BRITISH INFORMATICS OLYMPIAD 2006–2025 ROUND ONE QUESTIONS

- This document includes all BIO round one questions sorted by question number and series
- The mark schemes are provided at the end of the paper and are sorted by series

Question 1: ***Anagrams***

Two words are anagrams of each other if they can both be formed by rearranging the same combination of letters. For example, GADGET and TAGGED are anagrams because they both contain one occurrence of each of the letters A, D, E and T, and two occurrences of the letter G.

1(a) [24 marks]

Write a program which inputs two words (each of which will contain fewer than 10 uppercase letters) and then prints **`Anagrams`** if they are anagrams of each other, or prints **`Not anagrams`** otherwise. Your program should then terminate.

Sample run

```
SUMMER
RESUME
Not anagrams
```

1(b) [2 marks]

Suppose that any combination of the letters A, B and C makes a valid word. How many anagrams are there of the word ABC (including the word ABC itself)?

1(c) [4 marks]

Suppose that the only valid letters are A, B and C, and that any combination of these letters makes a valid word. A list is created that contains words that are 5 letters long; no two words on the list are anagrams of each other. What is the maximum number of words on list?

Question 1: ***Cards***

A card game is played with a deck of forty cards, containing each of the numbers from 1 to 10 exactly four times. The game is scored according to the following two rules: a point is given for each pair of cards with identical numerical values and for any group of cards whose numerical values sum to 15.

For example, the set of cards *8, 8 and 8* is worth three points since there are three different pairs of cards with identical numerical values. The set *10, 5, 2 and 3* is worth two points since there are two groups of cards whose numerical values sum to 15.

1(a) [24 marks]

Write a program which inputs 5 numbers (each of which will between 1 and 10 inclusive) indicating the numerical values on 5 different cards. Your program should print out the number of points these 5 cards are worth and then terminate.

Sample run

```
3 3 3 2 10
6
```

For the following parts remember that you are allowed to use each number from 1 to 10 at most *four* ***times.***

1(b) [2 marks]

Give a set of five cards that score 0 points.

1(c) [4 marks]

How many different sets of five cards are there where the sum of the values of all five cards is exactly 15? [You should assume that order does not matter and cards are only different if they contain different values; e.g. *1 2 3 4 5* is the same as *5 4 3 2 1*.]

Question 1: *Goldbach Conjecture*

A *prime number* is a whole number, greater than *1*, that can only be divided by itself and the number *1*. It is known that all even numbers between *4* and *300,000,000,000,000,000* are equal to the sum of two primes (a fact that is believed to be true for all larger even numbers as well, and called the *Goldbach Conjecture*).

For example, *30 = 7 + 23*. There are two other ways of expressing *30* as the sum of two primes, which are *11 + 19* and *13 + 17*. These are the only ways of expressing *30* as the sum of two primes, since the order of the numbers in the additions does not matter.

1(a) [25 marks]

Write a program which inputs a single *even* number (between *4* and *10,000* inclusive) and outputs a single number which is the number of different ways the input can be expressed as the sum of two primes.

Sample run

```
22
3
```

1(b) [3 marks]

There are four ways of expressing *46* as the sum of two primes. What are they?

1(c) [2 marks]

There are many odd numbers which cannot be expressed as the sum of two primes. How many such numbers are there between *4* and *50*?

Question 1: *Digit Words*

A *digit word* is a word where, after possibly removing some letters, you are left with one of the single digits: ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT or NINE.

For example:
- ~~B~~O~~U~~N~~C~~E and ~~ANN~~O~~U~~N~~C~~E are digit words, since they contain the digit ONE.
- ENCODE is not a digit word, even though it contains an O, N and E, since they are not in order.

1(a) [24 marks]

Write a program which reads in a single upper-case word (with at most fifteen letters) and determines if it is a *digit word*.

If the word is not a digit word you should output the word **NO**. If the word is a digit word you should output the digit it contains, as a number.

You will not be given any words which contain more than one digit.

Sample run 1

```
BOUNCE
1
```

Sample run 2

```
ENCODE
NO
```

1(b) [2 marks]

In how many different ways can letters be removed from TWOTWOTWO to leave the digit TWO?

1(c) [4 marks]

The made-up digit word TWFOUR contains the digit TWO and the digit FOUR. What is the length of the shortest made-up word which contains all the digits ONE to FIVE? How about ONE to NINE?

Question 1: ***Anagram Numbers***

An *anagram number* is a number that can be multiplied by at least one single digit number (**other than 1**) to become an anagram of itself. Any anagrams formed by multiplying an anagram number by a digit are said to be *generated* by that anagram number. Two numbers are anagrams of each other if they can both be formed by rearranging the same combination of digits.

For example:
- 1246878 is an anagram number; multiplying by 6 generates 7481268 or by 7 generates 8728146. These numbers all contain a single 1, 2, 4, 6, 7 and two 8s.
- 1246879 is not an anagram number.

1(a) [25 marks]

Write a program which reads in a single number (between 1 and 123456789 inclusive) and determines if it is an *anagram number*.

If the number is not an anagram number you should output the word **NO**. If it is an anagram number you should output each single digit it can be multiplied by to make an anagram of itself.

Sample run 1

```
123456789
2 4 5 7 8
```

Sample run 2

```
100
NO
```

1(b) [2 marks]

85247910 is generated by which anagram numbers?

1(c) [3 marks]

How many anagram numbers between 100,000 and 999,999 contain no duplicated digits?

Question 1: *Fibonacci Letters*

Each letter in the alphabet can be given a value based on its position the alphabet, A being 1 through to Z being 26. Two letters can produce a third letter by adding together their values, deducting 26 from the sum if it is greater than 26, and finding the letter whose value equals this result.

For example:

- A and B produce the letter C since 1+2=3 and A, B and C are respectively letters 1, 2 and 3 in the alphabet.
- P and Q produce the letter G since 16+17=33, 33-26=7 and P, Q and G are respectively letters 16, 17 and 7 in the alphabet.

We can generate a sequence of letters by starting with two letters and repeatedly using the last two letters in the sequence to produce another letter.

For example (starting with A and A) we have: A, A, B, C, E, H, M, U, H, C, …

1(a) [24 marks]

Write a program which reads in two capital letters (the 1st letter in a sequence followed by the 2nd letter) then an integer n ($1 \leq n \leq 1{,}000{,}000$). You should output a single capital letter, the n^{th} letter in the sequence that starts with the input letters.

Sample run 1

```
A A 7
M
```

1(b) [3 marks]

What letter together with F produces X? What letter together with Q produces H?

1(c) [3 marks]

Consider the sequence whose first two letters are both C. What is the 1,000,000,000,000,000,000th letter?

Question 1: *Distinct Prime Factorisation*

A *prime number* is a whole number, greater than 1, that can only be divided by itself and the number 1. Every integer greater than 1 can be uniquely expressed as the product of prime numbers (ignoring re-ordering those numbers). This is called the *prime factorisation* of the number.

For example:
- 100 = 2 × 2 × 5 × 5
- 101 = 101 (since 101 is a prime number)

In this question we are interested in the product of the distinct prime factors of a given number; in other words each number in the prime factorisation is to be used only once.

For example:
- Since 100 = 2 × 2 × 5 × 5 the product we require is 10 (i.e. 2 × 5)

1(a) [24 marks]

Write a program which reads in a single integer n ($1 < n < 1{,}000{,}000$) and outputs a single integer, the product of the distinct prime factors of n.

Sample run 1

```
100
10
```

Sample run 2

```
101
101
```

1(b) [2 marks]

Which are the 10 lowest numbers having 10 as the product of their distinct prime factors?

1(c) [4 marks]

Which product of distinct prime factors, for n between 1 and 1,000,000, occurs the most frequently?

Question 1: *Watching the Clock*

Two *clocks*, which show the time in hours and minutes using the 24 hour clock, are running at different speeds. Each clock is an exact number of minutes per hour fast. Both clocks start showing the same time (00:00) and are checked regularly every hour (starting after one hour) according to an accurate timekeeper. What time will the two clocks show on the first occasion when they are checked and show the same time?

NB: For this question we *only* care about the clocks matching when they are checked.

For example, suppose the first clock runs 1 minute fast (per hour) and the second clock runs 31 minutes fast (per hour).

- When the clocks are first checked after one hour, the first clock will show 01:01 and the second clock will show 01:31;
- When the clocks are checked after two hours, they will show 02:02 and 03:02;
- After 48 hours the clocks will both show 00:48.

1(a) [25 marks]

Write a program which reads in a two integers, each between 0 and 50,000 inclusive, indicating the number of minutes fast (per hour) of the first and second clock respectively.

You should output the time shown on the clocks when they first match. Both the hour and the minutes should be displayed with two digits.

Sample run 1

```
1 31
00:48
```

1(b) [3 marks]

Suppose the first clock is accurate and the clocks do *not* show 00:00 when they first match. The second clock is less than 20 minutes fast (per hour). How many minutes fast is the second clock? Write out all the possible answers.

1(c) [4 marks]

Suppose the two clocks can be *any* number of full minutes fast per hour. What is the largest number of hours that can pass before the clocks first match?

Question 1: *Lucky Numbers*

The *lucky numbers* are produced by taking a list of all the odd numbers and systematically removing some of the numbers. Our first lucky number is 1. We now repeatedly take the next highest number *n* that is still in the list, mark it as a lucky number and then remove every *n*th number from the list.

For example, with lucky numbers underlined as we progress:

- Initially we have: $\underline{1}$, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, ...
- The next lucky number is 3, giving us: $\underline{1}$, $\underline{3}$, 7, 9, 13, 15, 19, 21, 25, 27, 31, ...
- Next is 7: $\underline{1}$, $\underline{3}$, $\underline{7}$, 9, 13, 15, 21, 25, 27, 31, ...
- Next is 9: $\underline{1}$, $\underline{3}$, $\underline{7}$, $\underline{9}$, 13, 15, 21, 25, 31, ...

1(a) [25 marks]

Write a program which reads in a single integer between 2 and 10,000 inclusive.

You should output two numbers. Firstly the closest lucky number that is *less* than the input, followed by the closest lucky number that is *greater* than the input.

Sample run

```
5
3 7
```

1(b) [2 marks]

How many numbers less than 100 are lucky?

1(c) [3 marks]

Suppose that you are told the 1,000,000,000th lucky number is *X*. A program that *correctly* solves 1(a) (on inputs of any size >2) is run on all numbers from 2 to *X* inclusive. How many *different* results will the program produce?

(NB: A result is a pair of numbers produced by the program; e.g. **3 7** is a single result.)

Question 1: *Block Palindromes*

A *palindrome* is a word that shows the same sequence of letters when reversed. If a word can have its letters grouped together in two or more blocks (each containing one or more adjacent letters) then it is a *block palindrome* if reversing the order of those blocks results in the same sequence of blocks.

For example, using brackets to indicate blocks, the following are *block palindromes*:

- BONBON can be grouped together as (BON)(BON);
- ONION can be grouped together as (ON)(I)(ON);
- BBACBB can be grouped together as (B)(BACB)(B) or (BB)(AC)(BB) or (B)(B)(AC)(B)(B)
  Note that (BB)(AC)(B)(B) is *not* valid as the reverse (B)(B)(AC)(BB) shows the blocks in a different order.

1(a) [23 marks]

Write a program which reads in a word of between 2 and 10 (inclusive) uppercase letters.

You should output a single number, the number of different ways the input can be grouped to show it is a *block palindrome*.

Sample run

```
BBACBB
3
```

1(b) [2 marks]

Give all the groupings of AABCBAA that show it is a *block palindrome*.

1(c) [6 marks]

Suppose that all the groupings of a *block palindrome* contain an even number of blocks. What can you say about the length of the *block palindrome?* How many different groupings can it have? Justify both your answers.

Question 1: ***Promenade Fractions***

A *promenade* is a way of uniquely representing a fraction by a succession of "left or right" choices. As successive choices are made the value of the promenade changes by combining the values of the promenade before the most recent left choice with the value before the most recent right choice.

If the value before the most recent left choice was *l/m* and the value before the most recent right choice was *r/s* then the new value will be (*l*+*r*) / (*m*+*s*). If there has never been a left choice we use *l*=1 and *m*=0; if there has never been a right choice we use *r*=0 and *s*=1.

Fractions are **always** used in their lowest form; recall that *a/b* is in its lowest form if it is not possible to divide *a* and *b* by a common factor. Values generated by the formula will automatically be in their lowest form. Fractions are allowed to have *a* larger than *b*.

We will write our promenades as a sequence of Ls (for left choices) and Rs (for right choices).

For example, to form the promenade LRLL (using ∅ to represent the promenade before any choices are made):
- The value of ∅ is (1+0)/(0+1) = 1/1;
- The value of L is 1/2. Before the most recent left choice we bhad ∅ (= 1/1). There has not yet been a right choice, so we use *r*=0 and *s*=1. So the value of L is (1+0)/(1+1) = 1/2;
- LR = 2/3 as we use the values of ∅ (before the left choice) and L (before the right choice);
- LRL = 3/5 as we use the values of LR and L;
- LRLL = 4/7 as we use the values of LRL and L.

1(a) [23 marks]

Write a program which reads in a promenade of between 1 and 10 (inclusive) uppercase letters (each `L` or `R`).

You should output the promenade's value as a fraction in its lowest form.

Sample run 1

```
LRLL
4 / 7
```

Sample run 2

```
RL
3 / 2
```

1(b) [2 marks]

What is LRL + LLLL as a promenade?

1(c) [3 marks]

How many Ls and how many Rs does the promenade representing 1 / 1,000,000 contain?

1(d) [3 marks]

Does any promenade represent a negative fraction? Justify your answer.

Question 1: *Coloured Triangles*

A *coloured triangle* is created from a row of squares, each of which is *red*, *green* or *blue*. Successive rows, each containing one fewer square than the last, are generated by considering the two touching squares in the previous row. If these squares are identical, the same colour is used in the new row. If they are different, the missing colour is used in the new row. This is continued until the final row, with only a single square, is generated.

In the following three example triangles `R`, `G` and `B` have been used to represent the colours. Note that each row is generated from the row above.

```
G G          R B          R R G B R G B B
 G            G            R B R G B R B
                            G G B R G G
                             G R G B G
                              B B R R
                               B G R
                                R B
                                 G
```

1(a) [23 marks]

Write a program which reads in a starting row of between 1 and 10 (inclusive) uppercase letters (each `R`, `G` or `B`).

You should output the colour (`R`, `G` or `B`) of the square in the final row of the triangle.

Sample run 1

```
RG
B
```

Sample run 2

```
RBRGBRB
G
```

1(b) [3 marks]

How many possible rows of nine squares are there that generate `RRGBRGBB`? List them.

1(c) [4 marks]

Suppose you have a triangle where, on each row, only the colour of a single square is known. How many ways can the unknown squares be coloured so that the triangle is consistent with the rules for generating a *coloured triangle*? Justify your answer.

1(d) [3 marks]

If the first row contains 4 squares, the colour of the square in the final row only depends on the extreme left and right of the first row. This is not true if, for example, the first row contains 5 squares. Find a larger size of row which shares this unusual property.

Question 1: *Debt Repayment*

At the end of every month interest is added to a *debt* (initially 100) and then repayments are taken off. The *interest* is a fixed percentage of the current debt. The *repayment* is also a fixed percentage of the debt (after the interest has been added) or 50, whichever is larger. If the repayment is greater than the debt it is reduced to match the debt. The cycle of interest and repayment is continued until the debt has been reduced to 0 and paid off.

For example, suppose the interest is 10% and repayments are 50%:

- At the end of the first month the interest is 10, increasing the debt to 110;
- The repayment is 55 (50% of 110) leaving the debt at 55;
- At the end of the second month the interest is 5.5, increasing the debt to 60.5;
- The repayment is 50 (as 50% of 60.5 is less than the minimum amount), leaving the debt at 10.5;
- At the end of the third month the interest is 1.05, increasing the debt to 11.55;
- The repayment is 11.55 (the minimum payment of 50 is reduced to match the debt) and the debt is paid off.
- The total amount repaid on the debt is 116.55

When calculating percentages the amount is rounded *up* to 2 decimal places. E.g. 10.201 and 10.207 would both be rounded up to 10.21.

1(a) [26 marks]

Write a program that reads in the interest percentage followed by the repayment percentage. Percentages will be integers between 0 and 100 (inclusive).

You should output the total amount repaid.

You will always be given input that allows the debt to be paid off.

Sample run 1

```
10 50
116.55
```

1(b) [2 marks]

Given an interest percentage of 43% and a repayment percentage of 46%, how many payments will be made to pay off the debt?

1(c) [3 marks]

Which interest and repayment percentages (integers between 0 and 100) lead to the largest amount repaid on a debt that is paid off?

Question 1: *Palindromic Numbers*

A *palindromic number* is one that is the same when its digits are reversed.

For example:
- 98789 is a palindrome;
- 12344321 is a palindrome;
- 12345 is not a palindrome as it becomes 54321 when reversed.

Except for 0, a palindromic number's leftmost digit must be non-zero.

1(a) [25 marks]

Write a program that reads in a positive integer of up to 20 digits.

You should output the smallest palindromic number that is higher than the input.

Sample run 1

```
17
22
```

Sample run 2

```
343
353
```

1(b) [2 marks]

What is the largest difference between a palindromic number (of up to 20 digits) and the next highest palindromic number?

1(c) [4 marks]

How many integers are there, between 1 and 99999 inclusive, that are *not* the sum of two palindromic numbers?

Question 1: *Roman Look-and-Say*

The *Roman look-and-say* description of a string (of `I`s, `V`s, `X`s, `L`s, `C`s, `D`s and `M`s) is made by taking each block of adjacent identical letters and replacing it with the number of occurrences of that letter, given in *Roman numerals* **(*)***,* followed by the letter itself. A block of adjacent identical letters is never broken into smaller pieces before describing it.

For example:

- `MMXX` is described as "two `M`s followed by two `X`s". Since two is `II` in Roman numerals, this is written as `IIMIIX`;
- `IIMIIX` is described as `IIIIMIIIIX`, which is "two `I`s, one `M`, two `I`s, one `X`";
- `IIIIMIIIIX` is described as `IVIIMIVIIX;`
- It is *not* valid to describe `III` as, "two `I`s, one `I`" `IIIII`.

Note that Roman look-and-say descriptions are *not* necessarily Roman numerals.

1(a) [25 marks]

Write a program that reads in a Roman numeral representing a number between 1 and 3999 inclusive, followed by *n* (1 ≤ *n* ≤ 50).

You should apply the *Roman look-and-say* description *n* times and then output the number of `I`s in the final description, followed by the number of `V`s.

Sample run 1

```
MMXX 1
4 0
```

Sample run 2

```
MMXX 3
6 2
```

1(b) [2 marks]

How many Roman numerals (from 1 to 3999 inclusive) have a Roman look-and-say description that is also a Roman numeral? List these *descriptions*.

1(c) [4 marks]

The Roman look-and-say descriptions are generated for all the Roman numerals from 1 to 3999 (inclusive). How many distinct descriptions are there?

(*) *Roman numerals* are conventionally defined to represent numbers using seven letters: `I`=1, `V`=5, `X`=10, `L`=50, `C`=100, `D`=500 and `M`=1000. Numbers other than these are formed by placing letters together, from left to right, in descending order of size, and adding their values. The basic rule is to always use the biggest numeral possible (e.g. 15 is represented as `XV` but never as `VVV, VX` or `XIIIII`).

Letters may not appear more than three times in a row, so there are six exceptions to these rules – the combinations `IV`, `IX`, `XL`, `XC`, `CD` and `CM`. In these cases a letter is placed before one of greater value and the smaller value is subtracted from the larger. E.g. `CD` = 400. These are the *only* exceptions so, for example, `MIM` is not valid.

Question 1: *Down Pat*

A *pat* is a single letter or a string of letters which can be split into a left and right string (of at least 1 letter) where: each is the reverse of a *pat*; and all the letters in the left string are later in the alphabet than all the letters in the right string.

For example:

- `BA` is a pat as it splits into `B` and `A`, both of which are single letters and therefore pats, and `B` is alphabetically after `A`. `AB` is not a pat as the alphabetical rule would be broken;
- Similarly `ED` is a pat but `DE` is not;
- `DEC` is a pat as it splits into `DE` (whose reverse `ED` is a pat) and `C`.
- `CEDAB` splits into `CED` and `AB`, whose reverses are pats and `C`, `E`, and `D` are after `A` and `B` alphabetically.

1(a) [24 marks]

Write a program that reads in two strings from a line, s_1 then s_2, each between 1 and 6 *uppercase* letters inclusive.

You should output three lines, each containing a `YES` or `NO` indicating, in order, if s_1 is a pat, if s_2 is a pat, and if s_1s_2 (the combination of the two words) is a pat.

You must get all three lines of output correct to score marks.

Sample run

```
DE C
NO
YES
YES
```

1(b) [3 marks]

Which permutations of `ABCD` are pats?

1(c) [5 marks]

How many permutations of the alphabet, beginning with the letter `B`, are pats?

Question 1: *Decrypt*

Each letter in the alphabet can be given a value based on its position in the alphabet, A being 1 through to Z being 26. Two letters can produce a third letter by adding together their values, deducting 26 from the sum if it is greater than 26, and finding the letter whose value equals this result.

A simple *encryption* scheme for a string is as follows. Working from left to right, after leaving the left-most character alone, each character is replaced with the character produced by adding it to the character to its immediate left.

For example:

- The string ENCRYPT goes through the steps ENCRYPT → ESCRYPT → ESVRYPT → ESVNYPT → ESVNMPT -> ESVNMCT -> ESVNMCW

1(a) [24 marks]

Write a program that reads in an *encrypted* string, between 1 and 10 *uppercase* letters inclusive, and outputs the *original* version of that string.

Sample run

```
ESVNMCW
ENCRYPT
```

1(b) [2 marks]

Give a five letter string whose encrypted version matches the original.

1(c) [2 marks]

How many times (≥1) must you encrypt OLYMPIAD before it becomes the original string again?

1(d) [4 marks]

How many three letter strings, if encrypted 999,999,999,999 times, become the original string again?

Question 1: *Zeckendorf Representation*

In the *Fibonacci sequence* each number is generated by adding the previous two numbers in the sequence. We will start with the numbers 1 and 2, so the sequence is 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

The *Zeckendorf representation* of the number *n* consists of the distinct numbers from the Fibonacci sequence which sum to *n*, where no two adjacent numbers from the Fibonacci sequence are used. There is always a unique representation.

For example:

- 21 is represented by the single number 21;
- 21 is not represented by 8 13, even though they sum to 21, as those numbers are adjacent in the Fibonacci sequence;
- 100 is represented by 3 8 89.

The Zeckendorf representation for *n* always includes the largest number from the Fibonacci sequence that is no greater than *n*.

1(a) [24 marks]

Write a program that reads in a number (between 1 and 1,000,000 inclusive) and outputs the numbers (in any order) in the corresponding Zeckendorf representation.

Sample run

```
100
89 8 3
```

1(b) [2 marks]

What is the highest number under 1,000,000 whose Zeckendorf representation consists of a single number?

1(c) [2 marks]

How many numbers under 53,316,291,173 have a Zeckendorf representation consisting of three numbers?

1(d) [4 marks]

How many numbers between 1,000,000,000 and 5,000,000,000 (inclusive) do *not* have 701,408,733 in their Zeckendorf representation?

Question 1: Integer Strings

A string is generated by joining together, in order, all the integers starting with *n*. We are interested in which *digit* appears in the i^{th} position in this string.

For example:

- If we start at 6 the string will begin 678910111213141516…;
- The 11th digit in this string is 1 and the 12th is 3;
- If we start at 999 the string will begin 999100010011002… and the 10th digit is 0.

1(a) [25 marks]

Write a program that reads in integers *n* then *i* (both between 1 and 2^{59} inclusive), indicating the first number that appears in the string and the required digit from the string respectively. You should output the *digit* in the i^{th} position in the string.

Sample run

```
999 11
1
```

1(b) [2 marks]

Consider the string generated when *n = 1*. How many occurrences of the digit 5 appear in the first 101 digits?

1(c) [5 marks]

Consider the string generated when *n = 1*. The substring 123456789 first appears in this string between positions 1 and 9 inclusive.

Where does the substring 11111 first appear? Where does the substring 987654321 first appear?

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: *Rules*

Some computer programs try to prevent easily guessable passwords being used, by rejecting those that do not meet some simple rule. Your task for this question is to determine if a password is accepted by a rule.

The only characters that can occur in a password are the digits 0 to 9.

A rule is made up from a sequence of symbols, each with a special meaning. Symbols can be combined to make more complicated rules, and are written in order. Rules are described as follows:

- The symbol `x` means any digit is acceptable.

  For example: The rule `x` means the password has to be a single digit. The rule `xxx` means the password can be any combination of three digits.

- The symbol `u` means any digit that is higher than the previous digit.
- The symbol `d` means any digit that is lower than the previous digit.

  For example: The rule `xu` means the password has exactly two digits, the second of which is higher than the first; 46 would be accepted, 64 or 66 would be rejected. The rule `xud` means the password has exactly three digits, the second of which is higher than the first, and the third of which is lower than the second; 040 would be accepted.

Combinations of the `x`, `u` and `d` symbols (and only these symbols) can be placed inside brackets without changing their meaning; brackets must contain some symbols. The following two symbols can only follow an `x`, `u`, `d` or bracketed expression. If it is an `x`, `u` or `d` they change the meaning of the previous symbol. If they follow a bracketed expression, they change the meaning of everything inside the brackets.

- The symbol `?` means ignore or apply once.

  For example: The rule `xxx?` means the password must match either the rule `xx` or the rule `xxx`. The rule `x(xx)?` means the password must match either the rule `x` or the rule `xxx`.

- The symbol `*` means ignore, or repeat any number of times.

  For example: The rule `x(xu)*` means the password must match one of the rules `x`, `xxu`, `xxuxu`, `xxuxuxu`, etc...

Some combinations of these symbols are invalid, such as those that break any of the above conditions or, if for some interpretation of their `*` and `?` symbols, are equivalent to an invalid combination. For example, the rule `u` is invalid since there is no previous digit. The rule `x?u` is invalid, since it means match either the rule `xu` or the rule `u`, and the rule `u` is invalid. The rule `(x(ud)*)*` is invalid because the only valid symbols within a pair of brackets are `x`, `u` and `d`.

You will not be given any invalid rules.

2(a) [24 marks]

Write a program to test passwords.

Your program should first read in a single line, containing the rule; this rule will have between 1 and 12 symbols (inclusive, and including any brackets). You should then read in two more lines, each of which will contain a single password; these passwords will contain between 1 and 12 digits (inclusive).

No rule will contain more than two `?` or `*` symbols (combined).

You should output two lines, the first line indicating whether the first password is accepted by the rule, and the second line indicating whether the second password is accepted by the rule. Your should output **`Yes`** if a password is accepted and **`No`** if the password is rejected.

Sample run

```
xu*
02468
4688

Yes
No
```

2(b) [3 marks]

Consider the rule `x(xd)?(xx)?`. How many different passwords will be accepted this rule?

2(c) [3 marks]

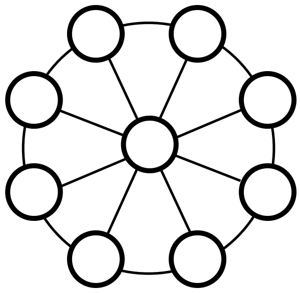
A zig-zag password is one where the digits alternatively rise and fall, always rising initially if the password has more than one digit. For example, 5 and 08362 are zig-zag passwords. Find the rule that only matches zig-zag passwords and uses the smallest number of symbols. (No justification is necessary.)

2(d) [5 marks]

Is there a rule, containing exactly 11 symbols, that can only match a single password? Justify your answer.

Question 2: ***Mu Torere***

The Maori strategy game of *Mu Torere* is played on a circular board with eight positions (called *kawai)* on the circumference and one position (called *putahi*) on the centre. It is a two-player game, where each player has four markers. Each position is either empty, or contains a single marker.



On a player's turn they can move one of their markers to an adjacent position. Adjacent positions are those joined directly by a line in the above diagram; i.e. the *putahi* is adjacent to all the *kawai*, and each *kawai* is also adjacent to the *kawai* immediately clockwise and anti-clockwise. The only exception is that a player cannot move their marker from a *kawai* if the two adjacent *kawai* also contain markers that belong to them.

Play alternates between the two players. A player loses the game if they are unable to move on their turn.

We will represent this board by a string of 9 characters; the first will represent the *putahi* and the last eight will represent the *kawai* (starting at the left of the two top positions and running clockwise around the board). The character `O` will represent a position containing a marker belonging to the first player, `X` will represent a position containing a marker belonging to the second player and `E` will represent an empty position.

