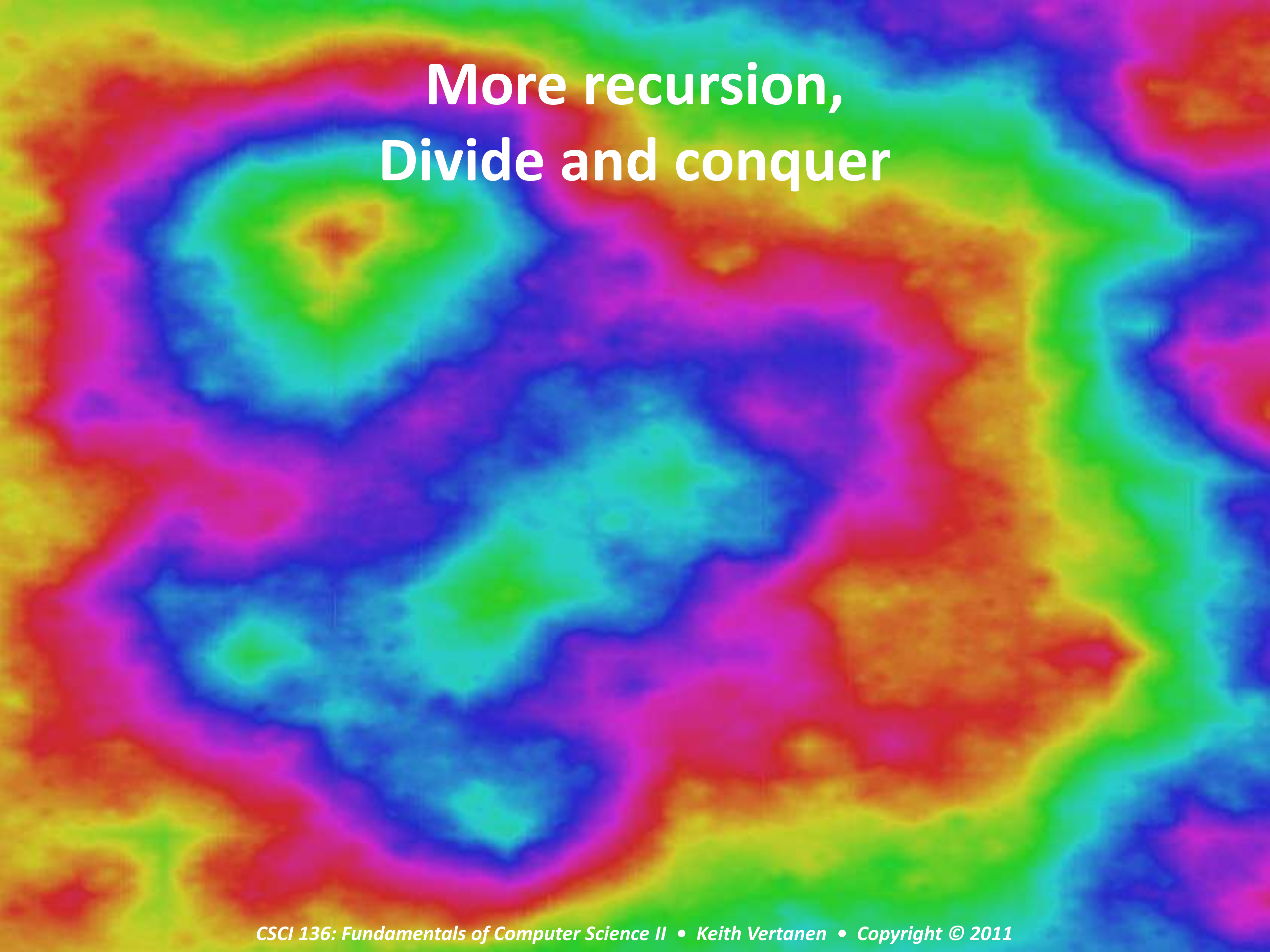


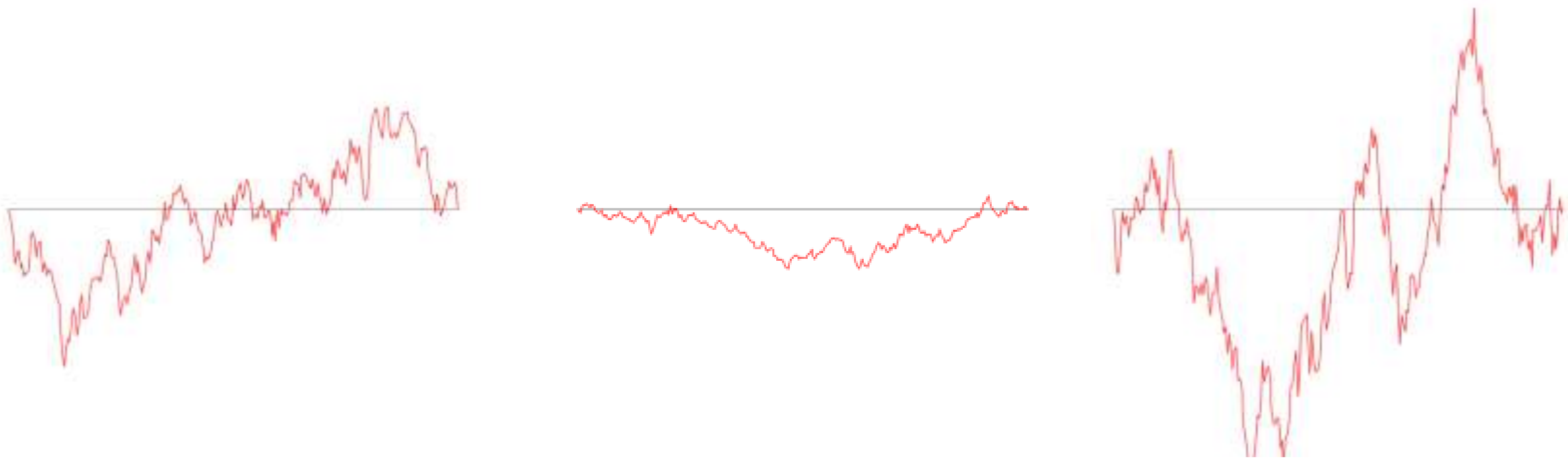
# More recursion, Divide and conquer

# Overview

- More recursion
  - Recursion + randomness = pretty pictures
  - Example 1: Brownian motion
  - Example 2: Plasma cloud
- Divide and conquer
  - Useful example of recursion
  - Common computer science way to solve problems
  - Example: Sorting

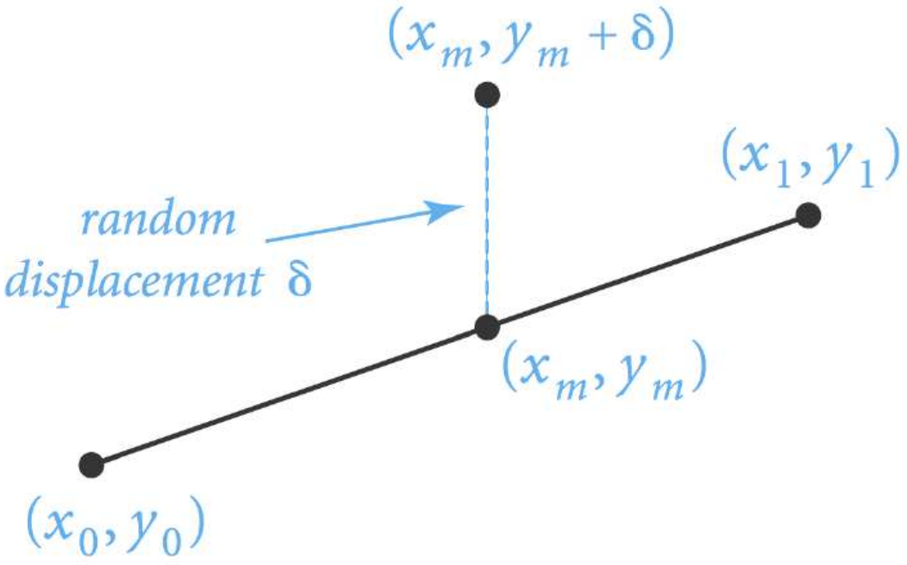
# Brownian motion

- 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.

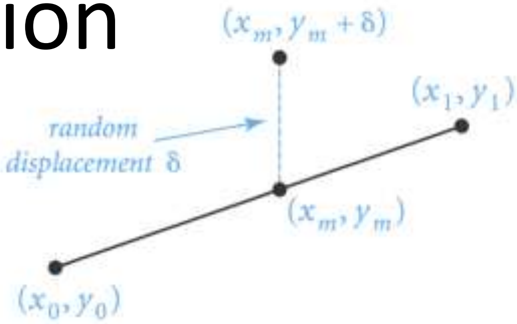


# Simulating Brownian Motion

- Midpoint displacement method
  - Track interval ($x_0$, $y_0$) to ($x_1$, $y_1$)
  - Choose from $\delta$ 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

