2342. Max Sum of a Pair With Equal Sum of Digits

Medium Array Hash Table Sorting **Heap (Priority Queue)**

Problem Description

This problem presents us with an array nums filled with positive integers, and our goal is to find two different indices i and j, where i is not equal to j, such that the sum of the digits of nums[i] is equal to the sum of the digits of nums[j]. Once such indices are found, we need to return the maximum value of the sum nums[i] + nums[j] that we can obtain by considering all possible pairs of indices that meet the given condition.

Leetcode Link

To clarify, if we pick nums [i] as "123" and nums [j] as "51", both have the sum of digits equal to 6 and thus meet the criteria since 1+2+3 = 5+1.

Intuition

maximum pair that meets the criteria. However, this would be inefficient with a time complexity of O(n^2) which is not optimal for large arrays. The given solution leverages hashing to optimize the process. It uses a dictionary (or hash map) to keep track of the highest value

The immediate brute-force approach would be to compute the sum of digits for every pair of numbers in the array and then find the

number v for each unique sum of digits y. This way, when a new number is processed, we can easily check if there is already a stored number with the same sum of digits using the hash map. While iterating through the nums array, for each number:

2. If y is already in the dictionary d, it means we have encountered another number previously with the same sum of digits. We can

3. Iterate over each number v in the input nums array.

for the pair-wise comparison.

hash map operations (insert and lookup) are O(1) on average.

then:

1. We calculate the sum of digits y.

- Calculate a potential new maximum by adding the current number v and the stored number d[y]. Compare it with the current known maximum ans and update ans if the new potential maximum is greater.
- 3. Regardless of whether y was already present or not, we update the dictionary with the highest value v for the sum of digits y. We use max(d[y], v) to ensure we always keep the larger number for that sum of digits, which is crucial for maximizing the eventual
 - sum of nums[i] + nums[j].
- The result of this process is the maximum sum that can be formed under the given constraints, which the function returns. If no such pair exists, the answer remains as the initialized value of -1.

Solution Approach The implementation of the solution makes use of a hash map to efficiently track the maximum number encountered for each unique

values for non-existing keys. The algorithm proceeds as follows:

1. Initialize ans as -1. This will store the eventual maximum sum nums[i] + nums[j] if a valid pair is found. If no valid pairs are found, ans will return -1. 2. Initialize a defaultdict named d to store pairs of (sum of digits: maximum number with that sum).

sum of digits. The Python code utilizes a defaultdict from the collections module which simplifies the management of default

- Calculate the sum of the digits y of the current number v using a while loop that adds v % 10 to y and then floor divides v by 10. This loop effectively extracts and sums up each digit of the number.
- o If the sum of digits y is already a key in the dictionary d, then there exists a different number with the same sum of digits
 - encountered earlier. In this case: Compute the possible new maximum sum d[y] + v and compare it with ans. Update ans with this new maximum if it is greater.

Update the dictionary with the key y by ensuring it stores the maximum v: d[y] = max(d[y], v). This step is crucial since it

is possible to encounter multiple numbers with the same sum of digits and we are only interested in storing the largest one

The data structure used is crucial for optimizing the time complexity of this problem. By using a hash map, accessing and updating the maximum number for a sum of digits is done in constant time, bringing the overall time complexity of the algorithm down to O(n *

k), where n is the number of elements in nums and k is the average number of digits in numbers within nums. This is assuming that the

At the end of the loop, ans contains the maximum sum of nums[i] + nums[j] where sums of their digits are equal or remains -1 if no such pair exists. This value is then returned as the answer. Example Walkthrough

1. Initialize ans as -1. 2. Initialize the hash map d as a defaultdict with the default type as int. It will hold the sum of digits as the key and the respective

3. Start with the first number 42. The sum of its digits is 4 + 2 = 6. Since there is no entry in d with 6 as key, add it with d[6] = 42.

maximum number as the value.

4. Move to the next number 33. The sum of its digits is 3 + 3 = 6. There is already an entry with the sum 6, so we check if the sum

other pairs are left to be considered, we would return 102 as the result.

Initialize the maximum sum as -1 (assuming no answer is found yet)

Initialize a dictionary to store the maximum number for each digit sum

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

digit_sum_max_num = defaultdict(int)

sumOfDigits += tempNumber % 10;

if (maxNumWithDigitSum[sumOfDigits] > 0) {

// check if the two numbers form a larger pair sum.

// If there's already a number with the same digit sum encountered,

// Update the array with the maximum number for the current digit sum.

1 // Define the maxDigits value representing the maximum possible sum of digits which is 9*5=45

// Function to calculate the sum of the digits of a number

const sumOfDigits = (number: number): number => {

number = Math.floor(number / 10);

digitSum += number % 10;

// Return the maximum pair sum of numbers with the same digit sum, else -1.

maxPairSum = Math.max(maxPairSum, maxNumWithDigitSum[sumOfDigits] + number);

maxNumWithDigitSum[sumOfDigits] = Math.max(maxNumWithDigitSum[sumOfDigits], number);

Let's consider a small example using the array nums = [42, 33, 60].

- of d[6] (which is 42) and 33 is greater than ans. The current sum is 42 + 33 = 75, so we update ans = 75. Then we update d[6]
- to be the max of 42 and 33, which is 42, so no change is required. 5. Proceed to the last number 60. The sum of its digits is 6 + 0 = 6. Again, there's an entry for sum 6. We compute the potential
- then update d[6] with the new maximum number 60.

