

# The British Informatics Olympiad 2011

Eliot Ball – Greenhead College

The British Informatics Olympiad is part of the family of science Olympiads which cover various subjects including the three main branches of experimental science, and mathematics. The BIO is focussed on the design and implementation of algorithms, placing it in the shared area between computer science and mathematics. The purpose of all of the science Olympiads is to encourage students to take an interest in the subjects they enjoy, and to offer the opportunity for the best students to compete in the international versions of the Olympiads. The four best competitors in the BIO are selected to represent Great Britain at the International Olympiad in Informatics.

## The First Round

The first round is held at any time within a period of approximately a month during November and December in the competitor's school or college. It consists of a three hour exam with three questions, and is usually taken by around one thousand students. I entered because I thought it looked interesting, and there was a chance to progress to the final. I used Python because it is a modern, simple and powerful language that allows you to write code quickly without worrying too much about the details of how the computer does what you are asking it to do. Programs are marked by the teacher in college, and each test case had a time limit of two seconds this year.

The first question, entitled 'Fibonacci Letters,' was quite simple. Each letter in the alphabet is given a value with A = 1 and Z = 26 and the task is to write a program that accepts two letters as the first two elements of a sequence and a positive integer  $n \leq 2^{31}$  and returns the  $n$ th element of the sequence, by taking element  $n$  to be the sum of element  $n - 1$  and element  $n - 2$ . (ie the Fibonacci Sequence) This took me about 15 minutes and I scored full marks for this question. After Z the alphabet wrapped around so you had to take the number modulo 26, noting that there is an issue with A = 1 rather than A = 0.<sup>1</sup> There were a few short written questions asking you to solve a couple of simple equations with letters rather than numbers and a question requiring you to notice that the sequence has a constant period and hence calculate the  $10^{18}$ th element of the sequence starting with {C, C}. This would take too long by simply iterating through the terms of the sequence. (Also, most programming languages use 32-bit integers by default, which do not have a large enough range to store  $10^{18}$  for the iteration variable)

The second question was an implementation question, which did not require the design of an algorithm. A method for shuffling a deck of cards was described, and then the rules for a fairly complicated single-player game were explained, along with three strategies for playing the game. The task was to write a program that would take six integers required to perform the shuffle, and then play the game by the three strategies, displaying the results. This took a while but I gained full marks for my program. I got some of the written questions right, and some wrong. The questions focussed on permutations of cards and what was possible in different situations in the game.

The third question was a problem solving question, where an algorithm had to be designed to solve a problem. The question defined an 'upside down number' as a number where the  $i$ th digit from the left of the number and the  $i$ th digit from the right sum to ten. The program would be presented with a positive integer  $n \leq 2^{31}$ , and had to return the  $n$ th upside down number (in order of increasing magnitude). Unfortunately I

---

<sup>1</sup> Because A = 1, if you take 26 mod 26 you get 0, so Z would be converted to letter 0, which we don't have a definition for. One way to deal with this problem is to subtract one, take the modulo and then add one afterwards.

got a bit distracted and took the wrong approach entirely, looking for a numerical pattern in how the numbers increased. A correct approach was to look at how to change the digits in each number to get to the next one, treating it as a permutations problem. In the end I wrote a brute force solution which simply tried all integers in increasing order until it had found the correct number and returned the last one. Since the bounds were as high as they were this ran too slowly for most test cases. I have since written a solution using the permutations approach which ran within the specified time on all test cases.

Because of my error on the last question, I had abandoned hope of progressing, so when I received the email inviting me to the final I was extremely surprised, but it turned out that the paper had been considered harder than normal that year anyway.

### **The Final**

The final was held at Trinity College Cambridge from 26 – 28 March. We travelled to Cambridge on the Friday afternoon and I collected the key for my room. Our accommodation was in the newer part of the college, Burrell's Field, across the road from the old part of college, but we walked between the two several times during the weekend. I had found another competitor on an internet forum a few weeks before so we met in the town centre and discussed the training materials we had been encouraged (instructed) to work through before the final. When we first met the other competitors, the conversation was a bit strained (computer scientists are not known for their social skills, and some of the contestants did not seem keen to break the mould) but after the main organiser, Dr Richard Forster, arrived, he got the conversation going a bit. He explained that whenever he came across anything interesting, and particularly any intriguing mathematical law or principle, he writes it in a notebook and then uses these for inspiration when writing problems. He said that the third question on the first round paper had originated from certain lines of sheet music that, when rotated through  $180^\circ$ , would produce the same sound when played. We then left for dinner, which was not spectacular in culinary terms, but this was more than made up for in my opinion by being eaten in the candlelit Great Hall. After dinner we returned to the part of the college where our accommodation and the computer rooms were situated.

