**Linux Command-Line Learning**

**1. Arrays in Bash**

Arrays store multiple values in a single variable.

**Declaration:**

# Indexed array

arr=("Pratik" "Mithi" 1)

# Accessing elements

echo ${arr[0]} # Output: Pratik

echo ${arr[1]} # Output: Mithi

echo ${arr[2]} # Output: 1

# Length of array

echo ${#arr[@]} # Output: 3

# Iterate through array

for i in "${arr[@]}"; do

echo "$i"

done

**Key Takeaways:**

* **Indexing starts from 0.**
* Use "${arr[@]}" to iterate through all elements.
* **${#arr[@]}** gives the array length.

**2. Conditional Structures: [] vs. [[]]**

**[] (POSIX-compliant)**

Used in traditional shell scripting but has limitations.

str="hello"

if [ "$str" = "hello" ]; then

echo "Match"

fi

**[[]] (Bash-specific)**

Provides advanced features like pattern matching and logical expressions.

str="hello"

if [[ "$str" == h\* ]]; then

echo "Pattern Match"

fi

**Key Differences:**

| **Feature** | **[] (POSIX)** | **[[]] (Bash)** |
| --- | --- | --- |
| Pattern Matching | No | Yes (== h\*) |
| Logical Operators | -a and -o | && and ` |
| Syntax Strictness | High | More Flexible |

**💡 Tip:** Prefer [[ ]] for modern Bash scripting due to its flexibility and readability.

**3. File Test Operators**

Used to check file properties like existence, permissions, and type.

**Common File Test Flags:**

| **Flag** | **Description** | **Example** |
| --- | --- | --- |
| -e | Check if the file exists | [ -e file.txt ] |
| -d | Check if it's a directory | [ -d mydir ] |
| -f | Check if it's a file | [ -f file.txt ] |
| -r | Check read permission | [ -r file.txt ] |
| -w | Check write permission | [ -w file.txt ] |
| -x | Check execute permission | [ -x script.sh ] |
| -s | Check if file is non-empty | [ -s file.txt ] |

**Example:**

file="example.txt"

if [ -f "$file" ]; then

echo "$file exists and is a regular file"

fi

**4. Understanding su vs. sudo**

| **Feature** | **su (Switch User)** | **sudo (Superuser Do)** |
| --- | --- | --- |
| Purpose | Switch to another user (root) | Run commands as another user |
| Authentication | Needs target user password | Needs current user password |
| Scope | Full root shell | Single command |
| Usage | su - or su username | sudo command |

**Example:**

# Switch to root user

su -

# Run command as root without full shell access

sudo apt update

**5. read Command and Its Flags**

The read command takes user input from the terminal.

**Basic Usage:**

read -p "Enter your name: " name

echo "Hello, $name!"

**Common Flags:**

| **Flag** | **Description** | **Example** |
| --- | --- | --- |
| -p | Display prompt text | read -p "Enter name: " name |
| -s | Silent mode (hide input) | read -s password |
| -t | Set a timeout for input | read -t 5 input |
| -n | Read specific number of chars | read -n 3 chars |

**6. case Statement**

Used for multi-way branching based on pattern matching.

**Example:**

read -p "Enter option [1-3]: " choice

case $choice in

1) echo "Checking account";;

2) echo "Savings account";;

3) echo "Current account";;

\*) echo "Invalid option";;

esac

**7. Regular Expressions in Bash**

**Basic Patterns:**

| **Pattern** | **Description** | **Example Match** |
| --- | --- | --- |
| . | Any single character | a.b matches acb |
| \* | Zero or more occurrences | ab\* matches ab or abb |
| ^ | Start of line | ^Hello matches Hello at the beginning |
| $ | End of line | World$ matches World at the end |
| [a-z] | Range of characters | [a-c] matches a, b, c |
| \d | Digit (in extended regex) | \d matches 0-9 |

