• **Piazza**
  • You should have gotten an invite, if not, please email me ASAP

• **Recitations**
  • Assignments will still be hosted on the HUST virtual judge

• **Always bring a laptop to recitation**
  • Unless otherwise noted
  • If this is infeasible, please email/see me after class
Problem A

AUTOMATIC ANSWER

File name: A.(java,c,cpp)

Last month Alice nonchalantly entered her name in a draw for a Tapmaster 4000. Upon checking her mail today, she found a letter that read:

“Congratulations, Alice! You have won a Tapmaster 4000. To claim your prize, you must answer the following skill testing question.”

Alice’s initial feelings of surprised joy turned quickly to those of dismay. Her lifetime record for skill testing questions is an abysmal 3 right and 42 wrong.

Mad Skills, the leading skill testing question development company, was hired to provide skill testing questions for this particular Tapmaster 4000 draw. They decided to create a different skill testing question to each winner so that the winners could not collaborate to answer the question.

Can you help Alice win the Tapmaster 4000 by solving the skill testing question?

Program Input

The input begins with \( t (1 \leq t \leq 100) \), the number of test cases. Each test case contains an integer \( n (-1000 \leq n \leq 1000) \) on a line by itself. This \( n \) should be substituted into the skill testing question below.

Program Output

For each test case, output the answer to the following skill testing question on a line by itself: “Multiply \( n \) by 567, then divide the result by 9, then add 7492, then multiply by 235, then divide by 47, then subtract 498. What is the digit in the tens column?”
Problem A

AUTOMATIC ANSWER

Problem narrative
- Can be unnecessarily long or misleading

Input and output description
- Usually very precise
- You may assume that all input will be formatted like this

Sample input and output
- One or more inputs and expected outputs

Last month Alice nonchalantly entered her name in a draw for a Tapmaster 4000. Upon checking her mail today, she found a letter that read:

"Congratulations, Alice! You have won a Tapmaster 4000. To claim your prize, you must answer the following skill testing question."

Alice's initial feelings of elation turned quickly to those of dismay. Her lifetime record for skill testing questions is an abysmal 1 right and 42 wrong.

Mad Skills, the leading skill testing question development company, was hired to provide skill testing questions for this particular Tapmaster 4000 draw. They decided to create a different skill testing question to each winner so that the winners could not collaborate to answer the question.

Can you help Alice win the Tapmaster 4000 by solving the skill testing question?

Program Input
The input begins with $n$ (1 ≤ n ≤ 1000), the number of test cases. Each test case contains an integer $a$ (2 ≤ a ≤ 1000) on a line by itself. This $a$ should be substituted into the skill testing question below.

Program Output
For each test case, output the answer to the following skill testing question on a line by itself: “Multiply $a$ by 985, then divide the result by 8, then add 294, then multiply by 195, then divide by 47, then subtract 498. What is the digit in the tens column?”
• Read the problem and extract key details
• Decide on an algorithm to solve the problem
  • What type of problem is it?
• Implement the solution
  • Be familiar with the language
  • Know how to parse I/O
• Test the solution
  • Check that the program works on the sample input/output
  • Come up with your own test cases
• Debug the solution
import java.io.*;

public class Main {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        int nCases = Integer.parseInt(in.readLine());

        for (int caseNum = 0; caseNum < nCases; caseNum++) {
            // Parse the input number
            int n = Integer.parseInt(in.readLine());

            // Calculate the answer
            n *= 567;
            n /= 9;
            n += 7492;
            n *= 235;
            n /= 47;
            n -= 498;

            // Digit in the tens column
            int tens = (n / 10) % 10;

            // Print it out!
            System.out.println(tens);
        }
    }
}
• Most problems will have numerous test cases
• Different problems ask for different ways of parsing test cases
  • e.g., Automatic Answer tells you how many test cases there are
  • Some problems say “parse until a termination line of all zeros”
  • Others will have you read until end of file
import java.io.*;

public class Main {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        
        while (true) {
            // Parse the input number
            int n = Integer.parseInt(in.readLine());
            
            // Quit if the input is -99999
            if (n == -99999) {
                break;
            }
            
            // Calculate the answer
            n *= 567;
            n /= 9;
            n += 7492;
            n *= 235;
            n /= 47;
            n -= 498;
            
            // Digit in the tens column
            int tens = (n / 10) % 10;
            
            // Print it out!
            System.out.println(tens);
        }
    }
}
import java.io.*;

public class Main {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        String line;
        while ((line = in.readLine()) != null) {
            // Parse the input number
            int n = Integer.parseInt(line);

            // Calculate the answer
            n *= 567;
            n /= 9;
            n += 7492;
            n *= 235;
            n /= 47;
            n -= 498;

            // Digit in the tens column
            int tens = (n / 10) % 10;

