Assignment -1

Advanced Algorithms (CPT_S 515)

Name: Coral Jain
WSU ID: 011632780

1. In Lesson 1, algorithm's complexity is measured on input size instead of input values. Please indicate the input size for an algorithm that solves the following problem:

Given: a number n and two primes p, q, Question: is it the case that n = p · q?

Solution:

Considering that for any number a the input size as a, we find that when performing the operation $p.q$, the input size should be n^2 as the multiplication of two numbers p and q will take $n.n$ input size.

Also, if n is expected to be a prime number it is not possible because the product of two prime numbers is never a prime number. Hence, in that case the input size is 0.

2. In Lesson 2, we learned linear-time selection algorithm where the input array of numbers is cut into groups of size 5. Show that, when the group size is 7, the algorithm still runs in linear time.

Solution:

Case 1: Input array is cut into groups of size 5:

When the group size is 5 and median of medians (MM) is determined, we notice that in every group in LOW region (elements on left side of MM) there are 3 elements that are smaller than their medians. Every median of a group in LOW region will be smaller than MM. This clearly implies that there are at least 3 elements in LOW that are smaller than MM.

Since there are $\frac{n}{10}$ groups in LOW region, total number of elements smaller than MM is $\frac{3n}{5}$. Similarly, there are at most $\frac{7n}{10}$ elements greater than MM.

$$\Rightarrow \therefore T(n) = T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + O(n)$$

where $T\left(\frac{n}{5}\right)$ is time complexity due to group division-by-5 step $O(n)$ is time complexity obtained by linear and sorting steps.

$$\Rightarrow \because T(n) \leq c.n$$

$$from\ above, \qquad T(n) \leq c.\frac{7n}{10} + c.\frac{n}{5} + a.n$$

where c and a are constants.

$$\Rightarrow T(n) \leq \frac{9cn}{10} + a.n$$

Since, $\frac{9}{10}$ is a quantity less than 1 and a is a positive constant, this implies that for larger input values of n, time complexity of algorithm would be:

$$\Longrightarrow T(n) \leq c.n \text{ where } c \text{ is a very large constant.}$$

This means that $T(n)$ is linear.

Case 2: Input array is cut into group of size 7:

When the group size is 7 and median of medians (MM) is determined, we notice that in every group in LOW region (elements on left side of MM) there are 4 elements that are smaller than their medians. Every median of a group in LOW region will be smaller than MM. This clearly implies that there are at least 4 elements in LOW that are smaller than MM.

Since there are $\frac{n}{14}$ groups in LOW region, total number of elements smaller than MM is $\frac{4n}{14}$. Similarly, there are at most $\frac{10n}{14}$ elements greater than MM.

$$\Rightarrow\therefore T(n) = T\left(\frac{10n}{14}\right) + T\left(\frac{n}{7}\right) + O(n)$$

where $T\left(\frac{n}{7}\right)$ is time complexity due to group division-by-7 step and $O(n)$ is time complexity obtained by linear and sorting steps.

$$from\ above, \qquad T(n) \leq c.\frac{10n}{14} + c.\frac{n}{7} + a.n$$

where a is a positive constant.

$$\Rightarrow T(n) \leq \frac{12cn}{14} + a.n$$

This can be rewritten as $T(n) \leq cn - \frac{2cn}{14} + a.n$

For $T(n) \leq c.n$ we need $a.n - \frac{2cn}{14} \leq 0$

Hence, we have $a \leq c/7$

So, when the above condition is true, $T(n)$is linear.

$$\Longrightarrow T(n) \leq c.n \text{ where } c \text{ is a very large constant.}$$

This means that $T(n)$ is linear.

3. In Lesson 2, we learned closest pair algorithm that runs in O(nlogn) time. However, that algorithm can be improved further when additional assumption is made. Here is one. Suppose that there are n^2 bugs sitting on a piece of paper of size n by n. Any two bugs must stay away by at least 1. Each bug is given as a pair of coordinates. Design a linear-time algorithm that finds the closest pair of bugs. (Hint: since the input size is n^2, the linear time here really means the running time is $O(n^2)$ where the n^2 is the number of bugs.)

