This is the first of several lectures about recursion.

For reasons that nobody can explain, recursion is one of the most misunderstood topics in computing. Many people do not understand recursion and in fact are afraid of recursion. Over the years, students ask me why we need to study recursion. Some students believe that recursion is slow, difficult to understand, difficult to analyze, use too much memory, useful for only exams.

The truth is that recursion is slow, useless, difficult, inefficient, and really really bad if you do not understand recursion.

Unfortunately, many books do not explain recursion well. Many programming books do not talk about recursion at all. Some books talk about recursion but have only very brief explanation.

If you understand recursion and use it properly, recursion can be fast and efficient.

This class will explain recursion and give you clear understanding when recursion is good and when recursion is not good.

When and where is recursion used? Almost everywhere, actually. Homework 7 asks you to use the Q sort function in C. . The q sort function implements the quick sort algorithm and this is a recursive algorithm.

Recursion is widely used for solving strategy games, such as chess and go.

Recursion is also often used in solving optimization problems.

Recursion is a way to solve problems. Some problems can be solved by recursion well. Some other problems should not use recursion.

You can think of recursion as a tool. You need many different tools, such as hammer, scissors, screw driver, wrench, etc. . Hammer is good for hitting a nail. However, a hammer is not good for cutting paper. You should not say a hammer is not good because a hammer cannot cut paper. Some people say recursion is not good. This statement is as invalid as saying a hammer is not good. Whether a tool is good or not depends on the problems you want to solve.

This lecture and several future lectures will explain recursion so that you have thorough understanding about recursion. Then, you can decide whether recursion is the right way for solving your problems.

Recursion naturally exists around us. In fact, everyone is part of recursion.

You have parents. Your parents have parents. They are your grandparents. Your grandparents have their parents. This is recursion. This is a recurring pattern.

Recursion is more than simply repeating things. Each generation is different. You are younger than your parents. Your parents are younger than your grandparents.

Recursion has to stop somewhere. If you have no child, then recursion stops at you. If you have a child or children and you have no grandchild, then recursion stops at your child, or children.

Recursion has three essential components: recurring patterns, changes, and stop condition.

Now, you already understand that you are part of recursion. Your parents are also part of recursion. Please call your parents and tell them that.

Please call your parents and thank them for supporting you to study at Purdue.

Please tell your parents that you love them very much.

If you call your parents and tell them that you are learning recursion, you will understand recursion much better and faster.

Let me explain the concept of quick sort. This algorithm first selects a value in the array as a reference. Let’s call it R1. This is also called the pivot. Then, the algorithm puts all values smaller than R 1 on one side of the array. The values greater than R 1 are put at the other side of the array. Because of transitivity, there is no need to compare these two sides. Transitivity says that if ei is greater than bee and bee is greater than cee, ei must be greater than cee. It is unnecessary comparing ei and cee.

The quick sort algorithm divides the array into three parts: the part that is smaller than R 1, R 1, and the part that is greater than R 1. Then, the quick sort algorithm sorts the first and the third parts separately.

For example, for the part that is greater than R 1, the algorithm finds another reference called are two. This part is divided into three parts: the part that is smaller than are two, are two, and the part that is great than are two.

Quick sort is a recursive algorithm and it is quick. It is widely used for sorting. We will study the quick sort algorithm in more details later.

Let’s consider another recursive algorithm called binary search.

Consider an array and the elements are sorted in the ascending order. We want to find whether a value X is in this array. The first step compares X with the value at the center of this array. Let’s call the value at the center vee one. If the two values match, we know X is stored in the array. If the two values do not match, we will keep searching. Suppose X is greater than vee one, then we can discard half of the array that is smaller than vee one because it is not possible finding X among this half of the array. Then, the algorithm compares X with the value at the center of the remaining part. The array to be searched becomes smaller and smaller. Eventually, we will reach the condition when the array has nothing left and we know X is not stored in the array.

As you can see, this algorithm follows the three components of recursion. First, there is the recurring pattern.

Second, the array to be searched gets smaller and smaller. This is the change.

Third, search stops when X is found or the array has no element.

This method is called binary search and it is very efficient.

The third example uses recursion in a board game like chess or go. Imagine that the current state of the board has several pieces. There are several options from the current state. For example, in the chess game, a player may move one of several pieces. These options can be expressed by the new states of the board.

Do you remember the three essential components of recursion mentioned earlier? The first component is the stop condition or conditions. Also called terminating conditions. This is the condition when recursion ends and nothing needs to be done. For quick sort, the stop condition is when the array has been sorted. For binary search, the stop condition is when we know whether the searched value is in the array or not.

