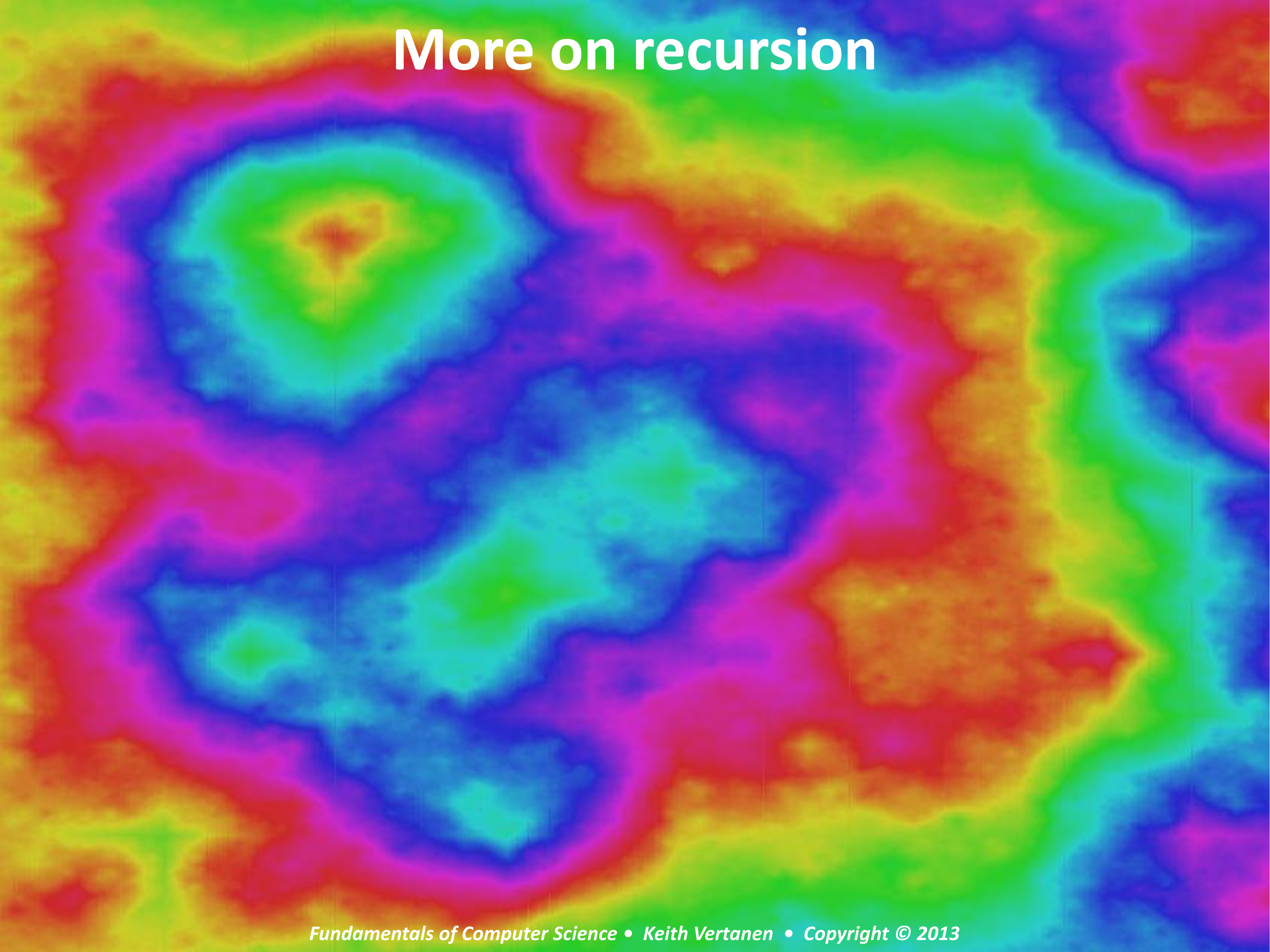
More on recursion



Overview

- Recursion
  - A method calling itself
    - A new way of thinking about a problem
    - A powerful programming paradigm
- Examples:
  - Last time:
    - Factorial, binary search, H-tree, Fibonacci
  - Today:
    - Greatest Common Divisor (GCD)
    - Brownian Motion
    - Sorting things

Greatest Common Divisor

- GCD
  - Find largest integer d that evenly divides p and q
  - e.g. gcd(4032, 1272) = 24
    - $4032 = 2^6 \times 3^2 \times 7^1$
    - $1272 = 2^3 \times 3^1 \times 53^1$
    - $\gcd = 2^3 \times 3^1 = 24$
- Applications
  - Simplify fractions: 1272/4032 = 53/168
  - RSA cryptography

Greatest Common Divisor

- GCD
  - Find largest integer d that evenly divides p and q
- Euclid's algorithm (300 BC)

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \% q) & \text{otherwise} \end{cases}$$

base case

reduction step,
converges to base case

```
gcd(4032, 1272) = gcd(1272, 216)
                = gcd(216, 192)
                = gcd(192, 24)
                = gcd(24, 0)
                 = 24
```