The first part of the evening was taken up by a simple programming task which did not count towards the score for the weekend and was mainly intended to allow us to familiarise ourselves with the programming environment available for the contest. The programming languages available were C, C++ and Delphi, but nobody used Delphi, and nobody who did well used C. I used C++, and the environment was Microsoft Visual Studio 2005. I was used to a later version (2010) but it was essentially the same. The programming task consisted of finding the  $n$ th number (in increasing magnitude) where any substring of digits taken from the left hand side of the number would be prime. Prime checking by trial division is a short piece of code that most competitors at the final should have been able to quote straight out of their heads or write with little thought, and was suitably fast for this problem. More ingenuity was involved in generating the numbers without wasting time, and this involved using previous small results again when computing bigger results. If a faster prime checking algorithm was needed then most competitors should have been able to implement a process called the Sieve of Eratosthenes (also known simply as a 'prime sieve' for obvious reasons) which generates a list of primes much more quickly than checking each one for divisors, but in this case the prime checking algorithm had a negligible effect on the runtime. This problem took most people about twenty minutes. Afterwards we went to the 'party' room and played cards all evening. The game involved introduction of new rules during the game, with the order of application of these rules being important. This led one contestant to comment during an argument over what to do in one situation, that the problem was that 'function composition isn't commutative', which raised a few smiles. I'm sure parties do happen in that room, but not at the BIO final! Some previous team members were there, along with first and second year

students of mathematics and computer science, and third year physics PhD student. They were helping to run the final.

The next morning we went for breakfast which was once again in the Great Hall. It was a traditional English Cooked Breakfast and was enjoyed by most people. The discussion over breakfast was dominated by the discussion of mathematical problems, and two of the competitors who were also finalists in the British Mathematical Olympiad along with a few others were discussing a problem with a first year maths student. This problem turned out to be of a similar difficulty to the problems found at the International Mathematical Olympiad, but it was a graph problem so those of us with most of our experience in computer science were able to follow. After breakfast we did two written papers which took about two hours. The first was a structured question and answer paper on interleaving lists, which included some questions on permutations and combinations and informal proof questions asking us to explain why certain properties are true. The second paper was a free essay question on image recognition. I was happy with my performance on both papers, especially the essay as I knew a fair bit about that field of computer science.

We then returned to the Great Hall for lunch which turned out to be just as underwhelming as the dinner from the night before, but the discussion of various logic problems continued, which was interesting even though it effectively barred us from getting to know each other on a more human level. After lunch the main part of contest was held in the computer room.

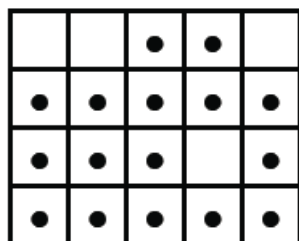
### **The Programming Session**

The programming session was held in the computer room back over at Burrell's Field. The ventilation in the room (or lack thereof) caused the session to actually become physically as well as mentally challenging, not something you expect from a science Olympiad final! There were four problems given to us in a random order at the beginning of the session (the order was explicitly stated to not be indicative of difficulty) and we were given 4 hours and 45 minutes to do as much as we could. We were informed that completing two problems fully would be considered far more favourably than half solutions to all four problems, and so we should not spread ourselves too thinly or give up on problems after a relatively short time. The problems were entitled Earth, Air, Water and Fire, and each problem took the form of a story with an underlying computational problem. There was a time limit of approximately five seconds for each test case, which on modern computers corresponds to up to around five billion operations (in compiled languages like C++, and depending on how expensive the operations are). This is useful to bear in mind when deciding how clever your algorithm needs to be. For the first half hour of the time we were not allowed to type, which was to encourage us to think about the problems and how to efficiently solve them instead of diving straight into the programming without developing a proper solution first.

The first problem that I worked on was Earth, which explained that a will had been hidden at some point around the equator (I didn't really pay much attention to the details of the story in the contest) and a special assignment scheme had been used to number the parts of the equator. The equator had been split into  $n$  sections, which had been numbered according to the alphabetical order of the digits of each number written out in words. For example, 123 would be written as ONETWOTHREE using English digit names. If the equator was separated into 100 sections, the first section would be 8, the second section 88, the third section 85 and so on, in alphabetical order. The program would be given two integers  $1 \leq m \leq n \leq 2^{31}$  and the names of the digits 0 through 9 (these were not necessarily the English names, and Richard said later that the last test case used the Dutch names instead) and the task was to output the number of the  $m$ th section when the equator is divided into  $n$  sections. The intelligent way to do this is to order the digits alphabetically and then use an approach to count all of the permutations in order until the correct one is found. Unfortunately I

