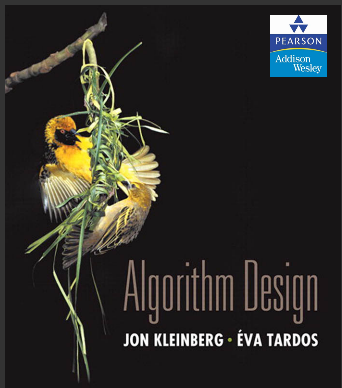


5. Divide And Conquer I

- *mergesort*
- *counting inversions*
- *randomized quicksort*
- *median and selection*
- *closest pair of points*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 3/12/18 9:21 AM

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems (of the same kind).
- Solve (conquer) each subproblem recursively.
- Combine solutions to subproblems into overall solution.

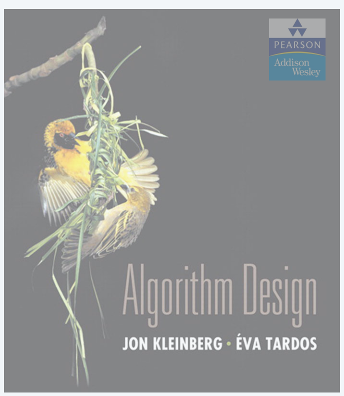
Most common usage.

- Divide problem of size n into two subproblems of size $n/2$. ← $O(n)$ time
- Solve (conquer) two subproblems recursively.
- Combine two solutions into overall solution. ← $O(n)$ time

Consequence.

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

attributed to Julius Caesar



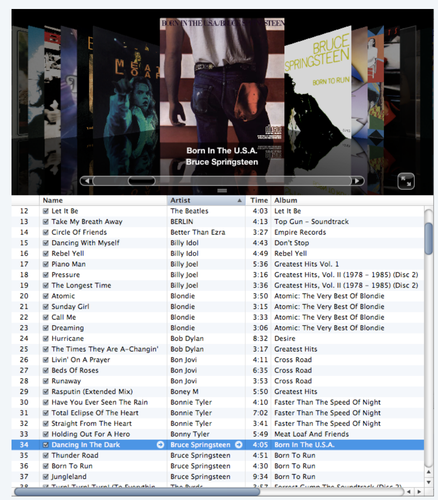
Sections 5.1–5.2

5. Divide and Conquer

- *mergesort*
- *counting inversions*
- *randomized quicksort*
- *median and selection*
- *closest pair of points*

Sorting problem

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



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.

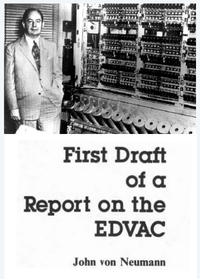
- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

Non-obvious applications.

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

Mergesort

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



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 $a_i \le b_j$, 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).

Mergesort implementation

Input. List L of n elements from a totally ordered universe.
Output. The n elements in ascending order.

MERGE-SORT(L)

IF (list L has one element)
  RETURN L.

Divide the list into two halves A and B.
$A \leftarrow$ MERGE-SORT(A). ← $T(n/2)$
$B \leftarrow$ MERGE-SORT(B). ← $T(n/2)$
$L \leftarrow$ MERGE(A, B). ← $\Theta(n)$
RETURN L.

A useful recurrence relation

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

Recurrence.

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

between $\lfloor n/2 \rfloor$ and $n - 1$ compares

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

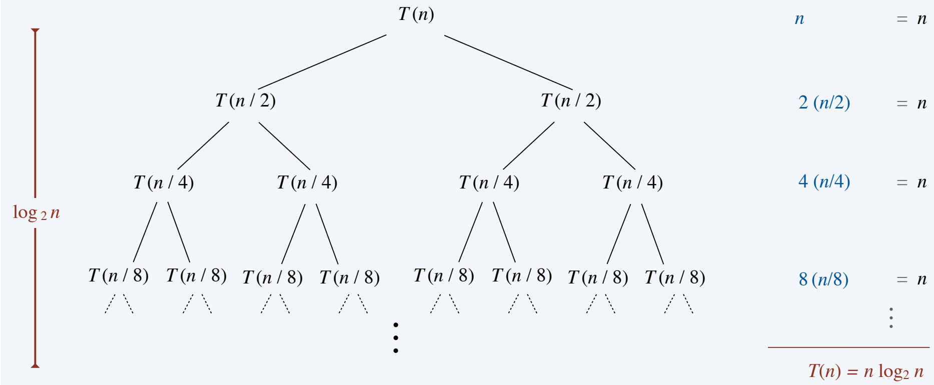
Assorted proofs. We describe several ways to solve this recurrence. Initially we assume n is a power of 2 and replace $\le$ with = in the recurrence.

Divide-and-conquer recurrence: recursion tree

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

assuming n is a power of 2

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



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

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

Pf. [by induction on n]

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

recurrence

$$T(2n) = 2\,T(n) + 2n$$

inductive hypothesis →

$$= 2n \log_2 n + 2n$$

$$= 2n (\log_2 (2n) - 1) + 2n$$

$$= 2n \log_2 (2n). \quad \blacksquare$$

Divide-and-conquer: quiz 1

Which is the exact solution of the following recurrence?

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

no longer assuming n is a power of 2

A. $T(n) = n \lfloor \log_2 n \rfloor$

B. $T(n) = n \lceil \log_2 n \rceil$

C. $T(n) = n \lfloor \log_2 n \rfloor + 2^{\lfloor \log_2 n \rfloor} - 1$

D. $T(n) = n \lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1$

E. Not even Knuth knows.

Analysis of mergesort recurrence

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

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

no longer assuming n is a power of 2

Pf. [by strong induction on n]

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

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

inductive hypothesis →

$$n_2 = \lceil n/2 \rceil \le \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil = 2^{\lceil \log_2 n \rceil} / 2$$

$$\log_2 n_2 \le \lceil \log_2 n \rceil - 1$$

an integer

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

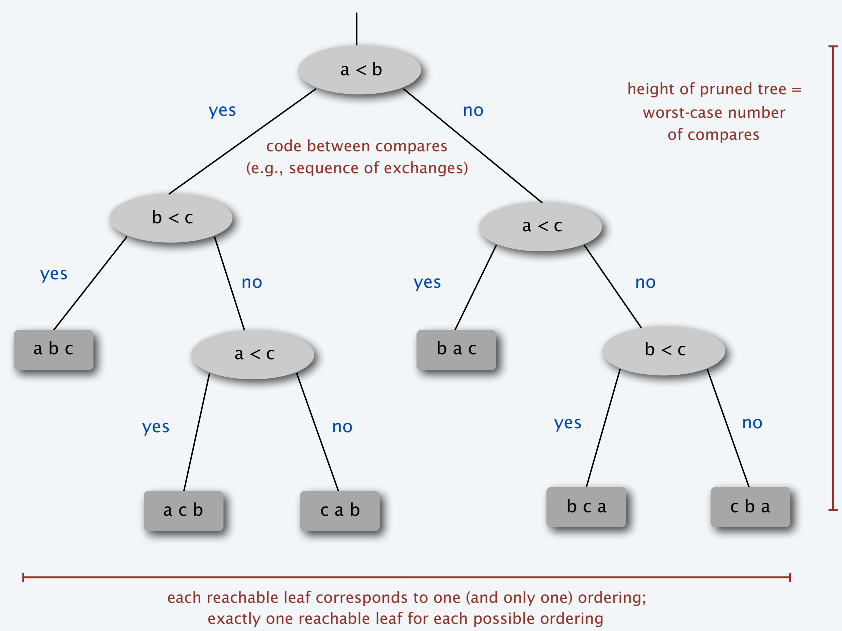
Cost model. Number of compares.

