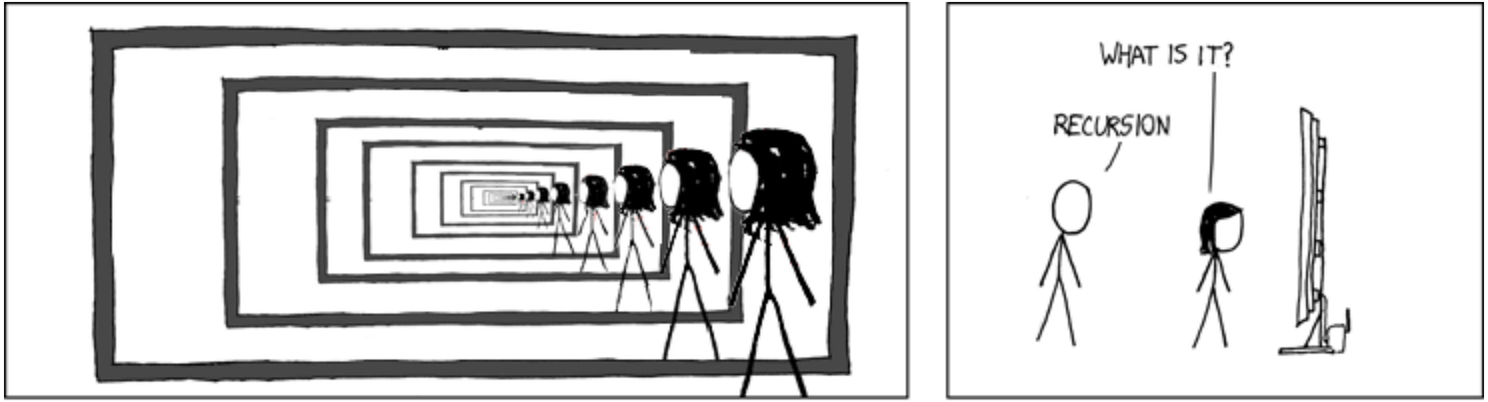
Recursion

Chapter 14



http://xkcdsw.com/1105

Recursion

- A recursive function is a function that calls itself

Recursion

- A recursive function is a function that calls itself

https://www.google.com/?gws_rd=ssl#q=recursion

Recursion

- A simple recursive function looks like this

```
void recurse()
{
	cout << “aaaaaa”;
	recurse();
}
cout << “... go to the castle A”;
recurse();
cout << “rrrrrrrg” << endl;
```

Recursion

- A simple recursive function looks like this

```
void recurse()
{
        cout << “aaaaaa”;
        recurse();
}
cout << “... go to the castle A”;
recurse();
cout << “rrrrrrrg” << endl;
```

When does it stop?

Recursion

- When writing a recursive function, don’t forget a base stopping case!

Recursion

- When writing a recursive function, don’t forget a base stopping case!

```
void recurse()
{
	cout << “aaaaaa”;
	recurse();
}
cout << “... go to the castle A”;
recurse();
cout << “rrrrrrrg” << endl;
```

How can we make this function stop?

Recursion

- There are two forms of recursion

Recursion

- There are two forms of recursion
  - Direct
    - The form we have seen so far
    - A function that calls itself, ie function B calls function B

Recursion

- There are two forms of recursion
  - Direct
    - The form we have seen so far
    - A function that calls itself, ie function B calls function B
  - Indirect
    - A function that calls another function
    - Function A calls B calls C calls A

Recursion

- What good are recursive functions?

Recursion

- What good are recursive functions?

- Some algorithms work better, or seem cleaner, written as a recursive function

Recursion

- What good are recursive functions?

- Some algorithms work better, or seem cleaner, written as a recursive function
  - Binary Search
  - ...

Recursion

- Now, we will look at some recursive examples

Recursion Examples: Factorial

- A recursive Factorial function

Recursion Examples: Factorial

- A recursive Factorial function
  - In mathematics, the **factorial** of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example, The value of 0! is 1, according to the convention for an empty product.