4032 = 3 x 1272 + 216

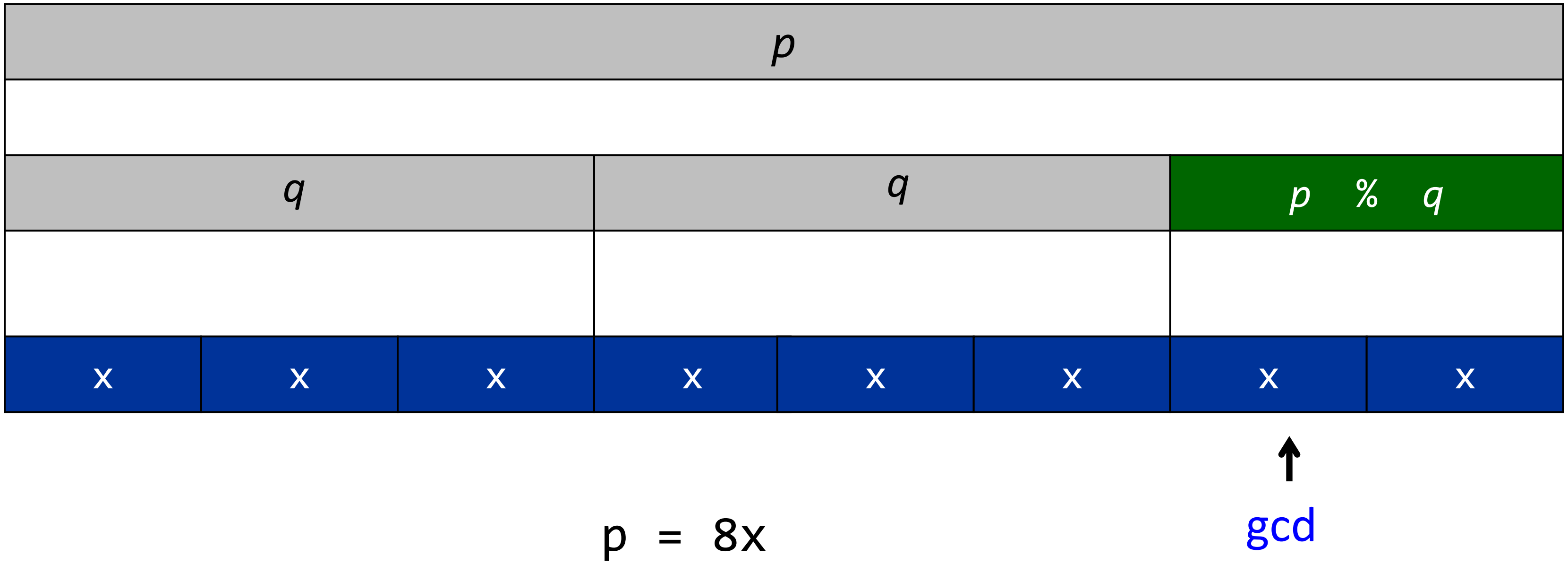
Greatest Common Divisor

- GCD
  - Find largest integer d that evenly divides p and q

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \% q) & \text{otherwise} \end{cases}$$

← base case

← reduction step, converges to base case



```
p = 8x
q = 3x
gcd(p, q) = x
```

Greatest Common Divisor

- GCD
  - Find largest integer d that evenly divides p and q

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q,\ p \,\%\, q) & \text{otherwise} \end{cases}$$

base case

reduction step, converges to base case

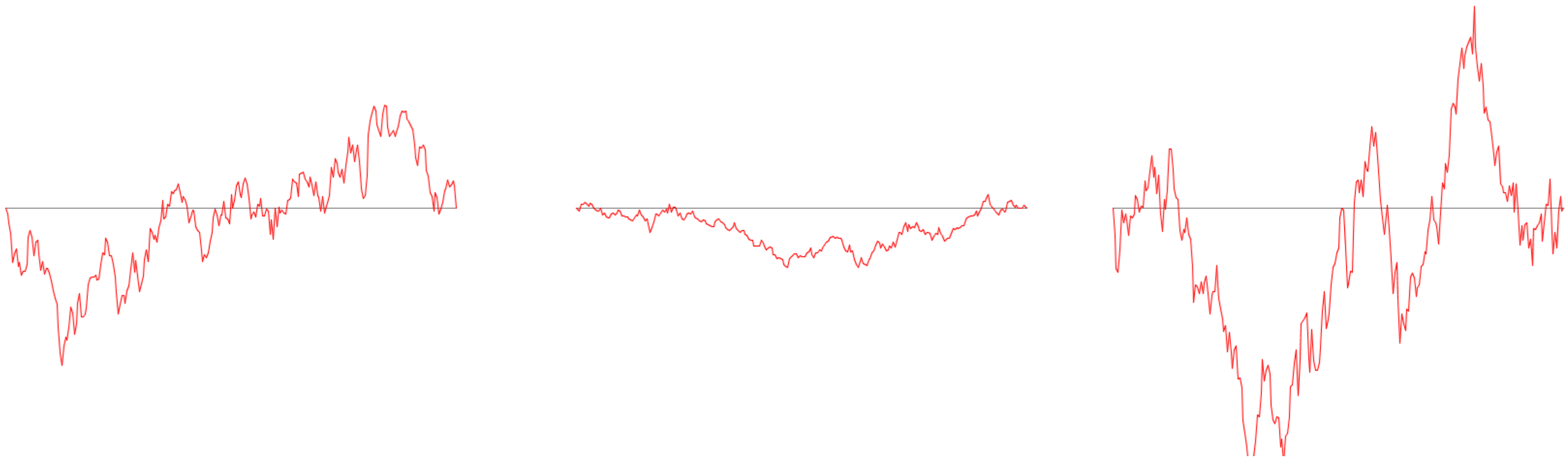
```java
public static int gcd(int p, int q)
{
    if (q == 0)
        return p;
    else
        return gcd(q, p % q);
}
```

