CSE373: Data Structures and Algorithms

More Asymptotic Analysis (Examples)

Steve Tanimoto
Autumn 2016

This lecture material represents the work of multiple instructors at the University of Washington. Thank you to all who have contributed!

Give Asymptotic Analyses for the Following

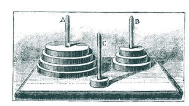
1. $g(n) = 45\, n \log n + 2n^2 + 65$
2. $g(n) = 1000000\, n + 0.01 \cdot 2^n$
3.
```
int sum = 0;
    for (int i = 0; i < n; i=i+2){
        sum = sum + i;
    }
```
4.
```
int sum = 0;
    for (int i = n; i > 1; i=i/2){
        sum = sum + i;
    }
```


Next Compare Two Recursive Algorithms

- Towers of Hanoi Puzzles (including the Towers of Brahma puzzle where n=64).
- Mergesort (for sorting an array of n numbers or other comparable keys such as strings)

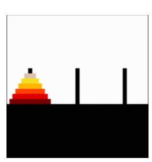
The Time to Solve the Towers of Brahma Puzzle

- The Towers of Brahma problem is a 64-disk Towers of Hanoi puzzle.
- All disks start on the Left peg.
- Goal: move all disks to the Right peg.
- Constraints:
  - move 1 disk at a time;
  - only the topmost disk can be moved from a pile.
  - a disk may never be placed on top of one smaller than it.
- Time: 1 second per move (according to legend).

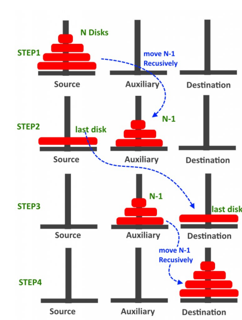


The n-Disk Towers of Hanoi Puzzle

A good solution approach:
- If n=1, move the (only) disk from the start peg to the goal peg.
- Otherwise,
  - first move the top n-1 disks to the non-goal (and non-start) peg (recursively);
  - then move the bottom peg to the goal peg;
  - finally, move the n-1 disks from the non-goal peg to the goal peg (recursively).



La Tour d'Hanoi was originally invented by French mathematician Eduardo Lucas in 1883.
http://www.puzzlemuseum.com/month/picm07/2007-03_hanoi.htm



http://algorithms.tutorialhorizon.com/towers-of-hanoi/

Time for Solving Towers of Hanoi

- Let the time to move one disk be 1 unit (e.g., one second).
- T(n) represents the (minimum) time (= number of moves) required to solve an n-disk Towers of Hanoi puzzle.
- If there is only 1 disk, 1 unit of time is required:
  T(1) = 1.
- If there are n>1 disks, the time required is:
  T(n) = T(n-1) + T(1) + T(n-1)
  = 2 T(n-1) + 1
  = 2 (2 T(n - 2) +1) + 1 (if n > 2)
  = 4 T(n - 2) + 3
  = 8 T(n - 3) + 7 (if n > 3)
  ...
  = 2^{n-1} T(1) + (2^{n-1} - 1)
  = $2^{n-1} + 2^{n-1} - 1$ = **$2^n - 1$** $\in$ **$\Theta(2^n)$**

The Tower of Brahma Puzzle Takes

2^{64} - 1 seconds
= 18,446,744,073,709,551,615 seconds
≈ 585 billion years
≈ about 127 times the current age of the sun.

After the monks have finished moving the disks, then the world will end, according to the Brahmin legend.

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

Example: Analyzing Mergesort

Mergesort is a recursive algorithm for sorting an array of number of other comparable keys such as strings.

It uses an algorithm paradigm known as "divide and conquer" in which the problem is conceptually split up into parts, and each part is solved separately, and then the results from the parts are combined into an overall solution.

Merge sort

8	2	9	4	5	3	1	6

- To sort array from position `lo` to position `hi`:
  - If range is 1 element long, it is already sorted! (Base case)
  - Else:
    - Sort from `lo` to `(hi+lo)/2`
    - Sort from `(hi+lo)/2` to `hi`
    - Merge the two halves together
- Merging takes two sorted parts and sorts everything
  - *O*(*n*) but requires auxiliary space…

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 "fingers"
and 1 more array

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 "fingers"
and 1 more array

1							

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2						

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2	3					

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2	3	4				

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2	3	4	5			

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2	3	4	5	6		

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 “fingers”
and 1 more array

1	2	3	4	5	6	8	

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

Merge:
Use 3 "fingers"
and 1 more array

1	2	3	4	5	6	8	9

(After merge,
copy back to
original array)

Example, focus on merging

Start with:

8	2	9	4	5	3	1	6

After recursion:
(not magic ☺)

2	4	8	9	1	3	5	6

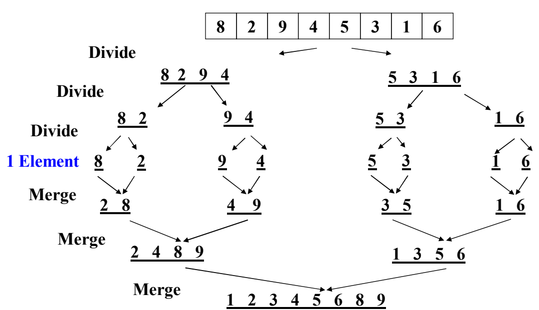
Merge:
Use 3 "fingers"
and 1 more array

1	2	3	4	5	6	8	9

(After merge,
copy back to
original array)

1	2	3	4	5	6	8	9

Example, Showing Recursion



Mergesort Analysis

(One of the recurrence classics)

For simplicity let constants be 1 (no effect on asymptotic answer)

$T(1) = 1$
$T(n) = 2T(n/2) + n$
$= 2(2T(n/4) + n/2) + n$
$= 4T(n/4) + 2n$
$= 4(2T(n/8) + n/4) + 2n$
$= 8T(n/8) + 3n$
….
$= 2^k T(n/2^k) + kn$

So total is $2^k T(n/2^k) + kn$ where
$n/2^k = 1$, i.e., $\log n = k$
That is, $2^{\log n}\, T(1) + n \log n$
$= n + n \log n$
$\in \Theta(n \log n)$

Summary

The Towers of Hanoi recurrence leads to $\Theta(2^n)$ time behavior.

The Mergesort recurrence leads to $\Theta(n \log n)$ time behavior.

Although both algorithms use the divide-and-conquer approach, and two-way recursion, when we solve the recurrences, we find one to be much, much faster than the other.