Recursion Examples: Factorial

- Factorial example

$3! =$

Recursion Examples: Factorial

- Factorial example

3! = 3 * 2!

2! =

Recursion Examples: Factorial

- Factorial example

3! = 3 * 2!

2! = 2 * 1!

1! =

Recursion Examples: Factorial

- Factorial example

3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

0! =

Recursion Examples: Factorial

- Factorial example

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1 * 0!$

$0! = 1$

Recursion Examples: Factorial

- Factorial example

$3! = 3 * 2 * 1 * 1$

Recursion Examples: Factorial

- Factorial example

3! = 3 * 2 * 1 * 1

- How can we write this factorial function?

Recursion Examples: Factorial

- What’s the base case?

Recursion Examples: Factorial

- What’s the base case?
    - 0! == 1

Recursion Examples: Factorial

```
int factorial(int value)
{
    if (num == 0) // The algorithm base case
    {
        return 1;
    }
    else
    {
        return num * factorial(value - 1);
    }
}
```

Recursion Examples: GCD

- Another example of a recursive function is the Greatest Common Divisor

Recursion Examples: GCD

- Another example of a recursive function is the Greatest Common Divisor
    - In mathematics, the **greatest common diviso** (**gcd**) of two or more integers, when at least o them is not zero, is the largest positive intege divides the numbers without a remainder. For example, the **GCD** of 8 and 12 is 4.

Recursion Examples: GCD

- Euclids Algorithm

gcd(x,y)	= y	If y divides x with no remainder
	= gcd (y, remainder of x/y)	Otherwise

Recursion Examples: GCD

- GCD Example

gcd(49,28)

Recursion Examples: GCD

- GCD Example

gcd(49,28) == 7

Recursion Examples: GCD

- What is the base case?

Recursion Examples: GCD

- What is the base case?
  - x modulo y == 0

Recursion Examples: GCD

- What is the base case?
  - x modulo y == 0

- How do we implement GCD recursively?

Recursion Examples: GCD

```
int gcd(int x, int y)
{
        if (x % y == 0)  // The algorithm base case
        {
                return y;
        }
        else
        {
                return gcd (y, x % y);
        }
}
```

Recursion Examples: Fibonacci

- The Fibonacci Numbers calculations is a classic example of recursion

Recursion Examples: Fibonacci

- The Fibonacci Numbers
  - A sequence of numbers where
    - $F_0 = 0$
    - $F_1 = 1$
    - $F_N = F_{N-1} + F_{N-2}$ for all N>= 2

Recursion Examples: Fibonacci

- The Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Recursion Examples: Fibonacci

- What are the base case(s)?

Recursion Examples: Fibonacci

- What are the base case(s)?
  - $F_0 = 0$
  - $F_1 = 1$

Recursion Examples: Fibonacci

- What are the base case(s)?
  - $F_0 = 0$
  - $F_1 = 1$

- How would we implement this recursively?

Recursion Examples: Fibonacci

```
int fibonacci(int n)
{
	if (n <= 0)		// base case 1
		return 0;
	else if (n == 1)	// base case 2
		return 1;
	else
		return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Recursion Examples: Searching

- Recursion can also be used in searching

Recursion Examples: Searching

- Recursion can also be used in searching
- How can we use recursion to find an element in a list?

Recursion Examples: Binary Search

- You might remember from Chapter 9 that the Binary Search is more efficient than a recursive search

Recursion Examples: Binary Search

- You might remember from Chapter 9 that the Binary Search is more efficient than a recursive search

Recursion Examples: Binary Search

- A reminder of the algorithm...

- Where x is the desired item

1. Check the middle item, y
   1.1. If x == y, then return
   1.2. if x < y, go to step 1 with the smaller half of the array
   1.3. if x > y, go to step 1 with the larger half of the array

Recursion Examples: Binary Search

- How can we implement this as a recursive algorithm?

Recursion Examples: Binary Search

- How can we implement this as a recursive algorithm?
- What are our base cases?

Recursion Examples: Binary Search

