Algorithmic Toolbox at Coursera: Programming Challenges (Week 1)

February 19, 2020

Contents

To introduce you to our automated grading system, we will discuss two simple programming challenges and walk you through a step-by-step process of solving them. We will encounter several common pitfalls and will show you how to fix them.

Below is a brief overview of what it takes to solve a programming challenge in five steps:

Reading problem statement. The problem statement specifies the input-output format, the constraints for the input data as well as time and memory limits. Your goal is to implement a fast program that solves the problem and works within the time and memory limits.

Designing an algorithm. When the problem statement is clear, start designing an algorithm and don't forget to prove that it works correctly.

Implementing an algorithm. After you developed an algorithm, start implementing it in a programming language of your choice.

Testing and debugging your program. Testing is the art of revealing bugs. Debugging is the art of exterminating the bugs. When your program is ready, start testing it! If a bug is found, fix it and test again.

Submitting your program to the grading system. After testing and debugging your program, submit it to the grading system and wait for the message "Good job!". In the case you see a different message, return back to the previous stage.

1   Sum of Two Digits

Sum of Two Digits Problem
Compute the sum of two single digit numbers.

Input: Two single digit numbers.
Output: The sum of these numbers.

$$2 + 3 = 5$$

We start from this ridiculously simple problem to show you the pipeline of reading the problem statement, designing an algorithm, implementing it, testing and debugging your program, and submitting it to the grading system.

Input format. Integers a and b on the same line (separated by a space).

Output format. The sum of a and b.

Constraints. $0 \le a, b \le 9$.

Sample.

Input:

```
9 7
```

Output:

```
16
```

Time limits (sec.):

C	C++	Java	Python	C#	Haskell	JavaScript	Kotlin	Ruby	Rust	Scala
1	1	1.5	5	1.5	2	5	1.5	5	1	3

Memory limit. 512 Mb.

1.1 Implementing an Algorithm

For this trivial problem, we will skip "Designing an algorithm" step and will move right to the pseudocode.

SUMOFTWODIGITS(a, b):
return $a + b$

Since the pseudocode does not specify how we input a and b, below we provide solutions in C++, Java, and Python3 programming languages as well as recommendations on compiling and running them. You can copy-and-paste the code to a file, compile/run it, test it on a few datasets, and then submit (the source file, not the compiled executable) to the grading system. Needless to say, we assume that you know the basics of one of programming languages that we use in our grading system.

C++

```cpp
#include <iostream>

int sum_of_digits(int first_digit, int second_digit) {
    return first_digit + second_digit;
}

int main() {
    int a = 0;
    int b = 0;
    std::cin >> a;
    std::cin >> b;
    std::cout << sum_of_digits(a, b);
    return 0;
}
```

Save this to a file (say, aplusb.cpp), compile it, run the resulting executable, and enter two numbers (on the same line).

5

Java

```java
import java.util.Scanner;

class SumOfDigits {
    static int sumOfDigits(int first_digit, int second_digit) {
        return first_digit + second_digit;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int a = s.nextInt();
        int b = s.nextInt();
        System.out.println(sumOfDigits(a, b));
    }
}
```

Save this to a file SumOfDigits.java, compile it, run the resulting executable, and enter two numbers (on the same line).

Python3

```python
# python3


def sum_of_digits(first_digit, second_digit):
    return first_digit + second_digit

if __name__ == '__main__':
    a, b = map(int, input().split())
    print(sum_of_digits(a, b))
```

Save this to a file (say, aplusb.py), run it, and enter two numbers on the same line. (The first line in the code above tells the grading system to use Python3 rather Python2.)

Your goal is to implement an algorithm that produces a correct result under the given time and memory limits for any input satisfying the given constraints. You do not need to check that the input data satisfies the constraints, e.g., for the Sum of Two Digits Problem you do not need to check that the given integers a and b are indeed single digit integers (this is guaranteed).

1.2 Submitting to the Grading System at Coursera

This is what a fresh submission page looks like:

Programming Assignment: Programming Assignment 1: Programming Challenges

You have not submitted. You must earn 2/2 points to pass.

ⓘ It looks like this is your first programming assignment. Learn more   ✕

Deadline   Pass this assignment by January 14, 11:59 PM PST

Instructions   My submission   Discussions

The primary goal of this programming assignment is to become familiar with the grading framework and testing/debugging techniques.

Please download the following archive file. It contains problem statements and starter files for all the programming challenges in this course. In particular, the file

```
1    week1_programming_challenges/week1_programming_challenges.pdf
```

How to submit

When you're ready to submit, you can upload files for each part of the assignment on the "My submission" tab.

contains detailed instructions on solving programming challenges. Please read the document and then submit solutions to the the programming challenges under the "My submission" tab.

The blue bubble suggests you to read a general information about programming assignments. We encourage you to read it if you haven't solved programming assignments before.

