += dfs(graph, neighbor, visited, isInfected)
;
                }
```

Line: 81 Col: 50

Test Results | Custom Input | Run Code | Run Tests | Submit

Compiled successfully. 1/15 test cases passed

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

× **Test case 1**
× Test case 2
× Test case 3
× Test case 4

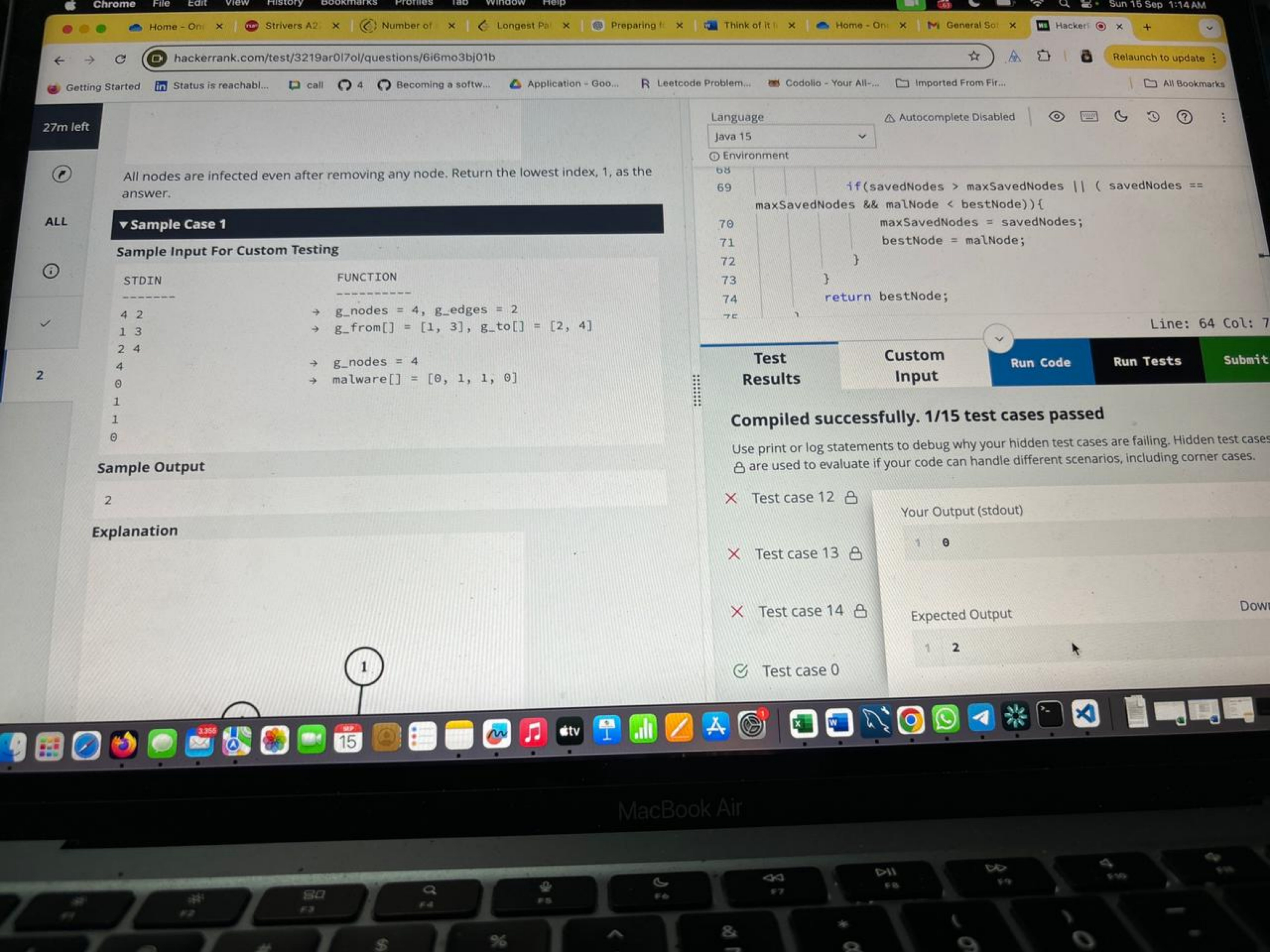
Your Output (stdout)

```
0
```

Expected Output

Download

```
2
```



All nodes are infected even after removing any node. Return the lowest index, 1, as the answer.

▼Sample Case 1

Sample Input For Custom Testing

```
STDIN                         FUNCTION
-------                       ----------
4 2                      →    g_nodes = 4, g_edges = 2
1 3                      →    g_from[] = [1, 3], g_to[] = [2, 4]
2 4
4                        →    g_nodes = 4
0                        →    malware[] = [0, 1, 1, 0]
1
1
0
```

Sample Output

```
2
```

Explanation

```java
            if(savedNodes > maxSavedNodes || ( savedNodes ==
maxSavedNodes && malNode < bestNode)){
                maxSavedNodes = savedNodes;
                bestNode = malNode;
            }
        }
        return bestNode;
```

Compiled successfully. 1/15 test cases passed

Use print or log statements to debug why your hidden test cases are failing. Hidden test cases are used to evaluate if your code can handle different scenarios, including corner cases.

✗ Test case 12
✗ Test case 13
✗ Test case 14
✓ Test case 0

Your Output (stdout)