For example, traditionally the game starts with both players having four adjacent *kawai*. We can represent this starting layout by the string `EOOOOXXXX`.

A simple strategy is for a player to choose their moves based on the following rules (where *leftmost* means *leftmost on our board representation*):

1. If there is a move which means my opponent will then lose, this move is played. If several such moves exist, choose the one that uses the leftmost of my markers.
2. If the first rule does not indicate which move to take and there are moves, after which my opponent will be able to make a move meaning that I will then lose, those moves are to be avoided (by moving the leftmost of my markers that avoids these moves). If it is not possible to avoid such a move, move the leftmost of my markers.
3. If the previous rules do not indicate which move to take, move the leftmost of my markers.

There are marks available for programs which are only able to deal with rules 1 and 3.

2(a) [24 marks]

Write a program that plays Mu Torere using the simple strategy for both players.

Your program should first read in a single line, containing 9 characters representing a starting layout. This will be a valid layout where each player will have exactly four markers.

Until your program terminates you should repeatedly wait for input, and then:

- If you receive the letter n, you should play the next turn of the game unless the current player has lost because there are no valid moves.
- If you receive the letter r, you should try to play the rest of the game, until one player has won or you believe that the game will never finish.
- Ignore any other input.

After the n command you should output the board layout. If the game has been finished you should also output **Player 1 wins** (for a first player win) or **Player 2 wins** (for a second player win) and then terminate. You should do the same thing after the r command, unless you believe the game will never finish, in which case you do not need to output a board layout and should just output **Draw** and then terminate.

Sample run

```
EOOOOXXXX
n

OEOOOXXXX

r

OXOEOXXXO
Player 1 wins
```

2(b) [3 marks]

If the board layout is XEOXXXOOO and it is the first player's turn, what possible moves can be made? You should show the board layouts after the valid moves. Which of these moves will be made if the simple strategy is followed?

2(c) [3 marks]

Do any layouts exist which, depending on whose turn it is to play, are winning layouts for both players? Justify your answer.

2(d) [5 marks]

Our board representation made two choices which do not affect the moves available to players during the game. It chose the starting position on the board (the left of the top two positions) and the direction (clockwise).

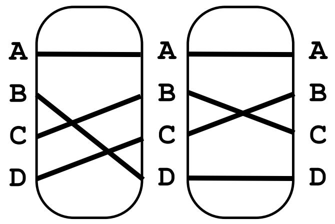
We can treat EOOOOXXXX and EXOOOOXXX as equivalent layouts, since the same board could lead to the two strings if we just change our starting position. We can also treat EOOOXOXXX and EOXXXOXOO as equivalent layouts, since the same board could lead to the two strings if we just change our direction from clockwise to anti-clockwise.

If we say two layouts are the same if, by choosing the starting position and direction, they could produce the same string, how many different layouts exist? [You might find it easier to consider the number of different layouts in the cases where the *putahi* is empty, contains one of the first player's markers or contains one belonging to the second player.]

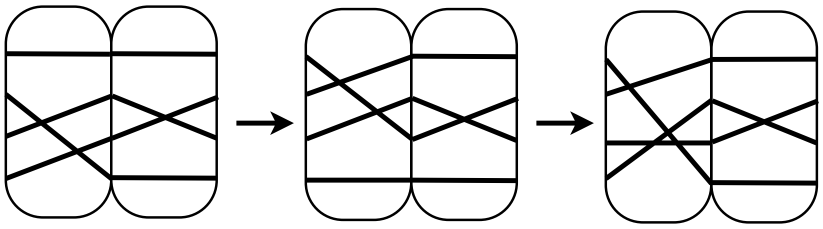
Question 2: ***Enigma Machine***

The *Enigma machine* was a device used to encrypt (and decrypt) text during the second world war. Your task in this question is to simulate a simplified version of this machine. Our simulation, which will encrypt the letters `A`, `B`, `C` and `D`, has two components: rotors and reflectors.

A rotor has a left side and a right side, each of which has four ports (`A`, `B`, `C` and `D`). Internal wiring links ports, so that each port on the left is linked to exactly one port on the right (and hence each port on the right will be connected to exactly one port on the left). Our simulation has two rotors which are placed next to each other so that the ports with the same value touch. Their wiring and starting positions are:

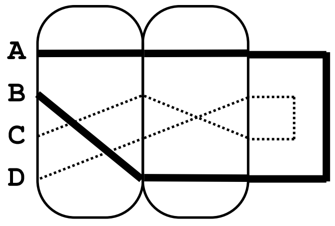


These rotors can turn (in the original Enigma machine they were disks). When a rotor turns, the ports (on both sides) which previously had the value `D` change to the value `C`, those which were `C` become `B`, `B`s become `A`s and `A`s become `D`s. The rotors are still arranged so that ports with the same values touch.


first rotor turning twice

On the right of the rotors is a reflector, which does not turn, and only has ports on its left side. Port `A` is wired to port `D`, and port `B` is wired to port `C`. The reflector is placed next to the second rotor so that the ports with the same value touch.

To encrypt a letter, we find its port on the left of the first rotor, and then follow the wiring through both rotors and the reflector until we get back to the left of the first rotor. The port we finish at gives us the encrypted version of our letter. After each letter has been encrypted we turn the first rotor. After every fourth letter has been encrypted we turn the second rotor; this happens ***in addition*** to turning the first rotor.



The turning of the rotors means that how we encrypt a letter depends on when we encrypt a letter. For example, with the rotors in their starting position, the letter `A` will be encrypted as `B`. The first rotor will then turn, and so a new letter `A` would be encrypted as `C`.

2(a) [25 marks]

Write a program to simulate an Enigma machine, with the specified rotors and reflector.

Your program should first read in a line containing a single number *n* ($0 \leq n \leq 2^{31}$) which indicates the number of letters which have already been encrypted by the machine. You should then read in a second line which will contain a single word (only using the letters A, B, C and D) with between 1 and 10 (inclusive) letters.

You should output a single line containing the encrypted version of the word.

Sample run

```
14
AAABBB

DBBDAD
```

2(b) [2 marks]

If no letters have been encrypted so far, how would AAAAAA be encrypted?

For the following two questions you should suppose that you are able to choose how the rotors are wired. This means you can choose how the ports on the left of a rotor are wired to those on the right of the rotor; it is still necessary that each port on the left is linked to exactly one port on the right. The reflector cannot be rewired.

2(c) [4 marks]

Suppose there was only a single rotor. In how many different ways could it be wired so that, no matter how many letters are encrypted A will always be encrypted to B, B will always be encrypted to A, C will always be encrypted to D and D will always be encrypted to C? How about if there were two rotors? *[NB: Two wirings are different if the rotors' wiring in their initial positions is different.]*

2(d) [4 marks]

Suppose that, in addition to being able to choose how the rotors are wired, you are allowed more than two rotors. Is it possible that, with the machine in its initial state, if we encrypted an A it would become a B, but if we encrypted a B it would become a C? Justify your answer.

Question 2: *Puzzle Game*

In some *puzzle games* points are scored by joining together pieces of the same colour. These pieces are then removed, new pieces are added, and the game continues until no more pieces can be removed. Your task in this question is to model a simple game.

Our game will be played on a 4 by 4 grid. The pieces, each of which fits in a single space on the grid, will be red, green or blue - to be indicated by `R`, `G` and `B` respectively.

When two or more adjacent pieces have the same colour they form a *block*. Pieces are adjacent if they touch horizontally or vertically, but not diagonally. All blocks are removed simultaneously from the grid and all the pieces that remain in the grid drop down to fill in any empty space.

```
RRGB          GB         G
GRBG  ---->  G BG  ---->   B
RRGB           G        G GB
GBRB         GBR        GBRG
```

The number of points scored is equal to the product of the sizes of the removed blocks. In the above example, a red block of 5 pieces and a blue block of 2 pieces were removed. This scores 10 points.

New pieces then drop down from above the grid to fill in any empty spaces. A *round* in the game is completed when these new pieces have appeared on the grid.

In our simple game, we will assume that the store of pieces above the grid is a repetition of the pattern in the original grid.

The following example shows the first three rounds in a game. The pieces above the grid show the next few pieces that are due to drop down from above to fill in the empty spaces. Observe that we ignore pieces above the grid when finding blocks of adjacent pieces; i.e. they only count when they have dropped down onto the grid.

```
RRGB         RRGB         RRRB
GRBG         GRBB         GRGB
RRGB         RBGB         RBBG
GBRB         GRRG         GRGB
RRGB         RRGB         RRRB
GRBG         GRBB         GRGB
RRGB         RBGB         RBBG
GBRB         GRRG         GRGB

RRGB         RRGB         RRRB
GRBG  ---->  GRBB  ---->  GRGB
RRGB         GBGB         RBGG
GBRB         GBRG         GRRG
```

2(a) [24 marks]

Write a program to play the puzzle game.

Your program should first read in four lines, giving the grid's rows in order; the first line being the top row of the grid. Each line will consist of four characters, each an `R`, `G` or `B`, giving a row from left to right.

Until your program terminates you should repeatedly input an integer, and then:

- If you receive a positive integer *n* (1 ≤ *n* ≤ 100) you should play *n* rounds of the game.
- If you receive the number 0 your program should terminate immediately.
- Ignore any other input.

After each positive integer you should output the board along with the total number of points that have been scored so far. If, while playing a round, there are no blocks to be removed, you should just output **`GAME OVER`** along with the final score, and then terminate.

Sample run

```
RRGB
GRBG
RRGB
GBRB
2

RRRB
GRGB
RBGG
GRRG

82

0
```

2(b) [3 marks]

Draw an example grid which will score 105 points in one round.

2(c) [4 marks]

What is the maximum score that can be achieved in a single round?

2(d) [4 marks]

Suppose that you are given the shape and the location of all the blocks on the grid, and you now need to place pieces on the grid to match those blocks. Will it always be possible to place the pieces? Justify your answer.

Question 2: *Die Tipping* *NB: Die is the singular of dice*

A normal *die* is a cube where the faces are numbered 1 through 6 and the numbers are arranged so that opposite faces add up to 7. In this question we will move a die on an 11 by 11 grid. Each square on this grid is the same size as the faces of the die; i.e. the die fits exactly onto each square on the grid.

Imagine the die is placed on the grid so that the number 1 is uppermost and 6 is touching the grid. The die is now rotated so that 2 is towards the top of the grid and 5 is towards the bottom. This question uses a die that has 3 on the left and 4 on the right. We will call this the *default orientation* of the die.

A *move* is made by tipping the die over one of the edges of the face that touches the grid. This will move the die one square on the grid and change the face that touches the grid. If the die is tipped off the left side of the grid it should be placed at the corresponding position (i.e. with the same y co-ordinate) on the right side of the grid, and vice versa. Similarly for the top and bottom of the grid (with the x co-ordinate staying the same).

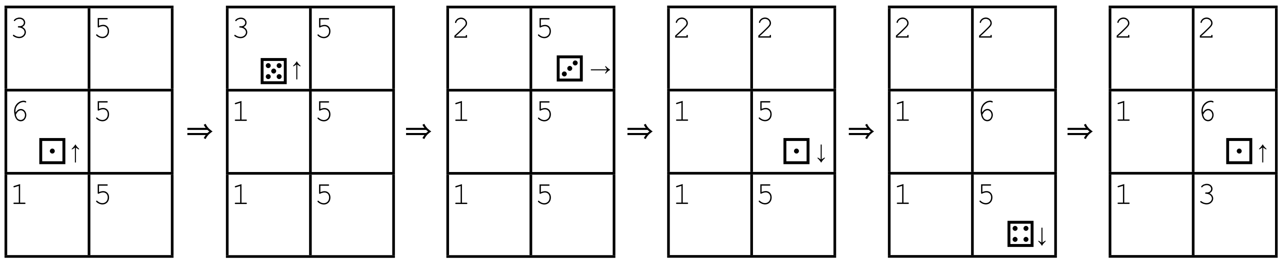
For example, suppose the die is in its default orientation in the middle of the grid. It now tips over the edge towards the bottom of the grid. The die moves one square closer to the bottom of the grid, and 5 now touches the grid. 1 is now towards the bottom of the grid, 6 the top, 2 will be uppermost and 3 and 4 will remain on the left and right respectively.

The die also has a *heading*, which is the direction on the grid it last moved.

The rules for moving the die are as follows:

1. Each square on the grid contains a number (1, 2, 3, 4, 5 or 6). Add the number on the current square to the value that is uppermost on the die. If this sum is greater than 6, subtract 6 from the sum.
2. Replace the number on the grid with this new value.
3. Depending on this value perform one of the following actions:
   - 1 or 6 – move one square according to the heading;
   - 2 – move one square in the direction 90° clockwise to the heading;
   - 3 or 4 – move one square in the opposite direction to the heading;
   - 5 – move one square in the direction 90° anti-clockwise to the heading.

In the following example part of the grid is shown at the end of each move. The numbers indicate the values written on the grid, the dots in squares indicate the value uppermost on the die (it starts in the default orientation) and the adjacent arrow indicates the current heading. A smaller than actual die is shown in the example for convenience.



2(a) [24 marks]

Write a program that models the die moving on the grid.

Your program should first read in a 3×3 grid, which will be in the form of three lines of three integers (between 1 and 6 inclusive). This 3×3 grid should be used as the centre section of the 11×11 grid for the simulation. The remaining squares of the grid will start with the value 1.

In each case the die begins in the centre of the 11×11 grid, in the default orientation, with a heading towards the top of the grid.

Until your program terminates you should repeatedly input an integer, and then:

- If you receive a positive integer *n* (1 ≤ *n* ≤ 100) you should make *n* moves.
- If you receive the number 0 your program should terminate immediately.
- Ignore any other input.

After each positive integer you should output the 3×3 grid surrounding the die. For each square on this grid that is outside the 11×11 grid you should output an **x**.

Sample run

```
1 3 5
1 6 5
1 1 5
1

111
135
115

1

111
251
151

3

221
161
131

0
```

2(b) [2 marks]

Suppose that the die is in the centre of the 11×11 grid, in the default orientation, with a heading towards the top of the grid. The numbers on the grid are such that it moves directly to and then over the right edge of the grid; i.e. on the sixth move it tips over the right edge. Considering only the squares it touches before tipping over the right edge, in how many different ways might they be numbered?

2(c) [4 marks]

Suppose that the die can be tipped in any direction on each move. How many sequences of 4 tips are there that finish with the die in its original position, but with a different face uppermost? How many such sequences are there taking 6 tips?

2(d) [5 marks]

Consider a sequence of squares occupied by the die as it travels. If, as the die continues to move, this sequence keeps repeating exactly and never changes, we will say that the die is *stuck in a loop*. Note that the die might have made several moves before reaching such a sequence.

Suppose that you are able to choose all the numbers of the board as well as the starting position, orientation and heading of die. Is it possible to select these values so that the die will never get stuck in a loop however long it moves? Justify your answer.

Question 2: *Accordion Patience*

Accordion Patience is a one-person game played with a normal *deck of cards*. Your task in this question will be to model various strategies for playing the game.

A deck of cards contains 52 cards. Each card has a *value* and a *suit* and there is one card of each possible combination in the deck. The suits are called Clubs, Hearts, Spades and Diamonds and the values in each suit are Ace, 2, 3, 4, 5, 6, 7, 8, 9, Ten, Jack, Queen and King. We will refer to the cards using the first letter of the value (or a digit) followed by the first letter of the suit. E.g. 4S for the "four of spades", TH for the "ten of hearts" and KD for the "king of diamonds".

Accordion patience is played by shuffling the cards, then dealing out all the cards into piles in a row from left to right. Initially all the piles will contain a singe card. Valid moves consist of taking a pile of cards and placing it on top of another pile of cards *to its left*. This can be done if the top cards of the two piles have either the same value or the same suit, and if the two piles are either next to each other in the row or separated by two other piles.

For example, suppose that we have piles with the following top cards 4S TH KD 4D. There are two valid moves, taking the pile with 4D on top and placing it either on the 4S pile (same value and separated by two piles) or on the KD pile (same suit and adjacent).

The game continues until either there is only one pile left (this is a *win*) or there are no more valid moves that can be made.

We will consider three different strategies for deciding which valid move to perform:

Strategy 1
Make the valid move using the right-most pile possible. If this pile can move to both an adjacent pile and a separated pile, move it onto the adjacent pile.

Strategy 2
Make the valid move that will create the largest pile possible. If there are several such moves, consider *only* those moves, and move the right-most pile possible (moving it to the adjacent pile if you still have a choice).

Strategy 3
Make the move that will leave the largest number of valid moves available next time. If there are several such moves, consider *only* those moves, and move the right-most pile possible (moving it to the adjacent pile if you still have a choice).

For example, suppose that we have the following piles:

	AD	8S	8D	4S	TH	KD	4D	TC
Number of cards in pile	1	3	1	2	1	2	2	1

By strategy 1 we will move the TC pile onto the TH pile. By strategy 2 we will move the 4D pile onto the KD one (the largest combined pile size is 4, the rightmost pile we can move to make this is the 4D pile and, since both moving it to the KD or 4S pile will create a pile of size 4, we pick the adjacent pile). By strategy 3 we will move the 8D pile onto the 8S pile which leaves 5 possible valid moves.

Initially the cards are ordered (from the top) with all the Clubs, then all the Hearts, then all the Spades then finally the Diamonds; with each suit ordered A, 2, …, 9, T, J, Q then K. They are shuffled by using six shuffling numbers *a, b, c, d, e* and *f* as follows: Cards are repeatedly taken, one at a time, from the top of the deck and placed on the bottom until the *a*th card is reached; this is dealt to the row rather than placed on the bottom of the deck. This is repeated but waiting until the *b*th card, then repeated for the *c*th, *d*th, *e*th and *f*th. We then repeat the process, starting with *a*. We continue until all the cards have been dealt.

For example, if the integers were all 1 the cards would be dealt in their original order and if the integers are 1, 3, 5, 7, 2 and 4 the cards would be dealt in the order:

AC, 4C, 9C, 3H, 5H, 9H, TH, KH, 5S, QS, AD, 5D, 6D, 9D, 2C, JC, KC, …, 7S 4H 3S

2(a) [24 marks]

Write a program that plays *accordion patience* by the different strategies.

You program should first read in six integers (each between 1 and 9 inclusive) indicating how the deck of cards should be shuffled. You should play the game three times, once for each strategy. Each time you play the game you should start with the cards in order, shuffle them according to the input, then play the strategy until you have won or cannot make another move.

The first line of your output should contain two cards, the left-most card after shuffling and dealing out the cards, followed by the right-most card. This should be followed by three more lines, the i^{th} of which containing information on the result of applying the i^{th} strategy — the number of piles that will be left after playing that strategy followed by the card on top of the left-most pile.

Sample run

```
1 3 5 7 2 4

AC 3S
6 9C
5 KC
4 4H
```

NB: The strategies will be marked separately.

2(b) [2 marks]

What would be the first 12 cards dealt if the six shuffling numbers were 2, 11, 3, 10, 4 then 9?

2(c) [4 marks]

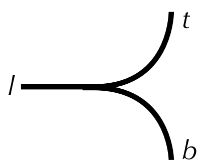
Two sets of cards are said to be *different* if there is at least one card that is one set but not the other; the order of the cards does not matter. How many different sets can be generated by shuffling the cards, using six shuffling numbers between 1 and 9 inclusive, and taking the first six cards? How about if each shuffling number could be between 1 and 10 inclusive?

2(d) [5 marks]

Suppose a winning game of accordion patience has been played; i.e. all the cards finished in a single pile. That pile is now picked up and, without shuffling, used to deal another game of accordion patience; the top card of the pile being the left-most card dealt. Is it always possible to make at least one move? Justify your answer.

Question 2: ***On the Right Track***

Consider a railway that consists of lengths of track that are connected together by simple *points*.



A train entering a point along one of the curved pieces of track must exit along the straight portion; i.e. a train entering from *t* or *b* will exit via *l*. A train entering via *l* will exit via *t* or *b* depending on the point.

We will consider two types of points:

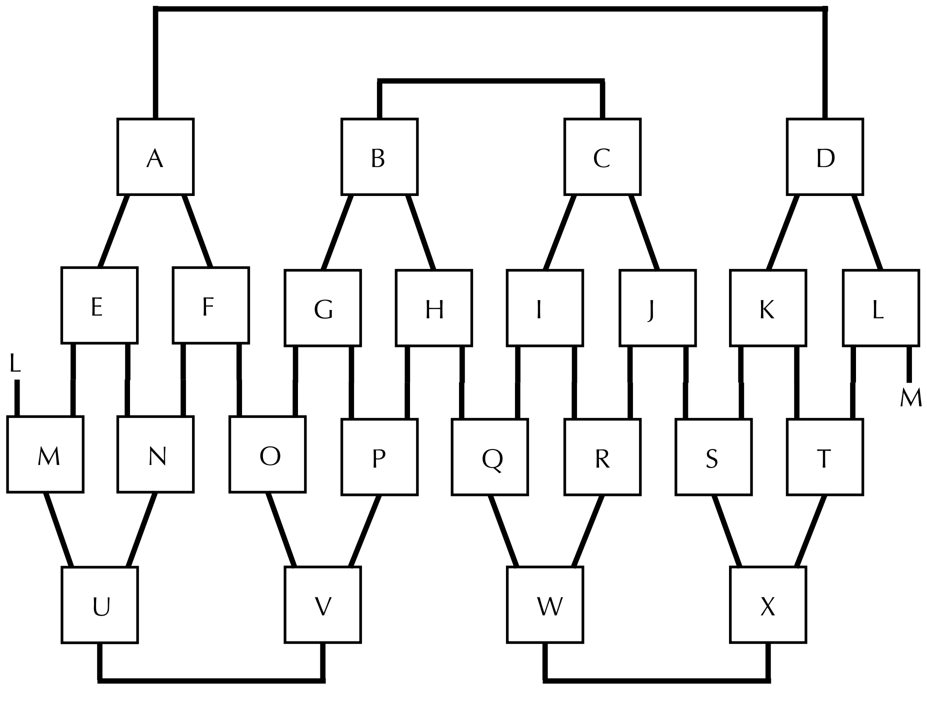
On a *lazy point*, entering from a curved portion sets the point so that, until the point is set again, a train entering from the straight portion will exit on the set curved portion. E.g. If a train enters the track from *t* (hence leaving via *l*) whenever it then enters the point from *l* it will exit from *t*, until the point's setting is altered by a train entering from *b*.

On a *flip-flop point*, every time a train has entered from the straight portion and exited on a curved portion, the point will change so that the next time a train enters the straight portion it will exit on the other curved track. Entering on the curved portion does not affect the point.

In this question we will be modelling a train moving along the railway network shown below. Each point is represented by a labelled square; each of which has one edge connected to a single piece of track (the straight portion for that point) and another edge connected to two pieces of track (the curved portions for that point.)

We will write $x \rightarrow y$ to indicate the train is between points x and y, moving towards y.

Initially each point is set to the left in the diagram. E.g. If the train starts at $A \rightarrow E$, and all the points are *lazy,* it would pass through points in the order E, M, U, V, O, F, A, …



NB: Points L and M are connected together by track, but for clarity this is not shown above.

2(a) [23 marks]

Write a program that models a train moving along the railway network.

Your program should read in three lines of input. The first line of input will contain six different uppercase letters, indicating that the corresponding points are *flip-flop* points; all other points will be *lazy* points. The second line of input will contain two letters, *x* then *y*, indicating the train is at $x \rightarrow y$. The final line of input will be an integer *n*, ($1 \le n < 10{,}000$), giving the number of points the train is to pass in the simulation.

You should output two letters, the first indicating the last point the train passed and the second indicating the next point the train will pass through.

Letters in the input and output should *not* be separated by spaces.

Sample run 1

```
GHIJKL
AE
6
FA
```

Sample run 2

```
GHIJKL
AE
100
VP
```

2(b) [2 marks]

Suppose all the points are *lazy* and the train is at $P \rightarrow V$. Write out the sequence of points the train passes up to and including when it passes P for the first time.

2(c) [6 marks]

Suppose, other than the layout of the railway, you know nothing about the points (either their type or how they are set) and that the train starts at $P \rightarrow V$. What can you say about the train's position after it has passed 1,000,000,000,000,000,000 points? Justify your answer.

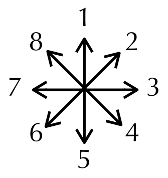
2(d) [4 marks]

A red and a blue train have been placed on different parts of the railway and will move simultaneously (i.e. whenever one train passes a point the other train will also pass a point). A safety system will kick-in, freezing all movement, if two trains attempt to pass the same point at the same time or try to occupy the same track between points at the same time. It is possible to place the trains on the railway (all points being lazy and initially set to the left) so that the safety system never kicks-in — in how many ways?

Question 2: *Neutron*

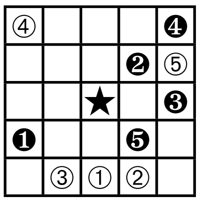
The game of *neutron*[1] is played on a 5×5 board between two players. The players have five pieces each and there is an additional neutron piece on the board. Each space on the board is either empty or contains a single piece. The object of the game is to manoeuvre the neutron to a space on either the top or bottom row of the board.

For convenience we will denote directions on the board as follows:

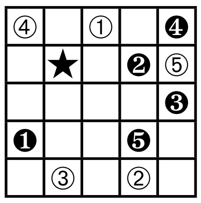
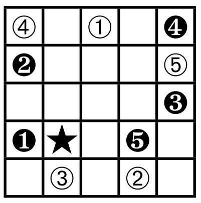


Players take alternate turns, each of which consists of first moving the neutron piece and then moving one of their own pieces. A piece is moved by sliding it *as far as possible* either horizontally, vertically or diagonally. If the neutron is moved to the bottom row the first player wins, if it reaches the top row the second player wins; it does not matter on whose turn the neutron was moved. If a player is unable to complete their turn (by being unable to move the neutron or subsequently one of their own pieces) the other player wins.

For example, consider the board on the right. The first player's pieces are denoted in white and the second player's pieces in black. The neutron is denoted by a star.



Suppose it is the first player's turn. The neutron has to be moved first. It can move in direction 1 (in which case it reaches the top of the board and the game ends with the second player winning) or direction 6 (reaching the bottom and winning the game for the first player) and cannot move in directions 2 or 4 (being blocked by ❷ and ❺ respectively). In can also move in directions 3, 5, 7 and 8 in which case the first player then needs to move one of their pieces.

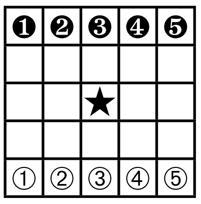



If the first player decided to move the neutron in direction 8, followed by ① in direction 1, the board would look as it does on the left. If the second player then moved the neutron in direction 5, followed by the ❷ in direction 7, the board would look as it does in the right. Note that, in each case, pieces are moved as far as possible.

For this question each player will use the following strategy:

- Each player has a fixed order in which they play their pieces (as the game progress). Once the last piece in their order has been played, they start again at the beginning of the order. E.g. if the player's order is 5, 4, 3, 2, 1 they will use (their) piece 5 on their first turn, 4 on their second, …, 5 on their sixth turn, etc…
- At the start of a turn, if they are able to move the neutron and immediately win the game *for themselves* the neutron is moved in the first such direction (from 1 to 8).
- If the only possible moves for the neutron will immediately win the game for their opponent, the neutron is moved in the first such direction (from 1 to 8).
- If the previous two conditions are not met, the player finds a way to play the next piece in their order. First the neutron is moved in the first direction (from 1 to 8) which does not end the game or prevent their chosen piece from moving, then the chosen piece is moved in the first direction (from 1 to 8) in which it can move.

Your program will not be asked to deal with any cases where there is no valid turn for the neutron and player's chosen piece.



At the start of the game the neutron is in the centre of the board. The first player's pieces occupy the bottom row and the second player's pieces occupy the top row. Pieces are in increasing order from left to right.

[1] The actual game of *neutron* is slightly different to the description given here; you should only implement the rules detailed in the question.

2(a) [26 marks]

Write a program that plays neutron according to the given strategy.

Your program should read in two lines of input. The first line of input will contain the digits `1`, `2`, `3`, `4` and `5` in an order that represents the first player's fixed order. The second line of input will contain, in a similar format, the second player's fixed order.

You should output the layout of the board at three moments during the game. The first layout should be after the first player's first turn; the second layout after the second player's first turn; the final layout after the game has terminated. All possible inputs to this question lead to a game that finishes.

Your board should use an `F` for pieces belonging to the first player, an `S` for pieces belonging to the second player, a `*` for the neutron and a `.` for an unoccupied space.

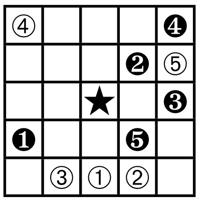
The three board layouts are marked independently and you can score marks if only one or two of your layouts are correct.