The line above the bubble indicates that there are two programming challenges in this assignment. To pass this assignment, you need to solve both programming challenges. This should not be difficult as in this document we give detailed instructions for solving them. For all the remaining programming assignments, the passing threshold is usually around 50%. To pass the course, you need to pass all the programming assignments.

The line below the blue bubble indicates the deadline for this assignment. Please not that this is a suggestion rather than a hard deadline. If you miss a deadline, you may safely switch to the next session of the course (a new session starts automatically every two weeks). All your progress will be transferred to the new session in this case. See Coursera help article on switching sessions.

At the right, there is a link to the discussion forum attached to this assignment. We encourage you to visit the forum on a regular basis to ask

questions and to help other learners. Note also that for each programming challenge, there is a separate thread at the forum. The link to this particular thread can be found at the end of the programming challenge statement.

When you are ready to submit a solution to a programming challenge, go the the "My submission" tab that looks as follows:
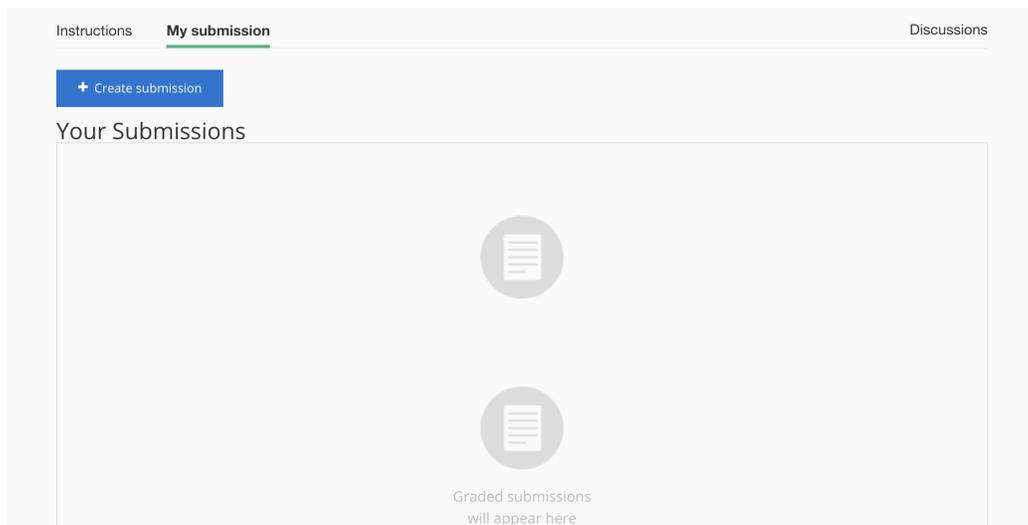
Click on the "Create submission" button. The page now looks as follows.
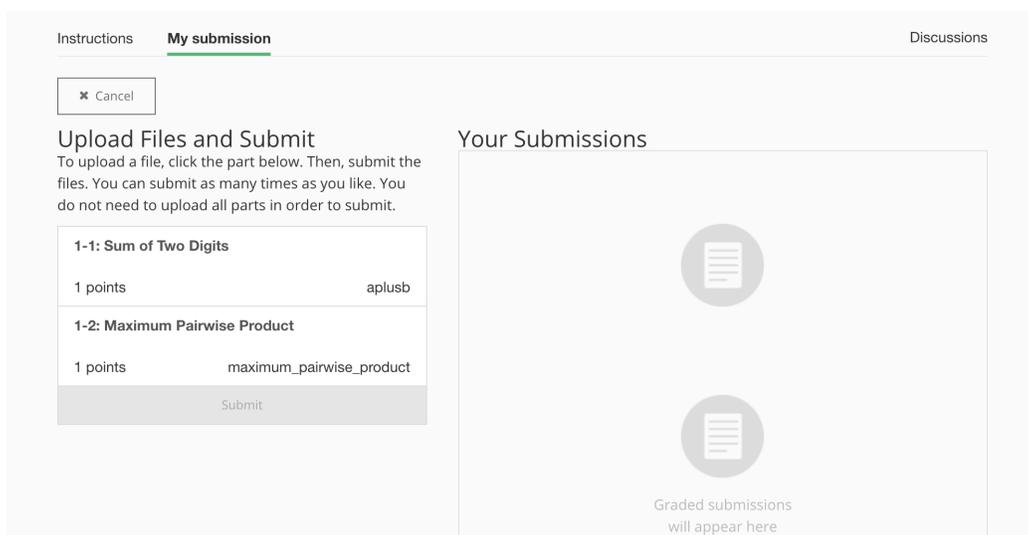
In this particular assignment, there are two programming challenges (Sum