```
0
```

Expected Output

```
2
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

hackerrank.com/test/3219ar0l7ol/questions/6i6mo3bj01b

27m left

2. Question 2

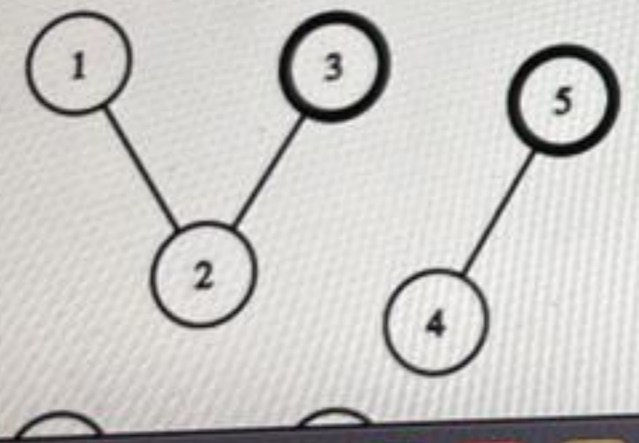
Implement a prototype service for malware spread control in a network.

There are *g_nodes* servers in a network and *g_edges* connections between its nodes. The *i*th bidirectional connection connects *g_from[i]* and *g_to[i]*. Some of the nodes are infected with malware. They are listed in the array *malware*, where if *malware[i] = 1 node i* is infected, and if *malware[i] = 0*, node *i* is not infected.

Any infected node infects other non-infected nodes, which are directly connected. This process goes on until no new infected nodes are possible. Exactly 1 node can be removed from the network. Return the index of the node to remove such that the total infected nodes in the remaining network are minimized. If multiple nodes lead to the same minimum result, then return the one with the lowest index.

Example

Suppose *g_nodes* = 9, *g_edges* = 5, *g_from[]* = [1, 2, 4, 6, 7], *g_to[]* = [2, 3, 5, 7, 8], *malware[]* = [0, 0, 1, 0, 1, 0, 0, 0, 0]

