ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives. Inventing the Future. www.iit.edu

CS430

Spring 2023

Introduction to Algorithms

Lec 2

Instructor: Dr. Lan Yao

Agenda

- Insertion Sort
- Merge Sort
- Runtime Analysis

Insertion Sort Pseudo-code

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    key ← A[j]
    //Insert A[j] into sorted sequence A[1 ..j - 1]
    i ← j - 1
    while i > 0 and A[i] > key
        A[i + 1] ← A[i]
        i ← i - 1
    A[i + 1] ← key
```

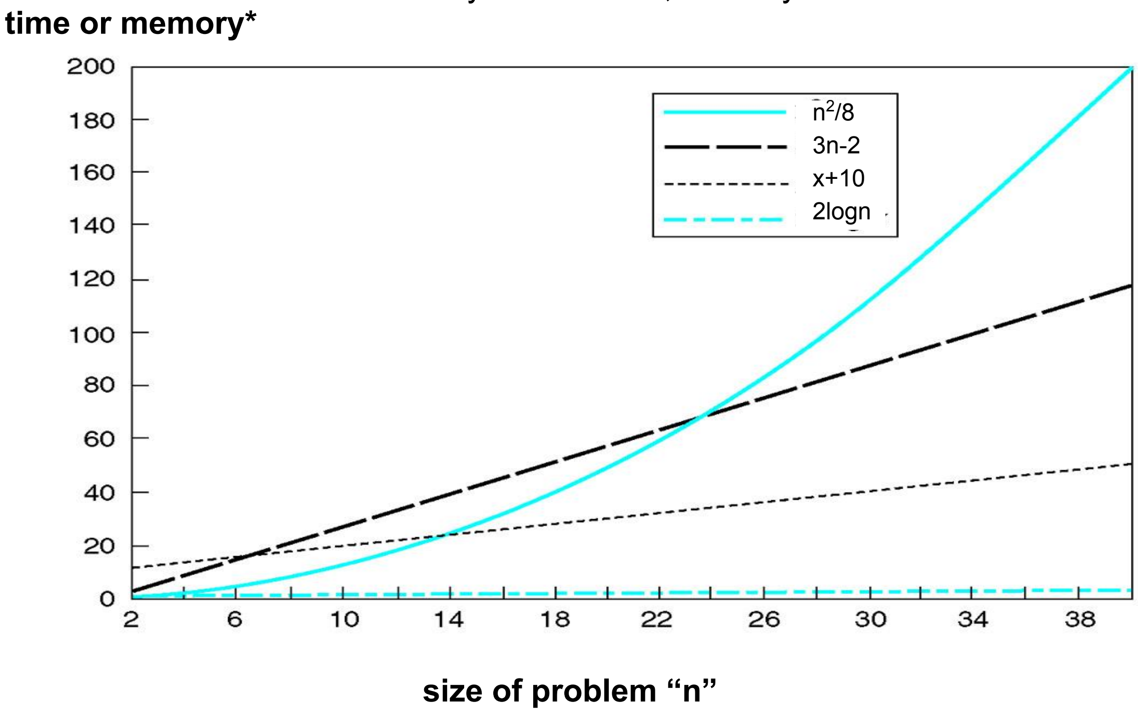
Analysis of Algorithms – summarize behavior*

Run-Time Analysis for Sorting –
Depends on input size & input itself

Kinds of Analysis

- worst case (usually used)
- average case (sometimes used)
- best case (hardly ever used)

Analysis – runtime, memory



Individual points on the graph are irrelevant, only the growth of the function matters

Run-time Analysis*

Functional Analysis

- ignore machine dependent constants
- look at the growth of T(n) as n-> infinity
- as you double n, what does T(n) do?? double?? square??

Runtime Analysis Approaches*

For iterative algorithms

- count the number of times each statement is executed
- define constants for the execution time of various types of statements
- develop a function describing the runtime as a function of the problem size.

For recursive algorithms, develop and solve a recurrence relation--later

Insertion Sort

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