of Two Digits and Maximum Pairwise Product). Click on a challenge you want to submit, upload a source file (rather than executable) of your solution, and press the "Submit" button. The page now looks as follows:
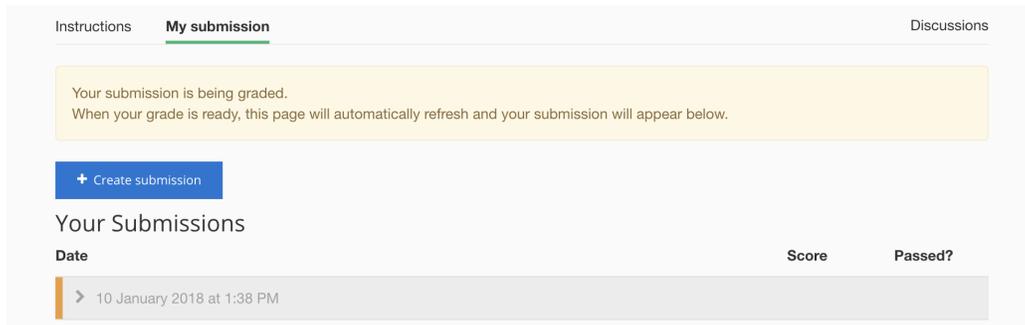
You may safely leave the page while your solution is being graded (in most cases, it is done in less than a minute, but when the servers are overloaded it may take several minutes; please be patient). When the grading is done, the submission page will look like this:
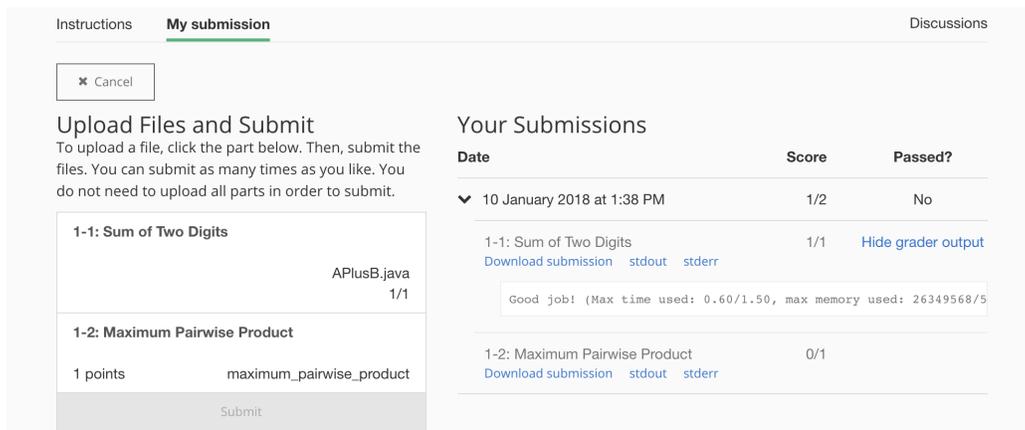
In this particular case, it says that the first programming challenge is passed (score is 1/1) but the whole programming assignment is not passed yet (since one needs to pass both programming challenges for this).

2 Maximum Pairwise Product

Maximum Pairwise Product Problem
Find the maximum product of two distinct numbers in a sequence of non-negative integers.

Input: A sequence of non-negative integers.
Output: The maximum value that can be obtained by multiplying two different elements from the sequence.

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		14	8
7	35	42	14		28
4	20	24	8	28	

Given a sequence of non-negative integers $a_1, \ldots, a_n$, compute

$$\max_{1 \le i \ne j \le n} a_i \cdot a_j.$$

Note that i and j should be different, though it may be the case that $a_i = a_j$.

Input format. The first line contains an integer n. The next line contains n non-negative integers $a_1, \ldots, a_n$ (separated by spaces).

Output format. The maximum pairwise product.

Constraints. $2 \le n \le 2 \cdot 10^5$; $0 \le a_1, \ldots, a_n \le 2 \cdot 10^5$.

Sample 1.
Input:
```
3
1 2 3
```
Output:
```
6
```

Sample 2.

    Input:

```
10
7 5 14 2 8 8 10 1 2 3
```

    Output:

```
140
```

Time and memory limits. The same as for the previous problem.

2.1 Naive Algorithm

A naive way to solve the Maximum Pairwise Product Problem is to go through all possible pairs of the input elements $A[1\dots n] = [a_1,\dots,a_n]$ and to find a pair of distinct elements with the largest product:

```
MaxPairwiseProductNaive(A[1 ... n]):
product ← 0
for i from 1 to n:
    for j from 1 to n:
        if i ≠ j:
            if product < A[i] · A[j]:
                product ← A[i] · A[j]
return product
```

This code can be optimized and made more compact as follows.

```
MaxPairwiseProductNaive(A[1 ... n]):
product ← 0
for i from 1 to n:
    for j from i + 1 to n:
        product ← max(product, A[i] · A[j])
return product
```

    Implement this algorithm in your favorite programming language. If you are using C++, Java, or Python3, you may want to download the starter files (we provide starter solutions in these three languages for all the problems in the book). For other languages, you need to implement your solution from scratch.

Starter solutions for C++, Java, and Python3 are shown below.