**Example with grep:**

# Find lines ending with 'good'

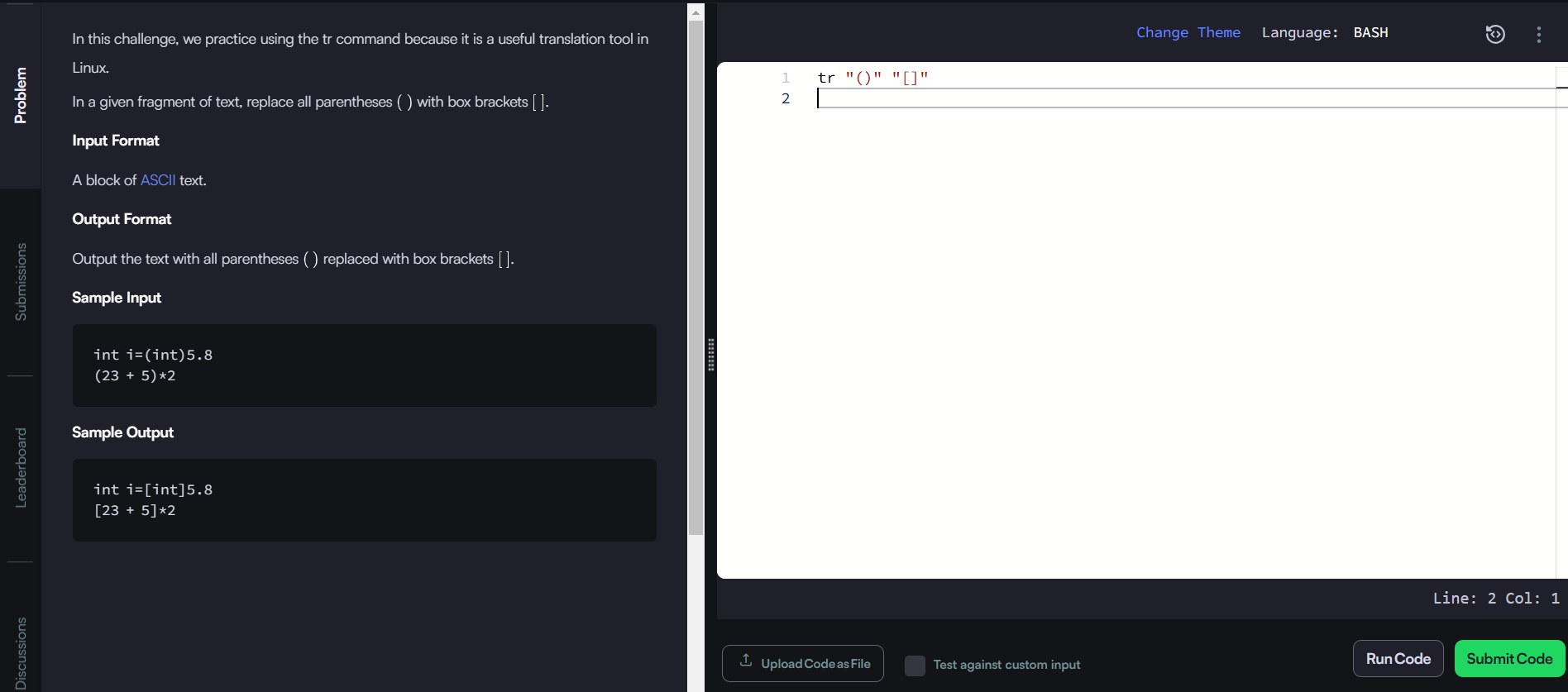
grep "good$" file.txt

# Find emails in a text file

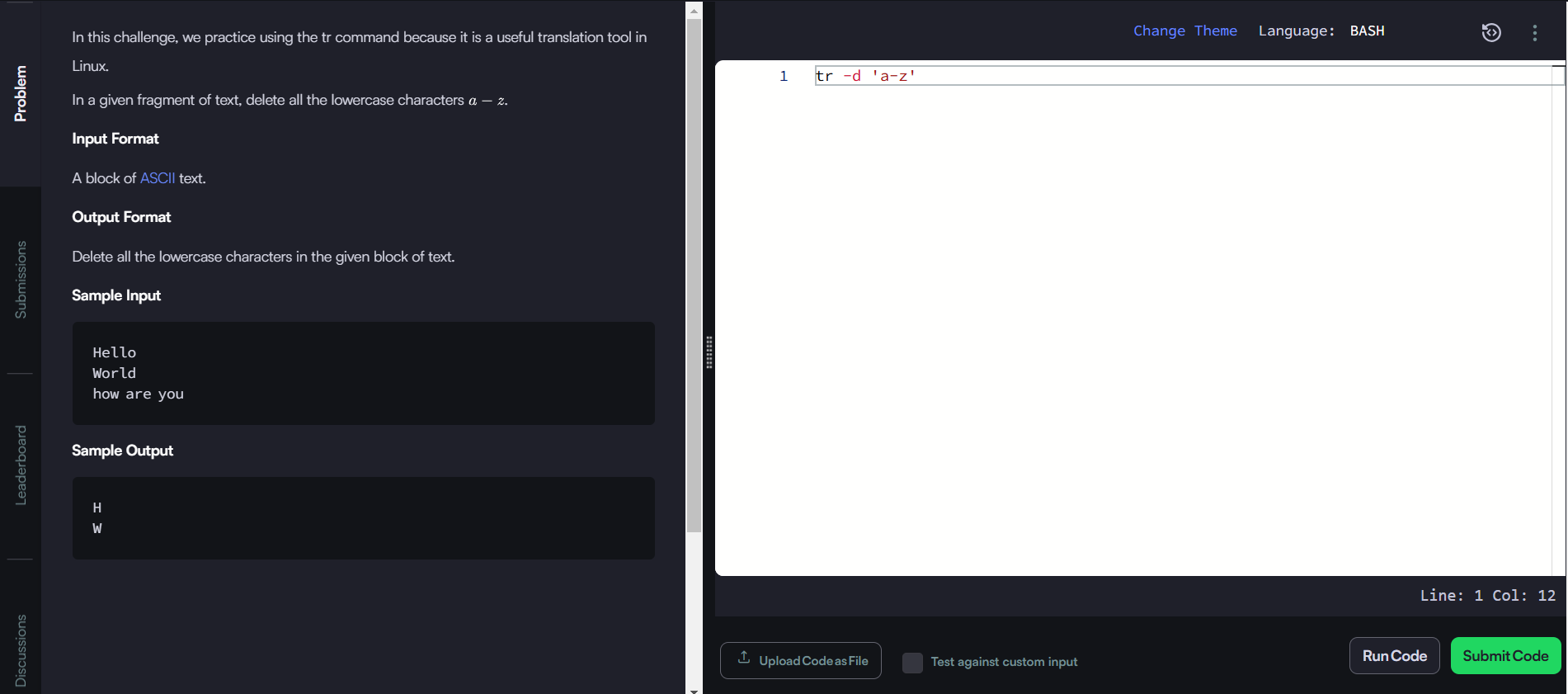
grep -E "[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" sample.txt

