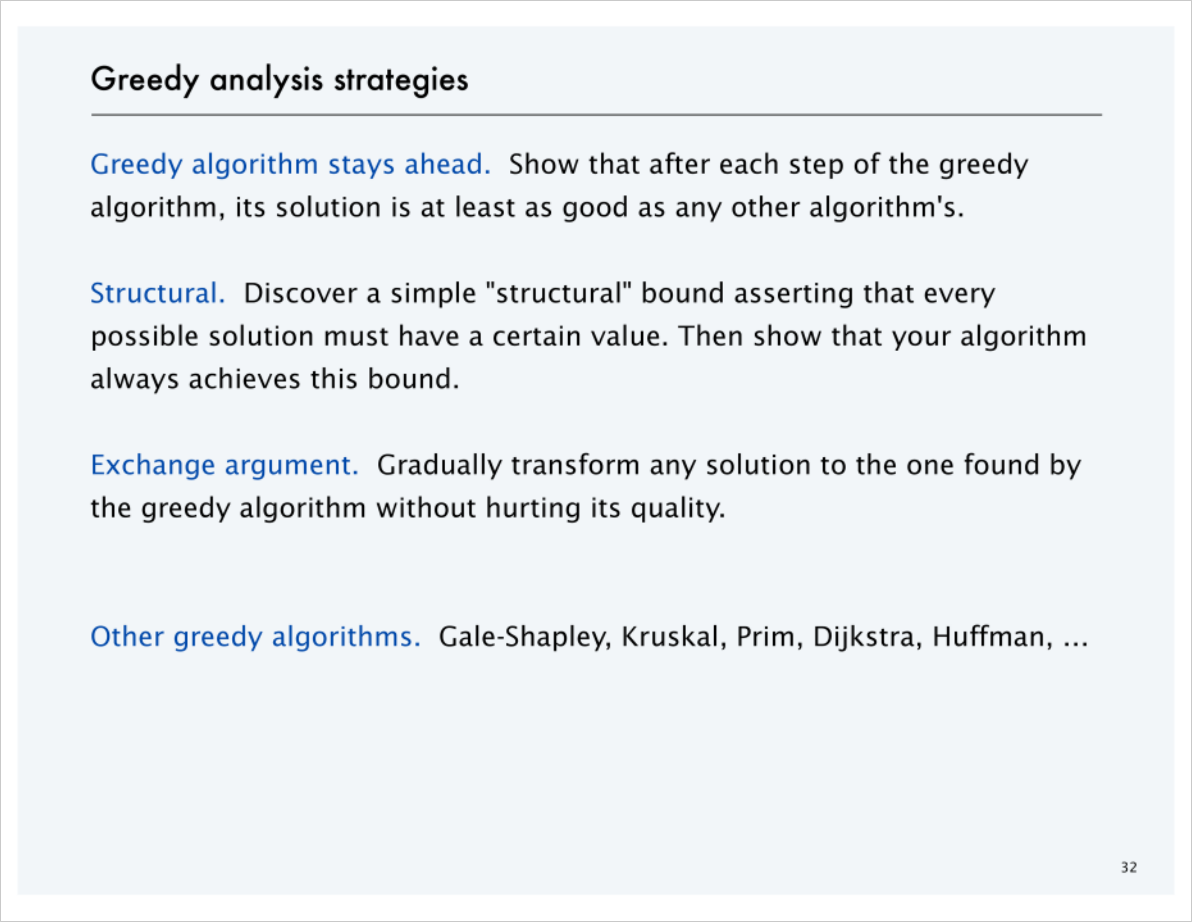
Lecture 8

Friday, June 2, 2017 5:38 PM



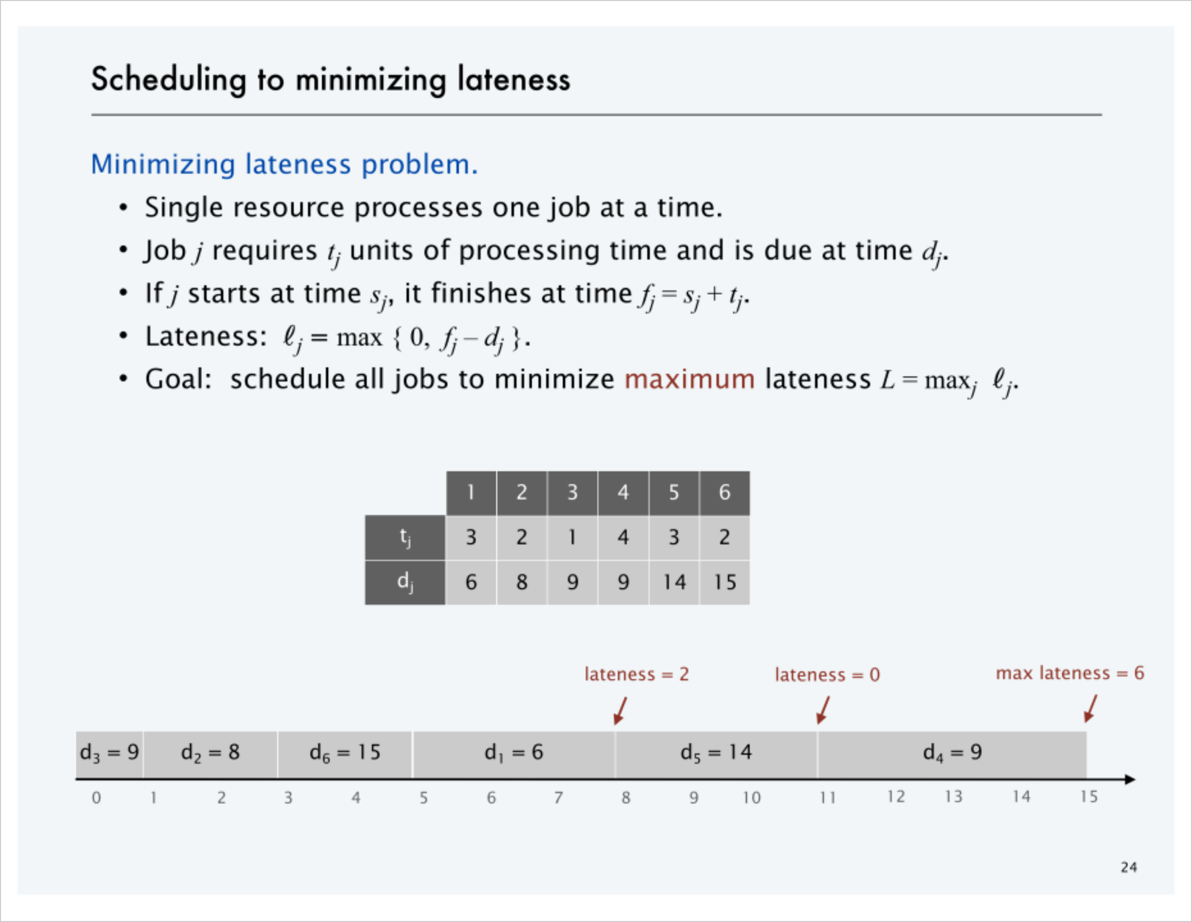
Greedy analysis strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Other greedy algorithms. Gale-Shapley, Kruskal, Prim, Dijkstra, Huffman, ...



Scheduling to minimizing lateness

Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires t_j units of processing time and is due at time d_j.
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max_j \ell_j$.