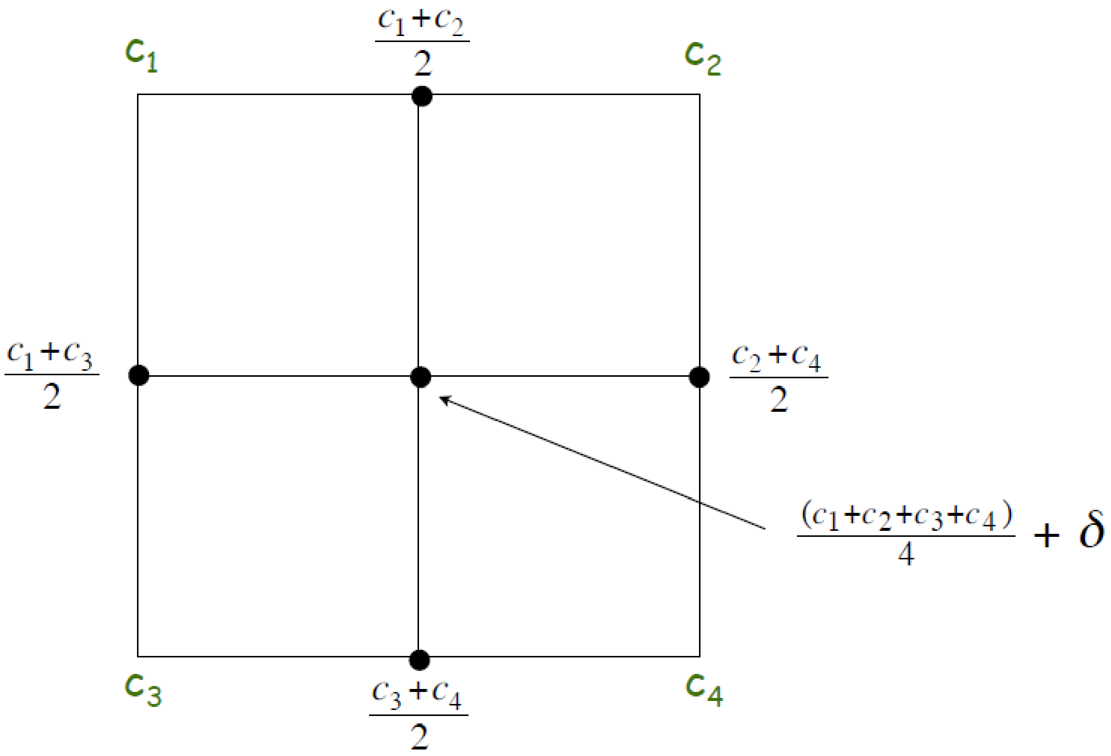


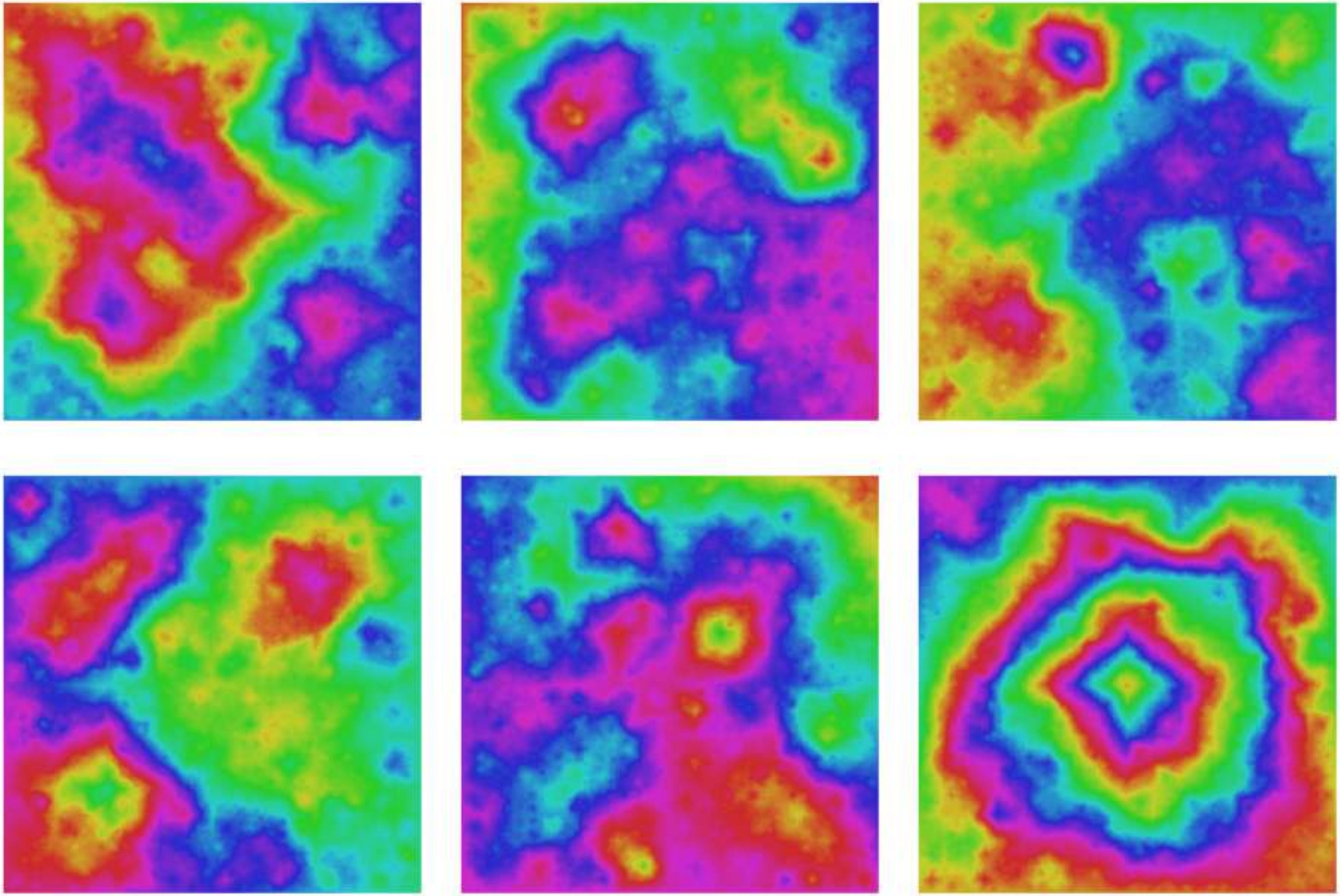
- Midpoint displacement method
  - Track interval $(x_0, y_0)$ to $(x_1, y_1)$
  - Choose from $\delta$ randomly from Gaussian distribution
  - Divide in half, $x_m = (x_0+x_1)/2$ and $y_m = (y_0+y_1)/2 + \delta$
  - Recur on the left and right intervals

```
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 are: avg. of original corners, or all four + random
  - Recur on four sub-squares





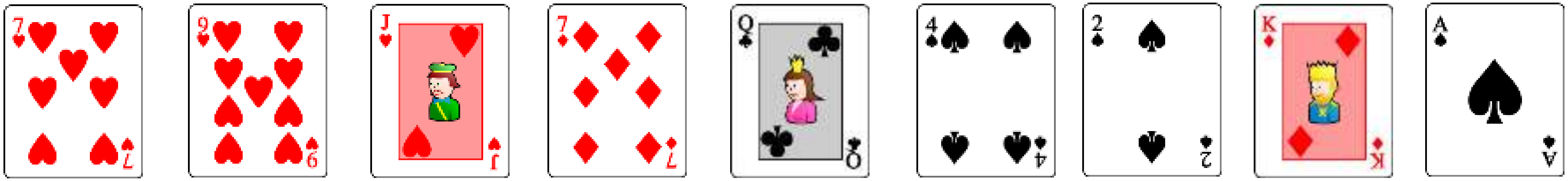
# Brownian landscape

# Divide and conquer

- Divide and conquer paradigm
  - Break big problem into small sub-problems
  - Solve sub-problems recursively
  - Combine results
- Used to solve many important problems