Sample run 1

```
1 2 3 4 5
1 2 3 4 5

SSSSS
F.*..
.....
.....
.FFFF

.SSSS
F...*
.....
...S.
.FFFF

..SSS
FF...
.....
...SS
.*FFF
```

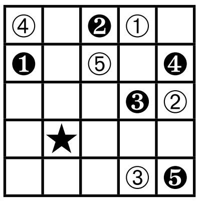
2(b) [3 marks]

Suppose the layout of the board is as follows and that it is the start of the second (Black) player's turn. Ignoring the strategy, in how many ways can the player take their turn?



2(c) [4 marks]

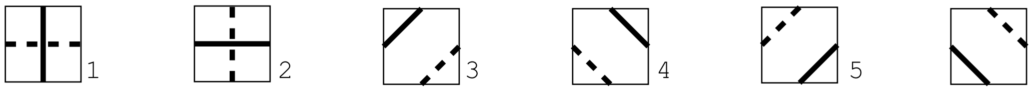
Suppose the layout of the board is as follows at the beginning of a turn and that you do not know whose turn it is. Ignoring the strategy, how many different layouts could the board have been in at the start of the previous turn?



Question 2: ***Loops***

Red and Green are playing a game which consists of placing square tiles on a grid and trying to form *loops* of their own colour. Each tile incorporates a red line and a green line, touching different edges of the tile.

There are six different types of tile that are used in the game. Red and green lines are shown by solid and dashed lines respectively. The numbers by the tiles will be used when inputting grids.



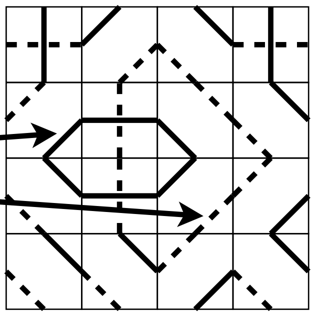
Tiles are placed adjacent to each other on the grid. A line meeting another line of the same colour — across a shared edge between tiles — is treated as a single continuous line across multiple tiles. If the lines are different colours the lines are treated as being separate. A loop is a line that begins on one tile and continues along additional tiles until it rejoins itself on the first tile.

At the end of the game each player scores a single point for each tile that contains part of a line forming a loop of their colour. A tile can contribute towards the score of both players.

In the example to the right there is a single loop; the numbers indicate how the grid will be input.

- This is a red (solid) loop covering six tiles.
- This is not a loop as the green (dashed) line does not rejoin itself.

The red player has scored 6 points and the green player 0.



```
1341
5264
4235
4454
```

2(a) [25 marks]

Write a program that reads in a grid and outputs the score for each player.

Your program should first read in a single integer ($2 \leq n \leq 6$) indicating the size of the (square) grid. You should then read in n lines representing the rows (starting at the top), each of which will contain n digits representing the tiles on the grid in order.

You should output two values: the score for the red played followed by the score for the green player.

Sample run

```
4
1 3 4 1
5 2 6 4
4 2 3 5
4 4 5 4

6 0
```

2(b) [2 marks]

A single tile is changed on the example grid so that Red no longer has more points than Green. How many possible changed grids are there?

2(c) [3 marks]

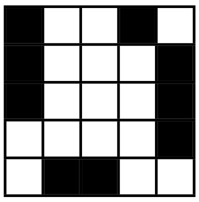
In how many ways can Red score 16 points on a 4 × 4 grid?

2(d) [5 marks]

Red and Green have just played the game on a 500 × 500 grid. Is it possible for Red to have scored just 1 point more than Green? Justify your answer.

Question 2: *Battleships*

In the game of *battleships* players secretly position ships of various sizes on a board and then try to determine the position of their opponent's ships. In this question we will consider the placement of one player's ships on a 10×10 board of squares. Each ship is placed either horizontally or vertically, covering an exact number of adjacent squares which are all on the board; a ship of size 1×*n* is called an *n*-ship. No part of any ship can be placed in a square that is adjacent (horizontally, vertically or diagonally) to part of another ship.



For example, part of a board showing four ships (two *3*-ships, a *2*-ship and a *1*-ship) is shown to the right. This is ***not*** valid as the *1*-ship is touching one of the *3*-ships diagonally. If the 1-ship was moved one square to the left everything shown would be valid.

The bottom left corner of the board has co-ordinate (0, 0). Co-ordinates are given as (x,y).

Our player starts by choosing (non-negative integer) values for `a`, `c`, `m` and `r` and then places each ship in turn using the following algorithm:

We are using the notation X←Y to mean "set X to Y"
X *mod* Y is equivalent to the remainder when X is divided by Y

1. `r` ← (`a` × `r` + `c`) *mod* `m`
2. Use the units digit of `r` as an *x* co-ordinate and the tens digit of `r` as a *y* co-ordinate
3. `r` ← (`a` × `r` + `c`) *mod* `m`
4. Starting at the calculated co-ordinates and *only* if it is valid, if `r` is even place the ship going horizontally to the right or if `r` is odd place it vertically upwards
5. If the ship is placed stop, otherwise continue from step 1.

Step 3 is never skipped, even if the value of `r` does not affect the validity of the ship in step 4.

For example, suppose the board contains a single *1*-ship at co-ordinate (3,0), that `a`, `c`, `m` and `r` are currently 3, 5, 53 and 20 respectively, and that the player is trying to place a *2*-ship on the board.

First `r` is set to 12 (3×20+5 = 65 and 65 *mod* 53 = 12) which represents co-ordinate (2,1), then `r` is set to 41 indicating that the ship is placed vertically upwards from (2,1) if valid. This is *not* valid as the ship would occupy (2,1) and (2,2) and the first square is diagonally adjacent to (3,0). `r` is then set to 22 representing co-ordinates (2,2), then set to 18 indicating that the ship is placed horizontally in squares (2,2) and (3,2) which is valid.

It is possible, for some values, that this algorithm will never find a valid position for the ship.

2(a) [27 marks]

Write a program that places ships in a game of battleships.

Your program should first read in three integers: a ($1 \le a \le 2^{15}$) then c ($1 \le c \le 2^{15}$) and finally m ($1 \le m \le 2^{15}$); the initial value of r (which is not input) is always 0. You should then follow the player's algorithm to place a *4*-ship, two *3*-ships, three *2*-ships and four *1*-ships (in that order) on an initially empty board.

You will only be given input that allows all the ships to be placed on the board.

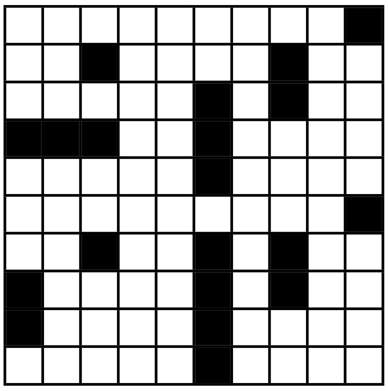
You should output ten lines, the *i*th of which containing information for the *i*th placed ship. Each line should contain the x then y starting co-ordinates for its ship, followed by an `H` or a `V` indicating whether the ship is placed horizontally or vertically. (For a *1*-ship you should still indicate the appropriate `H` or `V` based on the r value.)

Sample run

```
10 5 9999

5 0 V
5 5 V
0 6 H
0 1 V
7 2 V
7 7 V
2 8 H
2 3 V
9 4 V
9 9 H
```

2(b) [2 marks]

If a, c, m and r are set to 2, 3, 17 and 0 respectively, only four co-ordinates will ever be produced by the algorithm in step 2. What are they?



2(c) [3 marks]

The grid to the right corresponds to the sample run.

Another player secretly moves one ship to a valid position. A square can be guaranteed as empty if it is not possible for any ship to occupy it after this move. How many such squares are there?

2(d) [5 marks]

How many valid arrangements are there, on a 5x5 board, of a *4*-ship, a *3*-ship a *2*-ship and a *1*-ship? (All four ships must be on the board.)

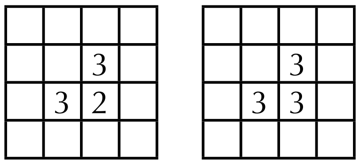
Question 2: ***Migration***

Your task in this question is to model people moving across a virtual landscape, consisting of an infinite grid of squares.

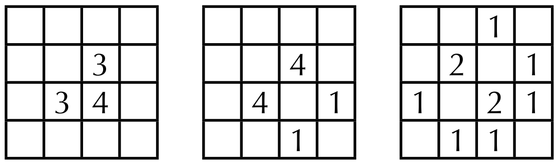
If a square contains 4 or more people it is *overcrowded*. If a square becomes overcrowded then people will *migrate* from the square to the neighbouring squares; these are the four squares directly adjacent horizontally and vertically. A migration consists of 4 people from a square moving simultaneously, one to each neighbouring square.

On each *step* of the simulation a new person is added to the landscape. Whilst there are any overcrowded squares, one of them (it does not matter which) will migrate; this is repeated until there are no more overcrowded squares. This ends this step of the simulation.

For example, suppose that the landscape starts with the only people as shown in the left figure. If the next step is to add a person to the square currently containing 2 people, there will be no overcrowding and the step will end with the grid as shown in the right figure.



If, on the next step, another person is added to the same square we will have an overcrowded square (left figure) and migration will take place (middle figure). This causes two overcrowded squares and these will successively migrate (right figure). As there are now no more overcrowded squares, the step will end.



The following method will be used to add people to the grid:

- We will place people (at the start of steps) within a 5×5 section of the landscape. The top row of this section contains positions 1 to 5 (from left to right), the next row positions 5 to 10, etc... so that the bottom right corner is position 25.
- You will be given the position of the square that receives a person in the first step.
- You will be given a sequence of numbers (each between 0 and 24). These will be used in order. When the last number has been used, return to the beginning of the sequence.
- On each successive step, generate the position for the next person by adding the next number in the sequence to the position used in the previous step. If this number is greater than 25, take 25 off total so that you are left with a position from 1 to 25.

For example, if the starting position is 10 and the sequence is 5 then 6, people will be placed at positions 10, 15, 21, 1, 7, 12, 18, 23, 4, etc...

NB: You must still model the landscape outside of this 5×5 section.
Migrations can still cause movements out of and into this section.

2(a) [24 marks]

Write a program that models migrations across an initially empty virtual landscape.

Your program should first read three integers: the starting position *p* ($1 \le p \le 25$) then a sequence size *s* ($1 \le s \le 6$) and finally *n* ($1 \le n \le 1000$). You should then read in *s* integers (each between 0 and 24 inclusive) indicating the sequence values in order.

You should output the 5×5 section of the landscape as it appears after *n* steps of the simulation. You should indicate the number of people in each square; do not indicate the position values.

Sample run

```
8 1 6
0

0 0 1 0 0
0 1 2 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

2(b) [2 marks]

Consider an empty virtual landscape. What is the smallest number of people that can be added to a 5×5 section of the landscape such that a migration occurs from outside to inside the section?

2(c) [4 marks]

The following grid has been generated with 8 steps and a sequence size of 3. Give a corresponding input that produces this output. How many such inputs are there?

```
1 0 1 0 0
1 0 1 0 0
1 0 0 0 0
1 0 1 0 0
1 0 0 0 0
```

2(d) [4 marks]

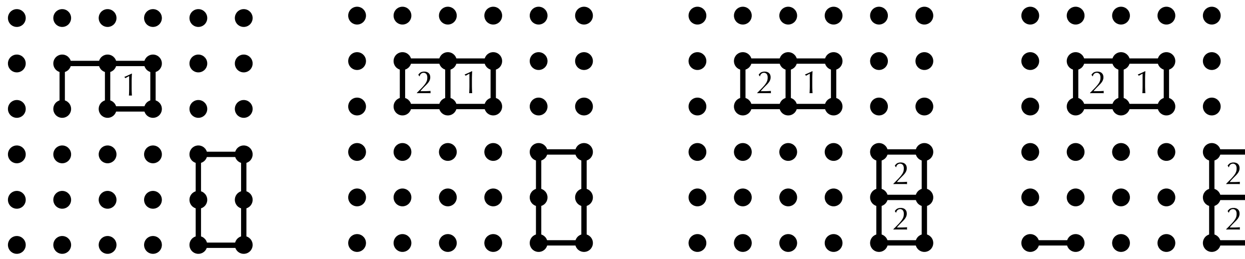
If you are given the landscape at the end of a step, along with the position where the person was added at the start of the step, can you always determine the landscape at the start of the step? Justify your answer.

Question 2: *Dots and Boxes*

The game of *dots and boxes* is being played on a 6×6 grid of dots. Two players take turns selecting two adjacent dots (horizontally or vertically) and joining them by an edge. If the edge finishes off any (1×1) squares then those squares are won by the current player who then gets another turn, otherwise it becomes the other player's turn. Two adjacent dots can only be joined once.

For convenience in this question, we will number the dots 1 to 36, with the top row running from 1 (left) to 6 (right), the next row from 7 to 12 etc.

For example, the following sequence of grids shows a game in progress:



We start with the grid on the left where player 1 has won a single square. It is player 2's turn and they choose to join dots 14 and 15, finishing off the square towards the top left. Player 2 now gets another turn and chooses 29—30 which simultaneously finishes off both bottom right squares. Player 2 now gets another turn and joins 31—32 on the bottom left. It is now player 1's turn again.

In this question you will simulate two players who follow a very simple strategy.

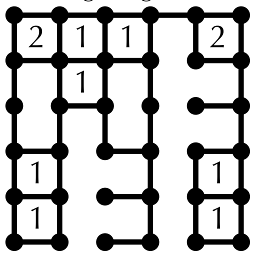
Each player uses a *modifier* and keeps track of a *position*. At the start of their turn, a player increases their position by their modifier and looks at the dot in the new position. If it is possible they then join an edge to that dot. If it is not possible they repeatedly increase the position by 1 (rather than the modifier) until they find a dot to which they can connect an edge.

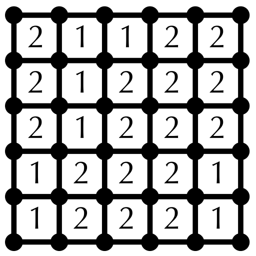
A player increasing their position past dot 36 starts again at dot 1. E.g. If a position of 35 is modified by 3, the new position will be 2.

When trying to join an edge from a dot each player first looks to see if an edge can be added going upwards. If this is not possible, player 1 tries edges in a clockwise direction (i.e. first seeing if an edge goes to the right, then downwards and finally to the left) and player 2 tries edges in a counterclockwise direction.

For example, suppose no dots have been joined, player 1 starts with position 4 and a modifier of 10, and player 2 starts with position 14 and a modifier of 23:

- Player 1 increases their position to 14. An edge is added upwards joining 14—8.
- Player 2 increases their position to 1 (14 + 23 = 37, which is a single position beyond 36). An edge cannot be added upwards or to left (dot 1 is at the upper left corner), so an edge is added downwards joining dot 1 to dot 7.
- Player 1 increases their position to 24. An edge is added upwards joining 24—18.
- Player 2 increases their position to 24. An edge already exists upwards, so an edge is added to the left joining 24—23.
- The bottom left figure shows the grid after 46 turns; it is player 1's turn and they are at position 8. They look at position 18 (no possible edges), then 19 (no possible edges), then 20 and join 20—21.
- The bottom right figure shows the grid after 60 turns.





2(a) [24 marks]

Write a program that plays a game of *dots and boxes*.

Your program should first read five integers: the starting position p_1 ($1 \le p_1 \le 36$) then modifier m_1 ($1 \le m_1 \le 35$) for player 1, followed by starting position p_2 ($1 \le p_2 \le 36$) then modifier m_2 ($1 \le m_2 \le 35$) for player 2, followed by the number of turns to simulate t ($1 \le t \le 60$).

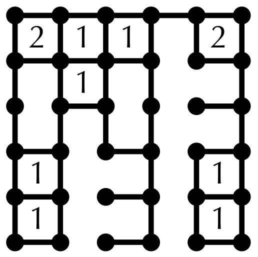
You should output a grid showing which squares have been won after t turns. Output an `X` for player 1, an `O` for player 2, and a `*` for a square that has not yet been won. This should be followed by the number of squares won by the first player, then the number of squares won by the second player.

Sample run

```
4 10 14 23 47

O X X * O
* X * * *
* X * * *
X * * * X
X * * * X

8 2
```

2(b) [2 marks]

Suppose the grid is in the following configuration and it is player 1's turn. If players can adopt any strategy (not just the very simple strategy), what is the maximum number of squares player 1 can win at the end of the game?



2(c) [3 marks]

How many different sets of input, where t=60, lead to player 2 winning all 25 squares?

2(d) [5 marks]

When an edge is added either 0, 1 or 2 squares are completed. Suppose the players are allowed to use any strategy, and the game is played until every edge has been added. If 2 squares have never been simultaneously completed during the game, what can you say about the pair of dots that were joined on the last turn? Justify your answer.

Question 2: ***Decoder Ring***

A *decoder ring* consists of two adjacent dials and is used to encrypt (or decrypt) messages. The two dials are lined up, so that each position on the first is touching one on the second, and dials can rotate so that they can be aligned in different ways. The first dial has 26 positions, lettered from A to Z in order, and the second dial has the same letters but not necessarily in the same order. A letter is encrypted by finding it on the first dial and using the corresponding touching letter on the second dial.

For example, suppose that the second dial has been lettered from Z to A (i.e. reverse order) and that the A on the first dial is touching Z on the second:

- The letter A would be encrypted to the letter Z, the letter B by the letter Y etc.
- If the second dial is now rotated until the A on the first dial is touching X on the second, the letter A would be encrypted to the letter X, the letter B by W etc.

The order of the letters on the second dial will be generated as follows, from the number *n*:

- Place the letters A to Z clockwise in order around a circle;
- Starting with A count clockwise round the circle until *n* is reached;
- *Remove* the selected letter from the circle — it becomes the first letter on the second dial;
- Starting from where you left off, count to *n* again to select the next letter;
- Continue until all the letters have been selected.

For example, if *n* is 5 the letters will be chosen in the order: E, J, O, T, Y, D, K, Q, W, ...

The dials will be aligned so that the first letter selected for the second dial is initially touching the letter A on the first dial, the second letter selected touching B etc. After each rotation of the dials, the letter on the second dial previously touching B on the first dial will be touching A on the first dial, etc.

A word is encoded by encrypting each letter in turn, after each encryption rotating the dial by a single position.

For example, if *n* is 5 the word ABCD will be encrypted as EOYK.

2(a) [24 marks]

Write a program that encrypts a word.

Your program should first input a single integer *n* ($1 \le n \le 1000000$) followed by a word *w* of between 1 and 8 upper case letters.

You should output the first 6 letters of the second dial generated from *n*, followed by the encrypted version of *w*.

Sample run

```
5 ABCD
EJOTYD
EOYK
```

2(b) [3 marks]

What are the first 6 letters of the second dial generated from *n* = 1,000,000,000?

2(c) [4 marks]

The word ABCDEFGHIJKLMNOPQRSTUVWXYZ is to be encrypted. Does a second dial exist that will encrypt this to a word that contains all 26 letters? Justify your answer.

2(d) [4 marks]

A word is encrypted *without rotating the dials* and this encrypted word encrypted again and again until the first time the original word is produced. This will always occur eventually. What is the largest number of encryptions that might be required?

Question 2: *Trail*

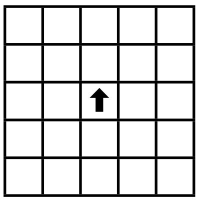
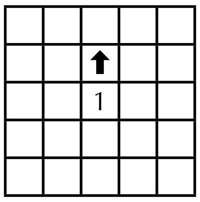
An *explorer* is moving across a rectangular grid of squares, leaving behind a *trail*. Keen to experience the unknown, they only move into squares that do not contain part of the trail. The trail fades over time and the explorer can re-enter a square once its trail has disappeared, although they will leave behind a new trail.

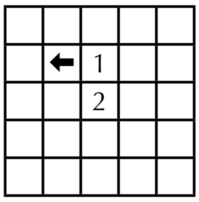
If the trail fades after *n* moves, it means that the trail in a square will disappear once the explorer has finished moving *n* times.

The explorer is following a set of instructions. If the instructions indicate *left* or *right* they will turn 90 degrees in the corresponding direction, otherwise they will stay facing *forward*. The explorer will then move to the adjacent square in the direction they are facing, so long as it does not contain the trail. If this is not possible, they will turn 90 degrees to the *right* and try to move. This will be repeated until they have moved or tried all four directions.

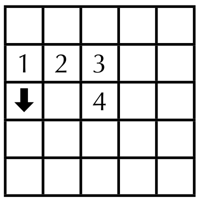
Each entry in the explorer's instructions is either `L` (*left*), `R` (*right*) or `F` (*forward*). The explorer works through the entries in order, restarting at the beginning once they have all been used.

For example, suppose the trail fades after the explorer has moved 8 times and their instructions are `FL`. (In the diagrams, the arrow will indicate the explorer and the direction they are facing, with the numbers indicating the age of the trail.)

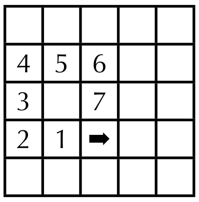





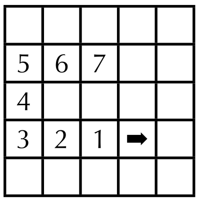
The explorer first moves forward, and then moves to their left. After this second move they are facing in a new direction and the trail covers two squares.



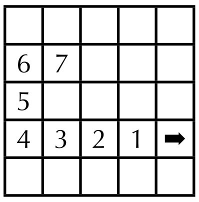
The explorer will now restart their instructions. Another move forward in the direction they are facing, followed by a move to the left.



After another three moves the explorer's next instruction is a *left*. They turn to the left but are facing a square that contains the oldest part of the trail. They therefore turn to the right …



… and move into the empty square. They have now moved 8 times since the oldest part of the trail was formed, so it disappears.



They continue with the next instruction, which is *forward*.

2(a) [24 marks]

Sample run

```
8 FL 9
(2,-1)
```

Write a program that tracks the explorer.

Your program should first input an integer *t* ($1 \le t \le 100$) indicating the number of moves for the trail to disappear, followed by a word *i* of between 1 and 10 upper case letters (each `L`, `R` or `F`) indicating the explorer's instructions, followed by an integer *m* ($1 \le m \le 10000$) indicating how many moves the explorer makes.

If the explorer is ever unable to make a move they will stop and not attempt to make any more moves.

You should output the co-ordinates of the explorer after making their moves.

The explorer starts at (0,0) facing (0,1). To their left is (-1,0) and to their right is (1,0).

2(b) [2 marks]

The introduction to this question gave the diagram after `8 FL 9`. Give the diagram after `8 FL 16`.

2(c) [3 marks]

If the trail never disappears and the explorer's instructions are `L`, how many moves are required for every square at (*x*,*y*) to be visited, where $-10 \le x \le 10$ and $-10 \le y \le 10$?

2(d) [5 marks]

The explorer's instructions are `LLRFFF` and they are able to move indefinitely. What is the largest possible number of moves before the trail disappears?

Question 2: *Alpha Complex*

Alpha Complex consists of many connected rooms. Each room is identified by a single (unique) letter and each exit from a room is marked with the letter identifying the room to which it is connected.

A *spy* is moving systematically through Alpha Complex. For each room they keep track of whether they have visited it an even or an odd number of times. They also record, for each room and for each exit, whether they have left the room through that exit an odd or even number of times.

The spy moves between rooms according to the following rules:

- If they have visited the room an odd number of times, they will leave through the exit which is marked with the first letter alphabetically;
- If they have visited the room an even number of times, they will find the first exit alphabetically that they have left through an odd number of times. If that is the last exit alphabetically in this room they will leave through it, otherwise they will leave through the next exit alphabetically.

When the spy starts exploring Alpha Complex, they have visited their starting room an odd number of times (i.e. once), each other room an even number of times (i.e. zero) and have left through each exit an even number of times (i.e. zero).

For example, suppose that they start in room `X`, which has exits to `B`, `I` and `O`:

- They have visited `X` an odd number of times, so they leave the room through the first exit alphabetically, which is `B`;
- When they are next in X they will have visited it an even number of times. The first exit alphabetically that they have left through an odd number of times is `B`. As this is not the last exit alphabetically, they leave through exit `I;`
- Subsequent visits to the room will see them leave through `B`, `O`, `B`, `I`, `B`, `O`, …

Alpha Complex consists of *r* (r ≥ 3) rooms, identified by the first *r* letters of the alphabet. The spy is in possession of a secret *plan*, giving the connections between the rooms. This plan is an ordered list of *r-2* letters and the spy can construct a map of the complex as follows:

- The spy will *choose* the first room alphabetically which has not yet been chosen and which is not in the plan. The chosen room is connected to the first room in the plan. The first room is then removed from the plan;
- The above step is repeated until the plan is empty;
- There will be two rooms which have not yet been chosen. These two rooms are connected together.

If two rooms are connected, each room has an exit to the other room.

For example, suppose Alpha Complex has 6 rooms and the plan is `E`, `A`, `E`, `D`:

- The first room which is not in the plan is `B`, which gets connected to `E` (the first room in the plan). The updated plan becomes `A`, `E`, `D`. Note that only `B` has been chosen and that the other instance of `E` in the plan is not removed;
- The next chosen room will be `C`, which gets connected to `A`;
- The next chosen room will be `A` (it has not been previously chosen and is no longer in the plan) and it is connected to `E`;
- `E` is now chosen and connected to `D`;
- The plan is now empty. As `D` and `F` were never chosen, they are now connected.

2(a) [24 marks]

Write a program that tracks the spy through Alpha Complex.

Your program should first input a string consisting of n ($1 \leq n \leq 8$) uppercase letters (each from the first n+2 letters of the alphabet) indicating the spy's plan of Alpha Complex, followed by p and q ($1 \leq p < q \leq 1{,}000{,}000$) indicating moves the spy will make.

The spy starts in room A.

You should first output $n+2$ lines. The first containing (in alphabetical order) the rooms connected to A, the second those connected to B, etc. These should be followed by a line containing the spy's location after p and q moves.

Sample run

