Divide-and-Conquer

Divide-and-conquer.

- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.

- Break up problem of size n into two equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.

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

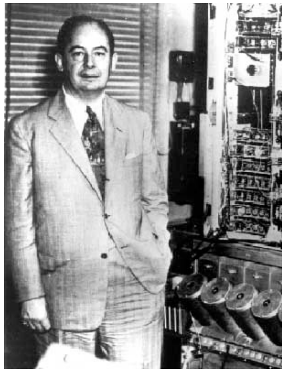
> Divide et impera.
> Veni, vidi, vici.
> - Julius Caesar

5.1 Mergesort

Mergesort

Mergesort.

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



Jon von Neumann (1945)

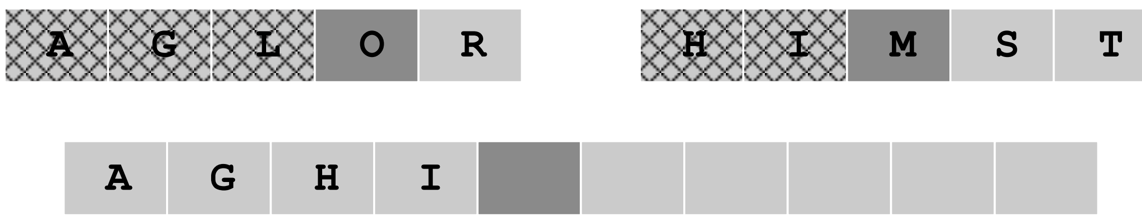
A	L	G	O	R	I	T	H	M	S				
A	L	G	O	R		I	T	H	M	S	divide	$O(1)$
A	G	L	O	R		H	I	M	S	T	sort	$2T(n/2)$
A	G	H	I	L	M	O	R	S	T		merge	$O(n)$

Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.



Challenge for the bored. In-place merge. [Kronrod, 1969]

using only a constant amount of extra storage

A Useful Recurrence Relation

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

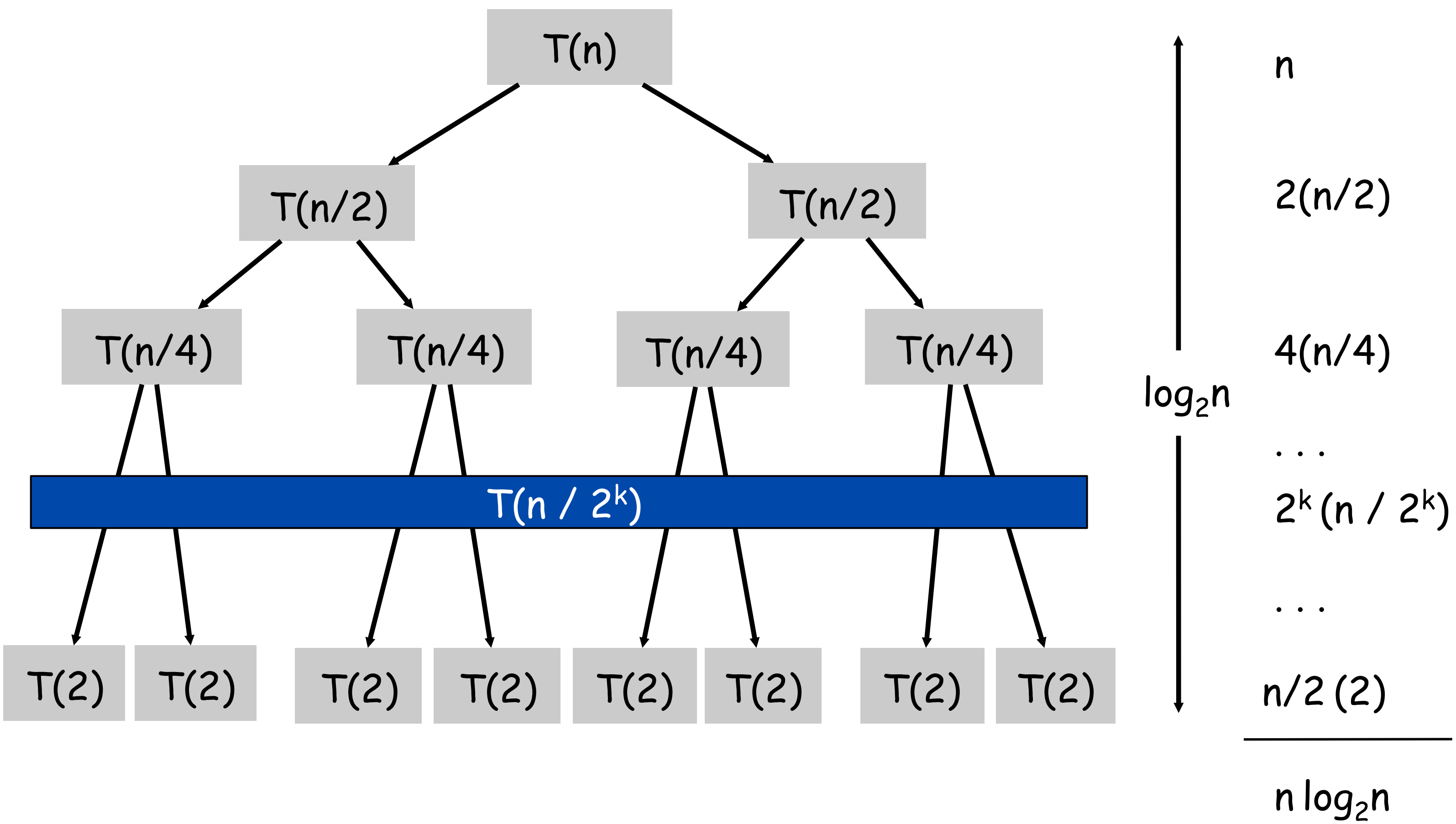
$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

