Lab 1 Algorithms

Introduction to Computer Science I

# Objectives:

After performing this lab, the students should be able to

* explain the concept of an algorithm
* describe general problem-solving strategies
* solve problems by devising algorithms in the form of pseudocode

Estimated time: 70 minutes

## Background

C:\Users\lavenderazura\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4YL3Y3GQ\MC900064971[1].wmfAn **algorithm** is a list of steps to solve a problem. An algorithm should be clear and specific enough so that anyone can obtain the same result by following the algorithm. You can write algorithms in your natural language (e.g. English). For example, here is an algorithm (recipe) to make some delicious chocolate cookies[[1]](#footnote-1):

1. Preheat oven to 350 degrees °F (175 degrees °C).
2. In a large bowl, beat 1 cup butter, 1½ cup white sugar, 2 eggs, and 2 teaspoons of vanilla extract until light and fluffy.
3. Add 2 cups all-purpose flour, 2/3 cup cocoa powder, ¾ teaspoon baking powder, and ¼ teaspoon salt into the butter mixture from step 2. Stir until well blended.
4. Add 2 cups semisweet chocolate chips and ½ cup chopped walnuts into the mixture from step 3.
5. Drop rounded teaspoonfuls of mixture from step 4 onto ungreased cookie sheets.
6. Bake for 8 to 10 minutes in the preheated oven.
7. Cool slightly on the cookie sheets before transferring to wire racks to cool completely.

Algorithms are used widely in computer science to describe how computational problems can be solved. Computer scientists design algorithms before they develop them in actual computer programs. If we want an algorithm for finding out which of two numbers, **a** and **b**, is larger, we can write an algorithm like this:

1. If **a** is greater than **b**, **a** is larger.
2. if **b** is greater than **a**, **b** is larger.
3. If neither is true, **a** and **b** are the same.

An algorithm, such as the one above, can be described in our natural language (English in this case). Using **pseudocode** allows computer scientists to focus on the logic of solving a problem without worrying about the syntax of any particular programming language.

If this algorithm is implemented in a computer programming language such as C++, the program would look like this:

**if (a > b)**

**cout << “a is larger” << endl;**

**else if (b > a)**

**cout << “b is larger” << endl;**

**else**

**cout << “a and b are the same” << endl;**

Although there is not a single algorithm that can be used to solve any problem, a number of general strategies can be applied to problem-solving:

#### Understanding the problem

Make sure that you have carefully read the problem statement and thoroughly understand the problem. Consider questions such as:

* What is the unknown?
* What are the data, conditions, or issues key to the solution?
* Can you illustrate the problem with a figure?

#### Designing a solution

Once you have thoroughly understood the problem, you can move on to the next phase of designing a solution for the problem. As you design your solution, it might be useful to consider the following questions:

* Could you break down the problem into manageable pieces?
* Are there any connection between the data and the unknown?
* Have you seen a related/similar problem?
* Could you use the solution to a similar problem?
* Could you solve a more accessible related problem?
* Could you restate the problem?
* Did you use all the data and conditions?

#### Considering alternatives to the solution and refining the solution

The first solution that comes to your mind might not be the best one; therefore, it is important to consider alternatives to the solution and refine the solution as necessary. It is easier to modify your approach to solving a problem at this point than at a later point when you have invested time, money, and effort into pursuing an incorrect or suboptimal solution.

#### Implementing the solution

Once you are satisfied with your solution, implement the solution by putting it into a usable form. In computer science, for instance, a solution is often implemented in the form of a computer program.

#### Testing the solution and fixing any problems that exist

After implementing a solution, it is important to examine the results, ensuring that the results are correct given all possible data and under all conditions.

## Activity

1. [Algorithm of a daily activity] Write an algorithm for an activity that you perform regularly (e.g. brushing your teeth, driving a car forward at a certain speed, making a cup of coffee). Be sure to include enough details so that any person can perform the exact same activity following your algorithm.
2. [Fox, chicken, corn] [[2]](#footnote-2) A man has to get a fox, a chicken, and a sack of corn across a river. He has a rowboat, and it can only carry him and one other thing. If the fox and the chicken are left together, the fox will eat the chicken. If the chicken and the corn are left together, the chicken will eat the corn. How does the man do it? Design an algorithm for the man to solve this problem.
3. [Tower of Hanoi] The Tower of Hanoi puzzle involves three towers with a number of disks (Figure 1). The goal is to move all the disks from the first tower over to the third tower, while never placing a larger disk onto a smaller disk. Devise an algorithm that solves the Tower of Hanoi puzzle of four disks. Here is an online interactive Tower of Hanoi: <http://www.mathsisfun.com/games/towerofhanoi.html>

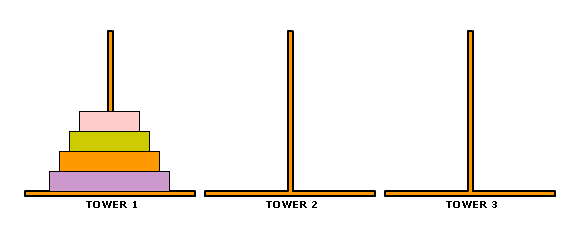


Figure Tower of Hanoi

1. Recipe for chocolate chip cookies. <http://allrecipes.com/recipe/chocolate-chocolate-chip-cookies-i/> [↑](#footnote-ref-1)
2. <http://www.mathisfun.com/> [↑](#footnote-ref-2)