```
int binarySearch(const int* pArray, int first, int last, int value){
	int middle;
	if (first > last)
		return -1;
	middle = (first + last) / 2;
	if (pArray[middle] == value)
		return middle;
	if (pArray[middle] < value)
		return binarySearch(array, middle + 1, last, value);
	else
		return binarySearch(array, first, middle - 1, value);
}
```

Recursion Examples: Quick Sort

- There are also sorting algorithms that can use recursion

Recursion Examples: Quick Sort

- There are also sorting algorithms that can use recursion
  - Quick Sort

Quick Sort

- Basic Algorithm
  - if the number of elements in array is 0 or 1, return
  - pick any element as pivot
  - partition array into two disjoint groups
    - left group are items less than the pivot
    - right group are items greater than the pivot
  - Perform the quick sort on the left and right sides

Quick Sort

Choose pivot

94	11	96	12	35	17	95	28

Quick Sort

Create two lists

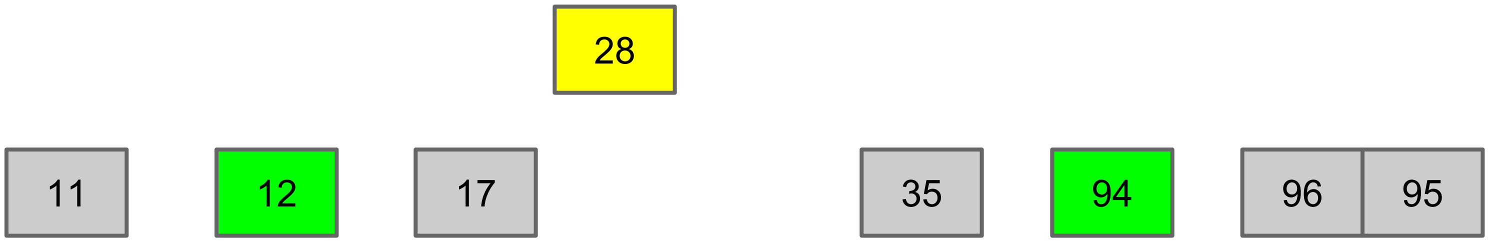
11	12	17		28		94	96	35	95

Quick Sort

Continue. quicksort(left), Quisort(right).
Choose pivot(s)

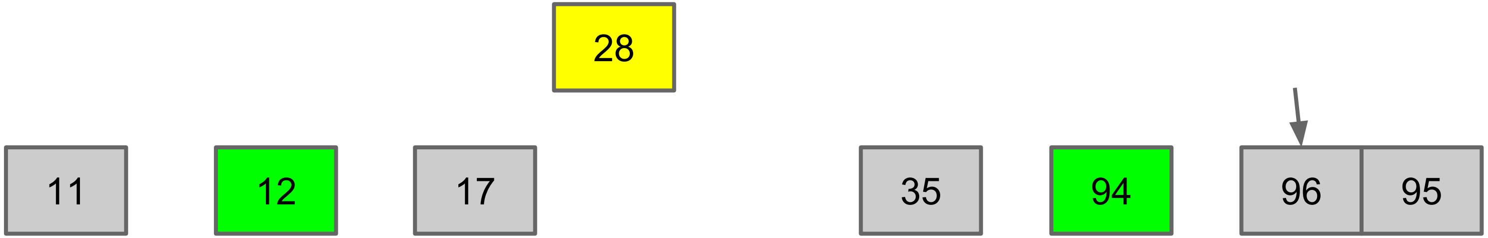
Quick Sort

Create two arrays



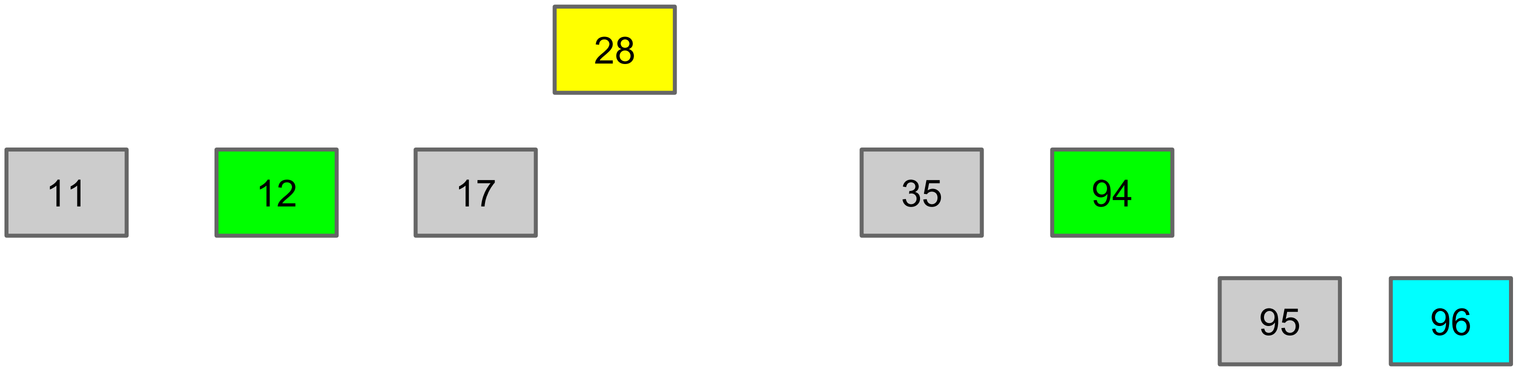
Quick Sort

Continue. quicksort(left) and quicksort (right). Choose pivot(s)



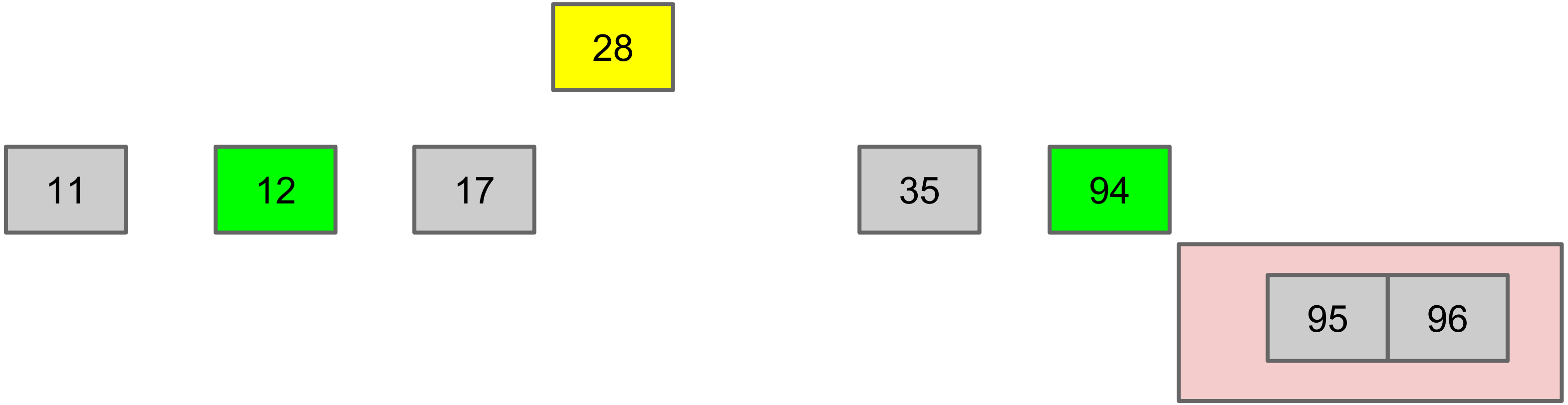
Quick Sort

Break into two arrays



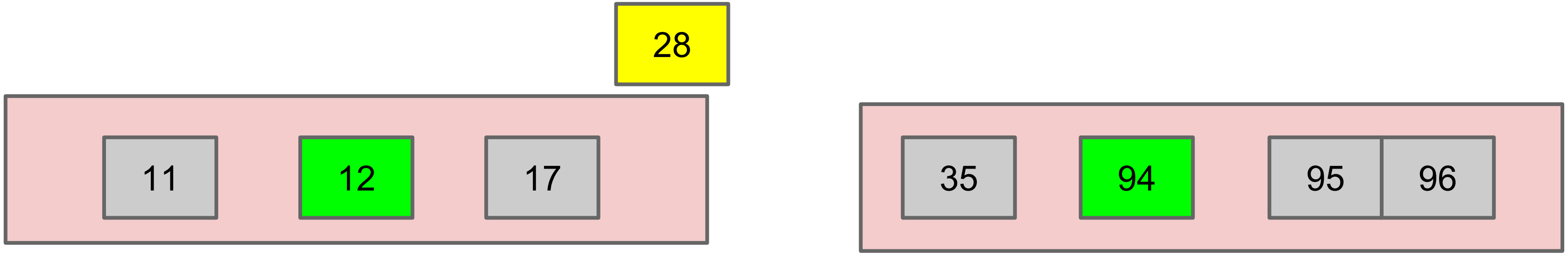
Quick Sort