```
EAED 1 2
CE
E
A
EF
ABD
D
CA
```

2(b) [2 marks]

What is the plan if Alpha Complex has 3 rooms and the connected rooms are A-B and A-C? What is the plan if there are 6 rooms and the connections are A-B, A-C, A-D, A-E and A-F?

2(c) [4 marks]

After an unknown number of moves in Alpha Complex the spy arrives in a room having forgotten if it has been visited an odd or an even number of times. How can they continue their systematic exploration without deviating from their original intended route? Justify your answer.

2(d) [4 marks]

If Alpha Complex contains 8 rooms, how many different plans are there where the first four connections the spy adds when constructing the map are (in some order) A-E, B-F, C-G and D-H?

Question 2: *Tri-iso-game*

A game is being played on an infinite *triangular grid*, made up from *equilateral triangles* (with side length of 1 unit). They are regularly tiled, with each edge (except at the ends) of a triangle touching one edge of another triangle. The grid is orientated so that each triangle has one edge horizontal.

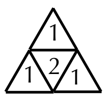
At the start of the game a single point-upwards triangle contains a `0`; all other triangles are *empty*. On each move of the game an empty triangle, which is adjacent to a triangle already containing a number, will be filled with a number. Player 1 will fill triangles with `1`s, player 2 will fill triangles with `2`s, etc. Players take it in turns to move; with *n* players this would be 1, 2, …, *n*, 1, 2, …, *n*, 1, …

When deciding which triangle to fill, players will traverse around the perimeter of the filled triangles. Each player is positioned on one triangle's edge (which is on this perimeter) and will traverse by moving to an adjacent edge. All players move around the perimeter in the same direction for the entire game. Note that players are positioned on edges and *not* in the triangles adjacent to edges.

Players start against the left-hand edge of the triangle containing a `0`. After a single traversal they will move to the right-hand edge; i.e. clockwise around the triangle.

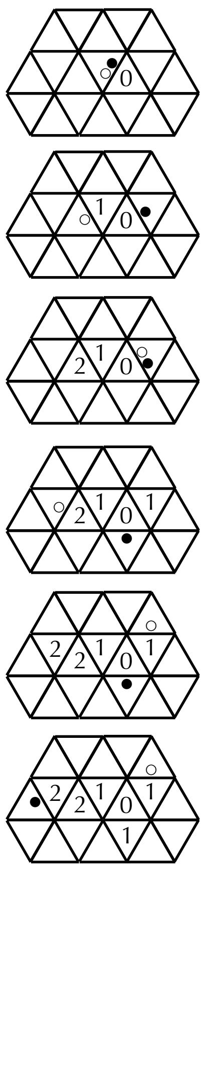
Each player has a maximum number of traversals they can use each move. After they have finished traversing the perimeter, they fill the empty triangle, adjacent to where they ***started*** the move, with their number. If, after a triangle is filled, *any* player finds that their current position is no longer touching an empty triangle, they reposition themselves on the left-hand edge of the left-most filled triangle on the highest row of the grid.

Four adjacent triangles (of length 1) can be seen as larger triangle (of length 2). A player scores 1 point for all such larger triangles where the three outer triangles are filled with their number. It does not matter if the inner triangle is empty or contains a different number. For example, the figure to the right is worth 1 point to player 1.



Players try to set themselves up for future turns. When traversing the perimeter, a player will stop before their maximum number of traversals in the following situation: they have reached an edge (after at least one traversal) whose empty adjacent triangle would score them a point *if* it were filled in. Note that it is not actually filled when they stop.

For example, suppose the game has 2 players. Player 1 can traverse sixteen edges a move; player 2 can traverse two edges. In the diagrams to the right, ● and ○ show the positions of the two players.



- Initially, there is a single filled triangle containing a `0` and both player 1 (●) and player 2 (○) are positioned against its left edge;
- Player 1 traverses sixteen edges, looping around the `0` triangle until finishing against its right edge. They then fill in the triangle adjacent to where they started with a `1`. As player 2 is no longer adjacent to an empty triangle they are repositioned.
- Player 2 now traverses two edges, finishing in the same position as player 1. They fill the triangle adjacent to their starting edge with a `2`.
- Player 1 starts to traverse the perimeter. Note that they do not stop after the first edge has been traversed; if they filled in the triangle at this stage they would not gain a point as the triangle adjacent to their starting position is not placed until the move's end. After sixteen edges they eventually stop and fill in a triangle. Once again Player 2 is repositioned.
- Player 2 moves two edges. Note that, even though two successive edges were horizontal, it still takes two traversals. A triangle is filled in.
- Player 1 moves and stops after 7 traversals as filling in the adjacent triangle would score them a point. They do not fill in the triangle on stopping. Their traversals now being completed they fill in their starting triangle, which is co-incidentally where they had stopped, score a point and are then repositioned.

2(a) [24 marks]

Write a program that plays the game.

Your program should first input a line containing two integers, p ($1 \leq p \leq 5$) then m ($1 \leq m \leq 5{,}000$), indicating the number of players and a number of moves to play. This will be followed by a line containing p integers, each between 1 and 100 inclusive, the i^{th} indicating the maximum number of traversals player i can make each move.

You should first output p lines. The i^{th} line should contain the final score for player i. This should be followed by a line containing the length of the *outside* perimeter of the filled triangles at the end of the game.

Sample run

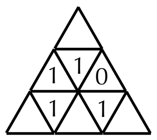
```
2 5
16 2
1
0
8
```

2(b) [4 marks]

In a 1 player game, what does the grid look like after 5 moves if the player can make 1 traversal on each move? What does the grid look like after 5 moves if the player can make 2 traversals each move?

2(c) [3 marks]

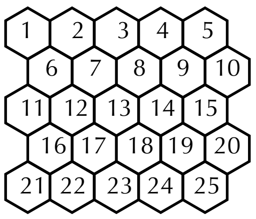
In a 1 player game, after 4 moves the grid is filled as in the diagram. What is the smallest possible value for the player's maximum number of traversals each move?



2(d) [5 marks]

In a 2 player game, where player 1 can make 5 traversals per move and player 2 can make 7 traversals per move, how many unfilled triangles are inside the perimeter after 5000 moves?

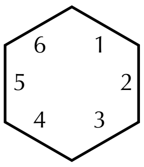
Question 2: *Game of Drones*



Two bee colonies are fighting for control of a *hive*.

The hive is represented by a 5x5 array of hexagons, numbered as in the diagram to the right. The six edges of each hexagon are numbered, as in the diagram below. Some edges are *owned* by either the *red* colony or the *blue* colony. If a colony owns more edges of a hexagon than another colony, that colony *controls* the hexagon. Initially, no edges are owned by either colony and hence no hexagons controlled.

Two drones are in the hive: a red drone on hexagon 1 facing edge 1 and a blue drone on hexagon 25 facing edge 6. Drones jump between hexagons in numeric order, returning to 1 after hexagon 25. Whilst jumping, drones do not change the direction they are facing.



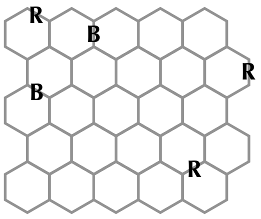
The game consists of a number of *skirmishes* followed by a number of *feuds*.

In each skirmish:

- The red drone takes ownership (for the red colony) of the edge it is facing, it then rotates 60° *clockwise* to face a new edge and finally it jumps *r* hexagons along the hive.
- The blue drone similarly takes ownership of the edge it is facing, it then rotates 60° *anti-clockwise* to face a new edge before finally jumping *b* hexagons.

For example, if *r* is 9 and *b* is 3:

- In the first skirmish, red takes ownership of edge 1 of hexagon 1 and blue takes ownership of edge 6 of hexagon 25.
- Red faces edge 2 (clockwise from 1) and jumps to hexagon 10 (1+9). Blue faces edge 5 (anti-clockwise from 6) and jumps to hexagon 3 (as 1 is the hexagon following 25).
- In the second skirmish, red takes ownership of edge 2 of hexagon 10 and blue takes ownership of edge 5 of hexagon 3.
- In the third skirmish, red takes ownership of edge 3 of hexagon 19 (which was previously owned by blue as that edge is shared with hexagon 25) and blue takes ownership of edge 4 of hexagon 6.

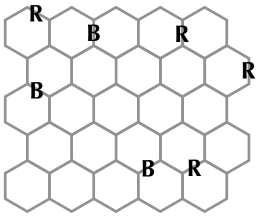


After the skirmishes it is time for feuding. In each feud:

- The red colony will take ownership of their preferred *un-owned* edge, followed by the blue colony doing likewise.
- When selecting an edge a colony *prefers* edges that gain them control over the most hexagons. Between edges that give them the same amount of control they prefer those that take away the most control from the other colony. After that, preference is based on hexagon number; lowest for the red colony and highest for the blue colony. After that, preference is based on direction number; lowest for the red colony and highest for the blue colony.

For example, continuing on from the *hive* after the previous skirmishes:

- Red takes ownership of edge 2 in hexagon 4. Taking ownership of this edge gives red control over hexagons 4 and 5. This is an edge in the lowest number hexagon that can gain red control over two additional hexagons. It is also the lowest edge in the hexagon that gains such control as edge 1 is only adjacent to a single hexagon. An edge in hexagon 1 cannot gain control of hexagon 1 (red *already* controls it) and at most removes the blue control from hexagon 2 or 6; a total gain of no hexagons. An edge in hexagons 2 or 3 would take control away from blue and, at most, gain control of either hexagon 4, 7 or 8; a total gain of control of only a single hexagon.
- Blue takes ownership of edge 6 in hexagon 24, similarly taking control of two hexagons.



2(a) [27 marks]

Write a program that plays the *Game of Drones*.

Your program should first input a line containing two integers, *r* (1 ≤ *r* ≤ 25) then *b* (1 ≤ *b* ≤ 25), indicating the number of hexagons the red and blue drones jump by during each of the skirmishes. This will be followed by a line containing two integers, *s* (0 ≤ *s* ≤ 1000) then *f* (0 ≤ *f* ≤ 40), indicating the number of skirmishes and feuds respectively.

Sample run

```
9 3
3 1
6
6
```

Marks are available for cases where there are no feuds

You should output two lines, the first containing the number of hexagons controlled by the red colony and the second the number controlled by the blue colony.

2(b) [3 marks]

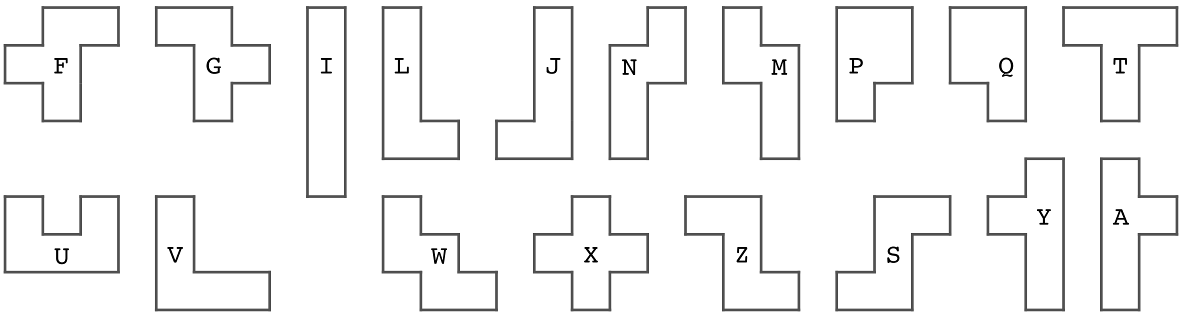
Show the hive after 0 skirmishes and 7 feuds, making it clear which edges are owned by each colony.

2(c) [4 marks]

Suppose the drones continue skirmishing until an edge changes ownership between the two colonies. Assuming 1 ≤ *r, b* ≤ 25 and the values are chosen so that an edge will eventually change ownership, what is the minimum number of skirmishes that take place before the skirmish where an edge changes ownership? What is the maximum number?

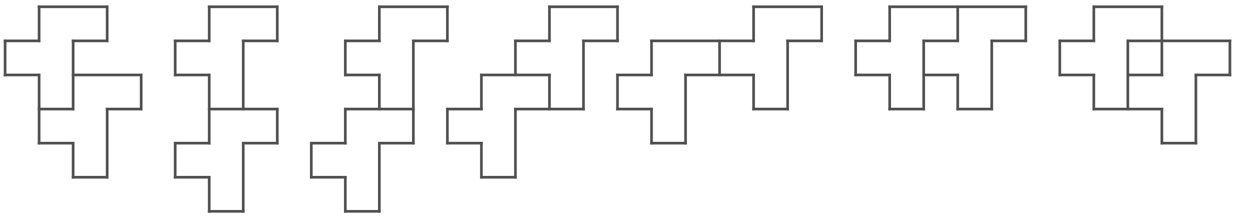
Question 2: ***Pentominoes***

Pentominoes are the shapes that can be made when five equal sized squares are connected, with each square always edge-to-edge with another square. The number of different pentominoes depends on whether they are considered different when rotated and/or flipped over. We will consider the set of 18 pentominoes shown below (which are distinguishable if rotations only were allowed) although **throughout this question we will not permit flipping or rotations**.



Pentominoes can be combined to make more complicated shapes. When connecting pentominoes *at least one* square in each pentomino must be edge-to-edge with a square in a different pentomino.

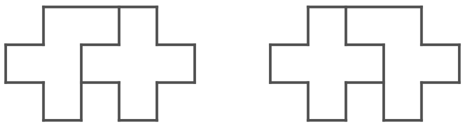
For example, two F pentominoes can be combined to make the following seven *distinguishable* shapes. These are the only combinations that can be formed because we are not permitting flipping or rotations.



Note that the last combination contains an *internal hole*. I.e. there is a gap inside the combined shape which is separated from the space outside the shape.

Two shapes are not distinguishable if, without flipping or rotation, the layout of their squares is the same, even if they were formed from different pentominoes.

For example, the following two shapes, generated from `FX` and `GX`, are not distinguishable:



2(a) [23 marks]

Write a program that calculates in how many ways a pair of pentominoes can be combined.

Your program should input a line containing a string of two capital letters, indicating the pentominoes to be joined. You should output the number of *distinguishable* shapes that can be formed by combining those pentominoes.

Sample run

```
FF
7
```

2(b) [2 marks]

How many shapes, made from combining `XW`, can also be made from two other pentominoes?

2(c) [5 marks]

How many distinguishable shapes can be formed by combining `III`? How about `LIV`?

2(d) [4 marks]

How many pairs of (potentially identical) pentominoes can be combined to form a shape with an internal hole?

Question 2: Parsing Lists

In this task you will manipulate lists of integers to create new lists. There are three *fundamental* lists that you will start from:

- `E` — the list of even integers, in order, starting with 2;
- `O` — the list of odd integers, in order, starting with 1;
- `T` — the list of integers, in order, where each one occurs the same number of times in the list as its value, starting 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, …

Given a list `L`, `L[i]` indicates the value in the *i*th position of `L`. If `L` and `R` are two lists, `L` on the left and `R` on the right, when combined they are replaced by list `C` where the *i*th position of C is `R[L[R[i]]]`. I.e. we find a value in list `R`, use it to find a value in list `L` and then use that value to find a final value in list `R`.

For example, if we combine `O` (on the left) and `E`:

- The 1st value in `E` is 2, the 2nd value in `O` is 3, and the 3rd value in `E` is 6;
- The 2nd value in `E` is 4, the 4th value in `O` is 7, and the 7th value in `E` is 14;
- The combined sequence is 6, 14, …

A *description,* giving details on the required manipulations, contains lists as well as brackets. Lists in the description will be combined until there is only a single list remaining. Where part of a description lies between brackets, it is treated as an independent description and that part will be manipulated (without combination with external lists) until reduced to a single list, at which point those brackets are then removed. When a description contains no brackets, the leftmost two lists are repeatedly combined forming a new description, until a single list remains.

For example:

- `E` is a description of a single list, so there is nothing further to do;
- `OE` involves a single combination leading to 6, 14, 22, 30, 38, …
- `E(OE)` will be the combination of `E` with `OE`, i.e. 94, 222, 350, …
- `EOT` is manipulated by first combining `EO`, then combining the resulting list with `T`, and is equivalent to `(EO)T`.

2(a) [25 marks]

Write a program that calculates the *i*th value in a fully manipulated description.

Your program should input a string *d* of up to 12 characters (`E`, `O`, `T` or brackets) followed by *i* ($1 \le i \le 2^{60}$). You will not be required to access any value over 2^{61}.

You should output the *i*th value in the fully manipulated description *d*.

Sample run

```
E(OE) 3
350
```

2(b) [2 marks]

How many 2s appear in `TT`?

2(c) [5 marks]

A *shortlist* of a list contains the first 100,000 entries of the list in order, and two shortlists are different if there is at least one position *p* where the *p*th value of the two shortlists differs. How many different shortlists, generated from lists whose descriptions contain only brackets and at most 4 fundamental lists, contain the value 99?

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 2*n* (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: *Drats*

The game of *drats* is played by throwing drats (which are small and dart-like) at a long board which is split into 20 adjacent segments, numbered from 1 to 20. The first drat thrown scores twice the value of the segment it lands in, and all subsequent drats score the value of the segment they land in.

Drats that miss the board are thrown again, so every drat must score some points. Several drats are allowed to land in the same segment.

For example, in a game with three drats, if the first drat lands in segment 4, the second in segment 10 and the third in segment 4, the total score is 8 + 10 + 4 = 22.

Some scores can be obtained in multiple ways. For example, in a game with three drats, the score 6 can be obtained in four ways: double-1 + 1 + 3, double-1 + 2 + 2, double-1 + 3 + 1 and double-2 + 1 + 1.

3(a) [25 marks]

Write a program to determine how many different ways a score can be obtained with a given number of drats.

Your program should input a single line containing two integers. The first integer s ($1 \le s \le 200$) will indicate the required score. The second integer d ($1 \le d \le 8$) will indicate the number of drats.

Sample run

```
7 3
6
```

3(b) [2 marks]

What is the next lowest score, after 2, that cannot be obtained with two drats? What is the next lowest score, after 3, that cannot be obtained with three drats?

3(c) [4 marks]

Two games of drats are played. In each game three drats are thrown. In how many different ways can the drats be thrown if the total score in both games is equal and all six drats land in different segments?

3(d) [4 marks]

On a regulation drat board the segments do not appear in numerical order. The segments 1 to 10 count as low-scores and the segments 11 to 20 count as high-scores. Is it possible for the segments to be arranged in such an order that high and low scores alternate and, at the same time, odd and even segments alternate? Justify your answer.

Total Marks: 100 End of BIO 2006 Round One paper

Question 3: ***String rewriting***

A *string rewriting rule* is a simple instruction for changing a string. It says that every occurrence of a given character should be changed into one or more other characters. In this question we will use five rules:

- `A` should be changed into `B`
- `B` should be changed into `AB`
- `C` should be changed into `CD`
- `D` should be changed into `DC`
- `E` should be changed into `EE`

In a single step, we apply these rules simultaneously and on every character in a string. For example, suppose we have the string `DECADE`, after one step we have the string `DCEECDBDCEE`, and after a second step we have the string `DCCDEEEECDDCABDCCDEEEE`.

3(a) [25 marks]

Write a program to determine the number of each type of character in part of a string after a given number of steps.

Your program should input two lines, the first containing a three letter string (where each letter will be `A`, `B`, `C`, `D` or `E`), and the second containing two integers s ($1 \leq s \leq 29$) followed by p ($1 \leq p \leq 2^{31}$). The first integer s indicating the number of rewriting steps which should take place and the second p indicating a position.

Sample run

```
DEC
2 10
0 0 3 3 4
```

You should output 5 numbers, the number of `A`s (then `B`s, `C`s, `D`s and `E`s) occurring in the first p (inclusive) positions of the string after s string rewriting steps.

The value p will never be larger than the number of characters in the string after s steps.

3(b) [2 marks]

How many letters will there be if you start from the string `C` and apply 8 steps? How about if you start from the string A?

3(c) [5 marks]

Given any starting position and number of steps (>1) is it ever possible to have three adjacent `C`s in the resulting string? Justify your answer.

3(d) [3 marks]

For some strings we can go backwards; i.e. we can find a previous string which leads to it after applying a step. With the rules given in this question, when we are able to move backwards there is always a *single* possible previous string.

For some sets of rules the situation might be ambiguous; when we try to move backwards there many be more than one possible previous string.

Find the smallest set of rules (both in terms of the number of rules and the number of characters used in those rules) which makes things ambiguous when moving backwards. Your rules must allow growth, so given any starting string it must be possible to apply enough steps to make the length of that string increase.

Total Marks: 100 End of BIO 2007 Round One paper

Question 3: *Shirts*

Seven shirts, with the numbers 1 to 7 embroidered on the back, are hanging on a washing line. The order of the shirts can be changed by four different operations; in each case removing one of the shirts, pushing three of the shirts along the washing line to make a gap, and replacing the removed shirt back on the washing line in the gap:

1. The leftmost shirt is temporarily removed and the adjacent three shirts pushed left.
2. The rightmost shirt is temporarily removed and the adjacent three shirts pushed right.
3. The middle shirt is temporarily removed and the leftmost three shirts pushed right.
4. The middle shirt is temporarily removed and the rightmost three shirts pushed left.

For example, suppose that the shirts were on the washing line in the order `1234567`. The first operation would change the order to `2341567`, the second would change it to `1237456`, the third to `4123567` and the fourth to `1235674`.

Given the starting order of the shirts we are interested in finding the smallest number of operations that are required to change the order to `1234567`. For example, suppose the starting order was `3452671`. The smallest number of operations is 3 (i.e. applying operation 3 then 2 then 3).

3(a) [24 marks]

Write a program to determine the smallest number of operations to change a given order to `1234567`.

Your program should input a single line containing a seven digit number. This represents the starting order of the shirts and will containing each of the digits 1 to 7 exactly once. You should output a single number; the smallest number of required operations.

It is possible to get to the order `1234567` from any starting position.

Sample run

```
6417352
5
```

3(b) [3 marks]

From a fixed starting order, how many different orderings can the shirts be in after exactly 2 operations? How many after exactly 6? *[NB: This question asks that a certain number of operations are performed, not that a certain number of operations are required.]*

3(c) [4 marks]

The hardest starting order is the one which requires the largest number of operations. How many operations does it require? Give a starting order which requires this many operations.

3(d) [4 marks]

Suppose that you were able to choose both the starting and finishing orders. Is it possible to select orders so that more operations are required than the hardest case when only the starting position is chosen? Justify your answer.

Total Marks: 100 End of BIO 2008 Round One paper

Question 3: ***Child's Play***

A toy set contains blocks showing the numbers from 1 to 9. There are plenty of blocks showing each number and blocks showing the same number are otherwise indistinguishable. We can consider the number of different ways of arranging the blocks so that the displayed numbers add up to a fixed sum.

For example, suppose the sum is 4. There are 8 different arrangements:

```
1 1 1 1
1 1 2
1 2 1
1 3
2 1 1
2 2
3 1
4
```

These have been listed in *dictionary* order. Just like in a dictionary, where all the words beginning with an 'a' come before those that begin with a 'b', and all those beginning 'aa' come before those beginning 'ab', we have listed the arrangements beginning with a 1 before those with a 2, and those beginning with a 1 then another 1, before those beginning with a 1 then a 2.

3(a) [23 marks]

Write a program to determine the n^{th} way of arranging the blocks to make a fixed sum.

Your program should input integers s ($1 \leq s \leq 32$) then n ($1 \leq n < 2^{31}$). You should output the n^{th} way of arranging the blocks, in dictionary order, to make the sum s.

You will not be given an n that is greater than the number of ways to arrange the blocks.

Sample run

```
4 5
2 1 1
```

3(b) [2 marks]

How many blocks in total need to be in the toy set to ensure that any of the arrangements summing up to 32 can be made? Blocks may only be used to show a single number; e.g. a block showing 6 cannot be turned upside-down to show a 9.

3(c) [5 marks]

Rather than listing the arrangements in dictionary order we could list them in *increasing numeric* order. In this case, the list for a sum of 4 would have been 4, 13, 22, 31, 112, 121, 211 and 1111. In general, what is the connection between the last entry in the dictionary list, and the first entry in the numeric list? Justify your answer.

3(d) [5 marks]

A *palindromic* arrangement of blocks is one where the numbers appear in the same order whether they are read from the left or from the right. E.g. 121 is a palindromic arrangement. How many palindromic arrangements sum to 8? Of the 540,142,091,243,007 different ways to arrange the blocks so that they sum to 50, how many are palindromic?

Total Marks: 100 End of BIO 2009 Round One paper

Question 3: *Juggl(ug)ing*

A cookery set contains several jugs of ***different*** sizes, each of which has a known capacity in oz (fluid ounces). There are no graduations on the jugs, so the only way to measure the liquid is by filling up a jug completely, pouring as much liquid as possible from one jug into another or by emptying a jug. A *step* consists of performing one of these three operations.

Initially all the jugs are empty. To measure *n* oz it is necessary to perform a sequence of steps and finish with exactly *n* oz in one of the jugs.

For example, suppose we have two jugs: jug A holds 3 oz and jug B holds 5 oz.

- If we fill up jug B and then pour as much as possible from jug B into jug A, we would have (after 2 steps) 3 oz in jug A and 2 oz in jug B. This is one way to measure 2 oz.
- If we now empty jug A, pour the 2 oz from jug B into jug A, fill jug B and finally pour as much as possible from jug B into jug A, we would finish (after 6 steps) with 3 oz in jug A and 4 oz in jug B. We have now measured 4 oz.

3(a) [23 marks]

Write a program that, given the capacities of several jugs, determines the shortest number of steps necessary to measure *n* oz.

The input will consist of two lines. The first line will contain two integers *j* ($1 \le j \le 3$) then *n* ($1 \le n \le 250$), indicating the number of jugs and the required amount to measure respectively. The second line will contain *j* integers, each between 1 and 250 inclusive, indicating the capacity of the jugs.

You should output a single integer giving the minimum number of steps necessary to measure *n* oz.

Sample run

```
2 4
3 5
6
```

Your program will only be asked to measure amounts that are possible.

3(b) [2 marks]

Given two jugs A and B, whose capacities are 3 oz and 8 oz respectively, show how to measure 2 oz in 4 steps.

3(c) [5 marks]

Two cookery sets that can measure the same values and no others are said to be *equivalent*. For example, the pair of jugs of capacity 5 oz and 9 oz, and the pair 2 oz and 9 oz can both measure every value from 1 oz to 9 oz. Ever cookery set is equivalent to itself and the order of the jugs in a set does not matter.

Consider the cookery set containing three jugs whose capacities are 6 oz, 18 oz and 20 oz. How many equivalent cookery sets are there that contain two jugs? How many are there that contain three jugs?

3(d) [5 marks]

Is it possible to find a cookery set, containing no 1oz capacity jugs, with a sequence of steps that leaves 1oz in all the jugs simultaneously? Justify your answer.

Total Marks: 100 End of BIO 2010 Round One paper

Question 3: ***Number Ladder***

A number can be transformed to another number if the letters in the digits of the first number are the same as those in the second number with at most 5 letters (in total) being added or deleted. Individual digits are spelt ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE and ZERO.

For example:

- 61 can be transformed to 26 since the letters SIXONE, with 4 changes (N and E deleted, and T and W added), can then be arranged to TWOSIX. Note that the order of the letters does not matter.

A *number ladder* is a sequence of transformations that changes one number to a *different* number. Between any two adjacent numbers on the ladder a valid transformation takes place. No number appears more than once on a ladder. The *size* of a ladder is the number of transformations that takes place.

For example:

- 1, 90, 610 is a number ladder (of size 2) since 1 can be transformed to 90 and 90 can be transformed to 610. There is no smaller number ladder that starts with 1 and finishes with 610 since 1 cannot be transformed to 610.

3(a) [23 marks]

Write a program to determine the size of smallest number ladders.

Your program should read in three lines, each containing two integers s then f, ($1 \leq s < f \leq 999$). You should output three numbers (one per line), the i^{th} of which giving the size of the smallest number ladder (that only uses numbers between 1 and 999 inclusive) going from s to f on the i^{th} line of the input.

Sample run

```
26 61
1 90
1 610
1
1
2
```

You will only score points if all three numbers in your output are correct.

3(b) [2 marks]

How many integers, excluding 0, can be transformed (directly) to ZERO?

3(c) [4 marks]

Suppose that, for each possible start and finish (between 1 and 999 inclusive) we have calculated the smallest possible number ladder. What is the largest size of these ladders? How many ladders have this size?

3(d) [6 marks]

Does a number ladder exist to change any integer, however large, into any other integer with the same number of digits? Justify your answer.

Total Marks: 100 End of BIO 2012 Round One paper

Question 3: ***Unlock***

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V	W	X	Y

A security system consists of a 5×5 keypad of buttons. Each button has a light which can be either *off*, *dim* or *bright*. When a button is pressed its light moves to the next state (*off*→*dim*, *dim*→*bright* and *bright*→*off*) as do the lights on any buttons touching it horizontally and vertically. The system is *unlocked* when all the lights are *off*.

The current lighting can be represented by a string, in alphabetical order, where a lowercase letter indicates the corresponding button is *dim* and an uppercase letter indicates it is *bright*. Lights that are *off* do not have a letter in the string.

For example, if all the buttons are *off* and button R is pressed, the lighting is `mqrsw`. If T is now pressed it becomes `moqrStwy`. Pressing S will make it `mnoqRTwxy`.

3(a) [24 marks]

Write a program to determine a sequence of button presses to *unlock* the system.

Your program should read in a string of between 1 and 12 *distinct* letters (inclusive) in alphabetical order, representing the current lighting of the security system.

You should output a string (in alphabetical order) that indicates a sequence of buttons which can be pressed to unlock the system, or output `IMPOSSIBLE` if it cannot be unlocked. A button can be pressed at most twice; a lowercase letter in your string will indicate that the corresponding button should be pressed once and an uppercase letter indicating that it should be pressed twice.

If there are multiple solutions you are only required to print out a single solution.

Sample run

```
mnoqRTwxy
RST
```

Alternative answer

```
mnoqRTwxy
aCeFhJprSUwY
```

3(b) [2 marks]

What is the lighting of the system if the system is unlocked and then B is pressed, followed by I and O? (Each button being pressed once.)

3(c) [4 marks]

Starting from an unlocked system, in how many ways can three different buttons be pressed once each, in alphabetical order, so that no light becomes bright?

3(d) [5 marks]

Suppose that the system has a configuration of lighting and that a particular sequence of button presses will unlock the system. Is it possible for the same sequence of button presses to unlock the system when it is in a different configuration? Justify your answer.

Total Marks: 100 End of BIO 2013 Round One paper

Question 3: *Increasing Passwords*

A password scheme accepts passwords that contain combinations of upper-case letters and digits. The scheme restricts passwords to ones in which letters appear in alphabetical order. The digits 0, ..., 9 are treated as coming after Z in alphabetical order; lower digits are treated as coming before higher digits alphabetically. Passwords must contain at least one character (letter or digit) and no duplicate characters are allowed.

For example:
- `BIO14` is a valid password;
- `OLYMPIAD` is not a valid password (letters are not in alphabetical order).

The passwords have been put into an ordered list. If two passwords contain a different number of characters, the password with the fewer characters comes first. If they both have the same number of characters then they are sorted alphabetically.

The ordered list of passwords looks like this: `A`, `B`, ..., `Z`, `0`, `1`, ... `9`, `AB`, `AC`, ..., `A9`, `BC`, ...

3(a) [25 marks]

Write a program to determine the *n*th password in the ordered list.

Your program should read in a single integer, ($1 \leq n \leq 1{,}000{,}000{,}000$). You should output the string which represents the *n*th password.

Sample run