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

Minimizing lateness: greedy algorithms

Greedy template. Schedule jobs according to some natural order.

- [Shortest processing time first] Schedule jobs in ascending order of processing time t_j.
- [Earliest deadline first] Schedule jobs in ascending order of deadline d_j.
- [Smallest slack] Schedule jobs in ascending order of slack $d_j - t_j$.

Minimizing lateness: greedy algorithms

Greedy template. Schedule jobs according to some natural order.

- [Shortest processing time first] Schedule jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

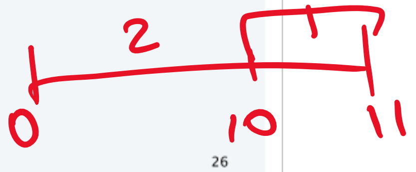
counterexample

- [Smallest slack] Schedule jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

1 0

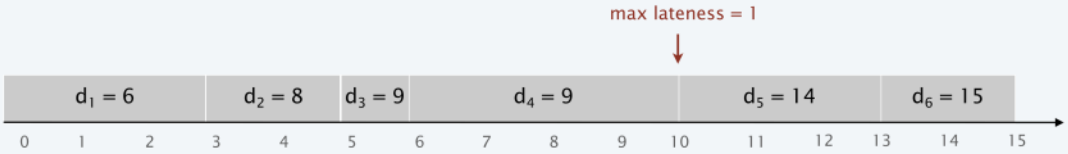
counterexample



Minimizing lateness: earliest deadline first

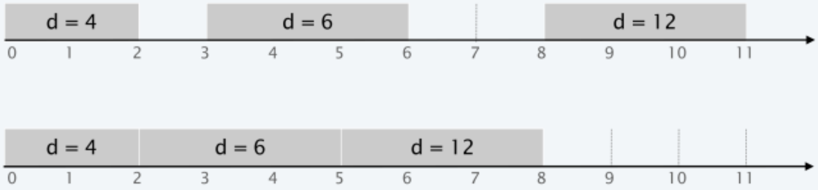
```
EARLIEST-DEADLINE-FIRST ($n$, $t_1$, $t_2$, …, $t_n$, $d_1$, $d_2$, …, $d_n$)

SORT $n$ jobs so that $d_1 \le d_2 \le \ldots \le d_n$.
$t \leftarrow 0$
FOR $j = 1$ TO $n$
    Assign job $j$ to interval $[t, t + t_j]$.
    $s_j \leftarrow t$ ; $f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$
RETURN intervals $[s_1, f_1]$, $[s_2, f_2]$, …, $[s_n, f_n]$.
```

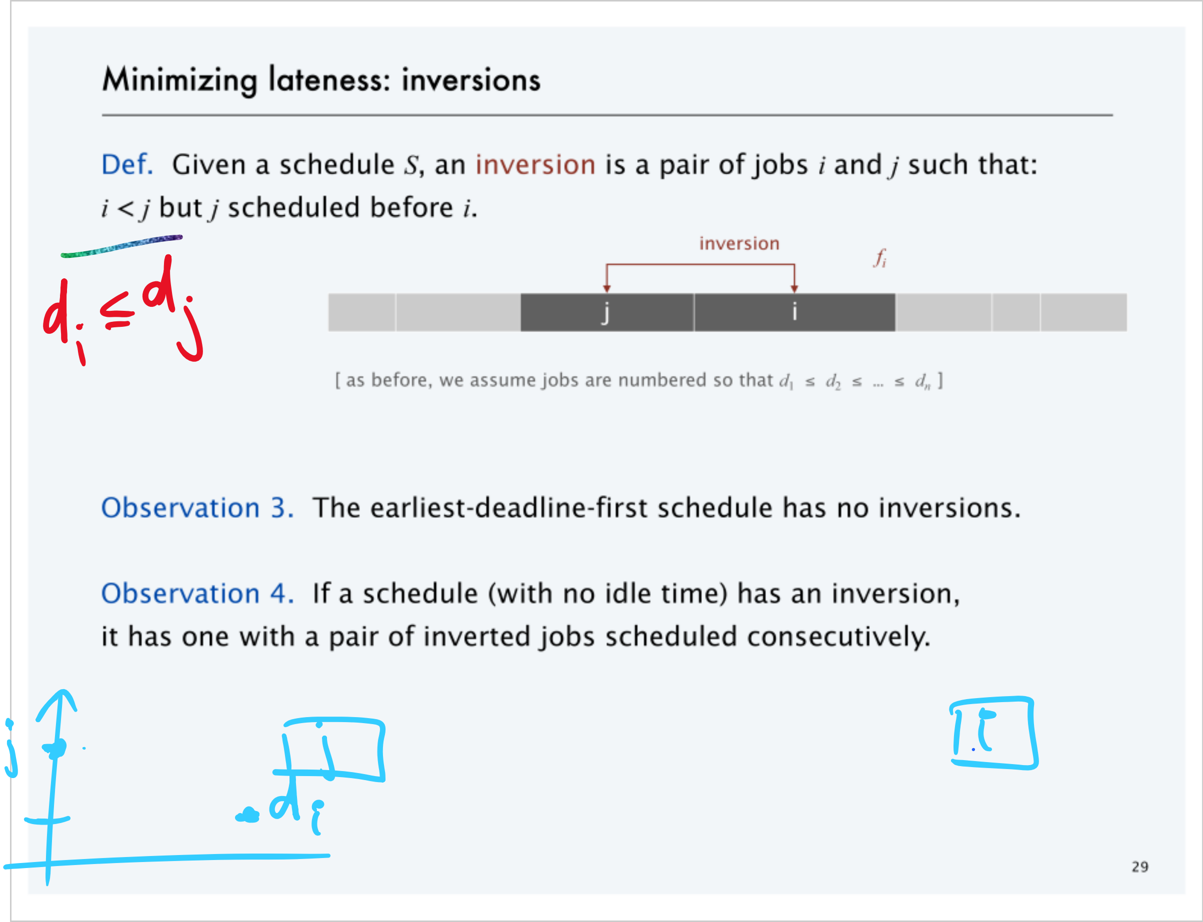


Minimizing lateness: no idle time

Observation 1. There exists an optimal schedule with no idle time.



Observation 2. The earliest-deadline-first schedule has no idle time.



