Analyzing Algorithmic Patterns Based on Real Coding Interview Questions

Ian Dempsey,
Computer Science Department,
Maynooth University,
Email: ian.dempsey.2013@mumail.ie

I. Abstract

This is my abstract. It is empty for now. Fill in later

II. Introduction

Nowadays interviews for jobs in the I.T industry are becoming more complex and demanding on the applicant. As this section of the working industry is usually quite technical, it is quite common for the applicants to have to perform some form of technical test. These tests normally involve a question or two that are based on a wide variety of topics in programming. These questions in particular are used to test potential employees on whether they have strong analytical and critical assessment skills. The technical tests also allow the employer to see how many standard algorithms the candidate knows and is able to use. However, it is difficult to master all of the different types of algorithms and their intricacies.

The purpose of this paper is to study the relationship between the classical and well known algorithms, and the interview questions which are commonly asked. This paper aims to give guidance to people who want to study the interview questions, but learn a core set of common ideas which will help them solve multiple problems. This would therefore allow them to save time as they prepare for the interview.

In this paper we present a comprehensive report summarising the similarities between interview questions and standard algorithms taught in any programming course. This distinguishes our work from interview preparation books which focus on the problem solving skills, simply showing the reader a solution to a single problem, and not reinforcing material they would already be familiar with. This paper's goal is to show people that understanding the fundamentals of several common algorithms, will allow them to be able to solve a wide array of problems. It is also the purpose of this paper that a beginner at programming or a fresh graduate can use this paper as guidance to show them how much time is needed for preparatory work before they have the interview.
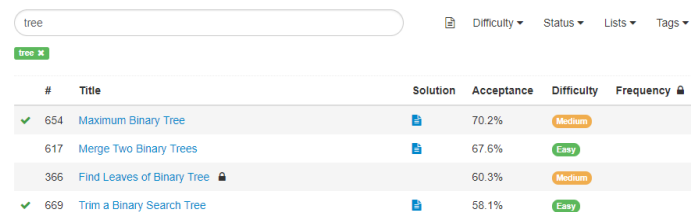
III. Technical Background

Nowadays there are a lot of online repositories for interview questions and interview related material in general. Websites such as hackerrank.com, glassdoor.ie, geeksforgeeks.org, and leetcode.com all offer information on what questions are common during interviews for different companies. They also offer ways to test potential solutions to some of the given questions.

For this paper we chose to use leetcode.com[1], as this website comes with a huge amount of online material that is generally taken from previously asked interview questions. Companies also use sites just like leetcode when they are building up a question bank to ask candidates. This was important in our decision for the repository, as it meant we were using a platform which was regularly updated. LeetCode itself also has some built-in features which really appealed to us. The website has an online discussion forum for each question, allowing the community to discuss solutions, issues and the questions themselves. LeetCode also host their own weekly coding competitions, which allows users to get even more experience and confidence in coding problems. One of the main features which this online judge website has in comparison to the others is a section for a user to perform a mock interview. This mock interview is under the time constraint of a normal real world interview, this was a big attraction for us in choosing our online judge. The online editor that is used by leetcode also allows a user to select from a multitude of programming languages, as shown in figure 3 below. For the purpose of this paper the language that was chosen was java. The reason for this is because it is a language most people learn first. However, any choice of programming language would be just as acceptable, as we present pseudocode throughout the paper which can be used by any major programming language.

The user interface of LeetCode is also very simple to use. The main page which lists all of the problems offers users the option to filter questions by difficulty, which company has asked it before, what area the questions focuses on and much more. As seen in figure 1, we have chosen to focus on questions which are tagged as being related to trees.

#	Title	Solution	Acceptance	Difficulty	Frequency
✓ 654	Maximum Binary Tree	📄	70.2%	Medium	
617	Merge Two Binary Trees	📄	67.6%	Easy	
366	Find Leaves of Binary Tree 🔒		60.3%	Medium	
✓ 669	Trim a Binary Search Tree	📄	58.1%	Easy	