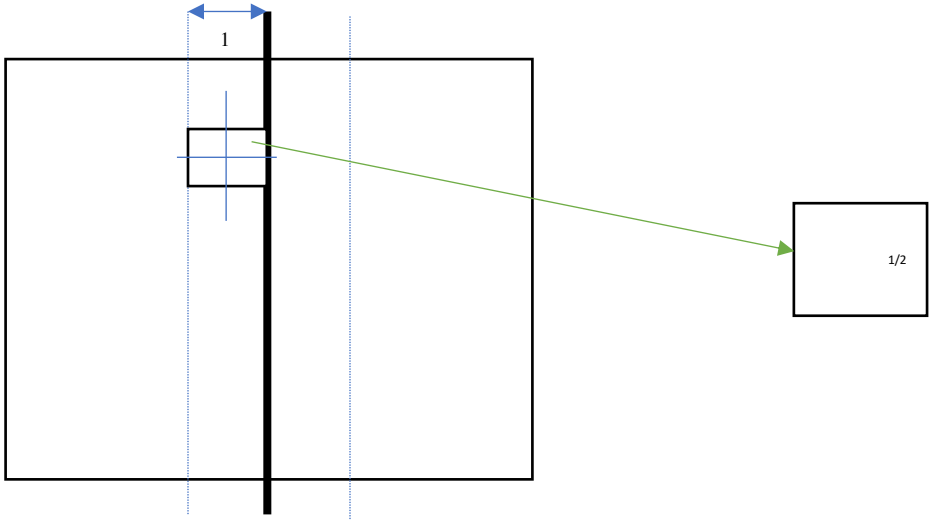
Solution:

In the closest pair algorithm, we have seen that in order to reduce the time complexity for n points, we divide the given plane into two equal parts $\frac{n^2}{2}$ and $\frac{n^2}{2}$. Because of this this step, the time complexity of $2T\left(\frac{n^2}{2}\right)$ is obtained for the first step since this is a recursive step. Next, it takes $O(n^2)$ to find the strip.

In yet another step, both the parts are separately sorted and a time complexity of $O(n^2 log n^2)$ is derived which is further reduced to $O(n^2)$ when presorting is done.

Finally, the closest points that lie on the two sides of the strip take $O(n^2)$ as time complexity as a point should only be compared to 7 other points in the strip.

However, with the condition given in the question here that two points are at least 1-unit distance away we must use the same mathematical analogy that we used to find the closest distance between a point on the right-hand side and a point on the left-hand side. Therefore, if the distance is below the given criteria, such a point is impossible, and a given point on the left-hand side or right-hand side has to be compared to only some other points on the left. Thus for $\frac{n^2}{2}$ points we draw hypothetical square of side ½ on the left-hand side of the square as shown below. Thus for this we realize that only 1 point can be inside the square. Thus, we have 3 other points with which this needs to be compared i.e. 2 pairs. For total $\frac{n^2}{2}$ points on the left there will be $\frac{2n^2}{2}$ pairs like this. Therefore, while the left-hand side or right-hand side, the complexity would be $O(n^2)$ which is linear in this case.



Hence, the total complexity in this case would be:

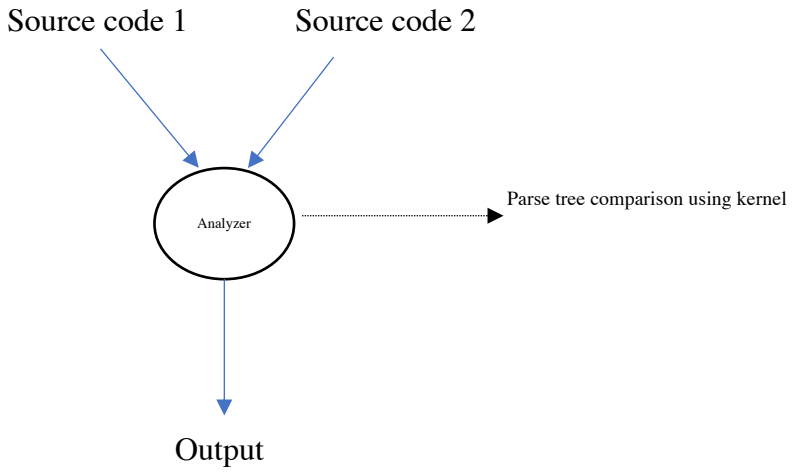
$$T(n^2) = 2O\left(\frac{n^2}{2}\right) + O(n^2) + O(n^2) + O(n^2)$$
$$T(n^2) = O(n^2)\ where\ n^2\ is\ the\ number\ of\ bugs\ in\ this\ case.$$

4. Algorithms are to solve problems, or more precisely, to solve problems that have a precise mathematical definition. However, in practice, figuring out what is exactly the problem is not easy at all. (You may search internet) Suppose that you would like to write an algorithm to decide the similarity between two C programs. What would you do?

Solution:

The similarity between two C programs can be defined as the exactness between two C programs in either in terms of their structure, syntax or parse tree. If the source codes of the programs are available, the similarity of the two C programs lies in the way the two programs behave and how their input-output relationship is defined. This is because a program can be similar to another if they look same in terms of their structure and syntax since this is what defines the word 'similar' as per the dictionary. Thus similarity can be defined in terms of their syntaxial and parse tree.