Minimizing lateness: inversions

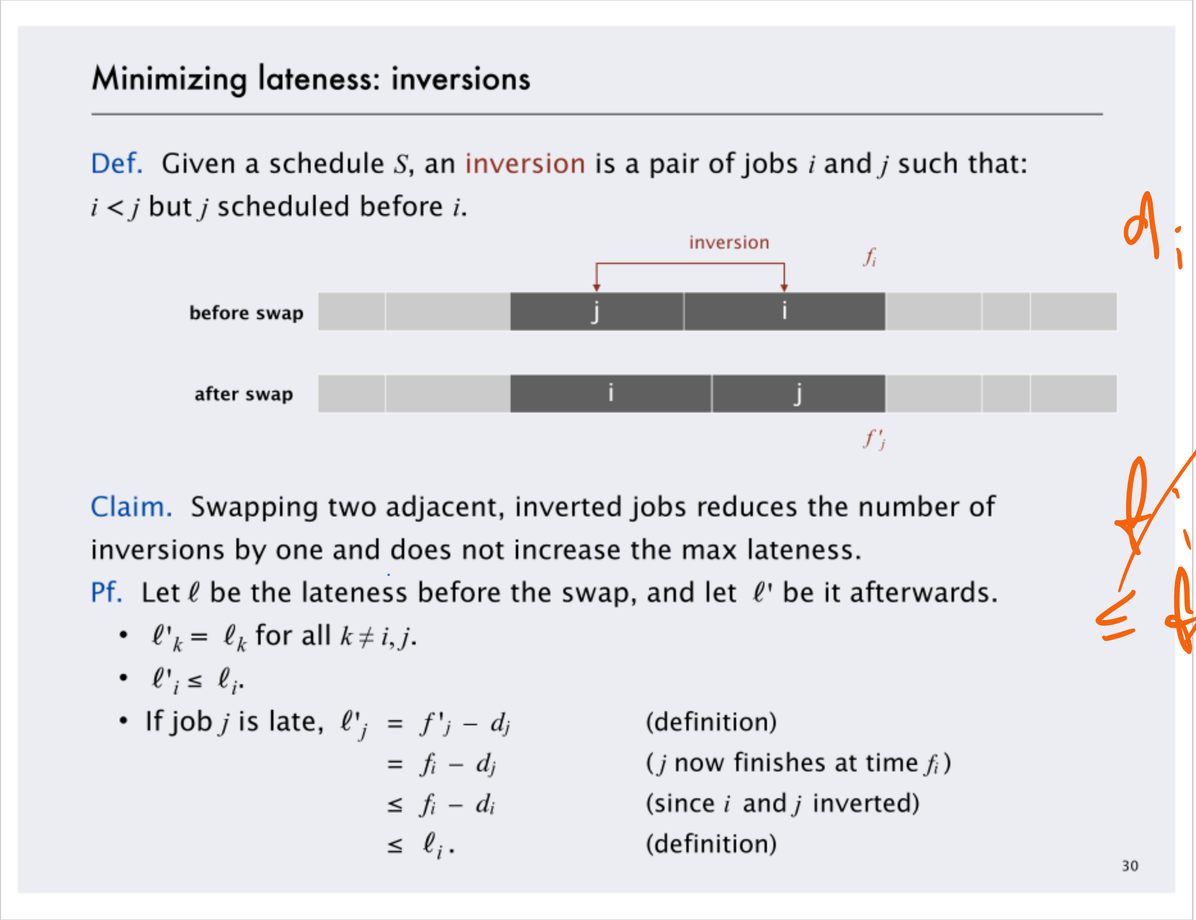
Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j scheduled before i.

$d_i \le d_j$

[as before, we assume jobs are numbered so that $d_1 \le d_2 \le \ldots \le d_n$]

Observation 3. The earliest-deadline-first schedule has no inversions.

Observation 4. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.



Minimizing lateness: inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $i < j$ but j scheduled before i.

$d_i \le d_j$

$\Downarrow$

$\cancel{f_i} - d_j$

$\le \cancel{f_i} - d_i$

Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

- $\ell'_k = \ell_k$ for all $k \ne i, j$.
- $\ell'_i \le \ell_i$.
- If job j is late, $\ell'_j = f'_j - d_j$ (definition)
  $= f_i - d_j$ (j now finishes at time f_i)
  $\le f_i - d_i$ (since i and j inverted)
  $\le \ell_i$. (definition)

Minimizing lateness: analysis of earliest-deadline-first algorithm

Theorem. The earliest-deadline-first schedule S is optimal.

Pf. [by contradiction]

Define S^* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S^* has no idle time.
- If S^* has no inversions, then $S = S^*$.
- If S^* has an inversion, let i–j be an adjacent inversion.
- Swapping i and j
  - does not increase the max lateness
  - strictly decreases the number of inversions
- This contradicts definition of S^* ▪

is still opt

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into two subproblems of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

$T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(n \log n)$.

n^2

$\downarrow$

$n \log n$

attributed to Julius Caesar

$\Rightarrow T(n) = O(n \log n)$

Sorting problem

Problem. Given a list of n elements from a totally-ordered universe, rearrange them in ascending order.

	Name	Artist	Time	Album
12	☑ Let It Be	The Beatles	4:03	Let It Be
13	☑ Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	☑ Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	☑ Dancing With Myself	Billy Idol	4:43	Don't Stop
16	☑ Rebel Yell	Billy Idol	4:49	Rebel Yell
17	☑ Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	☑ Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	☑ The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	☑ Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	☑ Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	☑ Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	☑ Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	☑ Hurricane	Bob Dylan	8:32	Desire
25	☑ The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	☑ Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	☑ Beds Of Roses	Bon Jovi	6:35	Cross Road
28	☑ Runaway	Bon Jovi	3:53	Cross Road
29	☑ Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	☑ Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	☑ Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	☑ Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	☑ Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	☐ Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	☑ Thunder Road	Bruce Springsteen	4:51	Born To Run
36	☑ Born To Run	Bruce Springsteen	4:30	Born To Run
37	☑ Jungleland	Bruce Springsteen	9:34	Born To Run

Sorting applications

Obvious applications.

- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

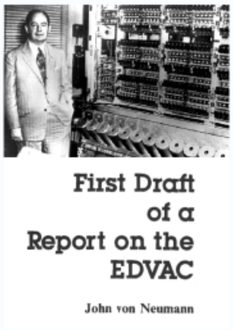
- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Minimum spanning trees (Kruskal's algorithm).
- Scheduling to minimize maximum lateness or average completion time.
- ...

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



First Draft
of a
Report on the
EDVAC

John von Neumann

input

A	L	G	O	R	I	T	H	M	S

sort left half

A	G	L	O	R	I	T	H	M	S

sort right half

A	G	L	O	R	H	I	M	S	T

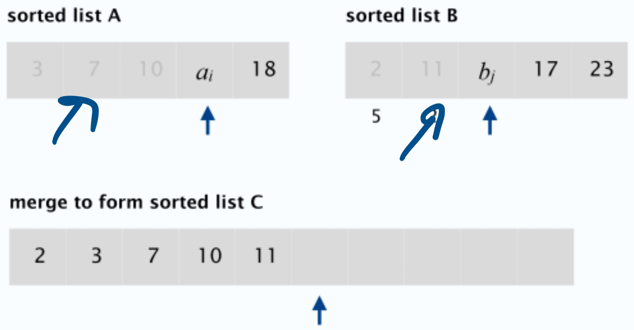
merge results

A	G	H	I	L	M	O	R	S	T

Merging

Goal. Combine two sorted lists A and B into a sorted whole C.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i \leq b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).



A useful recurrence relation

Def. $T(n)$ = max number of compares to mergesort a list of size $\leq n$.

Note. $T(n)$ is monotone nondecreasing.

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

Solution. $T(n)$ is $O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace $\leq$ with $=$.

