Digit Sum Comparison Algorithm

Multi-Language Implementation Guide

Problem Analysis

This program reads two integers and compares the sum of their digits. It outputs the numbers in ascending order based on their digit sums. If the digit sums are equal, it maintains the original order (first number, then second number).

Algorithm Logic

1. Read two integers `i` and `t`
2. Calculate the sum of digits for each number
3. Compare digit sums and output numbers in ascending order of their digit sums

---

Original C Code Analysis

```c
#include <stdio.h>
int main() {
    int i,t;
    int a=0;
    int b=0;
    scanf("%d %d",&i,&t);
    int m=i,n=t;
    while(m!=0) {
        a+=m%10;
        m/=10;
    }
    while(n!=0) {
        b+=n%10;
        n/=10;
    }
    if(a<=b)
        printf("%d %d",i,t);
    else
        printf("%d %d",t,i);
}
```

Line-by-Line Explanation

```c
#include <stdio.h>
```

Line 1: Include standard input/output library for `scanf` and `printf` functions.

```c
int main() {
```

Line 2: Main function declaration - entry point of the program.

```c
int i,t;
int a=0;
int b=0;
```

Lines 3-5: Variable declarations:

- `i, t`: Store the two input integers
- `a, b`: Store the sum of digits for `i` and `t` respectively, initialized to 0

```c
scanf("%d %d",&i,&t);
```

Line 6: Read two integers from user input and store them in variables `i` and `t`.

```c
int m=i,n=t;
```

Line 7: Create copies of the input numbers to preserve original values during digit extraction.

```c
while(m!=0) {
   a+=m%10;
   m/=10;
}
```

Lines 8-11: Calculate sum of digits for first number:

- `m%10`: Extract the last digit
- `a+=m%10`: Add the digit to sum
- `m/=10`: Remove the last digit by integer division

```c
while(n!=0) {
    b+=n%10;
    n/=10;
}
```

Lines 12-15: Calculate sum of digits for second number using same logic.

```c
if(a<=b)
    printf("%d %d",i,t);
else
    printf("%d %d",t,i);
```

Lines 16-19: Compare digit sums and output:

- If first number's digit sum ≤ second number's digit sum: output in original order
- Otherwise: output in reverse order

Multi-Language Implementations

C (Improved Version)

```c
```

```
#include <stdio.h>

int digitSum(int num) {
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

int main() {
    int i, t;
    scanf("%d %d", &i, &t);

    int sumI = digitSum(i);
    int sumT = digitSum(t);

    if (sumI <= sumT) {
        printf("%d %d\n", i, t);
    } else {
        printf("%d %d\n", t, i);
    }

    return 0;
}
```

C++

```
cpp
```

```
#include <iostream>
using namespace std;

int digitSum(int num) {
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

int main() {
    int i, t;
    cin >> i >> t;

    int sumI = digitSum(i);
    int sumT = digitSum(t);

    if (sumI <= sumT) {
        cout << i << " " << t << endl;
    } else {
        cout << t << " " << i << endl;
    }

    return 0;
}
```

C#

```
csharp
```

```
using System;

class Program {
    static int DigitSum(int num) {
        int sum = 0;
        while (num != 0) {
            sum += num % 10;
            num /= 10;
        }
        return sum;
    }

    static void Main() {
        string[] input = Console.ReadLine().Split();
        int i = int.Parse(input[0]);
        int t = int.Parse(input[1]);

        int sumI = DigitSum(i);
        int sumT = DigitSum(t);

        if (sumI <= sumT) {
            Console.WriteLine($"{i} {t}");
        } else {
            Console.WriteLine($"{t} {i}");
        }
    }
}
```

Rust

```
rust
```

```
use std::io;

fn digit_sum(mut num: i32) -> i32 {
    let mut sum = 0;
    while num != 0 {
        sum += num % 10;
        num /= 10;
    }
    sum
}

fn main() {
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Failed to read line");

    let numbers: Vec<i32> = input
        .trim()
        .split_whitespace()
        .map(|s| s.parse().expect("Invalid number"))
        .collect();

    let i = numbers[0];
    let t = numbers[1];

    let sum_i = digit_sum(i);
    let sum_t = digit_sum(t);

    if sum_i <= sum_t {
        println!("{} {}", i, t);
    } else {
        println!("{} {}", t, i);
    }
}
```

Python (Bonus)

```
python
```

```
def digit_sum(num):
    return sum(int(digit) for digit in str(abs(num)))

# Alternative implementation using while loop (like original C code)
def digit_sum_while(num):
    num = abs(num)  # Handle negative numbers
    total = 0
    while num != 0:
        total += num % 10
        num //= 10
    return total

def main():
    i, t = map(int, input().split())

    sum_i = digit_sum(i)
    sum_t = digit_sum(t)

    if sum_i <= sum_t:
        print(i, t)
    else:
        print(t, i)

if __name__ == "__main__":
    main()
```

Example Walkthrough

Input: `23 41`

Step 1: Read numbers

- `i = 23`, `t = 41`

Step 2: Calculate digit sum for 23

- Extract digits: 3, 2
- Sum: 2 + 3 = 5
- `a = 5`

Step 3: Calculate digit sum for 41

- Extract digits: 1, 4

- Sum: 4 + 1 = 5
- `b = 5`

Step 4: Compare and output

- `a <= b` (5 <= 5) is true
- Output: `23 41`

Input: `456 123`

Step 1: Read numbers

- `i = 456`, `t = 123`

Step 2: Calculate digit sum for 456

- Extract digits: 6, 5, 4
- Sum: 4 + 5 + 6 = 15
- `a = 15`

Step 3: Calculate digit sum for 123

- Extract digits: 3, 2, 1
- Sum: 1 + 2 + 3 = 6
- `b = 6`

Step 4: Compare and output

- `a <= b` (15 <= 6) is false
- Output: `123 456`

Algorithm Analysis

Time Complexity

- **O(log n)** where n is the larger of the two input numbers
- Each number requires O(log n) time to extract all digits

Space Complexity

- **O(1)** - Only uses a constant amount of extra space

Edge Cases

1. **Equal digit sums:** Output maintains original order
2. **Single digit numbers:** Works correctly (sum equals the number itself)
3. **Numbers with leading zeros:** Not applicable for integer input
4. **Negative numbers:** Original code doesn't handle negatives properly

---

Key Programming Concepts

Digit Extraction Pattern

```
while (num != 0) {
    digit = num % 10;    // Get last digit
    sum += digit;        // Process digit
    num /= 10;           // Remove last digit
}
```

This pattern is fundamental for:

- Calculating digit sums
- Reversing numbers
- Checking palindromes
- Converting number bases

Language-Specific Features

Language	Input Method	Output Method	Key Features
C	`scanf()`	`printf()`	Manual memory management
C++	`cin/cout`	`cin/cout`	Object-oriented, STL
C#	`Console.ReadLine()`	`Console.WriteLine()`	.NET framework, managed memory
Rust	`stdin().read_line()`	`println!`	Memory safety, ownership

Common Mistakes & Improvements

Original Code Issues

1. **No input validation**
2. **Variable naming could be clearer**
3. **No function decomposition**

4. **Missing return statement**

Improvements Made

1. **Function extraction:** `digitSum()` function for reusability
2. **Better variable names:** `sumI`, `sumT` instead of `a`, `b`
3. **Input validation:** Added in some implementations
4. **Proper return statements:** Added `return 0`