  - Mergesort, sorting things, O(N log N)
  - Syntactic analysis, parsing programming languages
  - Discrete FFT, signal processing
  - Multiplying large numbers
  - Traversing multiply linked structures (stay tuned)
    - e.g. Visiting all the nodes in a tree or graph

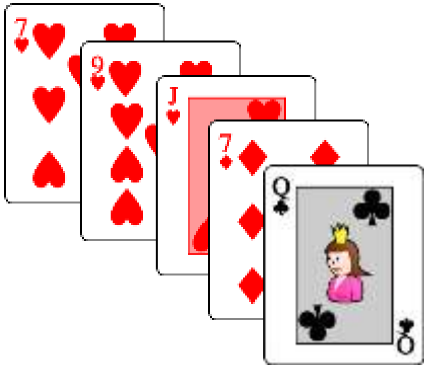
# Divide and conquer: sorting

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

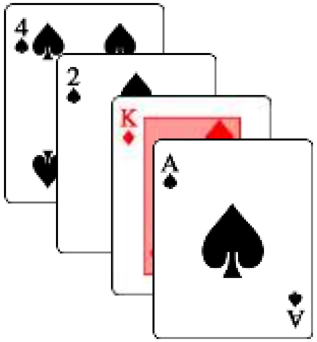


**Approach**

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



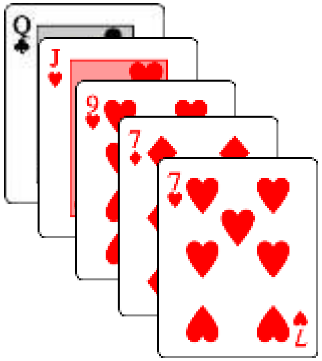
Unsorted pile #1



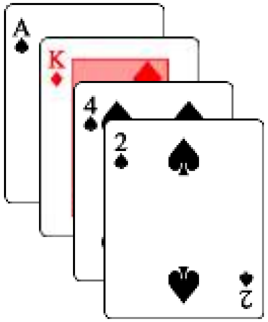
Unsorted pile #2

**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



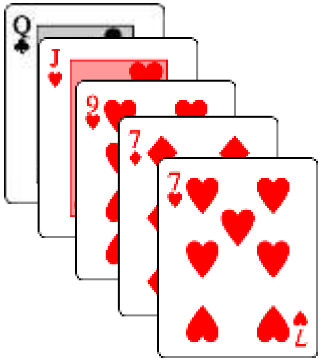
Sorted pile #2