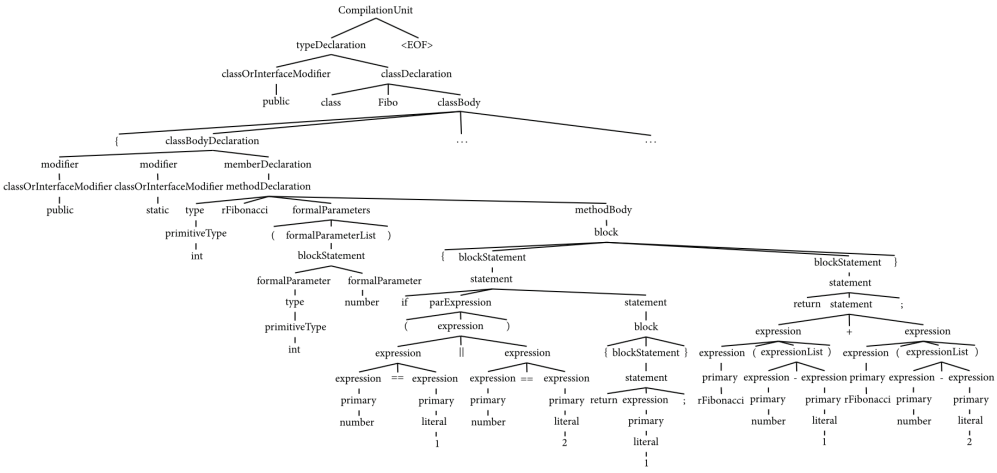
If we are writing an algorithm for figuring out the similarity between two C programs, we should consider the fact that it is only possible by comparing the syntax or comparing the functionality of the two C programs. If we want to find the similar type of C programs their parse tree must be compared using parse tree kernel and then compare the source code of the two given programs. In the cases where the source codes of C programs are available, we can design a similar algorithm which can be as shown below:



The basic algorithm is as shown above where the two codes are compared parse tree kernel. For a structured data the finding the similarity using the parse tree kernel appears to be best. The analyzer in the above algorithm generates the parse trees of the source codes given as inputs and the kernel function evaluates the product of the two elements.

An example of a parse tree is given as below. The parse tree can be obtained through analyzer to obtain the parse trees. This can be done very well and is done by many algorithm designers by taking the source codes and obtained the tokens which provide certain operations based on the

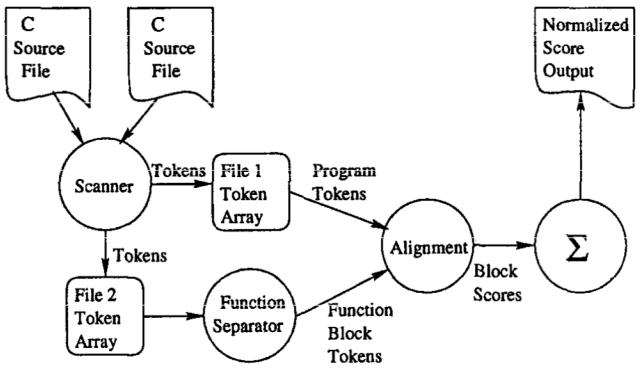
source codes. This ranges from a simple keyboard or a string to a loop statement. All this is done in order to reduce this to a parse tree. Such an example of a parse tree is given as below.



In order to compare parse trees of natural language, a kernel function is defined. Such a function maps a tree that goes over all the subtrees A kernel function for the two trees can be defined as following:

$$K_{tree}(T_1, T_2) = \sum \quad \sum \quad C(n_1, n_2)$$

For example, David Gitchell and Nicholas Tran designed a similar program and called it SIM that tells the users similarity between two C programs with the help of two source codes that are input into the system. The algorithm for such a program is as shown below. In the program as shown below the two source codes are input into the system which pass through the scanner that generates tokens (stream of integers) in the form of arrays. After the tokens are formed, the second source code is divided into sections that denote the modules of source codes added as inputs. The algorithm for such a program is shown as below.



The module from the second source code is then mapped with the tokenized output from the first source code and scored are predicted which define the similarity between the two programs.

However, when the source codes of the given two C programs are *not available*, we need to know the input and output of the program and the next state with the help of which we can predict th

References:

[1] D. Gitchell, N. Tran. "Sim: A Utility for Detecting Similarity in Computer Programs," Wichita State University. *Retrieved from* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.452.9879&rep=rep1&type=pdf.

[2] Mathematical Problems in Engineering, Volume 2015, Article ID 429807, 12 pages. *Retrieved from* http://dx.doi.org/10.1155/2015/429807