**8. Problem-Solving Exercises**

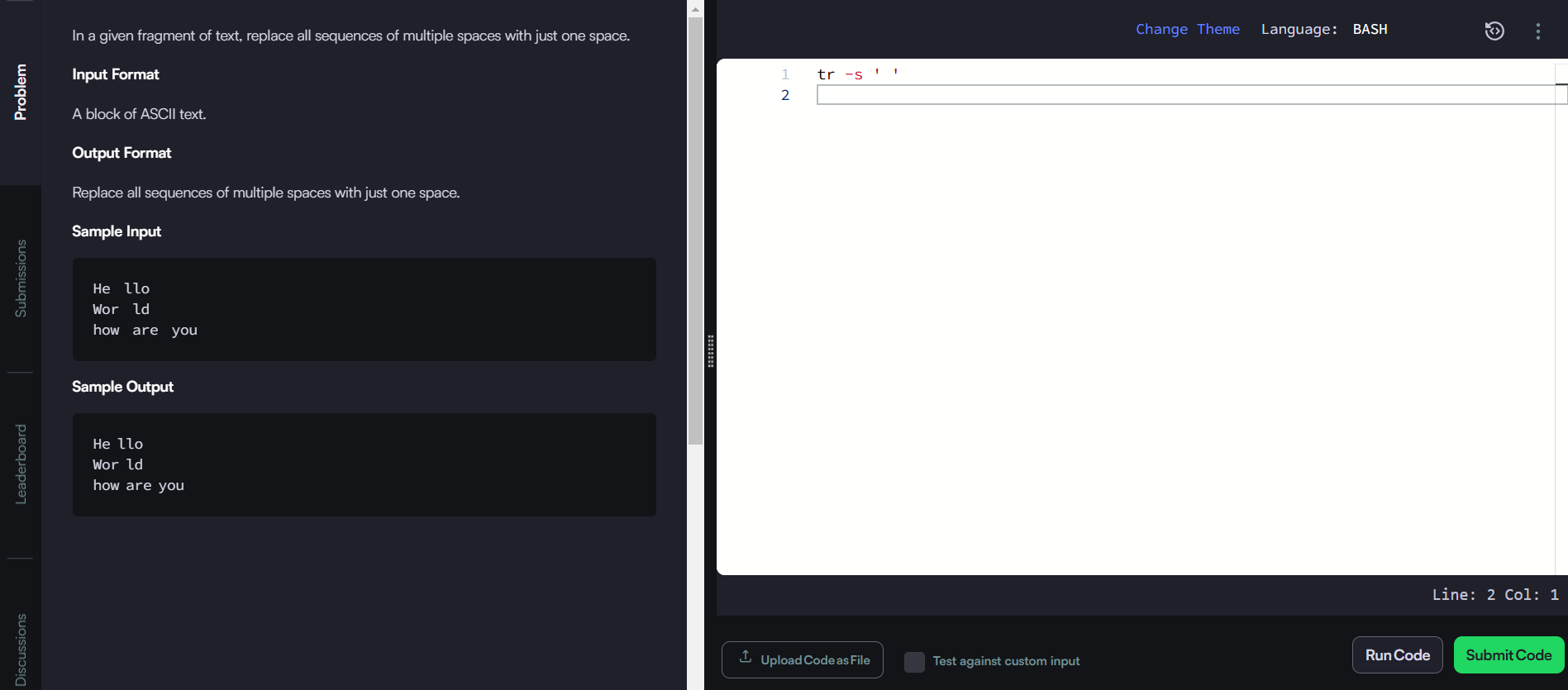
1. <https://www.hackerrank.com/challenges/text-processing-tr-1/problem?isFullScreen=true>