**Merging**

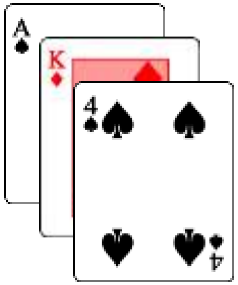
Take card from whichever pile has lowest card

**Approach**

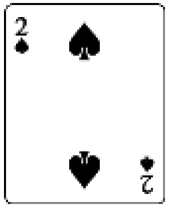
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

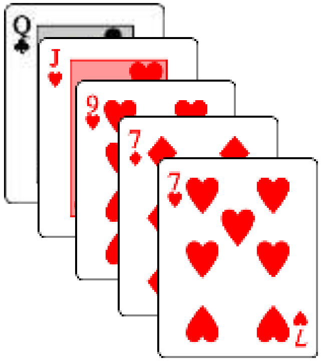


Sorted pile #2

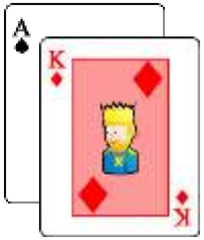


**Approach**

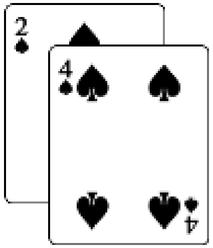
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

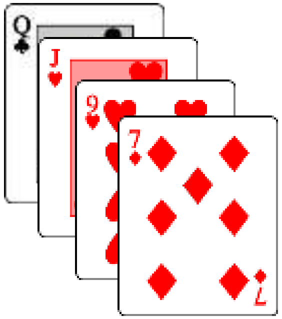


Sorted pile #2

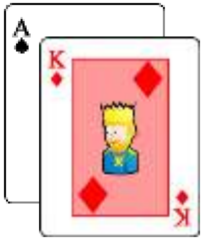


**Approach**

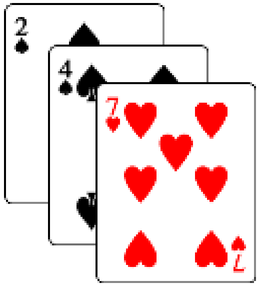
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

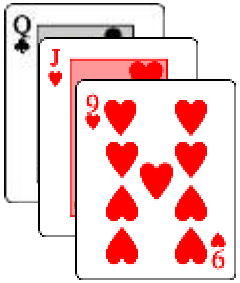


Sorted pile #2

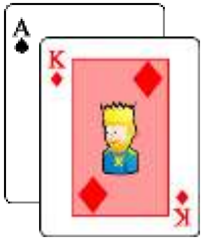


**Approach**

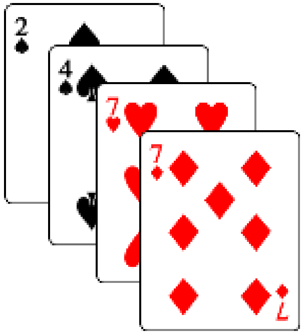