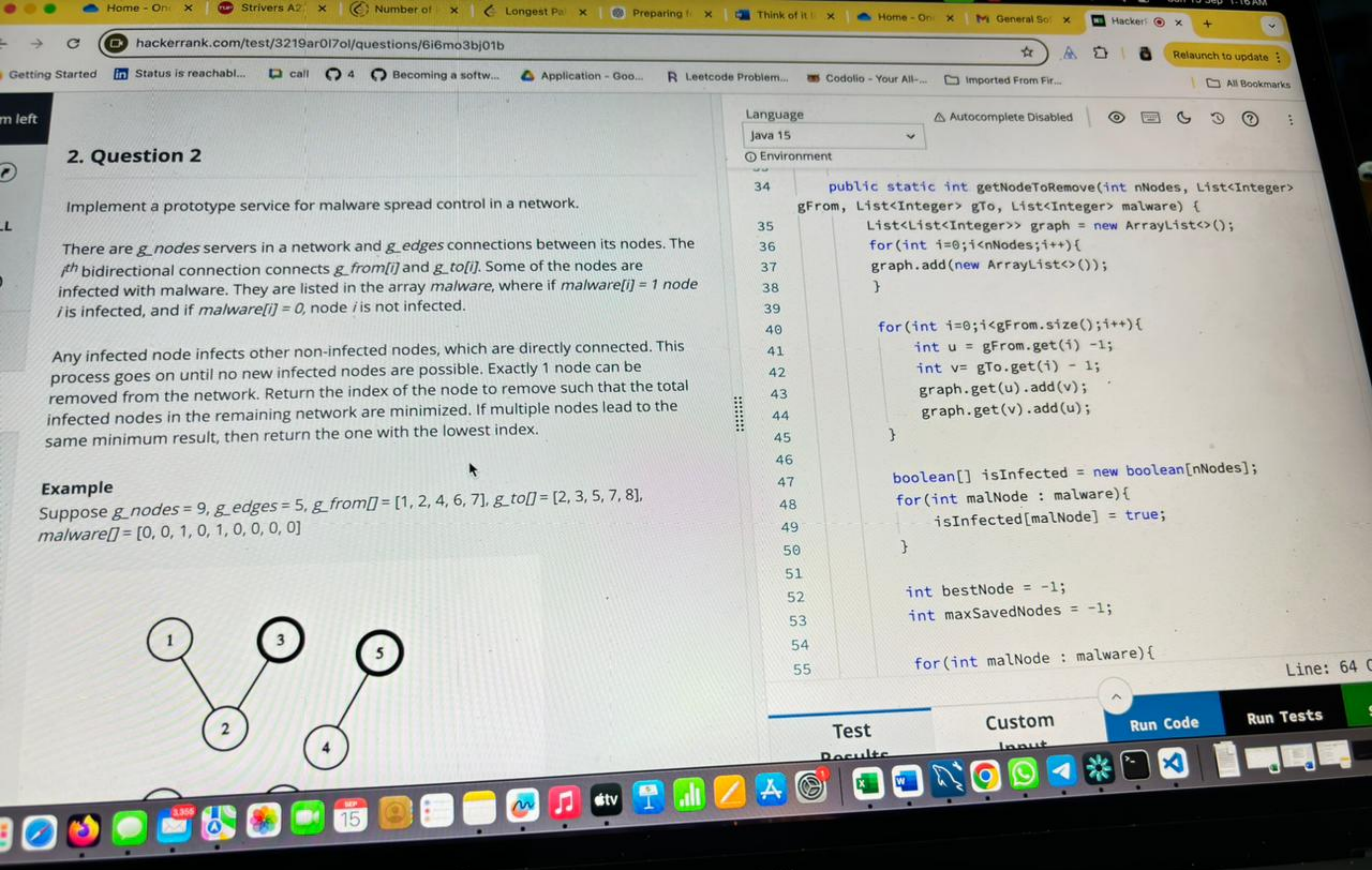


Language: Java 15
Autocomplete Disabled
Environment

```
        //
        
        boolean[] tempInfected = Arrays.copyOf(isInfected, 
nNodes);
        tempInfected[malNode] = false;
        boolean[] visited = new boolean[nNodes];
        int savedNodes = 0;
        for(int i=0;i<nNodes;i++){
            if(!visited[i] && !tempInfected[i]){
                int componentSize = dfs(graph, i, visited, 
tempInfected);
                savedNodes += componentSize;
            }
        }
        
        if(savedNodes > maxSavedNodes || ( savedNodes == 
maxSavedNodes && malNode < bestNode)){
            maxSavedNodes = savedNodes;
            bestNode = malNode;
        }
    }
    return bestNode;
}
```

Line: 64 Col

Test Results | Custom Input | Run Code | Run Tests | Sub



hackerrank.com/test/3219ar0l7ol/questions/6i6mo3bj01b

2. Question 2

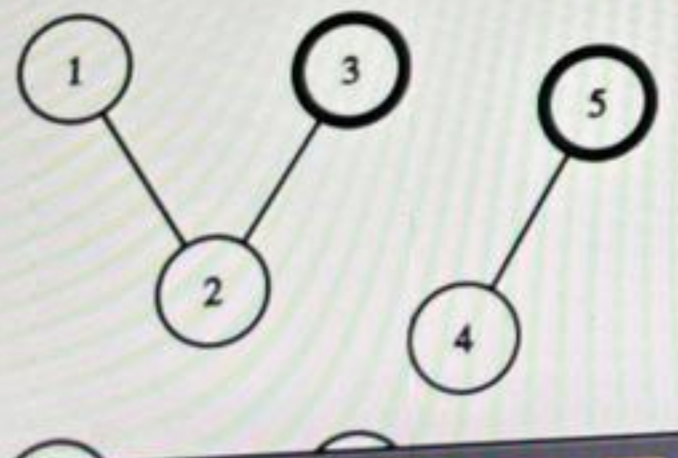
Implement a prototype service for malware spread control in a network.

There are *g_nodes* servers in a network and *g_edges* connections between its nodes. The *i*th bidirectional connection connects *g_from[i]* and *g_to[i]*. Some of the nodes are infected with malware. They are listed in the array *malware*, where if *malware[i] = 1 node i* is infected, and if *malware[i] = 0*, node *i* is not infected.

Any infected node infects other non-infected nodes, which are directly connected. This process goes on until no new infected nodes are possible. Exactly 1 node can be removed from the network. Return the index of the node to remove such that the total infected nodes in the remaining network are minimized. If multiple nodes lead to the same minimum result, then return the one with the lowest index.

Example

Suppose *g_nodes* = 9, *g_edges* = 5, *g_from[]* = [1, 2, 4, 6, 7], *g_to[]* = [2, 3, 5, 7, 8], *malware[]* = [0, 0, 1, 0, 1, 0, 0, 0, 0]



