Algorithm II

5. Divide and Conquer I

WU Xiaokun 吴晓堃

xkun.wu [at] gmail

Divide-and-conquer paradigm

Divide-and-conquer.

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

Divide-and-conquer paradigm

Divide-and-conquer.

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

Most common usage.

- Divide problem of size n into *two* sub-problems of size $n/2$.
- Solve (conquer) two sub-problems recursively.
- Combine two solutions into overall solution.

Divide-and-conquer paradigm

Divide-and-conquer.

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

Most common usage.

- Divide problem of size n into *two* sub-problems of size $n/2$.
- Solve (conquer) two sub-problems recursively.
- Combine two solutions into overall solution.

Benefit. Closest Pair Problem:

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

Mergesort

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.

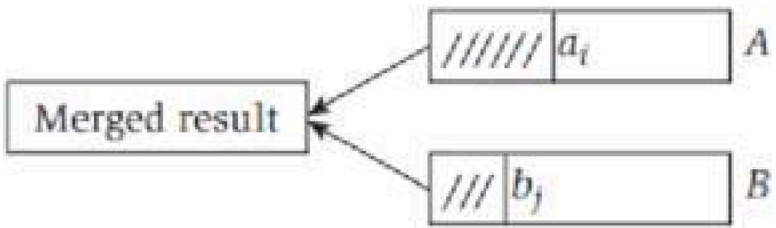
Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	**E**	**R**	**G**	**E**	**S**	**O**	**R**	**T**	**E**	**X**	**A**	**M**	**P**	**L**	**E**
E	**M**	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	**G**	**R**	E	S	O	R	T	E	X	A	M	P	L	E
E	**G**	**M**	**R**	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	**E**	**S**	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	S	**O**	**R**	T	E	X	A	M	P	L	E
E	G	M	R	**E**	**O**	**R**	**S**	T	E	X	A	M	P	L	E
E	**E**	**G**	**M**	**O**	**R**	**R**	**S**	T	E	X	A	M	P	L	E
E	E	G	M	O	R	R	S	**E**	**T**	X	A	M	P	L	E
E	E	G	M	O	R	R	S	E	T	**A**	**X**	M	P	L	E
E	E	G	M	O	R	R	S	**A**	**E**	**T**	**X**	M	P	L	E
E	E	G	M	O	R	R	S	A	E	T	X	**M**	**P**	L	E
E	E	G	M	O	R	R	S	A	E	T	X	M	P	**E**	**L**
E	E	G	M	O	R	R	S	A	E	T	X	**E**	**L**	**M**	**P**
E	E	G	M	O	R	R	S	**A**	**E**	**E**	**L**	**M**	**P**	**T**	**X**
A	**E**	**E**	**E**	**E**	**G**	**L**	**M**	**M**	**O**	**P**	**R**	**R**	**S**	**T**	**X**

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).



Mergesort implementation

Input. List L of n elements from a totally ordered universe.

Output. The n elements in ascending order.

1. IF (list L has one element) RETURN L;
2. Divide the list into two halves A and B;
3. A = MERGE-SORT(A): $T(n/2)$;
4. B = MERGE-SORT(B): $T(n/2)$;
5. L = MERGE(A, B): $\Theta(n)$;
6. RETURN L;

Recurrence relation: divide-by-2

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

Recurrence.

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

- Base case: $T(2) \leq 2c$ is a constant; $cn = O(n)$.

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

Recurrence relation: divide-by-2

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

Recurrence.

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

- Base case: $T(2) \leq 2c$ is a constant; $cn = O(n)$.