let t_j be the number of times the **while** loop test in line 5 is executed for that value of j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) +$$
$$c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Best Case: the array has been sorted, then

$t_j = 1$ for all j

$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$

$= (c_1+c_2+c_4+c_5+c_8)n - (c_2+c_4+c_5+c_8)$

$= an+b$ -- Linear Growth Function

- Worst Case t_j = j for all *j*

while j lies in 2 to n

$T(n) = c_1n+c_2(n-1)+c_4(n-1)$

$+c_5(n-1)(n+2)/2+c_6n(n-1)/2$

$+c_7n(n-1)/2+ c_8(n-1)$

$=an^2+bn+c$---Quadratic Polynomial

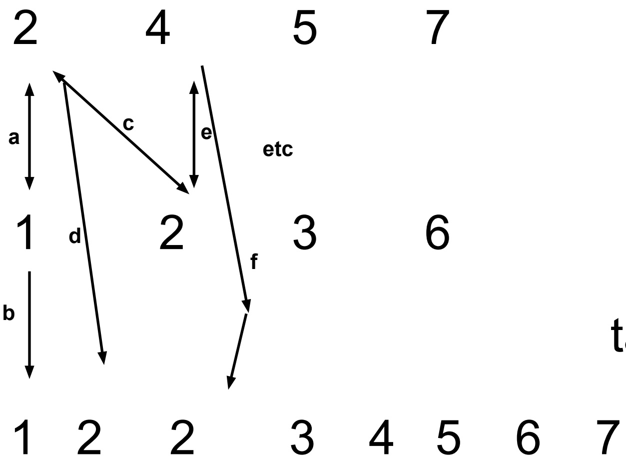
- Average Case t_j = j/2 for all *j* roughest.
  $T(n) = an^2+bn+c$ Quadratic Polynomial

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

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

Merge Sort

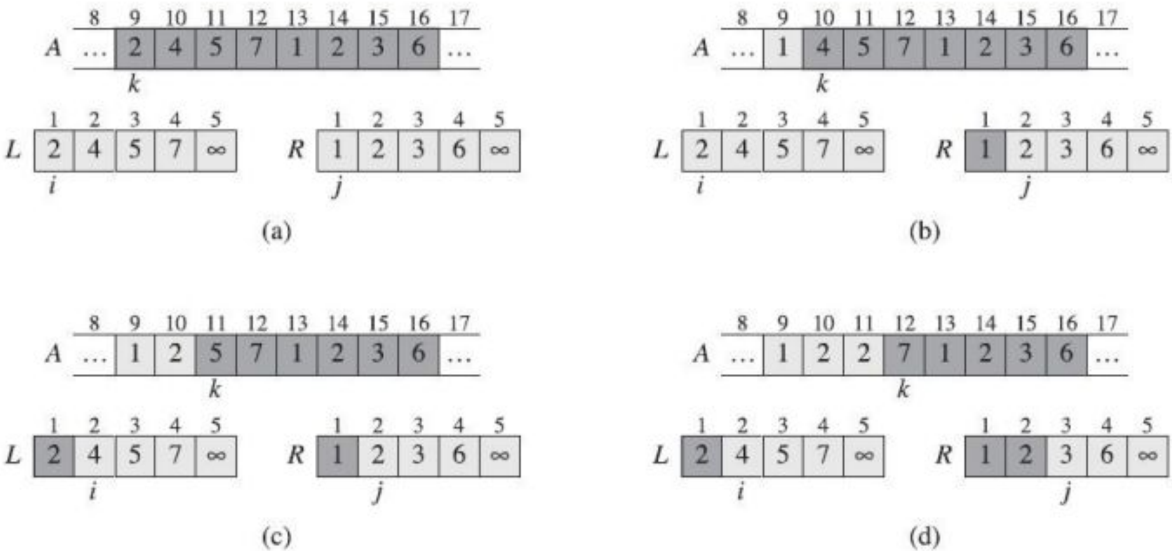
You can merge 2 sorted lists in a linear time



compare

take smaller

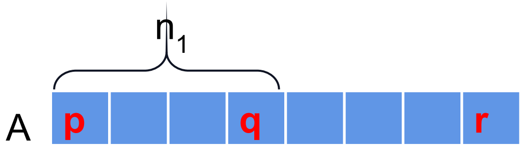
Merge Sort



8 9 10 11 12 13 14 15 16 17
A … 2 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(a)
8 9 10 11 12 13 14 15 16 17
A … 1 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(b)
8 9 10 11 12 13 14 15 16 17
A … 1 2 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(c)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(d)

MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
        L[i] = A[p + i - 1]
for j = 1 to n2
        R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else A[k] = R[j]
                j = j + 1
```