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

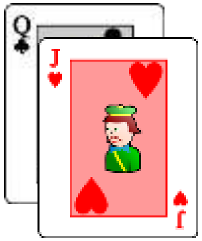


Sorted pile #2

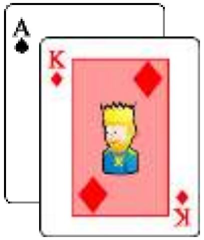


**Approach**

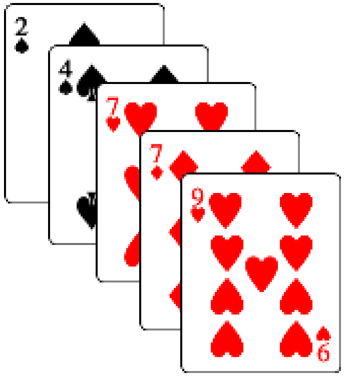
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together



Sorted pile #1

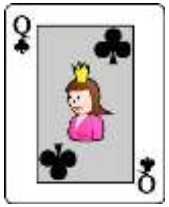


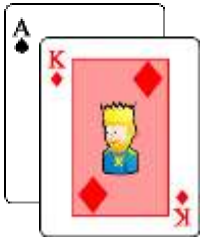
Sorted pile #2



**Approach**

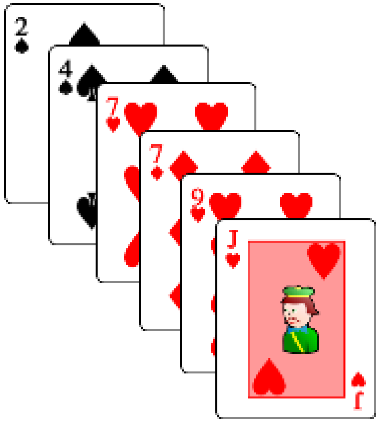
1) Split in half (or as close as possible)
2) Give each half to somebody to sort
3) Take two halves and merge together





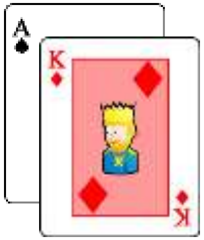
Sorted pile #1

Sorted pile #2



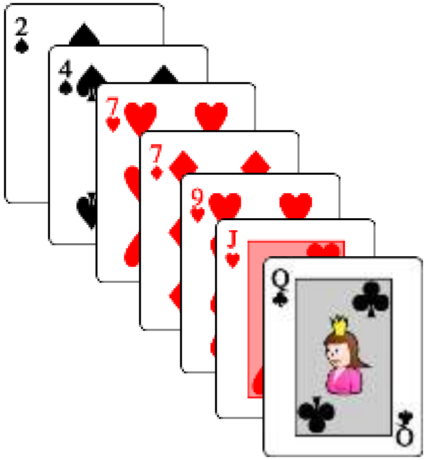
## Approach

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