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

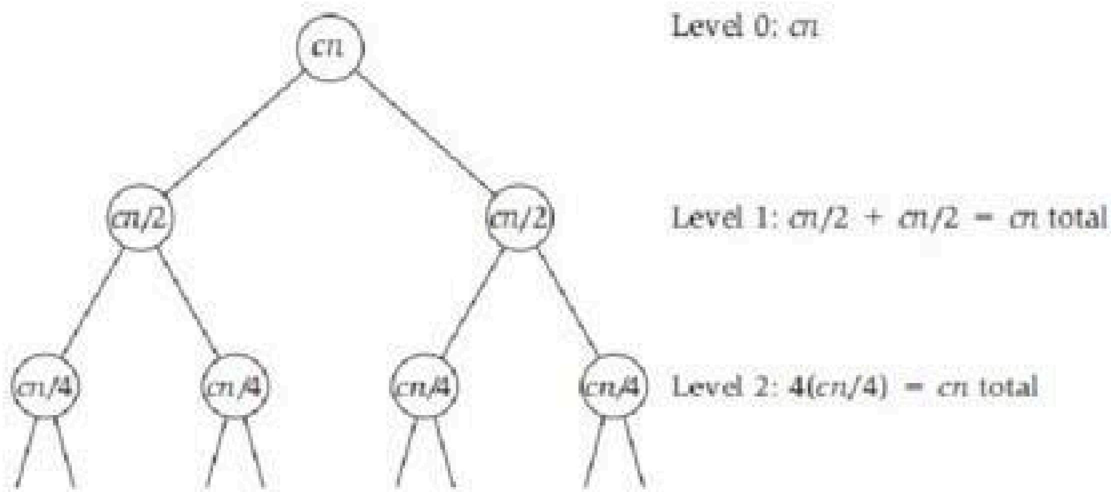
Assorted proofs. several ways to solve this recurrence (following slides).

- Initially, assume n is a power of 2 and replace $\leq$ with $=$.
- Asymptotic bounds are not affected by ignoring floors/ceilings.

Recurrence Pf. 1: unrolling tree

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

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



Pf. [by identifying a pattern]
cn per level, $\log_2 n$ levels.

Recurrence Pf. 2: substitution

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

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

Pf. [by induction on n]

- $T(1) = 0 = cn \log_2 n$.
- assume $T(n/2) = c(n/2) \log_2(n/2)$.

$$\begin{aligned} T(n) &= 2T(n/2) + cn \\ &= 2c(n/2) \log_2(n/2) + cn \\ &= cn[(\log_2 n) - 1] + cn \\ &= (cn \log_2 n) - cn + cn \\ &= cn \log_2 n \end{aligned}$$

Recurrence Pf. 3: partial substitution

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

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

Pf. [guess $T(n) = kn \log_b n$]

- $T(n) = 2k(n/2) \log_b (n/2) + cn$.
  - $b = 2$ looks good for halving.
  - $T(n) = (kn \log_2 n) - kn + cn$.
    - $k = c$ makes the guess right.

Quiz: divide-by-2 recurrence

Which is the exact solution of the following recurrence?

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

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.

Quiz: divide-by-2 recurrence

Which is the exact solution of the following recurrence?

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

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.

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

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

Mergesort: analysis

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

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

Pf. [by strong induction on n]
Induction step: assume true for $1, 2, \ldots, n{-}1$.
Let $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$: note that $n = n_1 + n_2$.

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

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

$$\Downarrow$$

$$(*) \log_2 n_2 \le \lceil \log_2 n \rceil - 1$$

Digression: sorting lower bound

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

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. Number of compares.

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.