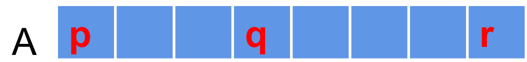


MergeSort

5	2	4	7	1	3	2	6

initial sequence

Recursive Solution to Merge Sort




```
Mergesort(A, p, r)     // initial call Mergesort (A, 1, n)
{
    if (p<r)
    {
        q = ⌊(p+r)/2⌋          // integer division
        Mergesort(A, p, q)      // recursively sort 1st half
        Mergesort(A, q+1, r)   // recursively sort 2nd half
        Merge(A, p, q, r)        // merge 2 sorted sub-lists
    }
}
```

Merge Sort Runtime Analysis

- divide and conquer (and combine) approach, recursive algorithm
- basic step, you can merge two sorted lists of total length n in a linear time
- second key idea, a list of length one element is sorted

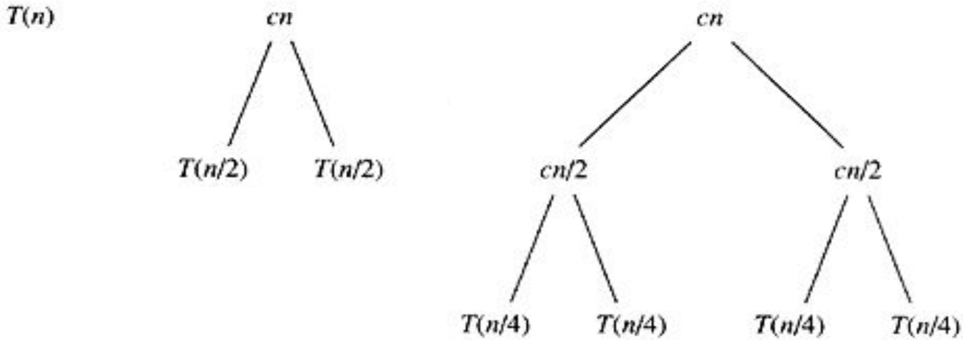
$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + c(n)$$

$$\text{Re}\,currence\ \text{Re}\,lation$$

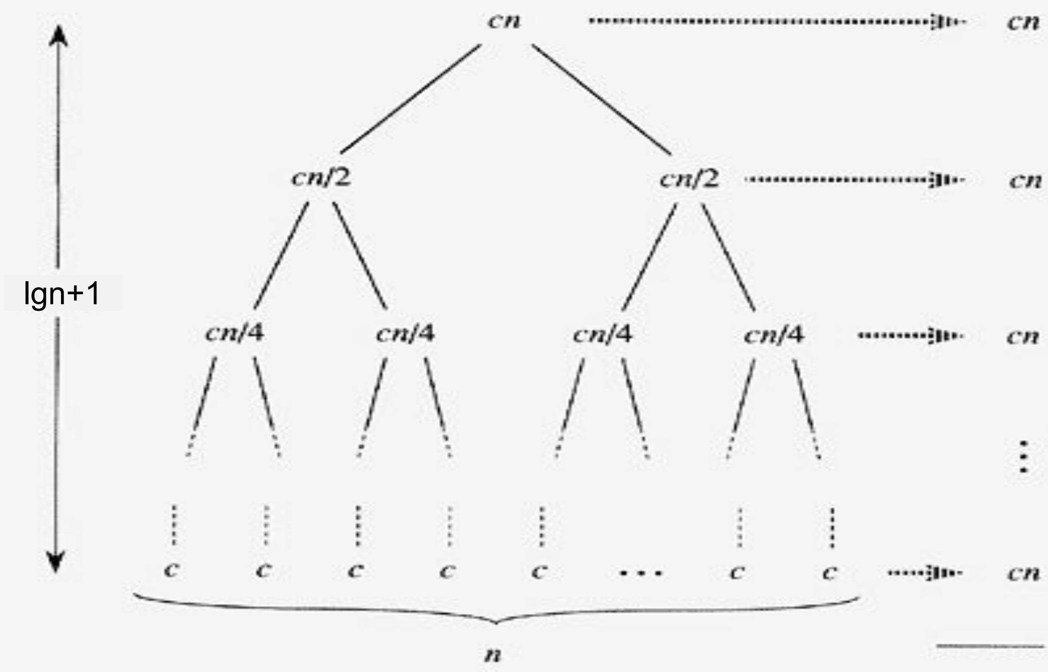
$$T(n) = 2T\left(\frac{n}{2}\right) + c(n)$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c(\frac{n}{2})$$

Divide: Split a problem into a sequence of subproblems that are equivalent to the original.
Conquer: solve subproblems recursively to make original problem solved.
General form for a recursion algorithm：
T(n)=aT(n/b)+D(n)+C(n)



(a) (b) (c)


lgn+1
cn
cn/2
cn/2
cn/4
cn/4
cn/4
cn/4
c c c c c ... c c
n
cn
cn
cn
cn
Total: cn lg n + cn

(d)

T(n)= the number of levels*cn

Find the number of levels:

$$n\left(\frac{1}{2}\right)\left(\frac{1}{2}\right)\left(\frac{1}{2}\right)\cdots\left(\frac{1}{2}\right)=1$$

$$1 \bullet 2 \bullet 2 \ldots \bullet 2 = n$$

$$2^{?} = n$$

$$? = \log_2 n = \lg n$$

$$The \quad number \quad of \quad levels = \quad \lg n + 1$$

$$T(n) = cn(\lg n + 1) = cn \lg n + cn$$

Insertion and Merge Sort

Which one is better?
$2n^2$ or $10n\lg n+100n$

How to evaluate their performance in terms of runtime?

