Introduction to Computer Science Olympiad

In Computer Science Olympiads, you will be presented with a scenario that you will be expected to solve. The scenario will not be static with predefined numbers doled out to you, but rather it will merely conform to certain specified guidelines. Your program has to read in the input and formulate an answer *dependant* on the input. Of course the variable input means that it will not be as simple as merely printing out a single number.

One example of a question: Analyze a map of a country, and find the shortest travelling time between 2 specified landmarks, given the extensive transport system network the country has. The map could range anywhere in size from the microscopic Vatican City to the entire Unites States! Your program will have to work correctly on all possible maps and, since users hate lags, you must produce results unfairly fast without consuming much memory.

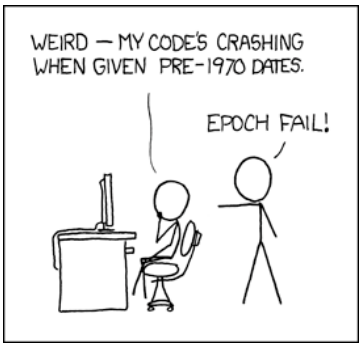
This is a more basic example that we will learn from: Devise an algorithm to find the median of N numbers. Even an easy problem such as this, will allow for good doses of learning and will prove to be much more difficult when we delve into it. Do not judge a problem by a cursory glance!

Why Computer Science Olympiad

Now before delving into the actual material and becoming Computer Science Olympians, you might wonder why you should brazenly undertake this risky venture. Firstly, it is an excellent platform for learning and improving problem solving abilities, which are extensively used in all areas of life. It also trains you to think clearly, logically and at times on your feet, again a very handy trait. Perhaps more important is that it is very fun. Of course at times you will face hindrances and get bugged by it (no pun intended), but as a whole you should enjoy it. Computer Science Olympiads also offer attractive prizes but hopefully that's not why you are here! Lastly, to those who intend to pursue Computer Science beyond high school, it will equip you with the fundamental styles of

thinking and a basic grounding. In other words, it will give you confidence for tackling challenging issues, and a good feel for the subject.

A note about problem solving is crucial at this point. A key issue in higher echelons of the Olympiads (all of them) is solving problems. Now, there is a crucial difference between problems and exercises that it is exigent to understand. Exercises are what you have probably been doing most of your life, and fill standard school lessons. For these questions, you have a reasonable idea of how to go about approaching them. If they are difficult exercises (such as some projects), you might take hours or days to do them or you might get 'tricked', but at no time would you feel completely lost at what to do next. For example, coding a program to find the N^{th} Fibonacci Number is an exercise, even if it might take you a while to straighten bugs and minor errors.



A problem is a completely different beast, because you don't know what you're dealing with. When you see it, you have *no* clue about how to solve it. You experiment with it, play around with it a little and have some fun. Slowly, the enigmatic nature of it will begin to lessen, and some insights will be yielded to you. Some of the easiest problems will take a few minutes, such as the round 1 Mathematics Olympiad problems, while others could take decades, such as Fermat's Last Theorem. But they are unified in that they behold insights and present you with magnanimous scoops of learning. One example of a problem (for most) is to derive a general formula for the Fibonacci Sequence (Do try it and read up on it!). Do not however get demoralized if you can't solve a problem, enhance your abilities and gain some confidence and intuition by trying some simple ones first.

Problem Solving is extremely valuable, because just about any field uses it. Mathematics, Science, Literature, Philosophy, Crime Investigations, you name it and problem solving plays an integral role in it. It is the first step of becoming an intellectual. Computer Science is ideal for this, as it does not require a large knowledge base to start solving problems. It is true that Biology Olympiads have many gems, but first you have got to memorize the textbook! Lastly, as long as you don't attempt problems that are beyond your reach, you will have fun (while increasing your reach).

A Brief Walkthrough

In Computer Science, there are essentially 5 steps in solving problems. Of course these may not always be in order and there will be many bypasses and detours, as we will see soon.

1. Understand the Problem
2. Brainstorm possible solutions/ideas
3. Prove/Disprove/Explain correctness
4. Analyze the speed and memory consumption
5. Code it effectively and Without Bugs

The first step to do is to understand the problem well. Problem descriptions can often be quite long with many restrictions, so you have to be able to clear your mind and think logically, keeping track of all the rules. Some problems will become trivial once you get a grasp of them, but certainly all problems are intractable if you don't understand them. So first, play around with the problem manually, try and propose some ideas and observations, and get a feel for the numbers. It will be depressing if you get the problem wrong because of misreading, so spend some time on this!

The next step is to brainstorm possible solutions that can solve the problem. Your primary aim is correctness so you will be searching for *correct* algorithms. Try not to start by looking for speed, as it can be quite difficult for some problems, but stay loose and propose ideas. When you devise your algorithm, it is quite likely to be correct since that is what you were logically searching for in the first place. But there might be some 'tricks' so you should prove, or at least explain, your algorithm. Try to attack it with obscure possibilities.

Then, you should analyze the speed and memory of the algorithm. If it is not good enough, don't immediately throw it away but think about why it did not make the cut. Are there ways in which you can improve it? Either ways, having these ideas and intermediate learning points fermenting at the back of your head makes you more receptive towards identifying good solutions. Finally, when you have a solution that you are convinced is good, you have to work out the precise details and code it. A good method is the top-down approach, where you start from a broad view and slowly make the details more and more extensive.

An Example Problem

Now we are going to work on the following problem: Find the median of *N* integers, where N ranges from 1 to 9,999,999 (odd) and each integer ranges between 1 and 65,536 (16-bit). Before you read on, please try to actively brainstorm ideas and solutions to make the best out of this. At least find one correct solution (which is very easy and should be a *simple exercise*)! While reading, pause and try and continue the problem yourself each time.