Fig. 1. Homepage

A user simply chooses the question which they wish to

attempt, and is promptly brought to the problem's specific screen, as seen in figure 2. On this screen there is a general description of the problem and a few examples for more clarification. This page also contains the online editor that the user will use to attempt the problem. The online editor is quick and easy to understand, and as stated has a variety of options for the user to choose from.
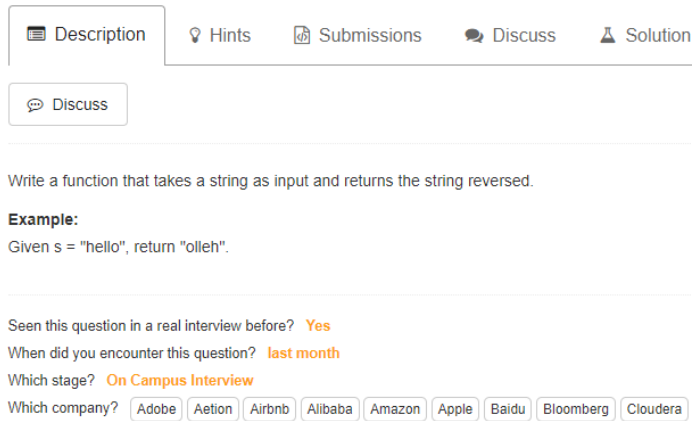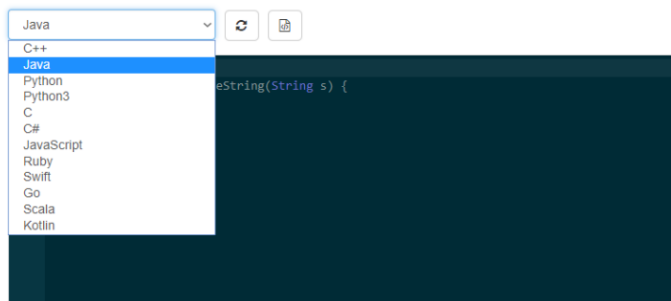

Fig. 2. Problem Specific Page

Fig. 3. Online editor with options displayed

The user will attempt to solve the problem they have selected and they will then want to submit their answer and see if it was correct, they perform this by clicking the submit solution button underneath the editor. Leetcode allows a user to sumbit their code and be informed almost instantly if they are correct or not. There are two possible outcomes once the user clicks the submit button. Either it is accepted and leetcode returns an accepted result with suggestions for the next question a user could attempt, shown in figure 4, or their answer failed a certain testcase or even their code failed to compile and run, shown in figure 5. In the case of failure, leetcode will report where an error is in the code by referencing the line.
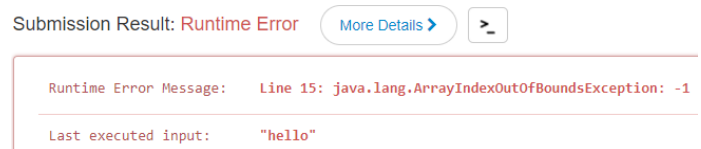

Fig. 4. User solution passes all testcases

Fig. 5. User solution failing submission

IV. PROBLEM STATEMENT

In order to fulfill the goal of this paper, we will answer the following research questions:

1) What algorithms to choose, and what categories do these algorithms fall under.
2) How do these algorithms relate to the classic algorithms and how similar are they.
3) How easy is it for a person to indentify the pattern of the algorithm.

V. THE SOLUTION

To answer question one above, we had to perform some information gathering. Many companies choose similarly styled questions for their technical tests, so we gathered information from forums, like glassdoor.ie [2] and websites which discussed experiences people had at interviews. We also read through the main categories listed in books which prepare people for interviews, such as Cracking the Coding Interview [3] These discussions listed the questions they were asked, and by what companies. The website geeksforgeeks.org has a list of the top interview questions that a user should study prior to an interview [4]. Therefore this paper focuses on the following areas:

- Sorting Algorithms
- Searching Algorithms
- Graphs
- Trees
- Dynamic Programming

In each of these areas we broke them down into some key patterns. These are patterns or types of the above listed algorithms which commonly appeared. These include but are not limited to:

- Palindrome
- Merge Sort
- Binary Search
- Tree Traversal
- Breadth-First Search
- Depth-First Search

Leetcode has over six hundred questions, which is impossible for us to answer in such a short amount of time. So we chose a representative amount of this total. We chose them based on reputation, if they were asked in interview and the solve rate of the question. Relating back to figure 1, it can be seen that we are able to filter out the questions we chose.

A. Palindrome

Palindromes are extremely useful for searching algorithms, as Palindromes are meant to be the same in both directions,

one can easily discover if the input is an actual palindrome. This is helpful for searching because one can search and find the odd character out, or the unique piece of data in some text. This style of algorithm is also space efficient, they normally have a space analysis of O(n/2) as the algorithm works over two elements of the input at a time. The basic approach of a Palindrome algorithm is to work inwards with both pointers starting at either end of the input and constantly moving towards one another and comparing if the elements are the same.

