M, 09/02/19

Fall'19 CSCE 629

Analysis of Algorithms

Fang Song
Texas A&M U

Lecture 2

- Recursion
- Merge Sort

Review: sort by asymptotic order of growth

1. $n \log n$
2. $\sqrt{n}$
3. $\log n$
4. n^2
5. 2^n
6. n
7. $n!$
8. $n^{1,000,000}$
9. $n^{1/\log n}$
10. $\log n!$

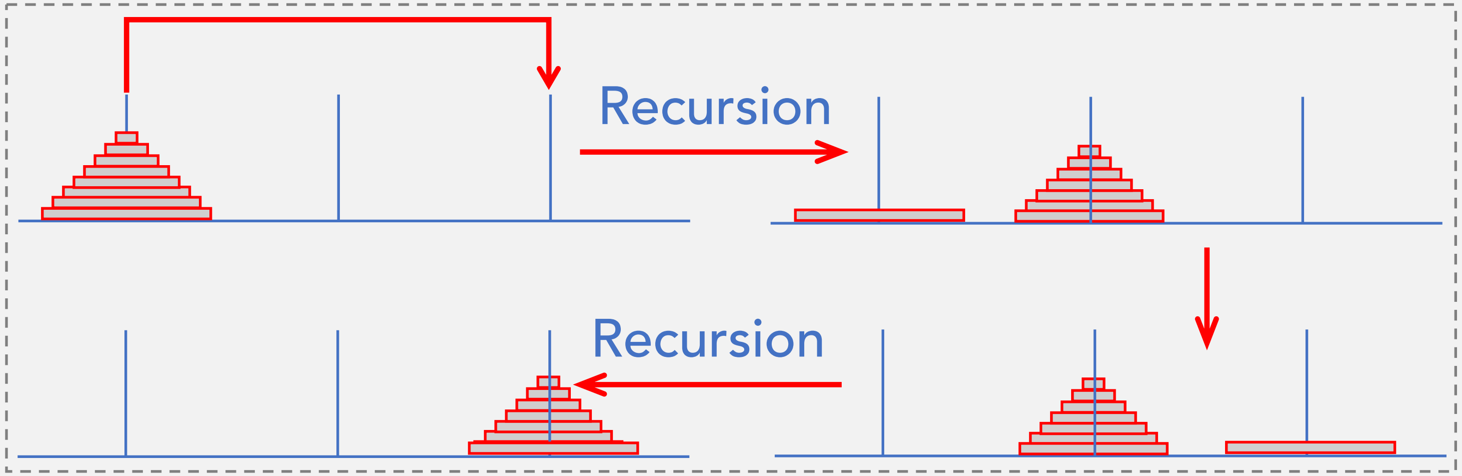
$$9, 3, 2, 6, (1 = 10), 4, 8, 5, 7$$

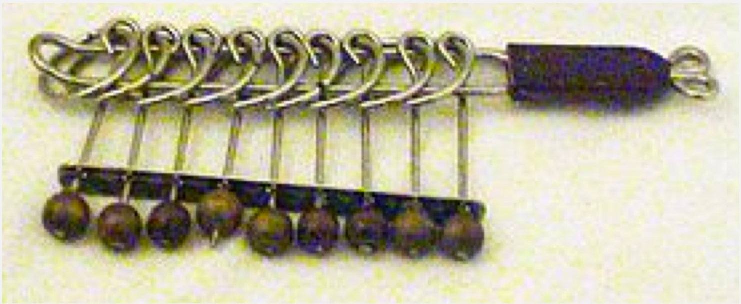
Recursion: "self" reduction

- Simplify and delegate
  - If the given instance of the problem can be solved directly, solve it directly
  - Otherwise, reduce it to one or more simpler instances of the same problem.

- Induction (in disguise)
  - Base case
  - Induction hypothesis

… and think no further, imagining a Recursion Fairly solves the subproblems for you





https://youtu.be/cHxJFMsvAII

Recap: principal questions to ask

0 What problem to solve?

be as precise as possible

1 Is the algorithm correct?

2 How much resource it costs?

(time, space, …)

3 Can we do better?

DESIGN

ANALYSIS

Sorting

- Given: n elements (numbers, letters, etc.) $a[1, \ldots, n]$
- Goal: rearrange in ascending order

- Applications
  - Display google page rank results
  - Find the median
  - Binary search in a database
  - Data compression
  - Computational biology
  - …

Obvious apps

Easy once sorted

More clever apps

Exercise. Name your familiar sorting algorithms

Merge sort

- Main Idea
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.

