British Informatics Olympiad

Time allowed: 3 hours

Sponsored by Jane Street

The 2025 British Informatics Olympiad

Instructions

You should write a program for part (a) of each question, and produce written answers to the remaining parts. Programs may be used to help produce the answers to these written questions but are not always necessary.

You may use a calculator and the on-line help that your programming language provides. You should have a pen, some blank paper, and an empty USB stick (or other storage device) on which to save your programs. You must not use any other material such as files on a computer network, code written outside of the contest, AI generated code, books or other written information. You may not communicate with anyone, other than the person invigilating this paper.

Mark the first page of your written answers with your name, age in years and school/college. Number all pages in order if you use more than one sheet. All of your computer programs should display your name and school/college when they are run, and the storage device you use to submit the programs should also show your name and school/college.

For your programs to be marked, the source code must be saved, along with executables if your language includes a compiler; this includes programs used to help answer written questions. You must clearly indicate the name given to each program on your answer sheet(s).

Sample runs are given for parts 1(a), 2(a) and 3(a). **`Bold text`** indicates output from the program, and `normal text` shows data that has been entered. Where multiple items of input appear on the same line they should be separated by a single space. The output format of your programs should follow the 'sample run' examples. Your programs should take less than *1 second* of processing time for each test.

Attempt as many questions as you can. Do not worry if you are unable to finish this paper in the time available. Marks allocated to each part of a question are shown in square brackets next to the questions. Partial solutions (such as programs that only get some of the test cases correct within the time limit, or partly completed written answers) may get partial marks.

Questions can be answered in any order, and you may answer the written questions without attempting the programming parts.

Hints

- If you can only see how to solve part of a problem it is worth writing a program that solves that part. We want to give you marks and questions are evaluated using multiple tests of differing difficulty. ***Remember, partial solutions may get partial marks.***
- Question 2 is an implementation challenge and question 3 is a problem solving challenge.
- Some written questions can be solved by hand without solving the programming parts.
- The final written part of each question is intended to be a difficult challenge.
- Do not forget to indicate the name given to your programs on your answer sheet(s).

Question 1: Palindromic Sums

Every positive integer can be represented by a *palindromic sum* of at most three *palindromic* numbers (numbers that remain the same when their digits are reversed).

To be a valid palindromic sum for an integer, the sum *must* contain the *smallest* possible number of palindromic numbers. It can contain duplicates.

For example:

- 12321 is a palindromic number already, so can be formed from just 12321;
- 9610 is equal to 161 + 9449, which is the only pair of palindromic numbers that sum to 9610. There are triplets that sum to 9610, such as 282 + 1771 + 7557, but those contain too many palindromic numbers;
- 1031 requires three palindromic numbers such as 2 + 494 + 535 and 4 + 88 + 939.

1(a) [25 marks]

Write a program that reads in an integer (between 1 and 1,000,000 inclusive) and outputs 1, 2 or 3 palindromic numbers which together form a minimal length *palindromic sum* for the input.

Sample run

```
1031
1 101 929
```

If there are several possible palindromic sums you should output the one containing the lowest palindromic number. If there are still several solutions remaining you should select the one containing the highest palindromic number.

1(b) [2 marks]

List the palindromic sums that represent 54.

1(c) [5 marks]

How many integers (between 1 and 1,000,000 inclusive) require a palindromic sum containing three numbers?

Question 2: Safe Haven

Red and Green are playing a game on an square grid, trying to create as many *safe havens* of their own colour as possible. Each square on the grid is either *empty* or is controlled by *Red* or *Green*. Squares are neighbours if they touch on an edge; i.e. immediately horizontally or vertically adjacent. A *haven* is a group of non-empty squares where it is possible to go between any two squares in the haven by a sequence of neighbours in the haven, and no other non-empty square in the game neighbours a square in the haven. A *safe haven* is a haven that only contains squares of a single colour.

The top row squares on the grid have positions 1 to n (from left to right), the next row positions n+1 to 2n (left to right), etc. so that the bottom right square is position n^2.

Before the game begins, the grid is set up by marking all squares as empty and then having players alternate taking control of empty squares, until all are controlled. Red starts by taking control of 1 and then repeatedly for their turn, starting with Green:

- A player will visit successive positions on the grid. If they reach n^2 they will next visit 1.
- They start at the position immediately after the most recently controlled square.
- A player keeps track of the number of times they visit an empty square. When this reaches a fixed modifier (r for Red and g for Green) they take control of the square and their turn ends.

For example, suppose they are playing on a 3×3 grid and that both r and g are 5.

