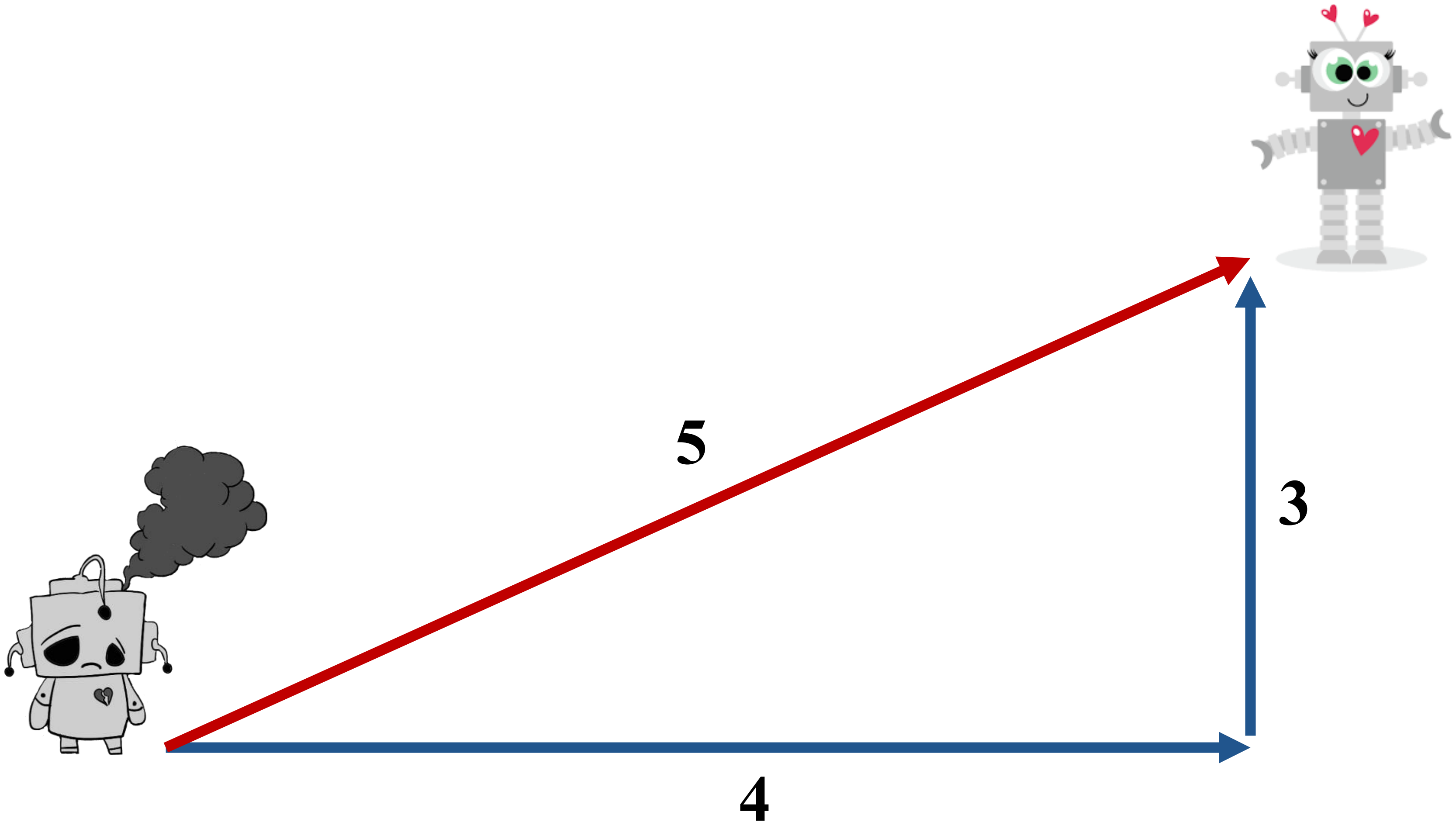
Design and Analysis of Algorithms

Problems



Problems

Write a program to calculate the sum of numbers from 1 to 1,000,000

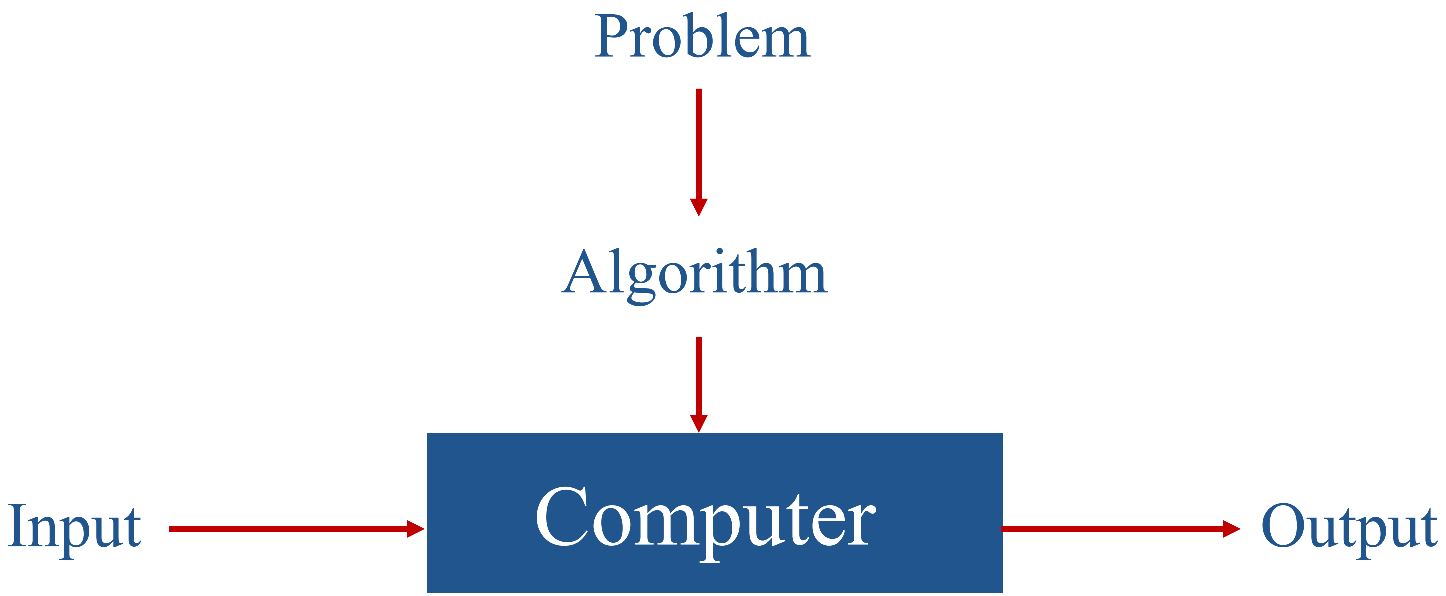
```
unsigned long sum = 0;
for (unsigned long i = 1; i <= 1e6; i++)
        sum += i;
```

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

```
unsigned long n = 1e6;
unsigned long sum = n*(n+1)/2;
```

What Is an Algorithm?

- An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.



Algorithms

- We can consider algorithms to be procedural solutions to problems.
- These solutions are not answers but specific instructions for getting answers.
- Algorithms is more than a branch of computer science.
- It is the core of computer science.
- Computer programs would not exist without algorithms.

Design and Analysis of Algorithms

Two main issues related to algorithms:

- How to **design** algorithms?
- How to **analyze** algorithm efficiency?

Greatest Common Divisor (GCD)

- The greatest common divisor of two nonnegative, not-both-zero integers m and n, denoted $\gcd(m, n)$, is defined as the largest integer that divides both m and n evenly.
- For example, the GCD of 8 and 12 is 4.

  The divisors of 8 are 1, 2, **4**, 8

  The divisors of 12 are 1, 2, 3, **4**, 6, 12

  $$\gcd(8, 12) = 4$$
- Just as with many other problems, there are several algorithms for computing the greatest common divisor.

Consecutive Integer Checking Algorithm

- A common divisor cannot be greater than the smaller of these numbers, which we will denote by $t = \min\{m, n\}$.
- So we can start by checking whether t divides both m and n.
- If it does, t is the answer.
- if it does not, we simply decrease t by 1 and try again.
- How do we know that the process will eventually stop?
- For example, for numbers 60 and 24, the algorithm will try first 24, then 23, and so on, until it reaches 12, where it stops.