For the board game, the stop condition arises when one of the player wins the game.

The second essential component of recursion is changes. In quick sort, each step breaks the original array into three parts and each part has fewer elements than the original array. Thus, the change is the number of elements. For binary search, the number of elements to be considered is also getting smaller and smaller. For the board game, when a player move, add, or remove one piece, the board is changed.

The third component is the recurring pattern. For quick sort, the first part is smaller than the reference value. The third part is greater than the reference value. We need to sort the first and the third parts. These are similar to the original problem, that is, sorting elements in an array. This is the recurring pattern.

For binary search. If the value X is different from the value at the center of the array, then we need to search X in either of the two parts. This is similar to the original problem, except the new array is smaller.

For the board game, after one move, the board looks almost the same, except that particular move. Thus, this is the recurring pattern.

Earlier, we said recursion is good for solving some problems. What are these problems?

Recursion is particularly good for solving problems that have branches. What are the branches? Think about the board game. There are several options for the player. Each move chosen by the player will lead to a different game.

Recursion is particularly helpful if we do not know the number of options in advance. As the game progresses, sometimes there are more options to choose from. Sometimes, there are fewer options. Recursion is often a good way to solve this type of problems.

Quick sort is another example of branches. After breaking the original array into three parts, the algorithm has to sort the first and the third parts. This is a two-way branch.

Next, let’s consider how to use recursion to solve some other problems. The first problem determines how many ways we can select blue and red balls.

Imagine that you have unlimited number of balls of two colors: blue and red. You want to select some balls. We consider the orders. Thus, a blue ball followed by a red ball is considered different if the first ball is red and the second ball is blue.

If there is no restriction about how to select the balls, there are two options for selecting the first ball, either blue or red. There are two options selecting the second ball, either blue or red.

If we want to select N balls, there are two power of N options.

What will happen if we add a restriction: two adjacent balls must not be both red? How do we solve this problem?

The first way to solve the problem is to list the solutions. When we need to select only one ball, there are two solutions: either red or blue.

When we need to select two balls, there are three solutions: Red Blue, Blue Red, and Blue Blue.

If we want to select three balls, there are five solutions.

As the number of balls to select grows, this is obviously not a good solution because the number of options also grows. We need a more systematic way to solve the problem.

Let’s think about how to solve the problem. Instead of thinking about selecting N balls, let’s select only one ball.

If the first ball is blue, the second ball can be blue or red. There is no restriction of the second ball. If the second ball is red, the third ball must be blue.

If the first ball is red, the second ball must be blue. Thus, we have no other option for the second ball. After selecting blue for the second ball, the third ball can be blue or red.

This slide explains the rule.

We can use F of N as a function to express the number of options when selecting N balls. We know F of one is 2 and F of two is three because we have counted.

We want to select N balls. However, this problem is too complex to solve directly. Instead, we want to shrink the problem a little bit by selecting only one ball.

If the selected ball is blue, there is no restriction of the second ball. Thus, it is the same problem, with only one difference: we need to select N minus one balls. The problem becomes slightly smaller.

If the selected ball is red, the second ball must be blue. Then, it becomes the same problem again, with one difference: we need to select N minus two balls.

Thus, F of N can be divided into two cases: When the first ball is blue, the problem becomes F of N minus one. When the first ball is red, the problem becomes F of N minus two.

How should we treat these two problems? Should we add them together or multiply them?

The principle is this: if there are two cases, choose one from either A or B, then we add them. If these cases are independent and can be both chosen, we should multiply them.

Let me use an example to explain. Suppose you go to a restaurant. On the menu, there are four options for beef, three options for chicken, four options for fish, and five options for salad. How many options do you have to choosing a meal? Since you need to choose one from these options, you should add the options together, 4 plus 3 plus 4 plus 5 and you have 16 options to choose a meal.

Suppose there are three options for desserts after the meal, how many options do you have for the meal and the dessert? In this case, since you can choose a meal and a dessert, the number of option is 16 multiply 3 equals to 48.

Based on this principle, let’s go back to the question whether F of N should be F of N minus one plus F of N minus two, or F of N should be the product of the two?

The answer is the sum, not the product because the first ball is either blue or red. We can choose only one of the two possibilities.

Let’s summarize what we have learned so far.

F of one is two.

F of two is three.

These are the stop conditions.

We know F of one and F of two because we counted.

F of N is the sum of F of N – 1 and F of N – 2.

F of N – 1 is the sum of F of N – 2 and F of N – 3.

As you can see, there is this recurring pattern.

The argument gets smaller and smaller. This is the change. As the argument becomes smaller, eventually it will be small enough to reach the stop condition.

We have found all three components of recursion.