TJ IOI 2017 Study Guide

TJ IOI Officers

Thomas Jefferson High School for Science and Technology

Saturday, May 13, 2017

Contents

Preface

The Thomas Jeffersion Intermediate Olympiad in Informatics (TJ IOI) is a high-school computer science competition, to be held on Saturday, May 13, 2017, at TJHSST. We hope to be able to share our passion for computer science with our peers, in hopes of inspiring our fellow students through exposure to concepts not usually taught in the classroom. In order to help our participants gain a bit of background before the event, we have created a study guide to help ensure that students are equipped with the tools they need to be successful. If any mistakes are found in this study guide, or if you simply have questions, do not hesitate to contact us at tjioiofficers@gmail.com.

We'd like to thank Mr. Thomas Rudwick, our faculty sponsor, for his tireless efforts in helping us plan the event, along with the countless times we've used his room for lunch meetings. We'd also like to thank Mrs. Nicole Kim for her guidance and wisdom from having sponsored TJ IOI in the past, as well as Mr. Stephen Rose for making sure our planning was airtight. Finally, we'd like to thank Dr. Evan Glazer, our principal, and the entire TJ Computer Science Department, for creating a wonderful environment where we can foster our passion for Computer Science.

We hope that you find this study guide useful, and we look forward to seeing you at TJ IOI 2017!

The TJ IOI Officers
Kevin Geng, Official Title Here
Everyone Else, Something Something

Chapter 1

Fundamentals

1.1 Algorithms

In the field of computer science, programmers use **algorithms** to solve complicated problems. What is an algorithm? An algorithm is a procedure, or series of problem-solving operations, in order to accomplish a certain task. Algorithms can range from simple operations, such as finding the smallest value in an array, to incredibly complex jobs, such as Facebook's facial recognition or Google's PageRank.

Algorithm 1 Finding the Maximum Value in an Array

$M \leftarrow 0$
for all a_i in A **do**   ▷ iterate through elements of A
  if $a_i > M$ **then** $M \leftarrow a_i$

For example, here is an algorithm to find the largest value in an array. We assign a value of 0 to M, and then **iterate** over the contents of the array. Iteration is a computer science term, meaning we examine each element of the array one at a time. While we iterate over the array, we check each value against our existing maximum value. If the value is greater than the largest value we've already found, then we will update our maximum to reflect that value. Once we have iterated over all elements in the array, we are guaranteed to have found the largest value in the array.

Note that this algorithm was not written in a formal programming language above, nor was it described in a programming language. This is because an algorithm is an idea, not lines of code. It is important to remember that before one writes code to accomplish a task, one first thinks through the problem, and comes up with a solution in words rather than code. The language that one uses is merely the medium of communication to the computer: just as one can express the same ideas in English and French, an algorithm should be able to be implemented in Java, C/C++, Python, or any language.

1.2 Computational Complexity

Remember when we talked about how algorithms can range from simple to complex? How can we tell the difference between a simple algorithm and a complicated one? The answer lies

in **Computational Complexity**. Computational Complexity is a measure of how efficiently a program will run. Although computers appear to be instant, all operations take time. Computers are limited to a finite, yet extremely large, number of operations per second. In most modern computers, approximately 10^8 operations can be done per second. This seems like a huge number, but in practice, we find that we quickly use it up, especially when dealing with large amounts of data. This is why it is incredibly important to write **efficient** algorithms - a poorly written algorithm that solves a task in three days of computation time could be completed in under a second if written efficiently! Thus far, we have only developed a qualitative idea of efficiency, but computer scientists want rigor, and thus use **Big-O notation** in order to quantify the efficiency of their algorithms.

1.2.1   Big-O Notation

Big-O notation is the computer scientists way of describing the efficiency of an algorithm. Think back to our first algorithm (finding the maximum in a list). We examine each element of the array, one at a time. Thus, if our array is of length n, we will need n operations to complete our algorithm. This is true regardless of the value of n, so the operation is said to be $O(n)$. But what about the initial assignment? Shouldn't the algorithm be $O(n + 1)$? In computer science, we care about the efficiency when the data sets are **large**, and as n becomes large, the number of operations will grow at the same rate that n grows. We also only care about the **order** of the growth, that is, $O(cn)$ is the same thing as $O(n)$, where c is any constant factor.

An algorithm grows only as fast as its slowest operation. For example, a nested for-loop will be $O(n^2)$, and so another for-loop afterwards with $O(n)$ efficiency will be negligible compared to the nested loop. Note that an algorithm with $O(n^2)$ efficiency is not guaranteed to run slower than one of $O(n)$, as the faster operations may have constant terms associated with them that are very large. However, we know for certain that for sufficiently large data set size, the $O(n^2)$ algorithm will be slower.

Chapter 2

Data Structures

Structure your data before it structures you.

Chapter 3

Graph Theory

National GraphTheory Research and Development Center.

Chapter 4

Search Techniques

Did you try looking for it?