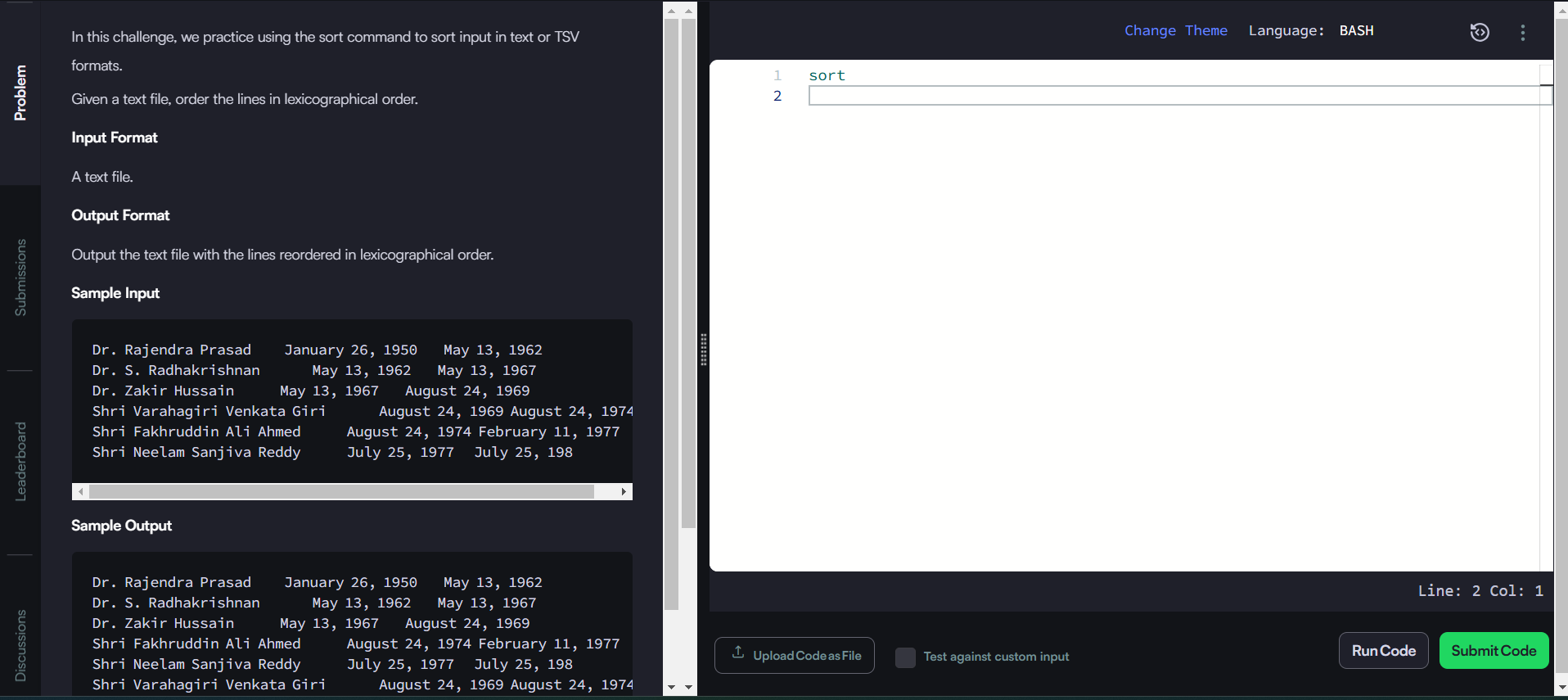
2. <https://www.hackerrank.com/challenges/text-processing-tr-2/problem?isFullScreen=true>



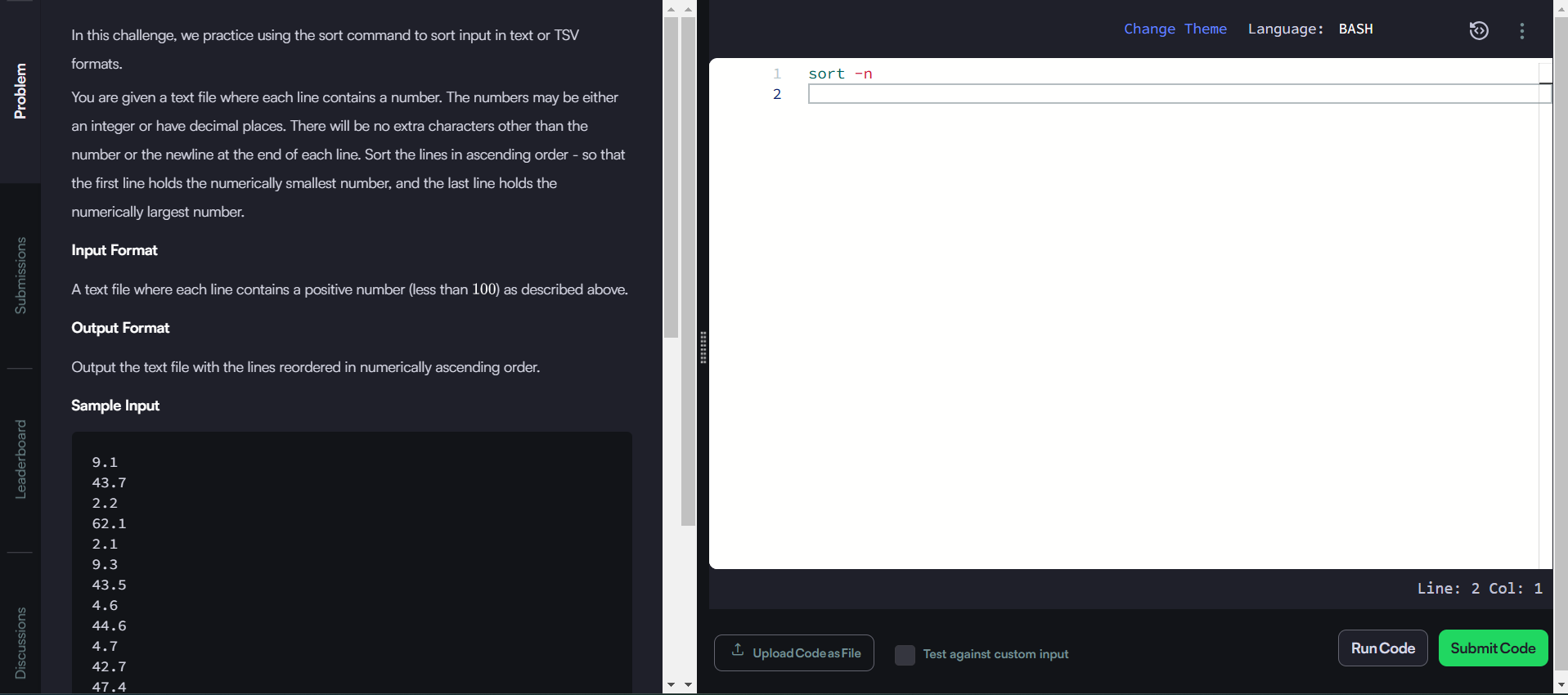
3. <https://www.hackerrank.com/challenges/text-processing-tr-3/problem?isFullScreen=true>