base case

reduction step

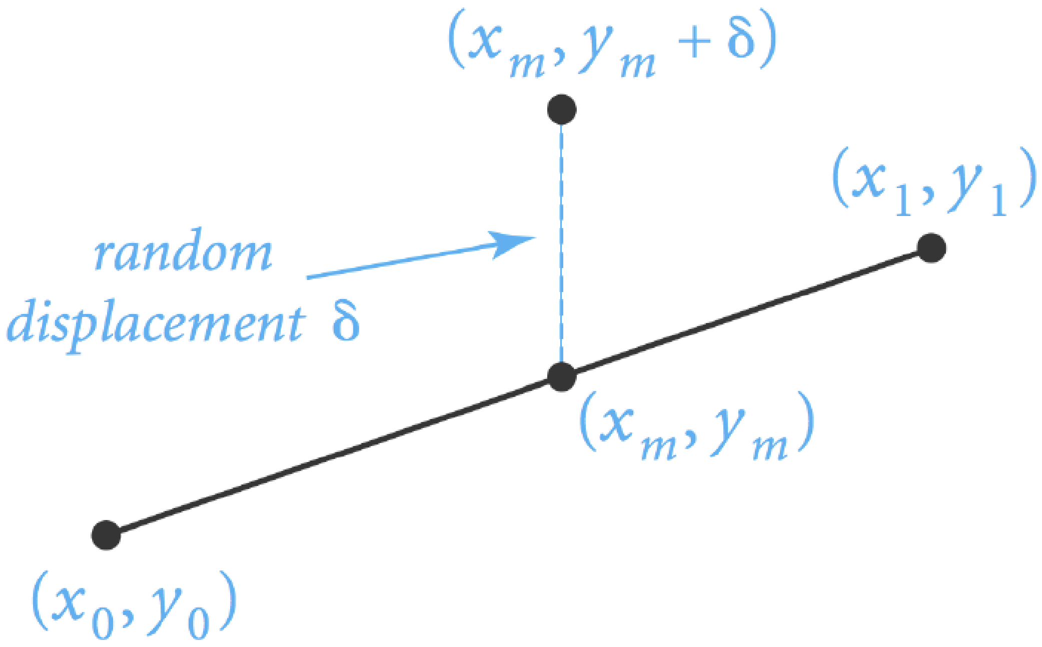
Brownian motion

- Physical process that models many natural and artificial phenomenon
  - – Price of stocks
  - – Rugged shapes of mountains and clouds
  - – Fractal landscape and textures for computer graphics

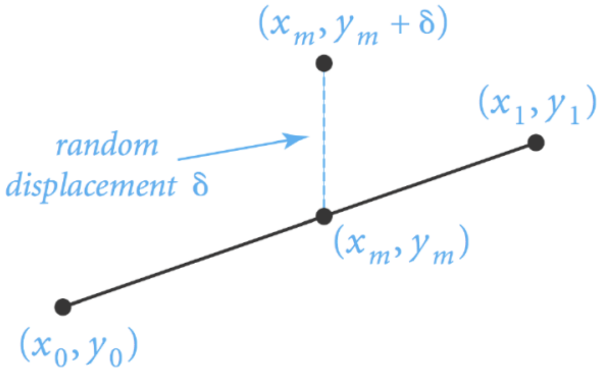


Simulating Brownian Motion

- Midpoint displacement method
  - Track interval (x_0, y_0) to (x_1, y_1)
  - Choose δ randomly from Gaussian distribution
  - Divide in half, $x_m = (x_0+x_1)/2$ and $y_m = (y_0+y_1)/2 + \delta$
  - Recur on the left and right intervals