Q. Realistic model?
A1. Yes. Java, Python, C++, …
A2. Yes. Mergesort, insertion sort, quicksort, heapsort, …
A3. No. Bucket sort, radix sorts, …

sort(**, key=None, reverse=False*)

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

Comparison tree (for 3 distinct keys a, b, and c)

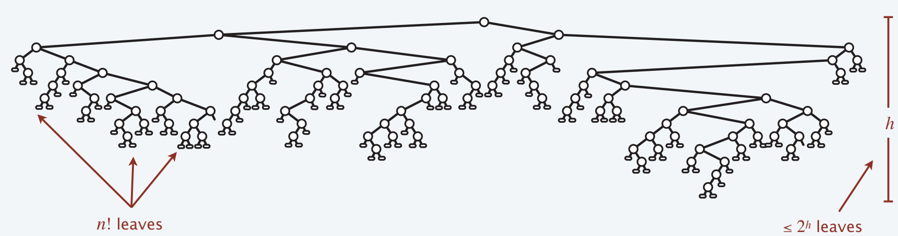


Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\le 2^h$ leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.

$$2^h \geq \text{\# leaves} \geq n!$$
$$\Rightarrow h \geq \log_2(n!)$$
$$\geq n \log_2 n - n / \ln 2 \quad \blacksquare$$

Stirling's formula

Note. Lower bound can be extended to include randomized algorithms.

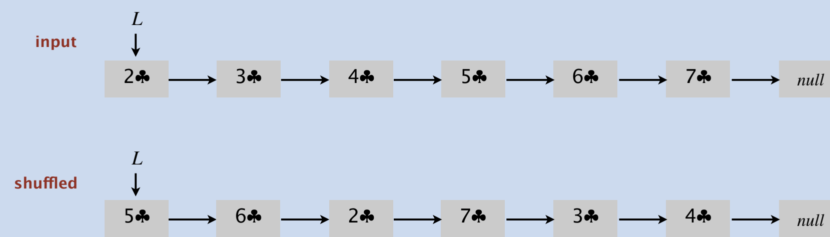
Shuffling a linked list

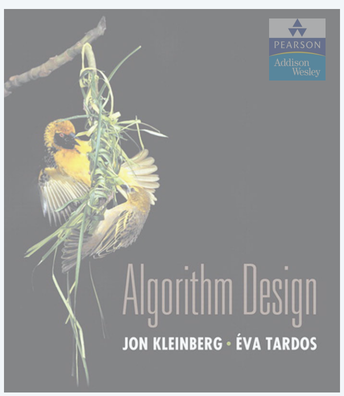
Problem. Given a singly linked list, rearrange its nodes uniformly at random.

Assumption. Access to a perfect random-number generator.

all $n!$ permutations equally likely

Performance. $O(n \log n)$ time, $O(\log n)$ extra space.





Section 5.3

5. Divide and Conquer

- mergesort
- *counting inversions*
- randomized quicksort
- median and selection
- closest pair of points

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 $a_i > a_j$.

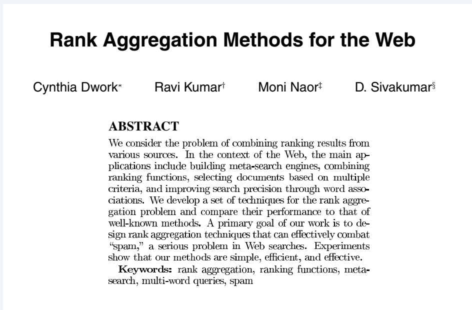
	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

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.

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!

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

20	23	2	11	16

sort A

3	7	10	14	18

sort B

2	11	16	20	23

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

3	7	10	14	18

2	11	16	20	23
5	2	1	0	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.

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

3	7	10	a_i	18

2	11	b_j	20	23
5	2			

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 L in sorted order.

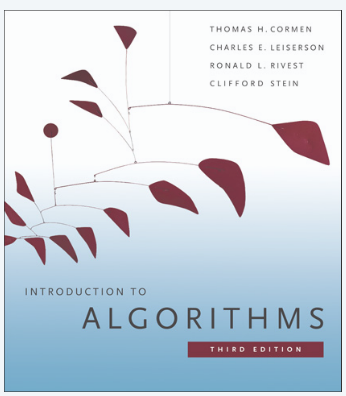
```
SORT-AND-COUNT(L)

IF (list L has one element)
    RETURN (0, L).

Divide the list into two halves A and B.
(r_A, A)  ← SORT-AND-COUNT(A).        ← T(n / 2)
(r_B, B)  ← SORT-AND-COUNT(B).        ← T(n / 2)
(r_AB, L) ← MERGE-AND-COUNT(A, B).    ← Θ(n)

RETURN (r_A + r_B + r_AB, L).
```


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:

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



SECTION 7.1–7.3

5. DIVIDE AND CONQUER

- *mergesort*
- *counting inversions*
- *randomized quicksort*
- *median and selection*
- *closest pair of points*

3-WAY PARTITIONING

Goal. Given an array A and pivot element p, partition array so that:
- Smaller elements in left subarray L.
- Equal elements in middle subarray M.
- Larger elements in right subarray R.

Challenge. $O(n)$ time and $O(1)$ space.

the array A

7	6	12	3	11	8	9	1	4	10	2
	p									

the partitioned array A

3	1	4	2	6	7	12	11	8	9	10
L				M	R					

Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L, M, and R.
- Recursively sort both L and R.

the array A	7	6	12	3	11	8	9	1	4	10	2
		p									
partition A	3	1	4	2	6	7	12	11	8	9	10
sort L	1	2	3	4	6	7	12	11	8	9	10
sort R	1	2	3	4	6	7	8	9	10	11	12
the sorted array A	1	2	3	4	6	7	8	9	10	11	12

Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L, M, and R.
- Recursively sort both L and R.

```
RANDOMIZED-QUICKSORT(A)

IF (array A has zero or one element)
    RETURN.
Pick pivot p ∈ A uniformly at random.
(L, M, R) ← PARTITION-3-WAY(A, p).      ← Θ(n)
RANDOMIZED-QUICKSORT(L).                ← T(i)
RANDOMIZED-QUICKSORT(R).                ← T(n – i – 1)
```

new analysis required
(i is a random variable—depends on p)

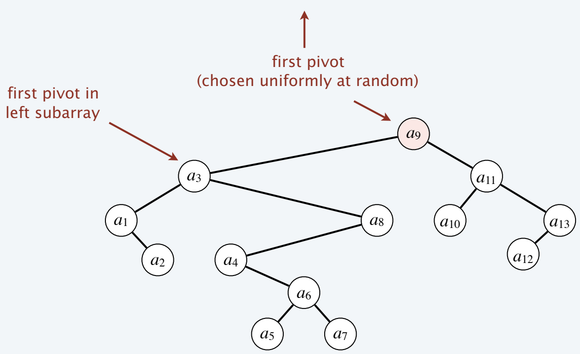
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

the original array of elements A

a_7	a_6	a_{12}	a_3	a_{11}	a_8	a_9	a_1	a_4	a_{10}	a_2	a_{13}	a_5

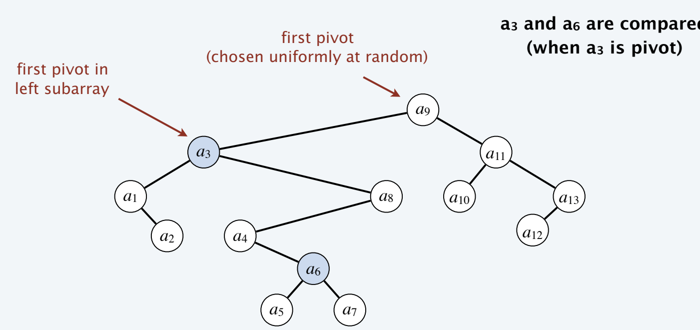


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.