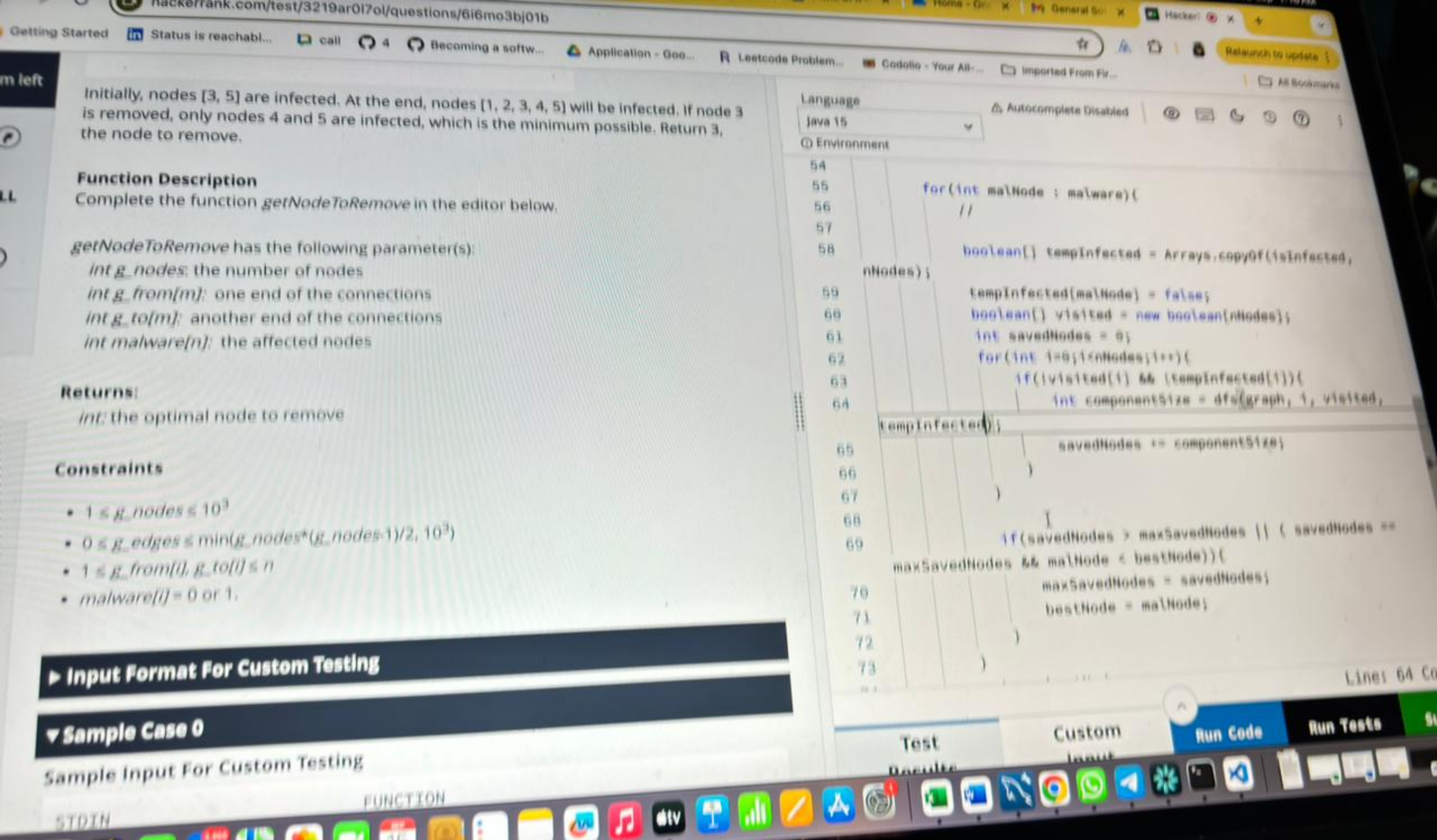
Language: Java 15

Autocomplete Disabled

```
    public static int getNodeToRemove(int nNodes, List<Integer>
gFrom, List<Integer> gTo, List<Integer> malware) {
        List<List<Integer>> graph = new ArrayList<>();
        for(int i=0;i<nNodes;i++){
        graph.add(new ArrayList<>());
        }

        for(int i=0;i<gFrom.size();i++){
            int u = gFrom.get(i) -1;
            int v= gTo.get(i) - 1;
            graph.get(u).add(v);
            graph.get(v).add(u);
        }

        boolean[] isInfected = new boolean[nNodes];
        for(int malNode : malware){
            isInfected[malNode] = true;
        }

        int bestNode = -1;
        int maxSavedNodes = -1;

        for(int malNode : malware){
```

Line: 64

Test Results | Custom Input | Run Code | Run Tests



Initially, nodes [3, 5] are infected. At the end, nodes [1, 2, 3, 4, 5] will be infected. If node 3 is removed, only nodes 4 and 5 are infected, which is the minimum possible. Return 3, the node to remove.

Function Description

Complete the function *getNodeToRemove* in the editor below.

getNodeToRemove has the following parameter(s):

int g_nodes: the number of nodes
int g_from[m]: one end of the connections
int g_to[m]: another end of the connections
int malware[n]: the affected nodes

Returns:

int: the optimal node to remove

Constraints

- $1 \le g\_nodes \le 10^3$
- $0 \le g\_edges \le \min(g\_nodes*(g\_nodes-1)/2, 10^3)$
- $1 \le g\_from[i], g\_to[i] \le n$
- $malware[i] = 0$ or 1.