The following description is a general algorithm for solving the palindrome problem which is a common problem in Java and other languages. This approach can be used to solve numerous other problems by altering the inside of the loop. Pseudocode:

Input : Given input of characters, S
Output: Boolean
1   $leftIndex \longleftarrow$ S[0];
2   $rightIndex \longleftarrow$ S.length-1;
3   **while** *leftIndex < rightIndex* **do**
4   compare leftIndex with rightIndex;
5   **if** *leftIndex !=rightIndex* **then**
6   return false;
7   leftIndex++;
8   rightIndex++;
9   return true;

Algorithm 1: The Palindrome Algorithm

Whilst studying and working on this project, I have answered a number of questions from Leetcode.com which I was able to solve using an altered version of the above Palindrome pattern.**ID 1** TwoSum is one example. This question is described as: *Given an array of integers, return indices of the two numbers such that they add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice.* The following pseudocode is my answer to this question. In it I have taken the basic idea of the Palindrome algorithm of having two pointers used for looking through the input data, but I have altered the way these two pointers behave.In this pseudocode I have highlighted any differences in blue.

As can be seen from the above answer, the main differences are the conditionals inside the while loop. Instead of the normal approach of a Palindrome where one checks if the two elements are equal, and if so moves the two pointers are moved at the same time towards eachother, I have changed the condisitionals such that instead of moving both of these pointers together at the same time, I only ever move one pointer at a time. Depending on the result of adding the two current elements at each pointer together. If the total from adding the two integers together was greater than the target sum, then I knew that I had to move the right-pointer left once, as this would allow me to have a smaller sum and potentially the correct sum. If on the other hand the sum was smaller than the total, I would only move the left-pointer right once and add the two elements and get a new result. This process

---

Input : Array of integers nums, target S
Output: Indices i,j
1   $leftIndex \longleftarrow$ S[0];
2   $rightIndex \longleftarrow$ S.length-1;
3   **while** *leftIndex < rightIndex* **do**
4   Sort the input array nums
5   **if** *nums[leftIndex]+nums[rightIndex] > S* **then**
6   rightIndex−−
7   **else if** *nums[leftIndex]+nums[rightIndex]<S* **then**
8   leftIndex++
9   **else**
10   return leftIndex, rightIndex
11   return 0;

Algorithm 2: LeetCode Q1 TwoSum

---

was repeated until the target sum was found, or until the two pointers crossed which meant the target was not found.

B. Merge Sort

Merge Sort is a very powerful algorithm. It is more efficient than most styles of insertion, with a time analysis of $O(n * log_n)$, whereas insertion is $O(n^2)$. The idea of merge sort is to divide an array or some input in half and then sort each half before joining it back together. They do not have to be the same size which is useful.

Merge Sort uses the idea of divide and conquer, this means the list to be sorted should be divided up into equal parts first, then these new smaller parts should be sorted individually first before recreating the full list. Pseudocode:

Input : List of unsorted data
Output: Sorted List
1   **if** *length of A is 1* **then**
2   return 1
3   **else**
4   Split A into two halves , L and R. Repeat until size of part =1
5   Sort each part individually
6   Merge with another subdivided section into B, the sorted list
7   Return B, the sorted structure

Algorithm 3: The Merge Sort Algorithm through Recursion

C. Graphs

Graphs are common in our lives. News media use them to help us visualize certain statistics. Though these are not the graphs that are studied by Computer Scientists. Graphs studied by Computer Scientists are usually based on the tree structure, and the relationships among data elements. A tree is just one of the special types of graphs that can be studied, where the parent-child relationship is used to organise data. In this section I have focused on the Tree Abstract Data Type,

Breadth-First Traversal, Depth-First Traversal and Graphs in general.

1) Trees: Trees are one of the most powerful styles of data structures for processing data, this is because they allow rapid searching and fast insetion/deletion of a node. Trees are made up of nodes, these are Objects which hold some data and have a key. This key allows one to determine where this node should be in the tree. The important distinction here with these nodes in comparison to other nodes used in various data structures, is that these nodes contain references to children instead of just the next Link. Each node has exactly one parent, but can have many children.A Binary Tree is a special type of tree, this is a tree which has between 0 and 2 children. The first node in a tree is the Root, and it is possible to traverse to any node in the tree from this node. With trees the main function is traversal. There are three basic styles of traversal: inorder, preorder and postorder. Inorder visits every Node in ascending order based on their key values. Preorder is where the root is visited first, followed by it's left subtree and then it's right subtree. Finally, postorder is where the left subtree is followed by the right subtree and then the root.
The following is an example of how one might search for a particular Node in a Tree.
Pseudocode:

Input : Given a key to search for
Output: The desired Node, or null
1 $Node current \longleftarrow$ root;
2 **while** *current.data is not key* **do**
3   **if** *current is null* **then**
4   return null;
5   **if** *current.data > key* **then**
6   move left on the tree;
7   **else**
8   move right on the tree;
9 return current;

Algorithm 4: Finding a specific Node in a tree based on the key

---

$postOrder(Node localRoot)$
if $localRoot != null$ **then**
3 postOrder (localRoot leftChild)
4 postOrder (localRoot rightChild)
5 Print(localRoot data);

Algorithm 5: Basic Tree Traversal using PostOrder Traversal

ID 104, *Maximum Depth of Binary Tree*, is an example of a problem where I used the basic tree traversal styles to solve the question.The problem description is given as: *Given a binary tree, find its maximum depth.The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.* In this algorithm I used postorder traversal. Pseudocode can be found in Algorithm 6

of this paper.

$maxDepth(TreeNode localRoot)$
if $localRoot == null$ **then**
3 return 0
int ldepth =maxDepth (localRoot leftChild);
int rdepth =maxDepth (localRoot rightChild);
if $ldepth > rdepth$ **then**
7 return ldepth+1
else
9 return rdepth+1

Algorithm 6: Leetcode Q104 Max. Depth of Binary Tree

As highlighted above, the key diference is inside the if statement. I have changed the conditional itself as it is the base case to stop the recursive execution. I have created a variable *ldepth* which is the total depth of the left subtree. This is a recursive call on the leftChild of the current node. Line 4 will execute until it can't go left anymore, if it reaches null it will try to go right once, and then return going left. This is repeted until both calls return null. It then will go back all the way up until we reach the first call to go left, and set this summation of steps to ldepth. It then performs this same process but on the right subtree of the current node(which is the original node given), and set it to the variable *rdepth*. Finally there is a conditional checking if ldepth is greater than rdepth. If so it returns ldepth+1,or it returns rdepth+1. This+1 is required as I need to take into account the root node's level, if I didn't take this into account my answer would always be one less than the actual depth of the tree. The major similarities between the postorder algorithm and this solution is the way I went down the left subtree first, then the right subtree and finally to take into account the root I added one to the height. This particular way of traversing the tree, left-right-root, is exactly how postorder traversal is performed.

2) Breadth-First Traversal: This is a special way to visit the nodes in a tree, the ordering in this traversal pattern is to visit the root node, then move onto the children of the root node, printing each child in turn. It then will repeat this for each child of these nodes.Figure 1 shows the nodes in numerical order of visitation when using Breadth-First Traversal.

3) Depth-First Traversal: This is a second way of visiting nodes in a tree. This pattern involves starting at the root node, then going to the left most child, then repeating this movement until the traversal reaches a leaf node (a node which has no children), then it will move back up one node and try to visit the next child node of this current node. It repeats this until the traversal finds no univisited node. Figure 2 shows the nodes in numerical order of visitation when using Depth-First Traversal.
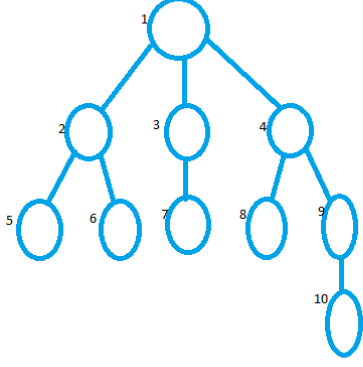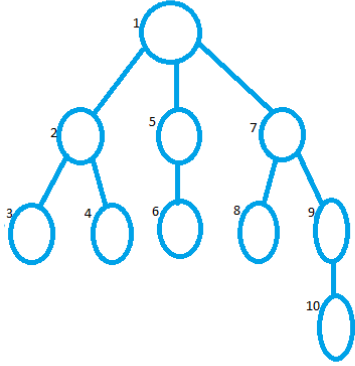
Fig. 6. Breadth-First Traversal