            // Print it out!
            System.out.println(tens);
        }
    }
}
• Read the problem and extract key details
  • Pay attention to constraints on input and formatting specs
  • If any key details are missing, you can request clarification
• Decide on an algorithm to solve the problem
  • What type of problem is it?
  • UVA has limit about 512MB; stack size about 8.9MB giving a recursion depth limit around 475K for a very simple function
• Implement the solution
• Test the solution
  • java -Xss8900K -Xmx512M MyClass < Test.in
  • g++ -O2 -std=c++11
• Debug the solution
• Once you have an algorithm for a solution in mind, will it run fast enough and in the memory given?
• Good strategy is to think of several algorithms that will work and pick the easiest one to implement
• Rule of thumb: modern computer can process up to 100M operations within a few seconds
  • Use this to determine if your algorithm will run in time
  • Standard time limit on the virtual judge is 3s
• Program bounds are one of the most important parts of the problem statement
• Beware of high overhead times, floating point calculations, I/O time
• Be familiar with these bounds:
  • $2^{10} = 1024$, $2^{20} \approx 1M$
  • Ints are 32 bits +/-2B, long longs are 64 bits (+/- $9 \times 10^{18}$)
  • Usually input in the problems we’ll see are upper bounded by 1M, beyond that I/O time becomes a factor
<table>
<thead>
<tr>
<th>$n$</th>
<th>Worst AC Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq [10..11]$</td>
<td>$O(n!)$, $O(n^6)$</td>
<td>e.g. Enumerating permutations (Section 3.2)</td>
</tr>
<tr>
<td>$\leq [15..18]$</td>
<td>$O(2^n \times n^2)$</td>
<td>e.g. DP TSP (Section 3.5.2)</td>
</tr>
<tr>
<td>$\leq [18..22]$</td>
<td>$O(2^n \times n)$</td>
<td>e.g. DP with bitmask technique (Section 8.3.1)</td>
</tr>
<tr>
<td>$\leq 100$</td>
<td>$O(n^4)$</td>
<td>e.g. DP with 3 dimensions + $O(n)$ loop, $nC_{k=4}$</td>
</tr>
<tr>
<td>$\leq 400$</td>
<td>$O(n^3)$</td>
<td>e.g. Floyd Warshall’s (Section 4.5)</td>
</tr>
<tr>
<td>$\leq 2K$</td>
<td>$O(n^2 \log_2 n)$</td>
<td>e.g. 2-nested loops + a tree-related DS (Section 2.3)</td>
</tr>
<tr>
<td>$\leq 10K$</td>
<td>$O(n^2)$</td>
<td>e.g. Bubble/Selection/Insertion Sort (Section 2.2)</td>
</tr>
<tr>
<td>$\leq 1M$</td>
<td>$O(n \log_2 n)$</td>
<td>e.g. Merge Sort, building Segment Tree (Section 2.3)</td>
</tr>
<tr>
<td>$\leq 100M$</td>
<td>$O(n)$, $O(\log_2 n)$, $O(1)$</td>
<td>Most contest problem has $n \leq 1M$ (I/O bottleneck)</td>
</tr>
</tbody>
</table>

Table 1.4: Rule of thumb time complexities for the ‘Worst AC Algorithm’ for various single-test-case input sizes $n$, assuming that your CPU can compute $100M$ items in 3s.
In each test case you are given some permutation of the numbers 1, …, \( n \) as a space-delimited list with one number missing. Determine the missing number and output it using one line per test case. Here \( 2 \leq n \leq 10^6 \). Each test case is on a separate line, and there are 5 cases. The input is terminated by EOF.
• **Solution 1: Sorting**
  1. Sort the input
  2. Iterate through sorted list, find missing element

• **Exercise**: what is the running time of this solution?

• **Make it more efficient**
  • Use binary search instead of linear search

• **Can we do better?**
• **Solution 2: Adding**
  - You have a closed form solution $1 + 2 + \ldots n$
  - Iterate through input and subtract from expected sum

• **Exercise:** any pitfalls with this solution?
• **Exercise:** what is the expected running time?
• **Solution 3: XOR**
  • Similar to Solution 2, but use XOR (^) instead of addition
• Properties of XOR:
  • \( x ^ x = 0, x ^ 0 = x \)
• Benefit: no need to worry about overflows
Suppose someone writes the following Java snippet to compute 100!:

```java
long result = 1;
for (int i = 1; i <= 100; ++i) {
    result *= i;
}
System.out.println(result);
```

What is written to the screen?
• Recall that in Java, longs are 64 bits
• When overflows occur, the lowest 64 bits are kept
• How many 0-bits will be in the least-significant 64 bits of 100!?
  • How many times do $2^n$ divide into 1, 2, ..., 100?