```
37
AB
```

3(b) [2 marks]

In what order do the following passwords appear in the list: `BIO`, `BIO14`, `NTU`, `14`, `ABCDE` ?

3(c) [3 marks]

How many passwords does the scheme accept?

3(d) [5 marks]

How many *pairs* of adjacent passwords (in the ordered list) contain all 36 characters between them, with each character appearing in only one of the passwords? Justify your answer.

Total Marks: 100 End of BIO 2014 Round One paper

Question 3: *Modern Art*

A gallery is displaying pieces of *modern art* by several artists and is considering the different ways of arranging the exhibition. As this is modern art all pieces by the same artist are indistinguishable.

For example, suppose there is a single piece by artist `A`, two by artist `B` and one by artist `C`. There are 12 ways the gallery might arrange the exhibition:

```
ABBC
ABCB
ACBB
BABC
BACB
BBAC
BBCA
BCAB
BCBA
CABB
CBAB
CBBA
```

These have been listed in *alphabetical* order.

3(a) [25 marks]

Write a program to determine the n^{th} way of arranging the exhibition.

Your program should input five integers: *a, b, c* and *d* (each between 0 and 5 inclusive) indicating the number of works by artists `A`, `B`, `C` and `D` in order, and finally n ($1 \leq n \leq 2^{34}$).

You will only be given input where at least one artist is exhibiting a work and *n* is no greater than the number of possible exhibitions.

You should output the string which represents the n^{th} arrangement.

Sample run

```
1 2 1 0 8
BCAB
```

3(b) [2 marks]

If the gallery is exhibiting `AABCCBDD` which arrangement is this?

3(c) [5 marks]

An unspecified number of artists are exhibiting an unspecified number of pieces of modern art. The gallery has exhibited the n^{th} arrangement followed by the $n+1^{st}$ arrangement. For poetic reasons, for each position in the n^{th} arrangement the artist providing the work in the same position in the $n+1^{st}$ arrangement was different. Is it possible that, in the $n+2^{nd}$ arrangement, the artist providing the work in each position is different to that in the $n+1^{st}$ arrangement? Justify your answer.

Total Marks: 100 End of BIO 2015 Round One paper

Question 3: ***Prime Connections***

A *prime number* is a whole number, greater than 1, that can only be divided by itself and the number 1. Two prime numbers are *connected* if the difference between them is 2^n for some whole number $n \geq 0$; e.g. possible differences are 1, 2, 4, 8, 16, 32, …

A *path* is a sequence of (at least two) prime numbers, without repetition, where adjacent numbers in the sequence are *connected*. If the first number in the sequence is p and the last number is q then we say the path is *between* p and q.

The *length* of a path is the total number of prime numbers used. There may be multiple paths between two prime numbers; the lengths of these paths may be different.

For example:

- 13 is connected to 5 ($13 - 5 = 8 = 2^3$), 5 is connected to 3 ($5 - 3 = 2 = 2^1$) and 3 is connected to 2 ($3 - 2 = 1 = 2^0$);
- As 13 and 5 are connected there is a path between them (13—5) whose length is 2;
- There is a path from 13 to 2 (13—5—3—2) whose length is 4;
- There is a longer path from 13 to 2 (13—17—19—3—2) whose length is 5.

You will be given an upper limit on the primes you are allowed to use. For example, if the limit was 18 then the path 13—17—19—3—2 would *not* be permitted as it includes a prime above this limit.

3(a) [27 marks]

Write a program to determine the length of the *shortest* path between two primes.

Your program should input three integers in order: l ($4 \leq l \leq 2^{24}$) indicating the highest value you are allowed to use, followed by the primes p then q ($2 \leq p < q < l$). You will only be given input where there is a path between p and q using values below l.

You should output the length of the shortest path.

Sample run

```
100 2 13
4
```

3(b) [2 marks]

How many different *paths* are there between 2 and 19 with a upper limit of 20?

3(c) [3 marks]

How many pairs of *connected* primes are there with an upper limit of 250,000? (Reversing the order of the primes does *not* count as a different pair.)

3(d) [3 marks]

Suppose that there are two (different) paths of length n between p and q, and that both of these paths contain exactly the same primes. What can you say about the length of the shortest path between p and q?

Total Marks: 100 End of BIO 2016 Round One paper

Question 3: ***Mystery Parcel***

A shop is running a promotion and giving away *mystery parcels*, containing one or more items. As these are promotional, items are not very exciting (items that weigh the same are indistinguishable) and the total weight of items in each parcel is the same. Parcels containing the same combination of items (order does not matter) are themselves indistinguishable.

For advertising purposes the shop wishes to calculate the number of ways it can distribute the parcels.

For example, suppose there are 2 parcels containing a total of 4 items, each item weighing 1, 2 or 3 units. There are 10 different ways the parcels can be distributed. They might be constructed in 8 ways:

```
 1 1    —    1 1
 1 2    —    1 2
 2 2    —    2 2
 1 3    —    1 3
 2 3    —    2 3
 3 3    —    3 3
 3 1    —    2 2
1 1 1   —     3
```

In each of the first 6 pairings the parcels are indistinguishable, so there is only a single way in which they can be distributed. In each of the last 2 pairings the parcels can be distinguished, so they could each be distributed in 2 different ways; i.e. it can be distinguished which is distributed first.

3(a) [25 marks]

Write a program to determine the number of ways parcels can be distributed.

Your program should input four integers in order: p ($1 \le p \le 5$) indicating the number of parcels, i ($1 \le i \le 10$) indicating that items can weigh any integer from 1 to i inclusive, n ($1 \le n \le 25$) indicating the total number of items in all the parcels, and w ($1 \le w \le 25$) indicating the weight of each parcel.

You should output the number of ways parcels can be distributed. You will not be given input that requires an answer greater than 2^{31}.

Sample run

```
2 3 4 3
3
```

Marks are available for the case where *p* = 1

3(b) [2 marks]

How many ways can 3 parcels, containing a total of 6 items weighing 1, 2 or 3 units, be distributed?

3(c) [6 marks]

There are 92378 distinguishable parcels containing 10 items (when $i = 10$) and the shop has created an index of these parcels. Given two parcels, the one with the largest number of weight-10 items appears earliest in the index. If those are equal, the order depends on the number of weight-9 items, and so on. E.g. `8888855555` is before `8888855554` which is before `8888777777`.

What is the contents of the parcel at position 50,000? What is the position of the parcel containing one item of each weight from 1 to 10?

Total Marks: 100 End of BIO 2017 Round One paper

Question 3: ***Serial Numbers***

A *serial number* can be transformed into another, by swapping two adjacent digits, so long as one of those two digits is adjacent to a digit whose value lies between them. Two *serial numbers* will be called *equivalent* if there is a sequence of transformations which changes one into the other, and the *distance* between them is the *minimum* number of such transformations required.

For example, the following shows all the serial numbers that are equivalent to `146235`, the lines showing all the valid transformations:

$$\begin{tikzcd}
 & & & 146235 \arrow[d, dash] & & & \\
 & & & 142635 \arrow[dl, dash] \arrow[dr, dash] & & & \\
461235 \arrow[r, dash] & 416235 \arrow[r, dash] & 412635 \arrow[dr, dash] & & 142365 \arrow[r, dash] \arrow[dl, dash] & 124365 \arrow[r, dash] & 124635 \\
 & & & 412365 & & &
\end{tikzcd}$$

The distance from `461235` to `124635` is 6, and from `412635` to `142635` is 1.

3(a) [24 marks]

Write a program that determines the largest distance between a *serial number* and any equivalent number.

Your program should input a single integer *d* (1 ≤ *d* ≤ 9) indicating the number of digits in the serial number, followed by a *d* digit serial number which will consist of the numbers 1,...,*d* without repetition.

You should output a single integer giving the largest distance.

Sample run

```
6
461235
6
```

3(b) [4 marks]

What is the largest distance between *any* two serial numbers equivalent to 326451? How about 183654792?

3(c) [6 marks]

What is the size of the largest set of serial numbers, none of which are equivalent, each consisting of the digits 1,...,5 without repetition? What is the size if the serial numbers consist of the digits 1,...,9 without repetition?

Total Marks: 100 End of BIO 2018 Round One paper

Question 3: *Block-chain*

A set of children's blocks, each illustrated with a single different letter, have been chained together in a line. They have been arranged so that it is not possible to find three (not necessarily adjacent) letters, from left to right, that are in alphabetical order.

For example, if there are four blocks (A, B, C and D) the possible block-chains are:

```
ADCB    BADC    BDAC    BDCA    CADB
CBAD    CBDA    CDAB    CDBA    DACB
DBAC    DBCA    DCAB    DCBA
```

3(a) [24 marks]

Write a program that enumerates block-chains.

Your program should input a single integer *l* ($1 \leq l \leq 19$) indicating that the blocks are illustrated with the first *l* letters of the alphabet, followed by a word *p* of between 1 and *l* uppercase letters indicating (in order) the leftmost letters of the block chain. *p* will only contain letters take from the first *l* letters of the alphabet and will not contain any duplicates.

You should output a single integer giving the number of possible block-chains that begin with *p*.

Sample run

```
4 CB
2
```

3(b) [2 marks]

List the valid block-chains containing the blocks `B`, `I` and `O`.

3(c) [4 marks]

A block-chain containing the first *n* ($1 \leq n \leq 13$) letters of the alphabet is attached to the left of a block-chain containing the last *m* ($1 \leq m \leq 13$) letters of the alphabet, forming a new block-chain. What were the original block-chains? Justify your answer.

3(d) [5 marks]

Suppose all the valid block-chains containing the first 19 letters of the alphabet are sorted into alphabetical order. Which one comes first? Which one comes 1,000,000,000th?

Total Marks: 100 End of BIO 2019 Round One paper

Question 3: *False Plan*

A spy, currently working their way through an enemy compound, has been given a *false plan*. The plan is an ordered list of letters. In order to make the plan look realistic, the number of adjacent identical letters in the plan has been limited.

For example, suppose that the plan contains four letters, each of which is an `A` or a `B`, and that there are never more than two adjacent identical letters. There are 10 possible plans:

```
AABA
AABB
ABAA
ABAB
ABBA
BAAB
BABA
BABB
BBAA
BBAB
```

These have been listed in *alphabetical* order.

3(a) [27 marks]

Write a program to determine the n^{th} possible plan.

Your program should input a line containing three integers p, q, r ($1 \leq p, q, r \leq 12$) indicating (in order) that the first p letters of the alphabet can be used, no more than q adjacent identical letters are permitted and that the plan should contain exactly r letters. You should then input a line containing a single number n ($1 \leq n < 2^{63}$).

You will only be given input where n is no greater than the number of possible plans.

You should output the n^{th} possible plan.

Sample run

```
2 2 4
7
BABA
```

3(b) [3 marks]

Suppose `A`s, `B`s, `C`s and `D`s are permitted and there are never more than two adjacent identical letters. Consider the list of possible three letter plans; which plan is `CCA` within this list? How about (for eight letter plans) `CCABABCC`?

3(c) [5 marks]

Suppose there is a plan that is in the n^{th} position both when the plans are ordered alphabetically and when they are ordered reverse alphabetically. What can you determine about p, q, and r? Justify your answer.

Total Marks: 100 End of BIO 2020 Round One paper

Question 3: *Window Dressing*

A shop is looking to display some boxes (conveniently labelled `A`, `B`, …) in their window. There is a desired order for the boxes to appear in the window but unfortunately the boxes arrive from the warehouse in alphabetical order and access to the window limits how the boxes can be added to the display.

There are three possible operations the store manager can employ to change the order of the boxes in the window:

- (Add) The next box from the warehouse can be added to the end of the boxes currently on display;
- (Swap) The first two boxes in the window display can be swapped;
- (Rotate) The first box can be removed from the display and replaced at the other end of the display.

Given a desired order for the display, the manager wishes to find the minimum number of operations needed.

For example, suppose the desired order is ACBD:

- The manager could add A then B then C then D (giving ABCD after 4 operations), then rotate (giving BCDA), swap (CBDA) and rotate another 3 times (ACBD). A total of 9 operations.
- The manager could add A then B (giving AB), then swap (BA), add C (BAC), rotate (ACB) and add D (ACBD). A total of 6 operations.

3(a) [24 marks]

Write a program to determine the minimum number of operations to display the boxes.

Your program should input a string of *b* ($1 \le b \le 8$) uppercase letters, a permutation of the first *b* letters of the alphabet.

You should output the minimum number of operations required to reach this permutation.

Sample run

```
ACBD
6
```

3(b) [3 marks]

Which orderings of the 5 boxes `A`, …, `E` require a minimum of 6 operations?

3(c) [5 marks]

The arrangement HGFEDCBA takes a minimum of 24 operations. How many different sequences of 24 operations lead to this arrangement?

Total Marks: 100 End of BIO 2021 Round One paper

Question 3: ***Parking***

A one-way street, running left to right, has several parking spaces each large enough for a single vehicle. Each arriving car has a preferred parking spot and is driven along the street, ignoring empty spaces, until it reaches its preferred spot. If that spot is available it is parked there, otherwise it continues along the street and is parked in the first unoccupied space. If all those spaces are occupied it leaves the street and does not park anywhere.

Spaces are labelled `A`, `B`, … in the order they are encountered. Cars are labelled `a`, `b`, … in order of their arrival. There are the same number of cars as parking spaces and all the spaces are initially empty.

A *preference list* shows the preferred parking spaces for the cars (in order of their arrival).

For example, suppose after each car has arrived they are parked in the order `cabd`:

- Car `a`'s preferred space was `B`. When it arrived it drove past space `A` to space `B`, found it empty and parked.
- Car `b` might have a preferred space of `C` (driving past spaces `A` and `B` on arrival, finding space `C` empty and parking) but it might also have a preferred space of `B` (driving past `A` on arrival, finding `B` full and parking at the next available space `C`).
- Car `c`'s preferred space must be `A`.
- Car `d` might prefer any of the spaces. If their preferred space is `A`, `B` or `C` when they arrive they find it full and the first available space, in each case, will be `D`.
- There are eight difference preference lists for the cars: `BBAA`, `BBAB`, `BBAC`, `BBAD`, `BCAA`, `BCAB`, `BCAC` and `BCAD`. These have been listed in *alphabetical* order.

3(a) [25 marks]

Write a program to determine the *i*th preference list for a given final arrangement of cars *where all the cars have managed to park*.

Your program should input a string of *n* ($1 \leq n \leq 16$) lowercase letters (a permutation of the first *n* letters of the alphabet) giving the final arrangement of the cars, followed by an integer *i* ($1 \leq i \leq 2^{63}$).

You should output the *i*th preference list for the cars that will lead to the final arrangement.

Sample run

```
cabd 5
BCAA
```

3(b) [2 marks]

Suppose the preference list is `EECCGGAAAA`. What is the final arrangement of the cars?

3(c) [3 marks]

Consider the cases where there are 16 cars, which all park, and exactly two possible preference lists lead to the final arrangement. How many arrangements are there?

3(d) [4 marks]

The final arrangement for 16 cars is `abcdefghijklmnop` and exactly 3 cars are in their preferred position. How many potential preference lists are there?

Total Marks: 100 End of BIO 2022 Round One paper

Question 3: Dreaming Spires

In the *Tower of Oxford* game there are four pegs and four balls (numbered 1, 2, 3, 4) which can be stacked on the pegs. Each peg has a *height* of four, i.e. it can support a stack of all four balls.

We will denote a stack, running from the bottom to the top, by a sequence of digits running left (bottom) to right (top). E.g. `1234` is the stack with the balls in order and `1` on the bottom. An empty stack will be denoted by the digit `0`. The current *state* of the game will be shown by giving each peg's stack.

A valid move in the game consists of moving the top ball from a peg's stack and placing it on top of a (possibly empty) stack on another peg. Given a start position, the object of the game is to make the minimum number of moves to get to a given end position.

For example:

- Suppose the starting configuration has a single ball on each peg: `1 2 3 4`. There are 12 possible moves as each ball is on top of a stack and can be moved to any of the other pegs.
- The balls on the left-most two pegs could be swapped over by the following sequence of moves: `1 2 3 4` → `0 2 31 4` → `2 0 31 4` → `2 1 3 4`; i.e. by stacking ball `1` on top of ball `3`, then moving ball `2` to its final position ahead of ball `1` being moved.
- We could move from `1 2 3 4` to `2 3 4 1` using an equivalent swap sequence several times to swap `1` & `2`, then `3` & `1`, then `4` & `1`. This takes 9 moves.
- `1 2 3 4` → `12 0 3 4` → `12 3 0 4` → `12 3 4 0` → `1 32 4 0` → `0 32 4 1` → `2 3 4 1` requires only 6 moves.

3(a) [23 marks]

Write a program to determine the minimum number of moves required to get between two game states.

Your program should first input four numbers showing the initial state of the game. The next line of input will contain another four numbers giving the end state of the game.

You should output the minimum number of moves required to get from the start to the end state. It is possible to get between any two game states.

Sample run

```
12 0 3 4
1 32 4 0
3
```

3(b) [2 marks]

How many game states have 9 moves available?

3(c) [5 marks]

Suppose the starting configuration is `1 2 3 4`. How many different states can the game be in after 2 moves? How about after 4 moves?

3(d) [4 marks]

Consider a variant of the game where there are 8 pegs, 8 identical balls, and each peg has a height between one and eight inclusive. If there are 2023 different states of the game what are the heights of the pegs?

Total Marks: 100 End of BIO 2023 Round One paper

Question 3: Word Game

In a *word game* letters are scored according to their position in the alphabet (A=1, …, Z=26) and a word's score is the sum of scores of its letters. Any combination of letters is allowed in a word, so long as there are no adjacent copies of the same letter.

Given a word, we are interested in finding its position in the alphabetically ordered list of words with the same score.

For example, there are 7 words which score 5. The word **BAB** is the 3rd word in the list, which is:

```
ACA
AD
BAB
BC
CB
DA
E
```

3(a) [25 marks]

Write a program to determine the position of a word in its list.

Your program should first input a string of 1 to 12 uppercase letters. The score of this string will be ≤ 75.

You should output the position of this string in the alphabetically ordered list of words with the same score.

Sample run

```
BAB
3
```

3(b) [2 marks]

What is the first word with the same score as ACE?

3(c) [4 marks]

How many words have a score of 26?

3(d) [5 marks]

What is the largest difference in word length between adjacent words in the list for a score of 54? How many adjacent pairs of words have this difference?

Total Marks: 100

End of BIO 2024 Round One paper

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

Question 3: ***Upside-Down***

An *upside-down* number is an integer where the *i*th digit from the left plus the *i*th digit from the right is always equal to 10. For example 13579 is an upside-down number since 1+9 = 10, 3+7 = 10 and (since 5 is both the 3rd digit from the left and from the right) 5+5 = 10.

The first few upside-down numbers, in numerical order, are 5, 19, 28, 37, … , 82, 91, 159, …

3(a) [24 marks]

Write a program to determine the *n*th upside-down number (in numerical order).

The input will consist of a single integer n ($1 \le n \le 2^{31}$). You should output a single integer giving the *n*th upside-down number.

Sample run

```
11
159
```

3(b) [2 marks]

Consider all the different 9 digit numbers that use each of the digits 1, … , 9 once each. How many of these are upside-down numbers?

3(c) [3 marks]

How many digits are in the 1,000,000,000,000,000,000th upside-down number?

3(d) [6 marks]

Are there more upside-down numbers with 1000 digits that contain at least one 5, or more upside-down numbers with 1001 digits that contain at least one 5? Justify your answer.

Total Marks: 100 End of BIO 2011 Round One paper

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type in two words (in capitals). Students' programs will *probably* expect the words to be on separate lines - you may enter words separated by a space or comma if required by a student's program. The correct response is given to the right of each pair. There are no marks for incorrect answers.

[1]	**OLYMPIAD** **OLYMPIAD**	`Anagrams`
[2]	**LEMON** **MELON**	`Anagrams`
[2]	**COVERSLIP** **SLIPCOVER**	`Anagrams`
[2]	**TEARDROP** **PREDATOR**	`Anagrams`
[2]	**ABBCCCDDD** **DDDCCCBBA**	`Anagrams`
[1]	**I** **A**	`Not anagrams`
[2]	**FORTY** **FORT**	`Not anagrams`
[2]	**ONE** **SIX**	`Not anagrams`
[2]	**GREEN** **RANGE**	`Not anagrams`
[2]	**ABBCCCDDD** **AAABBBCCD**	`Not anagrams`

Additional marks are available for general program behaviour:

[2] Program inputs two words
[2] For each test, either `Anagrams` or `Not anagrams` is output
[2] Program terminates with crashing / hanging

Question 1(b) [2 marks available]

[2] 6

Question 1(c) [4 marks available]

[4] 21

(**Supplementary:** 6 is worth [2] marks.)

Question 2(a) [24 marks available]

For each test of the program for 2(a) you need to type in three lines of input. The first line will be a combination of the symbols **x**, **u**, **d**, *****, **?**, **(** and **)**. The next two lines will be integers, which may have leading **0**s.

Output should consist of two lines, each of which will be either `Yes` or `No`. *Both* lines of output must be correct for marks to be scored. Each test is worth [2] marks.

Test 1

```
     x
     7
     12
[2]  Yes
     No
```

Test 2

```
     xux
     667
     65
[2]  No
     No
```

Test 3

```
     xddu
     8654
     2102
[2]  No
     Yes
```

Test 4

```
     xuxd
     9801
     4543
[2]  No
     Yes
```

Test 5

```
     xx*
     0
     2006
[2]  Yes
     Yes
```

Test 6

```
     xd*u?
     56
     54322
[2]  Yes
     No
```

Test 7

```
     x?x
     1
     11
[2]  Yes
     Yes
```

Test 8

```
     x?x
     123
     4567
[2]  No
     No
```

Test 9

```
     x*xu
     7654
     2
[2]  No
     No
```

Test 10

```
     x(xud)?
     789
     7890
[2]  No
     Yes
```

Test 11

```
     x(xd)?(xu)?
     421
     422
[2]  Yes
     No
```

Test 12

```
     x(ud)*u?
     0836
     19181716151
[2]  Yes
     Yes
```

Question 2(b) [3 marks available]

[3] 46010

(**Supplementary:** 46460 is worth [1] mark.)

Question 2(c) [3 marks available]

[3] `x(ud)*u?`

Question 2(d) [5 marks available]

[1] No

[1] 9 successive `u`s (or `d`s) after an `x` are required to match a unique password.

Up to [1] mark is available for some justification for the above point. Suitable justifications are:

• The only symbols that restrict the choices of digit are `u` and `d`.
• An example of how a shorter successive sequence leads to multiple passwords.
• An example of how a mixed sequence leads to multiple passwords.

[1] Adding a `u` (in the case of successive `u`s) leads to zero valid passwords.

Up to [1] from either:

[1] Adding a `*` or `?` to one of the above sequences leads to an increased number of valid passwords.
[1] Adding an `x` or `u` (in the case successive `d`s) leads to an increased number of valid passwords. (Alternatively for `d` in the case of successive `u`s).

Question 3(a) [25 marks available]

Each test for 3(a) consists of two integers for input and a single integer for output. There are no marks for incorrect answers.

[1]	**7 3**	6
[2]	**33 1**	0
[2]	**101 4**	0
[2]	**8 7**	1
[2]	**28 1**	1
[2]	**18 2**	8
[2]	**9 4**	22
[2]	**36 3**	191
[2]	**33 4**	2075
[2]	**83 5**	40000
[3]	**73 6**	1402584
[3]	**95 8**	515725220

Question 3(b) [2 marks available]

[1] 61 (after 2 drats)
[1] 81 (after 3 drats)

Question 3(c) [4 marks available]

[4] 510176

Question 3(d) [4 marks available]

[1] No

[1] High & even segments could only be next to low & odd segments and vice-versa. (Similarly for high & odd / low & even).

[1] A board which alternates high & even with low & odd segments cannot have high & odd or low & even segments.

[1] The segments 1 to 20 include high & even, high & odd, low & even and low & odd segments.

End of BIO 2006 marks scheme

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type in five integers. Students' programs will *probably* expect the numbers to be on the same line; you may enter words separated by a new-line or comma if required by a student's program. The correct response is given to the right of each pair. There are no marks for incorrect answers.

[1]	**3 3 3 2 10**	6
[2]	**3 4 7 9 10**	0
[2]	**7 9 2 2 10**	1
[2]	**2 7 9 10 2**	1
[2]	**2 2 3 3 4**	2
[2]	**7 6 6 6 10**	3
[2]	**1 6 2 4 9**	2
[2]	**1 8 2 3 9**	1
[2]	**2 2 7 8 2**	4
[2]	**5 5 10 5 5**	14

Additional marks are available for general program behaviour:

[2]	Program inputs five numbers
[1]	For each test a single number is output
[2]	Program terminates without crashing / hanging

Question 1(b) [2 marks available]

There are eight possible answers and only a single answer is required. Award [2] marks for any of the following sets of five numbers, where the numbers in the set can appear in any order:

1 2 7 9 10
or
1 2 8 9 10
or
1 3 6 7 10
or
1 3 7 9 10
or
1 3 8 9 10
or
2 4 6 8 10
or
3 4 6 7 10
or
3 4 7 9 10

Question 1(c) [4 marks available]

[4] 28

(**Supplementary:** 29 is worth [3] marks.)

Question 2(a) [24 marks available]

There are seven multiple part tests used to check program 2(a). Marks are given within the tests, besides the expected output from the program; this will be a 9 character string consisting of four **X**s, four **O**s and an **E**, or a single line of text.

Incorrect output at any stage gets no marks for that stage. In a 9 character string every character must be correct.

If the program crashes / hangs (appears to get stuck) part way through a test the rest of that test should be discarded.

Test 1

```
      XXEXOOOXO
      n
[1]   XXEXOOOXO
[1]   Player 2 wins
```

Test 2

```
      EOOOOXXXX
      n
[1]   OEOOOXXXX
      r
[1]   OXOEOXXXO
[1]   Player 1 wins
```

Test 3

```
      OEXXOOOXX
      n
[1]   EOXXOOOXX
      n
[1]   XOEXOOOXX
      n
[1]   XEOXOOOXX
      n
[1]   XXOXOOOXE
[1]   Player 2 wins
```

Test 4

```
      EXXXXOOOO
      n
[1]   OXXXXEOOO
      n
[1]   OXXXEXOOO
      r
[2]   Draw
```

Test 5

```
    XEOXXXOOO
    n
[2] XOOXXXOOE
    r
[1] OXOXXXOEO
[1] Player 1 wins
```

Test 6

```
    OOEXOOXXX
    r
[2] XOXOOOXEX
[1] Player 2 wins
```

Test 7

```
    EXOXOXXOO
    r
[3] Draw
```

Question 2(b) [3 marks available]

To score the first [2] marks both of the following board layouts must be given, but *no* additional ones:

```
[2] XOEXXXOOO
    XOOXXXOOE
```

(**Supplementary:** If the student gives both layouts plus any additional layouts, or lists one of these layouts and no others, award [1] mark.)

[1] The strategy gives `XOOXXXOOE`

Question 2(c) [3 marks available]

[1] No

[1] If a player has a marker in the putahi (central position) that player will be able to move.

In addition, up to [1] mark can be gained from either of the following points:

[1] If neither player has a marker in the putahi both players must have a marker adjacent to a marker of the opposing player and be able to move.
[1] All the losing positions have a marker in the putahi.

Question 2(d) [5 marks available]

[5] 46

(**Supplementary:** If the student has the total incorrect there are up to [4] marks for calculating the following: [2] for identifying 8 different layouts when the putahi is empty; [2] for identifying 19 different layouts when the putahi contains one of the first (or second) player's markers)

Question 3(a) [25 marks available]

Each test for 3(a) consists of a three letter string (capital letters from A, B, C, D and E), followed by two numbers on a separate line. The output should consist of five integers, all of which need to be correct for the test to score marks.

Tests must complete in **2 seconds** to score marks.