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

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

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Proof by Induction

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

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

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n\big(\log_2(2n) - 1\big) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

Analysis of Mergesort Recurrence

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

$\uparrow$ $\log_2 n$

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on n)

- Base case: $n = 1$. $T(1) = 0 = 1 \lceil \lg 1 \rceil$.
- Define $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$.
- Induction step: Let $n \ge 2$, assume true for 1, 2, ... , n-1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\le \lceil \lg n \rceil - 1 \end{aligned}$$

5.4 Closest Pair of Points

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version. O(n log n) easy if points are on a line.

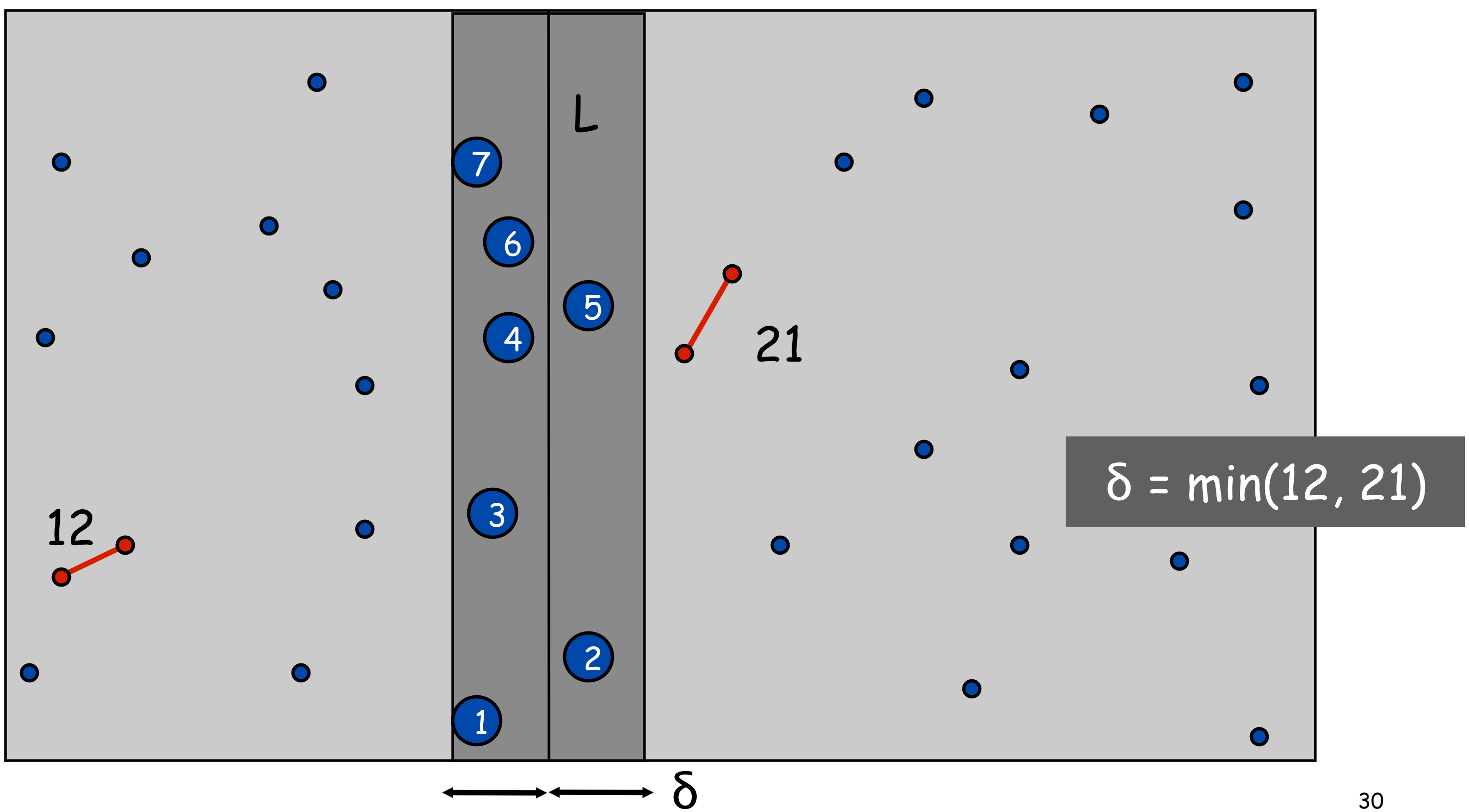
Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!