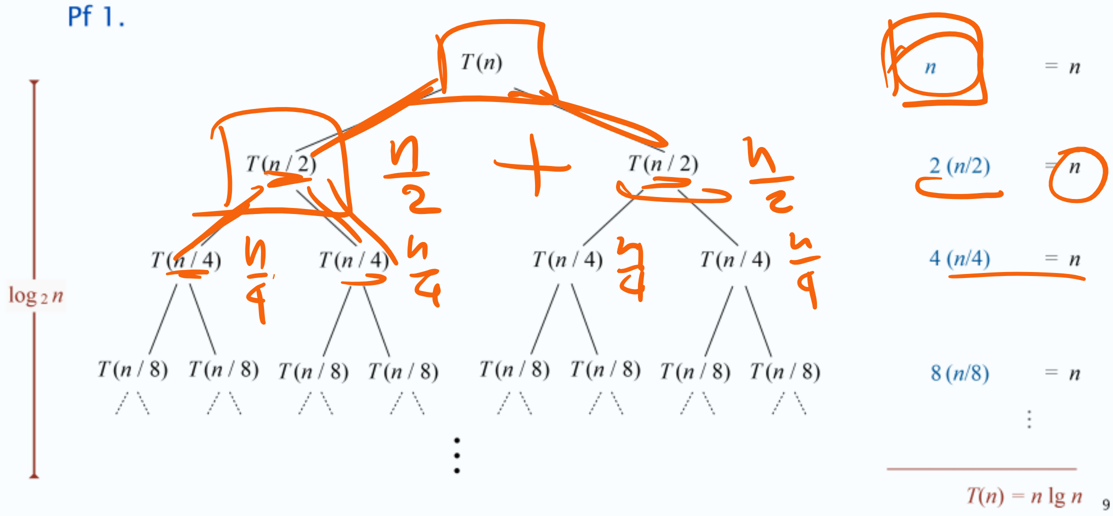
Divide-and-conquer recurrence: proof by recursion tree

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{otherwise} \end{cases}$$

Pf 1.



$\log_2 n$

$T(n)$

$T(n/2)$ $\frac{n}{2}$ + $T(n/2)$ $\frac{n}{2}$

$T(n/4)$ $\frac{n}{4}$ $T(n/4)$ $\frac{n}{4}$ $T(n/4)$ $\frac{n}{4}$ $T(n/4)$ $\frac{n}{4}$

$T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$ $T(n/8)$

$\vdots$

$n = n$

$2(n/2) = n$

$4(n/4) = n$

$8(n/8) = n$

$\vdots$

$T(n) = n \lg n$

Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

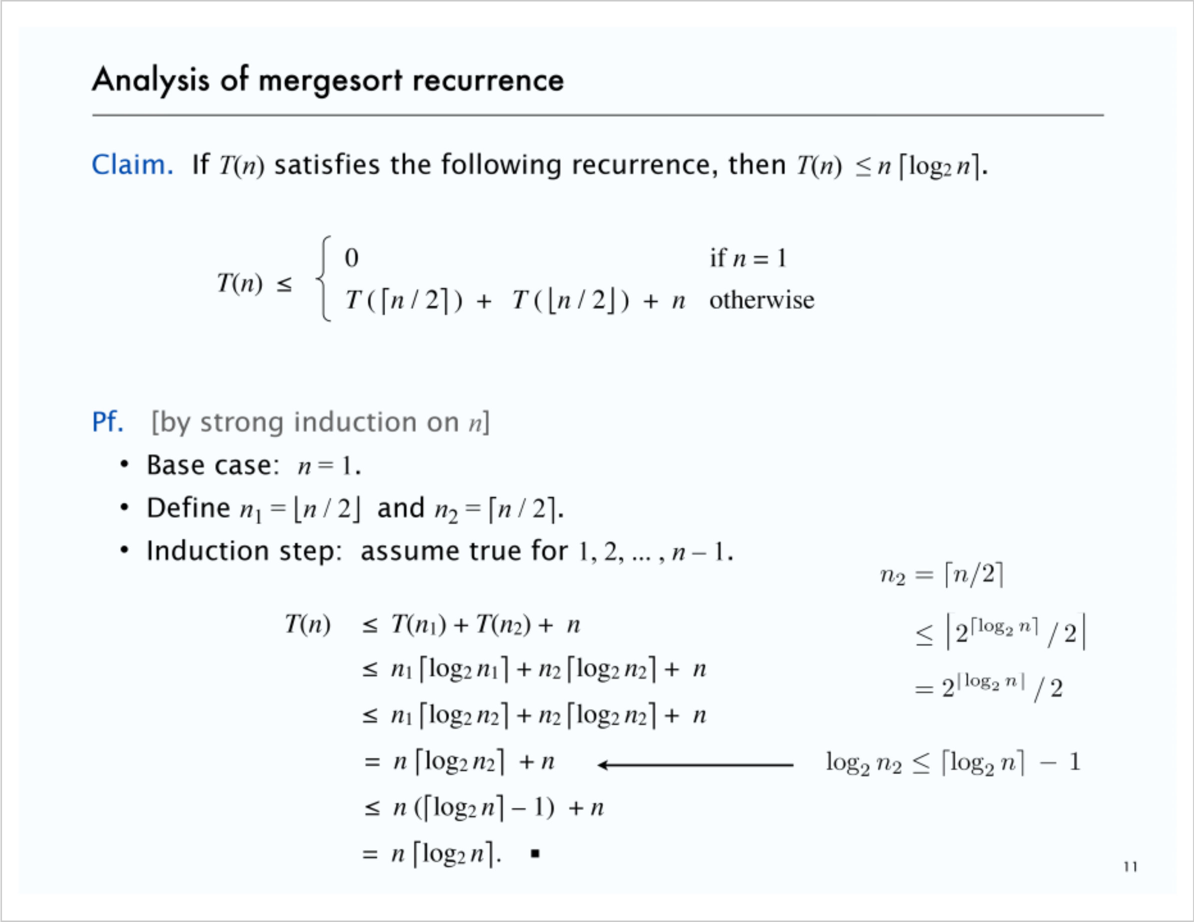
assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{otherwise} \end{cases}$$

Pf 2. [by induction on n]

- Base case: when $n = 1$, $T(1) = 0$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &= 2\,T(n) + 2n \\ &= 2\,n \log_2 n + 2n \\ &= 2\,n\,(\log_2 (2n) - 1) + 2n \\ &= 2\,n \log_2 (2n). \quad \blacksquare \end{aligned}$$



Analysis of mergesort recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \log_2 n \rceil$.