Move back up recursion. Join lists.
Right + pivot + left



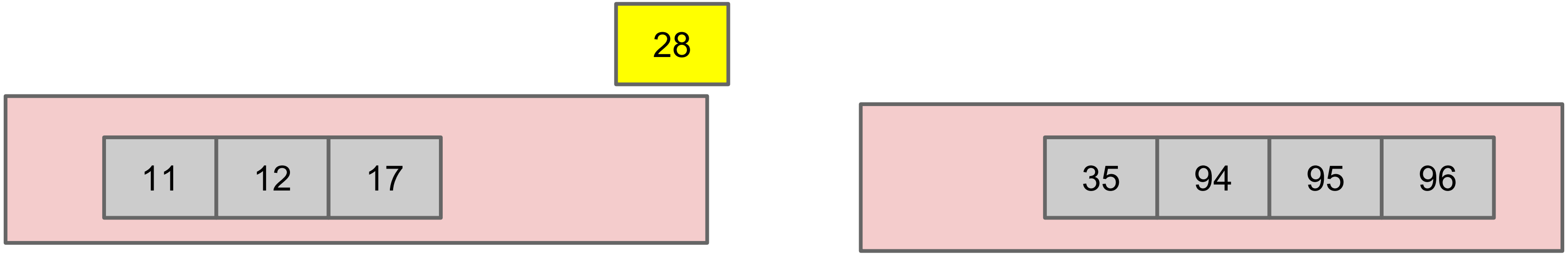
Quick Sort

Move back up recursion. Join lists.
Right + pivot + left



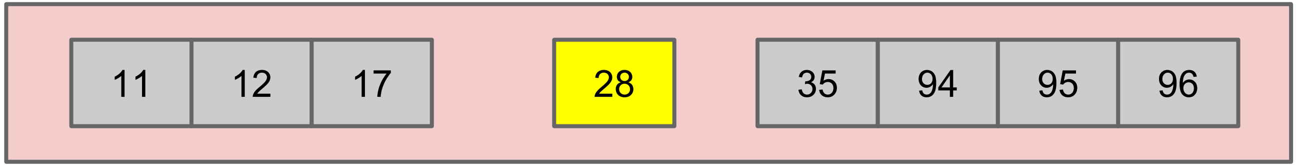
Quick Sort

Move back up recursion. Join lists.
Right + pivot + left



Quick Sort

Move back up recursion. Join lists.
Right + pivot + left



Quick Sort

Move back up recursion. Join lists.
Right + pivot + left

11	12	17	28	35	94	95	96

Quick Sort

Done

11	12	17	28	35	94	95	96

Recursion Examples: Quick Sort

- How can we implement this recursively?

Recursion Examples: Quick Sort

- How can we implement this recursively?
- What are the base cases?

Recursion Examples: Quick Sort