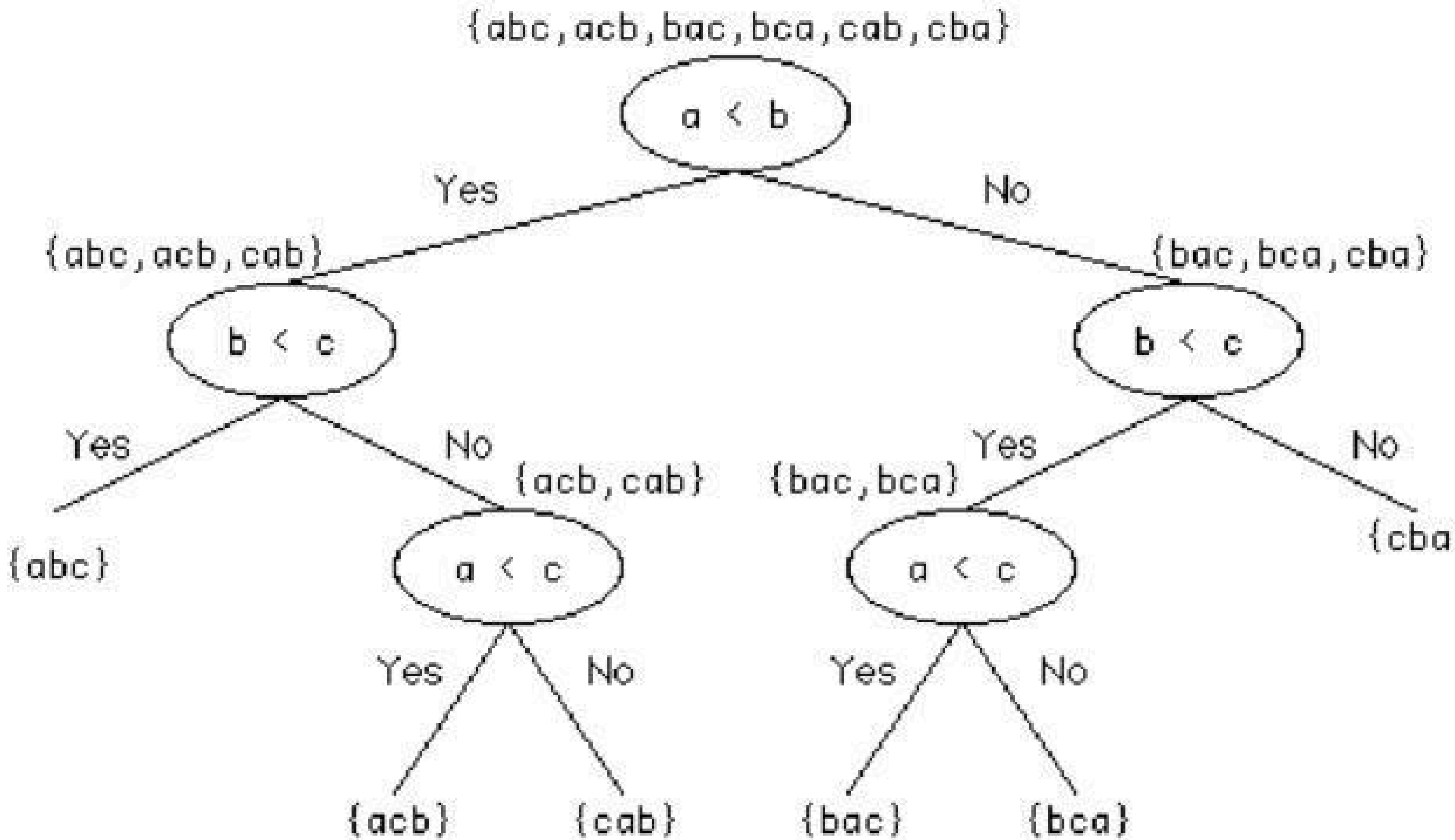
Cost. Number of compares.

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

`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 (3 distinct keys)