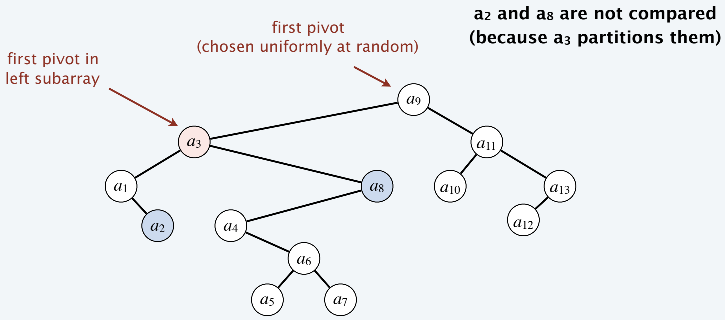


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.



Divide-and-conquer: quiz 2

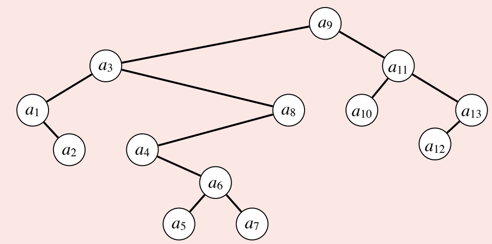
Given an array of $n \ge 8$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_7 and a_8 are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

D. 1



Divide-and-conquer: quiz 3

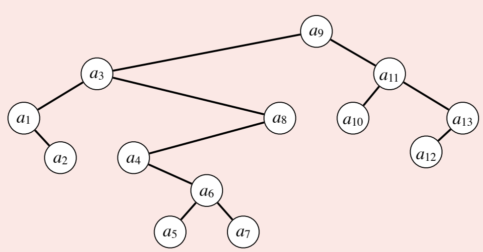
Given an array of $n \ge 2$ distinct elements $a_1 < a_2 < \dots < a_n$, what is the probability that a_1 and a_n are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

D. 1



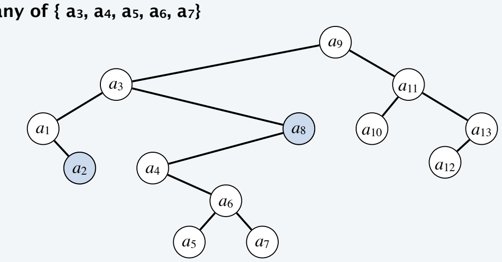
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \dots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- **Pr** [a_i and a_j are compared] $= 2 \,/\, (j - i + 1)$, where $i < j$.

Pr[a_2 and a_8 compared] = 2/7
compared iff either a_2 or a_8 is chosen
as pivot before any of { a_3, a_4, a_5, a_6, a_7}



Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- **Pr** [a_i and a_j are compared] = $2 \,/\, (j - i + 1)$, where $i < j$.
- Expected number of compares = $\displaystyle\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$

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

$$\leq 2n\sum_{j=1}^{n}\frac{1}{j}$$

$$\leq 2n\,(\ln n + 1) \quad ▪$$

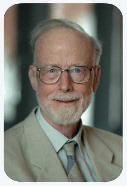
all pairs i and j

harmonic sum

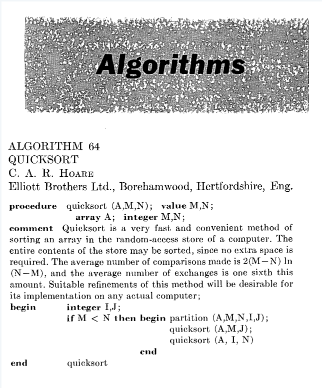
Remark. Number of compares only decreases if equal elements.

Tony Hoare

- Invented quicksort to translate Russian into English.
  [but couldn't explain his algorithm or implement it!]
- Learned Algol 60 (and recursion).
- Implemented quicksort.



Tony Hoare
1980 Turing Award



Algorithms

ALGORITHM 64
QUICKSORT
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

```
procedure   quicksort (A,M,N);   value M,N;
              array A;   integer M,N;
comment  Quicksort is a very fast and convenient method of
sorting an array in the random-access store of a computer. The
entire contents of the store may be sorted, since no extra space is
required. The average number of comparisons made is 2(M−N) ln
(N−M), and the average number of exchanges is one sixth this
amount. Suitable refinements of this method will be desirable for
its implementation on any actual computer;
begin       integer I,J;
            if M < N then begin partition (A,M,N,I,J);
                                quicksort (A,M,J);
                                quicksort (A, I, N)
                          end
end         quicksort
```

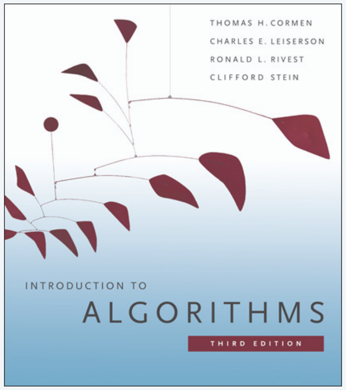
Communications of the ACM (July 1961)

Nuts and bolts

Problem. A disorganized carpenter has a mixed pile of n nuts and n bolts.

- The goal is to find the corresponding pairs of nuts and bolts.
- Each nut fits exactly one bolt and each bolt fits exactly one nut.
- By fitting a nut and a bolt together, the carpenter can see which one is bigger (but cannot directly compare either two nuts or two bolts).

Brute-force solution. Compare each bolt to each nut—$\Theta(n^2)$ compares.
Challenge. Design an algorithm that makes $O(n \log n)$ compares.



Section 9.3

5. Divide and Conquer

- mergesort
- counting inversions
- randomized quicksort
- *median and selection*
- closest pair of points

Median and selection problems

Selection. Given n elements from a totally ordered universe, find k^{th} smallest.

- Minimum: $k = 1$; maximum: $k = n$.
- Median: $k = \lfloor (n+1) / 2 \rfloor$.
- $O(n)$ compares for min or max.
- $O(n \log n)$ compares by sorting.
- $O(n \log k)$ compares with a binary heap. ← max heap with k smallest

Applications. Order statistics; find the "top k"; bottleneck paths, …

Q. Can we do it with $O(n)$ compares?

A. Yes! Selection is easier than sorting.

Randomized quickselect

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L, M, and R.
- Recur in one subarray—the one containing the k^{th} smallest element.