Fig. 7. Depth-First Traversal

Question Number	PassRate	Difficulty	Completion Time(mins)	Evaluation	
1	36.0%	Easy	9	Easy	
2	28.1%	Medium	20	Medium	
20	33.7%	Easy	12	Easy	
21	39.3%	Easy	25	Easy	
23	27.7%	Hard	27	Medium	
35	39.8%	Easy	10	Easy	
53	39.9%	Easy	20	Medium	
58	32.0%	Easy	15	Easy	
88	32.1%	Easy	12	Medium	
102	40.9%	Medium	30	Medium	
104	53.5%	Easy	28	Medium	
111	33.3%	Easy	17	Easy	
114	35.7%	Medium	33	Medium	F
136	54.9%	Easy	10	Easy	
147	33.4%	Medium	26	Medium	
148	29.2%	Medium	35	Hard	
167	47.1%	Easy	16	Easy	
205	34.2%	Easy	18	Easy	
290	33.2%	Easy	28	Medium	
326	40.4%	Easy	25	Easy	
336	26.5%	Hard	18	Medium	
344	59.5%	Easy	8	Easy	
387	47.1%	Easy	19	Medium	
389	50.9%	Easy	10	Easy	
404	47.3%	Easy	16	Medium	
654	70.1%	Medium	46	Hard	
669	58.1%	Easy	18	Easy	
690	52.9%	Easy	45	Medium	

TABLE I
ALL SOLUTIONS SOLVED BY US

VI. EVALUATION

In the previous section of this paper, we only gave specific examples of the questions that we answered. Here we give a detailed summary of all solved questions in table I. All of these were accepted by leetcode's online judging system.

VII. THE QUESTIONS CHOSEN

I based all of my questions off of the six-hundred or so that were available on www.leetcode.com[1]. To shrink the question pool that I was choosing from I would try to pick the quesitons which were under the filter of Top Interview Questions. I did pick questions which were not listed under this filter, as I felt that solving some questions from outside of this restriction allowed me to get a more diverse seletion of questions. Also the questions were tagged by the user base of the site. Meaning that some of the questions which were not listed as being interview related, could have been asked in one, and vice versa. This does lead to the issue of having faith in the honesty of the community on the site.

VIII. PROBLEMS WITH THE QUESTIONS

Whilst working on the numerous questions that I have solved on leetcode.com, I did come across issues which related to the questions themselves. These problems stemmed from the questions being worded strangely, or the example given for the solution was not explicitly clear in the way the question worked. These were just some of the issues which occurred whilst attempting to solve the problems. An example of such an issue is leetcode question 654. Maximum Binary Tree. This question's problem was easy to understand:

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

1) *The root is the maximum number in the array.*
2) *The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.*
3) *The right subtree is the maximum tree constructed from right part subarray divided by the maximum number.*

Construct the maximum tree by the given array and output the root node of this tree.

The issue which I personally came across was when I tried to understand the way the left and right subtree should be structured. The description for them was not clear to me, as I was unsure what was meant by ″left part subarray divided by the maximum number″. I then took a look at the provided example.

For me personally this did not make it any clearer as to which way I should structure the subtrees. I was able to understand that the biggest number of each subarray should be the next number and work down in a descending order from the root. I was unable to understand why the 2 was to the right of 3, and yet 0 was to the left of 5. This confusion did cause me to take longer to solve this question than anticipated.

Example 1:

```
Input: [3,2,1,6,0,5]
Output: return the tree root node representing the following tree:

    6
  /   \
 3     5
  \   /
   2 0
    \
     1
```
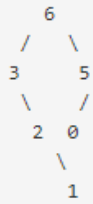
Fig. 8. Q654 Given Example

	Achievable Invs	Unachievable Invs
(a)	$\{1, 3, 4, 6, 7, 8\}$	$\{2, 8\}$
(b)	$\{1, 3, 4, 5, 7, 8\}$	$\{2, 6\}$
(c)	$\{1, 3, 4, 5, 6, 8\}$	$\{2, 7\}$
(d)	$\{1, 3, 4, 5, 6, 7\}$	$\{2, 8\}$
(e)	$\{2, 3, 4, 6, 7, 8\}$	$\{1, 5\}$
(f)	$\{2, 3, 4, 5, 7, 8\}$	$\{1, 6\}$
(g)	$\{2, 3, 4, 5, 6, 8\}$	$\{1, 7\}$
(h)	$\{2, 3, 4, 5, 6, 7\}$	$\{1, 8\}$

TABLE II

A TOTAL OF 8 POSSIBLE WAYS OF ACHIEVING A MAXIMUM NUMBER OF 6 CLASS INVARIANTS FROM THE METAMODEL IN FIGURE **??**. THE NUMBERS 1 - 8 REPRESENT 8 INVARIANTS IN FIGURE **??**.

REFERENCES

[1] LeetCode, "Leetcode."
[2] GlassDoor.
[3] G. McDowell, *Cracking the Coding Interview: 189 Programming Questions and Solutions*. CareerCup, LLC, 2015.
[4] GeeksForGeeks, "Top 10 algorithms in interview questions."