Closest Pair of Points

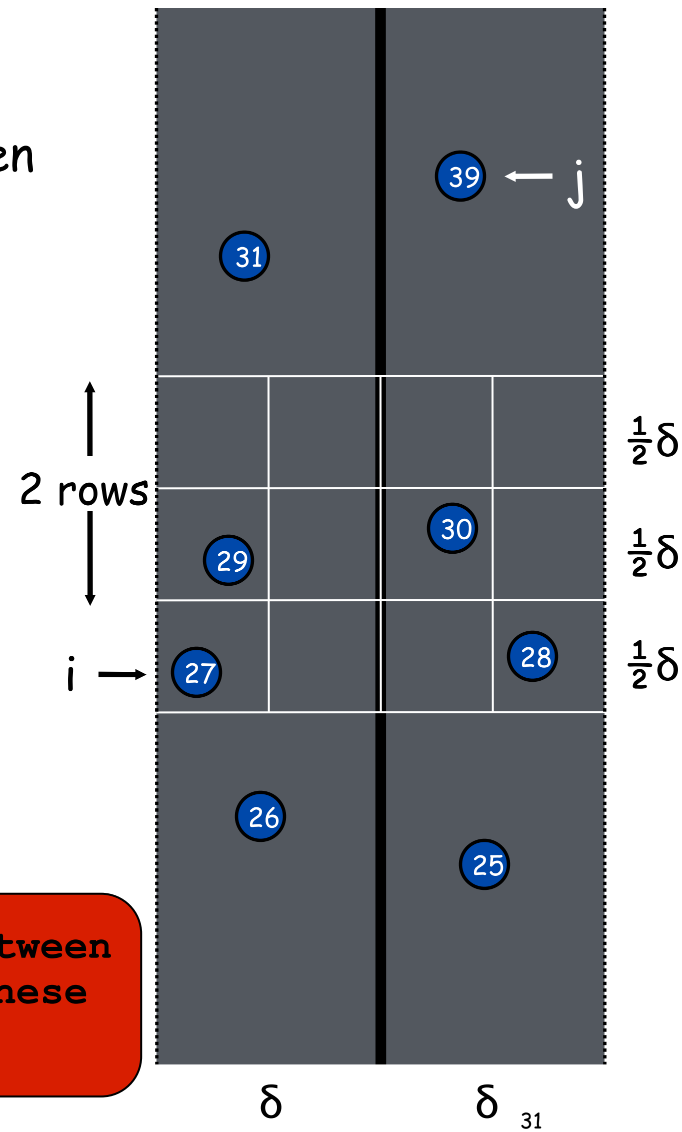
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta) = \delta$. ▪

Fact. Still true if we replace 12 with 7.



Scan points in y-order and compare distance between each point and next 11 neighbours. If any of these distances is less than δ, update δ.

Closest Pair of Points

```
Smallest-Dist(p₁, …, pₙ) {

   if n=2 then return dist(p₁,p₂)

   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ′ = Smallest-Dist(left half)
   δ″ = Smallest-Dist(right half)
   δ  = min(δ′,δ″)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbours. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

Closest Pair of Points

```
Closest-Pair(p₁, …, pₙ) {

   if n=2 then return dist(p₁,p₂),p₁,p₂

   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ',p',q'  = Closest-Pair(left half)
   δ'',p'',q'' = Closest-Pair(right half)
   δ, p, q  = min(δ', δ'')(p',q',p'',q'')

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbours. If any of these
   distances is less than δ, update δ,p,q.

   return δ,p,q.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

Closest Pair of Points: Analysis

Running time.

$$T(n) \le 2T(n/2) + O(n \log n) \;\Rightarrow\; T(n) \in O(n \log^2 n)$$

Q. Can we achieve O(n log n)?

A. Yes. First sort all points according to x coordinate before algo.
Don't sort points in strip from scratch each time.

- Each recursion returns a list: all points sorted by y coordinate.
- Sort by merging two pre-sorted lists.

$$T(n) \le 2T(n/2) + O(n) \;\Rightarrow\; T(n) \in O(n \log n)$$

Matrix Multiplication

Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices A and B, compute C = AB.

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Brute force. $\Theta(n^3)$ arithmetic operations.

Fundamental question. Can we improve upon brute force?

Matrix Multiplication: Warmup

Divide-and-conquer.

- Divide: partition A and B into $\frac{1}{2}$n-by-$\frac{1}{2}$n blocks.
- Conquer: multiply 8 $\frac{1}{2}$n-by-$\frac{1}{2}$n recursively.
- Combine: add appropriate products using 4 matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= \left(A_{11} \times B_{11}\right) + \left(A_{12} \times B_{21}\right) \\ C_{12} &= \left(A_{11} \times B_{12}\right) + \left(A_{12} \times B_{22}\right) \\ C_{21} &= \left(A_{21} \times B_{11}\right) + \left(A_{22} \times B_{21}\right) \\ C_{22} &= \left(A_{21} \times B_{12}\right) + \left(A_{22} \times B_{22}\right) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T\left(n/2\right)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) \in \Theta(n^3)$$

Matrix Multiplication: Key Idea

Key idea. multiply 2-by-2 block matrices with only 7 multiplications.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