C++

```cpp
#include <iostream>
#include <vector>

using std::vector;
using std::cin;
using std::cout;
using std::max;

int MaxPairwiseProduct(const vector<int>& numbers) {
  int product = 0;
  int n = numbers.size();
  for (int i = 0; i < n; ++i) {
    for (int j = i + 1; j < n; ++j) {
      product = max(product, numbers[i] * numbers[j]);
    }
  }
  return product;
}

int main() {
    int n;
    cin >> n;
    vector<int> numbers(n);
    for (int i = 0; i < n; ++i) {
        cin >> numbers[i];
    }

    int product = MaxPairwiseProduct(numbers);
    cout << product << "\n";
    return 0;
}
```

Java

```java
import java.util.*;
import java.io.*;
```

```java
public class MaxPairwiseProduct {
    static int getMaxPairwiseProduct(int[] numbers) {
        int product = 0;
        int n = numbers.length;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                product = Math.max(product,
                                    numbers[i] * numbers[j]);
            }
        }
        return product;
    }

    public static void main(String[] args) {
        FastScanner scanner = new FastScanner(System.in);
        int n = scanner.nextInt();
        int[] numbers = new int[n];
        for (int i = 0; i < n; i++) {
            numbers[i] = scanner.nextInt();
        }
        System.out.println(getMaxPairwiseProduct(numbers));
    }

    static class FastScanner {
        BufferedReader br;
        StringTokenizer st;

        FastScanner(InputStream stream) {
            try {
                br = new BufferedReader(new
                    InputStreamReader(stream));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        String next() {
            while (st == null || !st.hasMoreTokens()) {
                try {
```

```
                st = new StringTokenizer(br.readLine());
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return st.nextToken();
    }

    int nextInt() {
        return Integer.parseInt(next());
    }
  }

}
```

Python

```
# Uses python3
n = int(input())
a = [int(x) for x in input().split()]

product = 0

for i in range(n):
    for j in range(i + 1, n):
        product = max(product, a[i] * a[j])

print(product)
```

After submitting this solution to the grading system, many students are surprised when they see the following message:

```
Failed case #4/17: time limit exceeded
```

After you submit your program, we test it on dozens of carefully designed test cases to make sure the program is fast and error proof. As the result, we usually know what kind of errors you made. The message above tells that the submitted program exceeds the time limit on the 4th out of 17 test cases.

Stop and Think. Why does the solution not fit into the time limit?

14

MaxPairwiseProductNaive performs of the order of n^2 steps on a sequence of length n. For the maximal possible value $n = 2 \cdot 10^5$, the number of steps is of the order $4 \cdot 10^{10}$. Since many modern computers perform roughly 10^8–10^9 basic operations per second (this depends on a machine, of course), it may take tens of seconds to execute MaxPairwiseProductNaive, exceeding the time limit for the Maximum Pairwise Product Problem.

We need a faster algorithm!

2.2 Fast Algorithm

In search of a faster algorithm, you play with small examples like [5, 6, 2, 7, 4]. Eureka—it suffices to multiply the two largest elements of the array—7 and 6!

Since we need to find the largest and the second largest elements, we need only two scans of the sequence. During the first scan, we find the largest element. During the second scan, we find the largest element among the remaining ones by skipping the element found at the previous scan.

MaxPairwiseProductFast($A[1 \dots n]$):
$index_1 \leftarrow 1$
for i from 2 to n:
  if $A[i] > A[index_1]$:
    $index_1 \leftarrow i$
$index_2 \leftarrow 1$
for i from 2 to n:
  if $A[i] \neq A[index_1]$ and $A[i] > A[index_2]$:
    $index_2 \leftarrow i$
return $A[index_1] \cdot A[index_2]$

2.3 Testing and Debugging

Implement this algorithm and test it using an input $A = [1, 2]$. It will output 2, as expected. Then, check the input $A = [2, 1]$. Surprisingly, it outputs 4. By inspecting the code, you find out that after the first loop, $index_1 = 1$. The algorithm then initializes $index_2$ to 1 and $index_2$ is never

updated by the second for loop. As a result, $index_1 = index_2$ before the return statement. To ensure that this does not happen, you modify the pseudocode as follows:

```
MaxPairwiseProductFast(A[1...n]):
index₁ ← 1
for i from 2 to n:
   if A[i] > A[index₁]:
      index₁ ← i
if index₁ = 1:
   index₂ ← 2
else:
   index₂ ← 1
for i from 1 to n:
   if A[i] ≠ A[index₁] and A[i] > A[index₂]:
      index₂ ← i
return A[index₁] · A[index₂]
```

Check this code on a small datasets $[7, 4, 5, 6]$ to ensure that it produces correct results. Then try an input

```
2
100000  90000
```

You may find out that the program outputs something like $410\,065\,408$ or even a negative number instead of the correct result $9\,000\,000\,000$. If it does, this is most probably caused by an *integer overflow*. For example, in C++ programming language a large number like $9\,000\,000\,000$ does not fit into the standard int type that on most modern machines occupies 4 bytes and ranges from -2^{31} to $2^{31} - 1$, where

$$2^{31} = 2\,147\,483\,648.$$

Hence, instead of using the C++ int type you need to use the int64_t type when computing the product and storing the result. This will prevent integer overflow as the int64_t type occupies 8 bytes and ranges from -2^{63} to $2^{63} - 1$, where

$$2^{63} = 9\,223\,372\,036\,854\,775\,808.$$

You then proceed to testing your program on large data sets, e.g., an array $A[1 \ldots 2 \cdot 10^5]$, where $A[i] = i$ for all $1 \le i \le 2 \cdot 10^5$. There are two ways of doing this.