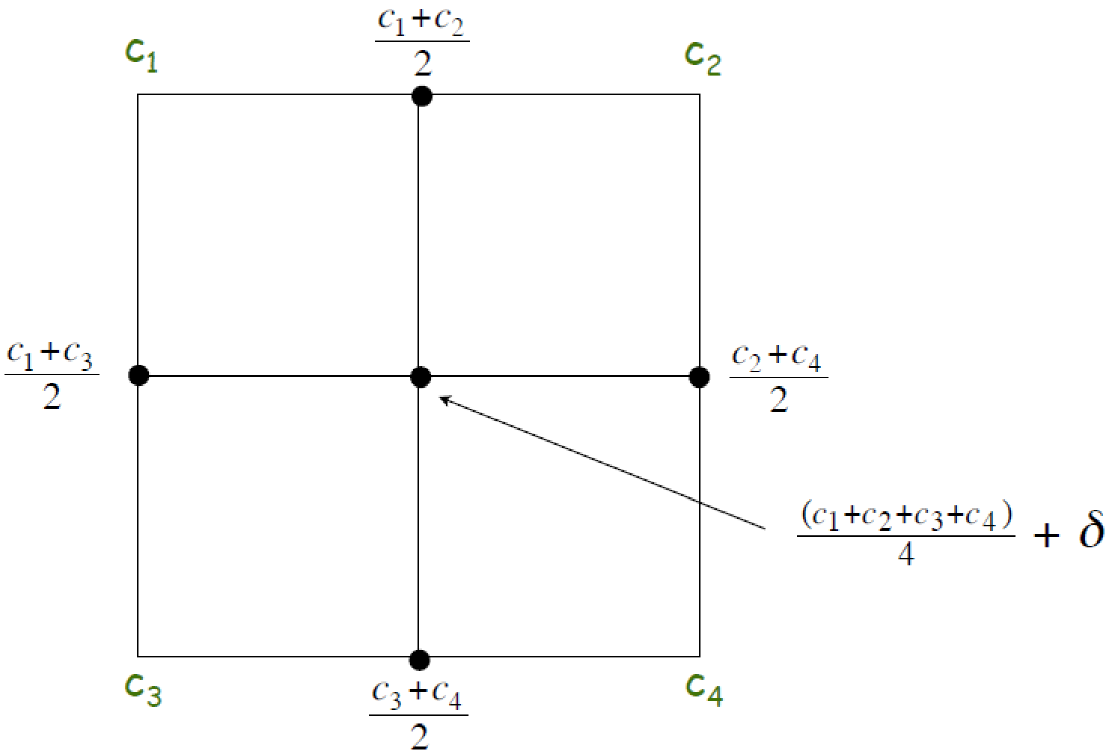
Simulating Brownian Motion

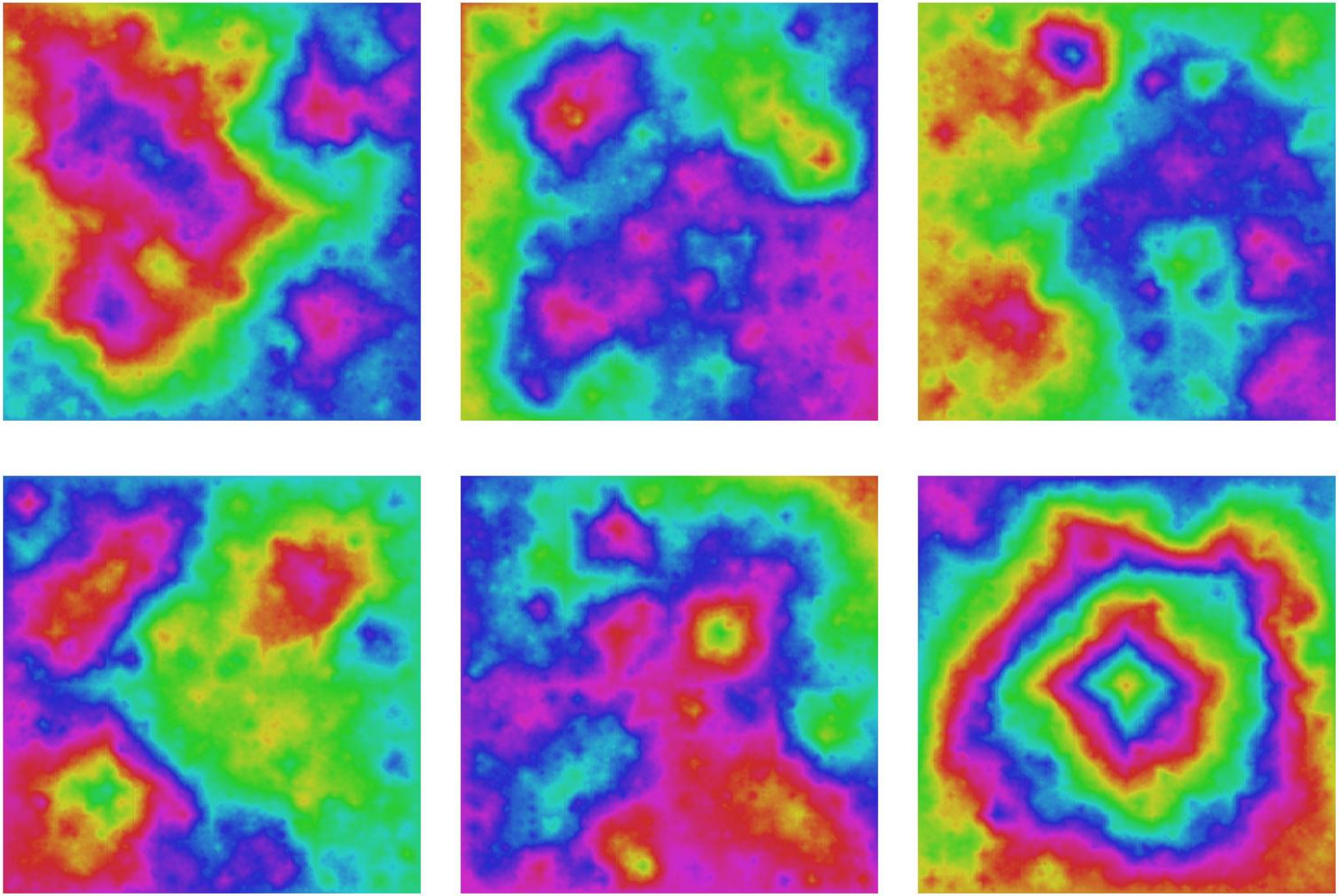


```
void curve(double x0, double y0, double x1, double y1, double var)
{
   if (x1 - x0 < .005)
   {
      StdDraw.line(x0, y0, x1, y1);
      return;
   }

   double xm = (x0 + x1) / 2.0;
   double ym = (y0 + y1) / 2.0;

   ym = ym + StdRandom.gaussian(0, Math.sqrt(var));

   curve(x0, y0, xm, ym, var / 2.0);
   curve(xm, ym, x1, y1, var / 2.0);
}
```