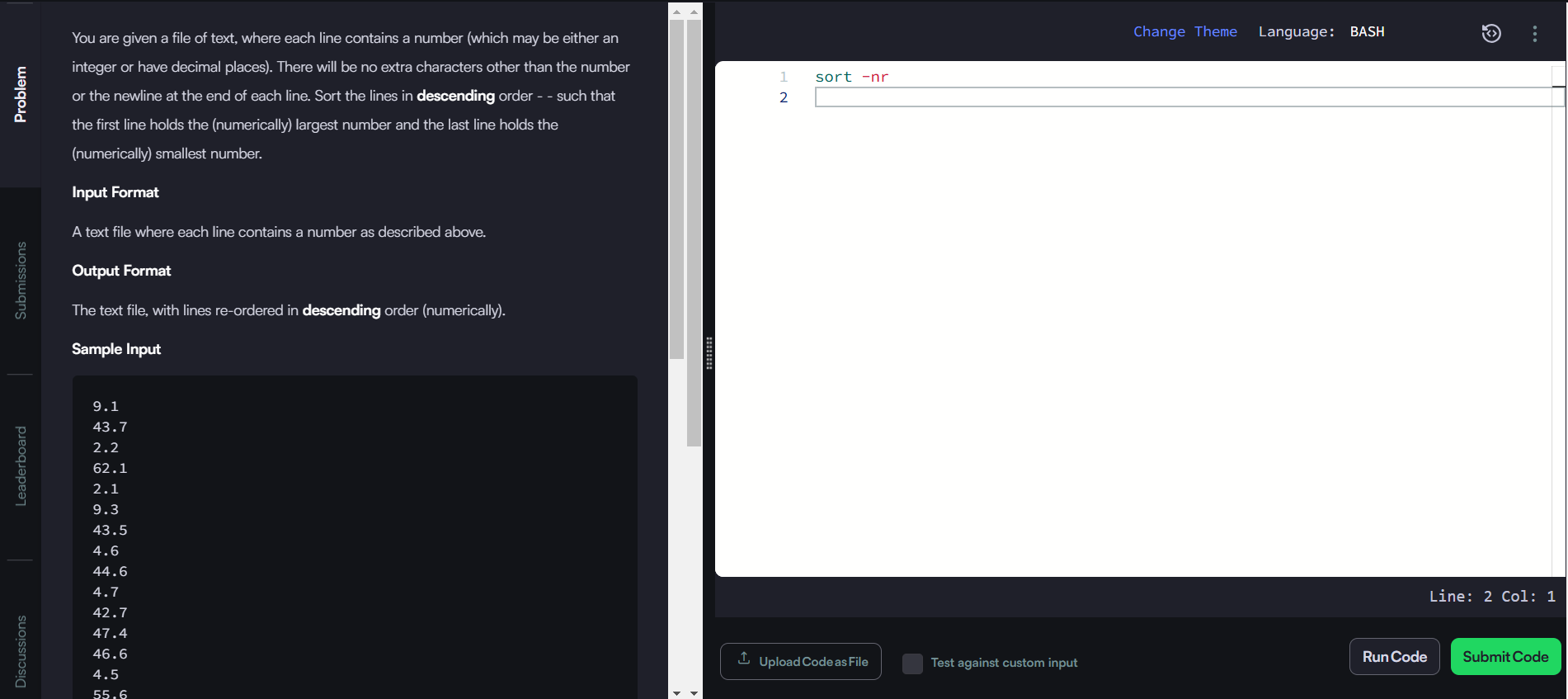
4. <https://www.hackerrank.com/challenges/text-processing-sort-1/problem?isFullScreen=true>