Asymptotic Bounds

Asymptotic Notation

We have to investigate how the running time of an algorithm increases in the limit with the size of input going infinite.

For a given function g(n), we denote by $\Theta(n)$ the set of functions:

$\Theta(g(n))=$ { f(n), there exits positive constants c_1,c_2 and n_0 such that $0<=c_1g(n)<=f(n)<=c_2g(n)$ for all $n> n_0$ }

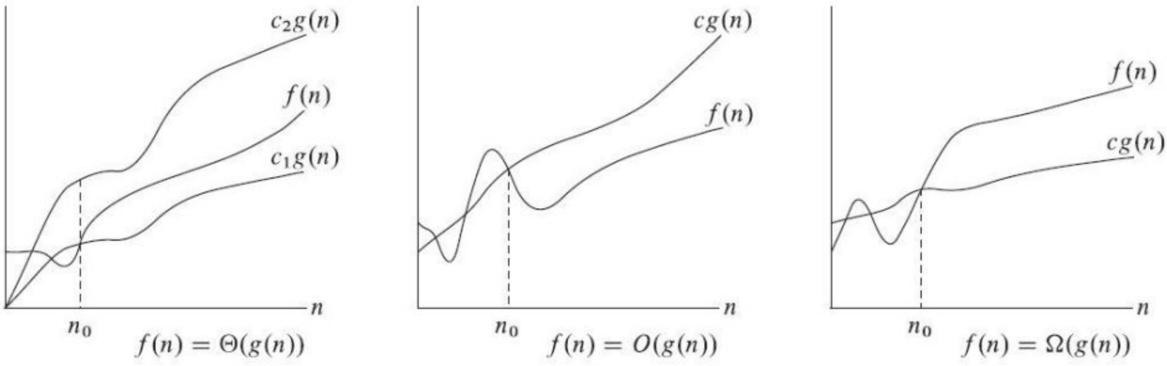
If the above definition stands, $f(n) \in \Theta(g(n))$.

We use $f(n)=\Theta(g(n))$ instead of $f(n) \in \Theta(g(n))$ for simplification.

- Asymptotic bound:

For all $n>n_0$, the function f(n) is equal to g(n) to within a constant factor, we say that g(n) is an asymptotically tight bound for f(n)

Asymptotic Notation



By Lan Yao

- Asymptotic upper bound:

When we only have an asymptotic **upper bound**, we use O notation for a given function g(n), we denote by O(g(n)) the set of functions:

O(g(n))={ f(n) there exists positive constant c, n_0 such that 0<=f(n)<=cg(n) for all n>=n_0}

T(n)=O(g(n))--the asymptotic upper bound of the algorithm is g(n)

- Asymptotic lower bound:

  When we only have an asymptotic **lower bound**, we use Ω notation for a given function g(n), we denote by Ω(g(n)) the set of functions:

  $\Omega(g(n))=\{ f(n)$ there exists positive constant c, n_0 such that $0<=cg(n) <=f(n)$ for all $n>=n_0\}$

  $T(n)=\Omega(g(n))$--the asymptotic lower bound of the algorithm is g(n)

Examples

1. Compare the complexity of insertion and merge

$T_1(n)=an^2+bn+c$--n^2 ⇨

$T_2(n)=cnlgn+cn$--$nlgn$ ⇨

Theta Notation

- **Drop lower order terms**
- **Ignore leading constants**
- **Concentrates on the growth**

$$\lim_{n\to\infty}\left(\frac{nlgn}{n^2}\right) = ?$$

-
- $$\lim_{n\to\infty}\left(\frac{nlgn}{n^2}\right)$$

$$= \lim_{n\to\infty}\left(\frac{lgn}{n}\right)$$

$$= \lim_{n\to\infty}\left(\frac{1/n(ln2)}{1}\right)$$

$$\lim_{n\to\infty}\left(\frac{1}{ln2*n}\right)$$

=0

L'hopital's Rules:

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \lim_{n\to\infty}\left(\frac{f'(n)}{g'(n)}\right)$$

Where f'(n) is the derivative of f(n)

some basic derivatives: https://www.dummies.com/article/academics-the-arts/math/calculus/the-most-important-derivatives-and-antiderivatives-to-know-188540/

By Lan Yao

Desmo Classroom

https://student.desmos.com/?prepopulateCode=b33pqa