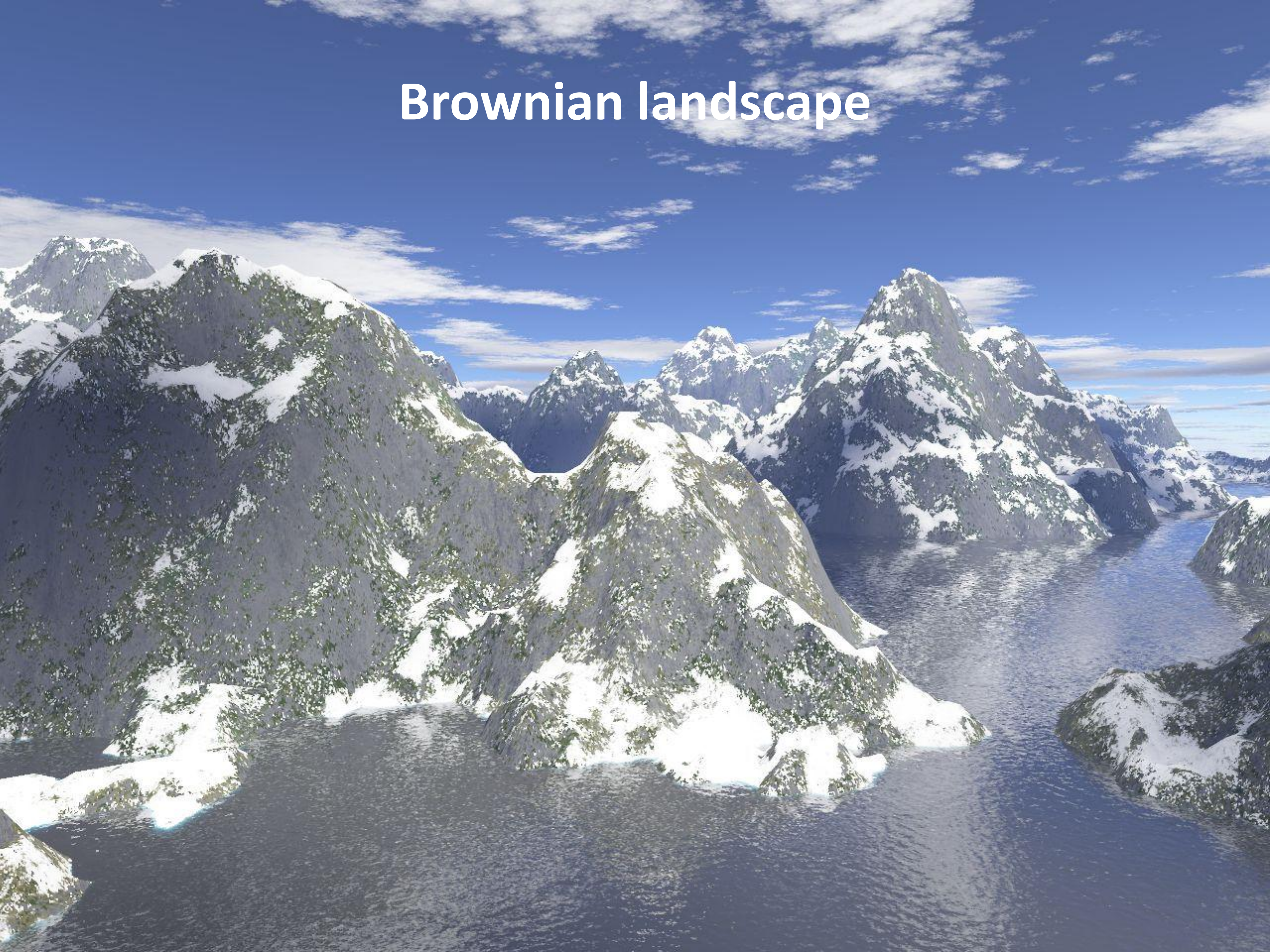
Plasma cloud

- Same idea, but in 2D
  - Each corner of square has some greyscale value
  - Divide into four sub-squares
  - New corners: avg of original corners, or all 4 + random
  - Recur on four sub-squares





Brownian landscape



Divide and conquer

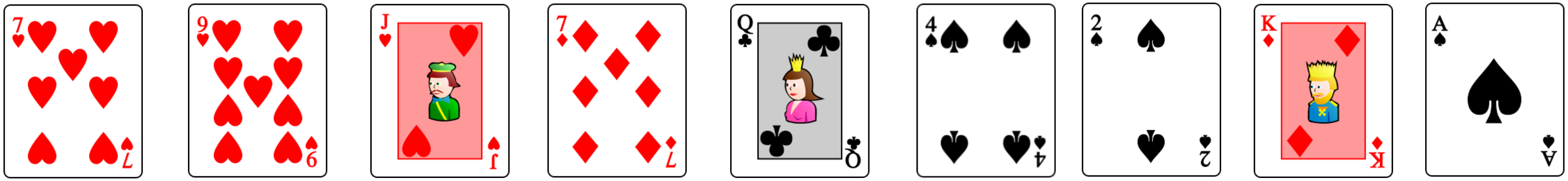
- Divide and conquer paradigm
  - Break big problem into small sub-problems
  - Solve sub-problems recursively
  - Combine results