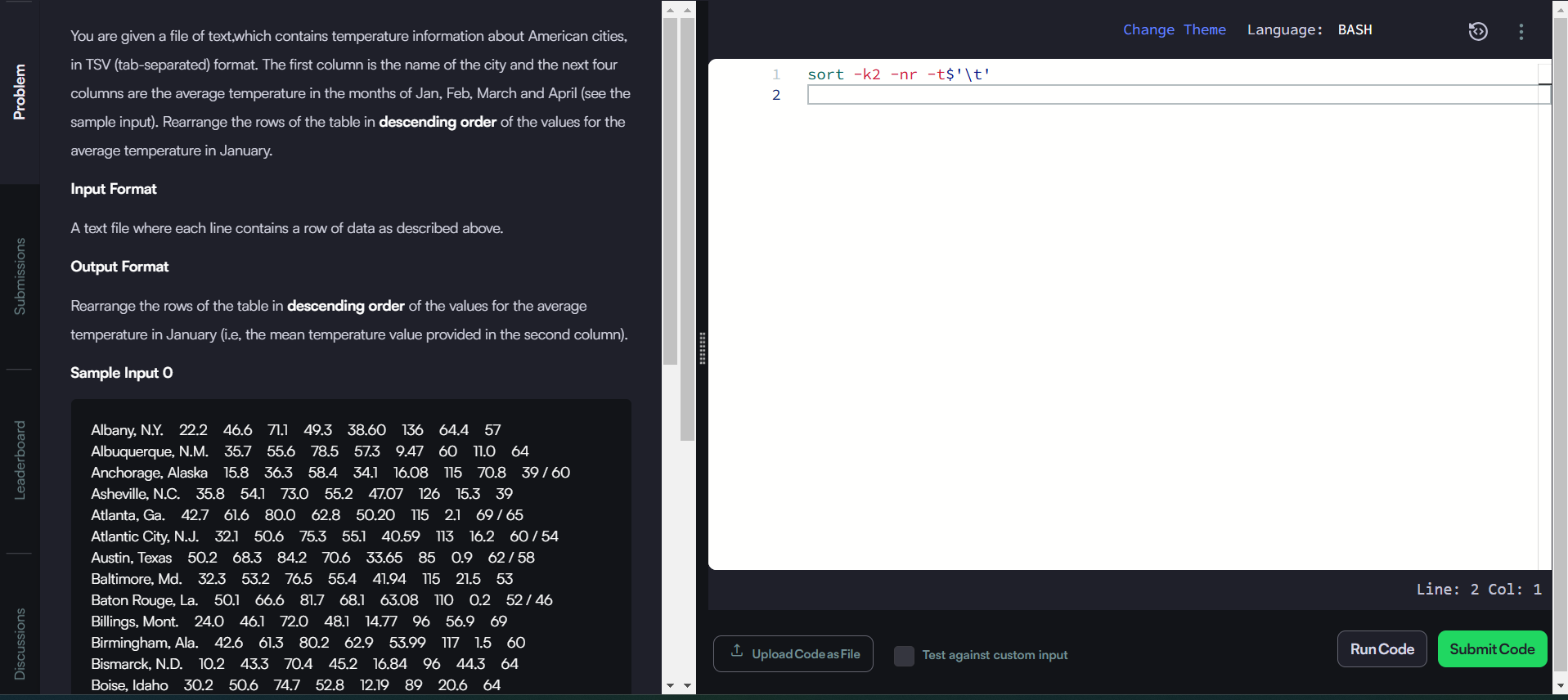
5. <https://www.hackerrank.com/challenges/text-processing-sort-3/problem?isFullScreen=true>