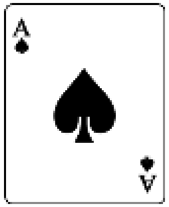
Sorted pile #1

Sorted pile #2



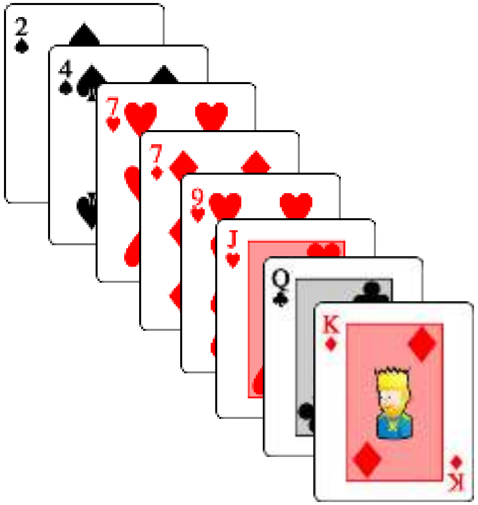
**Approach**

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

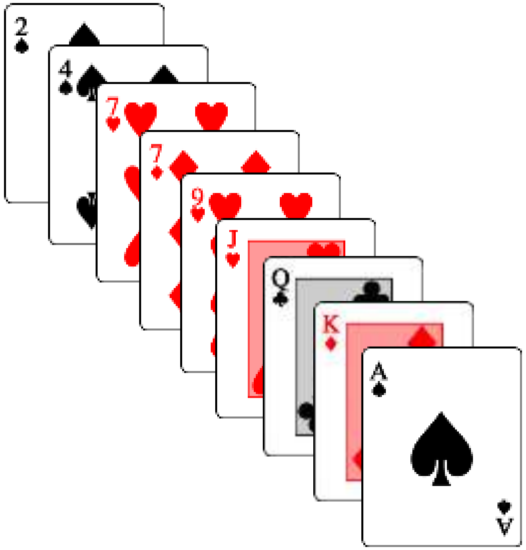


**Approach**

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

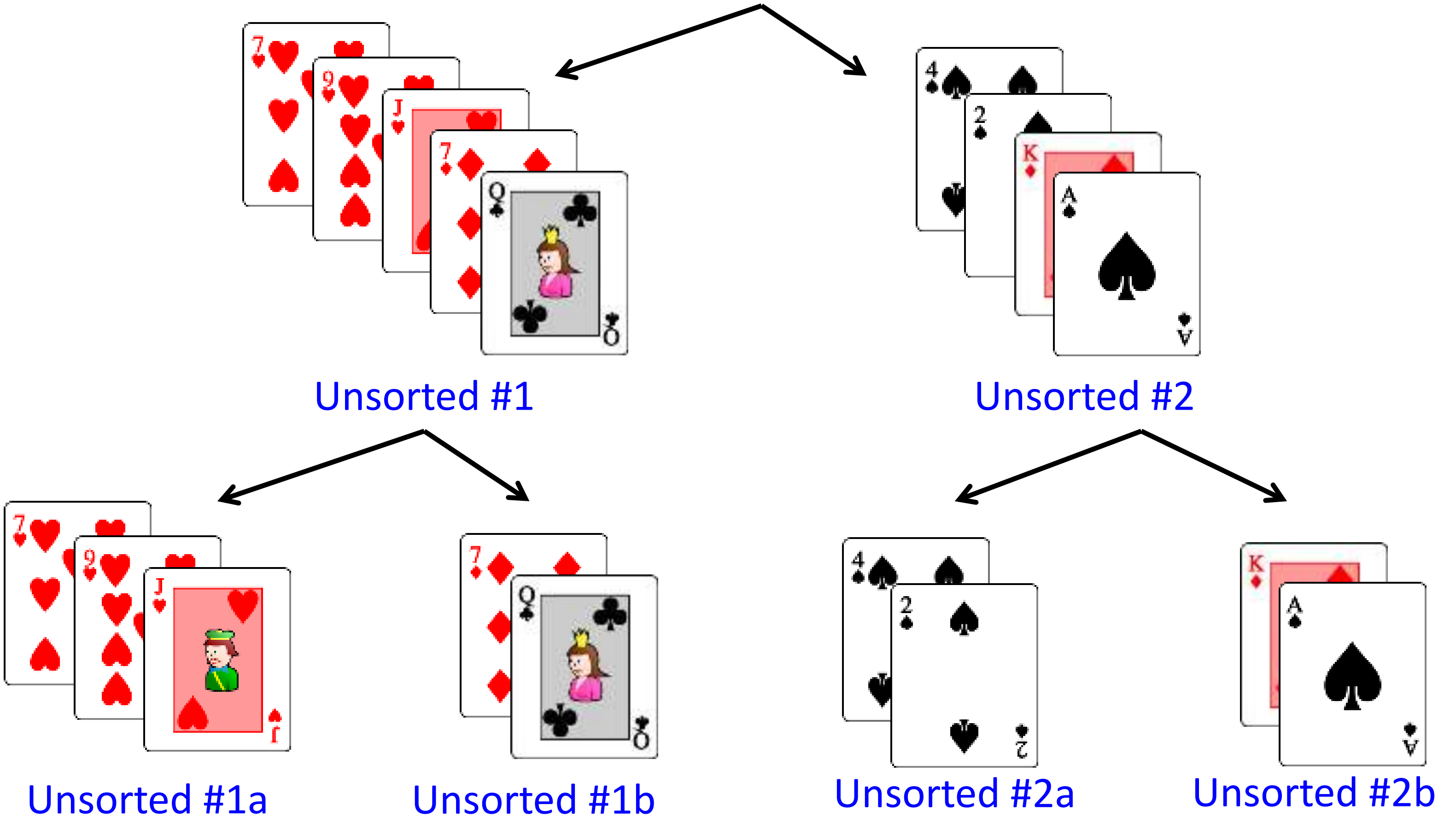
Sorted pile #2



How many operations to do the merge?

Linear in the number of cards, O(N)

**But how did pile 1 and 2 get sorted?**



Unsorted #1

Unsorted #2

Unsorted #1a

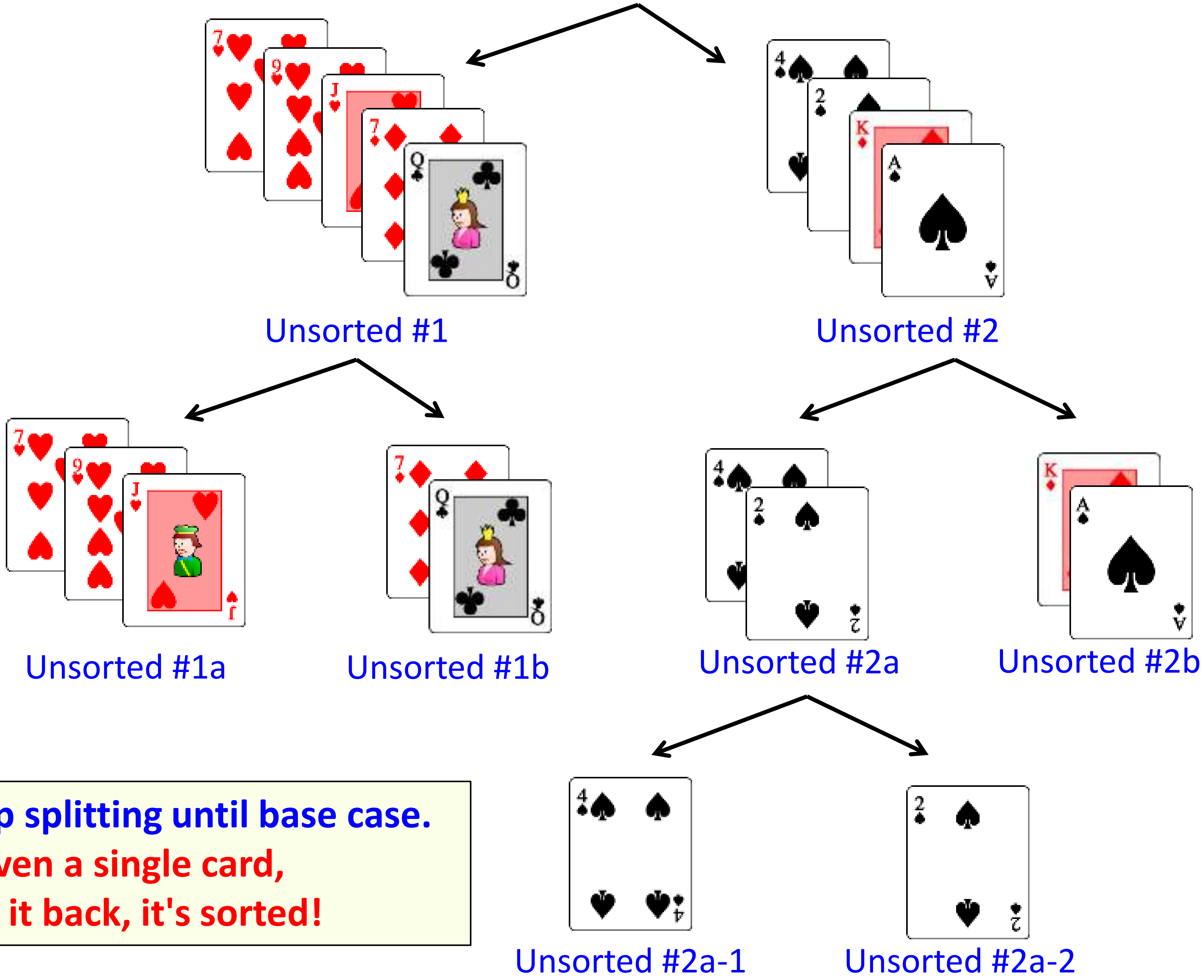
Unsorted #1b

Unsorted #2a

Unsorted #2b

**Divide and conquer!**

**If somebody give you a pile of unsorted cards, divide in half and give them to two other people**



Unsorted #1

