

# Algorithmic Techniques

9/27/07

- Brute-force algorithms - Take the most direct or obvious solution. Often a basis for comparison.
- Divide and conquer algorithms - Break problem down into subproblems that are similar to the original problem, but smaller in size. Typically leads to recursive algorithms.
- Greedy Algorithms - Applied to optimization problems, at each step choose the locally optimal choice if made.
- Dynamic Programming - Break down problem into subproblems, and use the solution to these subproblems to solve larger subproblems (reusing results). Applied to optimization problems.
- Randomized algorithms - Behavior of the algorithm is controlled by a random number generator.

## Ex: Sorting (comparison-based sorting)

Given: A sequence of  $n$  data items  $a_1, a_2, \dots, a_n$  where each item  $a_i$  has an associated key ( $key[a_i]$ )

Output: Data in a non-decreasing sequence

Assumptions: A total ordering relationship exists

eg.  $key[a_e], key[a_m], key[a_n]$

1. Exactly one is true:

$$key[a_e] < key[a_m]$$

$$key[a_e] = key[a_m]$$

$$key[a_e] > key[a_m]$$

and

2. Transitivity:

$$key[a_e] < key[a_m] \text{ and } key[a_m] < key[a_n]$$

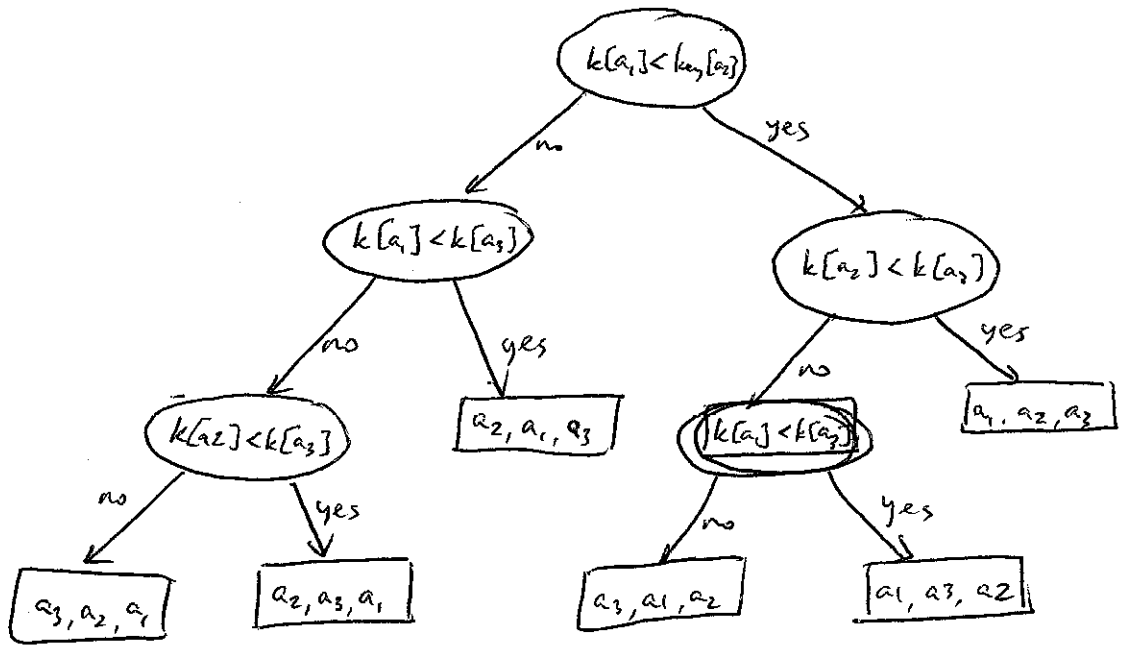
$$\text{then } key[a_e] < key[a_n]$$

$\Theta(1)$  operations are

- Comparison: binary comparison made between two key values.
- Swap: interchange two items in the array

Theoretical lower bound (comparison-based sorting)

Decision tree to sort  $a_1, a_2, a_3$ :



if there are  $n$  items to sort:

- there are  $n!$  permutations of these items.
- A decision tree would have at least  $n!$  leaves

A perfect binary tree of height  $h$  has  $2^h$  leaves:

- Any binary tree will have at most  $2^h$  leaves
- So tree must have a depth of at least  $\lg(n!)$  to have  $n!$  leaves