▶ Input Format For Custom Testing

▼ Sample Case 0

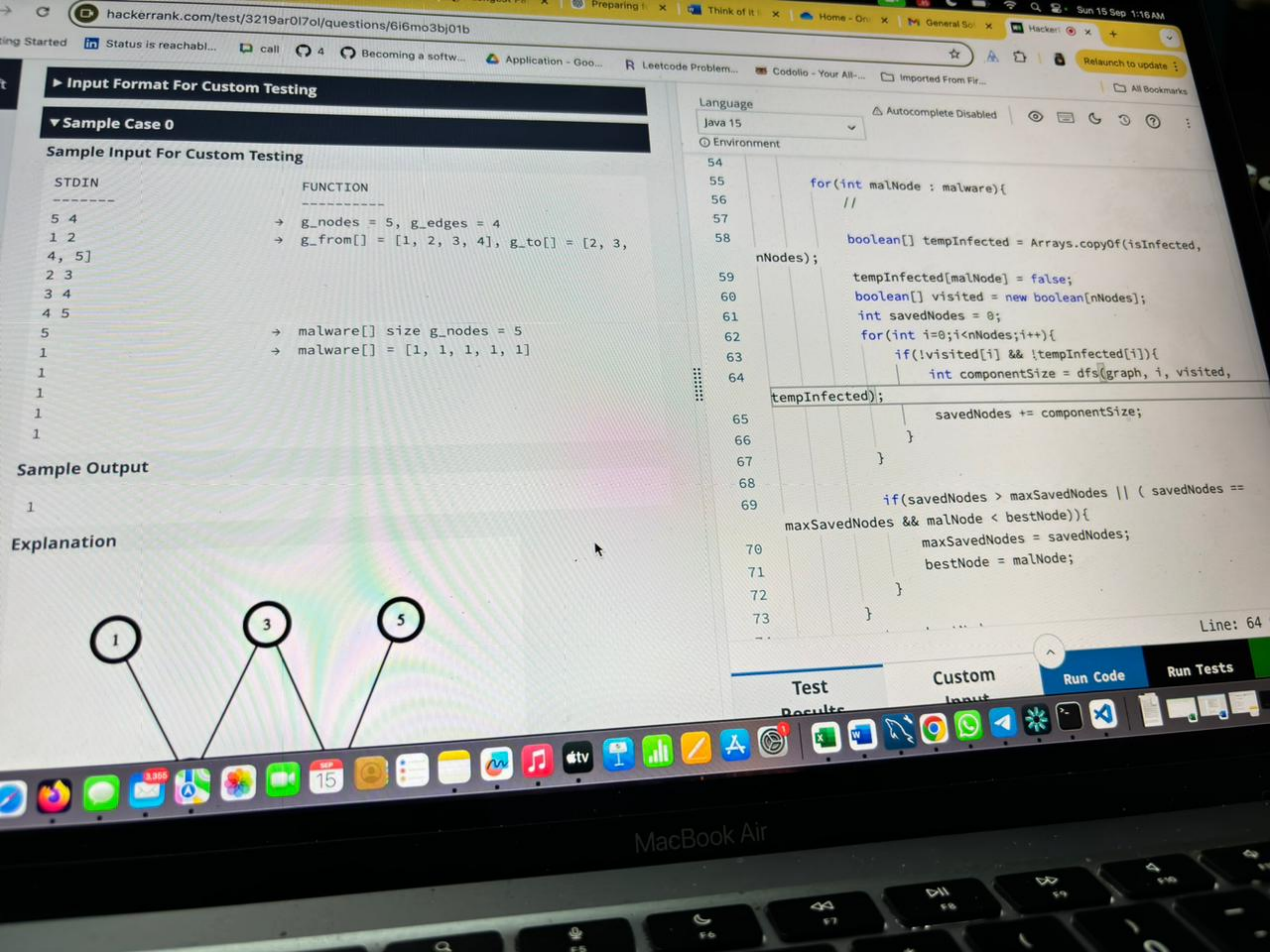
Sample Input For Custom Testing

STDIN FUNCTION

Language: Java 15

```
        for(int malNode : malware){
            //

            boolean[] tempInfected = Arrays.copyOf(isInfected,
nNodes);
            tempInfected[malNode] = false;
            boolean[] visited = new boolean[nNodes];
            int savedNodes = 0;
            for(int i=0;i<nNodes;i++){
                if(!visited[i] && !tempInfected[i]){
                    int componentSize = dfs(graph, i, visited,
tempInfected);
                    savedNodes += componentSize;
                }
            }

            if(savedNodes > maxSavedNodes || ( savedNodes ==
maxSavedNodes && malNode < bestNode)){
                maxSavedNodes = savedNodes;
                bestNode = malNode;
            }
        }
```

Line: 64



hackerrank.com/test/3219ar0l7ol/questions/6i6mo3bj01b

▸ Input Format For Custom Testing

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN          FUNCTION
-------        ----------
5 4         →  g_nodes = 5, g_edges = 4
1 2         →  g_from[] = [1, 2, 3, 4], g_to[] = [2, 3,
4, 5]
2 3
3 4
4 5
5           →  malware[] size g_nodes = 5
1           →  malware[] = [1, 1, 1, 1, 1]
1
1
1
1
```

Sample Output