QUICK-SELECT(A, k)

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow$ PARTITION-3-WAY(A, p). ← $\Theta(n)$

IF $(k \le |L|)$ RETURN QUICK-SELECT(L, k). ← $T(i)$

ELSE IF $(k > |L| + |M|)$ RETURN QUICK-SELECT$(R, k - |L| - |M|)$ ← $T(n - i - 1)$

ELSE IF $(k = |L|)$ RETURN p.

Randomized quickselect analysis

Intuition. Split candy bar uniformly $\Rightarrow$ expected size of larger piece is ¾.

$$T(n) \le T(3n/4) + n \quad \Rightarrow \quad T(n) \le 4n$$

not rigorous: can't assume $E[T(i)] \le T(E[i])$

Def. $T(n, k)$ = expected # compares to select k^{th} smallest in array of length $\le n$.

Def. $T(n) = \max_k T(n, k)$.

Proposition. $T(n) \le 4n$.

Pf. [by strong induction on n]

- Assume true for $1, 2, \ldots, n-1$.
- $T(n)$ satisfies the following recurrence:

can assume we always recur of larger of two subarrays since $T(n)$ is monotone non-decreasing

$$\begin{aligned} T(n) &\le n + 1/n \,[\, 2T(n/2) + \ldots + 2T(n-3) + 2T(n-2) + 2T(n-1) \,] \\ &\le n + 1/n \,[\, 8(n/2) + \ldots + 8(n-3) + 8(n-2) + 8(n-1) \,] \\ &\le n + 1/n\,(3n^2) \\ &= 4n. \quad \blacksquare \end{aligned}$$

inductive hypothesis

tiny cheat: sum should start at $T(\lfloor n/2 \rfloor)$

Selection in worst-case linear time

Goal. Find pivot element p that divides list of n elements into two pieces so that each piece is guaranteed to have $\le 7/10\, n$ elements.

Q. How to find approximate median in linear time?

A. Recursively compute median of sample of $\le 2/10\, n$ elements.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(7/10\, n) + T(2/10\, n) + \Theta(n) & \text{otherwise} \end{cases}$$

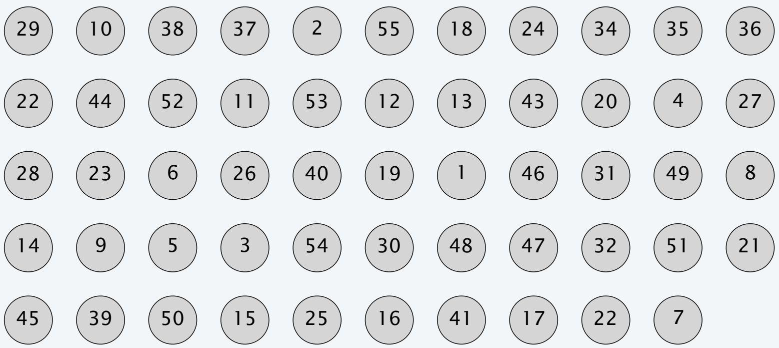
two subproblems of different sizes!

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

we'll need to show this

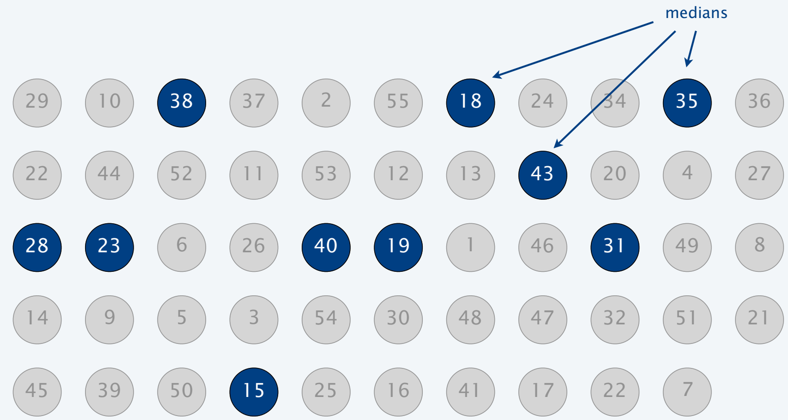
Choosing the pivot element

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).



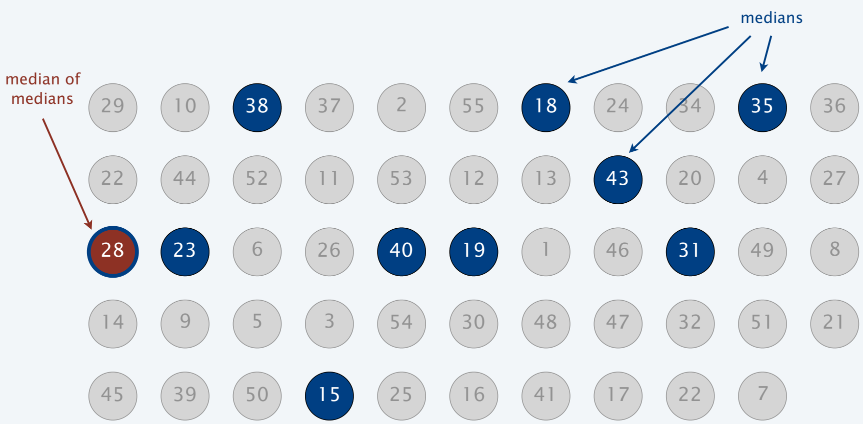
Choosing the pivot element

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).



Choosing the pivot element

- Divide n elements into $\lfloor n / 5 \rfloor$ groups of 5 elements each (plus extra).
- Find median of each group (except extra).
- Find median of $\lfloor n / 5 \rfloor$ medians recursively.
- Use median-of-medians as pivot element.

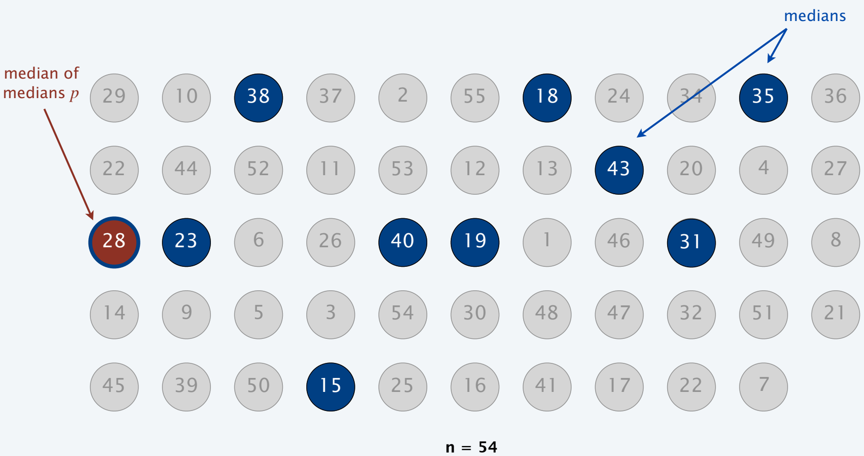


Median-of-medians selection algorithm

```
MOM-SELECT(A, k)

n ← |A|.
IF (n < 50)
   RETURN kth smallest of element of A via mergesort.

Group A into ⌊n / 5⌋ groups of 5 elements each (ignore leftovers).
B ← median of each group of 5.
p ← MOM-SELECT(B, ⌊n / 10⌋)          ←—— median of medians

(L, M, R) ← PARTITION-3-WAY(A, p).
IF       (k ≤ |L|)          RETURN MOM-SELECT(L, k).
ELSE IF  (k > |L| + |M|)    RETURN MOM-SELECT(R, k – |L| – |M|)
ELSE                        RETURN p.
```