> "Divide et impera. Vendi, vidi, vici."
> *-Julius Caesar*

- Used to solve many important problems
  - Mergesort, sorting things, O(N log N)
  - Parsing programming languages
  - Discrete FFT, signal processing
  - Multiplying large numbers
  - Traversing multiply linked structures (stay tuned)

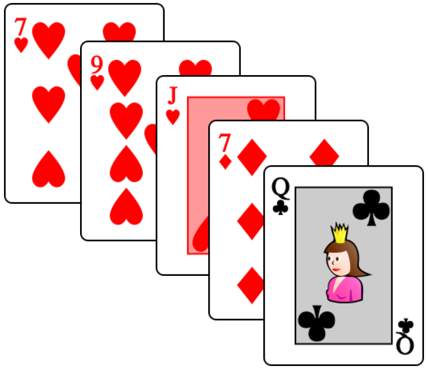
Divide and conquer: sorting

- Goal: Sort by number, ignore suit, aces high

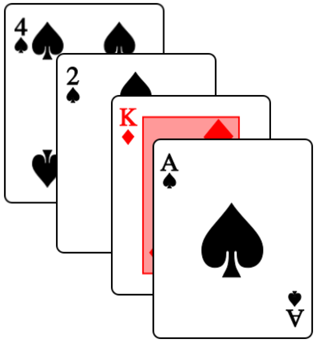


Approach

1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



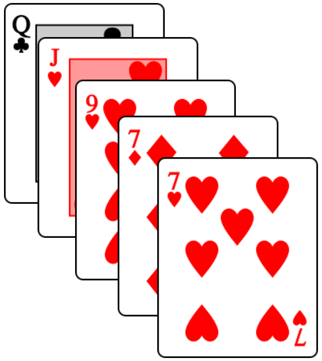
Unsorted pile #1



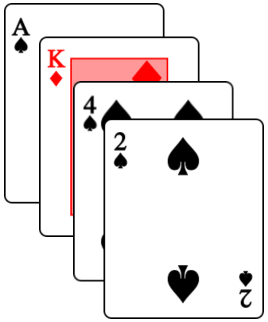
Unsorted pile #2

Approach

1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1



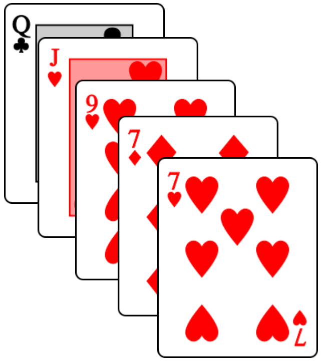
Sorted pile #2

Merging

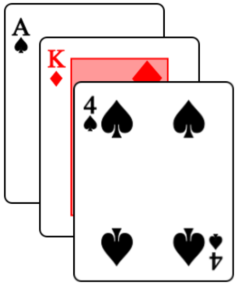
Take card from whichever pile has lowest card

Approach

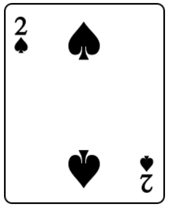
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

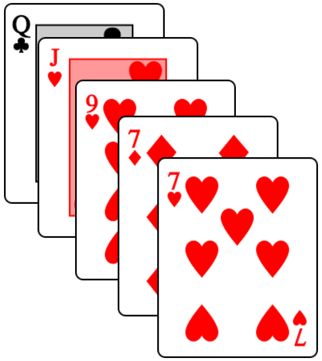


Sorted pile #2

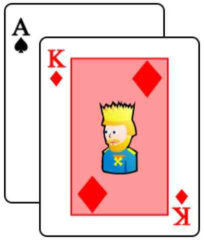


Approach

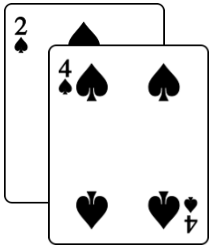
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

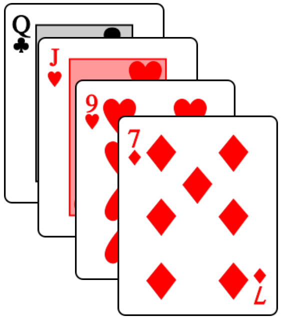


Sorted pile #2

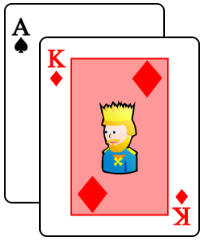


Approach

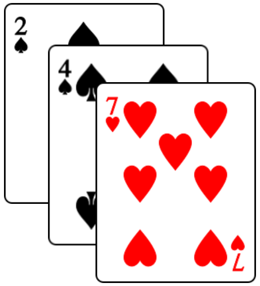
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

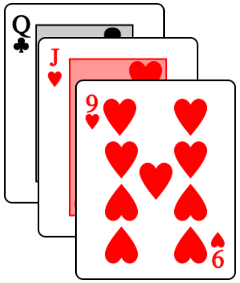


Sorted pile #2

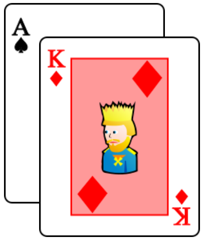


Approach

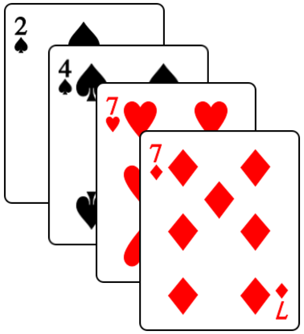
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