1. Create this array in your program and pass it to MAXPAIRWISEPRODUCTFAST (instead of reading it from the standard input).

2. Create a separate program, that writes such an array to a file dataset.txt. Then pass this dataset to your program from console as follows:

```
yourprogram < dataset.txt
```

Check that your program processes this dataset within time limit and returns the correct result: 39 999 800 000. You are now confident that the program finally works!

However, after submitting it to the testing system, it fails again...

```
Failed case #5/17: wrong answer
```

But how would you generate a test case that make your program fail and help you to figure out what went wrong?

2.4 Can You Tell Me What Error Have I Made?

You are probably wondering why we did not provide you with the 5th out of 17 test datasets that brought down your program. The reason is that nobody will provide you with the test cases in real life!

Since even experienced programmers often make subtle mistakes solving algorithmic problems, it is important to learn how to catch bugs as early as possible. When the authors of this book started to program, they naively thought that nearly all their programs are correct. By now, we know that our programs are *almost never* correct when we first run them.

When you are confident that your program works, you often test it on just a few test cases, and if the answers look reasonable, you consider your work done. However, this is a recipe for a disaster. To make your program *always* work, you should test it on a set of carefully designed test cases. Learning how to implement algorithms as well as test and debug your programs will be invaluable in your future work as a programmer.

2.5 Stress Testing

We will now introduce *stress testing*—a technique for generating thousands of tests with the goal of finding a test case for which your solution fails.

A stress test consists of four parts:

1. Your implementation of an algorithm.

2. An alternative, trivial and slow, but correct implementation of an algorithm for the same problem.

3. A random test generator.

4. An infinite loop in which a new test is generated and fed into both implementations to compare the results. If their results differ, the test and both answers are output, and the program stops, otherwise the loop repeats.

The idea behind stress testing is that two correct implementations should give the same answer for each test (provided the answer to the problem is unique). If, however, one of the implementations is incorrect, then there exists a test on which their answers differ. The only case when it is not so is when there is the same mistake in both implementations, but that is unlikely (unless the mistake is somewhere in the input/output routines which are common to both solutions). Indeed, if one solution is correct and the other is wrong, then there exists a test case on which they differ. If both are wrong, but the bugs are different, then most likely there exists a test on which two solutions give different results.

Here is the the stress test for MAXPAIRWISEPRODUCTFAST using MAXPAIRWISEPRODUCTNAIVE as a trivial implementation:

```
StressTest(N, M):
while true:
    n ← random integer between 2 and N
    allocate array A[1...n]
    for i from 1 to n:
        A[i] ← random integer between 0 and M
    print(A[1...n])
    result₁ ← MaxPairwiseProductNaive(A)
    result₂ ← MaxPairwiseProductFast(A)
    if result₁ = result₂:
        print("OK")
    else:
        print("Wrong answer: ", result₁, result₂)
        return
```

The `while` loop above starts with generating the length of the input sequence n, a random number between 2 and N. It is at least 2, because the problem statement specifies that $n \geq 2$. The parameter N should be small enough to allow us to explore many tests despite the fact that one of our solutions is slow.

After generating n, we generate an array A with n random numbers from 0 to M and output it so that in the process of the infinite loop we always know what is the current test; this will make it easier to catch an error in the test generation code. We then call two algorithms on A and compare the results. If the results are different, we print them and halt. Otherwise, we continue the while loop.

Let's run StressTest(10, 100 000) and keep our fingers crossed in a hope that it outputs "Wrong answer." We see something like this (the result can be different on your computer because of a different random number generator).

```
...
OK
67232 68874 69499
OK
6132 56210 45236 95361 68380 16906 80495 95298
OK
62180 1856 89047 14251 8362 34171 93584 87362 83341 8784
OK
```

21468 16859 82178 70496 82939 44491
OK
68165 87637 74297 2904 32873 86010 87637 66131 82858 82935
Wrong answer: 7680243769 7537658370

Hurrah! We've found a test case where MaxPairwiseProductNaive and MaxPairwiseProductFast produce different results, so now we can check what went wrong. Then we can debug this solution on this test case, find a bug, fix it, and repeat the stress test again.

Stop and Think. Do you see anything suspicious in the found dataset?

Note that generating tests automatically and running stress test is easy, but debugging is hard. Before diving into debugging, let's try to generate a smaller test case to simplify it. To do that, we change N from 10 to 5 and M from 100 000 to 9.