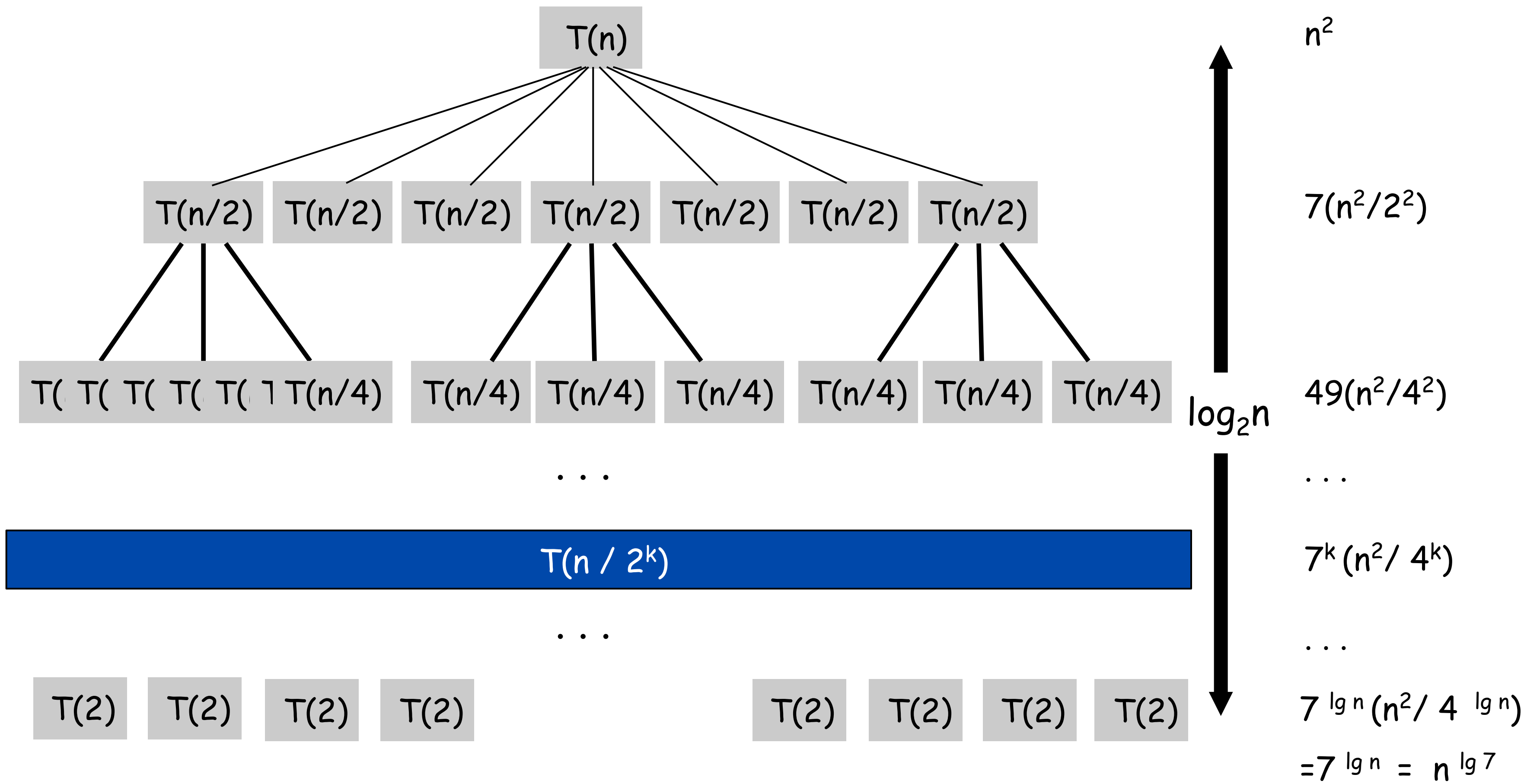
- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).

Strassen: Recursion Tree

$$\sum_{k=0}^{n-1} ar^k = a\,\frac{1-r^n}{1-r}$$

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 7T(n/2) + n^2 & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\log_2 n} n^2\left(\tfrac{7}{4}\right)^k = n^2\,\frac{\left(\tfrac{7}{4}\right)^{1+\log_2 n} - 1}{\tfrac{7}{4} - 1} \approx \tfrac{7}{3}\, n^{\log_2 7}$$



Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)

- Divide: partition A and B into $\frac{1}{2}$n-by-$\frac{1}{2}$n blocks.
- Compute: 14 $\frac{1}{2}$n-by-$\frac{1}{2}$n matrices via 10 matrix additions.
- Conquer: multiply 7 $\frac{1}{2}$n-by-$\frac{1}{2}$n matrices recursively.
- Combine: 7 products into 4 terms using 18 matrix additions.

Analysis.

- Assume n is a power of 2.
- T(n) = # arithmetic operations.

$$\mathrm{T}(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow \mathrm{T}(n) \in \Theta(n^{\log_2 7}) \in O(n^{2.81})$$

Fast Matrix Multiplication in Practice

Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around n = 128.

Common misperception: "Strassen is only a theoretical curiosity."

- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when n ~ 2,500.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" Ax=b, determinant, eigenvalues, and other matrix ops.

Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?
A. Yes! [Strassen, 1969] $\Theta(n^{\log_2 7}) \in O(n^{2.81})$

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?
A. **Impossible.** [Hopcroft and Kerr, 1971] $\Theta(n^{\log_2 6}) \in O(n^{2.59})$

Q. Two 3-by-3 matrices with only 21 scalar multiplications?
A. **Also impossible.** $\Theta(n^{\log_3 21}) \in O(n^{2.77})$

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?
A. Yes! [Pan, 1980] $\Theta(n^{\log_{70} 143640}) \in O(n^{2.80})$

Decimal wars.

- December, 1979: $O(n^{2.521813})$.
- January, 1980: $O(n^{2.521801})$.

