

# CS 199 Lecture 16

## Algorithms

**Programming vs. Algorithms:** A program is the way we tell the computer what we want it to do; the algorithm is the idea behind the program.

Algorithms are step-by-step procedures that must terminate. Often, they have input, and produce output. A common example is a recipe for baking a cake. The input is all the ingredients, and the output is the cake. Steps might involve heating the oven, folding the walnuts into the batter, etc.

How would you describe an algorithm to make a peanut butter & jelly sandwich? You might begin by saying “Take the bread, and put peanut butter on it.” Would I be following your directions if I put a jar of peanut butter on top of a loaf of bread? Of course not, because you meant for me to spread a little peanut butter on a slice of bread. Similarly, if you say “Put the bread in the toaster”, you also mean that the toaster should be turned on. When writing programs, you have to be really careful to consider all the conditions, and the steps have to be very clear. English is ambiguous, but it works for humans because everyone knows what you mean. The computer isn’t intelligent, and so it can’t figure out what you mean. So computer commands are really precise, and you have to tell a computer - by writing a program - to do exactly what you mean.

In future lectures, we’ll learn about specific commands in a programming language called Python, and we’ll talk about the efficiency/running time of algorithms, which we use to measure how good an algorithm is.

How do we use a computer to solve a problem?

- We begin with the Problem Specification:
  - What is wanted?
  - What is the input?
  - What is the output expected?
- Ideas on How to Solve
- Converting the ideas into an Algorithm, which we can’t run on an a computer directly.
- Convert the algorithm into a program
- Run the program on a computer, possibly find errors, and so repeat this process.

## Swapping and Sorting

Do you remember the swapping/sorting sheets we used last lecture? We explained how to do conditional swaps in a way the computer could understand. (A conditional swap is a statement like : If  $M_1 > M_2$ , swap them.) What is the effect of this swap? Whatever the initial values of  $M_1$  and  $M_2$ , we end up with  $M_1$  containing the smaller value, and  $M_2$  the larger one.

This single conditional statement is a program to sort 2 numbers, because whatever the two numbers in those memory locations are, after we execute the statement, they'll be in order. How about a program to sort 3 numbers?

If  $M_1 > M_3$ , swap them

If  $M_1 > M_2$ , swap them.

(By this point, we know that  $M_1$  is the smallest of the three. Now we only need to make sure that  $M_2 < M_3$ ).

If  $M_2 > M_3$ , swap them.

What about sorting 4 numbers, or 5 numbers? Read the algorithm for Bubble Sort.

How many steps does it take to bubble sort 4 items? In the first phase, 3, in the second phase, 2, and in the last phase 1.

With 5 items, how many steps does it take?  $4 + 3 + 2 + 1 = 10$ . With 6 items,  $5 + 4 + 3 + 2 + 1 = 15$ .

What about  $n$  items? It takes  $(n - 1) + (n - 2) + \dots + 2 + 1$ . How do we find the sum? We can try a trick that is attributed to the famous mathematician Gauss, by writing the sum out in 2 different ways.

$$sum = (n - 1) + (n - 2) + (n - 3) + \dots + 3 + 2 + 1$$

$$sum = 1 + 2 + 3 + \dots + (n - 3) + (n - 2) + (n - 1)$$

We can add up these 2 equations, to get  $2 \times sum = n(n - 1)$ , so  $sum = \frac{n(n-1)}{2}$ .

The bubble sort algorithm isn't very fast; there are others that are faster, such as quicksort.

(Demonstration of Quicksort using numbers 1 through 9.)