could not get this implementation to work so I had to use a slightly more naïve algorithm. I took each integer from 1 to  $n$  and inserted it into a binary tree, sorting by the alphabetical ordering of the digits. The problem with this algorithm is that it requires  $O(n \log n)$  time<sup>2</sup> and  $O(n)$  space to build the tree and then traversal to the correct element takes  $O(m)$  time afterwards. The permutation approach only takes  $O(m)$  time and  $O(1)$  space for the whole thing, therefore being quite a bit faster. This was reflected in the actual speed of my program, which only ran within the time on two thirds of the test cases. This type of problem mainly just tests programming skill and isn't particularly applicable to real life situations.

I then turned to Water, which was much easier. The problem said that a large area of swamp land was to be reclaimed, and that there were stilts stuck in the swamp land at grid positions (ie integral coordinates). The people who were planning to do the reclaiming could take any square area with at least four stilts and with all sections under the area containing stilts. This example was given:



Here there are six reclaimable areas: one  $3 \times 3$  area and five  $2 \times 2$  areas. The task was to write a program that would, when given a grid with dimensions  $1 \leq x, y \leq 256$ , return the number of reclaimable areas. The first thing that struck me about this problem is that the dimensions are quite small. This is probably to allow the matrix to be stored without a stack overflow, but it made the brute force approach just about feasible. Looking at each cell in the grid, I checked all of the cells that would need to have stilts for that cell to be the top left corner of each of the sizes of reclaimable area. So starting at cell  $(0,0)$ , I checked the cells for a  $2 \times 2$  area below and to the right, and the cells for a  $3 \times 3$  area, and so on, and I did this for every cell on the grid. This is very slow and the optimisations are very obvious but because the dimensions are so small this brute force solution run in the required time on all of the test cases. We think that the algorithm takes  $O(n^5)$  time, which (while it is still polynomial time unlike a lot of brute force algorithms which tend to be  $O(n!)$  or  $O(2^n)$  time) is very slow. However, discussing this problem with Sean, the physics PhD student, he said that because not all of the dimensions run to the full 256, the worst case is probably under  $100^5$  operations, which is just fast enough for a five second time limit. Richard said when he ran my programs that the biggest test case did indeed only just complete. One test case was fast but wrong. I suspect this was a  $1 \times 1$  grid and I must have overlooked that case. A faster algorithm for this problem would exploit the fact that four  $2 \times 2$  squares overlapping on one square implies that there is a  $3 \times 3$  square there (and likewise for bigger sizes) so there is no need to explicitly check these. An algorithm exploiting this would be  $O(n^3)$  and Sean suggested that a faster algorithm could be achieved with more work. This problem is not really relevant to real life either, although problems related to this come up in scheduling (which is NP-complete and so would not have been asked at any programming contest) and similar situations.

---

<sup>2</sup> Big O notation:  $O(f(n))$  time means that as the input size  $n$  increases, the worst case time for the program to run grows with  $f(n)$  in this way: All terms in any sums are discarded except the fastest growing one, as are coefficients, so if the number of steps taken by a program is  $4 \times 2^n + 4n^3 + 6n^2$  we say that it runs in  $O(2^n)$  time because as  $n \rightarrow \infty$ , the worst term,  $4 \times 2^n$  dominates, and we ignore the coefficient because we are only interested in the type of growth, not the actual runtime. This allows us to easily compare algorithms as the input size becomes big enough that the growth type becomes important.

The third and final problem that I attempted, Air, described a strange pigeon racing game where birds race through a course that basically consists of a network of waypoints. For some reason some birds prefer to fly the second fastest route rather than the fastest, and the problem was, given an undirected, weighted graph describing the waypoints, find the length of the second fastest route from the start to the finish. Finding the fastest route is easily done with Dijkstra's Algorithm<sup>3</sup>, which serious competitors should have been able to remember, and a list of the edges involved in the fastest route can be found. The size of the network was small enough that time complexity in the problem was not a serious concern, although trying all of the possible routes would have been impossible for all but the very smallest test cases. Dijkstra's algorithm runs in  $O(|E| + |V| \log|V|)$  time, but the bounds were not big enough to make this a concern (and Dijkstra's is one of the best algorithms for shortest path anyway). My solution went through the edges in the shortest path blocking them one at a time, and recording the best path found each time (using Dijkstra's again). Unfortunately, this is not correct in all cases (and it is not really difficult to see why – imagine a graph with three vertices and two edges, where the start is the middle of a chain of the three: the second fastest route is to go backwards and then along the shortest path again, which would have been blocked in my algorithm). This meant that I only got half marks for this one. In hindsight I realise that a generalisation of Dijkstra's algorithm is probably what the question is looking for, but I haven't taken the time to work this out yet. Graph problems like this are becoming increasingly relevant in the modern world because of extremely large social networks which can be viewed as graphs with hundreds of millions of vertices. Algorithms for advertising, suggesting friends based on mutual friends and similar problems are all reduced to graph problems such as shortest paths, minimum spanning trees and various types of partitioning. Indeed, Facebook calls the Application Programming Interface that it opens up to third party developers the 'Graph API'.