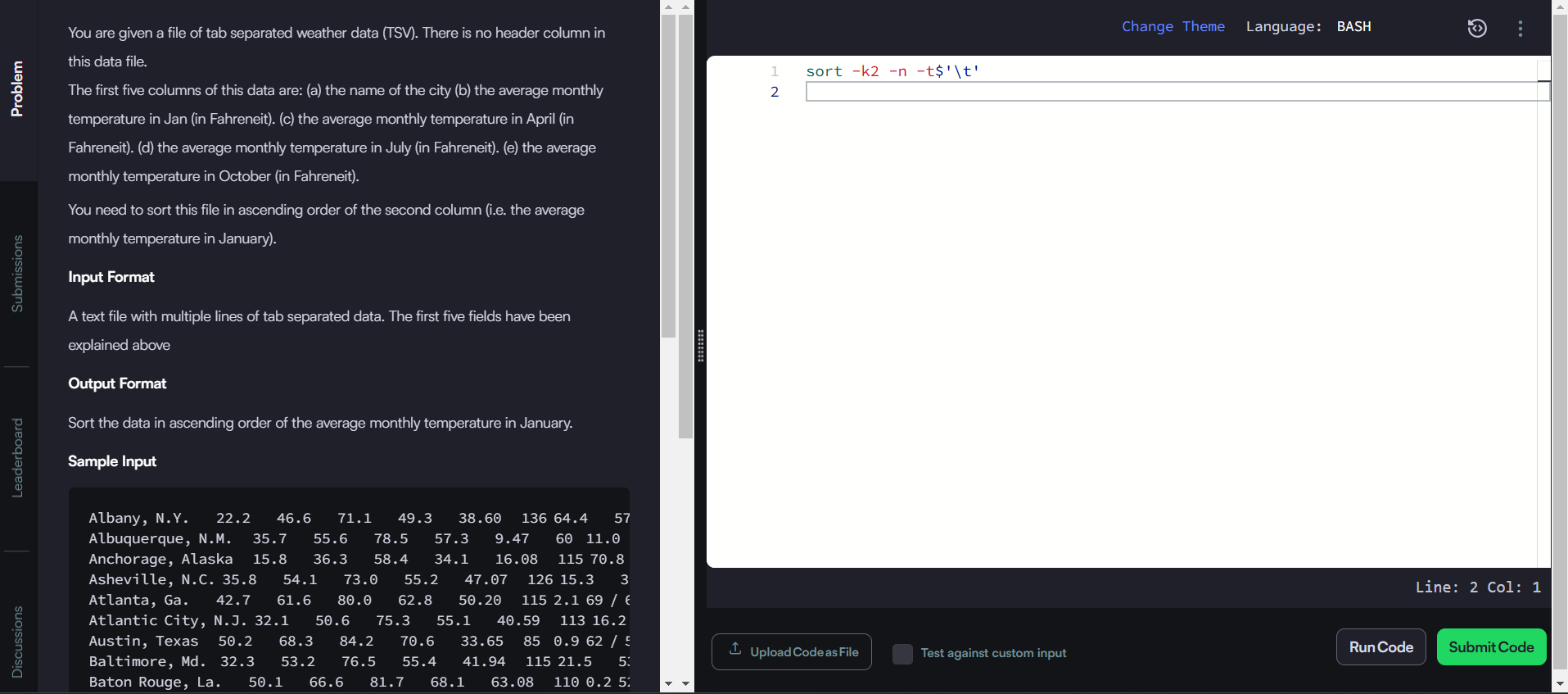
6. [https://www.hackerrank.com/challenges/text-processing-sort-4/problem?isFullScreen=true](about:blank)



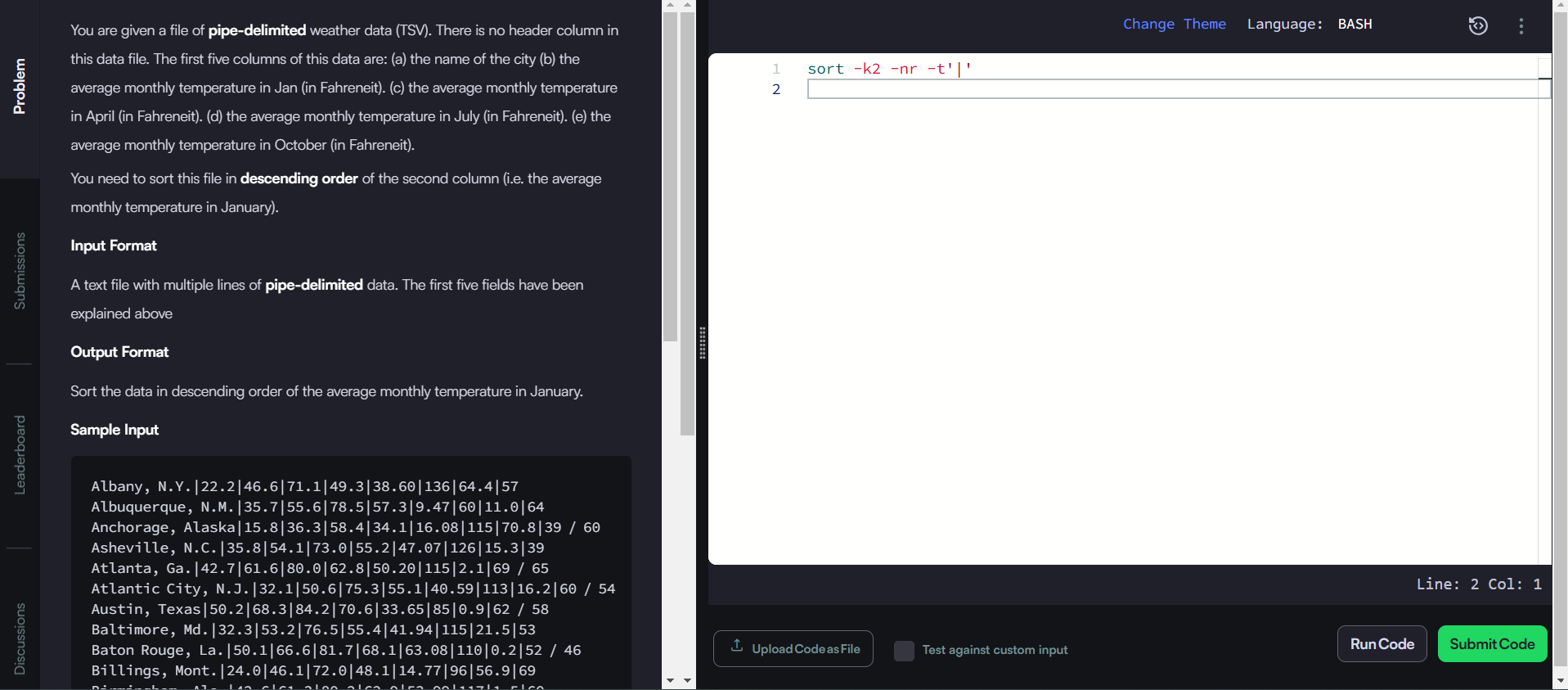
7. <https://www.hackerrank.com/challenges/text-processing-sort-5/problem?isFullScreen=true>