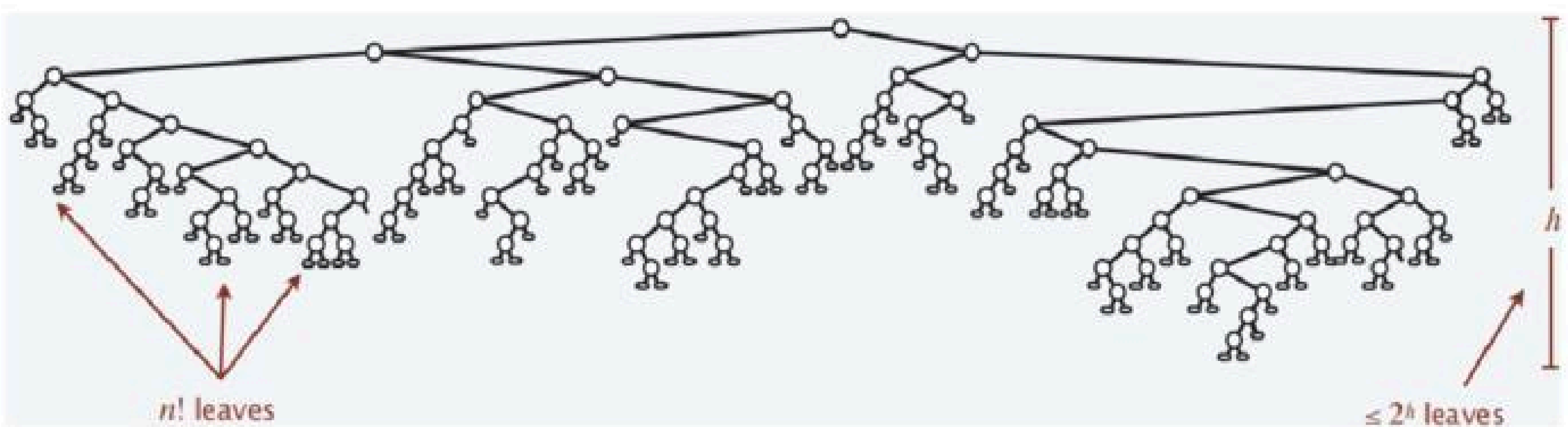


Sorting lower bound

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

Pf. [information theoretic]

- Assume array consists of n distinct values $a_1..a_n$.
- 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.



Sorting lower bound (cont.)

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

Pf. [information theoretic]

- Assume array consists of n distinct values $a_1..a_n$.
- 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.

$$2^h \geq n!$$
$$\Rightarrow h \geq \log_2 n!$$
$$(Stirling\text{'}s) \geq n \log_2 n - n/\ln 2$$

Further Recurrence Relations

Recurrence Relations

More general formulation: recursively solve q sub-problems of size $n/2$ each:

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

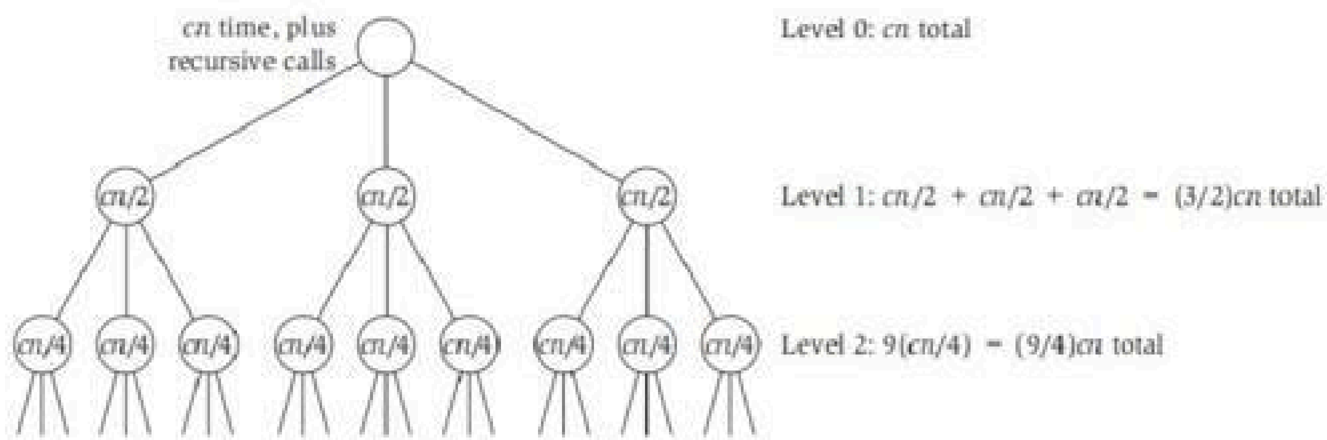
- Base case: $T(2) \leq 2c$ is a constant; $cn = O(n)$.

