Recursive Sorting with Mergesort

University of Massachusetts Amherst
ECE 242 – Data Structures and Algorithms
Lecture 16

Divide and conquer

- Thinking "recursively" about a problem
  - Describe problem based on smaller sub-problems
  - Figure out when to stop dividing the problem
- Programming of recursive function
  - Check if termination condition has been reached
    - Return value of function is likely easy to determine
  - Call function itself to solve smaller problems
  - Combine smaller problems and return value

More recursion examples

- Multiplication using addition:
  - m*n = n+(m-1)*n and 0*n = 0
- Binomial coefficient:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k : 1 \leq k \leq n-1,$$

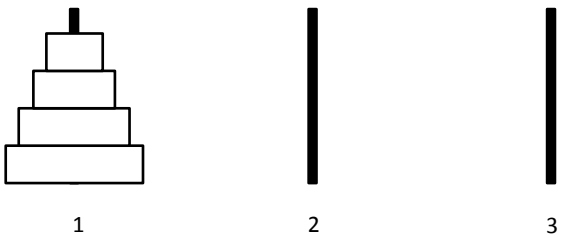
and

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \text{for all integers } n \geq 0,$$

- Maze solving
  - Find path in remaining unexplored map

Tower of Hanoi example

- Tower of Hanoi
  - How to move four pieces from stack 1 to stack 3?



Tower of Hanoi example

- Code

```
// pieces are numbers top to bottom (smallest=1)
// stack positions are 1, 2, and 3
static private void move(int top, int bottom, int from, int to) {
    if (top==bottom) { // only need to move single piece
        System.out.println("move "+top+" from "+from+" to "+to);
    } else { // break into multiple smaller subproblems
        int other=6-(from+to);
        move(top,bottom-1,from,other);
        move(bottom,bottom,from,to);
        move(top,bottom-1,other,to);
    }
}
```


Recursive sorting

- Apply divide and conquer to sorting problem
  - How can we solve sorting recursively?

Mergesort

- Idea
  - Divide array into two halves and sort each half recursively ("divide")
  - Merge sorted halves into one sorted array ("conquer")
- Merging of two sorted arrays
  - How to merge in a single pass?

Mergesort in practice

- Merging is convenient for "linear" processing
  - Examples: data tapes, memory lines in cache, …

Mergesort

- Code: “divide”

```
int middle = values.length/2;
int[] left = new int[middle];
for (int i=0; i<middle; i++) {
   left[i] = values[i];
}
int[] right = new int[values.length-middle];
for (int i=0; i<values.length-middle; i++) {
   right[i] = values[middle+i];
}
sort(left);
sort(right);
```


Mergesort

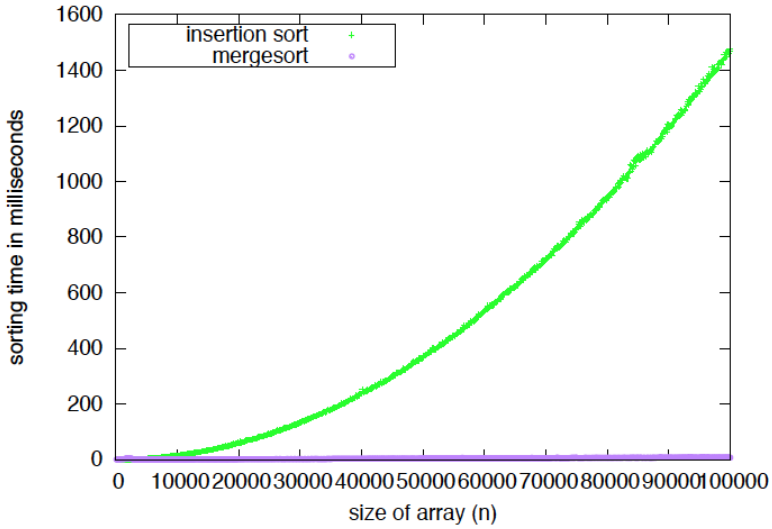
- Code: “conquer”

```
int l=0, r=0;
for (int i=0; i<values.length; i++) {
   if (r>=right.length ||
      (l<left.length && left[l]<right[r])) {
      values[i]=left[l];
      l++;
   } else {
      values[i]=right[r];
      r++;
   }
}
```

Complexity analysis

- Dividing and merging of array of length n
  - Cost: O(n) time (two passes through array)
- Division
  - One division: O(n) + 2*O(n/2)
  - Second division: O(n) + 2*O(n/2) + 4*O(n/4)
  - Etc.
- Division stops when array is size 1
  - After O(log n) steps
- Total complexity: O(n log n)

Mergesort performance



Mergesort performance

- Difficult to compare due to cost of different operations
  - Artificial 1ms penalty per comparison

	n=100	n=200	factor
insertion sort	2655	11530	4.34
mergesort	794	1815	2.29

- Mergesort growth:
  - Less than quadratic, more than linear

Next Steps

- Class next Tuesday (Monday schedule)
- Project 2 do 10/16, 11PM