2815. Max Pair Sum in an Array

Hash Table Easy

Problem Description

In this problem, we are provided with an array of integers called nums, which is indexed starting from 0. Our task is to find the maximum sum obtainable by selecting two different numbers from this array where the largest digit present in both numbers is exactly the same.

For instance, if the array contains the numbers 34 and 42, we can't pair them to form a sum because their maximum digits (4 in 34) and 4 in 42) are not equal. However, if we had the numbers 34 and 43, their sum is a valid candidate because the maximum digit 4 is present in both numbers.

If it is possible to find at least one pair of numbers with this property, we should return the sum of that pair which is the largest

To summarize, the goal is to maximize the sum of a number pair, under the constraint that the pair's maximum digits are equal.

ntuition

The process of arriving at the solution involves enumerating all possible pairs from the array to check two primary conditions:

2. Whether the sum of these numbers is greater than any sum we've previously recorded.

We start with an answer variable ans which is initialized to -1, assuming initially that there are no pairs satisfying the condition. We then iterate over each possible pair of numbers in the array, checking the conditions for every such pair.

1. Whether the largest digit in both numbers of the pair is the same.

among all such valid pairs. If no such pair exists, we need to return -1.

For the first condition, we are required to identify the largest digit of each number. This is achieved by converting the integers to strings, and then using the max function which finds the maximum character in each string representation (which corresponds to

the maximum digit in the number). If the maximum digits of both numbers in the pair match, we proceed to the second condition.

For the second condition, we simply calculate the sum of the two numbers and check if this sum is larger than our current recorded maximum sum (ans). If it is, we update ans with this new sum. **Solution Approach**

The implementation of this problem's solution uses a straightforward approach: a nested loop to enumerate all possible pairs

Here's a step-by-step walkthrough of the algorithm, utilizing the reference solution approach: 1. We start by initializing an answer variable ans with a value of -1. This variable will keep track of the maximum sum we find

highest digit in each number.

the criteria of having the same largest digit.

○ Since 88 is greater than -1, we update ans to 88.

Now, there are no more pairs left to be compared.

def maxSum(self, nums: List[int]) -> int:

for i, current_val in enumerate(nums):

for other_val in nums[i + 1:]:

max_sum = pair_sum

that satisfies the condition. We use a for-loop to iterate over each element of the array nums by its index i. This outer loop picks the first number of the

pair.

meaning no valid pairs have been found, the initial -1 value will be returned.

We initialize ans to -1 to handle the case where we do not find any valid pairs.

We then compare nums[1] = 71 with the last number nums[3] = 42.

from the array and check the conditions for each pair to identify the one with the highest sum.

- Inside the outer loop, we use a nested for-loop to iterate over the rest of the elements in the array starting from index i + 1. This inner loop picks the second number of the pair.
- We calculate the potential sum v of the two numbers nums[i] (from the outer loop) and nums[j] (from the inner loop). We convert both numbers nums[i] and nums[j] to strings using str() to be able to utilize the max() function and extract the
- We compare the maximum digit of both numbers using max(str(nums[i])) == max(str(nums[j])) to check if they are equal. If they are not equal, we continue to the next iteration without executing further code.

If the maximum digits are equal, we then check if the sum of these two numbers v is greater than the current ans. If it is, we

- update ans with the new, larger sum v. After iterating over all possible pairs, the loop concludes, and the maximum sum ans is returned. If ans has not been updated,
- a simple and direct method, its time complexity is 0(n^2) due to the nested loops. This complexity arises because for each element in the array, we are iterating through all other elements that follow it to form pairs. Despite its simplicity, this approach guarantees we do not miss any valid pairs and subsequently the maximum sum that meets

This algorithm leverages the brute-force paradigm by checking each possible pair in the array against the condition. Although it is

Example Walkthrough Let us consider a small example using the array nums = [51, 71, 17, 42]. We will illustrate the solution approach with this array.

We start with the first number (at index i=0), nums[0] = 51, and compare it with every other number in the array. Now, we move to the second number (at index i=1), nums[1] = 71, and compare it with numbers after it. The first comparison

• The maximum digit in 71 is 7 and in 17 is also 7.

is with nums [2] = 17.

 \circ Since both have the same maximum digit, we compute the sum 71 + 17 = 88. ○ We compare this sum 88 with ans which is currently -1.

By the end of this process, we've examined all possible pairs and determined that the 71 and 17 pair provides the highest sum

(88) among pairs with matching maximum digits. If no such pair existed, the function would return the default ans value of -1.

- The maximum digit in 71 is 7, whereas in 42 it is 4. Since the digits are not the same, we do not update ans.
- Next, we move to the third number nums[2] = 17, and compare it with the only number after it, nums[3] = 42. • The maximum digit in both 17 and 42 is not the same (7 vs 4), so we do not update ans.
- We return ans, which is 88.

Having finished the loop, the largest sum we found from valid pairs is 88.