Recurrence Relations: $q \geq 2$

Proposition. If $T(n)$ satisfies the following recurrence with $q \geq 2$, then $T(n) = O(n^{\log_2 q})$.

$$T(n) \leq qT(n/2) + cn$$

Pf. Geometric sum: $\sum_{j=0}^{\log_2 n - 1} (r)^j = \frac{r^{\log_2 n} - 1}{r - 1}$.

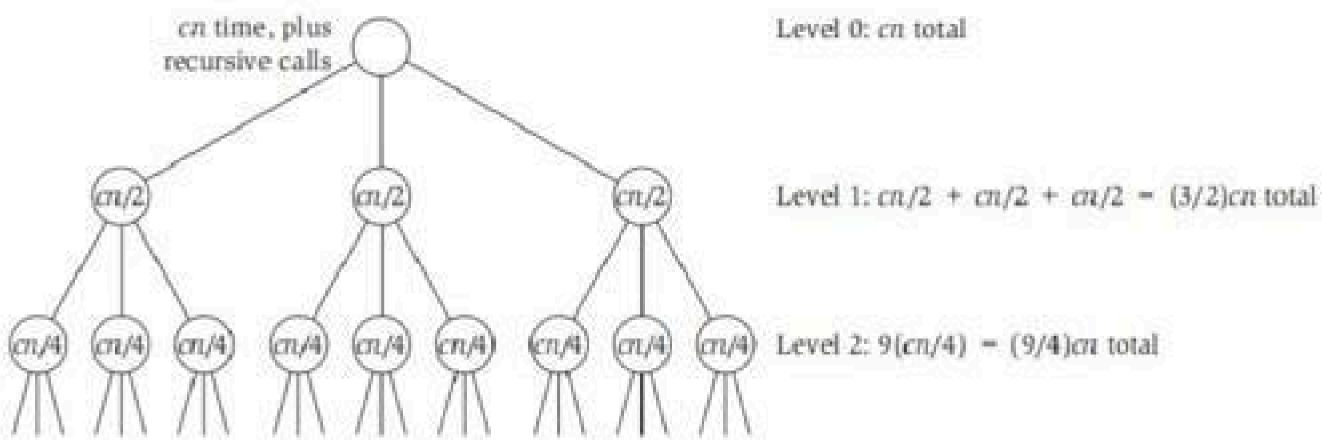


Recurrence Relations: $q \geq 2$

Proposition. If $T(n)$ satisfies the following recurrence with $q \geq 2$, then $T(n) = O(n^{\log_2 q})$.

$$T(n) \leq qT(n/2) + cn$$

Pf. Geometric sum: $\sum_{j=0}^{\log_2 n - 1} (r)^j = \frac{r^{\log_2 n} - 1}{r - 1}$.



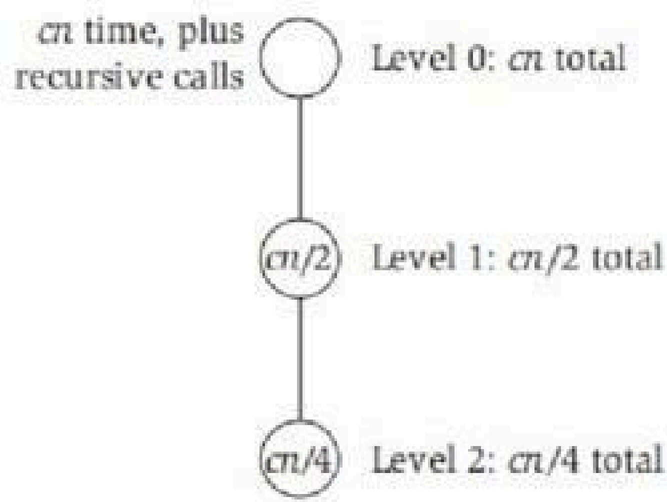
Especially, $O(n^{\log_2 3}) = O(n^{1.59})$, $O(n^{\log_2 4}) = O(n^2)$.

