Problem 1878: Get Biggest Three Rhombus Sums in a Grid

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an

m x n

integer matrix

grid

.

A

rhombus sum

is the sum of the elements that form

the

border

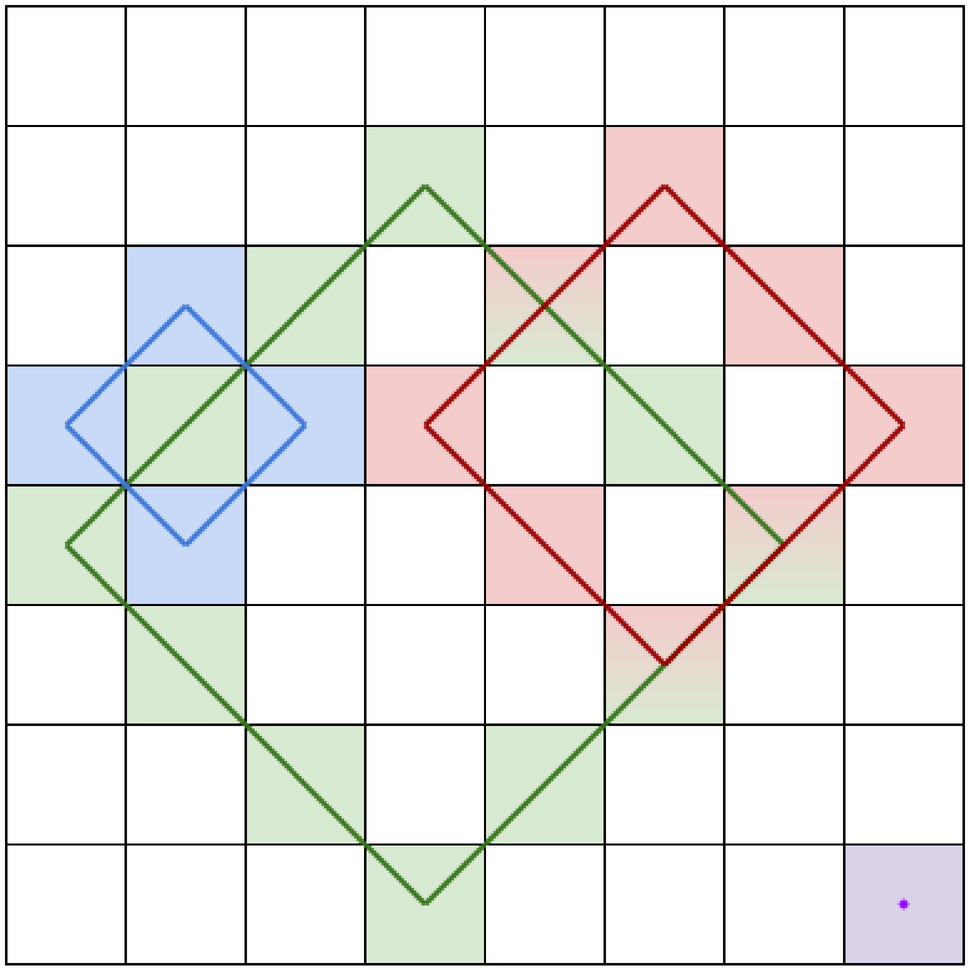
of a regular rhombus shape in

grid

. The rhombus must have the shape of a square rotated 45 degrees with each of the corners centered in a grid cell. Below is an image of four valid rhombus shapes with the corresponding colored cells that should be included in each

rhombus sum

:



Note that the rhombus can have an area of 0, which is depicted by the purple rhombus in the bottom right corner.