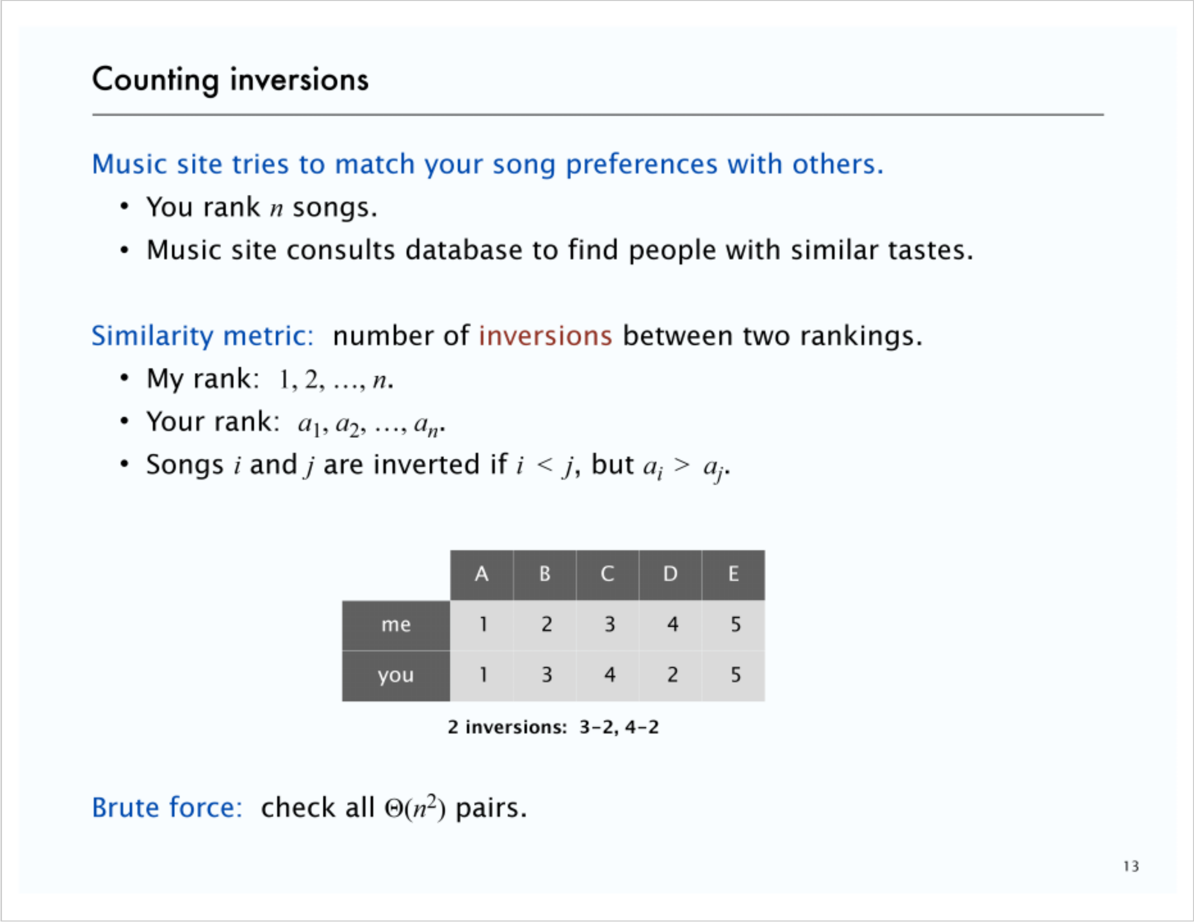
$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

Pf. [by strong induction on n]

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for $1, 2, \ldots, n-1$.

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\leq \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}$$

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \quad \longleftarrow \quad \log_2 n_2 \leq \lceil \log_2 n \rceil - 1 \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \\ &= n \lceil \log_2 n \rceil. \quad \blacksquare \end{aligned}$$



Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

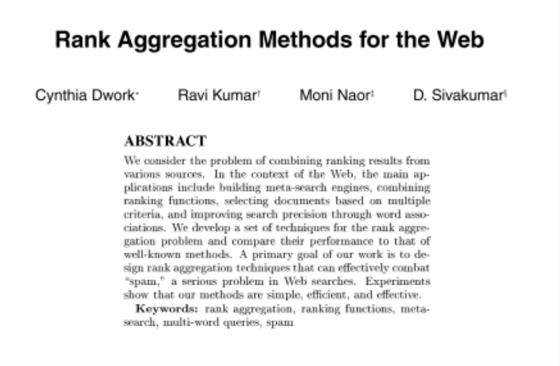
	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs.

Counting inversions: applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's tau distance).



Rank Aggregation Methods for the Web

Cynthia Dwork[*] Ravi Kumar[†] Moni Naor[‡] D. Sivakumar[§]

ABSTRACT

We consider the problem of combining ranking results from various sources. In the context of the Web, the main applications include building meta-search engines, combining ranking functions, selecting documents based on multiple criteria, and improving search precision through word associations. We develop a set of techniques for the rank aggregation problem and compare their performance to that of well-known methods. A primary goal of our work is to design rank aggregation techniques that can effectively combat "spam," a serious problem in Web searches. Experiments show that our methods are simple, efficient, and effective.

Keywords: rank aggregation, ranking functions, meta-search, multi-word queries, spam

---

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with $a \in A$ and $b \in B$.
- Return sum of three counts.

~~$T(n) = 2 \cdot T(\frac{n}{2}) + n$~~

input

1	5	4	8	10	2	6	9	3	7

count inversions in left half A

1	5	4	8	10

5-4

count inversions in right half B

2	6	9	3	7

6-3 9-3 9-7

count inversions (a, b) with a ∈ A and b ∈ B

1	5	4	8	10	2	6	9	3	7

4-2 4-3 5-2 5-3 8-2 8-3 8-6 8-7 10-2 10-3 10-6 10-7 10-9

output 1 + 3 + 13 = 17

Counting inversions: how to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

$T(n) = 2 \cdot T(\frac{n}{2}) + \cancel{O(n \cdot \log n)} + O(n)$

Warmup algorithm.

- Sort A and B.
- For each element $b \in B$,
  - binary search in A to find how elements in A are greater than b.

list A

7	10	18	3	14

list B

17	23	2	11	16

sort A

3	7	10	14	18

sort B

2	11	16	17	23

⑤+2+1+1

binary search to count inversions (a, b) with a ∈ A and b ∈ B

3	7	10	14	18

2	11	16	17	23
5	2	1	1	0

Counting inversions: how to combine two subproblems?

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

$3 > 2$

count inversions (a, b) with a ∈ A and b ∈ B

3	7	10	a_i	18

2	11	b_j	17	23
5	2			

$5 + 0 +$

merge to form sorted list C

2	3	7	10	11					

Counting inversions: divide-and-conquer algorithm implementation

Input. List L.

Output. Number of inversions in L and sorted list of elements L'.

```
SORT-AND-COUNT ($L$)
IF list $L$ has one element
    RETURN $(0, L)$.
DIVIDE the list into two halves $A$ and $B$.
$(r_A, A) \leftarrow$ SORT-AND-COUNT($A$).
$(r_B, B) \leftarrow$ SORT-AND-COUNT($B$).
$(r_{AB}, L') \leftarrow$ MERGE-AND-COUNT($A$, $B$).
RETURN $(r_A + r_B + r_{AB}, L')$.
```

sorted version of input A

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

$$T(1) = 0$$

Counting inversions: divide-and-conquer algorithm analysis

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Pf. The worst-case running time $T(n)$ satisfies the recurrence:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise} \end{cases}$$

Integer addition

Addition. Given two n-bit integers a and b, compute $a + b$.

Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

Integer multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

Conjecture. [Kolmogorov 1952] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is wrong.

Divide-and-conquer multiplication

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
- Multiply four ½n-bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

use bit shifting to compute 4 terms

$$(2^m a + b)(2^m c + d) = \underset{①}{2^{2m} ac} + 2^m (\underset{②}{bc} + \underset{③}{ad}) + \underset{④}{bd}$$

Ex. $x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b}$ $\quad y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$

$x = \boxed{a}\ b$

$y = c\ d$

$x = a \cdot 2^{\frac{n}{2}} + b$

$y = c \cdot 2^{\frac{n}{2}} + d$

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

$T(1) = 1$

$T(n) = \Theta(n^2)$

Divide-and-conquer multiplication

```
MULTIPLY(x, y, n)

IF (n = 1)
    RETURN x × y.
ELSE
    m ← ⌈ n / 2 ⌉.
    a ← ⌊ x / 2^m ⌋;   b ← x mod 2^m.
    c ← ⌊ y / 2^m ⌋;   d ← y mod 2^m.
    e ← MULTIPLY(a, c, m).
    f ← MULTIPLY(b, d, m).
    g ← MULTIPLY(b, c, m).
    h ← MULTIPLY(a, d, m).
    RETURN 2^{2m} e + 2^m (g + h) + f.
```

Divide-and-conquer multiplication analysis

Proposition. The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two n-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$\boxed{bc + ad = ac + bd - \boxed{(a-b)(c-d)}}$$