$$\left\lfloor \frac{100}{2} \right\rfloor + \left\lfloor \frac{100}{2^2} \right\rfloor + \left\lfloor \frac{100}{2^3} \right\rfloor + \left\lfloor \frac{100}{2^4} \right\rfloor + \left\lfloor \frac{100}{2^5} \right\rfloor + \left\lfloor \frac{100}{2^6} \right\rfloor = 50 + 25 + 12 + 6 + 3 + 1 = 97$$
Given a set of up to 10 distinct integers given as a space-delimited list, output all distinct non-empty subsets that sum to zero. Each individual subset should be output as a space-delimited list in the order it occurred in the input, and the list of subsets should be output in lexicographic order, one per line. For example, the input 0 −4 4 will yield −4 4, 0, and 0 −4 4 on three separate lines.
• **Exercise**: how many possible subsets of 10 elements are there?
• What is lexicographic order?
  • Outputting lists in lexicographic order is very common in programming contests

**Definition**: lexicographic order:
Given $A = a_1a_2a_3...a_j$ and $B = b_1b_2b_3...b_k$ then $A$ comes before $B$ iff at the first $i$ where $a_i \neq b_i$, $a_i$ comes before $b_i$. If one is a prefix of the other, then the shorter one comes first.
static class LexicoComp implements Comparator<ArrayList<Integer>> {
    public int compare(ArrayList<Integer> a, ArrayList<Integer> b) {
        int m = Math.min(a.size(), b.size());
        for (int i = 0; i < m; ++i) {
            int av = a.get(i), bv = b.get(i);
            if (av != bv)
                return av < bv ? -1 : 1;
        }
        if (a.size() != b.size())
            return a.size() < b.size() ? -1 : 1;
        return 0;
    }
}

Lexicographic Comparator
A coin is flipped \( n \) times, where \( n \) is given on a single line. You win if there is ever a run of 7 heads or 7 tails in a row. What is the probability that you win? Here \( 1 \leq n \leq 100 \). Output the result with 6 digits after the decimal.
Recurrence: \( P(\text{run\_length} = R, \text{tosses\_left} = T) = \)

- 1, if \( R = 7 \)
- 0, if \( R + T < 7 \)
- \( \frac{P(\text{run\_length} = R + 1, \text{tosses\_left} = T - 1) + P(\text{run\_length} = 1, \text{tosses\_left} = T - 1)}{2} \)

Memoize!
• Should you use Java or C++?
  • Java:
    • Easier to learn than C++ (you don’t have to worry about pointers and references)
    • Array indexing exceptions (C++ will just die with a segfault)
    • Excellent string libraries
    • Extensive library including BigInteger, BigDecimal, GregorianCalendar, regexes
      • More recent contests have started to de-emphasize these advantages of Java
  • C++:
    • Use C++11
    • Faster
    • Preprocessor (#DEFINE) and typedefs can abbreviate commonly used idioms
    • More built-in algorithms (e.g., next_permutation)
// Shortcuts for "common" data types in contests, especially TC
typedef long long ll
typedef pair<int, int> ii
typedef vector<ii> vii
typedef vector<int> vi
#define INF 1000000000

Just remember not to bring this to work!
• In an engineering environment, readability and maintainability is of very high importance
• For this class and competition, either language is fine
• Some things to keep in mind:
  • Java:
    • Commonly used objects: BigInteger, Random, ArrayList, Arrays, Collections, HashMap, HashSet, PriorityQueue, TreeMap, TreeSet, ArrayDeque, StringBuilder, StringTokenizer, BufferedReader, Comparator, Scanner, String
    • Strings are immutable, therefore costly operations. Use StringBuilder
  • C++:
    • Commonly used STL data structures: vector, map, set, hash_set,multimap, deque, queue, stack, sort, next_permutation.
Suppose your program repeatedly iterates over a list of numbers in sequence. Would it be faster to use an int[], ArrayList, or LinkedList and why?
Which will run faster and why?

a) int[][] arr = new int[1000000][2];
b) int[][] arr = new int[2][1000000];
Suppose you need to build a TreeMap of $10^6$ strings and then query it $10^6$ times, what is the estimated running time?
• Given a string of brackets, ‘()’, ‘{}’, ‘[]’, how do you determine if they are correctly matched?
• E.g., ‘(())’, ‘{}’, ‘()[]{{}}’ are correctly matched
• E.g., ‘(’, ‘((())’, ‘}‘ are incorrectly matched
• Understand the memory hierarchy
• Write profiling tests, run them many times
• Logging and instrumentation incur a cost
• Use tools! E.g., Valgrind
• Our rule of thumb (30ns/op) is just an estimate, but will give you the right order of magnitude
Next class we will start talking about basic data structures (stacks, queues), binary search, union-find
Sections 1.1, 1.2, 1.3