```
1
```

Explanation

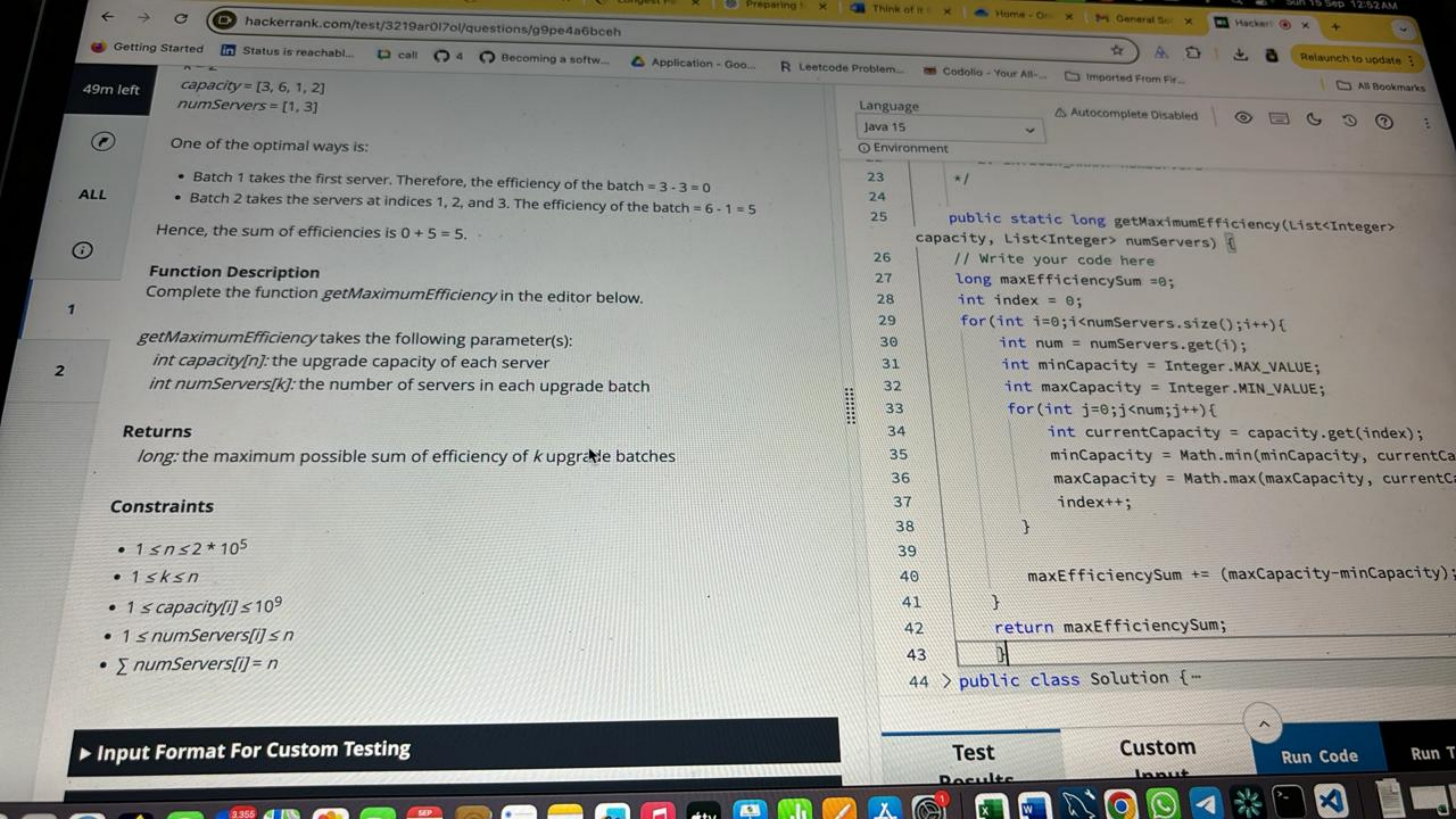
Language: Java 15

Autocomplete Disabled

```java
            for(int malNode : malware){
                //

                boolean[] tempInfected = Arrays.copyOf(isInfected, nNodes);
                tempInfected[malNode] = false;
                boolean[] visited = new boolean[nNodes];
                int savedNodes = 0;
                for(int i=0;i<nNodes;i++){
                    if(!visited[i] && !tempInfected[i]){
                        int componentSize = dfs(graph, i, visited, tempInfected);
                        savedNodes += componentSize;
                    }
                }