Fast Matrix Multiplication in Theory

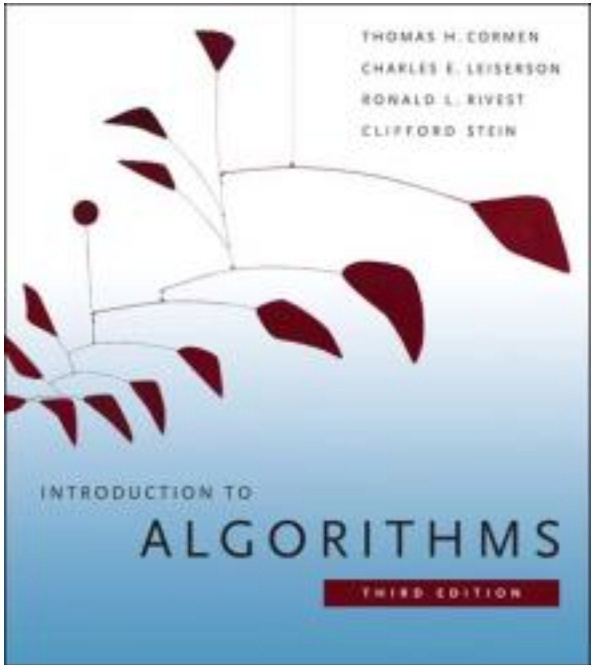
Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987-2010.]

In 2010, Andrew Stothers gave an improvement to the algorithm $O(n^{2.374})$.
In 2011, Virginia Williams combined a mathematical short-cut from Stothers' paper with her own insights and automated optimization on computers, improving the bound $O(n^{2.3728642})$.
In 2014, François Le Gall simplified the methods of Williams and obtained an improved bound of $O(n^{2.3728639})$.

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical (hidden constant gets worse).

CLRS 4.3 Master Theorem



Master Theorem from CLRS 4.3

Used for many divide-and-conquer recurrences

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n) > 0$.

a = (constant) number of sub-instances,
b = (constant) size ratio of sub-instances,
f(n) = time used for dividing and recombining.

Based on the ***master theorem*** (Theorem 4.1).

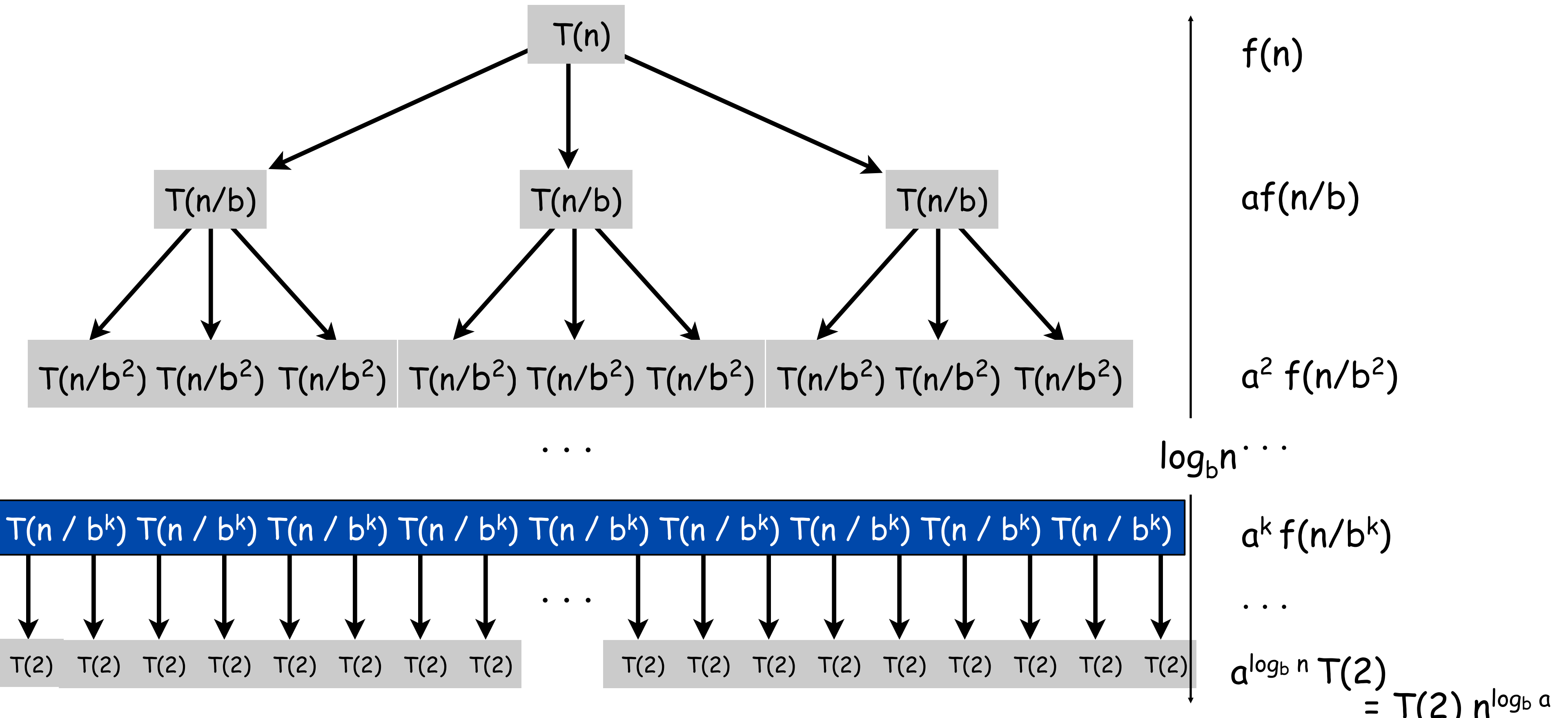
Compare $n^{\log_b a}$ vs. $f(n)$:

Proof by Recursion Tree

Used for many divide-and-conquer recurrences

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n) > 0$.



