Problem 970: Powerful Integers

Problem Information

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

Problem Description

Given three integers

x

,

y

, and

bound

, return

a list of all the

powerful integers

that have a value less than or equal to

bound

.

An integer is

powerful

if it can be represented as

x

i

+ y

j

for some integers

i >= 0

and

j >= 0

.

You may return the answer in

any order

. In your answer, each value should occur

at most once

.

Example 1:

Input:

x = 2, y = 3, bound = 10

Output:

[2,3,4,5,7,9,10]

Explanation:

2 = 2

0

+ 3

0

3 = 2

1

+ 3

0

4 = 2

0

+ 3

1

5 = 2

1

+ 3

1

7 = 2

2

+ 3

1

9 = 2

3

+ 3

0

10 = 2

0

+ 3

2

Example 2:

Input:

x = 3, y = 5, bound = 15

Output:

[2,4,6,8,10,14]

Constraints:

1 <= x, y <= 100

0 <= bound <= 10

6

Code Snippets

C++:

```
class Solution {
public:
vector<int> powerfulIntegers(int x, int y, int bound) {

}
};
```

Java:

```
class Solution {
public List<Integer> powerfulIntegers(int x, int y, int bound) {

}
}
```

Python3:

```
class Solution:
def powerfulIntegers(self, x: int, y: int, bound: int) -> List[int]:
```

Python:

```
class Solution(object):
def powerfulIntegers(self, x, y, bound):
"""
:type x: int
:type y: int
:type bound: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @param {number} bound
```

```
* @return {number[]}
*/
var powerfulIntegers = function(x, y, bound) {

};
```

TypeScript:

```
function powerfulIntegers(x: number, y: number, bound: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> PowerfulIntegers(int x, int y, int bound) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* powerfulIntegers(int x, int y, int bound, int* returnSize) {

}
```

Go:

```
func powerfulIntegers(x int, y int, bound int) []int {

}
```

Kotlin:

```
class Solution {
fun powerfulIntegers(x: Int, y: Int, bound: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func powerfulIntegers(_ x: Int, _ y: Int, _ bound: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn powerful_integers(x: i32, y: i32, bound: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} bound
# @return {Integer[]}
def powerful_integers(x, y, bound)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $bound
* @return Integer[]
*/
function powerfulIntegers($x, $y, $bound) {

}
}
```

Dart:

```
class Solution {
List<int> powerfulIntegers(int x, int y, int bound) {

}
}
```

Scala:

```
object Solution {
def powerfulIntegers(x: Int, y: Int, bound: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec powerful_integers(x :: integer, y :: integer, bound :: integer) ::
[integer]
def powerful_integers(x, y, bound) do

end
end
```

Erlang:

```
-spec powerful_integers(X :: integer(), Y :: integer(), Bound :: integer())
-> [integer()].
powerful_integers(X, Y, Bound) ->
.
```

Racket:

```
(define/contract (powerful-integers x y bound)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Powerful Integers
* Difficulty: Medium
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> powerfulIntegers(int x, int y, int bound) {

}
};
```

Java Solution:

```
/**
* Problem: Powerful Integers
* Difficulty: Medium
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<Integer> powerfulIntegers(int x, int y, int bound) {

}
}
```

Python3 Solution:

```
"""
Problem: Powerful Integers
Difficulty: Medium
Tags: math, hash
```

```
Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class Solution:
def powerfulIntegers(self, x: int, y: int, bound: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def powerfulIntegers(self, x, y, bound):
"""
:type x: int
:type y: int
:type bound: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Powerful Integers
* Difficulty: Medium
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} x
* @param {number} y
* @param {number} bound
* @return {number[]}
*/
var powerfulIntegers = function(x, y, bound) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Powerful Integers
* Difficulty: Medium
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

function powerfulIntegers(x: number, y: number, bound: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Powerful Integers
* Difficulty: Medium
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<int> PowerfulIntegers(int x, int y, int bound) {

}
}
```

C Solution:

```
/*
* Problem: Powerful Integers
* Difficulty: Medium
```

```
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* powerfulIntegers(int x, int y, int bound, int* returnSize) {

}
```

Go Solution:

```
// Problem: Powerful Integers
// Difficulty: Medium
// Tags: math, hash
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

func powerfulIntegers(x int, y int, bound int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun powerfulIntegers(x: Int, y: Int, bound: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func powerfulIntegers(_ x: Int, _ y: Int, _ bound: Int) -> [Int] {
```

```
}
}
```

Rust Solution:

```
// Problem: Powerful Integers
// Difficulty: Medium
// Tags: math, hash
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

impl Solution {
pub fn powerful_integers(x: i32, y: i32, bound: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @param {Integer} bound
# @return {Integer[]}
def powerful_integers(x, y, bound)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @param Integer $bound
* @return Integer[]
*/
function powerfulIntegers($x, $y, $bound) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> powerfulIntegers(int x, int y, int bound) {

}
}
```

Scala Solution:

```
object Solution {
def powerfulIntegers(x: Int, y: Int, bound: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec powerful_integers(x :: integer, y :: integer, bound :: integer) ::
[integer]
def powerful_integers(x, y, bound) do

end
end
```

Erlang Solution:

```
-spec powerful_integers(X :: integer(), Y :: integer(), Bound :: integer())
-> [integer()].
powerful_integers(X, Y, Bound) ->
.
```

Racket Solution:

```
(define/contract (powerful-integers x y bound)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?))
)
```