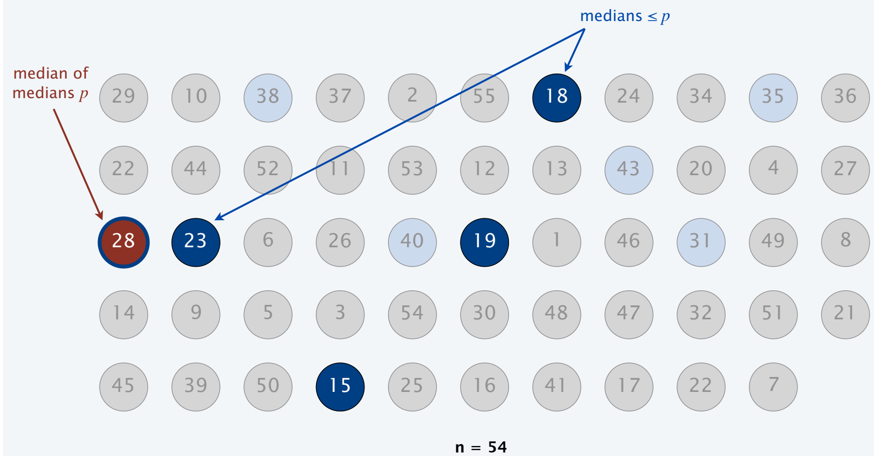
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≤ p.



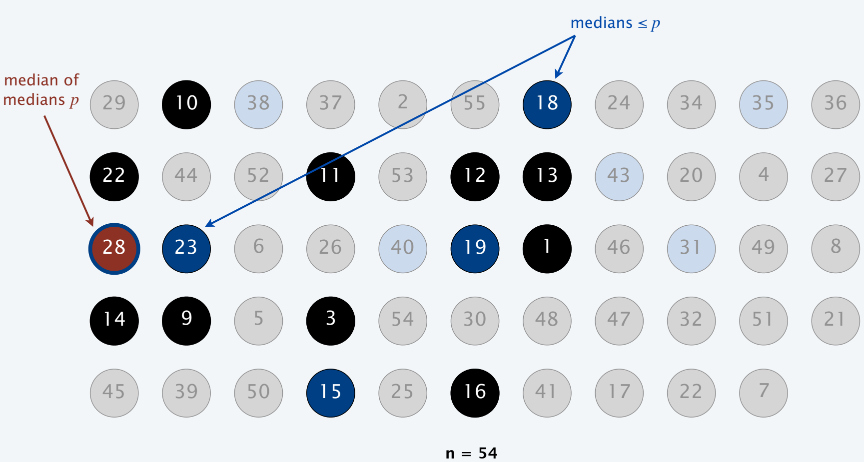
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≤ p.
- At least $\lfloor \lfloor n / 5 \rfloor / 2 \rfloor = \lfloor n / 10 \rfloor$ medians ≤ p.



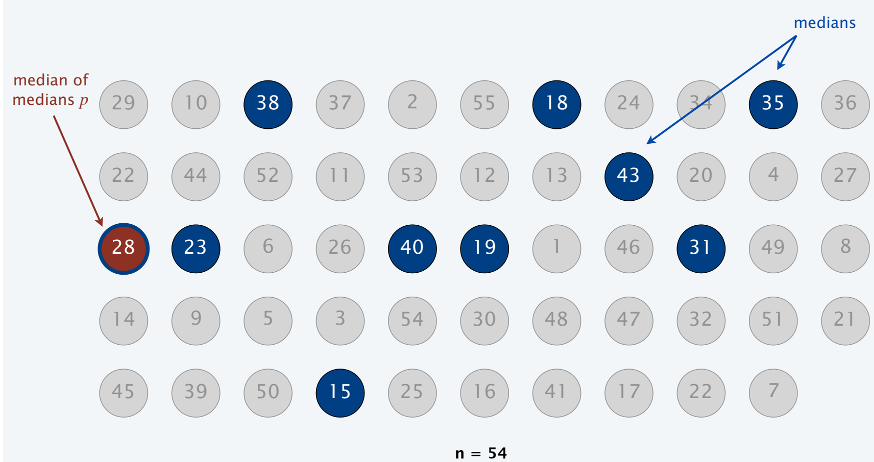
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≤ p.
- At least $\lfloor \lfloor n / 5 \rfloor / 2 \rfloor = \lfloor n / 10 \rfloor$ medians ≤ p.
- At least $3 \lfloor n / 10 \rfloor$ elements ≤ p.



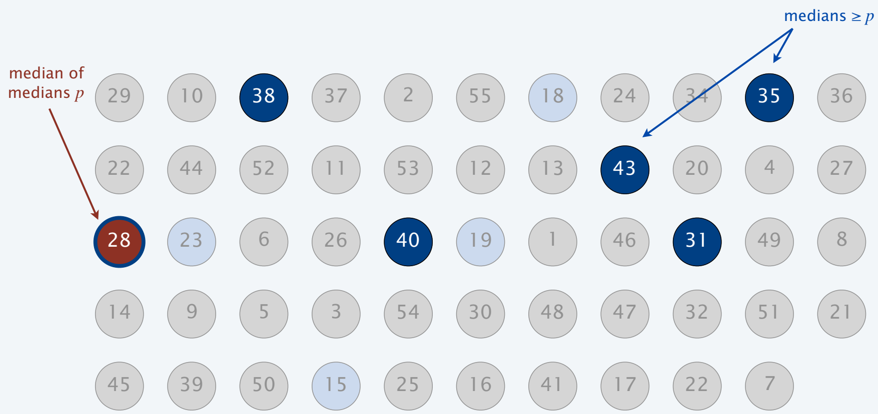
Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≥ p.



Analysis of median-of-medians selection algorithm

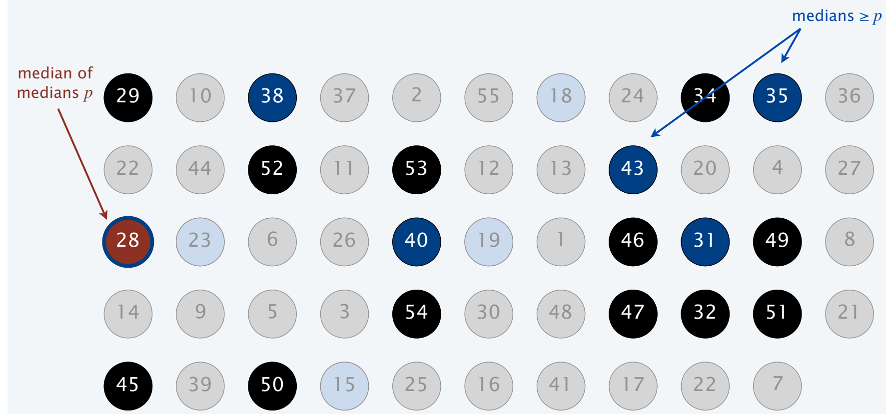
- At least half of 5-element medians ≥ p.
- At least $\lfloor\lfloor n/5\rfloor/2\rfloor = \lfloor n/10\rfloor$ medians ≥ p.



n = 54

Analysis of median-of-medians selection algorithm

- At least half of 5-element medians ≥ p.
- At least $\lfloor\lfloor n/5\rfloor/2\rfloor = \lfloor n/10\rfloor$ medians ≥ p.
- At least $3\lfloor n/10\rfloor$ elements ≥ p.



n = 54

Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with $\lfloor n/5\rfloor$ elements to compute MOM p.
- At least $3\lfloor n/10\rfloor$ elements ≤ p.
- At least $3\lfloor n/10\rfloor$ elements ≥ p.
- Select called recursively with at most $n - 3\lfloor n/10\rfloor$ elements.