$m = \lceil n/2 \rceil$

$a = \lfloor x/2^m \rfloor \quad b = x \bmod 2^m$

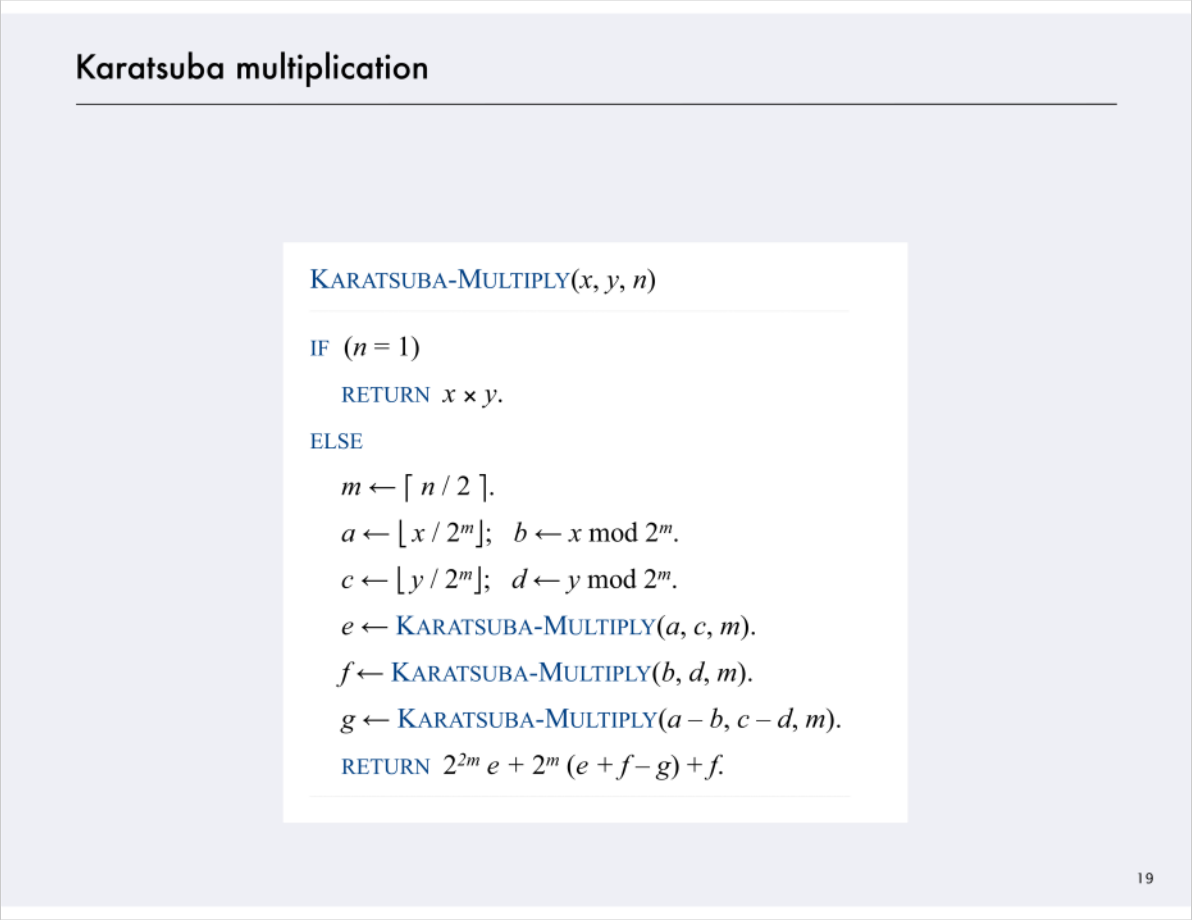
$c = \lfloor y/2^m \rfloor \quad d = y \bmod 2^m$

$$(2^m a + b)(2^m c + d) = 2^{2m}\,ac + 2^m \overset{\text{middle term}}{\overset{\downarrow}{(bc + ad)}} + bd$$

$$= 2^{2m}\,\underset{❶}{ac} + 2^m\,(\underset{❶}{ac} + \underset{❸}{bd} - \underset{❷}{(a-b)(c-d)}) + \underset{❸}{bd}$$

$T(n) = 3 \cdot T(\frac{n}{2}) + n$

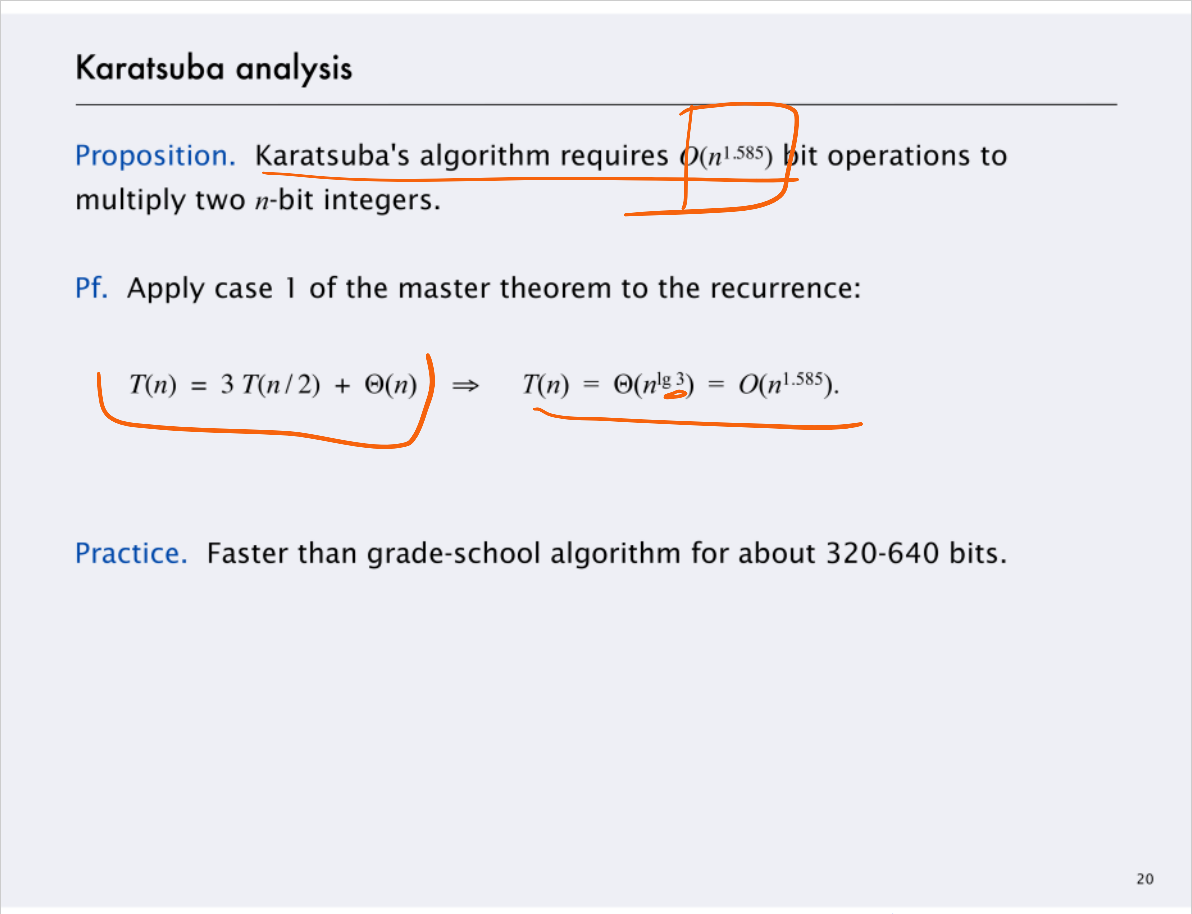
Bottom line. Only three multiplication of $n/2$-bit integers.