```
void quickSort(int* pArray, int start, int end){
	if (start < end){
		int p = partition(pArray, start, end);

		quickSort(pArray, start, p - 1);

		quickSort(pArray, p + 1, end);
	}
}
```

Recursion Examples: Quick Sort

```
int partition(int* pArray, int start, int end){
	int pivotValue = pArray[start];
	int pivotPosition = start;
	for (int pos = start + 1; pos <= end; pos++){
		if (pArray[pos] < pivotValue){
			swap(pArray[pivotPosition + 1], pArray[pos]);
			swap(pArray[pivotPosition], pArray[pivotPosition + 1]);
			pivotPosition++;
		}
	}
	return pivotPosition;
}
```

Recursion Examples: Towers of Hanoi

- The Towers of Hanoi is a classic example of a problem that is simple with recursion but otherwise difficult

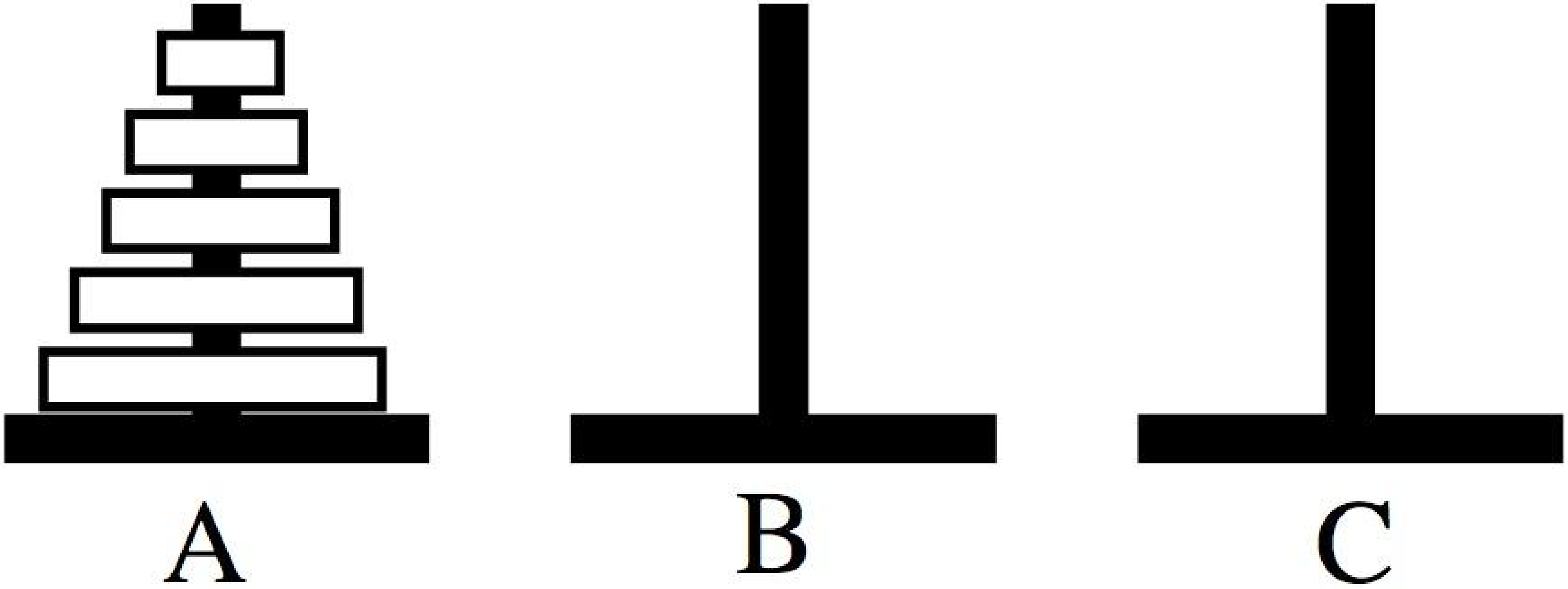
Recursion Examples: Towers of Hanoi

- In the Towers of Hanoi, the rules are
  - All disks must rest on a peg except while moving
  - Only one disk may move at a time
  - No large disk can be placed on top of a smaller disk

Recursion Examples: Towers of Hanoi

- The game
  - A board exists with three pegs in a row
  - All of the disks are stacked greatest to least on a single peg
  - The pegs must be moved from peg one to peg three

Recursion Examples: Towers of Hanoi



Recursion Examples: Towers of Hanoi

- The myth/Legend
  - There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the immutable rules of the Brahma, since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. According to the legend, when the last move of the puzzle will be completed, the world will end

http://en.wikipedia.org/wiki/Tower_of_Hanoi

Recursion Examples: Towers of Hanoi

- The myth/Legend