Def. $C(n)$ = max # compares on any array of n elements.

$$C(n) \le C(\lfloor n/5\rfloor) + C(n - 3\lfloor n/10\rfloor) + \tfrac{11}{5}n$$

median of medians — recursive select — computing median of 5 (≤ 6 compares per group); partitioning (≤ n compares)

Intuition.

- $C(n)$ is going to be at least linear in n ⇒ $C(n)$ is super-additive.
- Ignoring floors, this implies that $C(n) \le C(n/5 + n - 3n/10) + 11/5\, n$
  $= C(9n/10) + 11/5\, n$
  $\Rightarrow C(n) \le 22n.$

Median-of-medians selection algorithm recurrence

Median-of-medians selection algorithm recurrence.

- Select called recursively with $\lfloor n/5\rfloor$ elements to compute MOM p.
- At least $3\lfloor n/10\rfloor$ elements ≤ p.
- At least $3\lfloor n/10\rfloor$ elements ≥ p.
- Select called recursively with at most $n - 3\lfloor n/10\rfloor$ elements.

Def. $C(n)$ = max # compares on any array of n elements.

$$C(n) \le C(\lfloor n/5\rfloor) + C(n - 3\lfloor n/10\rfloor) + \tfrac{11}{5}n$$

median of medians — recursive select — computing median of 5 (≤ 6 compares per group); partitioning (≤ n compares)

Now, let's solve given recurrence.

- Assume n is both a power of 5 and a power of 10 ?
- Prove that $C(n)$ is monotone non-decreasing.

Divide-and-conquer: quiz 4

Consider the following recurrence

$$C(n) = \begin{cases} 0 & \text{if } n \le 1 \\ C(\lfloor n/5 \rfloor) + C(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

Is $C(n)$ monotone non-decreasing?

A. Yes, obviously.

B. Yes, but proof is tedious.

C. Yes, but proof is hard.

D. No.

Median-of-medians selection algorithm recurrence

Analysis of selection algorithm recurrence.

- $T(n)$ = max # compares on any array of $\le n$ elements.
- $T(n)$ is monotone non-decreasing, but $C(n)$ is not!

$$T(n) \le \begin{cases} 6n & \text{if } n < 50 \\ \max\{\, T(n-1),\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + \frac{11}{5}n) \,\} & \text{if } n \ge 50 \end{cases}$$

Claim. $T(n) \le 44n$.

Pf. [by strong induction]

- Base case: $T(n) \le 6n$ for $n < 50$ (mergesort).
- Inductive hypothesis: assume true for $1, 2, \ldots, n-1$.
- Induction step: for $n \ge 50$, we have either $T(n) \le T(n-1) \le 44n$ or

$$T(n) \le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5\, n$$

inductive hypothesis → $\le 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + 11/5\, n$

$\le 44(n/5) + 44n - 44(n/4) + 11/5\, n$ ← for $n \ge 50$, $3\lfloor n/10 \rfloor \ge n/4$

$= 44n$. ▪

Divide-and-conquer: quiz 5

Suppose that we divide n elements into $\lfloor n/r \rfloor$ groups of r elements each, and use the median-of-medians of these $\lfloor n/r \rfloor$ groups as the pivot. For which r is the worst-case running time of select $O(n)$?

A. $r = 3$

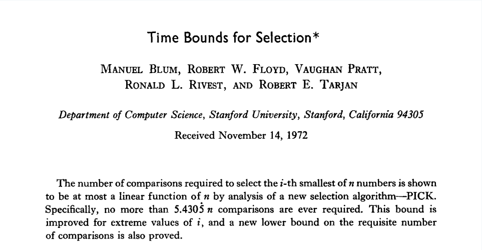
B. $r = 7$

C. Both A and B.

D. Neither A nor B.

Linear-time selection retrospective

Proposition. [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based selection algorithm whose worst-case running time is $O(n)$.



Time Bounds for Selection*

Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan

Department of Computer Science, Stanford University, Stanford, California 94305

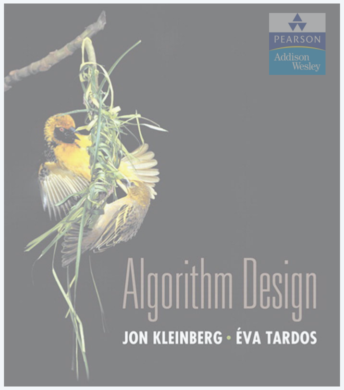
Received November 14, 1972

The number of comparisons required to select the i-th smallest of n numbers is shown to be at most a linear function of n by analysis of a new selection algorithm—PICK. Specifically, no more than $5.4305\,n$ comparisons are ever required. This bound is improved for extreme values of i, and a new lower bound on the requisite number of comparisons is also proved.

Theory.

- Optimized version of BFPRT: $\le 5.4305\,n$ compares.
- Upper bound: [Dor–Zwick 1995] $\le 2.95\,n$ compares.
- Lower bound: [Dor–Zwick 1999] $\ge (2 + 2^{-80})\,n$ compares.

Practice. Constants too large to be useful.



Section 5.4

5. Divide and Conquer

- *mergesort*
- *counting inversions*
- *randomized quicksort*
- *median and selection*
- ***closest pair of points***

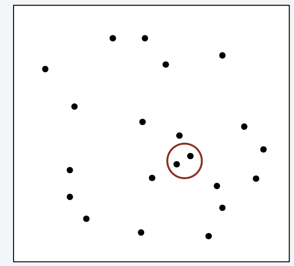
Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the 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



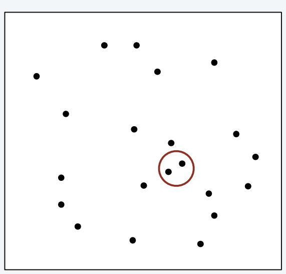
Closest pair of points

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

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

1D version. Easy $O(n \log n)$ algorithm if points are on a line.

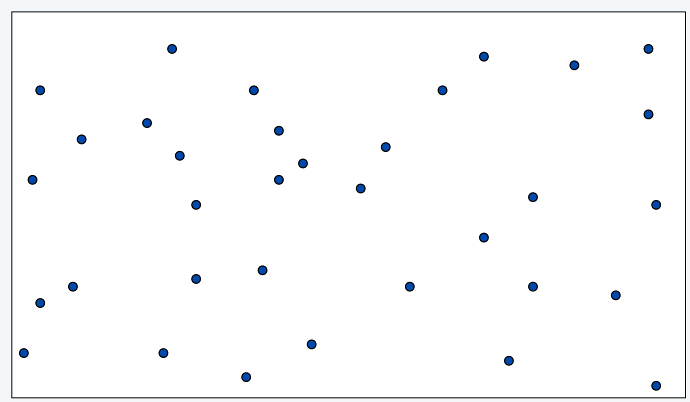
Non-degeneracy assumption. No two points have the same x-coordinate.