```
     DEC
     2 10
[1]  0 0 3 3 4

     BDA
     1 5
[1]  1 2 1 1 0

     ABA
     5 29
[2]  11 18 0 0 0

     EEE
     10 3072
[2]  0 0 0 0 3072

     CDD
     12 12001
[2]  0 0 6000 6001 0

     CDC
     12 12001
[2]  0 0 6001 6000 0

     ACE
     15 987
[2]  377 610 0 0 0

     ACE
     15 17371
[2]  377 610 8192 8192 0

     ACE
     15 66523
[2]  377 610 16384 16384 32768

     DAE
     29 1
[1]  0 0 0 1 0

     EEE
     29 1000000000                (nine 0s)
[2]  0 0 0 0 1000000000           (nine 0s)

     BAB
     29 1664080
[2]  635622 1028458 0 0 0

     BED
     29 1000000000                (nine 0s)
[4]  514229 832040 230891410 230891409 536870912
```

Question 3(b) [2 marks available]

[1]	256	(starting with the string C)
[1]	34	(starting with the string A)

Question 3(c) [5 marks available]

[1] No

[1] Every C must have come from a C or D at the previous step.

[1] After each step, each C is transformed into a C followed by a D. Similarly, each D is transformed into a C preceded by a D.

[1] Every C will be followed or preceded by a D.

[1] If there are two adjacent Cs the first must be preceded by a D and the second must be followed by a D.

Question 3(d) [3 marks available]

Either of the following pairs of rules is sufficient, where A and B can be any two letters so long as they are different, and X can be either of the two previous letters or a different letter.

[3] A should be changed to XX
B should be changed to X

or

A should be changed to X
B should be changed to XX

End of BIO 2007 marks scheme

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type a single integer. The response should be a single integer; the correct responses are given on the right. There are no marks for incorrect answers.

[1]	**22**	3
[2]	**4**	1
[2]	**8**	1
[2]	**62**	3
[2]	**114**	10
[2]	**346**	9
[2]	**1000**	28
[2]	**2326**	35
[2]	**5000**	76
[2]	**9240**	329

Additional marks are available for general program behaviour:

[2] Program inputs a single number
[2] For each test a single number is output
[2] Program terminates without crashing / hanging

Question 1(b) [3 marks available]

There is [1] mark for the following pair of numbers:

[1] 23 23

There are [2] marks for getting all three of the following pairs (numbers can be given in any order). Award *only* [1] mark if the student is missing any of the pairs, or has written any additional pairs (other than 23 23).

[2] 3 43
5 41
17 29

Question 1(c) [2 marks available]

[2] 9

Question 2(a) [25 marks available]

There are nine tests used to check program 2(a). For each test you will need to type in two lines of input, the first containing an integer and the second a string of *uppercase* letters. The correct responses are given on the right.

There are no marks for incorrect answers and tests *must* complete within 2 seconds.

[2]	**14** **AAABBB**	DBBDAD
[2]	**0** **A**	B
[2]	**0** **C**	D
[3]	**0** **AAAA**	BCBC
[3]	**0** **ABCDABCDAB**	BDDBDABCBD
[3]	**12** **DDDDDD**	ACACCB
[3]	**255** **CCCCCCCC**	DDADABDB
[3]	**1234567** **ABCDABCDAB**	BAACCCDABA
[4]	**2001001001** **ABCDABCD**	CAAABCDC

Question 2(b) [2 marks available]

[2] BCBCDB

Question 2(c) [4 marks available]

[2] 4 (with one rotor)
[2] 32 (with two rotors)

Question 2(d) [4 marks available]

[1] No

[1] Every ports is connected to a single wire.

Up to [1] mark is available for either of the following:

[1] Each letter is wired to a single other letter.
[1] Wirings between letters can be reversed.

Up to [1] mark is available for either of the following:

[1] If A was encrypted to B, and B was encrypted to C, then A would be connected to two letters.
(A can be replaced by B or C in the last clause)
[1] If A is encrypted to B, then B must be encrypted to A.

Question 3(a) [24 marks available]

Each test for 3(a) consists of a seven digit string (a permutation of the digits `1234567`). The output is a single integer.

There are no marks for incorrect answers, and tests *must* terminate in 2 seconds to receive marks.

[1]	**6417352**	5
[2]	**1234567**	0
[2]	**1235674**	1
[2]	**7123456**	2
[2]	**2371456**	2
[2]	**2741356**	4
[2]	**5627413**	6
[2]	**7612543**	8
[3]	**6245173**	11
[3]	**3412765**	13
[3]	**5674321**	14

Question 3(b) [3 marks available]

[1]	11	(after 2 operations)
[2]	403	(after 6 operations)

Question 3(c) [4 marks available]

[1] 14

[3] `7654123` *or* `5674321`

Question 3(d) [4 marks available]

[1] No

[1] It is possible to get to `1234567` from any starting order.
[1] Finishing in a different order is equivalent to changing the embroidering on the shirts.
[1] Choosing the starting and finish orderings covers the same set of problems as just choosing the starting position.

End of BIO 2008 marks scheme

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type a single upper-case word. The response will either be a single integer or the word `NO`; the correct responses are given on the right. There are no marks for incorrect answers.

[1]	**BOUNCE**	1
[1]	**ENCODE**	NO
[2]	**EIGHT**	8
[2]	**BLACKJACK**	NO
[2]	**FABULOUS**	NO
[2]	**EXERCISE**	NO
[2]	**DRIFTWOOD**	2
[2]	**SERVICEMAN**	7
[2]	**INSIGNIFICANCE**	9
[2]	**THROWDOWN**	2

Additional marks are available for general program behaviour:

[2] Program inputs a single word
[2] For each test a single number, or the word `NO`, is output
[2] Program terminates without crashing / hanging

Question 1(b) [2 marks available]

[2] 10

Question 1(c) [4 marks available]

[1] 12 (for ONE to FIVE)
[3] 18 (for ONE to NINE)

Question 2(a) [24 marks available]

There are 6 multiple part tests used to check program 2(a). Marks are given within the tests, besides the expected output from the program; this will be a 4 by 4 grid containing the letters `R`, `G` and `B` followed by a single integer, or the words `GAME OVER` along with a single integer (the order is not important).

When a grid and an integer occur they are scored separately. Otherwise incorrect output at any stage gets no marks for that stage. For a grid every character must be correct.

For each test you will first need to type in a grid of *uppercase* letters, followed by an integer. Additional input will consist of single integers.

If the program crashes / hangs part way through a test, or takes longer than 2 seconds, the rest of that test should be discarded.

Test 1

```
      RRGB
      GRBG
      RRGB
      GBRB
      2

      RRRB
      GRGB
[1]   RBGG
      GRRG

[1]   82

      0
```

Test 2

```
      RBRB
      GRGR
      BGBG
      RBRB
      1

[1]   0
[1]   GAME OVER
```

Test 3

```
      RGRB
      BGBR
      RBRB
      BRBR
      1

      RBRB
      BRBR
[2]   RBRB
      BRBR

[2]   2

      3

[1]   2
[1]   GAME OVER
```

Test 4

```
      BBBB
      BBBB
      BBBB
      BBBB
      1

      BBBB
      BBBB
[1]   BBBB
      BBBB

[1]   16

      0
```

Test 5

```
      RRGG
      RRGG
      BBRR
      BBRR
      1

      RRGG
      RRGG
[1]   BBRR
      BBRR

[1]   256

      99

      RRGG
      RRGG
[1]   BBRR
      BBRR

[1]   25600

      0
```

Test 6

```
      RRGR
      RBBG
      RRBR
      GBBG
      1

      RRBR
      RBBG
[1]   GRBR
      GBGG

[1]   25

      1

      RRBG
      RBBR
[1]   GRBG
      RBGR

[1]   73

      1

      GRBG
      RBBR
[1]   GRGG
      RBGR

[1]   85

      12

      GBGR
      GRBG
[1]   RBBR
      GRBG

[1]   482

      0
```

Question 2(b) [3 marks available]

When two or more adjacent pieces (horizontally or vertifcally) have the same colour they form a *block*. To score [3] a correct answer requires a 4 x 4 grid where each square is either an `R`, `G` or `B`, and the grid contains one block consisting of 7 pieces, one of 5 and one of 3. There will also be a single piece that is not adjacent to another piece of the same colour.

Question 2(c) [4 marks available]

[4] 324

Question 2(d) [4 marks available]

[1] No

There is [1] mark available for giving an example.

In addition, either of the following approaches can be used for up to [2] additional marks. Students may only score marks from one of the two arguments:

[1] When blocks touch each other they must be different colours.
[1] It is possible to give four blocks which touch each other.

or

Up to [2] marks can be awarded for a valid logical argument justifying an example by a sequence of implications or cases. Award [1] mark if the argument is incomplete, but otherwise valid.

Question 3(a) [23 marks available]

Each test for 3(a) consists of two integers for input and a line containing one or more integers for output. Each integer in the output must be a single digit (excluding 0) and the integers must appear in the correct order.

There are no marks for incorrect answers, and tests *must* terminate in 2 seconds to receive marks.

[1]	**4 5**	2 1 1
[2]	**1 1**	1
[2]	**5 1**	1 1 1 1 1
[2]	**6 32**	6
[2]	**7 63**	6 1
[2]	**8 74**	2 1 2 1 2
[2]	**12 1752**	3 3 3 3
[2]	**14 5000**	2 1 4 1 4 1 1
[2]	**21 1000000**	5 3 3 2 1 2 2 1 2
[3]	**27 50789789**	3 1 1 2 2 1 1 4 4 1 7
[3]	**32 1234567890**	2 1 2 3 1 4 2 1 5 3 3 3 2

Question 3(b) [2 marks available]

[2] 88

Question 3(c) [5 marks available]

[1] They are the reverse of each other.
(It is also valid to include that, for sums from 1 to 9, they are the same; but this on its own is insufficient for this mark).

For the list in dictionary order:

[1] The last entry in the list must start with as many 9s as possible.
[1] If there is a digit which is not 9 it must appear at the end of the arrangement.

For the list in increasing numeric order, up to [2] marks are available from:

[1] The first entry in the list must be as small (in length) as possible.
[1] To minimise the length we require as many 9s as possible.
[1] If there is digit which is not 9 it must appear at the start of the arrangement.

(**Supplementary:** A student who *only* mentions that, for sums from 1 to 9 both arrangements are the same, may be awarded [1] mark in total for this question)

Question 3(d) [5 marks available]

[1]	16	(summing to 8)
[4]	31993503	(summing to 50)

End of BIO 2009 marks scheme

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type a single number. The response will either be one or more single digits or the word `NO`; the correct responses are given on the right.

There are no marks for incorrect answers, however if the solution requires several digits they may appear in any order.

[1]	**123456789**	2 4 5 7 8
[1]	**100**	NO
[2]	**1**	NO
[2]	**148258**	NO
[2]	**555**	NO
[2]	**1035**	3
[2]	**123876**	3 7
[2]	**142857**	2 3 4 5 6
[2]	**49271085**	2
[3]	**123450186**	7

Additional marks are available for general program behaviour:

[2] Program inputs a single number.
[2] For each test one or more single digits, or the word `NO`, is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [2 marks available]

There are [2] marks for getting all three of the following numbers (in any order). Award *only* [1] mark if the students is missing any of the numbers or has written any additional numbers.

[2] 14207985
17049582
28415970

Question 1(c) [3 marks available]

[3] 138

Question 2(a) [24 marks available]

There are 6 multiple part tests used to check program 2(a). Marks are given within the tests, besides the expected output from the program; this will be a 3×3 grid containing the digits `1` to `6` and `x`s.

To score marks for a grid every entry must be correct.

For each test you will first need to type in a grid of digits, followed by an integer. Digits on the same line *must* be separated by spaces. Additional input will consist of single integers.

If the program crashes / hangs part way through a test, or takes longer than 5 seconds, the rest of that test should be discarded.

Test 1

```
        1 3 5
        1 6 5
        1 1 5
        1

        111
[1]     135
        115

        1

        111
[1]     251
        151

        3

        221
[1]     161
        131

        0
```

Test 2

```
        1 1 1
        4 4 4
        1 1 1
        65

        111
[2]     141
        111

        0
```

Test 3

```
        2 1 2
        1 2 1
        2 1 2
        1

        131
[1]     212
        111

        1

        212
[1]     131
        232

        1

        141
[1]     232
        111

        2

        121
[2]     411
        551

        0
```

Test 4

```
        6 6 6
        6 6 6
        6 6 6
        1

        111
[1]     666
        616

        2

        111
[1]     456
        616

        32

        111
[2]     111
        111

        27

        161
[3]     111
        xxx

        0
```

Test 5

```
        4 5 4
        4 5 4
        4 5 4
        19

        111
[2]     564
        111

        81

        111
[2]     355
        125

        64

        xxx
[3]     51x
        32x

         0
```

Question 2(b) [2 marks available]

[2] 32

Question 2(c) [4 marks available]

[1] 8 (with 4 tips)
[3] 160 (with 6 tips)

Question 2(d) [5 marks available]

[1] Yes

[1] The current numbers on the grid and the die's orientation and heading fully determine the state of simulation.
[1] There are only a finite number of different states.
[1] Since there are only a finite number of states we must, eventually, encounter one for the second time.

Up to [1] mark is available for either of the following:

[1] If we return to a state that we have previously visited the die will repeat the same sequence of moves which lead back to the same state.
[1] If we return to a state that we have previously visited the die will be stuck in a loop.

Question 3(a) [23 marks available]

Each test for 3(a) consists of two lines, the first containing 2 integers and the second between 1 and 3 integers. The output will always be a single integer.

There are no marks for incorrect answers, and tests *must* terminate in 5 seconds to receive marks.

[1]	**2 4** **3 5**	6
[2]	**1 20** **20**	1
[2]	**2 10** **10 20**	1
[2]	**2 10** **1 30**	20
[2]	**2 15** **1 20**	10
[2]	**2 9** **249 18**	86
[2]	**3 48** **158 62 14**	2
[2]	**3 24** **158 62 14**	27
[2]	**2 3** **31 79**	12
[3]	**2 19** **31 79**	56
[3]	**3 13** **241 181 31**	40

Question 3(b) [2 marks available]

It is necessary for all four steps listed below to be given, in order, to receive [2] marks.

[2] Fill jug B
Pour from jug B into jug A
Empty jug A
Pour from jug B into jug A

The student is permitted to refer to jug A as the 3 oz jug, and to jug B as the 8 oz jug.

Question 3(c) [5 marks available]

[2] 4 (with 2 jugs)
[3] 30 (with 3 jugs)

(**Supplementary:** There are [2] marks for saying 57 for 3 jugs; this would have been the answer if duplicate jugs were allowed in a cookery set.)

Question 3(d) [5 marks available]

[1] No

Additionally, up to [4] marks can be gained from the following points:

[1] Pouring liquid between two jugs either empties the pourer, fills the recipient, or both.
[1] Every step (*or* each of the three operations) either empties or fills at least one jug.
[1] After each step there must be at least one empty or one full jug.
[1] There are no 1oz capacity jugs, so a jug containing 1oz is neither full or empty.
[1] If no jug is empty or full then no sequence of steps will work.

End of BIO 2010 marks scheme

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type two capital letters and a number, with a single space between the three items. The response should be a single capital letter.

There are no marks for incorrect answers, however students should not be penalised for outputting lower-case letters.

[1]	**A A 7**	M
[3]	**A M 1**	A
[2]	**A A 8**	U
[2]	**P Q 101**	S
[2]	**Y Y 1000**	U
[2]	**Z M 5005**	Z
[2]	**K V 10000**	C
[2]	**K Y 20000**	W
[2]	**B I 987654**	W

Additional marks are available for general program behaviour:

[2] Program inputs two letters and a number.
[2] For each a test single letter is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [3 marks available]

[1] R (together with F to produce X)
[2] Q (together with Q to produce H)

Question 1(c) [3 marks available]

[3] K

Question 2(a) [24 marks available]

There are 6 multiple part tests used to check program 2(a). For each test you will first need to type in six integers separated by spaces. Marks are given within the tests, besides the expected output from the program; this will be four lines output, each containing two items.

(**Supplementary:** If a student gets only one item correct on a line, award half the marks for that line. When recording the total marks for this question *round up* to the nearest integer.)

If the program crashes / hangs part way through a test, or takes longer than 2 seconds, the rest of that test should be discarded.

Test 1

1 3 5 7 2 4

```
[1]   AC 3S
[1]   6 9C
[1]   5 KC
[1]   4 4H
```

Test 2

5 3 5 8 1 8

```
[1]   5C TD
[1]   4 5H
[1]   2 8C
[1]   3 QC
```

Test 3

8 7 2 5 9 9

```
[1]   8C 5D
[1]   7 8C
[1]   12 8C
[1]   1 9D
```

Test 4

2 9 3 6 6 3

```
[1]   2C TH
[1]   22 JC
[1]   19 JC
[1]   17 JC
```

Test 5

8 9 5 9 7 7

```
[1]   8C 8H
[1]   26 8C
[1]   27 8C
[1]   3 9C
```

Test 6

```
1 1 1 1 1 1
```

[1]	AC KD
[1]	4 JC
[1]	4 JC
[1]	2 KD

Question 2(b) [2 marks available]

The following 12 values should appear in order:

[2] 2C KC 3H KH 4S KS 2D KD 4C 2H 7H 5S

(**Supplementary:** A student who gets the first 8 values correct — i.e. from 2C to KD — but makes some other mistake, should be awarded [1] mark.)

Question 2(c) [4 marks available]

[1]	531434	(1 … 9)
[3]	998284	(1 … 10)

Question 2(d) [5 marks available]

[1] Yes

The following pieces of justification are worth marks:

[1] The first pile that is moved in the first game will only contain a single card.
[1] If a pile with a single card is moved onto another pile, that single card and the face card of the other pile will be adjacent when dealt for the second time.
[1] If a pile with top card A can be moved onto a pile with top card B, cards A and B match.
[1] Two adjacent piles can be combined if their top cards match.

Question 3(a) [24 marks available]

Each test for 3(a) consists of a single integer. The output will always be a single integer.

There are no marks for incorrect answers, and tests *must* terminate in 2 seconds to receive marks.

[1]	**11**	159
[2]	**1**	5
[2]	**100**	9911
[2]	**160**	76543
[2]	**1234**	4995116
[3]	**8448**	141555969
[3]	**14659**	987654321
[4]	**12345678**	1398485825262179
[5]	**987654321**	13732787982132387379

Question 3(b) [2 marks available]

[2] 384

Question 3(c) [3 marks available]

[3] 38

Question 3(d) [6 marks available]

[1] More 1001 digits that contain at least one 5.

The following pieces of justification are worth marks:

[1] Every 1001 digit upside-down number contains a 5 in the middle.
[1] Not every 1000 digit upside-down number contains a 5.
[1] Deleting the middle digit of a 1001 digit upside-down number makes a 1000 digit upside-down number. Adding a 5 in the middle of a 1000 digit upside-down number makes a 1001 digit upside-down number.
[1] The above deletions / additions are unique; i.e. each pair of 1000 and 1001 digit upside-down numbers are uniquely linked.
[1] There are the same number of 1000 digit upside-down numbers and 1001 digit upside-down numbers.

End of BIO 2011 marks scheme

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to enter a single integer. The response should also be a single integer.

There are no marks for incorrect answers.

[1]	**100**	10
[1]	**101**	101
[2]	**2**	2
[2]	**1001**	1001
[2]	**371293**	13
[2]	**789774**	789774
[2]	**999883**	999883
[3]	**561125**	335
[3]	**661229**	4379

Additional marks are available for general program behaviour:

[2] Program inputs an integer.
[2] For each test a single integer is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [2 marks available]

To get [2] marks on this question the student needs to give (all) the following ten numbers without any additional numbers. The order is not important:

[2] 10, 20, 40, 50, 80,
100, 160, 200, 250 and 320

(**Supplementary:** If a student omits at most three of the numbers and / or has at most three incorrect numbers, they receive [1] mark.)

Question 1(c) [4 marks available]

[4] 210

Question 2(a) [23 marks available]

There are 8 tests used to check program 2(a). For each test you will type in three lines of input: the first containing six *uppercase* letters (without spaces); the second two *uppercase* letters; the third a single integer. Output will consist of a pair of letters, both of which must be correct to score marks.

Students should *not* be penalized for outputting spaces or lowercase letters.

If the program crashes / hangs part way through a test, or takes longer than 5 seconds, the rest of that test should be discarded.

[1]	**GHIJKL** **AE** **6**	FA
[2]	**ABCDEF** **HP** **1**	PV
[2]	**ABCDEF** **PH** **1**	HB
[3]	**AEFMNO** **DK** **13**	SK
[3]	**AEFMNS** **DK** **13**	SJ
[4]	**ABCDEF** **GO** **100**	QI
[4]	**FJLMQU** **GO** **100**	RJ
[4]	**FDEGNQ** **AE** **9876**	WQ

Question 2(b) [2 marks available]

The following 11 letters should appear in order:

[2] V U M L D A E M U V P

Question 2(c) [6 marks available]

For convenience we will use A–D as a shorthand for *"one of the points A, B, C or D"*; similarly for other ranges. The students do not have to use the same notation but do need to refer to the same groups of points. For example, the students may refer to the groups by their positions in the diagram (e.g. using *top* for A–D) or by some explicit labelling on a drawn diagram, or may use *up* and *down* to refer to the direction of movement.

At most [2] marks are available from the following alternative ways of expressing the train's position:

[1] The train will be between M–T and U–X.
[1] The train is facing towards the U–X point (or *down*).

or

[2] The train is at M–T → U–X

or

[1] The train will have just passed M–T.
[1] The train will next pass U–X.

The following pieces of justification are worth marks:

[2] A description of the following position cycle which the train follows:

M–T → U–X
U–X → U–X
U–X → M–T
M–T → E–L
E–L → A–D
A–D → A–D
A–D → E–L
E–L → M–T

(**Supplementary:** A student who fails to accurately describe the cycle but who indicates, directly or indirectly, that it has 8 steps, should be awarded [1] mark.)

Finally, the following pieces of justification are also worth marks:

[1] 1,000,000,000,000,000,000 is divisible by 8.
[1] The train will finish in the starting portion of the cycle.

Question 2(d) [4 marks available]

[4] 2572

Question 3(a) [23 marks available]

Each test for 3(a) consists of three lines each containing two integers separated by a space. The output should be three lines each containing a single number.

There are no marks for incorrect answers, *all three* numbers in the output must be correct, and tests *must* terminate in 5 seconds to receive marks.

[1]	**26 61** **1 94** **1 610**	1 1 2
[2]	**1 2** **1 3** **1 4**	1 2 1
[2]	**14 543** **5 75** **71 713**	2 1 1
[3]	**21 911** **329 927** **66 71**	2 2 3
[3]	**250 361** **34 756** **18 735**	3 4 3
[4]	**77 383** **48 677** **232 471**	5 4 4
[4]	**220 691** **198 222** **410 666**	5 5 6
[4]	**402 788** **203 959** **404 777**	6 6 6

Question 3(b) [2 marks available]

[2] 15

Question 3(c) [4 marks available]

[2] The largest size is 6
[2] There are 10302 such ladders

Question 3(d) [6 marks available]

[1] Yes

The following pieces of justification are worth marks. Where "number ladder" is used below the student may refer to a sequence of transformations. Whilst not strictly correct to only refer to a transformation in these cases (rather than a sequence), do not deduct marks if the student uses this terminology:

[1] The digit 0 can be transformed into a non-zero digit.
[1] A number ladder exists between any two non-zero digits, that does not involve a transformation to 0.
[1] If it is valid to transform one digit into another it is also valid to transform an arbitrary integer into another integer where a single equivalent transformation has taken place.
[2] An arbitrary integer can be transformed into another integer of the same size by a sequence of number ladders transforming the digits one at a time.

End of BIO 2012 marks scheme

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type two integers, with a single space between them. The response should be a 24 hour clock time.

The question specifies that both the hour and minutes in a time should be displayed with two digits. If a student fails to output two digits but is otherwise correct (e.g. displaying the hour `01` as `1`) only award [1] mark for that test.

[1]	**`1 31`**	`00:48`
[2]	**`0 0`**	`01:00`
[2]	**`18 18`**	`01:18`
[2]	**`67 1507`**	`02:07`
[2]	**`0 15`**	`00:00`
[2]	**`0 7`**	`00:00`
[2]	**`17 26`**	`13:20`
[2]	**`17 215`**	`06:40`
[2]	**`5779 5864`**	`19:12`
[2]	**`21923 26268`**	`14:24`

Additional marks are available for general program behaviour:

[2] Program inputs two numbers.
[2] For each a test a time is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [3 marks available]

[1] 0;
[2] 8, 9, 16 and 18.

(**Supplementary:** If a student has missed out at most one of the numbers 8, 9, 16 and 18, and / or has added at most one additional erroneous number, award [1] mark.)

Question 1(c) [4 marks available]

[4] 1440 hours

Question 2(a) [26 marks available]

There are 5 tests used to check program 2(a). For each test you will need to type in two lines, each containing the digits `1`, ..., `5` in some order; separate the digit with spaces.

For each test three grids should be output. Marks for each grid are independent. A grid must be entirely correct to receive marks.

(**Supplementary:** If all of a student's grids are upside-down then deduct [4] marks in total but otherwise treat the grids as though they had the correct orientation.)

If the program crashes / hangs part way through a test, or takes longer than 2 seconds, the rest of that test should be discarded.

Test 1

```
      1 2 3 4 5
      1 2 3 4 5

      SSSSS
      F.*..
[1]   .....
      .....
      .FFFF

      .SSSS
      F...*
[1]   .....
      ...S.
      .FFFF

      ..SSS
      FF...
[1]   .....
      ...SS
      .*FFF
```

Test 2

```
      5 2 4 1 3
      1 4 2 3 5

      SSSSS
      ..*.F
[1]   .....
      .....
      FFFF.