Karatsuba multiplication

```
KARATSUBA-MULTIPLY($x, y, n$)

IF ($n = 1$)
    RETURN $x \times y$.
ELSE
    $m \leftarrow \lceil n / 2 \rceil$.
    $a \leftarrow \lfloor x / 2^m \rfloor$;   $b \leftarrow x \bmod 2^m$.
    $c \leftarrow \lfloor y / 2^m \rfloor$;   $d \leftarrow y \bmod 2^m$.
    $e \leftarrow$ KARATSUBA-MULTIPLY($a, c, m$).
    $f \leftarrow$ KARATSUBA-MULTIPLY($b, d, m$).
    $g \leftarrow$ KARATSUBA-MULTIPLY($a - b, c - d, m$).
    RETURN $2^{2m} e + 2^m (e + f - g) + f$.
```



Karatsuba analysis

Proposition. Karatsuba's algorithm requires $\boxed{O(n^{1.585})}$ bit operations to multiply two n-bit integers.

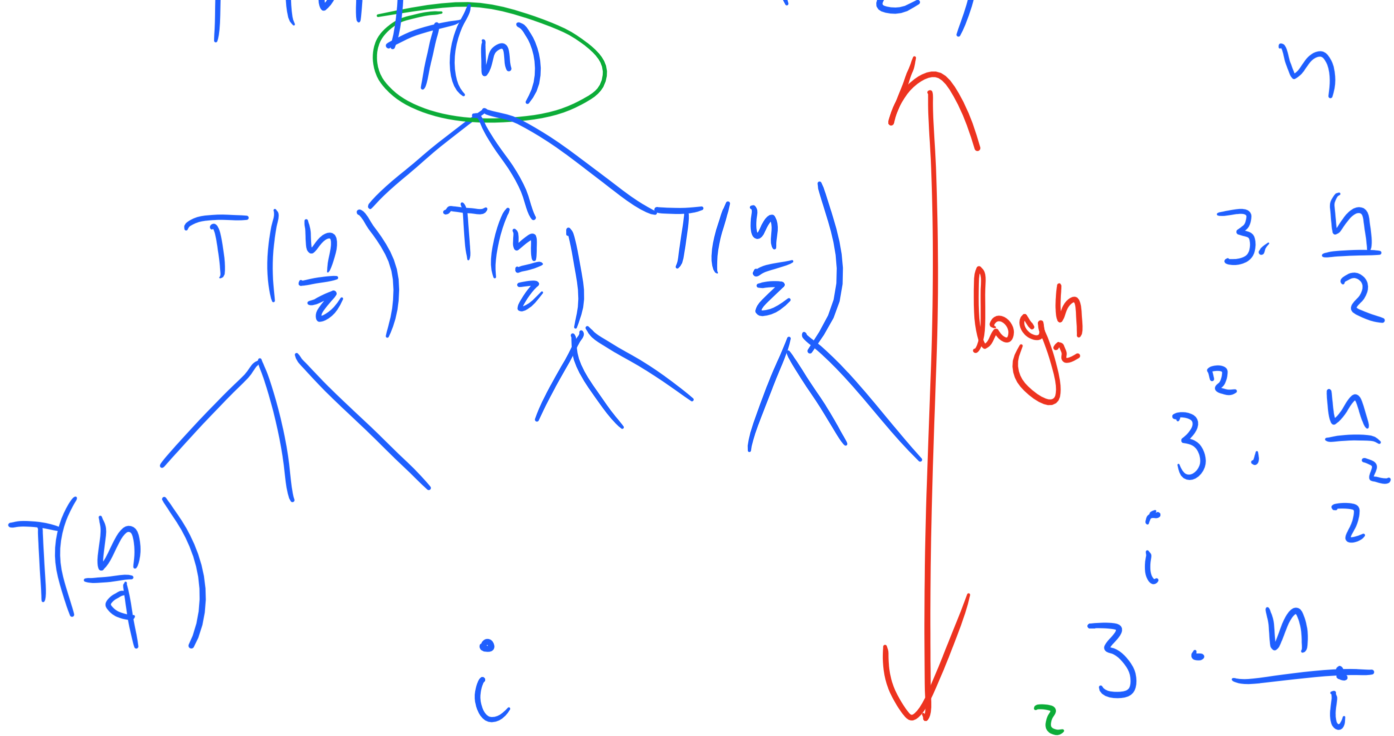
Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) \;=\; 3\,T(n/2) \;+\; \Theta(n) \;\Rightarrow\; T(n) \;=\; \Theta(n^{\lg 3}) \;=\; O(n^{1.585}).$$

Practice. Faster than grade-school algorithm for about 320-640 bits.

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + n$$

$T(n) = 3 \cdot T(\frac{n}{2}) + n$



Total time: $\underbrace{n + \frac{3}{2} n + \left(\frac{3}{2}\right)^2 n + \dots}_{\log_2 n}$

$= n \cdot \left(\underbrace{1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \dots + \left(\frac{3}{2}\right)^i + \dots}_{\left(\frac{3}{2}\right)^{\log_2 n}} \right)$