Stop and Think. Why did we first run StressTest with large parameters N and M and now intend to run it with small N and M?

We then run the stress test again and it produces the following.

...
7 3 6
OK
2 9 3 1 9
Wrong answer: 81 27

The slow MaxPairwiseProductNaive gives the correct answer 81 ($9 \cdot 9 = 81$), but the fast MaxPairwiseProductFast gives an incorrect answer 27.

Stop and Think. How MaxPairwiseProductFast can possibly return 27?

To debug our fast solution, let's check which two numbers it identifies as two largest ones. For this, we add the following line before the return statement of the MaxPairwiseProductFast function:

print($index_1$, $index_2$)

After running the stress test again, we see the following.

...
7 3 6
1 3

```
OK
5
2 9 3 1 9
2 3
Wrong answer: 81 27
```

Note that our solutions worked and then failed on exactly the same test cases as on the previous run of the stress test, because we didn't change anything in the test generator. The numbers it uses to generate tests are pseudorandom rather than random—it means that the sequence looks random, but it is the same each time we run this program. It is a convenient and important property, and you should try to have your programs exhibit such behavior, because deterministic programs (that always give the same result for the same input) are easier to debug than non-deterministic ones.

Now let's examine $index_1 = 2$ and $index_2 = 3$. If we look at the code for determining the second maximum, we will notice a subtle bug. When we implemented a condition on i (such that it is not the same as the previous maximum) instead of comparing i and $index_1$, we compared $A[i]$ with $A[index_1]$. This ensures that the second maximum differs from the first maximum by the value rather than by the index of the element that we select for solving the Maximum Pairwise Product Problem. So, our solution fails on any test case where the largest number is equal to the second largest number. We now change the condition from

$$A[i] \neq A[index_1]$$

to

$$i \neq index_1$$

After running the stress test again, we see a barrage of "OK" messages on the screen. We wait for a minute until we get bored and then decide that MaxPairwiseProductFast is finally correct!

However, you shouldn't stop here, since you have only generated very small tests with $N = 5$ and $M = 10$. We should check whether our program works for larger n and larger elements of the array. So, we change N to $1\,000$ (for larger N, the naive solution will be pretty slow, because its running time is quadratic). We also change M to $200\,000$ and run. We again see the screen filling with words "OK", wait for a minute, and then

decide that (finally!) MaxPairwiseProductFast is correct. Afterwards, we submit the resulting solution to the grading system and pass the Maximum Pairwise Product Problem test!

As you see, even for such a simple problems like Maximum Pairwise Product, it is easy to make subtle mistakes when designing and implementing an algorithm. The pseudocode below presents a more "reliable" way of implementing the algorithm.

MaxPairwiseProductFast($A[1 \dots n]$):
$index \leftarrow 1$
for i from 2 to n:
    if $A[i] > A[index]$:
        $index \leftarrow i$
swap $A[index]$ and $A[n]$

$index \leftarrow 1$
for i from 2 to $n - 1$:
    if $A[i] > A[index]$:
        $index \leftarrow i$
swap $A[index]$ and $A[n - 1]$
return $A[n - 1] \cdot A[n]$

In this book, besides learning how to design and analyze algorithms, you will learn how to implement algorithms in a way that minimizes the chances of making a mistake, and how to test your implementations.

2.6   Even Faster Algorithm

The MaxPairwiseProductFast algorithm finds the largest and the second largest elements in about $2n$ comparisons.

Exercise Break. Find two largest elements in an array in $1.5n$ comparisons.

After solving this problem, try the next, even more challenging Exercise Break.

Exercise Break. Find two largest elements in an array in $n + \lceil \log_2 n \rceil - 2$ comparisons.

And if you feel that the previous Exercise Break was easy, here are the next two challenges that you may face at your next interview!

> **Exercise Break.** Prove that no algorithm for finding two largest elements in an array can do this in less than $n + \lceil \log_2 n \rceil - 2$ comparisons.

> **Exercise Break.** What is the fastest algorithm for finding three largest elements?

2.7 A More Compact Algorithm

The Maximum Pairwise Product Problem can be solved by the following compact algorithm that uses sorting (in non-decreasing order).

```
MaxPairwiseProductBySorting(A[1...n]):
Sort(A)
return A[n - 1] · A[n]
```

This algorithm does more than we actually need: instead of finding two largest elements, it sorts the entire array. For this reason, its running time is $O(n \log n)$, but not $O(n)$. Still, for the given constraints ($2 \leq n \leq 2 \cdot 10^5$) this is usually sufficiently fast to fit into a second and pass our grader.

3 Solving a Programming Challenge in Five Easy Steps

Below we summarize what we've learned in this chapter.

3.1 Reading Problem Statement

Start by reading the problem statement that contains the description of a computational task, time and memory limits, and a few sample tests. Make sure you understand how an output matches an input in each sample case.

If time and memory limits are not specified explicitly in the problem statement, the following default values are used.