Recurrence Relations: $q = 1$

Proposition. If $T(n)$ satisfies the following recurrence with $q = 1$, then $T(n) = O(n)$.

$$T(n) \leq qT(n/2) + cn$$

Pf. Geometric sum: $\sum_{j=0}^{\log_2 n - 1} \left(\frac{1}{2^j}\right) = 2$.



Recurrence Relations: quadratic time

Special case: spending quadratic time for the initial division and final recombining.

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

- Base case: $T(2) \leq 4c$ is a constant; $cn^2 = O(n^2)$.

Recurrence Relations: quadratic time

Special case: spending quadratic time for the initial division and final recombining.

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

- Base case: $T(2) \leq 4c$ is a constant; $cn^2 = O(n^2)$.

Solution. $T(n)$ is $O(n^2 \log_2 n)$.

Pf. Geometric sum: $\sum_{j=0}^{\log_2 n - 1} (\frac{1}{2^j}) = 2$.

Counting inversions

Music recommendation

Music recommendation. Music site tries to match your preference with others.

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

Music recommendation

Music recommendation. Music site tries to match your preference 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

Music recommendation

Music recommendation. Music site tries to match your preference 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

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

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 $a \in A, b \in B$.
- Return sum of three counts.

1	5	4	8	10	2	6	9	3	7
1	5	4	8	10					
2	6	9	3	7					

Output. $1 + 3 + 13 = 17$.

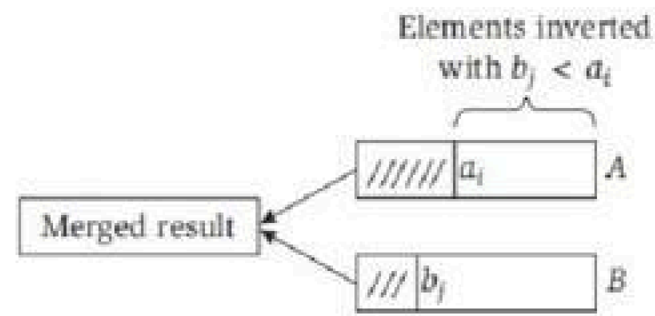
Counting inversions: how to combine?

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

A. Easy if A and B are sorted!

Warmup algorithm.

1. Sort A and B
2. For each element $b_j \in B$:
   1. binary search $\arg\min_i \{b_j < a_i\}$;



7	10	18	3	14	20	23	2	11	16
3	7	10	14	18					
2	11	16	20	23					

Output. $5 + 2 + 1 + 0 + 0 = 8$.

Counting inversions: merge-and-count

Count inversions (a, b) with $a \in A, b \in B$, 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.

7	10	18	3	14	20	23	2	11	16
3	7	10	14	18					
			a_i	18					
2	11	16	20	23					
5	2	b_j	20	23					
2	3	7	10	11					

Counting inversions: algorithm

Input. List L.

Output. Number of inversions in L and L in sorted order.

1. IF (list L has one element) RETURN $(0, L)$;
2. Divide the list into two halves A and B;
3. (r_A, A) = SORT-AND-COUNT(A): $T(n/2)$;
4. (r_B, B) = SORT-AND-COUNT(B): $T(n/2)$;
5. (r_{AB}, L) = MERGE-AND-COUNT(A, B): $\Theta(n)$;
6. RETURN $(r_A + r_B + r_{AB}, L)$;

Counting inversions: 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:

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

Randomized quicksort

3-way partitioning

Goal. Given an array A and pivot element p, partition array so that:

- Smaller elements in left sub-array L.
- Equal elements in middle sub-array M.
- Larger elements in right sub-array R.