Jon von Neumann, 1945

Merging

- Given: two sorted (sub)-lists
- Goal: combine into a sorted whole

- How to merge efficiently?
  - Use temporary array
  - Store smaller of L/R, and continue to the next in that list

Write up your algorithm

Problem description

- Input: a list of n (letters) $a[1, \ldots, n]$
- Output: $a[1, \ldots, n]$ sorted,
  i.e., $a[i] \leq a[j], \forall i < j \in [n]$

```
MergeSort(a[1,...,n]):
  if n > 1
    m ← ⌊n/2⌋
    MergeSort(a[1,...,m]) // recursion
    MergeSort(a[m+1,...,n]) // recursion
    Merge(a[1,...,n], m)
```

```
Merge(a[1,...,n], m):
  i ← 1; j ← m + 1
  for k ← 1 to n
    if j > n
      b[k] ← a[i]; i ← i + 1
    else if i > m
      b[k] ← a[j]; j ← j + 1
    else if a[i] < a[j]
      b[k] ← a[i]; i ← i + 1
    else
      b[k] ← a[j]; j ← j + 1
  for k ← 1 to n
    a[k] ← b[k]
```

Recap: principal questions to ask

0 What problem to solve?

be as precise as possible

1 Is the algorithm correct?

2 How much resource it costs?

(time, space, …)

3 Can we do better?

DESIGN

ANALYSIS

Correctness of MergeSort

- Think of Reduction again

MergeSort$(a[1,\dots,n])$:
  if $n > 1$
    $m \leftarrow \lfloor n/2 \rfloor$
    MergeSort$(a[1,\dots,m])$ // *recursion*
    MergeSort$(a[m+1,\dots,n])$ // *recursion*
    Merge$(a[1,\dots,n], m)$

Merge$(a[1,\dots,n], m)$:
  $i \leftarrow 1; j \leftarrow m+1$
  for $k \leftarrow 1$ to n
    if $j > n$
      $b[k] \leftarrow a[i]; i \leftarrow i+1$
    ……

1. Correctness of **MergeSort**$(a[1,\dots,n])$ (assuming **Merge**() is correct)
   - Induction on n.
2. Correctness of **Merge**$(a[1,\dots,n], m)$
   - Loop Invariant: $b[k]$ is the smallest of $a[i,\dots,m]$ and $a[j,\dots,n]$. Read CLRS

Resource analysis of MergeSort

- Running Time $T(n)$ $\quad T(n) = 2T(n/2) + O(n)$ $\quad = O(n \log n)$

Will show. You can already verify by induction!

$O(1)$

$2T(n/2)$

$O(n)$ Either pointer grows by 1 in each iteration

- Space (memory) $S(n)$ $\quad S(n) = O(n)$ $\quad a[\cdot], b[\cdot]$

Exercise. Can you merge in place, without temporary array?

Quicksort (if time permits)

- Main Idea
  - Divide array into two halves. — with condition: $L \leq pivot \leq R$
  - Recursively sort each half.
  - Merge two halves to make sorted whole. — trivially
- Demo (on board)

Tony Hoare, 1959

- Analysis
  - Correctness
  - Running time* $T(n) = 2T(n/2) + O(n)$ — Cost in divide, not merge

* best-case partition

- A lot more to say about quicksort, we'll come back to it

Name your familiar sorting algorithms

Algorithms	Idea	$T(n)$
Insertion		$O(n^2)$
Bubble		$O(n^2)$
Merge		$O(n\log n)$
Quick*		$O(n\log n)$
Non-comparison algorithms	Counting, radix, bucket, …	

* randomized

Exercise. Can you improve on $O(n\log n)$ for comparison-based sorting?

A template for a complete algorithm

Problem description

- Input: $a[1, \dots, n]$ …
- Output: …

Algorithm description

- Idea: divide, sort, merge …
- Pseudocode:

MergeSort($a[1, \dots, n]$):
if $n > 1$ ……

Analysis of algorithm

- Correctness: …
- Running Time: …
- Other analysis when needed: space …