I did not attempt Fire in the time available, but my score was fifth or so overall (a few other factors are taken into account when deciding the team, and our exact scores are not revealed).

### **Saturday Evening**

That evening we were all drained from the large amount of exams during the day. We all walked to Ask, an Italian restaurant in Cambridge to have a celebratory meal. The food was good but the restaurant was very busy and the food took a long time to be delivered. This did, however, give Sean a chance to give an overview of his research in cosmology, based on evidence that the Fine-Structure Constant ( $\alpha \approx 7.297 \times 10^{-3}$ ) is not actually a constant. He also explained that he had recently solved a problem in sociology known as the Discrete Core/Periphery Bipartitioning Problem, and written a paper on it, despite his main research area being far removed from this field! He was extremely enthusiastic about his research and gave a very positive image of academic science research in general.

After the meal we returned to Trinity College and a second year Computer Science student (whose name escapes me) showed us a game called the 'Game of a Thousand Blank White Cards'. This game begins with a deck of blank white cards in the middle of the table and each player holding three blank white cards. Each turn a player plays a card and takes a blank card from the deck. Blank cards don't do anything so players are instructed to design their own cards with the rules for the card being explained on the card itself (decided by the player). This leads to very creative gameplay that develops over the course of the game, and playing it with a highly mathematically/algorithmically minded group lead to some entertaining cards. Examples include Wikipedia, which allows any player to change any of the cards of the victim; the Halting Problem,

---

<sup>3</sup> Broadly speaking, Dijkstra's algorithm finds the shortest path between A and B by reaching out into the graph from A in all directions such that when it first hits B it will be the shortest path. Specifically Dijkstra's is for the single source shortest path in a weighted graph with non-negative edge weights.

which challenges the victim to determine whether the game will end before it actually does; and a proof of the Birch and Swinnerton-Dyer conjecture, which scores the person a million points from the Clay Mathematics Institute but causes them to be chased by angry cryptographers. Most of us went to bed by 3:30, but it should be noted that the clocks did go forward during that time!

### **Sunday Morning**

Relaxation was the name of the game for Sunday morning, and I took the opportunity to take some pictures of the spectacular college grounds. For lunch we were served lasagne in the Great Hall and some amazing apple juice (I mean this apple juice was really something). After lunch we went upstairs for the prize-giving ceremony. The representative from the sponsors, Lionhead Studios, offered us all jobs in the games industry (not for me but it's nice to have the offer) and the certificates were presented. This was done with the eight people not on the team in random order, the reserves in random order, and the main team in random order. I was the first reserve to be read out, but I was surprised enough to not be in the first eight that I wasn't bothered about my near miss of the team! We will hopefully be doing training between now and the summer where Richard promises that he teaches the algorithms part of a computer science degree in a weekend. I have another year yet, so I can hopefully make the main team next year! We were each given a stainless steel Memory Stick that doubles up as a bottle opener, a 1200 page textbook 'Introduction to Algorithms' and shirts, games and books from the sponsors. Overall taking part in the BIO has been an extremely interesting and rewarding experience, and I would encourage anyone to take part in the science Olympiad for the subject(s) they are best at and enjoy most. You've got nothing to lose and you may just end up with one or two expenses paid weekends away and a free holiday (not to mention a stronger university application). I have included links below to the relevant websites for this.

The British Informatics Olympiad	<a href="http://www.olympiad.org.uk">www.olympiad.org.uk</a>
The British Mathematical Olympiad	<a href="http://www.bmoc.maths.org">www.bmoc.maths.org</a>
The British Physics Olympiad	<a href="http://www.physics.ox.ac.uk/olympiad">www.physics.ox.ac.uk/olympiad</a>
The UK Chemistry Olympiad	<a href="http://www.rsc.org/education/schoolstudents/olympiad">www.rsc.org/education/schoolstudents/olympiad</a>
The British Biology Olympiad	<a href="http://www.biology-olympiad.org.uk">www.biology-olympiad.org.uk</a>

For anyone interested in programming, the Python programming language is an excellent place to start:  
[www.python.org](http://www.python.org)

Email me: [contact@eliotball.com](mailto:contact@eliotball.com)