      .SSSS
      ...*F
[1]   .....
      .....
      FFFFS

      .FF.F
      ...SS
[3]   ....S
      ...F.
      SS*.F
```

Test 3

```
      1 4 3 5 2
      2 3 4 1 5

      SSSSS
      F.*..
[1]   .....
      .....
      .FFFF

      S.SSS
      F...*
[2]   .....
      ....S
      .FFFF

      FFSSF
      .....
[3]   ....S
      S..F.
      *.F.S
```

Test 4

```
      1 3 5 4 2
      4 1 2 5 3

      SSSSS
      F.*..
[1]   .....
      .....
      .FFFF

      SSS.S
      F...*
[1]   .....
      ...S.
      .FFFF

      ..S..
      F.FSF
[4]   ..SF*
      .S.SF
      .....
```

Test 5

```
      3 4 1 5 2
      3 4 1 5 2

      SSSSS
      ..*..
[2]   ..F..
      .....
      FF.FF

      SS.SS
      ....*
[1]   ..F.S
      .....
      FF.FF

      ..S.*
      .S.F.
[3]   ..FSS
      SF..F
      ....F
```

Question 2(b) [3 marks available]

[3] 65

Question 2(c) [4 marks available]

[4] 21

Question 3(a) [24 marks available]

Each test for 3(a) consists of an input string of up to 12 distinct letters, and either an alphabetical output string or the word `IMPOSSIBLE`. The case of the letters for input and output is important.

Each test, other than those that are `IMPOSSIBLE` has multiple solutions. These have been listed in alphabetical order. A student is only required to give one solutions. *Since some of these solutions are long, it is acceptable to only verify the underlined characters.*

There are no marks for incorrect answers (this includes any letter with an incorrect case), and tests *must* terminate in 2 seconds to receive marks.

Test 1

`mnoqRTwxy`

```
     abCDefGhiKlNoPqrsTuw
     ABcdEFgHIkLnOpQTUW
     aBCdeghIjkLnOQrtwy
     AbcDEGHiJKlNoqsWY
     ABCdFhJKlNoprSVwxy
     abcDfHjkLnOPStvWXY
     AbCDghIjQrtuVwx
     aBcdGHiJqsUvWX
     ABdeFGijpqRstuvXY
     abDEfgIJPQRUVxy
     AbDekLnORSTUvXy
     aBdEKlNoRSTuVxY
     aCeFhJprSUwY
[1]  AcEfHjPStuWy
     ACfGhikLnOPqrsTUVwxY
     acFgHIKlNopQTuvWXy
     aEFGijkLnOpqRstVx
     AefgIJKlNoPQRvX
     BCdEfGhiPqrsTvwXy
     bcDeFgHIpQTVWxY
     bCDEFhJkLnOprSuvwX
     BcdefHjKlNoPStUVWx
     bDFGijKlNopqRstUy
     BdfgIJkLnOPQRuY
     CEghIjKlNoQrtUvwXY
     ceGHiJkLnOqsuVWxy
     RST
```

Test 2

`N`

[2] `IMPOSSIBLE`

Test 3

`abcdef`

[2] `IMPOSSIBLE`

See overleaf for additional tests.

Question 3(b) [2 marks available]

[2] abcdghiJNot

Question 3(c) [4 marks available]

[4] 598

Question 3(d) [5 marks available]

[1] No

The following pieces of justification are worth at most [4] marks. Where two points are separated by an *or* only one of the points can be awarded a mark.

[1] A button press changes the state of the lights by a procedure that does not depend on those lights.

or

[1] A button press leaves some lights alone and changes the state of the surrounding lights to their next state — whatever their current state.

or

[1] A sequence of button presses changes the state of the lights in a procedure that does not depend on those lights.

[1] If two systems have different lighting there must be at least one light (*L*) that is in a different state in the two systems.

[1] A sequence of button presses will change the state of light *L* in a consistent manner.

or

[1] If *L* is in a different starting state, a sequence of button presses will leave it in a different final state.

[1] If a sequence of button presses unlocks a system, it leaves light *L* off. If *L* finishes in a different state, the system is not unlocked.

Test 4 [2 marks]	**Test 5 [2 marks]**	**Test 6 [3 marks]**	**Test 7 [3 marks]**	**Test 8 [4 marks]**	**Test 9 [5 marks]**
ABF	**gklmq**	**ghkLNQSTwXy**	**ghkLmNrTWXy**	**abrs**	**deHi**
a	L	ABcdEFgHIklMnOpQUWY	ABcdEFgHIklMnOpQRUWY	AbcdEFgIkLMOpRtuvwX	abCdefGHiKnoQtUvwxy
AbCDefGhiKlNoPqRSuw	ceGHiJklnOqrStuVWxy	abCDefGhiKMNoPqrsuwY	abCDefGhiKMNoPqsuwY	abCdegHIjkLMOUvXy	AbcdEGiJKnopQRTuvWx
aBCdEfGhiPqRSvwXy	CEghIjKNoQRsTUvwXY	aBCdeghIjklMnOQrTw	aBCdeghIjklMnOQTw	AbCdFHJKlMnopqSTUy	abcdfjklOqRsuVW
abcDeFgHIpQrsVWxY	bDFGijKNopqSTUy	AbcDEGHiJKMNoqstWy	AbcDEGHiJKMNoqRstWy	abcdGiJMNQRsTuVwx	AbCdgHIjLNpSUwXy
aBcdefHjKlNoPrTUVWx	bcDeFgHILpQrsVWxY	abcDfHjklMnOPSTvWXy	abcDfHjklMnOPRSTvWXy	aBCeFHJMNpqSTuvXY	ABcEFgIklOPRStvWxy
abCDEFhJkLnOpRtuvwX	bCDEFhJklnOpRtuvwX	ABCdFhJKMNoprStVwx	ABCdFhJKMNopStVwx	ABcEfjMNPqRSvwXy	aBCegHIjklOpSuvwxY
ABCdeghIjkLnOQRsTwy	BdfgIJklnOPQstuY	AbCDghIjLMQrTuVwxY	aBcdGHiJLMqRstUvWXY	ABCfGHikLMOPQstuY	ABCFHJKnoPqsTuwXY
AbcDfHjkLnOPrTvWXY	BcdefHjKNoPrTUVWx	aBcdGHiJLMqstUvWXY	AbCDghIjLMQTuVwxY	aBcFgIKlMnopRtVwxy	aBcGiJLNpQRTVWy
ABcdGHiJqrStUvWX	BCdEfGhiLPqRSvwXy	ABdeFGijLMpqRsTuvXy	ABdeFGijLMpqrsTuvXy	AbdeFGhijMNpQrsVWxY	abdEfghIJLNrSTXY
AbDEfgIJPQstUVxy	acFgHIKNopQrsuvWXy	abDEfgIJLMPQRtUVx	abDEfgIJLMPQrtUVx	abdEhKlMnoqrStWY	AbdehklOpqrstVY
ABdEKlNouVxY	aCeFhJLpRtUwY	AbDeklMnORSUvX	AbDeklMnOrSUvX	ABefghIJKlMnoPrTUVWx	ABeFGhijLNPQrUV
aBdfgIJkLnOPQstuY	aEFGijklnOpqSTVx	aBdEKMNoRSuVxy	aBdEKMNorSuVxy	aBEFGhijkLMOpQrsUW	aBEhKnopqrstUX
abDFGijKlNopqSTUy	abcDfHjklnOPrTvWXY	aCeFhJLMprStUwy	AcEfHjLMPRSTuW	aCDefGHiKlMnoPQstvX	aCDeFHJLNPqsTvwx
ACeFhJpRtUwY	abCDefGhiKNoPqRSuw	AcEfHjLMPSTuW	aCeFhJLMpStUwy	AcDEGiJKlMnoQRsTUvwXY	AcDEfjLNqRsUvWxY
aCEghIjKlNoQRsTUvwXY	abDEfgIJLPQstUVxy	ACfGhiklMnOPqrsUVwxy	ACfGhiklMnOPqsUVwxy	acDfjkLMOPqRSUVwxY	ACDfGHiklOQtwX
aceGHiJkLnOqrStuVWxy	aBcdGHiJLqrStUvWX	acFgHIKMNopQuvWX	acFgHIKMNopQRuvWX	ACDgHIjMN	acDFgIKnoPRStUVWY
AcFgHIKlNopQrsuvWXy	aBCdeghIjklnOQRsTwy	aEFGijklMnOpqRsTVxY	aEFGijklMnOpqrsTVxY	aDEfghIJMNPrTuWy	aDEFGhijklOPQruXy
AEFGijkLnOpqSTVx	aBdEKNouVxY	AefgIJKMNoPQRtvXY	AefgIJKMNoPQrtvXY	ADehkLMOqrStuVWxy	ADefghIJKnorSTuVy
BcdEFgHIkLnOpQrsUW	ACfGhiklnOPqRSUVwxY	BCdEfGhiLMPqrsvwX	bcDeFgHILMpQRVWxy	bCdEfGHiMNPQstUVxy	bcdeFgILNPRStuWX
bcDEGHiJKlNoqrStWY	AefgIJKNoPQstvX	bcDeFgHILMpQVWxy	BCdEfGhiLMPqsvwX	bcdefjKlMnoPqRSuw	bCdEFHJklOPqsTUVwy
BCdFhJKlNopRtVwxy	AcEfHjLPrTuWy	bCDEFhJklMnOprStuvwXY	bCDEFhJklMnOpStuvwXY	BCEgHIjKlMnouVxY	BCEfGHiLNQtuVwY
bCDghIjQRsTuVwx	AbCDghIjLQRsTuVwx	BcdefHjKMNoPSTUVWxY	BcdefHjKMNoPRSTUVWxY	BceGiJkLMOQRsTwy	BcefjKnoqRsWXy
BdeFGijpqSTuvXY	AbDeklnOUvXy	BdfgIJklMnOPQRtuy	BdfgIJklMnOPQrtuy	bdfghIJkLMOPrTvWXY	bdFGhijKnoPQrvxY
bDekLnOUvXy	AbcDEGHiJKNoqrStWY	bDFGijKMNopqRsTU	bDFGijKMNopqrsTU	BhMNqrStUvWX	BfghIJklOrSTUvx
cEfHjPrTuWy	ABCdFhJKNopRtVwxy	ceGHiJklMnOqstuVWx	ceGHiJklMnOqRstuVWx	cDeFgIMNpRtUwY	CDEgHIjKnopSVw
CfGhikLnOPqRSUVwxY	ABdeFGijLpqSTuvXY	CEghIjKMNoQrTUvwXy	CEghIjKMNoQTUvwXy	CDEFHJkLMOpqSTVx	cDeGiJklOpQRTUWXY
efgIJKlNoPQstvX	ABcdEFgHIklnOpQrsUW	LMRSY	LMrSY	DFGhijKlMnopQrsuvWXy	DhLNpqrstuvxy

End of BIO 2013 marks scheme

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type in a single integer. The output should be a pair of integers, which may appear in *either* order. Both numbers must be correct to score marks.

[1]	**5**	3 7
[2]	**33**	31 37
[2]	**34**	33 37
[2]	**399**	393 409
[2]	**456**	451 463
[2]	**3301**	3297 3307
[2]	**3304**	3301 3307
[3]	**9703**	9691 9727
[3]	**10000**	9999 10003

Additional marks are available for general program behaviour:

[2] Program inputs a single integer.
[2] For each test two integers are output.
[2] Program terminates without crashing / hanging.

Question 1(b) [2 marks available]

[2] 23

Question 1(c) [3 marks available]

[3] 1,999,999,998

Question 2(a) [25 marks available]

There are 8 tests used to check program 2(a). For each test you will need to type in a line containing a single integer, followed by multiple lines containing the digits `1`, ..., `6` in some order; separate the digits with spaces.

For each test two integers should be output. These numbers must be correct and appear in the correct order to score marks.

There are no marks for incorrect answers, and tests *must* terminate in 2 seconds to receive marks.

[2]
```
2
3 4
6 5
```
```
0 4
```

[2]
```
2
2 4
1 5
```
```
0 0
```

[3]
```
3
3 4 6
6 5 6
4 4 3
```
```
4 4
```

[3]
```
3
5 2 6
1 6 1
4 2 3
```
```
8 0
```

[3]
```
4
5 6 5 6
4 3 4 3
5 6 5 6
4 3 4 3
```
```
16 4
```

[3]
```
4
5 6 3 4
4 3 5 2
5 6 4 2
4 3 6 5
```
```
8 10
```

[4]
```
5
3 3 1 4 4
3 5 2 6 4
2 1 1 1 2
6 4 2 3 5
6 6 1 5 5
```
```
8 16
```

[5]
```
6
1 2 3 4 5 6
5 6 6 5 4 3
1 4 2 2 6 6
4 2 6 3 1 4
5 2 3 6 1 5
4 2 2 2 3 3
```
```
24 10
```

Question 2(b) [2 marks available]

[2] 31

Question 2(c) [3 marks available]

[3] 18

Question 2(d) [5 marks available]

[1] No

The following pieces of justification are worth at most [4] marks.

[1] In the loop, whenever a line extends to a tile in any direction, it must be matched somewhere by the line extending over a tile in the opposite direction.
(This point can also be awarded if the justification is given in terms of specific directions, such as left/right.)

[1] Every loop covers an even number of tiles.

[1] Each player's score must be even as it consists of the sum of even numbers.

[1] The difference between two even numbers (the players' scores) must be even.

Question 3(a) [25 marks available]

Each test for 3(a) consists of an input integer and an alphanumeric output string. The case of the letters for the output can be ignored.

There are no marks for incorrect answers, and tests *must* terminate in 2 seconds to receive marks.

[1]	**1**	A
[2]	**21**	U
[2]	**321**	JP
[2]	**4321**	HPQ
[2]	**54321**	LNOV
[2]	**654321**	AHJSVW
[2]	**7654321**	EHJK025
[3]	**87654321**	CEILRU059
[4]	**234234234**	BEHJPVX267
[5]	**987654321**	MNOPQTUX026

Question 3(b) [2 marks available]

The following five passwords must be in the exact order listed below:

[2] `14, BIO, NTU, ABCDE, BIO14`

Question 3(c) [3 marks available]

[3] 68,719,476,735
Which can optionally be written as 2^{36}-1.

Question 3(d) [5 marks available]

[1] 1 pair

The following pieces of justification are worth at most [4] marks.

At most [2] marks from the following four points:

[1] Two adjacent passwords either have the same length or the second password is one character longer than the first.
[1] If the first password has fewer than 18 characters there are fewer than 36 characters in the combined passwords.
[1] If the second password has more than 18 characters the total number of characters in the passwords is more than 36, so at least one character appears in both passwords.
[1] Both passwords must contain 18 characters.

At most [2] marks from the following three points:

[1] As the passwords are ordered, the *first* password must contain the A.
[1] If the first password contains an A and the second does not, the first password can only be the last password (of that length) beginning with an A.
[1] ATUVWXYZ0123456789 and BCDEFGHIJKLMNOPQRS fulfill the criteria.

Question 1(a) [23 marks available]

For each test of the program for 1(a) you need to type in a single word consisting of uppercase letters. The output should be a single integer.

[1]	**BBACBB**		3
[2]	**XX**		1
[2]	**YZ**		0
[2]	**OLYMPIAD**		0
[2]	**RACECAR**		3
[2]	**KKKKKKK**	(7 Ks)	7
[2]	**BBIIOIIBB**		9
[2]	**PPPQQQQPPP**		19
[2]	**AAAAAAAAAA**	(10 As)	31

Additional marks are available for general program behaviour:

[2] Program inputs a word.
[2] For each a test an integer is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [2 marks available]

The following five groupings can be given in any order but all must be given (with no additional groupings) to score [2] marks.

[2] (A)(ABCBA)(A)
(A)(A)(BCB)(A)(A)
(A)(A)(B)(C)(B)(A)(A)
(AA)(BCB)(AA)
(AA)(B)(C)(B)(AA)

(**Supplementary:** A student who lists three or four of these groupings (and no additional ones) should be awarded [1] mark.)

Question 1(c) [6 marks available]

[1] The *block palindrome* has an even number of letters.

The following pieces of justification are worth marks.

[1] As the reverse of a *block palindrome* shows the same blocks, each block in the first half of the word is repeated in the second half of the word.
[1] 2 times any value is even.

[1] There is only 1 grouping.

The following pieces of justification are worth marks.

[1] If there is more than 1 grouping at least one grouping contains more than 2 blocks.
[1] A grouping containing more than 2 blocks can be changed into a grouping containing the first block, the last block and then the other blocks joined together. This is a contradiction as all the groupings must contain an even number of blocks.

Question 2(a) [27 marks available]

There are 6 tests used to check program 2(a). For each test you will need to type in a line containing three integers.

For each test ten lines should be output, each consisting of two digits (from 0 to 9 inclusive) and a single letter H or V. Lines must appear in the correct order and all three values matching the scheme to be correct.

For some tests there are marks for getting a group of lines correct; every line in a group must be correct to score marks. Groups are separated by horizontal lines in the mark scheme; the student's program will *not* indicate these groups.

Each group is to be scored independently. A program that outputs fewer than ten lines might still have some of the earlier groups correct. Lines must still appear in the correct order; e.g. if the first group covers the first three lines it must appear as the first three lines in the student's output.

Tests *must* terminate in 1 second to receive marks.

__Test 1__ **10 5 9999**

```
      5 0 V
      5 5 V
      0 6 H
      0 1 V
[2]   7 2 V
      7 7 V
      2 8 H
      2 3 V
      9 4 V
      9 9 H
```

Test 2 **`1 1 1000`**

```
[1]   1 0 H
      7 0 H
[1]   1 2 H
      5 2 H
[1]   1 4 H
      5 4 H
[1]   9 4 H
      1 6 H
[1]   3 6 H
      5 6 H
```

Test 3 **`2 2 12345`**

```
      2 0 H
      2 6 H
[1]   2 2 H
      4 9 H
      7 3 H
[2]   7 9 H
      9 5 H
      2 4 H
[2]   7 5 V
      2 9 H
```

Test 4 **`5 7 12346`**

```
[1]   1 1 H
      1 7 H
      1 5 H
[2]   7 8 H
      3 3 H
      7 3 H
      5 6 H
[2]   9 0 H
      1 9 H
      7 0 H
```

Test 5 **`1 29209 32719`**

```
[2]   9 0 V
      9 6 V
      7 6 V
      7 2 V
      5 0 V
[3]   5 8 V
      5 6 V
      5 4 V
      3 6 V
      3 4 V
```

Test 6 **`16807 1 9999`**

```
[2]   1 0 V
      9 0 V
      3 6 V
      3 3 H
      8 7 H
[3]   8 5 H
      6 6 H
      7 9 H
      5 8 V
      1 8 V
```

Question 2(b) [2 marks available]

The following four co-ordinates can be given in any order but all must be given (with no additional co-ordinates) to score [2] marks.

[2] (3,0)
(4,0)
(8,0)
(7,0)

Question 2(c) [3 marks available]

[3] 18

Question 2(d) [5 marks available]

[5] 1424

(**Supplementary:** 2848 is worth [2] marks)

Question 3(a) [25 marks available]

Each test for 3(a) consists of 5 input integers and an output string. The case of the letters for the output can be ignored.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**1 2 1 0 8**	BCAB
[2]	**1 0 0 0 1**	A
[2]	**1 1 0 0 2**	BA
[2]	**0 3 0 3 12**	DBBDBD
[2]	**5 5 0 0 56**	AABBBBBAAA
[2]	**2 2 2 2 2520**	DDCCBBAA
[4]	**2 3 4 5 1234567**	CCBDBDACDADBCD
[5]	**5 4 4 4 123456789**	CACBDAABDACBADCBD
[5]	**5 5 5 5 11223344556**	DDACBBABCDDDCAABCCBA

Question 3(b) [2 marks available]

[2] 10

Question 3(c) [5 marks available]

[1] No

The following pieces of justification are worth at most [3] marks:

[1] If every position has changed the first/left position must have changed.
[1] If the first/left position changes the other positions must have been in reverse alphabetical order.
[1] After the first/left position changes the other positions will be in alphabetical order.
[1] Positions can only be simultaneously in both alphabetical and reverse alphabetical order if they contain one artist.

An additional [1] mark is available for the following:

[1] If the positions (excluding the first/left) only contain one artist after the *n+1*st arrangement, there is no *n+2*nd arrangement.

Question 1(a) **[23 marks available]**

For each test of the program for 1(a) you need to type in a single word consisting of uppercase letters. The output should be a vulgar fraction.

[1]	**L**	1 / 2
[1]	**R**	2 / 1
[1]	**LRLL**	4 / 7
[2]	**LLRLR**	5 / 13
[2]	**LLLRRR**	4 / 13
[2]	**LLRRLL**	7 / 17
[2]	**RRRLRRR**	19 / 5
[2]	**LLLLRLLLL**	6 / 29
[2]	**LLLLLLLLLL** (10 Ls)	1 / 11
[2]	**LRLRLRLRLR**	89 / 144

Additional marks are available for general program behaviour:

[2] Program inputs a word.
[2] For each a test a vulgar fraction is output.
[2] Program terminates without crashing / hanging.

Question 1(b) **[2 marks available]**

[2] LRRR

Question 1(c) **[3 marks available]**

[2] 999999 Ls
[1] 0 Rs

Question 1(d) **[3 marks available]**

[1] No

The following pieces of justification are worth at most [2] marks:

[1] A negative fraction has either a negative numerator or denominator.
[1] Two non-negative numbers added together produce a non-negative number.
[1] The formula adds together non-negative numbers, so never produces a negative numerator or denominator.

Question 2(a) **[24 marks available]**

There are 8 tests used to check program 2(a). For each test you will need to type in 2 lines, the first containing 3 integers and the next containing between 1 and 6 integers.

For each test a 5×5 grid of integers should be output, each integer being a number from 0 to 3.

The entire grid must be correct to receive the marks for that test.

(**Supplementary:** If the student is consistently printing the grids upside-down, mark the grids as though they are in their correct orientation.)

Tests *must* terminate in 1 second to receive marks.

Test 1

```
8 1 4
0
```

[2]

```
0 0 1 0 0
0 1 0 1 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Test 2

```
6 3 18
3 5 11
```

[3]

```
1 1 1 1 0
1 1 1 2 1
0 0 1 1 1
1 0 0 1 0
1 1 0 0 1
```

Test 3

```
12 2 7
1 24
```

[3]

```
0 0 0 0 0
0 1 1 0 0
1 1 0 1 0
0 1 1 0 0
0 0 0 0 0
```

Test 4

```
7 3 23
2 9 14
```

[3]

```
0 2 1 2 0
2 1 0 1 2
0 2 2 2 0
0 1 3 1 0
0 0 1 0 0
```

Test 5

```
1 4 61
4 16 4 1
```

[3]

```
0 1 1 3 3
1 2 0 0 3
1 0 0 0 0
3 0 0 0 3
3 3 0 3 3
```

Test 6

```
18 5 76
2 2 24 23 4
```

[3]

```
1 3 3 3 1
3 1 2 1 3
3 2 3 2 3
3 1 3 1 3
1 3 3 3 1
```

Test 7

```
3 6 150
2 3 5 7 11 13
```

[3]

```
2 3 2 3 2
3 2 2 2 3
2 2 2 2 2
3 2 2 2 3
2 3 2 3 2
```

Test 8

```
3 6 999
2 3 5 7 11 13
```

[4]

```
2 3 2 3 0
2 2 3 2 2
3 2 3 2 1
3 1 1 2 2
0 3 3 1 3
```

Question 2(b) [2 marks available]

[2] 16

Question 2(c) [4 marks available]

[3] There are 20 possible inputs.

Only a single one of the following (20 options) needs to be given to get [1] mark.

```
1 3 8         1 3 8                       6 3 8         6 3 8
2 18 20       7 23 5                      2 3 5         22 23 20

3 3 8         3 3 8         3 3 8         3 3 8         3 3 8
3 20 17       8 20 12       13 20 7       18 20 2       23 20 22

8 3 8         8 3 8         8 3 8         8 3 8         8 3 8
3 5 2         8 5 22        13 5 17       18 5 12       23 5 7

11 3 8        11 3 8                      16 3 8        16 3 8
17 3 20       22 8 5                      12 8 20       17 13 5

21 3 8        21 3 8
7 13 20       12 18 5
```

Question 2(d) [4 marks available]

[1] No.

[1] There are some combinations of starting landscape and position that lead to the same final landscape.

There are [2] marks for either of the following:

[2] If migrations have occurred and at least one person remains on the given position, it is also possible that no migrations occurred and a person was placed in that position.

[2] An explicit example showing two combinations of starting landscape and position that lead to the same final landscape. The position *must* be the same in both.

Question 3(a) [27 marks available]

Each test for 3(a) consists of 3 input integers and an output integer.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	`100 2 13`	4
[2]	`20 2 3`	2
[2]	`20 2 13`	4
[2]	`100 73 89`	2
[2]	`100 19 97`	7
[2]	`1000 3 971`	9
[2]	`2000 977 997`	4
[2]	`5000 83 3643`	10
[3]	`614700 3643 90149`	18
[3]	`987654 3643 90149`	16
[3]	`1000000 2 968137`	18
[3]	`1000000 993851 995387`	3

Question 3(b) [2 marks available]

[2] 12

Question 3(c) [3 marks available]

[3] 41041

Question 3(d) [3 marks available]

[3] The shortest path between *p* and *q* has a length less than *n*.

Question 1(a) [23 marks available]

For each test of the program for 1(a) you need to type in a single word consisting of uppercase letters. The output should be a single uppercase letter.

[2]	**R**	R
[1]	**GG**	G
[1]	**GR**	B
[1]	**RRR**	R
[2]	**RGB**	G
[2]	**BGGRB**	B
[2]	**BRGRBG**	G
[2]	**GGGGGG**	G
[2]	**GRBRBRBRBR**	B
[2]	**RBGBGBGBGR**	R

Additional marks are available for general program behaviour:

[2] Program inputs a word.
[2] For each a test a single letter is output.
[2] Program terminates without crashing / hanging.

Question 1(b) [3 marks available]

[1] 3

The following three words can be given in any order but all must be given (with no additional groupings) to score [2] marks

[2] RRRBBGGRG
GBGGRRBBB
BGBRGBRGR

(**Supplementary:** A student who lists 1 or 2 of these groupings (and no additional ones) should be awarded [1] mark.)

Question 1(c) [4 marks available]

[1] 1

There is [1] mark for either of the following:

[1] Given three adjacent squares in a triangular pattern, if two are known there is only one possibility for the other square.
[1] Given a complete row, there are three possibilities for the adjacent larger row. Any square / colour combination appears in one of those possibilities.

The following pieces of justification are worth at most [2] marks:

[1] If a complete row is known and one value is known in an adjacent row, there is a single way that row can be completed.
[1] The smallest row is completely known.
[1] By induction there is a single way each of the larger rows can be completed.

Question 1(d) [3 marks available]

[3] 10

(**Supplementary:** Any answer of the form 3^k+1 ($k>1$) is correct, so other acceptable answers (worth the [3] marks) include 28, 82, 244, ... or giving the formulae.)

Question 2(a) [24 marks available]

There are 9 tests used to check program 2(a). For each test you will need to type in five integers.

For each test a 5×5 grid of symbols should be output, each being an `O`, `X` or `*`, followed by two integers.

The entire grid must be correct to receive the marks for that grid, and both integers must be correct for the additional marks. The grid is scored independently to the integers.

(**Supplementary:** If the student is consistently printing the grids upside-down, mark the grids as though they are in their correct orientation.)

Tests *must* terminate in 1 second to receive marks.

Test 1 **`4 10 14 23 46`**

```
      O X X * O
      * X * * *
[1]   * * * * *
      X * * * X
      X * * * X

[1]   7 2
```

Test 2 **`5 4 3 2 1`**

```
      * * * * *
      * * * * *
[1]   * * * * *
      * * * * *
      * * * * *

[1]   0 0
```

Test 3 **`1 2 2 2 8`**

```
      * * O * *
      * * * * *
[1]   * * * * *
      * * * * *
      * * * * *

[1]   0 1
```

Test 4 **`36 1 36 1 13`**

```
      X * * * *
      * * * * *
[2]   * * * * *
      * * * * *
      * * * * *

[1]   1 0
```

Test 5 **`36 1 36 1 22`**

```
      X X X X X
      X O * * *
[2]   * * * * *
      * * * * *
      * * * * *

[1]   6 1
```

Test 6 **`1 2 3 4 54`**

```
      X X * X O
      O X * O X
[2]   O X * X X
      X X * X *
      O X * * *

[1]   12 5
```

Test 7 **`12 3 7 2 60`**

```
      O O O O O
      O O O X X
[2]   X X O O X
      X X O O X
      X X O O X

[1]   11 14
```

Test 8 **`6 10 5 6 60`**

```
      O X X O O
      O X X O O
[2]   X X O X O
      X X O X O
      X X O X O

[1]   13 12
```

Test 9 **`33 10 33 8 60`**

```
      O X O X X
      O X X X X
[2]   O X X X X
      O X X X X
      X X X X X

[1]   20 5
```

Question 2(b) [2 marks available]

[2] 23

Question 2(c) [3 marks available]

[3] 9274

Question 2(d) [5 marks available]

At most [1] mark for either of the following:
[1] The last edge was on the boundary.
[1] The last pair of adjacent dots were on the boundary.

At most [1] mark for either of the following:
[1] Any internal edge lies between two squares.
[1] A pair of dots, where at least one does not lie on the boundary, lies between two squares.

At most [1] mark for either of the following:
[1] Any edge on the boundary is adjacent to a single square.
[1] A pair of adjacent dots, both on the boundary, are adjacent to a single square.

[1] The last move completes any squares adjacent to the last edge / joined points.
[1] On the last turn a single square was completed.

Question 3(a) [25 marks available]

Each test for 3(a) consists of 4 input integers and an output integer.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**2 3 4 3**	3
[1]	**1 1 20 20**	1
[1]	**1 1 10 15**	0
[1]	**1 2 10 15**	1
[2]	**1 3 10 20**	6
[2]	**1 10 5 25**	98
[2]	**2 2 10 6**	3
[2]	**2 2 21 14**	8
[2]	**3 3 10 6**	51
[2]	**4 4 20 12**	37610
[3]	**2 9 17 24**	98852
[3]	**3 9 17 24**	10092972
[3]	**5 8 23 17**	2093750500

Question 3(b) [2 marks available]

[2] 18

Question 3(c) [6 marks available]

[3] 9 9 9 9 8 8 8 5 4 2 (order is not important)
[3] 33099

Question 1(a) [26 marks available]

For each test of the program for 1(a) you need to type in two integers. The output should be a single number. It is the value, rather than the format, of the output that is important; e.g. `170` and `170.00` are the same number.

[1]	**10 50**	116.55
[1]	**0 0**	100
[1]	**0 70**	100
[1]	**49 0**	492.17
[1]	**70 100**	170
[2]	**21 21**	143.46
[2]	**31 31**	180.80
[2]	**24 30**	152.22
[2]	**76 79**	214.48
[2]	**98 69**	317.74
[2]	**61 52**	287.57
[2]	**36 37**	207.22
[2]	**61 38**	6755.51
[2]	**85 46**	34606.34

Additional marks are available for general program behaviour:

[1] Program inputs two integers.
[1] For each a test a single value is output.
[1] Program terminates without crashing / hanging.

Question 1(b) [2 marks available]

[2] 5

Question 1(c) [3 marks available]

[3] Interest = 96%
Repayment = 49%

Question 2(a) [24 marks available]

There are 9 tests used to check program 2(a). For each test you will need to type in an integer followed by a word consisting of *uppercase* letters.

For each test you should see as output a word of 6 letters, followed by a word with the same number of letters as the input word.

The two output words are scored independently. The first mark shown by each test is for the first word and the second mark for the second word. A word must be correct in every letter to score the marks.

Tests *must* terminate in 1 second to receive marks.

[1] [1]	**5 ABCD**	EJOTYD EOYK
[1] [1]	**1 A**	ABCDEF A
[1] [1]	**2 Z**	BDFHJL U
[1] [2]	**3 AZ**	CFILOR CC
[1] [2]	**4 BIO**	DHLPTX HRO
[2] [1]	**10 MZNOYW**	JTDOZL IJUVDT
[2] [1]	**27 ABCDEF**	ACFJOU AFODYG
[2] [1]	**31 ELEPHANT**	EKRZJV JPIOLVWE
[2] [1]	**999999 MOON**	MKAFSR YDVV

Question 2(b) [3 marks available]

[3] LKBXIY

Question 2(c) [4 marks available]

[2] No

At most [2] marks, taken from the following arguments. Both marks must be from the same argument; note that the last argument on its own is only worth [1] mark.