- Red controls 1;
- Green visits empty squares at 2, 3, 4, 5, 6 and takes control of 6;
- Red visits empty squares at 7, 8, 9, then 1 (which is non-empty), then empty squares 2, 3 and takes control of 3;
- Green controls 9. Red controls 8;
- Green will visit 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2 and controls 2. As there were only 4 empty squares at the start of their turn, it was necessary to visit one of the empty squares more than once;
- When the set up is complete Red controls 1, 3, 4, 5 and 8. Green controls the other squares.

The game now begins. A move involves two neighbouring squares which are controlled by different players. The current player will transfer their control to the neighbouring square; i.e. their currently controlled square will become empty and the neighbouring square switches to their control.

On their turn players will use the following strategy to determine their move:

- The player selects the non-safe haven containing the smallest number of squares controlled by their opponent. In the event of a tie, they select the haven containing the largest number of squares that they control. If there is still a tie, they select the one containing the square with highest value position.
- The player selects the lowest value position in this haven that they control and which neighbours a square controlled by their opponent.
- The player's move is a transfer involving this square and the lowest value neighbouring square controlled by their opponent.

When resolving a tie, only the tied havens are considered. Other havens on the grid are ignored.

For example, suppose there are havens with 1 Red and 0 Green, 2 Red and 3 Green, 4 Red and 3 Green and 9 Red and 4 Green. It is Red's turn:

- They will not select the haven with 1 Red and 0 Green, as this is a safe haven;
- The smallest number of Green squares in a haven is 3 but there is tie;
- The largest number of Red squares in one of the havens with 3 Green squares is 4. This haven will be selected.

Starting with Red, turns alternate until there are no more valid moves which can be taken. At the end of the game all controlled squares will be in safe havens and each player counts the number of safe havens that they control.

2(a) [25 marks]

Write a program that plays *haven*.

Your program should input a line containing three integers, n ($1 \leq n \leq 10$) then r ($1 \leq r \leq 5000$) then g ($1 \leq g \leq 5000$), indicating the size of the grid and the modifiers for Red and Green respectively.

You should output two integers, the number of safe havens controlled by Red followed by the number controlled by Green at the end of the game.

Sample run

```
3 5 5
2 1
```

2(b) [2 marks]

What is the grid layout at the end of set up if the input was `3 123456789 987654321`?

2(c) [3 marks]

Find modifiers for Red and Green, both less than 50, which will set up a 4×4 grid such that there are no neighbouring squares controlled by the same player.

2(d) [5 marks]

The strategy is updated to include the following step at the start of the turn:

If there are moves which make at least one safe haven for the current player, the player will play the one with the lowest value position for their controlled square, which takes control of the lowest value neighbouring square controlled by their opponent. If there is no such move, the existing strategy is applied.

How many safe havens does each player control, using this strategy, with the input `10 810 2025`?

Question 3: Short Fuse

A *fuse* is a length of material that can be burned. This is an irregular process but when lit from one end a fuse completes burning in a fixed amount of time. If lit from both ends it will take half this time. A fuse can only be used once and when it has started to burn it cannot be paused.

Given several fuses we can measure different *continuous* periods of time by lighting fuses when other fuses have finished burning. Fuses are lit at the instant another fuse stops burning and multiple fuses can be lit simultaneously. We cannot measure time in any other way.

For example, given two fuses with a burn time of 1, we can measure various times as follows:

- 0 — do not light any fuse;
- 1 — light a single fuse at one end;
- 2 — light a fuse at one end. When it finishes burning (at 1) light the other fuse at one end;
- 0.5 — light both ends of a fuse simultaneously;
- 1.5 — light both ends of a fuse simultaneously. When it finishes burning, light the other fuse at one end;
- 0.75 — light both ends of the first fuse and one end of the second fuse simultaneously. At 0.5 the first fuse has finished and the second has another 0.5 to burn. At this point light the second end of the second fuse, so it completes burning in 0.25;
- 0.25 — follow the directions for 0.75 but only start timing the period after the first fuse has finished.

3(a) [25 marks]

Write a program to determine the number of distinct periods that can be measured by a set of fuses.

Your program should first input a single integer, f ($1 \leq f \leq 4$), indicating the number of fuses. You should then input f integers, each between 1 and 5000 inclusive, giving the burn times for each fuse.

You should output the number of possible periods that we can time with these fuses.

Sample run

```
2
1 1
7
```

3(b) [2 marks]

What periods can be measured with two fuses with burn times 1 and 2?

3(c) [6 marks]

What is the maximum number of periods that can be measured with two fuses? How about three fuses?

Total Marks: 100

End of BIO 2025 Round One paper