                if(savedNodes > maxSavedNodes || ( savedNodes == maxSavedNodes && malNode < bestNode)){
                    maxSavedNodes = savedNodes;
                    bestNode = malNode;
                }
            }
```

Line: 64

Test Results | Custom Input | Run Code | Run Tests



hackerrank.com/test/3219ar0l7ol/questions/g9pe4a6bceh

49m left

capacity = [3, 6, 1, 2]
numServers = [1, 3]

One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

Hence, the sum of efficiencies is 0 + 5 = 5.

Function Description

Complete the function *getMaximumEfficiency* in the editor below.

getMaximumEfficiency takes the following parameter(s):
int capacity[n]: the upgrade capacity of each server
int numServers[k]: the number of servers in each upgrade batch

Returns

long: the maximum possible sum of efficiency of *k* upgrade batches

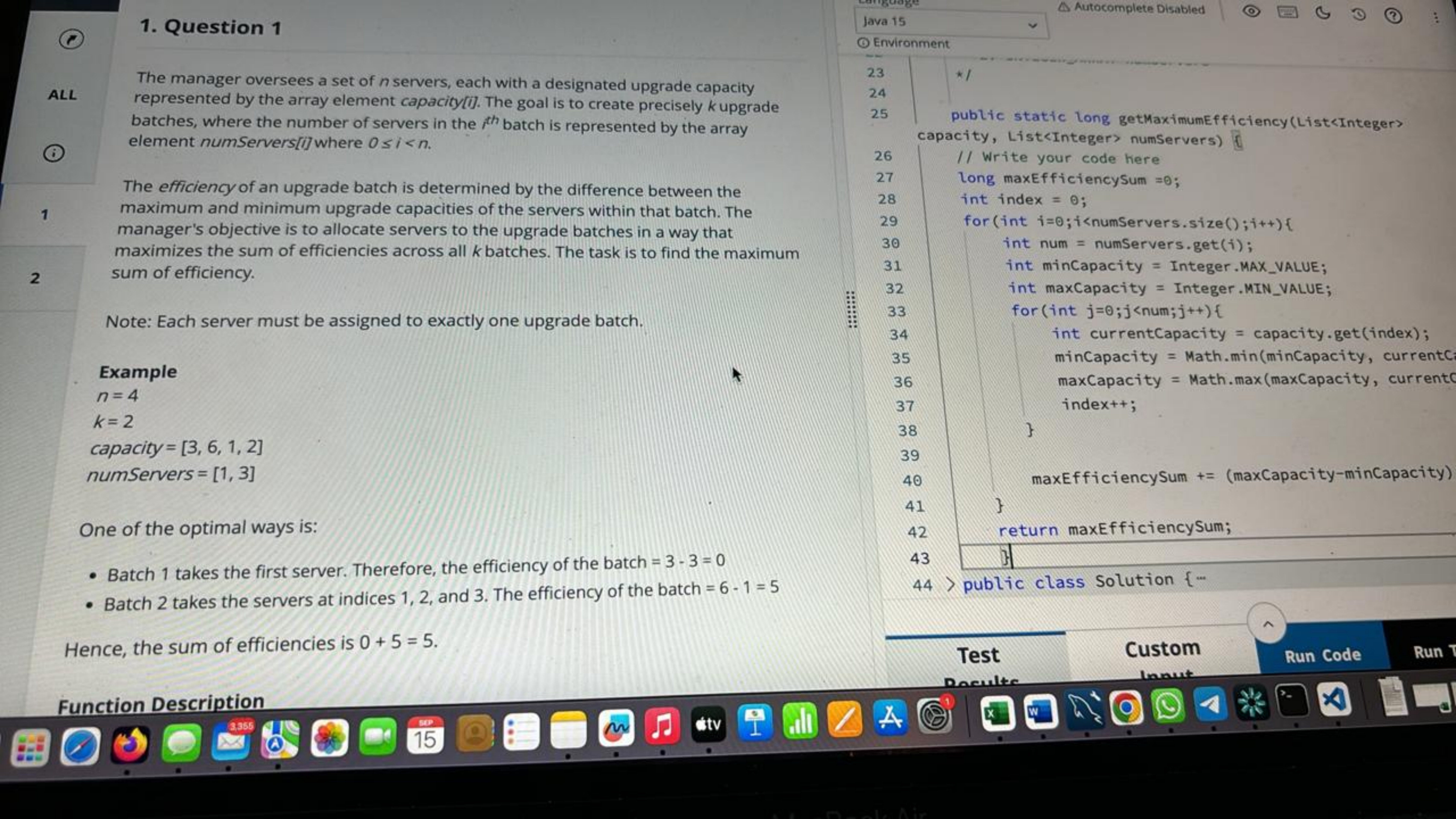
Constraints

- $1 \le n \le 2 * 10^5$
- $1 \le k \le n$
- $1 \le capacity[i] \le 10^9$
- $1 \le numServers[i] \le n$
- $\sum numServers[i] = n$

▶ **Input Format For Custom Testing**

Language: Java 15

```java
     */

    public static long getMaximumEfficiency(List<Integer>
capacity, List<Integer> numServers) {
    // Write your code here
    long maxEfficiencySum =0;
    int index = 0;
    for(int i=0;i<numServers.size();i++){
        int num = numServers.get(i);
        int minCapacity = Integer.MAX_VALUE;
        int maxCapacity = Integer.MIN_VALUE;
        for(int j=0;j<num;j++){
            int currentCapacity = capacity.get(index);
            minCapacity = Math.min(minCapacity, currentCa
            maxCapacity = Math.max(maxCapacity, currentC
            index++;
        }