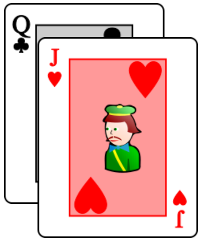


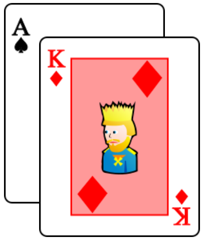
Sorted pile #2



Approach

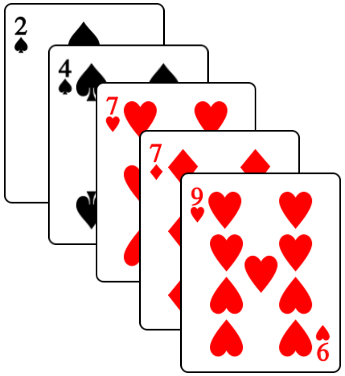
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together





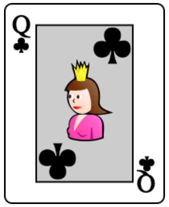
Sorted pile #1

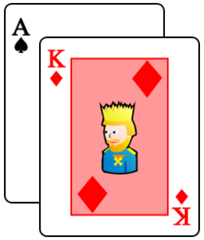
Sorted pile #2



Approach

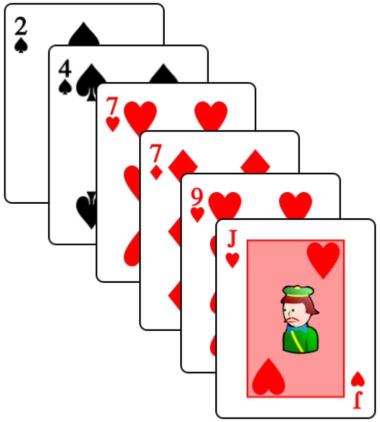
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together