$$\boxed{\sum_{i=0}^{t} q^i = \frac{1-q^{t+1}}{1-q}}$$

$q = \frac{3}{2}$ $\quad t = \log_2 n$

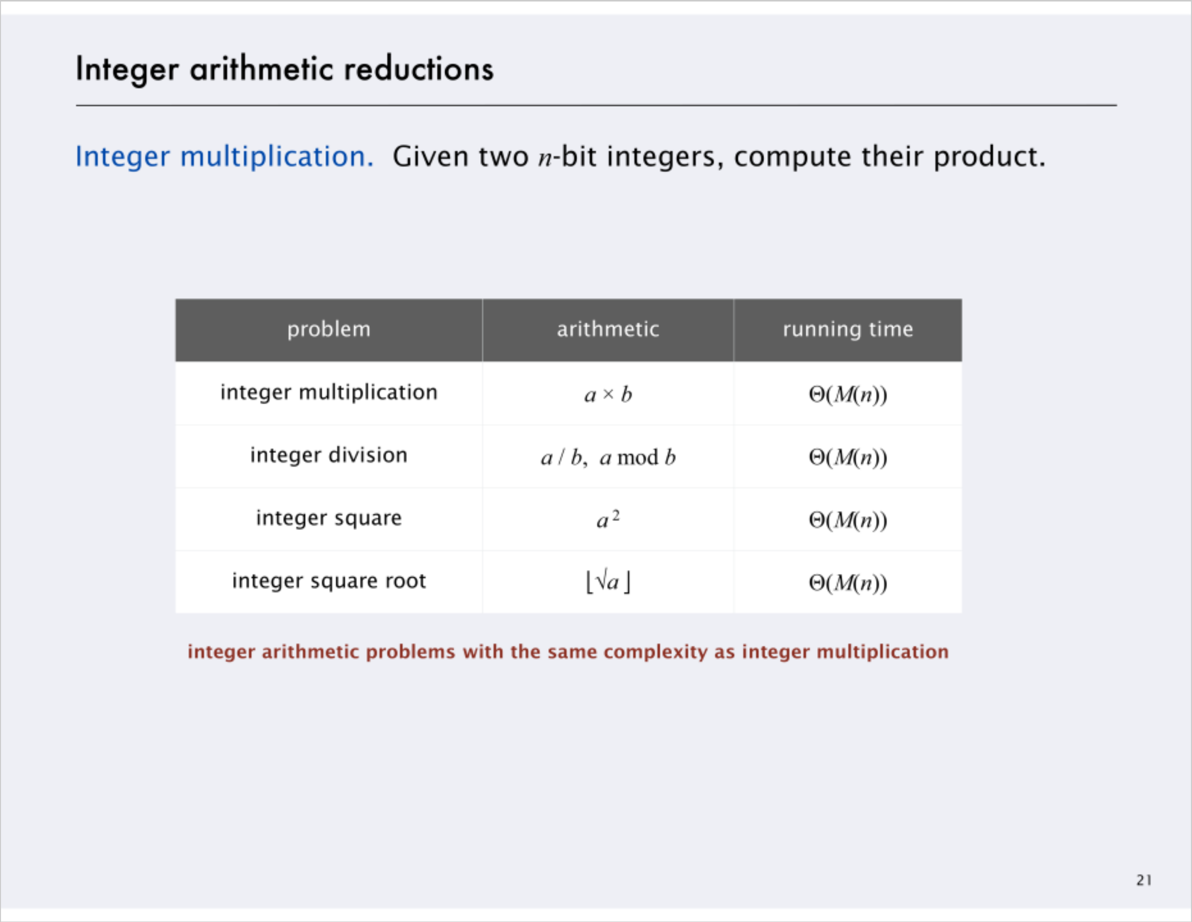
$\text{Time} = n \cdot \frac{1-\left(\frac{3}{2}\right)^{\log_2 n+1}}{1-\frac{3}{2}}$

$$\text{Time} = \frac{\quad}{1-\frac{3}{2}}$$

$$= \frac{n\cdot\left(\frac{3}{2}\right)^{\log_2 n ①} - \cancel{1}}{\cancel{1/2}}$$

$$\approx n\cdot\frac{3}{2}\cdot\left(\frac{3}{2}\right)^{\log_2 n} = n\cdot\frac{3}{2}\cdot\frac{3^{\log_2 n}}{2^{\log_2 n}}$$

$$= \cancel{n}\cdot\frac{3}{2}\cdot\frac{3^{\log_2 n}}{\cancel{n}} = \frac{3}{2}\cdot\underbrace{\left(2^{\log_2 3}\right)^{\log_2 n}}_{\left(2^{\log_2 n}\right)^{\log_2 3} \;?! \;=\; n^{\log_2 3}}$$

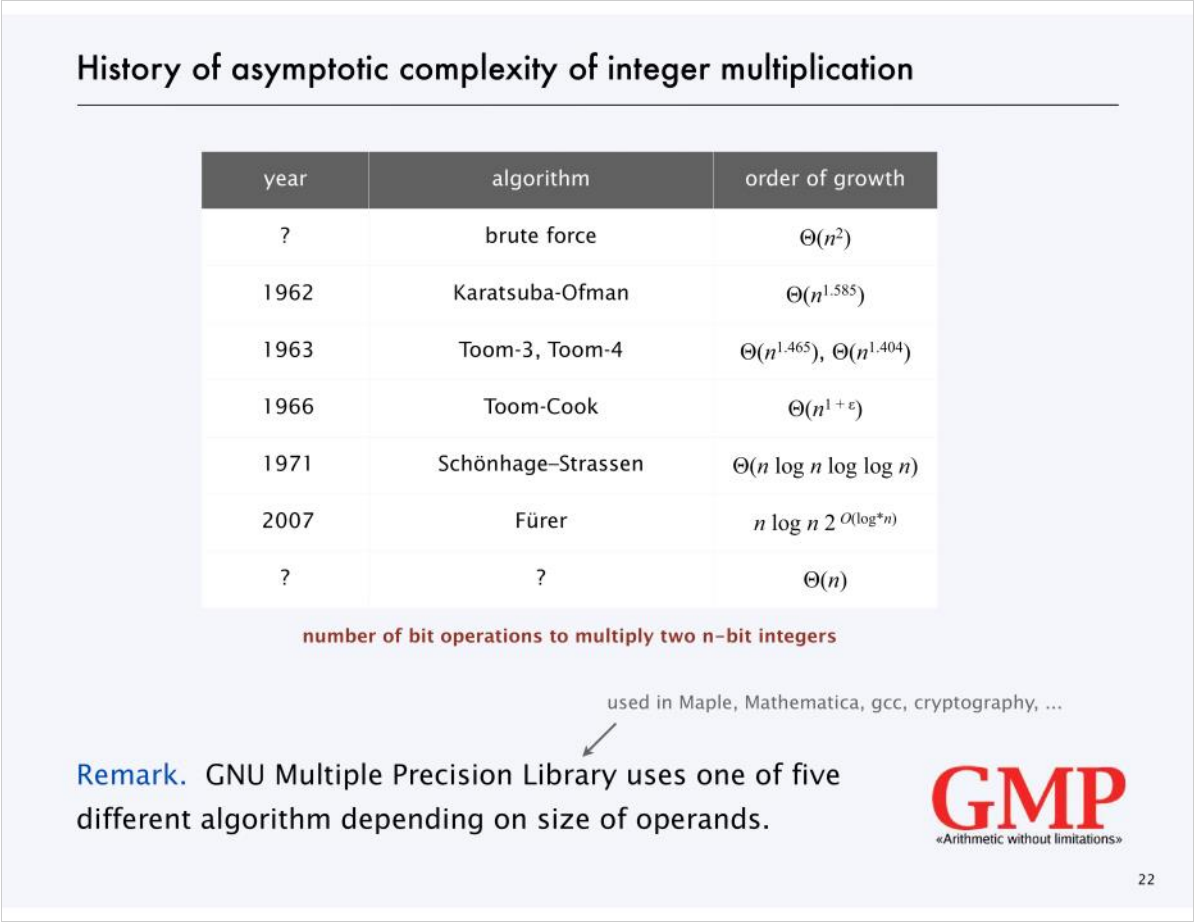


Integer arithmetic reductions

Integer multiplication. Given two n-bit integers, compute their product.

problem	arithmetic	running time
integer multiplication	$a \times b$	$\Theta(M(n))$
integer division	$a / b,\ a \bmod b$	$\Theta(M(n))$
integer square	a^2	$\Theta(M(n))$
integer square root	$\lfloor \sqrt{a} \rfloor$	$\Theta(M(n))$

integer arithmetic problems with the same complexity as integer multiplication



History of asymptotic complexity of integer multiplication

year	algorithm	order of growth
?	brute force	$\Theta(n^2)$
1962	Karatsuba-Ofman	$\Theta(n^{1.585})$
1963	Toom-3, Toom-4	$\Theta(n^{1.465})$, $\Theta(n^{1.404})$
1966	Toom-Cook	$\Theta(n^{1+\varepsilon})$
1971	Schönhage–Strassen	$\Theta(n \log n \log \log n)$
2007	Fürer	$n \log n \, 2^{O(\log^* n)}$
?	?	$\Theta(n)$

number of bit operations to multiply two n-bit integers

used in Maple, Mathematica, gcc, cryptography, ...

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

GMP
«Arithmetic without limitations»