This problem shouldn't be too difficult to understand, and is short, unlike most others. So you should easily be able to figure out what you're supposed to do without experimentation. Yet, experimenting manually might still give you some ideas about how to go about solving this. The most important thing, however, is to comprehend how large 10 million is. How many operations can your computer do in 1 second? Write a code to find out, because some of you will certainly be surprised by the speed and some others disappointed at the sluggishness.

First we aim for a correct algorithm, even if it happens to be slow. A naïve method is to extract the minimum value (N-1)/2 times. The current smallest number is then undoubtedly the median. As a bonus, the memory consumption is not much, as we only need the space required to store the numbers ≈ 20mb. These 2 do often come 'free' in contest problems and the main issue becomes speed. For this algorithm, at worst we have to do N + (N-1) + (N-2) + … + (N-1)/2 operations, thus N * N * (1/4) < operations (worst case) < N * N. We call this $\Theta(N^2)$ since it is bounded above and below by some non-zero constant × N^2. Θ analyzes the order of growth with respect to the input thus it ignores multiplicative constants. Unluckily, $N^2 \approx 100$ trillion which will take a long time.

Methods of speeding up the algorithm are elusive in this case, so instead we try and solve a simpler problem. Lists in real life often come in a certain order, perhaps ascending or descending order. So what if the data came in ascending order? Oh then it is trivial, as we just have to pick the middle element, which is just 1 *high level* operation (called $\Theta(1)$). But this can be clearly proven to be wrong in the general case, so do we give up here? No! If the list isn't already in ascending order, let us try and force it into ascending order. This technique is called sorting (read it up for more information). Sorting is a very renowned branch of Computer Science that is a recurring motif in problems. It can be done in $\Theta(n\log n)$ generally. So the number of operations is on the order of 200 million, which is much better. But competitions usually only allow 1s time limits and we are talking about high level operations here, not just adding or subtracting. Lets try to do better, if possible.

Now there are 2 possible ways to come up with a better solution. If you have read extensively on sorting, as mentioned above, you might have already figured out how to make sorting faster for *this particular case*. Unfortunately, for the rest of us, we are back to the drawing boards and a bit desperate. So we read the problem again and notice the other constraint that we dismissed initially. What does the 65,536 tell us? Well that it is 16-bit, but that doesn't help us. Its relative size compared to 10 million, however, should astonish us. This means that on an average there are about 150 repeats of each integer. Then if we sort 1 of them, is there a way to get all the repeats sorted free? It is certainly intuitive that there should be, as long as we can track the repeats. Try to think of how and resist the temptation of looking at the next page for a while!

Its certainly easy to track repeats if you put your mind into it for a while. We just need to count the number of occurrences of each integer and store that in an array. We might get something like this:

1	2	3	4	5	6	7
162	12	153	9001	1045	2194	864

Now it might be tempting to just remove 161 counts of 1, 11 counts of 2, 152 counts of 3, … from the array and then use a $\Theta(n\log n)$ sort as in our previous idea. Yet if we retain our wits, we will see that we do not need to waste our time. The list is already sorted! All we need to do is linearly scan through the list and find when the count exceeds $N/2$. That will be done in under 65,536 operations << 10,000,000. Can you think of a faster way to do this than a linear scan? In fact we can use a method called Binary Search to do this in $\Theta(\log(65536))$, but it is redundant *in this case* because 65,536 pales in comparison with 10,000,000 so we will not get any noticeable improvement as a whole. Overall our algorithm for this is $\Theta(N) \approx 10$ million (since we can ignore additive constants in orders of growth). This can be done well under 1 second, but you should retain a cynical mindset and try it!

A problem is not completed when we solve. Firstly we can generalize it, perhaps to finding the first inter-quartile range, or other similar applications. Next we can explore the applicability of the underlying methods that we used. Hopefully, some of you might have thought about the process of sorting yourself and tried to figure out a fast ($\Theta(n\log n)$) way to do it; that would be a sub-problem which would be challenging in itself. It would then be very useful to squeeze out more power of the methods that you created and think about other situations where sorting can be used in (it is not too late to think about it even now…).

We should also go back to the problem and learn from it. We started off with a correct solution, albeit a very naïve one. This gave us confidence because at the very least, we could *solve* the problem. Then we solved an easier variant of the problem and developed that into a reasonably fast solution. By trying to speed the new solution up (recall the crux move of removing duplicates) we came up with a new sorting method (why though are state of the art methods $\Theta(n\log n)$, what's the limitation of $\Theta(n)$ sorting). At the tactical level, we learnt the idea of simplifying the problem in terms of either constraints or situation, and improving from these. At the tool level, we learnt about sorting, analysis, and binary search. If you found coming up with the final method unintuitive and lengthy, do not worry, with experience you will be able to reach from start to finish under 5 minutes. If you found it too easy, that's great. There are tons of much more exciting and challenging problems eagerly awaiting you!

Map: http://www.ahotelhub.com/travel_info/english/singapore-info/image/singapore-map.jpg
Date Bug: http://xkcd.com/376/
Monster: http://en.wikipedia.org/wiki/File:Frankenstein%27s_monster_(Boris_Karloff).jpg
Brain Storm: http://www.clipartheaven.com/clipart/business_&_office/cartoons_(a_-_c)/brainstorm.gif
Car: http://en.wikipedia.org/wiki/File:LaudaNiki19760731Ferrari312T2.jpg
10-mil > 9000: http://images.wikia.com/dragonball/images/9/97/9000Techno.gif