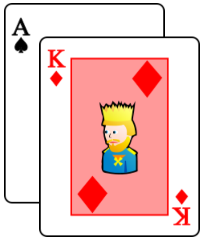
Sorted pile #1

Sorted pile #2



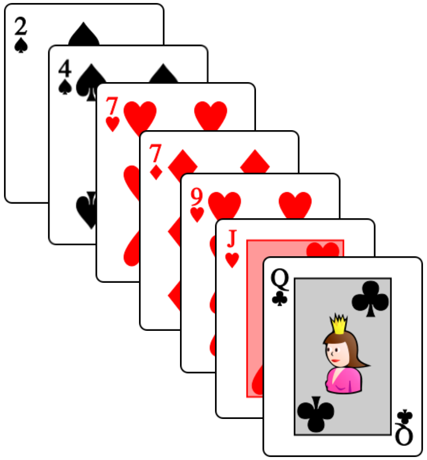
Approach

1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



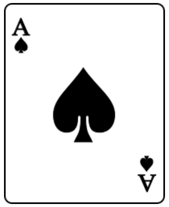
Sorted pile #1

Sorted pile #2



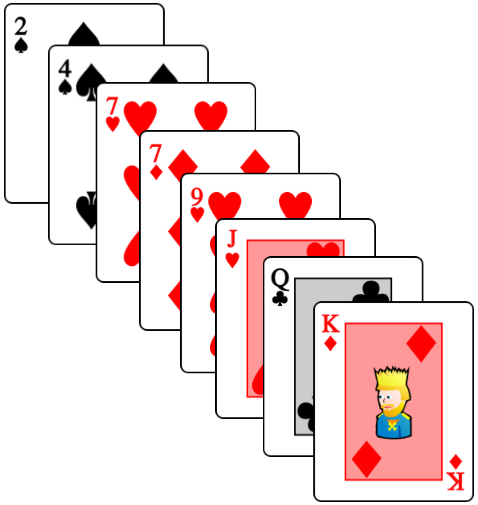
Approach

1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

Sorted pile #2

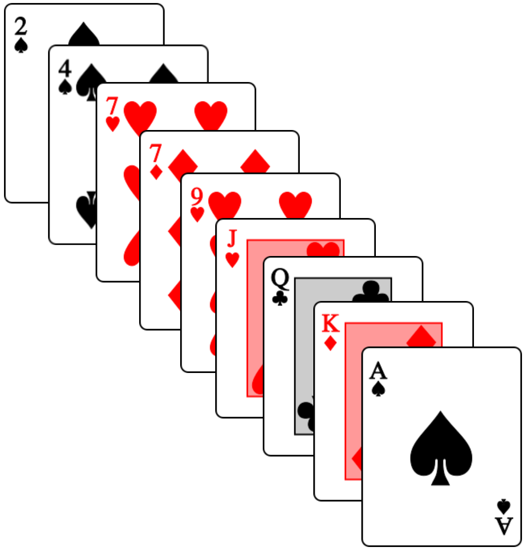


Approach

1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together

Sorted pile #1

Sorted pile #2



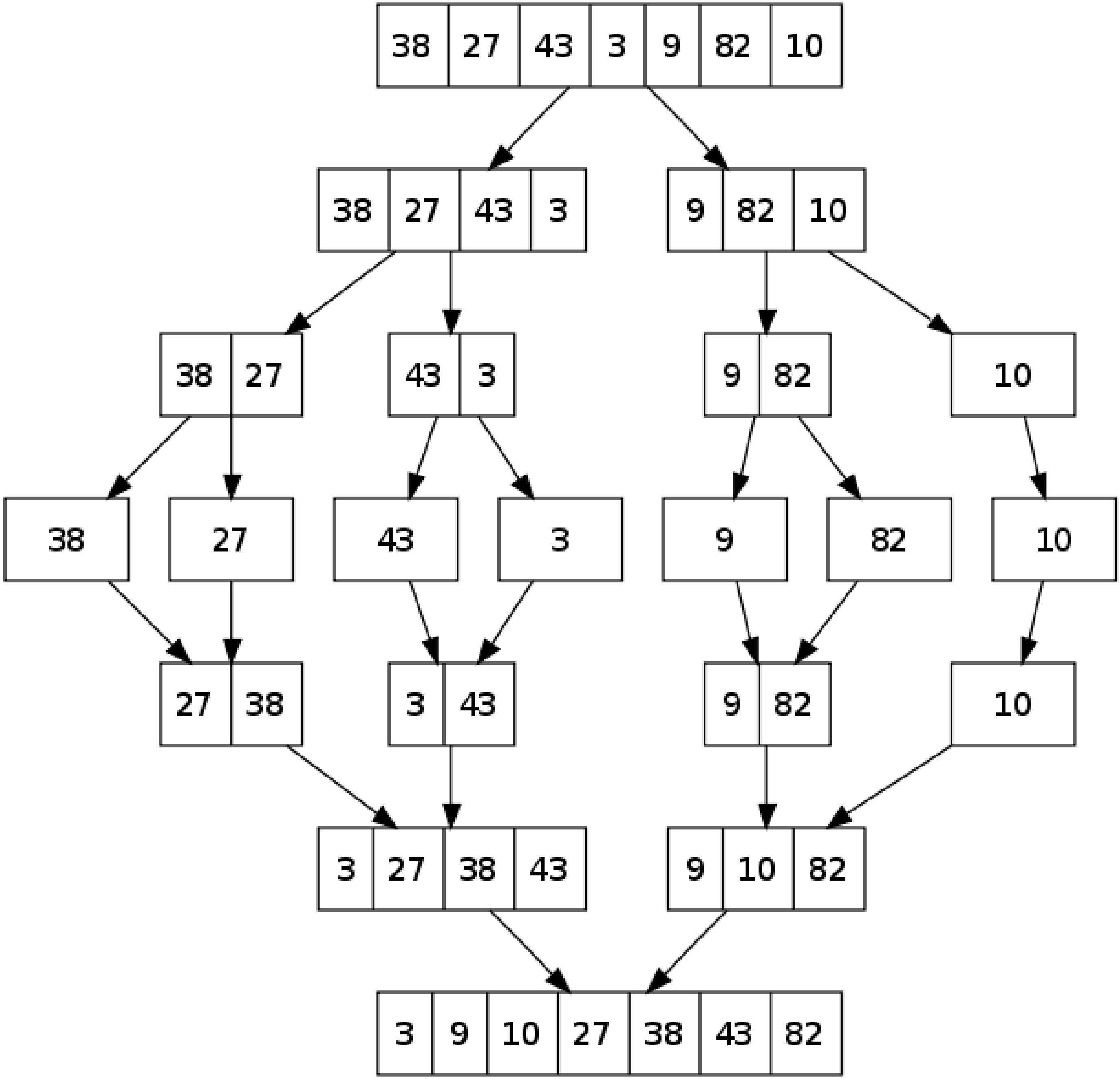
How many operations to do the merge?

Linear in the number of cards, O(N)

But how did pile 1 and 2 get sorted?

Recursively of course!

Split each pile into two halves, give to different people to sort.



38 27 43 3 9 82 10

38 27 43 3 9 82 10

38 27 43 3 9 82 10

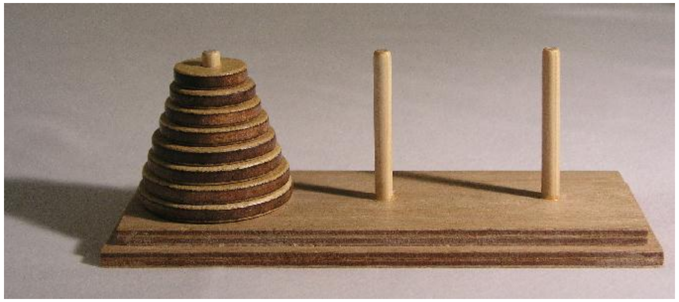
38 27 43 3 9 82 10

27 38 3 43 9 82 10

3 27 38 43 9 10 82

3 9 10 27 38 43 82

Summary



- Recursion
  - A method calling itself:
    - Sometimes just once, e.g. binary search
    - Sometimes twice, e.g. mergesort
    - Sometimes multiple times, e.g. H-tree
  - All good recursion must come to an end
    - Base case that does NOT call itself recursively
  - A powerful tool in computer science
    - Allows elegant and easy to understand algorithms
    - (Once you get your head around it)