  - There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the immutable rules of the Brahma, since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. According to the legend, when the last move of the puzzle will be completed, the world will end

  http://en.wikipedia.org/wiki/Tower_of_Hanoi

- The minimum number of moves is $2^N - 1$

Recursion Examples: Towers of Hanoi

- The myth/Legend
    - There is a story about an Indian temple in Kashi Vishwanath which contains a large room with three time-worn posts in it surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks, in accordance with the immutable rules of the Brahma, since that time. The puzzle is therefore also known as the Tower of Brahma puzzle. According to the legend, when the last move of the puzzle will be completed, the world will end

http://en.wikipedia.org/wiki/Tower_of_Hanoi

- The minimum number of moves is $2^N - 1$
    - for $N = 64$, $2^{64} - 1 == 1.8446744e+19$

Recursion Examples: Towers of Hanoi

- How do we solve this?

Recursion Examples: Towers of Hanoi

- How do we solve this?

To move n disks from peg 1 to peg 3, using peg 2 as a temporary peg

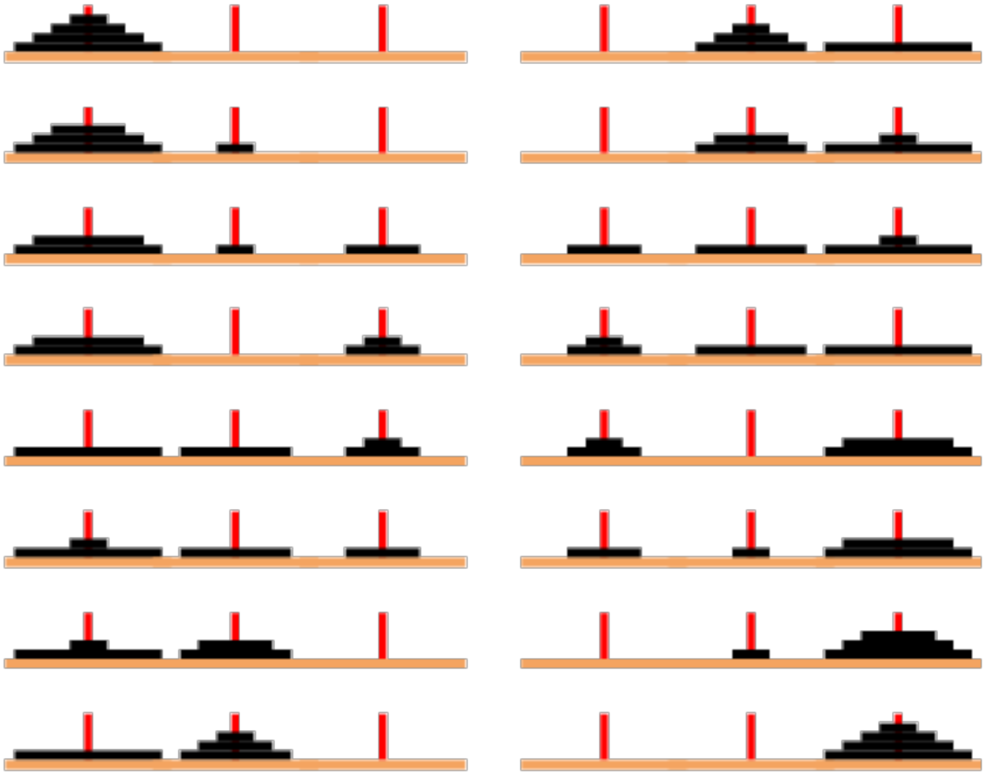
if n > 0 Then

Move n - 1 disks from peg 1 to peg 2, using peg 3 as temporary peg

Move a disk from peg 1 to peg 3

Move n - 1 disks from peg 2 to peg 3, using peg 1 as a temporary peg

Recursion Examples: Towers of Hanoi



Recursion Examples: Towers of Hanoi

- How do we implement the solution recursively?

Recursion Examples: Towers of Hanoi

- How do we implement the solution recursively?

- What are our base cases?

Recursion Examples: Towers of Hanoi

HINT:

```
void moveDisks(int n, string source, string dest, string temp){
...
}

moveDisks(3, “peg 1”, “peg 3”, “peg 2”);
```

Recursion Examples: Towers of Hanoi

```
void moveDisks(int n, string source, string dest, string temp){
        if (n > 0) {
                moveDisks(n - 1, source, temp, dest);

                cout << “Move a disk from “ << source << “ to “ << dest << endl;

                moveDisks(n - 1, temp, dest, source);
        }
}

moveDisks(3, “peg 1”, “peg 3”, “peg 2”);
```

Exhaustive & Enumeration Algorithms

- Some problems require a different approach

Exhaustive & Enumeration Algorithms

- Enumeration Algorithm
  - An algorithm that generates all possible combinations of items of a certain type

Exhaustive & Enumeration Algorithms

- Enumeration Algorithm
  - An algorithm that generates all possible combinations of items of a certain type
- Exhaustive Algorithm
  - An algorithm that searches through an enumeration set to find the best one

Exhaustive & Enumeration Algorithms

- IE examine all possible combinations and choose the best one

Exhaustive & Enumeration Algorithms

- IE examine all possible combinations and choose the best one

- One great example of this type of algorithm is making change in US currency

Exhaustive & Enumeration Algorithms

- What currency do we have?

Exhaustive & Enumeration Algorithms

- What currency do we have?

- How would you make change for $1.21?

Exhaustive & Enumeration Algorithms

- What currency do we have?

- How would you make change for $1.21?
  - What way is the best (least number of coins and bills)?

Exhaustive & Enumeration Algorithms

- What currency do we have?

- How would you make change for $1.21?
  - What way is the best (least number of coins and bills)? - Greedy Strategy
  - What way is best (most number of coins)?

Exhaustive & Enumeration Algorithms

- How can we implement this algorithm using recursion?

Exhaustive & Enumeration Algorithms

- How can we implement this algorithm using recursion?

- What are the base cases?

Exhaustive & Enumeration Algorithms

- The greedy algorithm doesn’t work for all monetary systems

Exhaustive & Enumeration Algorithms

- The greedy algorithm doesn't work for all monetary systems
  - Make change for $.44 with only coins .01, .20, .25

Exhaustive & Enumeration Algorithms

- The greedy algorithm doesn’t work for all monetary systems
  - Make change for $.44 with only coins .01, .20, .25
  - Greedy says .25, and 19 x .01

Exhaustive & Enumeration Algorithms

- The greedy algorithm doesn’t work for all monetary systems
  - Make change for $.44 with only coins .01, .20, .25
  - Greedy says .25, and 19 x .01
  - The best is 2 x .20 and 4 x .01

Exhaustive & Enumeration Algorithms

- How would we calculate the number of different ways to make change for a specified amount?

Exhaustive & Enumeration Algorithms

```
const int COIT_SET_SIZE = 6;
const int coinValues[] = {1, 5, 10, 25, 50, 100};
int mkChange(int amount, int largestIndex) {

}
```

Exhaustive & Enumeration Algorithms

```
const int COIT_SET_SIZE = 6;
const int coinValues[] = {1, 5, 10, 25, 50, 100};
int mkChange(int amount, int largestIndex) {
        while (coinValue[largestIndex] > amount)
                largestIndex--;
        if (amount == 0 || largestIndex == 0)
                return 1;
        int nWays = 0, nCouns = 0;
        while (nCoins <= amount / coinValues[largestIndex]) {
                int amountLeft = amount - nCouns * coinValues[largestIndex];
                nWays = nWays + mkChange(amountLeft, largestIndex - 1);
                nCoins++;
        }
        return nWays;
}
```

Exhaustive & Enumeration Algorithms

- How could we modify the example to give us the least number of coins?