        maxEfficiencySum += (maxCapacity-minCapacity);
    }
    return maxEfficiencySum;
    }
> public class Solution {…
```



1. Question 1

ALL

1

2

The manager oversees a set of *n* servers, each with a designated upgrade capacity represented by the array element *capacity[i]*. The goal is to create precisely *k* upgrade batches, where the number of servers in the i^{th} batch is represented by the array element *numServers[i]* where $0 \le i < n$.

The *efficiency* of an upgrade batch is determined by the difference between the maximum and minimum upgrade capacities of the servers within that batch. The manager's objective is to allocate servers to the upgrade batches in a way that maximizes the sum of efficiencies across all *k* batches. The task is to find the maximum sum of efficiency.

Note: Each server must be assigned to exactly one upgrade batch.

Example

$n = 4$

$k = 2$

capacity = [3, 6, 1, 2]

numServers = [1, 3]

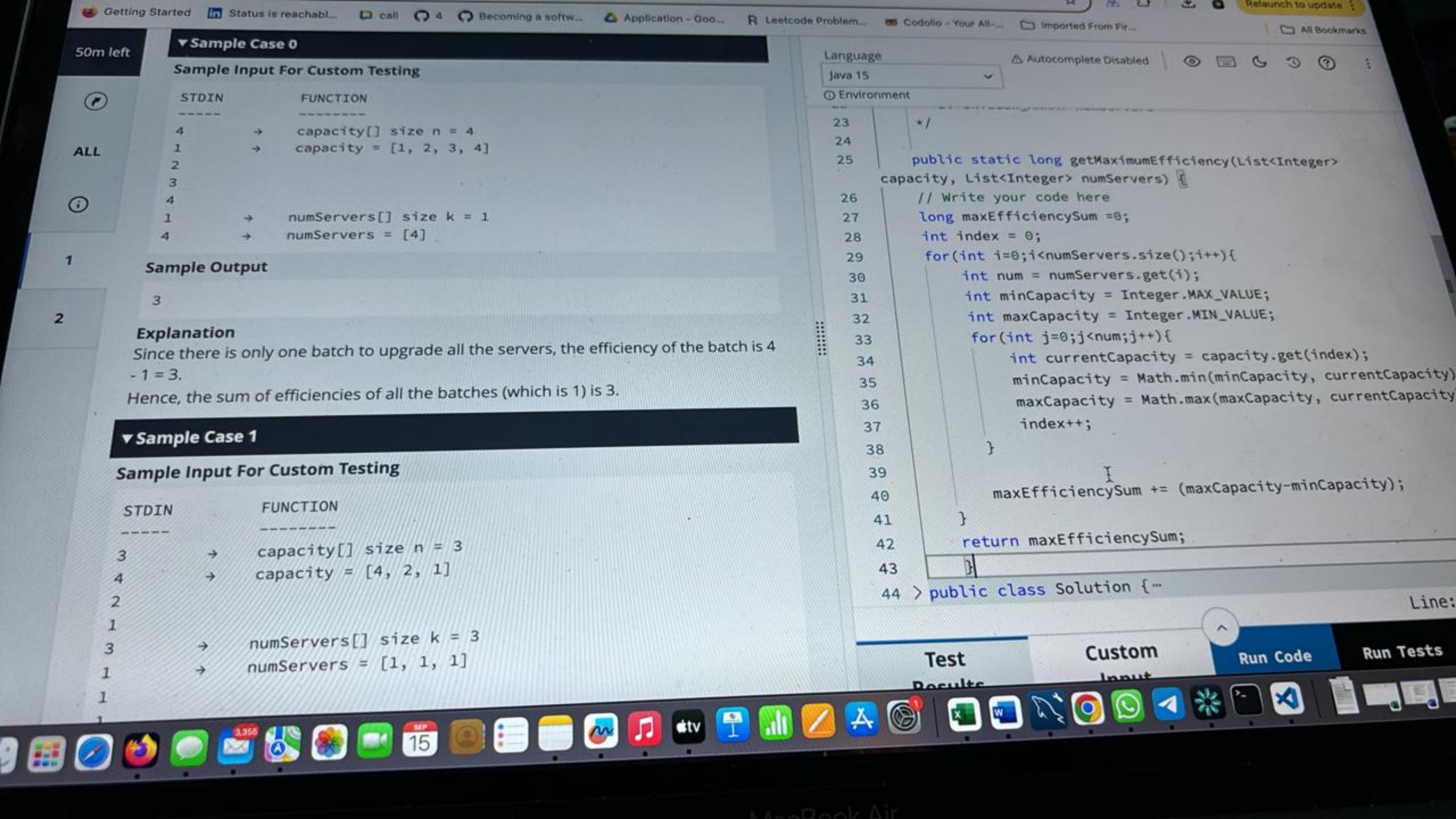
One of the optimal ways is:

- Batch 1 takes the first server. Therefore, the efficiency of the batch = 3 - 3 = 0
- Batch 2 takes the servers at indices 1, 2, and 3. The efficiency of the batch = 6 - 1 = 5

Hence, the sum of efficiencies is 0 + 5 = 5.

Function Description

```java
     */

    public static long getMaximumEfficiency(List<Integer>
capacity, List<Integer> numServers) {
     // Write your code here
     long maxEfficiencySum =0;
     int index = 0;
     for(int i=0;i<numServers.size();i++){
         int num = numServers.get(i);
         int minCapacity = Integer.MAX_VALUE;
         int maxCapacity = Integer.MIN_VALUE;
         for(int j=0;j<num;j++){
             int currentCapacity = capacity.get(index);
             minCapacity = Math.min(minCapacity, currentC
             maxCapacity = Math.max(maxCapacity, currentC
             index++;

         }

         maxEfficiencySum += (maxCapacity-minCapacity)
     }
     return maxEfficiencySum;
    }
public class Solution {⋯
```



50m left

ALL

1

2

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
4      →      capacity[] size n = 4
1      →      capacity = [1, 2, 3, 4]
2
3
4
1      →      numServers[] size k = 1
4      →      numServers = [4]
```

Sample Output

```
3
```

Explanation

Since there is only one batch to upgrade all the servers, the efficiency of the batch is 4 - 1 = 3.

Hence, the sum of efficiencies of all the batches (which is 1) is 3.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         FUNCTION
-----         --------
3      →      capacity[] size n = 3
4      →      capacity = [4, 2, 1]
2
1
3      →      numServers[] size k = 3
1      →      numServers = [1, 1, 1]
1
```

Language: Java 15

Autocomplete Disabled

Environment

```java
     */

    public static long getMaximumEfficiency(List<Integer> capacity, List<Integer> numServers) {
    // Write your code here
    long maxEfficiencySum =0;
    int index = 0;
    for(int i=0;i<numServers.size();i++){
        int num = numServers.get(i);
        int minCapacity = Integer.MAX_VALUE;
        int maxCapacity = Integer.MIN_VALUE;
        for(int j=0;j<num;j++){
            int currentCapacity = capacity.get(index);
            minCapacity = Math.min(minCapacity, currentCapacity)
            maxCapacity = Math.max(maxCapacity, currentCapacity
            index++;
        }

        maxEfficiencySum += (maxCapacity-minCapacity);
    }
    return maxEfficiencySum;
    }
> public class Solution {⋯
```

Line: