Problem 1356: Sort Integers by The Number of 1 Bits

Problem Information

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

Problem Description

You are given an integer array

arr

. Sort the integers in the array in ascending order by the number of

1

's in their binary representation and in case of two or more integers have the same number of

1

's you have to sort them in ascending order.

Return

the array after sorting it

.

Example 1:

Input:

arr = [0,1,2,3,4,5,6,7,8]

Output:

[0,1,2,4,8,3,5,6,7]

Explantion:

[0] is the only integer with 0 bits. [1,2,4,8] all have 1 bit. [3,5,6] have 2 bits. [7] has 3 bits. The sorted array by bits is [0,1,2,4,8,3,5,6,7]

Example 2:

Input:

arr = [1024,512,256,128,64,32,16,8,4,2,1]

Output:

[1,2,4,8,16,32,64,128,256,512,1024]

Explantion:

All integers have 1 bit in the binary representation, you should just sort them in ascending order.

Constraints:

1 <= arr.length <= 500

0 <= arr[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<int> sortByBits(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int[] sortByBits(int[] arr) {

}
}
```

Python3:

```
class Solution:
def sortByBits(self, arr: List[int]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sortByBits(arr: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] SortByBits(int[] arr) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sortByBits(int* arr, int arrSize, int* returnSize) {

}
```

Go:

```
func sortByBits(arr []int) []int {

}
```

Kotlin:

```
class Solution {
fun sortByBits(arr: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func sortByBits(_ arr: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_by_bits(arr: Vec<i32>) -> Vec<i32> {
}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer[]}
def sort_by_bits(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function sortByBits($arr) {

}
}
```

Dart:

```
class Solution {
List<int> sortByBits(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def sortByBits(arr: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_by_bits(arr :: [integer]) :: [integer]
def sort_by_bits(arr) do

end
end
```

Erlang:

```
-spec sort_by_bits(Arr :: [integer()]) -> [integer()].
sort_by_bits(Arr) ->
.
```

Racket:

```
(define/contract (sort-by-bits arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
* 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> sortByBits(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
* 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[] sortByBits(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Integers by The Number of 1 Bits
Difficulty: Easy
Tags: array, sort

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 sortByBits(self, arr: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
* 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[]} arr
* @return {number[]}
*/
var sortByBits = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
* 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 sortByBits(arr: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
```

```
* 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[] SortByBits(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Sort Integers by The Number of 1 Bits
* Difficulty: Easy
* Tags: array, sort
*
* 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* sortByBits(int* arr, int arrSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort Integers by The Number of 1 Bits
// Difficulty: Easy
// Tags: array, sort
//
// 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 sortByBits(arr []int) []int {
}
```

Kotlin Solution:

```
class Solution {
fun sortByBits(arr: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func sortByBits(_ arr: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Sort Integers by The Number of 1 Bits
// Difficulty: Easy
// Tags: array, sort
//
// 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 sort_by_bits(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer[]}
def sort_by_bits(arr)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function sortByBits($arr) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sortByBits(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def sortByBits(arr: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_by_bits(arr :: [integer]) :: [integer]
def sort_by_bits(arr) do

end
end
```

Erlang Solution:

```
-spec sort_by_bits(Arr :: [integer()]) -> [integer()].
sort_by_bits(Arr) ->
.
```

Racket Solution:

```
(define/contract (sort-by-bits arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```