Closest pair of points: first attempt

Sorting solution.

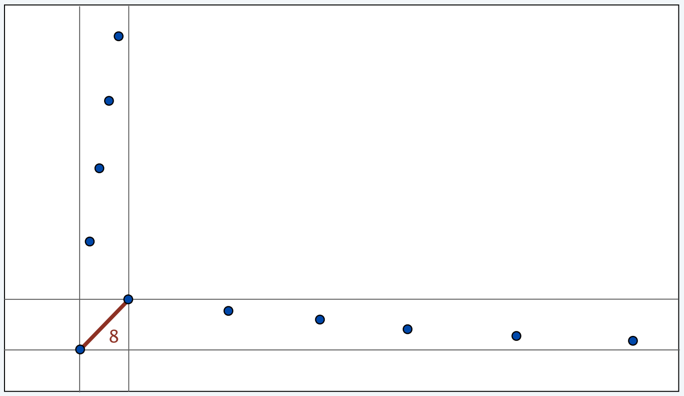
- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest pair of points: first attempt

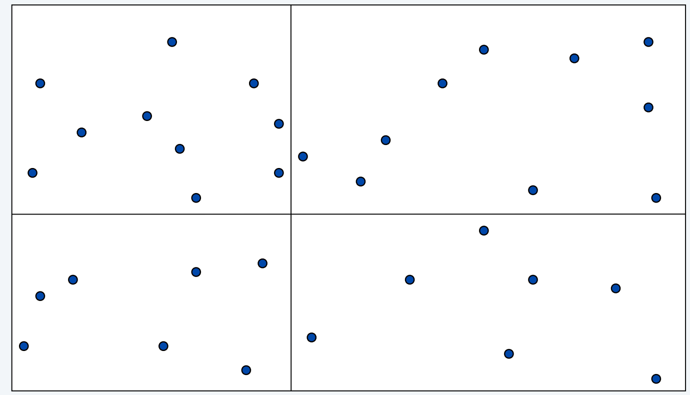
Sorting solution.

- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest pair of points: second attempt

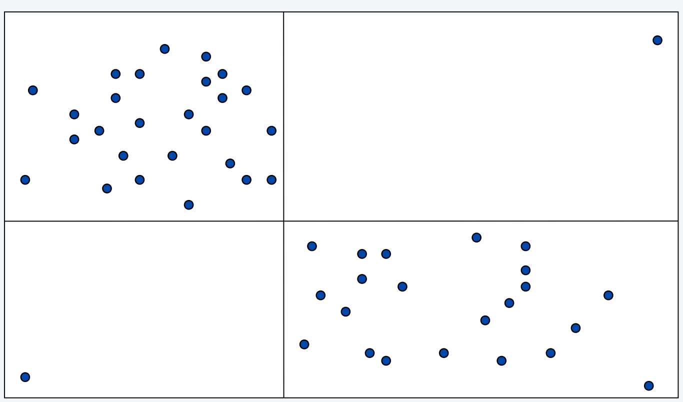
Divide. Subdivide region into 4 quadrants.



Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

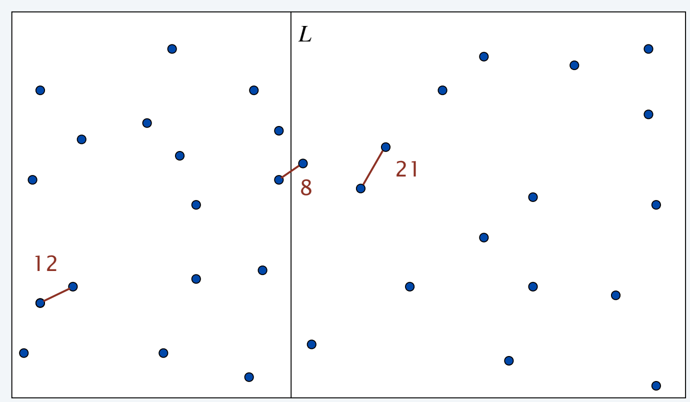
Obstacle. Impossible to ensure $n / 4$ points in each piece.



Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line L so that $n / 2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

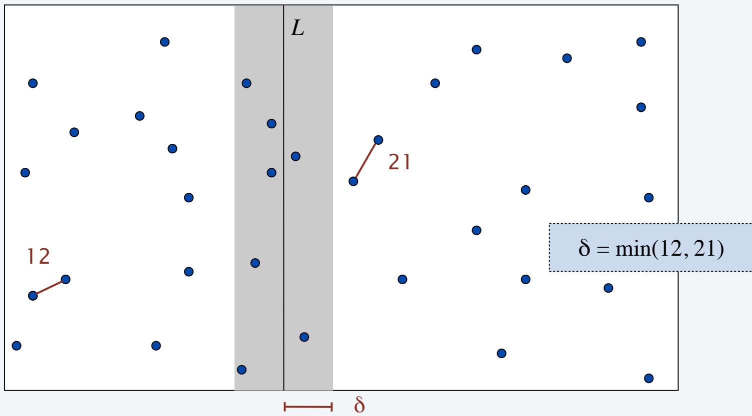
seems like $\Theta(n^2)$



How to find closest pair with one point in each side?

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

- Observation: suffices to consider only those points within δ of line L.

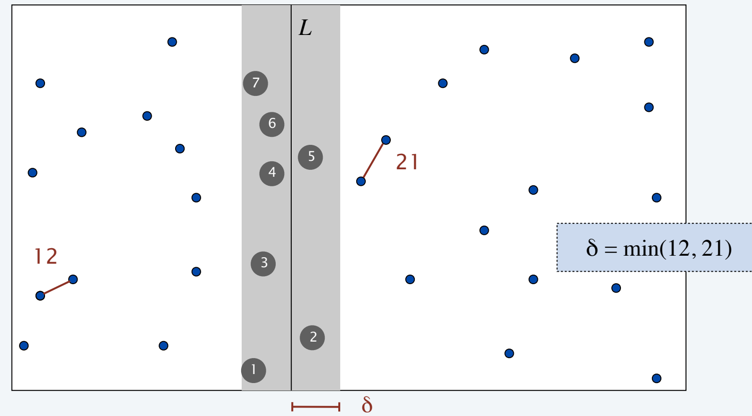


How to find closest pair with one point in each side?

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

- Observation: suffices to consider only those points within δ of line L.
- Sort points in 2δ-strip by their y-coordinate.
- Check distances of only those points within 7 positions in sorted list!

why?



How to find closest pair with one point in each side?

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

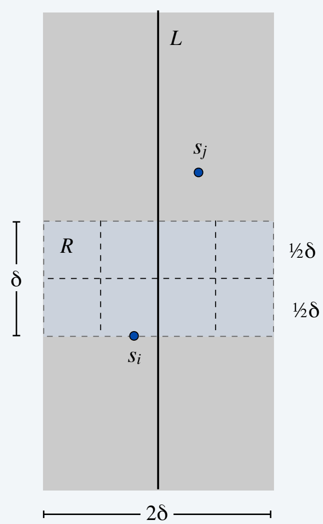
Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ.