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

3-way partitioning

Goal. Given an array A and pivot element p, partition array so that:

- Smaller elements in left sub-array L.
- Equal elements in middle sub-array M.
- Larger elements in right sub-array R.

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

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

Demo: 3-way partitioning

Randomized quicksort: idea

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

	7	**6**	**12**	**3**	**11**	**8**	**9**	**1**	**4**	**10**	**2**
partition	3	1	4	2	**6**	7	12	11	8	9	10
sort L	1	2	3	4	**6**						
sort R					**6**	7	8	9	10	11	12
sorted	1	2	3	4	**6**	7	8	9	10	11	12

Randomized quicksort: algorithm

Input. List A.

Output. A in sorted order.

1. IF (list A has zero or one element) RETURN;
2. Pick pivot $p \in A$ uniformly at random;
3. (L, M, R) = `PARTITION-3-WAY`(A, p): $\Theta(n)$;
4. `RANDOMIZED-QUICKSORT`(L): $T(i)$;
5. `RANDOMIZED-QUICKSORT`(R): $T(n - i - 1)$;

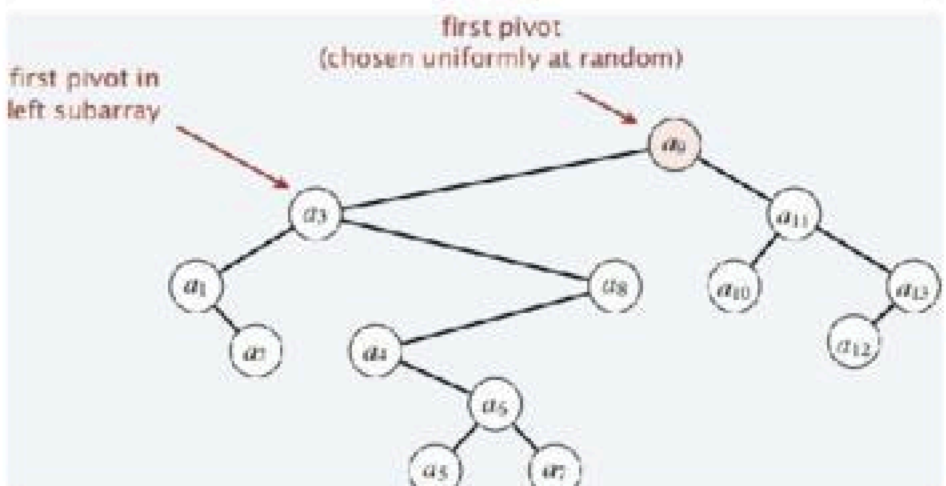
Demo: Randomized quicksort

Randomized quicksort: analysis

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

Pf. Consider BST representation of pivot elements.

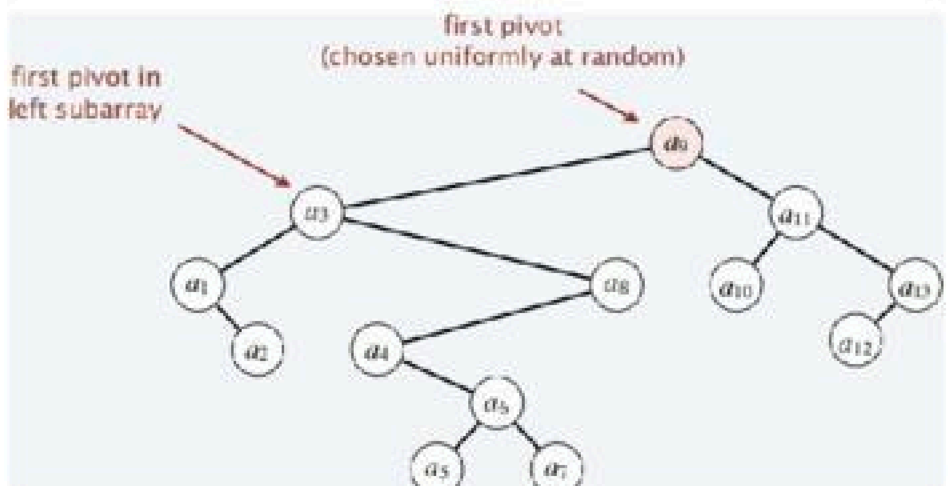
- a_i and a_j are compared once iff one is an ancestor of the other.
  - a_3 and a_6 are compared (when a_3 is pivot)
  - a_2 and a_8 are not compared (because a_3 partitions them)