At the end of this process, ans holds the value 102, which is the maximum sum of nums[i] + nums[j] with equal digit sums. Since no

new maximum which is d[6] (42) plus 60 equals 102, and since it is greater than the current ans (75), we update ans = 102. We

Python Solution from collections import defaultdict

10 # Iterate through each number in the given list for num in nums: 11 12 # Calculate the sum of digits for the current number 13

14

15

13

14

15

16

17

18

19

20

21

22

23

24

25

27

26 }

class Solution:

 $\max sum = -1$

digit_sum = 0

temp_num = num

while temp_num:

```
16
                   digit_sum += temp_num % 10
17
                   temp_num //= 10
18
               # If the sum of digits has been seen before,
               # check if the current number contributes to a larger max sum
20
21
               if digit_sum in digit_sum_max_num:
22
                   max_sum = max(max_sum, digit_sum_max_num[digit_sum] + num)
23
24
               # Update the maximum number for the current digit sum
25
               digit_sum_max_num[digit_sum] = max(digit_sum_max_num[digit_sum], num)
26
27
           # Return the maximum sum of two numbers having the same sum of digits
28
           return max_sum
29
Java Solution
   class Solution {
       public int maximumSum(int[] nums) {
           // This variable will hold the answer, initialized to -1 as per the problem statement.
           int maxPairSum = -1;
           // This array will store the maximum number encountered for each digit sum.
           int[] maxNumWithDigitSum = new int[100];
           // Iterate through all the numbers in the input array.
           for (int number : nums) {
9
               int sumOfDigits = 0;
10
               // Calculate the sum of digits of the current number.
11
               for (int tempNumber = number; tempNumber > 0; tempNumber /= 10) {
```

C++ Solution

1 class Solution {

return maxPairSum;

```
2 public:
       // Function to calculate the maximum sum of a pair of numbers with the same sum of digits
       int maximumSum(vector<int>& numbers) {
           // Initialize a vector of vectors to group numbers by their digit sums (up to 81 for 99999, which is maximum 5*9)
           vector<vector<int>> digitSumGroups(100);
           // Iterate through each number to calculate their digit sums and group them
           for (int& number : numbers) {
 9
               int digitSum = 0;
10
               // Calculate the sum of digits for the current number
11
               for (int value = number; value > 0; value /= 10) {
12
13
                   digitSum += value % 10;
14
15
               // Add the number to its corresponding digit sum group
16
               digitSumGroups[digitSum].emplace_back(number);
17
18
           int maxPairSum = -1; // Initialize max pair sum as -1 to handle cases with no valid pair
19
20
           // Iterate through all digit sum groups
           for (auto& group : digitSumGroups) {
22
               // Check if there are at least two numbers in the current digit sum group
23
               if (group.size() > 1) {
24
                   // Sort the numbers within the current group in descending order
                   sort(group.rbegin(), group.rend());
25
26
                   // Update the maxPairSum with the sum of the top two numbers in the current group
27
                   maxPairSum = max(maxPairSum, group[0] + group[1]);
28
29
30
           // Return the maximum pair sum found
31
           return maxPairSum;
32
33 };
34
Typescript Solution
```

13

9

10

11

2 const MAX_DIGITS_SUM = 45;

let digitSum = 0;

return digitSum;

while (number > 0) {

```
12 };
 14 // Function to calculate the maximum sum of a pair of numbers with the same sum of digits
 15 const maximumSum = (numbers: number[]): number => {
        // Initialize an array to group numbers by their digit sums
 16
 17
         const digitSumGroups: number[][] = Array.from({ length: MAX_DIGITS_SUM + 1 }, () => []);
 18
 19
         // Group numbers by their digit sum
         numbers.forEach(number => {
 20
             const digitSum = sumOfDigits(number);
 21
             digitSumGroups[digitSum].push(number);
 22
 23
         });
 24
 25
         let maxPairSum = -1; // Initialize max pair sum as -1 to indicate no valid pair yet
 26
 27
         // Iterate through all digit sum groups to find the maximum pair sum
         digitSumGroups.forEach(group => {
 28
             if (group.length > 1)
 29
 30
                 // Sort the group in descending order
                 group.sort((a, b) \Rightarrow b - a);
 31
 32
                // Update the maxPairSum with the sum of the two largest numbers in the current group
                maxPairSum = Math.max(maxPairSum, group[0] + group[1]);
 33
 34
 35
         });
 36
 37
         // Return the maximum pair sum found, or -1 if no valid pairs exist
         return maxPairSum;
 38
 39
    };
 40
    // The functions can now be used globally as part of the TypeScript codebase
 42
Time and Space Complexity
Time Complexity
The time complexity of the given code can be analyzed by considering two major parts:
 1. Iterating through each number in the nums list.
```

2. Summing the digits of each number and updating the dictionary.

For the first part, iterating through the nums list is O(n) where n is the length of nums. For the second part, we must consider the number of digits in each number for the summation of digits and updating the dictionary.

the number of digits. Since for each number, we perform a digit sum and update the dictionary, the overall time complexity for this part is 0(d * n).

Combining both parts, the total time complexity is 0(n * d). Note that if the input numbers have a bounded size (for example, they are all 32-bit integers), d can be considered a constant, and

The number of digits d in a number v is proportional to log10(v). Hence, summing the digits of a number takes 0(d) time where d is

Space Complexity

the time complexity can be viewed as O(n).

The space complexity is determined by the space required to store the d dictionary which holds the maximum number encountered for each digit sum.

• The number of unique digit sums is at most 9*m, where m is the maximum number of digits in a number of the nums list since the

• Each entry in the dictionary stores an integer value (which is constant space). Thus, the space complexity is O(m), where m represents the maximum number of digits across all numbers in nums.

largest digit sum for a number with m digits would be 9 * m (if each digit is 9).

If we again assume the input numbers are all 32-bit integers, m is a constant (10, since 2^31 - 1 has 10 digits), making the space complexity 0(1).