Pf.

- Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
- Distance between s_i and any point s_j above R is ≥ δ.
- Subdivide R into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in R. ▪

diameter is $\delta / \sqrt{2} < \delta$

constant can be improved with more refined geometric packing argument



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR($p_1, p_2, \ldots, p_n$)

Compute vertical line L such that half the points are on each side of the line.	$O(n)$
$\delta_1 \leftarrow$ CLOSEST-PAIR(points in left half).	$T(n / 2)$
$\delta_2 \leftarrow$ CLOSEST-PAIR(points in right half).	$T(n / 2)$
$\delta \leftarrow \min \{ \delta_1, \delta_2 \}$.	
Delete all points further than δ from line L.	$O(n)$
Sort remaining points by y-coordinate.	$O(n \log n)$
Scan points in y-order and compare distance between each point and next 7 neighbors. If any of these distances is less than δ, update δ.	$O(n)$
RETURN δ.	

Divide-and-conquer: quiz 6

What is the solution to the following recurrence?

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

A. $T(n) = \Theta(n)$.

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

C. $T(n) = \Theta(n \log^2 n)$.

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

Refined version of closest-pair algorithm

Q. How to improve to $O(n \log n)$?

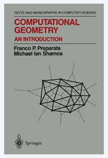
A. Don't sort points in strip from scratch each time.

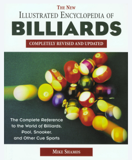
- Each recursive call returns two lists: all points sorted by x-coordinate, and all points sorted by y-coordinate.
- Sort by merging two pre-sorted lists.

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding a closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf.

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





Divide-and-conquer: quiz 7

What is the complexity of the 2D closest pair problem?

A. $\Theta(n)$.

B. $\Theta(n \log^* n)$.

C. $\Theta(n \log \log n)$.

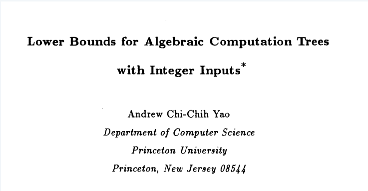
D. $\Theta(n \log n)$.

E. Not even Tarjan knows.

Computational complexity of closest-pair problem

Theorem. [Ben-Or 1983, Yao 1989] In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.

$(x_1 - x_2)^2 + (y_1 - y_2)^2$

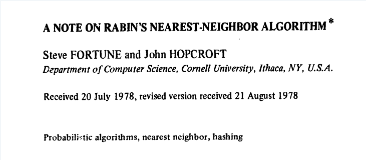


Lower Bounds for Algebraic Computation Trees with Integer Inputs*

Andrew Chi-Chih Yao
Department of Computer Science
Princeton University
Princeton, New Jersey 08544

Theorem. [Rabin 1976] There exists an algorithm to find the closest pair of points in the plane whose expected running time is $O(n)$.

not subject to $\Omega(n \log n)$ lower bound because it uses the floor function



A NOTE ON RABIN'S NEAREST-NEIGHBOR ALGORITHM*

Steve FORTUNE and John HOPCROFT
Department of Computer Science, Cornell University, Ithaca, NY, U.S.A.

Received 20 July 1978, revised version received 21 August 1978

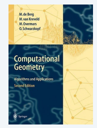
Probabilistic algorithms, nearest neighbor, hashing

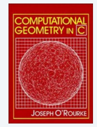
Digression: computational geometry

Ingenious divide-and-conquer algorithms for core geometric problems.

problem	brute	clever
closest pair	$O(n^2)$	$O(n \log n)$
farthest pair	$O(n^2)$	$O(n \log n)$
convex hull	$O(n^2)$	$O(n \log n)$
Delaunay/Voronoi	$O(n^4)$	$O(n \log n)$
Euclidean MST	$O(n^2)$	$O(n \log n)$

running time to solve a 2D problem with n points

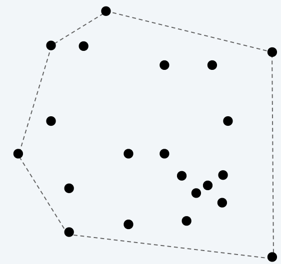




Note. 3D and higher dimensions test limits of our ingenuity.

Convex hull

The convex hull of a set of n points is the smallest perimeter fence enclosing the points.

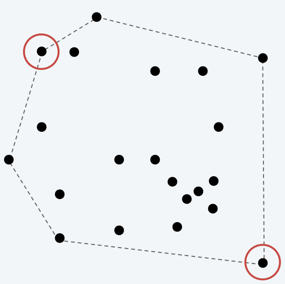


Equivalent definitions.

- Smallest area convex polygon enclosing the points.
- Intersection of all convex set containing all the points.

Farthest pair

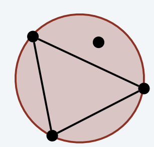
Given n points in the plane, find a pair of points with the largest Euclidean distance between them.



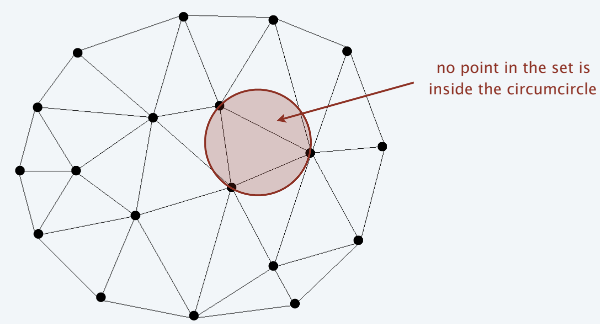
Fact. Points in farthest pair are extreme points on convex hull.

Delaunay triangulation

The Delaunay triangulation is a triangulation of n points in the plane such that no point is inside the circumcircle of any triangle.



point inside circumcircle of 3 points



Delaunay triangulation of 19 points

Some useful properties.

- No edges cross.
- Among all triangulations, it maximizes the minimum angle.
- Contains an edge between each point and its nearest neighbor.

Euclidean MST

Given n points in the plane, find MST connecting them.
[distances between point pairs are Euclidean distances]

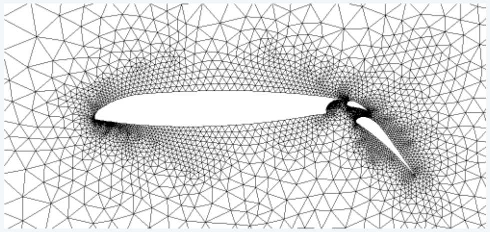
Fact. Euclidean MST is subgraph of Delaunay triangulation.
Implication. Can compute Euclidean MST in $O(n \log n)$ time.

- Compute Delaunay triangulation.
- Compute MST of Delaunay triangulation. ← it's planar ($\leq 3n$ edges)

Computational geometry applications

Applications.

- Robotics.
- VLSI design.
- Data mining.
- Medical imaging.
- Computer vision.
- Scientific computing.
- Finite-element meshing.
- Astronomical simulation.
- Models of physical world.
- Geographic information systems.
- Computer graphics (movies, games, virtual reality).



airflow around an aircraft wing

http://www.ics.uci.edu/~eppstein/geom.html