Return

the biggest three

distinct rhombus sums

in the

grid

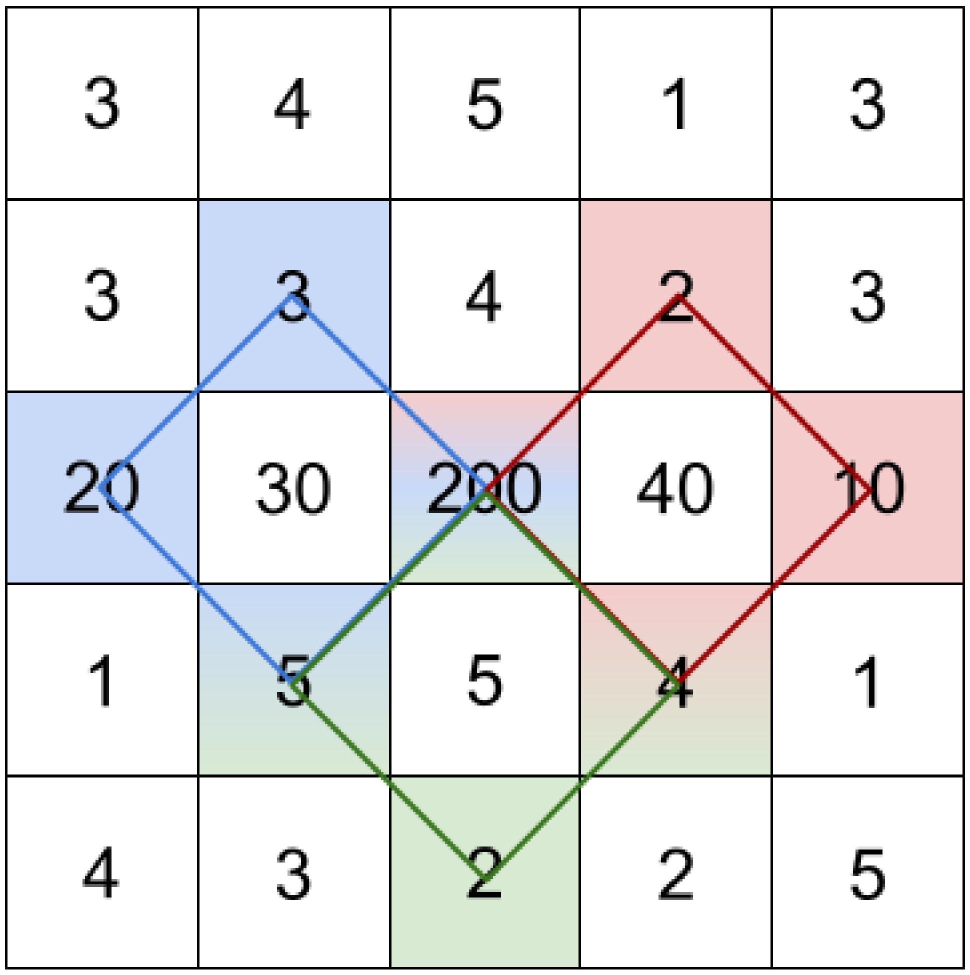
in

descending order

. If there are less than three distinct values, return all of them

.

Example 1:



Input:

grid = [[3,4,5,1,3],[3,3,4,2,3],[20,30,200,40,10],[1,5,5,4,1],[4,3,2,2,5]]

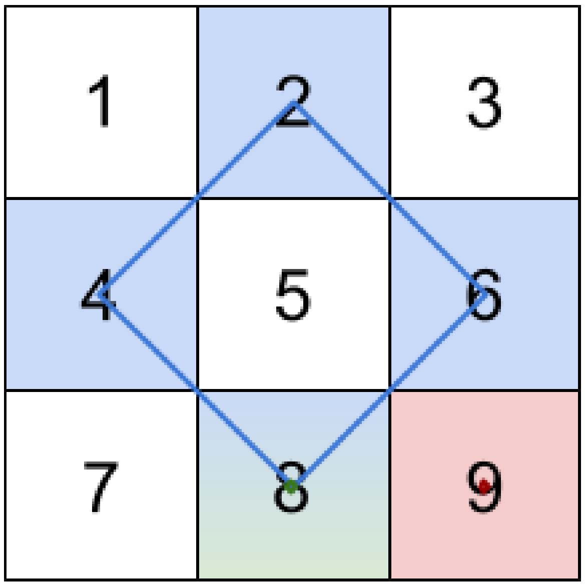
Output:

[228,216,211]

Explanation:

The rhombus shapes for the three biggest distinct rhombus sums are depicted above. - Blue: 20 + 3 + 200 + 5 = 228 - Red: 200 + 2 + 10 + 4 = 216 - Green: 5 + 200 + 4 + 2 = 211

Example 2:



Input:

grid = [[1,2,3],[4,5,6],[7,8,9]]

Output:

[20,9,8]

Explanation:

The rhombus shapes for the three biggest distinct rhombus sums are depicted above. - Blue: 4 + 2 + 6 + 8 = 20 - Red: 9 (area 0 rhombus in the bottom right corner) - Green: 8 (area 0 rhombus in the bottom middle)

Example 3:

Input:

grid = [[7,7,7]]

Output:

[7]

Explanation:

All three possible rhombus sums are the same, so return [7].

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

1 <= grid[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> getBiggestThree(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int[] getBiggestThree(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def getBiggestThree(self, grid: List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def getBiggestThree(self, grid):
"""
:type grid: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number[]}
*/
var getBiggestThree = function(grid) {

};
```

TypeScript:

```
function getBiggestThree(grid: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] GetBiggestThree(int[][] grid) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getBiggestThree(int** grid, int gridSize, int* gridColSize, int*
returnSize) {

}
```

Go:

```
func getBiggestThree(grid [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun getBiggestThree(grid: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func getBiggestThree(_ grid: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_biggest_three(grid: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer[]}
def get_biggest_three(grid)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer[]
*/
function getBiggestThree($grid) {

}
}
```

Dart:

```
class Solution {
List<int> getBiggestThree(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def getBiggestThree(grid: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_biggest_three(grid :: [[integer]]) :: [integer]
def get_biggest_three(grid) do

end
end
```

Erlang:

```
-spec get_biggest_three(Grid :: [[integer()]]) -> [integer()].
get_biggest_three(Grid) ->
.
```

Racket:

```
(define/contract (get-biggest-three grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> getBiggestThree(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public int[] getBiggestThree(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Get Biggest Three Rhombus Sums in a Grid
Difficulty: Medium
Tags: array, math, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def getBiggestThree(self, grid: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getBiggestThree(self, grid):
"""
:type grid: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} grid
* @return {number[]}
*/
var getBiggestThree = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function getBiggestThree(grid: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public int[] GetBiggestThree(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Get Biggest Three Rhombus Sums in a Grid
* Difficulty: Medium
* Tags: array, math, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getBiggestThree(int** grid, int gridSize, int* gridColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Get Biggest Three Rhombus Sums in a Grid
// Difficulty: Medium
// Tags: array, math, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func getBiggestThree(grid [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getBiggestThree(grid: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func getBiggestThree(_ grid: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Get Biggest Three Rhombus Sums in a Grid
// Difficulty: Medium
// Tags: array, math, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn get_biggest_three(grid: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer[]}
def get_biggest_three(grid)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $grid
* @return Integer[]
*/
function getBiggestThree($grid) {

}
}
```

Dart Solution:

```
class Solution {
List<int> getBiggestThree(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def getBiggestThree(grid: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_biggest_three(grid :: [[integer]]) :: [integer]
def get_biggest_three(grid) do

end
end
```

Erlang Solution:

```
-spec get_biggest_three(Grid :: [[integer()]]) -> [integer()].
get_biggest_three(Grid) ->
.
```

Racket Solution:

```
(define/contract (get-biggest-three grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```