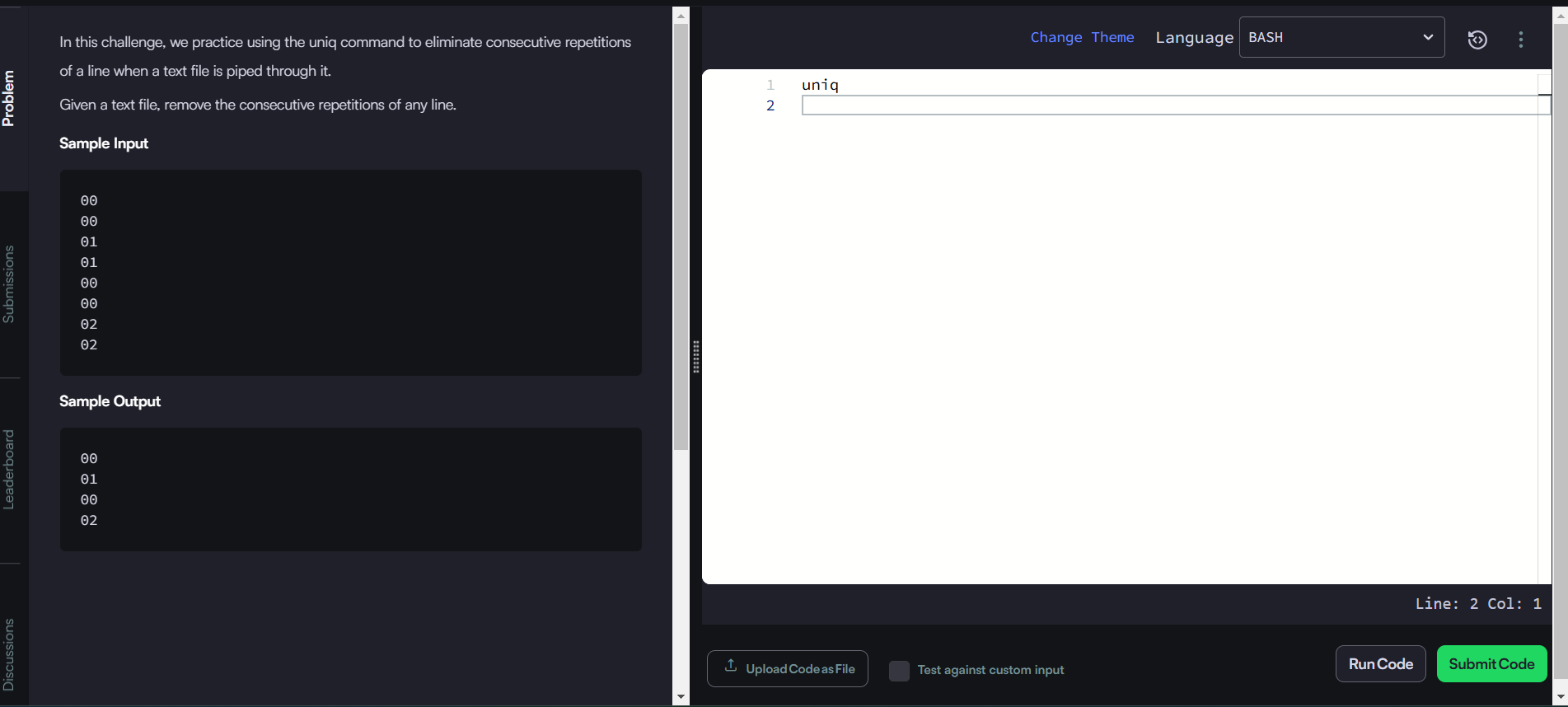
8. <https://www.hackerrank.com/challenges/text-processing-sort-6/problem?isFullScreen=true>



9. <https://www.hackerrank.com/challenges/text-processing-sort-7/problem?isFullScreen=true>



10, <https://www.hackerrank.com/challenges/text-processing-in-linux-the-uniq-command-1/problem?isFullScreen=true>



**9. Takeaways**

1. **Array:** Efficient for storing and iterating multiple values.
2. **[[ ]] vs. [ ]:** Prefer [[ ]] for pattern matching.
3. **File Test Operators:** Useful for checking file existence and permissions.
4. **su vs. sudo:** su switches users, while sudo runs commands as root.
5. **read Flags:** -p for prompts, -s for passwords.
6. **case:** Simplifies multi-way branching.
7. **Regex:** Essential for pattern matching in text.
8. **Problem-Solving:** Commands like sort, uniq, and tr simplify data manipulation.