[1] The i^{th} letter to be encrypted will take place after i-1 rotations of the dial.
[1] The i^{th} letter of the alphabet will be encrypted to the i+i-1^{th} letter on the second dial.
[1] The i^{th} encrypted letter will always match the i+13^{th} encrypted letter. [The student can either give a specific i in the range 1 to 13, or express the range.]

or

[2] The word is always encrypted as the following positions from the second dial:
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25,
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25.

or

[1] Any explicit example of an encryption of the word, for any second dial.
[The most likely example, from the ordered second dial, would be ACEGIKMOQSUWYACEGIKMOQSUWY]
[1] Any re-mapping of the letters on the second dial would just result in the corresponding re-mapping of this encrypted word.

or

[1] The word will always be encrypted as 13 letters which are then repeated.

Question 2(d) [4 marks available]

[4] 1260

Question 3(a) [24 marks available]

Each test for 3(a) consists of a single digit d, followed by a d digit number. The output is always a single integer.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**6** **461235**	6
[2]	**6** **412365**	3
[2]	**9** **123456789**	0
[2]	**1** **1**	0
[2]	**3** **132**	1
[2]	**5** **41235**	3
[2]	**6** **254631**	5
[2]	**7** **2756413**	9
[2]	**7** **7521436**	10
[2]	**8** **51438672**	8
[2]	**8** **51432687**	13
[3]	**9** **547389126**	19

Question 3(b) [4 marks available]

[2] 7
[2] 16

Question 3(c) [6 marks available]

[2] 26
[4] 2620

VERSION 1

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type in a single integer. The output should be a single number which is required to match exactly.

Tests *must* terminate in 1 second to receive marks.

[1]	**5**	6
[1]	**9**	11
[1]	**33**	44
[1]	**84**	88
[1]	**45653**	45654
[1]	**36460000**	36466463
[1]	**24355343**	24366342
[1]	**123450000**	123454321
[1]	**234567890**	234575432
[1]	**678999876**	679000976
[1]	**99999999999999** (14 9s)	100000000000001 (13 0s)
[1]	**999999999999999** (15 9s)	1000000000000001 (14 0s)
[2]	**123456789000000000** (9 0s)	123456789987654321
[2]	**987654321123456789**	987654322223456789
[2]	**1234567890000000000** (10 0s)	1234567890987654321
[2]	**9876543210123456789**	9876543211123456789
[2]	**9876543219123456789**	9876543220223456789

Additional marks are available for general program behaviour:

[1] Program inputs one integer.
[1] For each a test a single integer is output.
[1] First 7 tests (up to 24355343) terminate without without crashing / hanging.

Question 1(b) [2 marks available]

[2] 11,000,000,000

Question 1(c) [4 marks available]

[4] 9030

Question 2(a) [24 marks available]

There are 12 tests used to check program 2(a). For each test you will need to type in an integer followed by a word consisting of *uppercase* letters, followed by another integer.

For each test you should see a co-ordinate output. The X- and Y- co-ordinate must both be correct to score marks. Other formatting (e.g. brackets and a comma) are optional.

Tests *must* terminate in 1 second to receive marks.

[2]	**8 FL 2**	-1 1
[2]	**100 FLF 591**	9 -6
[2]	**39 LLLRFFF 50**	1 -1
[2]	**100 LLRR 5000**	-2500 0
[2]	**10 LLRFLR 5000**	-3 -3
[2]	**1 F 1000**	0 1000
[2]	**100 L 1000**	-4 -12
[2]	**39 LRFRRF 5000**	0 -1
[2]	**1 L 999**	0 -1
[2]	**9 LLR 5000**	0 -2
[2]	**100 R 1000**	0 -12
[2]	**10 FFRFRFFRRR 100**	1 1

Question 2(b) [2 marks available]

[2]

		3	4	5
	1	2		6
	⬇			7

Question 2(c) [3 marks available]

[3] 440

Question 2(d) [5 marks available]

[5] 1007

Question 3(a) [24 marks available]

Each test for 3(a) consists of an integer followed by a word consisting of *uppercase* letters. The output is always a single integer.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**1 A**	1
[1]	**2 AB**	1
[1]	**2 BA**	1
[1]	**4 C**	5
[1]	**4 AB**	0
[1]	**6 FED**	5
[1]	**8 HGFEDCBA**	1
[1]	**8 H**	429
[1]	**8 FED**	28
[1]	**8 FEH**	42
[1]	**12 LKJI**	1430
[1]	**13 MH**	13260
[2]	**14 N**	742900
[2]	**16 KHF**	5508
[2]	**16 FEDCBA**	1
[2]	**18 FRN**	0
[2]	**18 QPON**	2674440
[2]	**18 R**	129644790

Question 3(b) [2 marks available]

[2] BOI, IBO, IOB, OBI, OIB

(**Supplementary:** If there are only 4 of these values given, score [1] mark. Do not score *any* marks if BIO is given.)

Question 3(c) [4 marks available]

[1] Both original block-chains are in reverse alphabetical order.

[1] If the combination of the two block-chain is not a block-chain / contains three letters in alphabetical order, there must have been two letters in alphabetical order in one of the original block-chains as neither original block-chain can contain three letters in alphabetical order.

[1] Neither original block-chain can have two letters in alphabetical order, otherwise those two letters plus any letter from the other block-chain will give three letters in alphabetical order.

[1] If both block-chains are in reverse alphabetical order, if a pair of letters appears in alphabetical order in the combination one letter must have come from each block-chain.

Question 3(d) [5 marks available]

[1] ASRQPONMLKJIHGFEDCB

[4] RPSMJQHOFEDNLKICGBA

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type in a Roman numeral followed by an integer. The output should be a pair of integers both of which are required to match exactly.

Tests *must* terminate in 1 second to receive marks.

[1]	**MMXX 3**	6 2
[1]	**C 1**	1 0
[2]	**V 1**	1 1
[2]	**III 2**	2 1
[2]	**I 7**	6 2
[2]	**MMXX 8**	30 12
[2]	**II 20**	101 37
[2]	**IV 20**	121 46
[2]	**MDCLX 30**	1795 695
[2]	**M 50**	2858 1103
[2]	**V 50**	2858 1104
[2]	**MMDCCCLXXXVIII 50**	19013 7333

Additional marks are available for general program behaviour:

[1] Program inputs a Roman numeral and an integer.
[1] For each a test two integers are output.
[1] All tests terminate without without crashing / hanging.

Question 1(b) [2 marks available]

[1] 4
[1] II, III, IV, IX

(**Supplementary:** If a number lower than 4 is given and the corresponding examples are all from the list, score [1] mark)

Question 1(c) [4 marks available]

[4] 3919

Question 2(a) [24 marks available]

There are 9 tests used to check 2(a). For each test you will need to type a word consisting of *uppercase* letters, followed by two integers.

For each test you should see several lines of uppercase letters. All but the last line should be treated as a single block, and every line in the block needs to match (without additional or omitted letters) to score. Letters on each line *should* be in alphabetical order, however a student whose letters are not alphabetical order can still be awarded the marks.

The last line is graded separately and both letters on this line are required to match and must be in the order given on this mark scheme. A blank line has been added before the last line for clarity below – this will *not* be present in student output.

Tests *must* terminate in 1 second to receive marks.

Test 1 **A 1 2**

```
      BC
      A
[1]   A

[1]   BA
```

Test 2 **C 4 8**

```
      C
      C
[1]   AB

[1]   BB
```

Test 3 **BC 4 9**

```
      B
      AC
[1]   BD
      C

[1]   CD
```

Test 4 **BBACC 10 1010**

```
      BC
      ADE
[1]   AFG
      B
      B
      C
      C

[2]   FG
```

Test 5 **`ABCD 50 51`**

```
      BE
      AC
[1]   BD
      CF
      A
      D

[2]   CD
```

Test 6 **`FEDC 2020 876543`**

```
      F
      E
[1]   DF
      CE
      BD
      AC

[2]   CB
```

Test 7 **`AABGB 5000 9999`**

```
      BCD
      AFG
[1]   A
      A
      G
      B
      BE

[2]   AB
```

Test 8 **`CDCDCD 2020 876543`**

```
      C
      D
[1]   ADEG
      BCFH
      C
      D
      C
      D

[2]   DH
```

Test 9 **`FFFFFFFF 512 1022`**

```
      F
      F
      F
      F
      F
[1]   ABCDEGHIJ
      F
      F
      F
      F

[2]   JA
```

Question 2(b) [2 marks available]

[1] A
[1] AAAA

Question 2(c) [4 marks available]

In the following, "odd exit" is used as a shorthand for "an exit which has been left an odd number of times". Similarly for "even exit".

At most [2] marks, taken from the following arguments. Both marks must be from the same argument.

[1] If the number of odd exits in the room is even, the room has been visited an odd number of times. Otherwise it has been visited an even number of times.

[1] The first time a spy is in a room the number of odd exits is zero (even).

or

[1] If the number of even exits matches whether the the number of exits from the room is even, the room has been visited an odd number of times. Otherwise it has been visited an even number of times.

[1] The first time a spy is in a room the number of even exits equals the number of exits in the room.

At most [2] marks from the following (which can also be phrased in terms of the even exits):

[1] Every time the spy leaves the room either an odd exit becomes even, or an even exit becomes odd.
[1] Every time the spy leaves the room the number of odd exits either increases or decreases by 1.
[1] Every time the spy leaves the room the number of odd exits switches between being odd and even.

Question 2(d) [4 marks available]

[4] 96

Question 3(a) [27 marks available]

Each test for 3(a) consists of a line containing three integers, followed by a line containing a single integer. The output should be a string of uppercase letters.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**`1 1 1`**
	`1`
		`A`

[1]	**`1 12 12`**
	`1`
		`AAAAAAAAAAAA`

[1]	**`2 1 8`**
	`2`
		`BABABABA`

[1]	**`2 2 7`**
	`1`
		`AABAABA`

[1]	**`2 2 12`**
	`400`
		`BBAABBAABABA`

[1]	**`2 3 10`**
	`500`
		`BBABBBAABB`

[1]	**`6 3 10`**
	`1234567`
		`AAEDFEEEFD`

[1]	**`6 2 8`**
	`567890`
		`CCADDCFA`

[1]	**`8 8 8`**
	`16000111`
		`HFACCBFG`

[2]	**`6 2 12`**
	`1666111000`
		`FFBBFFCCBCFA`

[2]	**`12 2 12`**
	`9876543210`
		`AADACFGGLBJI`

[2]	**`10 10 10`**
	`7654334567`
		`HGFEDDEFGG`

[3]	**`7 2 12`**
	`11223344556`
		`GFEGAFFEADAF`

[3]	**`10 3 12`**
	`223344556677`
		`CCFCFJIFIGHE`

[3]	**`11 4 12`**
	`334455667788`
		`BBJKBCJGCGFE`

[3]	**`12 5 12`**
	`3344554433999`
		`EGACGKCJHCFI`

Question 3(b) [3 marks available]

[1] 39
[2] 29947

Question 3(c) [5 marks available]

[1] p is odd.
[1] $r \leq q$

At most [3] marks from the following:

[1] The n^{th} plan must be the middle plan in the list of plans, otherwise its position would be different in the alphabetical and reverse alphabetical lists.
[1] For the list of plans to have a middle position it must have an odd number of elements.
[1] The number of elements in the plan which contain more than one letter is even.
[1] The number of elements in the plan which contain a single letter is odd $r \leq q$, otherwise it is even (zero).

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type in two *uppercase* strings. The output should be three YES/NO answers; *all* are required to match (in the order given). YES/NO answers may appear one per line or separated by spaces.

Tests *must* terminate in 1 second to receive marks.

[1]	**DE C**	NO YES YES
[2]	**A A**	YES YES NO
[2]	**A B**	YES YES NO
[2]	**B A**	YES YES YES
[2]	**AB CD**	NO NO NO
[2]	**BEFCD A**	NO YES YES
[2]	**GEA DBCF**	NO NO YES
[2]	**EFCD GAB**	YES YES YES
[2]	**ECBDFA LKJIHG**	YES NO NO
[2]	**BDIGEF HCA**	NO NO YES
[2]	**JKHGIL ADFEBC**	NO NO YES

Additional marks are available for general program behaviour:

[1] Program inputs two strings.
[1] For each a test three YES/NO are output.
[1] All tests terminate without without crashing / hanging.

Question 1(b) [3 marks available]

[3] BDCA, CBDA, CDAB, DBAC, DACB

(**Supplementary:** Score [2] marks if only 4 correct strings are shown. Do not award marks if there are any incorrect strings.)

Question 1(c) [5 marks available]

[5] 343,059,613,650

Question 2(a) [24 marks available]

There are 10 tests used to check 2(a). For each test you will need to type in two integers on one line, followed by another line of between 1 and 5 integers.

For each test you should see several lines of numbers. All but the last line should be treated as a single block, and every line in the block needs to match exactly (and in order) to score.

The last line is graded separately and the number must match to score. A blank line has been added before the last line for clarity below – this will *not* be present in student output.

Tests *must* terminate in 1 second to receive marks.

Test 1
```
2 5
16 2
```
[1]
```
1
0
```
[1]
```
8
```

Test 2
```
1 1
1
```
[2]
```
0
```
[2]
```
4
```

Test 3
```
1 3
1
```
[2]
```
1
```
[2]
```
6
```

Test 4
```
1 36
3
```
[1]
```
17
```
[1]
```
21
```

Test 5
```
1 1250
15
```
[1]
```
989
```
[1]
```
216
```

Test 6

```
2 2
5 7
```

[1]
```
0
0
```

[1]
```
5
```

Test 7

```
2 16
11 13
```

[1]
```
0
0
```

[1]
```
15
```

Test 8

```
2 2000
5 13
```

[1]
```
474
478
```

[1]
```
275
```

Test 9

```
3 2222
13 19 23
```

[1]
```
207
220
222
```

[1]
```
292
```

Test 10

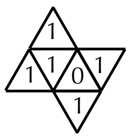
```
5 4999
2 3 5 7 11
```

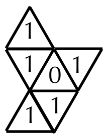
[1]
```
84
73
71
91
135
```

[1]
```
382
```

Question 2(b) [4 marks available]

For the following answers additional *empty* triangles can be drawn but no additional *filled* triangles can appear. All values need to match.

[2] 

[2] 

Question 2(c) [3 marks available]

[3] 51

Question 2(d) [5 marks available]

[5] 377

Question 3(a) [24 marks available]

Each test for 3(a) consists of a string of *uppercase* letters. The output should be a single number.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**ACBD**	6
[1]	**A**	1
[1]	**AB**	2
[1]	**BA**	3
[1]	**ACB**	5
[1]	**DCBA**	8
[2]	**ABCDEFGH**	8
[2]	**BACDE**	6
[2]	**AEDBC**	12
[2]	**BACDEFGH**	9
[2]	**CFBGAHDE**	15
[2]	**GADEFBC**	16
[2]	**GCFBEDA**	21
[2]	**CHDGABFE**	23
[2]	**AHGEFDCB**	27

Question 3(b) [3 marks available]

[3] BACDE, BCADE, BCDAE, BCDEA

(**Supplementary:** Score [1] mark if only 3 correct strings are shown. Do not award marks if there are any incorrect strings.)

Question 3(c) [5 marks available]

[5] 84

(**Supplementary:** Score [3] marks for the answer 72)

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type in a single *uppercase* string. The output should be a single string. Every letter in the output must be correct for the marks to be scored. If the output is in lowercase but the letters are otherwise correct, the marks can be awarded.

Tests *must* terminate in 1 second to receive marks.

[1]	**ESVNMCW**	ENCRYPT	
[2]	**H**	H	
[2]	**ZT**	ZT	
[2]	**IO**	IF	
[2]	**AA**	AZ	
[2]	**BIO**	BGF	
[2]	**TCCCB**	TIZZY	
[2]	**CRFZEXR**	CONTEST	
[2]	**CONTEST**	CLYFKNA	
[2]	**ABCDEFGHIJ**	AAAAAAAAAA	(10 As)
[2]	**STRAWBERRY**	SAXIVECMZG	

Additional marks are available for general program behaviour:

[1] Program inputs a string.
[1] For each test a string is output.
[1] All tests terminate without crashing / hanging.

Question 1(b) [2 marks available]

[2] Any five letter string whose first four characters are ZZZZ is a valid answer; i.e. ZZZZA, …, ZZZZZ. Only a single string needs to be given.

Question 1(c) [2 marks available]

[2] 104

Question 1(d) [4 marks available]

[4] 4394

Question 2(a) [27 marks available]

There are 15 tests used to check 2(a). For each test you will need to type in two lines, each containing two integers.

For each test you should see two lines output each with a single integer. Both integers need to be correct to score marks.

Tests *must* terminate in 1 second to receive marks.

Test 1
```
9 3
3 1
```
[1]
```
6
6
```

Test 2
```
2 11
0 0
```
[1]
```
0
0
```

Test 3
```
1 1
1 0
```
[2]
```
1
2
```

Test 4
```
1 1
4 0
```
[2]
```
1
3
```

Test 5
```
2 23
28 0
```
[2]
```
9
8
```

Test 6
```
11 5
20 0
```
[2]
```
17
7
```

Test 7
```
25 24
999 0
```
[2]
```
1
24
```

Test 8
```
2 11
0 1
```
[1]
```
2
2
```

Test 9
```
16 25
7 3
```
[2]
```
14
9
```

Test 10
```
25 15
3 13
```
[2]
```
10
13
```

Test 11
```
18 6
53 3
```
[2]
```
9
13
```

Test 12
```
25 24
11 3
```
[2]
```
7
16
```

Test 13
```
7 1
73 3
```
[2]
```
7
7
```

Test 14
```
1 2
41 15
```
[2]
```
9
2
```

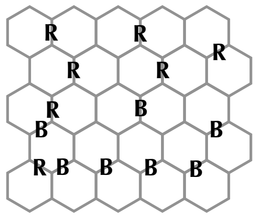
Test 15
```
1 14
31 19
```
[2]
```
5
3
```

Question 2(b) [3 marks available]

To score marks, all 14 edges (indicated below) need to be clearly indicated as *red* (R below) or *blue* (B below). No other edges are to be marked as *red* or *blue*.

The method for labelling the edges does not have to match the solution below so long as it is clear which edges are red and blue.

[3]



Question 2(c) [4 marks available]

[2] Minimum: 1
[2] Maximum: 28

Question 3(a) [25 marks available]

Each test for 3(a) consists of a string of *lowercase* letters followed by an integer. The output should be a string of *uppercase* letters.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**`cabd 5`**	`BCAA`
[2]	**`a 1`**	`A`
[2]	**`dacb 2`**	`BDCA`
[2]	**`fedcba 1`**	`FEDCBA`
[2]	**`badcef 90`**	`BADCEF`
[2]	**`dabcefgh 5000`**	`BBDAEFBH`
[2]	**`hefbdciajg 125`**	`HDFDBCJAGH`
[2]	**`bcadefghi 49999`**	`CAADBDBDD`
[2]	**`bcdefghijak 1000000`**	`JAABCAACFAA`
[2]	**`acbdefghijk 12345678`**	`ACBDCAEGDED`
[3]	**`abcdeghfklijnmop 2800700600`**	`ABCDDHEDKKAANFMH`
[3]	**`abcdefghijklmnop 12345678901234`**	`ABACAEFHBFJAMLCB`

Question 3(b) [2 marks available]

[2] ghcdabefij

Question 3(c) [3 marks available]

[3] 120

Question 3(d) [4 marks available]

[4] 6,165,817,614,720

Question 1(a) [24 marks available]

For each test of the program for 1(a) you need to type in a single number. The output should be one or more numbers. All numbers in the output must be correct for the marks to be scored; numbers in the output *may* appear in any order.

Tests *must* terminate in 1 second to receive marks.

[1]	**100**	89 8 3
[2]	**1**	1
[2]	**832040**	832040
[2]	**4**	3 1
[2]	**623**	610 13
[2]	**12**	8 3 1
[2]	**834629**	832040 2584 5
[2]	**33**	21 8 3 1
[2]	**2023**	1597 377 34 13 2
[2]	**5000**	4181 610 144 55 8 2
[2]	**514228**	317811 121393 46368 17711 6765 2584 987 377 144 55 21 8 3 1

Additional marks are available for general program behaviour:

[1] Program inputs a number.
[1] For each a test one or more numbers are output.
[1] All tests terminate without without crashing / hanging.

Question 1(b) [2 marks available]

[2] 832,040

Question 1(c) [2 marks available]

[2] 18,424

Question 1(d) [4 marks available]

[4] 2,998,107,957

Question 2(a) [23 marks available]

There are 12 tests used to check 2(a). For each test you will need to type in a string consisting of two *capital* letters

For each test you should see a single integer, which needs to be correct to score marks.

Tests *must* terminate in 1 second to receive marks.

[1]	**FF**	7
[2]	**XX**	6
[2]	**TT**	8
[2]	**II**	10
[2]	**GX**	13
[2]	**QP**	14
[2]	**MW**	15
[2]	**ZY**	16
[2]	**NU**	16
[2]	**AS**	16
[2]	**JL**	18
[2]	**VI**	19

Question 2(b) [2 marks available]

[2] 1

Question 2(c) [5 marks available]

[2] `III` : 122
[3] `LIV` : 672

Question 2(d) [4 marks available]

[4] 84

(**Supplementary:** The answer 161 scores [3] marks)

Question 3(a) [23 marks available]

Each test for 3(a) consists of two lines, each containing four numbers. For each test you should see a single integer output.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

```
[1]  12 0 3 4          3
     1 32 4 0

[2]  12 0 34 0         0
     12 0 34 0

[2]  1 23 0 4          1
     1 2 0 43

[2]  1 2 3 4           1
     0 2 3 41

[2]  1 2 3 4           3
     1 3 2 4

[2]  4 3 2 1           4
     3 2 4 1

[2]  0 0 241 3         4
     0 4 1 32

[2]  0 4 0 123         5
     12 3 4 0

[2]  1234 0 0 0        6
     4 213 0 0

[2]  12 3 4 0          7
     21 0 34 0

[2]  1234 0 0 0        7
     0 0 0 1234

[2]  1234 0 0 0        9
     4321 0 0 0
```

Question 3(b) [2 marks available]

[2] 288

Question 3(c) [5 marks available]

[2] After 2 moves, 73 states
[3] After 4 moves, 375 states

Question 3(d) [4 marks available]

The following numbers can be given in any order:

[4] 1, 1, 2, 3, 3, 3, 8, 8

Question 1(a) [25 marks available]

For each test of the program for 1(a) you need to type in two numbers. The output should be a single digit. The output must be correct for the marks to be scored.

Tests *must* terminate in 1 second to receive marks.

[1]	**999 11**	1
[1]	**1 1**	1
[1]	**98765 1**	9
[1]	**1 5**	5
[1]	**1 50**	3
[1]	**1 500**	0
[2]	**1 123456**	6
[2]	**1 9988776655443322**	6
[2]	**123 1000**	4
[2]	**123 87654321**	2
[2]	**1122 3456543**	7
[2]	**345 9988776655443322**	8
[2]	**22334455667788 11**	7
[2]	**12345678912345 987654321987**	9

Additional marks are available for general program behaviour:

[1] Program inputs two numbers.
[1] For each test a single digit is output.
[1] All tests terminate without crashing / hanging.

Question 1(b) [2 marks available]

[2] 12

Question 1(c) [5 marks available]

[2] Positions 223 – 227
[3] Positions 1,677,777,779 – 1,677,777,787

Question 2(a) [25 marks available]

There are 13 tests used to check 2(a). For each test you will need to type in a string consisting of *capital* letters and brackets, followed by an integer.

For each test you should see a single integer, which needs to be correct to score marks.

Tests *must* terminate in 1 second to receive marks.

[1]	**E(OE) 3**	350
[2]	**T 1**	1
[2]	**O 123456**	246911
[2]	**EE 1**	8
[2]	**TOE 1000**	354
[2]	**(TO)E 555**	266
[2]	**T(OE) 2000**	1430
[2]	**(EO)(OE) 5000**	2559830
[2]	**E(OO)E 4433**	2269458
[2]	**E(O(OE)) 5566**	182375894
[2]	**T(OO(EO))T 987654**	1108
[2]	**EEEOEEEOEEEO 12345**	103552892655
[2]	**EE(O(EEEE))O 12345678**	424194248808595199

Question 2(b) [2 marks available]

[2] 20

Question 2(c) [5 marks available]

[5] 41

(**Supplementary**: The answer 44 scores [3] marks)

Question 3(a) **[25 marks available]**

Each test for 3(a) consists of a single line containing a string of *capital* letters. For each test you should see a single integer output.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

[1]	**BAB**	3
[2]	**A**	1
[2]	**GA**	38
[2]	**DEAD**	980
[2]	**R**	10843
[2]	**BIO**	579377
[2]	**DOG**	824034
[2]	**ALGAE**	346221
[2]	**NICE**	15671736
[2]	**ALGOL**	44097895090
[3]	**ZY**	1141042340074
[3]	**LIKELY**	444474391799316244

Question 3(b) **[2 marks available]**

[2] ABABAB

Question 3(c) **[4 marks available]**

[4] 954,658

Question 3(d) **[5 marks available]**

[3] The largest difference in word length is 32.
[2] 1 pair of words have this difference.

Question 1(a) **[25 marks available]**

For each test of the program for 1(a) you need to type in a single integer. The output should be one, two or three integers which *can* be given in any order. The output must be correct for the marks to be scored.

Tests *must* terminate in 1 second to receive marks.

[1]	**1031**	1 101 929
[1]	**1**	1
[1]	**55**	55
[1]	**12321**	12321
[2]	**10**	1 9
[2]	**894**	6 888
[2]	**10219**	1221 8998
[2]	**21802**	10901 10901
[2]	**987654**	51015 936639
[2]	**21**	1 9 11
[2]	**109812**	22 989 108801
[2]	**109988**	88 10001 99899
[2]	**219803**	1 109901 109901

Additional marks are available for general program behaviour:

[1] Program inputs one number.
[1] For each test one, two or three numbers are output.
[1] All tests terminate without crashing / hanging.

Question 1(b) **[2 marks available]**

It is necessary to list all of the following triplets of numbers to score [2]. They can be given in any order.

[2] 1 9 44
2 8 44
3 7 44
4 6 44
5 5 44

(**Supplementary**: So long as *no* incorrect triplets are given, if some of the solutions are missing, award [1] mark)

Question 1(c) **[5 marks available]**

[5] 266948

Question 2(a) **[25 marks available]**

There are 11 tests used to check 2(a). For each test you will need to type in three integers.

For each test you should see a pair of integers, which both need to be correct to score marks.

Tests *must* terminate in 1 second to receive marks.

[1]	**3 5 5**	2 1
[2]	**1 100 200**	1 0
[2]	**2 1 1**	1 1
[2]	**2 7 23**	1 0
[2]	**4 8 94**	3 3
[2]	**6 213 1040**	3 6
[2]	**7 2025 7**	7 6
[3]	**8 19 22**	4 10
[3]	**9 510 4152**	9 5
[3]	**10 3548 872**	9 15
[3]	**10 4999 4999**	5 8

Question 2(b) **[2 marks available]**

```
[2] RGR
    RGG
    GRR
```

Question 2(c) **[3 marks available]**

[3] 25 41

Question 2(d) **[5 marks available]**

[5] 20 17

Question 3(a) [25 marks available]

For each test for 3(a) you will need to enter two lines of integers; the first with a single integer and second containing between 1 and 4 integers. For each test you should see a single integer output.

There are no marks for incorrect answers, and tests *must* terminate in 1 second to receive marks.

```
[1]  2
     1 1                     7

[2]  1
     1                       3

[2]  2
     2 4                     9

[2]  2
     1 3                     11

[2]  3
     100 100 100             16

[2]  3
     7 14 14                 24

[2]  3
     16 32 128               45

[2]  3
     10 11 12                89

[2]  3
     43 401 4757             101

[2]  4
     1 1 1 1                 35

[2]  4
     1 2 4 8                 133

[2]  4
     5 41 83 137             891

[2]  4
     1483 1801 3407 4999     1781
```

Question 3(b) [2 marks available]

It is necessary to list all of the following numbers to score [2]. They can be given in any order.

[2] 0.0
0.5
0.75
1.0
1.25
1.5
2.0
2.5
3.0

Question 3(c) [6 marks available]

[2]	17	(2 fuses)
[4]	163	(3 fuses)