Unsorted #2

Unsorted #1a

Unsorted #1b

Unsorted #2a

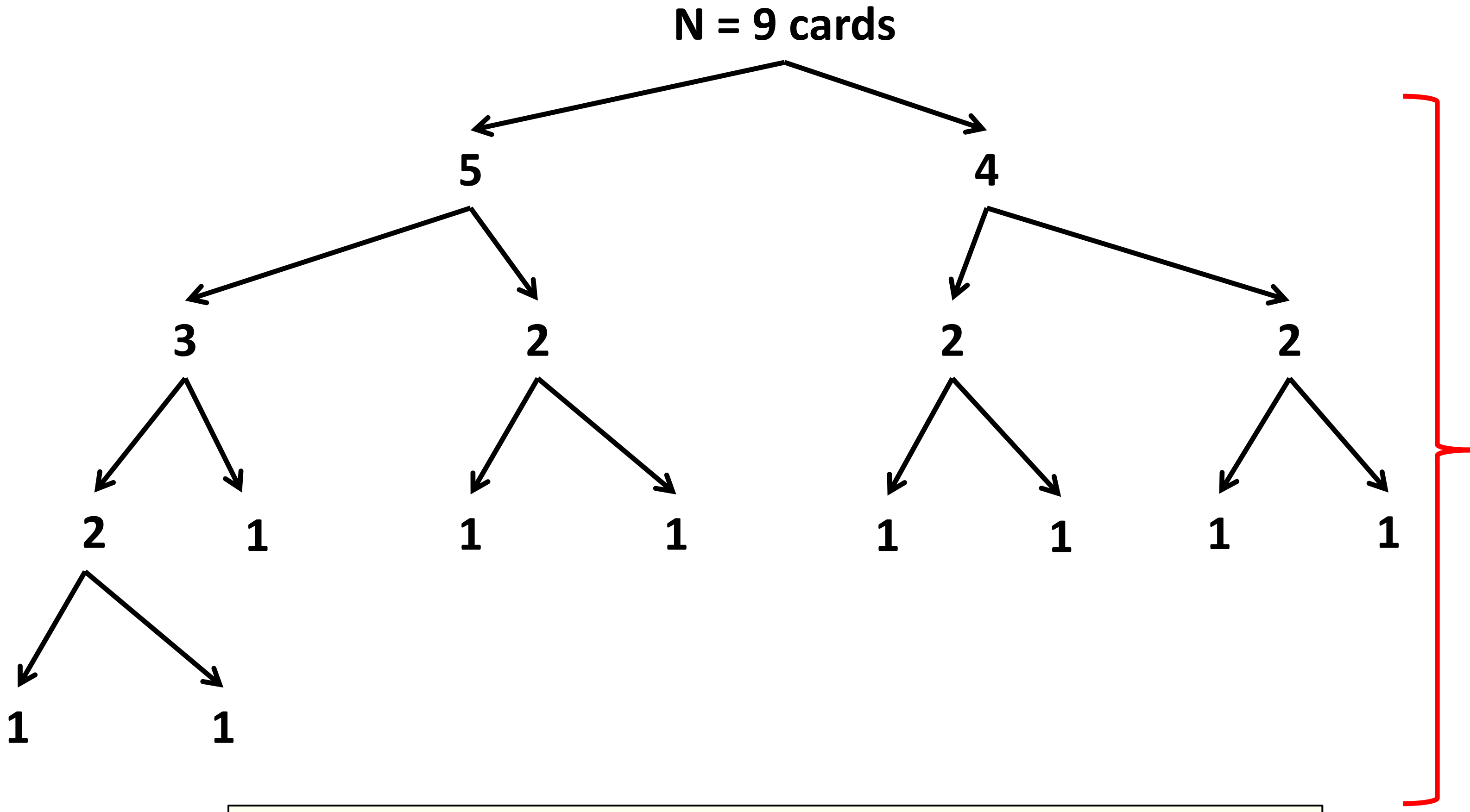
Unsorted #2b

Unsorted #2a-1

Unsorted #2a-2

**Keep splitting until base case.**
**If given a single card,**
**give it back, it's sorted!**

# How many levels?



4 levels requiring merging of two halves

$\log_2(9) \approx 3.17$
$2^{\wedge 3.17} \approx 9$

Order $\log_2(N)$ levels, each level takes order N ops to merge
Mergesort: O(N log N)

# Programming Activity

- Complete the MergeSort program
  - Download MergeSort.java from website
  - Step 1: Finish the sort method
    - Assume you have a working merge method
  - Step 2: Finish the merge method
    - Given two sorted arrays, merge into a single sorted array

# Mergesort, sorting method