Consecutive Integer Checking Algorithm: Pseudocode

ALGORITHM *ConsecutiveIntegerChecking(m, n)*

//Computes gcd*(m, n)* by Consecutive Integer Checking Algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

$t \leftarrow \min(m, n)$

while $t > 0$ **do**

  if $m \bmod t = 0$ **and** $n \bmod t = 0$ **then**

    break

  $t \leftarrow t - 1$

return t

Consecutive Integer Checking Algorithm: Structured Description

Step 1 Assign the value of min$\{m, n\}$ to t.

Step 2 Divide m by t.

If the remainder of this division is 0, go to Step 3;

otherwise, go to Step 4.

Step 3 Divide n by t.

If the remainder of this division is 0, return the value of t as the answer and stop;

otherwise, proceed to Step 4.

Step 4 Decrease the value of t by 1.

Go to Step 2.

Euclid’s Algorithm

- Euclid’s algorithm is based on applying repeatedly the equality

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

until $m \bmod n$ is equal to 0.

- For example, gcd(60, 24) can be computed as follows:

$$\gcd(60, 24) = \gcd(24, 12) = \gcd(12, 0) = 12.$$

- Hence, the value of the second integer eventually becomes 0, and the algorithm stops.

Euclid's Algorithm: Pseudocode

ALGORITHM *Euclid(m, n)*

//Computes gcd(m, n) by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

while $n \neq 0$ **do**

  $r \leftarrow m \bmod n$

  $m \leftarrow n$

  $n \leftarrow r$

return m

Euclid’s Algorithm: Pseudocode

Step 1 If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.

Step 2 Divide m by n and assign the value of the remainder to r.

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

Algorithm Design Techniques

- Brute Force and Exhaustive Search
- Divide-and-Conquer
- Decrease-and-Conquer
- Transform-and-Conquer
- Space and Time Trade-Offs
- Dynamic Programming
- Greedy Technique
- Iterative Improvement
- Backtracking
- Branch-and-Bound

Analysis of Algorithms

- Time Efficiency
- Space Efficiency