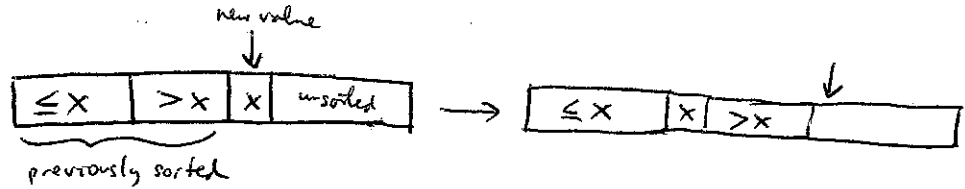
$O(\lg(n!)) = O(n \log(n))$  by Stirling's formula!

$n! \approx \sqrt{2\pi n} \frac{n^n}{e^n}$

# Ex. Insertion Sort

## Insertion Sort (A[n])

- 1 for  $i \leftarrow 1$  to  $n-1$  do
- 2 for  $j \leftarrow i$  to 1 in steps  $n-1$  do
- 3 if  $key[A[j]] < key[A[j-1]]$  then
- 4 swap  $A[j]$  at  $A[j-1]$



initial:	9	6	2	4	7	
after pass 1	6	9		2	4	7
after pass 2	2	6	9		4	7
after pass 3	2	4	6	9		7
after pass 4	2	4	6	7	9	

Sub-array is always sorted

Total # of comparisons:

$$\sum_{i=1}^{n-1} \sum_{j=1}^i 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Total # of swaps:

in worst case (always swap): reverse sorted order  $\frac{n(n-1)}{2}$

$$\text{Total ops: } \frac{n(n-1)}{2} + \frac{n(n-1)}{2} = n^2 - n$$

Insertion sort is  $\Theta(n^2)$

Ex:

# Merge sort

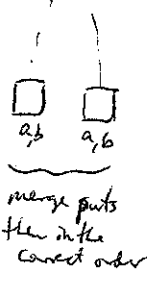
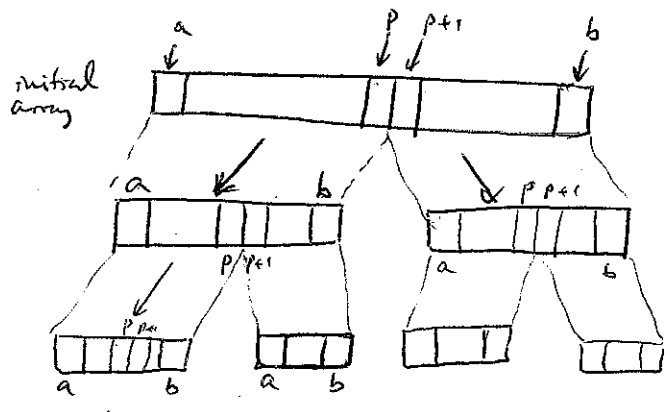
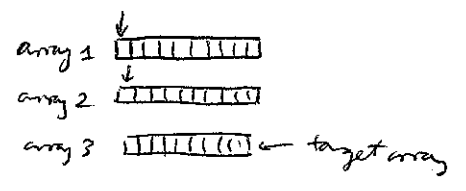
Divide and Conquer Algorithm  
initially

Mergesort (A[], integer  $\overset{0}{\downarrow} a$ , integer  $\overset{n-1}{\downarrow} b$ )

Recursive algorithm

- 1 if  $a < b$  then
- 2  $p \leftarrow \lfloor (a+b)/2 \rfloor$
- 3 Mergesort (A, a, p)
- 4 Mergesort (A, p+1, b)
- 5 Merge (A, a, p, b)

Merge() takes 2 sorted arrays and combines them to produce a sorted output in  $\Theta(n)$  time.



Recurrence relation:

$$T(n) = \begin{cases} \Theta(1) & , \text{ if } n=1 \text{ (base case)} \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & , \text{ if } n>1 \end{cases}$$

floor and ceiling do not change the asymptotic running time.