Time limits (sec.):

C	C++	Java	Python	C#	Haskell	JavaScript	Kotlin	Ruby	Rust	Scala
1	1	1.5	5	1.5	2	5	1.5	5	1	3

Memory limit: 512 Mb.

3.2   Designing an Algorithm

After designing an algorithm, prove that it is correct and try to estimate its expected running time on the most complex inputs specified in the constraints section. If you laptop performs roughly 10^8–10^9 operations per second, and the maximum size of a dataset in the problem description is $n = 10^5$, then an algorithm with quadratic running time is unlikely to fit into the time limit (since $n^2 = 10^{10}$), while a solution with running time $O(n \log n)$ will. However, an $O(n^2)$ solution will fit if $n = 1\,000$, and if $n = 100$, even an $O(n^3)$ solutions will fit. Although polynomial algorithms remain unknown for some hard problems in this book, a solution with $O(2^n n^2)$ running time will probably fit into the time limit as long as n is smaller than 20.

3.3   Implementing an Algorithm

Start implementing your algorithm in one of the following programming languages supported by our automated grading system: C, C++, C#, Haskell, Java, JavaScript, Kotlin, Python2, Python3, Ruby, or Scala. For all problems, we provide starter solutions for C++, Java, and Python3. For other programming languages, you need to implement a solution from scratch. The grading system detects the programming language of your submission automatically, based on the extension of the submission file.

We have reference solutions in C++, Java, and Python3 (that we don't share with you) which solve the problem correctly under the given constraints, and spend at most 1/3 of the time limit and at most 1/2 of the memory limit. You can also use other languages, and we've estimated the time limit multipliers for them. However, we have no guarantee that a correct solution for a particular problem running under the given time and memory constraints exists in any of those other languages.

In the Appendix, we list compiler versions and flags used by the grading system. We recommend using the same compiler flags when you test your solution locally. This will increase the chances that your program behaves in the same way on your machine and on the testing machine (note that a buggy program may behave differently when compiled by different compilers, or even by the same compiler with different flags).

3.4 Testing and Debugging

Submitting your implementation to the grading system without testing it first is a bad idea! Start with small datasets and make sure that your program produces correct results on all sample datasets. Then proceed to checking how long it takes to process a large dataset. To estimate the running time, it makes sense to implement your algorithm as a function like solve(dataset) and then implement an additional procedure generate() that produces a large dataset. For example, if an input to a problem is a sequence of integers of length $1 \leq n \leq 10^5$, then generate a sequence of length 10^5, pass it to your solve() function, and ensure that the program outputs the result quickly.

Check the boundary values to ensure that your program processes correctly both short sequences (e.g., with 2 elements) and long sequences (e.g., with 10^5 elements). If a sequence of integers from 0 to, let's say, 10^6 is given as an input, check how your program behaves when it is given a sequence $0, 0, \ldots, 0$ or a sequence $10^6, 10^6, \ldots, 10^6$. Afterwards, check it also on randomly generated data. Check degenerate cases like an empty set, three points on a single line, a tree which consists of a single path of nodes, etc.

After it appears that your program works on all these tests, proceed to stress testing. Implement a slow, but simple and correct algorithm and check that two programs produce the same result (note however that this is not applicable to problems where the output is not unique). Generate random test cases as well as biased tests cases such as those with only small numbers or a small range of large numbers, strings containing a single letter "a" or only two different letters (as opposed to strings composed of all possible Latin letters), and so on. Think about other possible tests which could be peculiar in some sense. For example, if you are generating graphs, try generating trees, disconnected graphs, complete graphs, bipartite graphs, etc. If you generate trees, try generating paths, binary trees,

stars, etc. If you are generating integers, try generating both prime and composite numbers.

3.5 Submitting to the Grading System

When you are done with testing, submit your program to the grading system! Go to the submission page, create a new submission, and upload a file with your program (make sure to upload a source file rather than an executable). The grading system then compiles your program and runs it on a set of carefully constructed tests to check that it outputs a correct result for all tests and that it fits into the time and memory limits. The grading usually takes less than a minute, but in rare cases, when the servers are overloaded, it might take longer. Please be patient. You can safely leave the page when your solution is uploaded.

As a result, you get a feedback message from the grading system. You want to see the "Good job!" message indicating that your program passed all the tests. The messages "Wrong answer", "Time limit exceeded", "Memory limit exceeded" notify you that your program failed due to one of these reasons. If you program fails on one of the first two test cases, the grader will report this to you and will show you the test case and the output of your program. This is done to help you to get the input/output format right. In all other cases, the grader will *not* show you the test case where your program fails.