$T(n) = \sum a^k f(n/b^k)$

$$T(n) = aT(n/b) + f(n)$$

Case 1: $f(n) \in O(n^L)$ for some constant $L < \log_b a$.

Solution: $T(n) \in \Theta(n^{\log_b a})$

Case 2: $f(n) \in \Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

Case 3: $f(n) \in \Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

Solution: $T(n) \in \Theta(f(n))$

Master Theorem

Case 2: $f(n) \in \Theta(n^{\log_b a} \log^k n)$, for some $k \geq 0$.

Solution: $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$

$$T(n) = 27T(n/3) + \Theta(n^3 \log n)$$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **Case 2**

Solution: $T(n) \in \Theta(n^3 \log^2 n)$

Master Theorem

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$, $b > 1$, and $f(n) > 0$.

Case 3: $f(n) \in \Omega(n^L)$ for some constant $L > \log_b a$

and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c<1$ and all large n.

($f(n)$ is polynomially greater than $n^{\log_b a}$.)

Solution: $T(n) \in \Theta(f(n))$

(Intuitively: cost is dominated by root.)

Master Theorem

$T(n) = 27T(n/3) + \Theta(n^3/\log n)$

Compare $n^{\log_3 27}$ vs. n^3.

Since $3 = \log_3 27$ use **<u>Case 2</u>**

<u>but</u> $n^3/\log n \in$ **not** $\Theta(n^3 \log^k n)$ for $k \geq 0$

Cannot use Master Method.

Median Finding

Median Finding

Median Finding. Given n distinct numbers $a_1, \ldots, a_n$, find i such that

$$|\{ j : a_j < a_i \}| = \lfloor n-1 / 2 \rfloor \text{ and } |\{ j : a_j > a_i \}| = \lceil n-1 / 2 \rceil .$$

22	31	44	7	12	19	20	35	3	40	27

3	7	12	19	20	22	27	31	35	40	44
9	4	5	6	7	1	11	2	8	10	3

Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$

k=4

22	31	44	7	12	19	20	35	3	40	27

3	7	12	19	20	22	27	31	35	40	44
9	4	5	6	7	1	11	2	8	10	3

$$\text{Median}(a_1, \ldots, a_n) = \text{Selection}(a_1, \ldots, a_n, \lfloor n+1 / 2 \rfloor)$$

Partition (from QuickSort)

Algorithm partition(A, start, stop)

Input: An array A, indices start and stop.

Output: Returns an index j and rearranges the elements of A
  such that for all i<j, A[i] ≤ A[j] and
  for all k>j, A[k] ≥ A[j].

pivot ← A[stop]

left ← start

right ← stop - 1

while left ≤ right **do**

  while left ≤ right **and** A[left] ≤ pivot) **do** left ← left + 1

  while (left ≤ right **and** A[right] ≥ pivot) **do** right ← right -1

  if (left < right **)** **then** exchange A[left] ↔ A[right]

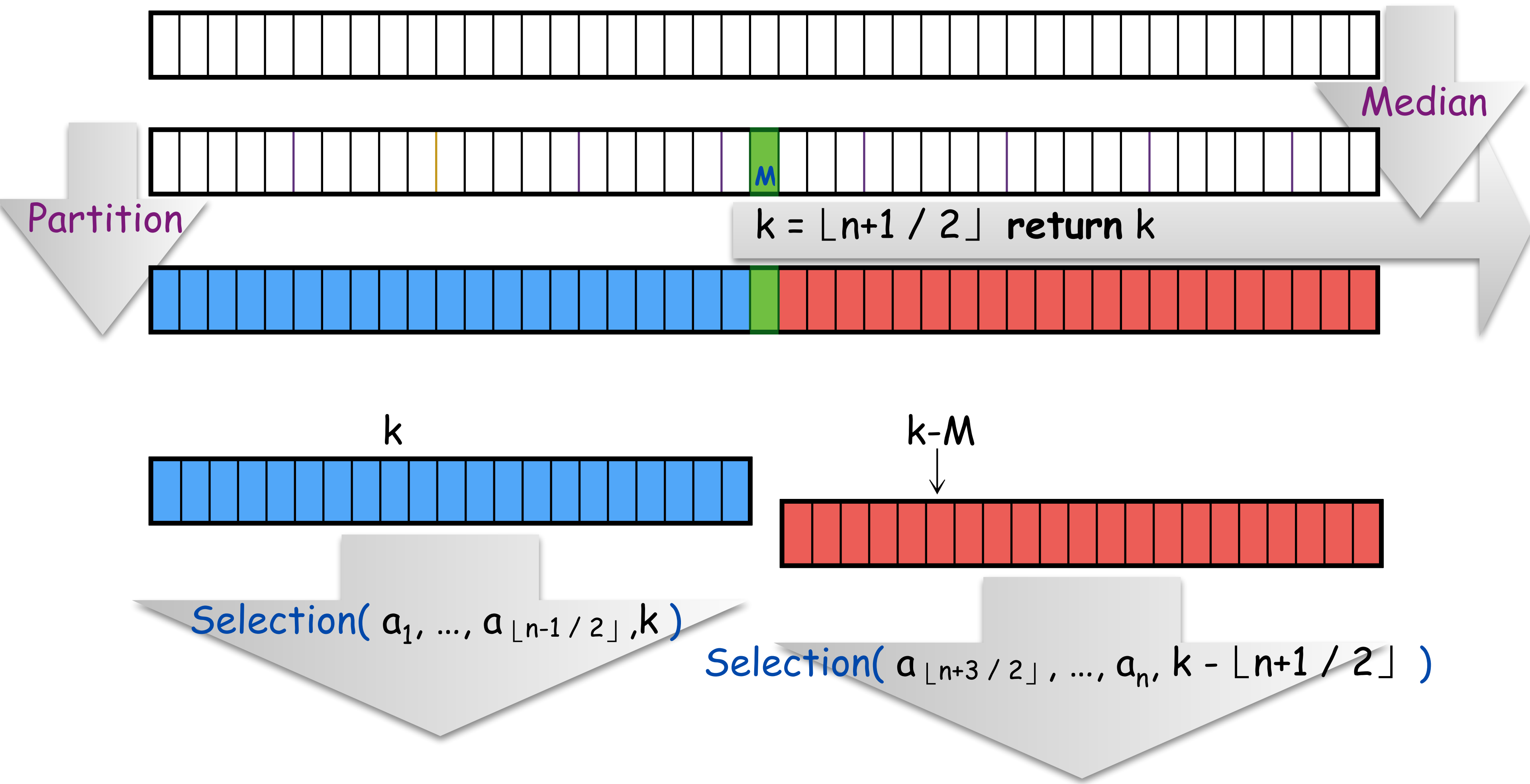
exchange A[stop] ↔ A[left]