Recurrence relation is defined in terms of itself

$$T(n) = \begin{cases} \Theta(1) & , \text{ if } n=1 \\ 2T(\frac{n}{2}) + \Theta(n) & , \text{ if } n>1 \end{cases}$$

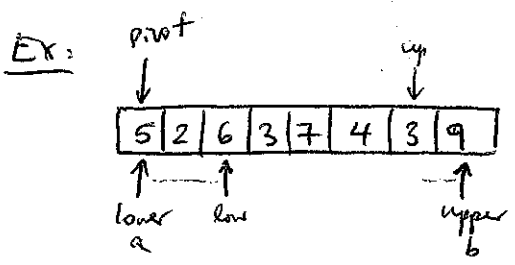
$$T(n) = \Theta(n \log n)$$

# Ex Quick sort (also divide and conquer)

Quicksort (A[], int a, int b)

1. if  $a < b$
2.  $k \leftarrow \text{Partition}(A, a, b)$
3. Quicksort (A, a, k-1)
4. Quicksort (A, k+1, b)

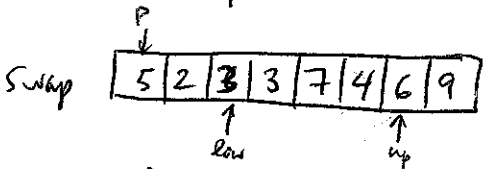
Partition picks a "pivot"  $k$ . It then orders all of the numbers less than the pivot to the left of the pivot, greater keys to the right, equal either way.



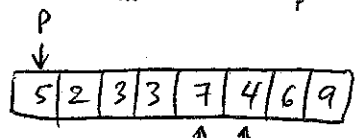
goal of partition:  
 get all the elements  $< 5$   
 (2, 3, 3, 4) to the left of 5  
 and all the elements  $> 5$   
 (6, 7, 9) to the right of 5

- Lower ptr moves until it finds an element larger than pivot,
- Upper ptr moves until it hits an element smaller than the pivot

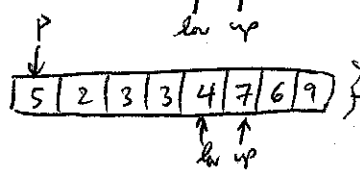
Now swap these



move

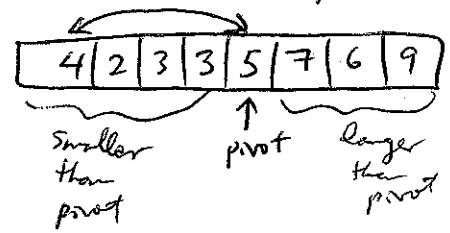


Swap



we are done!  
 everything 0 to low is  $< 5$   
 everything up to n-1 is  $> 5$

now swap low at pivot



Pivot is now in sorted order

Call Quicksort on smaller sub arrays

# Quicksort Analysis

Worst case: The pivot ends up always as the first or last element,

$$T(n) = \begin{cases} \Theta(1) & , n=1 \\ T(n-1) + \Theta(n) & , \text{if } n > 1 \end{cases}$$

Unfolding the recurrence relation:

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= T(n-2) + \Theta(n-1) + \Theta(n) \\ &= \sum_{i=1}^n \Theta(i) = \Theta\left(\sum_{i=1}^n i\right) \\ &= \Theta(n^2) \quad \leftarrow \text{not very good!} \end{aligned}$$

Best case: Each partitioning exactly divides the array in half (within 1)

$$T(n) = \begin{cases} \Theta(1) & , n=1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & , n > 1 \end{cases}$$

Same as merge sort

$$T(n) = \Theta(n \log n)$$

Average case: The probability of any element chosen as the pivot is  $\frac{1}{n}$  (uniform distribution)

$$\begin{aligned} T_{\text{av}}(n) &= \underbrace{cn}_{\text{average merge cost}} + \underbrace{\left(\frac{1}{n}\right)}_{\text{uniform distribution}} \sum_{p=1}^n \underbrace{[T_{\text{av}}(p-1) + T_{\text{av}}(n-p)]}_{\substack{\text{goes from } 0 \text{ to } n-1 \\ \text{goes from } n-1 \text{ to } 0}} \\ &= cn + \frac{2}{n} \sum_{p=1}^n T_{\text{av}}(p-1) \quad \leftarrow \text{average of all possible partitions} \\ &\quad \leftarrow \text{same! rewrite} \end{aligned}$$

This is a full-history recurrence

$$T_{\text{av}} = \Theta(n \log n)$$