4 Appendix

4.1 Compiler Flags

C (gcc 7.4.0). File extensions: .c. Flags:

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (g++ 7.4.0). File extensions: .cc, .cpp. Flags:

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

If your C/C++ compiler does not recognize -std=c++14 flag, try replacing it with -std=c++0x flag or compiling without this flag at all

(all starter solutions can be compiled without it). On Linux and Ma-cOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

C# (mono `4.6.2`). File extensions: `.cs`. Flags:

```
mcs
```

Go (golang `1.13.4`). File extensions: `.go`. Flags

```
go
```

Haskell (ghc `8.0.2`). File extensions: `.hs`. Flags:

```
ghc -02
```

Java (OpenJDK `1.8.0_232`). File extensions: `.java`. Flags:

```
javac -encoding UTF-8
java -Xmx1024m
```

JavaScript (NodeJS `12.14.0`). File extensions: `.js`. No flags:

```
nodejs
```

Kotlin (Kotlin `1.3.50`). File extensions: `.kt`. Flags:

```
kotlinc
java -Xmx1024m
```

Python (CPython `3.6.9`). File extensions: `.py`. No flags:

```
python3
```

Ruby (Ruby `2.5.1p57`). File extensions: `.rb`.

```
ruby
```

Rust (Rust `1.37.0`). File extensions: `.rs`.

```
rustc
```

Scala (Scala `2.12.10`). File extensions: `.scala`.

```
scalac
```

4.2 Frequently Asked Questions

Why My Submission Is Not Graded?

You need to create a submission and upload the *source file* (rather than the executable file) of your solution. Make sure that after uploading the file with your solution you press the blue "Submit" button at the bottom. After that, the grading starts, and the submission being graded is enclosed in an orange rectangle. After the testing is finished, the rectangle disappears, and the results of the testing of all problems are shown.

What Are the Possible Grading Outcomes?

There are only two outcomes: "pass" or "no pass." To pass, your program must return a correct answer on all the test cases we prepared for you, and do so under the time and memory constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. Your solution fails if it either crashes, returns an incorrect answer, works for too long, or uses too much memory for some test case. The feedback will contain the index of the first test case on which your solution failed and the total number of test cases in the system. The tests for the problem are numbered from 1 to the total number of test cases for the problem, and the program is always tested on all the tests in the order from the first test to the test with the largest number.

Here are the possible outcomes:

- `Good job!` Hurrah! Your solution passed, and you get a point!

- `Wrong answer.` Your solution outputs incorrect answer for some test case. Check that you consider all the cases correctly, avoid integer overflow, output the required white spaces, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement.

- `Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. Check again the running time of your implementation. Test your program locally on the test of maximum size specified in the problem statement and check how long it

works. Check that your program doesn't wait for some input from the user which makes it to wait forever.

- `Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. Estimate the amount of memory that your program is going to use in the worst case and check that it does not exceed the memory limit. Check that your data structures fit into the memory limit. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the tests of maximum size specified in the problem statement and look at its memory consumption in the system.

- `Cannot check answer. Perhaps the output format is wrong.` This happens when you output something different than expected. For example, when you are required to output either "Yes" or "No", but instead output 1 or 0. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (please follow the exact output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.

- `Unknown signal 6 (or 7, or 8, or 11, or some other).` This happens when your program crashes. It can be because of a division by zero, accessing memory outside of the array bounds, using uninitialized variables, overly deep recursion that triggers a stack overflow, sorting with a contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compiler and the same compiler flags as we do.

- `Internal error: exception...` Most probably, you submitted a compiled program instead of a source code.

- `Grading failed.` Something wrong happened with the system. Report this through Coursera or edX Help Center.

May I Post My Solution at the Forum?

Please do not post any solutions at the forum or anywhere on the web, even if a solution does not pass the tests (as in this case you are still revealing parts of a correct solution). Our students follow the Honor Code: "I will not make solutions to homework, quizzes, exams, projects, and other assignments available to anyone else (except to the extent an assignment explicitly permits sharing solutions)."

Do I Learn by Trying to Fix My Solution?

My implementation always fails in the grader, though I already tested and stress tested it a lot. Would not it be better if you gave me a solution to this problem or at least the test cases that you use? I will then be able to fix my code and will learn how to avoid making mistakes. Otherwise, I do not feel that I learn anything from solving this problem. I am just stuck.

First of all, learning from your mistakes is one of the best ways to learn.

The process of trying to invent new test cases that might fail your program is difficult but is often enlightening. Thinking about properties of your program makes you understand what happens inside your program and in the general algorithm you're studying much more.

Also, it is important to be able to find a bug in your implementation without knowing a test case and without having a reference solution, just like in real life. Assume that you designed an application and an annoyed user reports that it crashed. Most probably, the user will not tell you the exact sequence of operations that led to a crash. Moreover, there will be no reference application. Hence, it is important to learn how to find a bug in your implementation yourself, without a magic oracle giving you either a test case that your program fails or a reference solution. We encourage you to use programming assignments in this class as a way of practicing this important skill.

If you have already tested your program on all corner cases you can imagine, constructed a set of manual test cases, applied stress testing, etc, but your program still fails, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that by writing such a post you will realize that you missed some corner cases!), and only afterwards asking other learners to give you more ideas for tests cases.