return left

Selection from Median

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

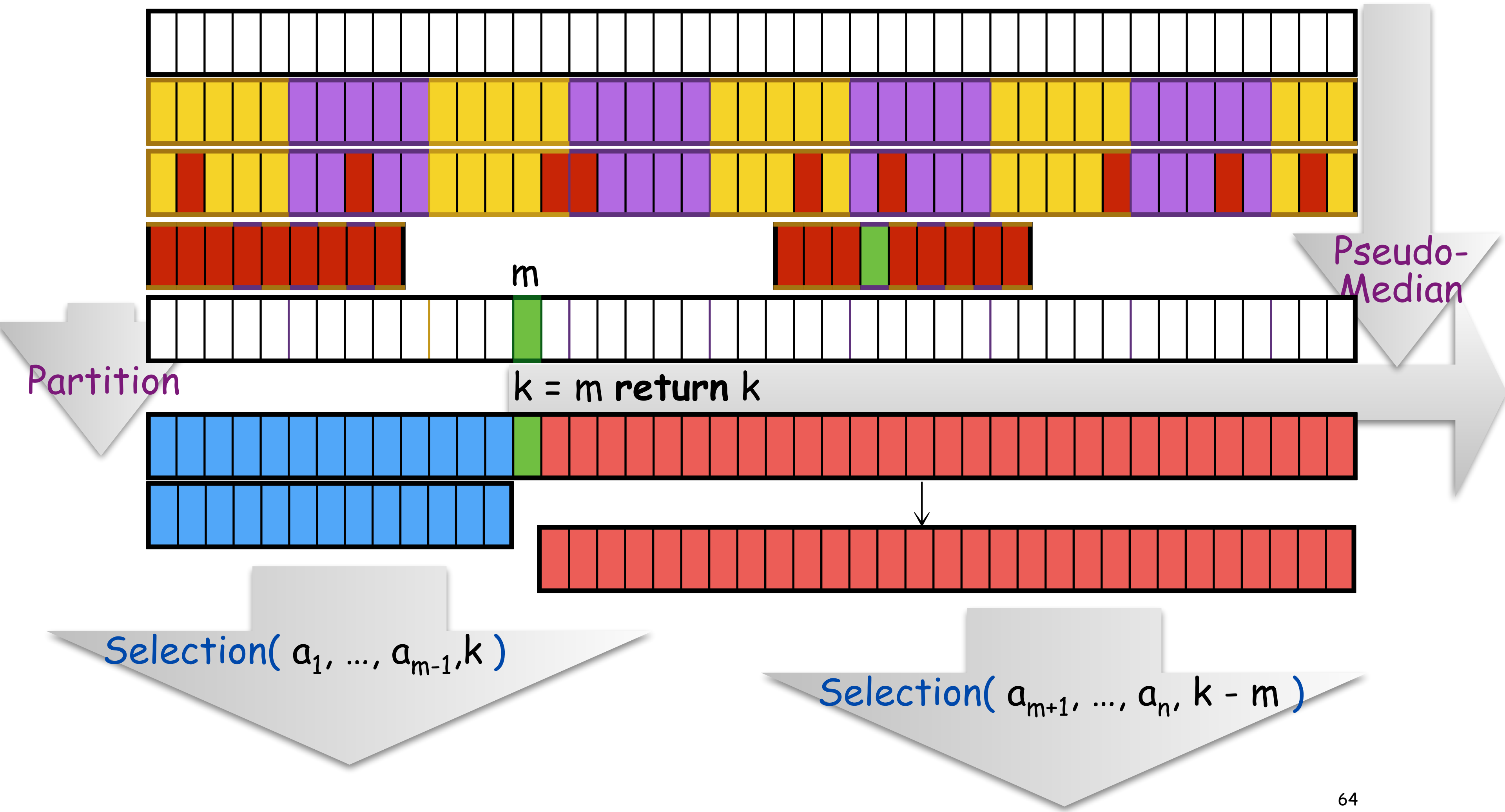
$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$



Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

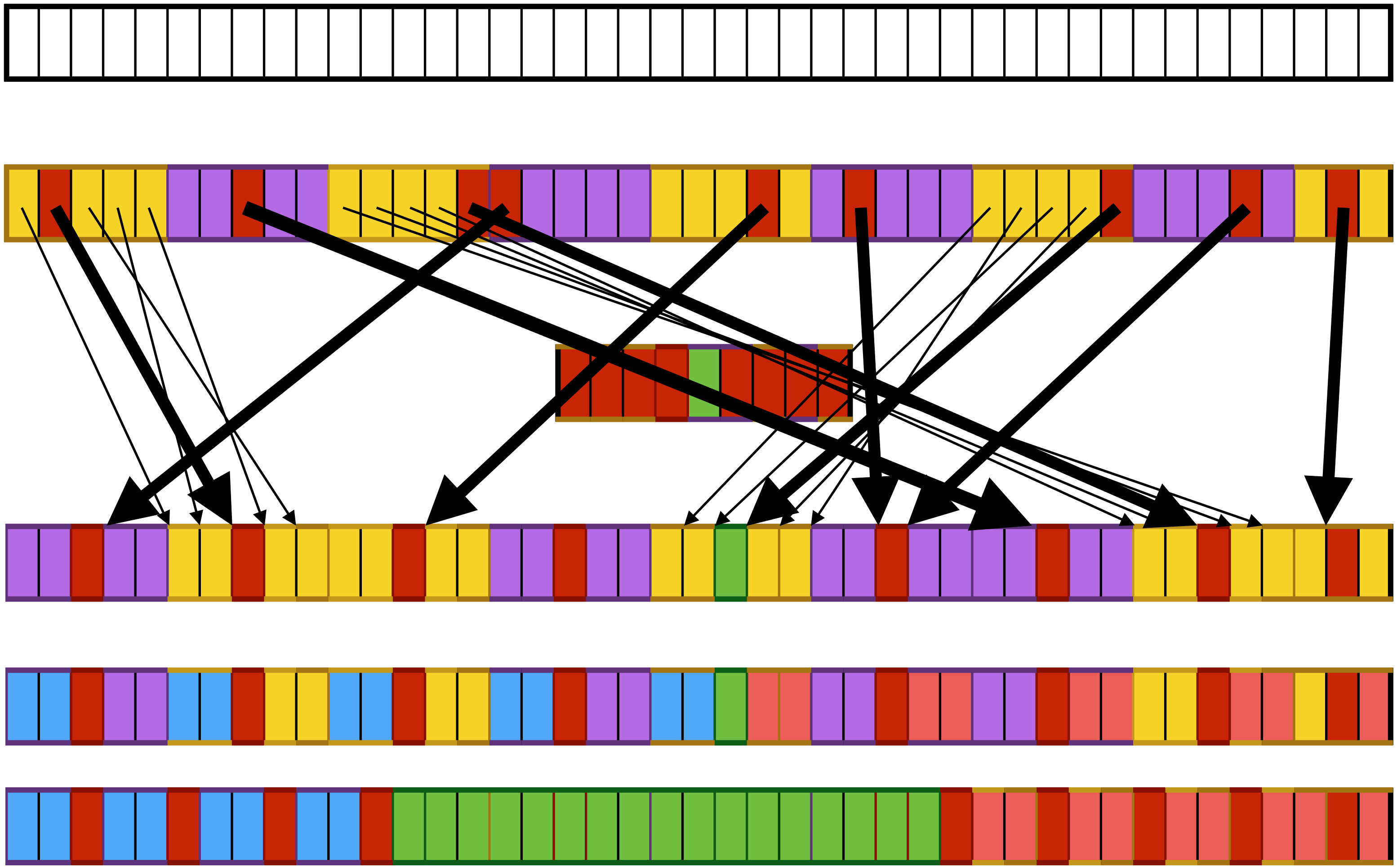
$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$



Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$



3n/10—————————————7n/10

Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$

$$T(n) \le T(\,n/5\,) + T(\,7n/10\,) + \Theta(n)$$

Solution: $T(n) \in \Theta(n)$

Assuming $T(i) \le d\,i$ for $1 \le i \le n$, $\Theta(n) \le cn$

$T(n+1) \le T(n+1\ /5) + T(7(n+1)/10) + c(n+1)$

$\le d(n+1)/5 + 7d(n+1)/10 + c(n+1)$

$= (2d+7d+10c)/10\ (n+1)$

$= (9d+10c)/10\ (n+1)$

$\le d\ (n+1)$ as long as $(9d+10c)/10 \le d$, or equivalently $10c \le d$.

Selection

Selection. Given n distinct numbers $a_1, \ldots, a_n$, and index k, find i such that

$$|\{ j : a_j < a_i \}| = k-1 \text{ and } |\{ j : a_j > a_i \}| = n-k.$$

$$T(n) \le T(\, n/5\,) + T(\, 7n/10\,) + \Theta(n)$$

<u>Solution:</u> $T(n) \in \Theta(n)$

<u>example</u>: d=10c,

Assuming $T(i) \le 10c\, i$ for $1 \le i \le n$, $\Theta(n) \le cn$

$T(n+1) \le T(n+1\,/5) + T(7/10\,(n+1)) + c(n+1)$

$\le 10c/5\,(n+1) + 7 \cdot 10c/10\,(n+1) + c(n+1)$

$= (2c+7c+c)(n+1)$

$= 10c\,(n+1)$