Problem 2231: Largest Number After Digit Swaps by Parity

Problem Information

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

Problem Description

You are given a positive integer

num

. You may swap any two digits of

num

that have the same

parity

(i.e. both odd digits or both even digits).

Return

the

largest

possible value of

num

after

any

number of swaps.

Example 1:

Input:

num = 1234

Output:

3412

Explanation:

Swap the digit 3 with the digit 1, this results in the number 3214. Swap the digit 2 with the digit 4, this results in the number 3412. Note that there may be other sequences of swaps but it can be shown that 3412 is the largest possible number. Also note that we may not swap the digit 4 with the digit 1 since they are of different parities.

Example 2:

Input:

num = 65875

Output:

87655

Explanation:

Swap the digit 8 with the digit 6, this results in the number 85675. Swap the first digit 5 with the digit 7, this results in the number 87655. Note that there may be other sequences of swaps but it can be shown that 87655 is the largest possible number.

Constraints:

1 <= num <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int largestInteger(int num) {

}
};
```

Java:

```
class Solution {
public int largestInteger(int num) {

}
}
```

Python3:

```
class Solution:
def largestInteger(self, num: int) -> int:
```

Python:

```
class Solution(object):
def largestInteger(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num
* @return {number}
*/
var largestInteger = function(num) {

};
```

TypeScript:

```
function largestInteger(num: number): number {

};
```

C#:

```
public class Solution {
public int LargestInteger(int num) {

}
}
```

C:

```
int largestInteger(int num) {

}
```

Go:

```
func largestInteger(num int) int {

}
```

Kotlin:

```
class Solution {
fun largestInteger(num: Int): Int {

}
}
```

Swift:

```
class Solution {
func largestInteger(_ num: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn largest_integer(num: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer}
def largest_integer(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function largestInteger($num) {

}
}
```

Dart:

```
class Solution {
int largestInteger(int num) {

}
}
```

Scala:

```
object Solution {
def largestInteger(num: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_integer(num :: integer) :: integer
def largest_integer(num) do

end
end
```

Erlang:

```
-spec largest_integer(Num :: integer()) -> integer().
largest_integer(Num) ->
.
```

Racket:

```
(define/contract (largest-integer num)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int largestInteger(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int largestInteger(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Number After Digit Swaps by Parity
Difficulty: Easy
Tags: sort, queue, heap

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def largestInteger(self, num: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def largestInteger(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} num
* @return {number}
*/
var largestInteger = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function largestInteger(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int LargestInteger(int num) {

}
}
```

C Solution:

```
/*
* Problem: Largest Number After Digit Swaps by Parity
* Difficulty: Easy
* Tags: sort, queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int largestInteger(int num) {

}
```

Go Solution:

```
// Problem: Largest Number After Digit Swaps by Parity
// Difficulty: Easy
// Tags: sort, queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func largestInteger(num int) int {

}
```

Kotlin Solution:

```
class Solution {
fun largestInteger(num: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func largestInteger(_ num: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Largest Number After Digit Swaps by Parity
// Difficulty: Easy
// Tags: sort, queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn largest_integer(num: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer}
def largest_integer(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function largestInteger($num) {

}
}
```

Dart Solution:

```
class Solution {
int largestInteger(int num) {

}
}
```

Scala Solution:

```
object Solution {
def largestInteger(num: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec largest_integer(num :: integer) :: integer
def largest_integer(num) do

end
end
```

Erlang Solution:

```
-spec largest_integer(Num :: integer()) -> integer().
largest_integer(Num) ->
.
```

Racket Solution:

```
(define/contract (largest-integer num)
(-> exact-integer? exact-integer?)
)
```