```
public static void sort(int [] nums)
{
   if (nums.length == 1)
      return;

   int mid = ?????;
   int [] d1 = new int[?????];
   int [] d2 = new int[?????];

   for (int i = 0; i < ?????; i++)
   {
      if (?????)
         d1[?????] = nums[i];
      else
         d2[?????] = nums[i];
   }

   ?????
   ?????;

   ?????;
}
```

Base case, incoming array is already sorted

Splitting incoming array into two (mostly) equal halves.

Recur, asking each half to be sorted.

Merge the now sorted half arrays.

# Programming Activity

- Complete the MergeSort program
  - Finish the body of merge method
  - Assume:

    `part1` and `part2` are sorted

    `part1.length + part2.length = result.length`

```
public static void merge(int [] part1, int [] part2, int [] result)
{
    ...
}
```

| | | |
|---|---|---|
| 1 | 3 | 5 |

| | | | |
|---|---|---|---|
| 2 | 3 | 8 | 9 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |

# Mergesort, merge method

```
public static void merge(int [] part1, int [] part2, int [] result)
{
   int index1 = 0;
   int index2 = 0;
   for (int k = 0; k < ?????; k++)
   {
      if (index1 == ?????)
      {
         result[k] = ?????
         index2++;
      }
      else if (index2 == ?????)
      {
         result[k] = ?????;
         index1++;
      }
      else if (?????)
      {
          result[k] = ?????;
          index1++;
      }
      else
      {
          result[k] = ?????;
          index2++;
      }
   }
}
```

Loop over the total elements to merge

Handle case when we've run out of data in array part1

Handle case when we've run out of data in array part2

Part 1 currently has the smallest one

Part 2 currently has the smallest one

# Summary

- More recursion
  - Randomness and recursion = pretty pictures
- Divide-and-conquer
  - If you don't know how solve the whole problem:
    - Split it into parts
    - Ask somebody to solve the parts
    - Merge the parts together into a solution
  - Mergesort
    - Optimal sorting, O(N log N)
    - Simple recursive divide-and-conquer algorithm