# Problem 500: Keyboard Row

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

Given an array of strings

words

, return

the words that can be typed using letters of the alphabet on only one row of American keyboard like the image below

.

Note

that the strings are

case-insensitive

, both lowercased and uppercased of the same letter are treated as if they are at the same row.

In the

American keyboard

:

the first row consists of the characters

"qwertyuiop"

,

the second row consists of the characters

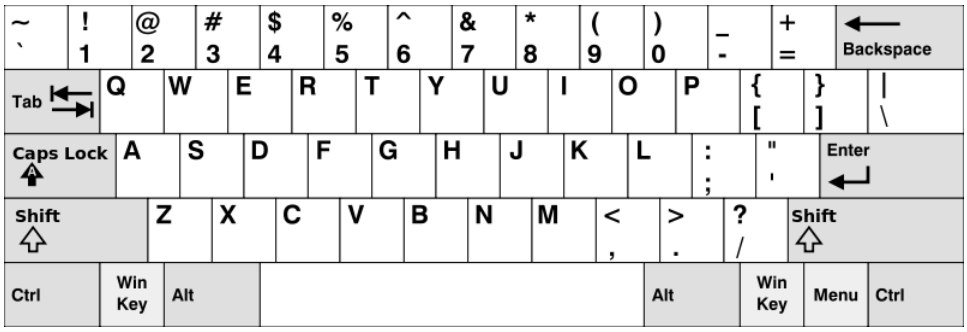
"asdfghjkl"

, and

the third row consists of the characters

"zxcvbnm"

.




Example 1:

Input:

words = ["Hello","Alaska","Dad","Peace"]

Output:

["Alaska","Dad"]

Explanation:

Both

"a"

and

"A"

are in the 2nd row of the American keyboard due to case insensitivity.

Example 2:

Input:

words = ["omk"]

Output:

[]

Example 3:

Input:

words = ["adsdf","sfd"]

Output:

["adsdf","sfd"]

Constraints:

1 <= words.length <= 20

1 <= words[i].length <= 100

words[i]

consists of English letters (both lowercase and uppercase).

## Code Snippets

**C++:**

```
class Solution {
public:
vector<string> findWords(vector<string>& words) {

}
};
```

**Java:**

```
class Solution {
public String[] findWords(String[] words) {

}
}
```

**Python3:**

```
class Solution:
def findWords(self, words: List[str]) -> List[str]:
```

**Python:**

```
class Solution(object):
def findWords(self, words):
"""
:type words: List[str]
:rtype: List[str]
"""
```

**JavaScript:**

```
/**
* @param {string[]} words
* @return {string[]}
*/
var findWords = function(words) {

};
```

**TypeScript:**

```
function findWords(words: string[]): string[] {

};
```

**C#:**

```
public class Solution {
public string[] FindWords(string[] words) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findWords(char** words, int wordsSize, int* returnSize) {

}
```

**Go:**

```
func findWords(words []string) []string {

}
```

**Kotlin:**

```
class Solution {
fun findWords(words: Array<String>): Array<String> {

}
}
```

**Swift:**

```
class Solution {
func findWords(_ words: [String]) -> [String] {

}
```

```
}
```

**Rust:**

```
impl Solution {
pub fn find_words(words: Vec<String>) -> Vec<String> {

}
}
```

**Ruby:**

```
# @param {String[]} words
# @return {String[]}
def find_words(words)

end
```

**PHP:**

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function findWords($words) {

}
}
```

**Dart:**

```
class Solution {
List<String> findWords(List<String> words) {

}
}
```

**Scala:**

```
object Solution {
def findWords(words: Array[String]): Array[String] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec find_words(words :: [String.t]) :: [String.t]
def find_words(words) do

end
end
```

**Erlang:**

```
-spec find_words(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_words(Words) ->
.
```

**Racket:**

```
(define/contract (find-words words)
(-> (listof string?) (listof string?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public:
vector<string> findWords(vector<string>& words) {

}
};
```

### Java Solution:

```
/**
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String[] findWords(String[] words) {

}
}
```

### Python3 Solution:

```
"""
Problem: Keyboard Row
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def findWords(self, words: List[str]) -> List[str]:
# TODO: Implement optimized solution
```

```
pass
```

### Python Solution:

```
class Solution(object):
def findWords(self, words):
"""
:type words: List[str]
:rtype: List[str]
"""
```

### JavaScript Solution:

```
/**
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} words
* @return {string[]}
*/
var findWords = function(words) {

};
```

### TypeScript Solution:

```
/**
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

function findWords(words: string[]): string[] {

};
```

### C# Solution:

```
/*
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string[] FindWords(string[] words) {

}
}
```

### C Solution:

```
/*
* Problem: Keyboard Row
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** findWords(char** words, int wordsSize, int* returnSize) {
```

```
}
```

### Go Solution:

```
// Problem: Keyboard Row
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func findWords(words []string) []string {

}
```

### Kotlin Solution:

```
class Solution {
fun findWords(words: Array<String>): Array<String> {

}
}
```

### Swift Solution:

```
class Solution {
func findWords(_ words: [String]) -> [String] {

}
}
```

### Rust Solution:

```
// Problem: Keyboard Row
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
impl Solution {
pub fn find_words(words: Vec<String>) -> Vec<String> {

}
}
```

### Ruby Solution:

```
# @param {String[]} words
# @return {String[]}
def find_words(words)

end
```

### PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return String[]
*/
function findWords($words) {

}
}
```

### Dart Solution:

```
class Solution {
List<String> findWords(List<String> words) {

}
}
```

### Scala Solution:

```
object Solution {
def findWords(words: Array[String]): Array[String] = {
```

```
}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec find_words(words :: [String.t]) :: [String.t]
def find_words(words) do

end
end
```

### Erlang Solution:

```
-spec find_words(Words :: [unicode:unicode_binary()]) ->
[unicode:unicode_binary()].
find_words(Words) ->
.
```

### Racket Solution:

```
(define/contract (find-words words)
(-> (listof string?) (listof string?))
)
```