Solution Implementation

max_digit_current = max(str(current_val))

and the maximum digits of both numbers are equal

Return the maximum sum found, or -1 if no such pair exists

int digit = x % 10; // Get the last digit of x

x /= 10; // Remove the last digit from x

#include <vector> // Include necessary header for vector

using std::vector; // Makes using vector easier without std:: prefix

if max_sum < pair_sum and max_digit_current == max_digit_other:</pre>

max_digit_other = max(str(other_val))

Python

Initialize the answer with -1, which will also be the return value if no valid pair is found

Update the max_sum if the current pair sum is greater than the previous max_sum

Loop through the remaining numbers (other_val) in the list starting from the index right after i

Enumerate gives both the index (i) and the number (current_val) from the list nums

Calculate the potential maximum sum of the current pair pair_sum = current_val + other_val # Convert both numbers to strings and find the maximum digit in each

return max_sum

 $max_sum = -1$

class Solution:

```
Java
class Solution {
    // Computes the maximum sum of pairs with equal highest digits
    public int maxSum(int[] nums) {
        int maxPairSum = -1; // Initialize maximum pair sum to -1 (indicating no pairs found yet)
       int n = nums.length; // Extract the length of the nums array.
       // Iterate over the array using two pointers to find all possible pairs
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                int currentSum = nums[i] + nums[j]; // Calculate the sum of the current pair
                // Check if the current pair has the same highest digit and if the sum is greater than maxPairSum
                if (maxPairSum < currentSum && getHighestDigit(nums[i]) == getHighestDigit(nums[j])) {</pre>
                    maxPairSum = currentSum; // Update the maximum pair sum if conditions are met
       return maxPairSum; // Return the computed maximum pair sum
    // Helper function to determine the highest digit in an integer
    private int getHighestDigit(int x) {
        int highestDigit = 0; // Initialize highest digit to 0
       // Iterate over the digits of x
```

highestDigit = Math.max(highestDigit, digit); // Update the highest digit found so far

```
return highestDigit; // Return the highest digit found
```

class Solution {

C++

while (x > 0) {

```
public:
    int maxSum(vector<int>& nums) {
        int maxSum = -1; // Use maxSum to track the maximum sum found
        int n = nums.size(); // Store the size of the input vector
       // Lambda function f extracts the highest digit from a given integer
        auto getHighestDigit = [](int x) {
            int highestDigit = 0;
           // Loop to find the highest digit
           while (x > 0) {
                highestDigit = std::max(highestDigit, x % 10);
                x /= 10;
            return highestDigit;
       // Double loop to check each pair of numbers in the vector
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                // Sum of the current pair
                int currentSum = nums[i] + nums[j];
                // Check if the highest digits are equal and update maxSum if necessary
                if (maxSum < currentSum && getHighestDigit(nums[i]) == getHighestDigit(nums[j])) {</pre>
                    maxSum = currentSum;
       return maxSum; // Return the maximum sum found
};
TypeScript
function maxSum(nums: number[]): number {
    const length = nums.length; // Get the length of the input array
    let maxPairSum = -1; // Initialize the maxPairSum with -1 as the lowest possible value
    // Function to calculate the highest single digit in a number
    const findMaxDigit = (number: number): number => {
        let maxDigit = 0;
       // Iterate through digits of the number to find the maximum digit
       while (number > 0) {
           maxDigit = Math.max(maxDigit, number % 10);
           number = Math.floor(number / 10); // Move to the next digit
```

```
// Return the highest sum found that satisfies the condition
      return maxPairSum;
class Solution:
   def maxSum(self, nums: List[int]) -> int:
       # Initialize the answer with -1, which will also be the return value if no valid pair is found
       \max sum = -1
       # Enumerate gives both the index (i) and the number (current_val) from the list nums
       for i, current_val in enumerate(nums):
           # Loop through the remaining numbers (other_val) in the list starting from the index right after i
           for other_val in nums[i + 1:]:
               # Calculate the potential maximum sum of the current pair
               pair_sum = current_val + other_val
               # Convert both numbers to strings and find the maximum digit in each
               max_digit_current = max(str(current_val))
               max_digit_other = max(str(other_val))
               # Update the max_sum if the current pair sum is greater than the previous max_sum
               # and the maximum digits of both numbers are equal
               if max_sum < pair_sum and max_digit_current == max_digit_other:</pre>
                   max_sum = pair_sum
       # Return the maximum sum found, or -1 if no such pair exists
       return max sum
Time and Space Complexity
Time Complexity
```

The time complexity of the code is primarily determined by two nested loops and the operation to find the maximum digit in numbers within the loops. The two nested loops over the array nums with length n give us 0(n^2) complexity since every element

return maxDigit;

for (let i = 0; i < length; ++i) {</pre>

for (let j = i + 1; j < length; ++j) {</pre>

// Iterate over all unique index pairs (i, j) to find the highest sum

const currentSum = nums[i] + nums[j]; // Calculate the sum of the current pair

maxPairSum = currentSum; // Update the maxPairSum with the current sum

if (maxPairSum < currentSum && findMaxDigit(nums[i]) === findMaxDigit(nums[j])) {</pre>

// with the condition that the max digit of both numbers is the same

// Check if the max digit of both numbers is the same

// and if the current sum is greater than the current maxPairSum

};

is compared with every other element in a brute force manner. For each pair (x, y) selected by the nested loops, the code converts x and y to strings and finds the maximum digit in each. Since the number of digits in a number is proportional to the logarithm of the number (to be precise, it's O(log M) where M is the

account for the largest possible number of digits to check. Combining the nested loops 0(n^2) with the operation to find the maximum digit 0(log M), the overall time complexity of the code is $0(n^2 * log M)$.

value of the number), finding the maximum digit is O(log M). M here represents the maximum value found within the array to

Space Complexity

The space complexity is 0(1). Aside from the input list nums, the only extra space used by the algorithm is a fixed number of single-value variables (like ans, x, y, v) that do not depend on the size of the input. The algorithm does not allocate any additional space that grows with the input size, hence it uses constant extra space.