Quiz: Quicksort 1

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

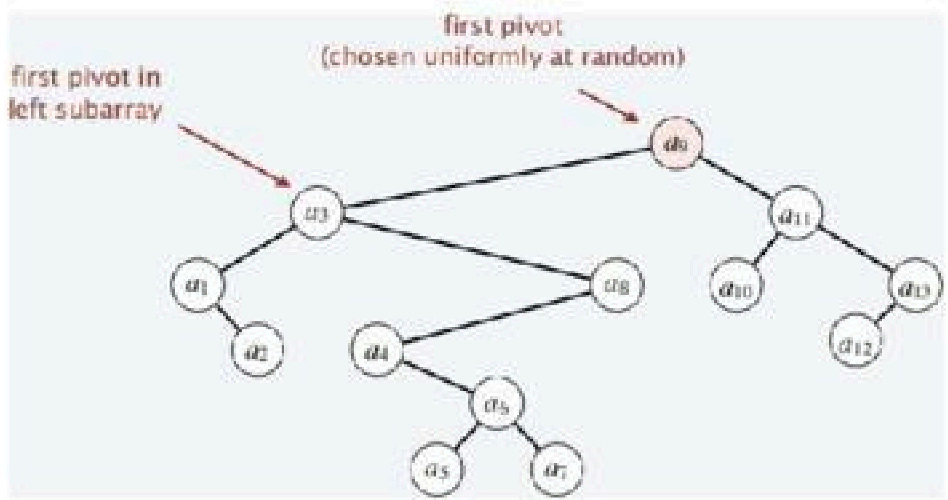
A. 0
B. 1 / n
C. 2 / n
D. 1



Quiz: Quicksort 1

Given an array of $n \geq 8$ distinct elements $a_1 < a_2 < \ldots < a_n$, 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

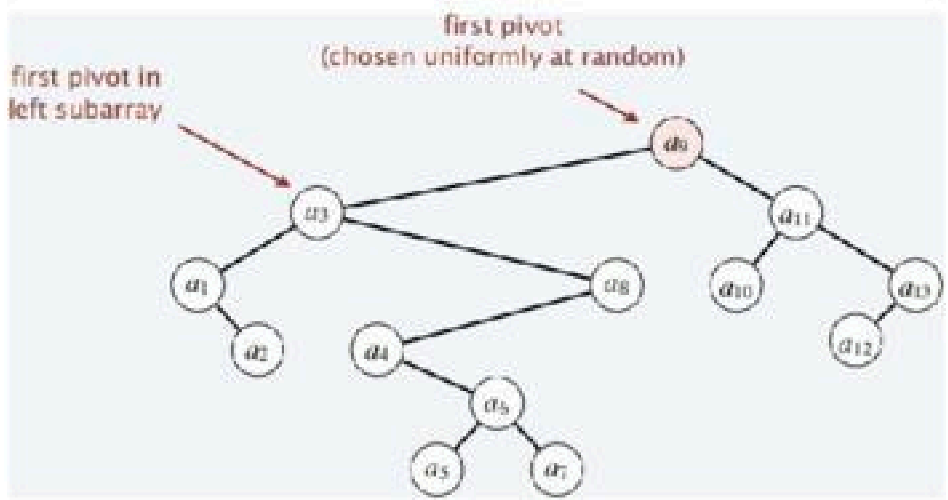


D : ancestry

Quiz: Quicksort 2

Given an array of $n \geq 2$ distinct elements $a_1 < a_2 < \ldots < a_n$, 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

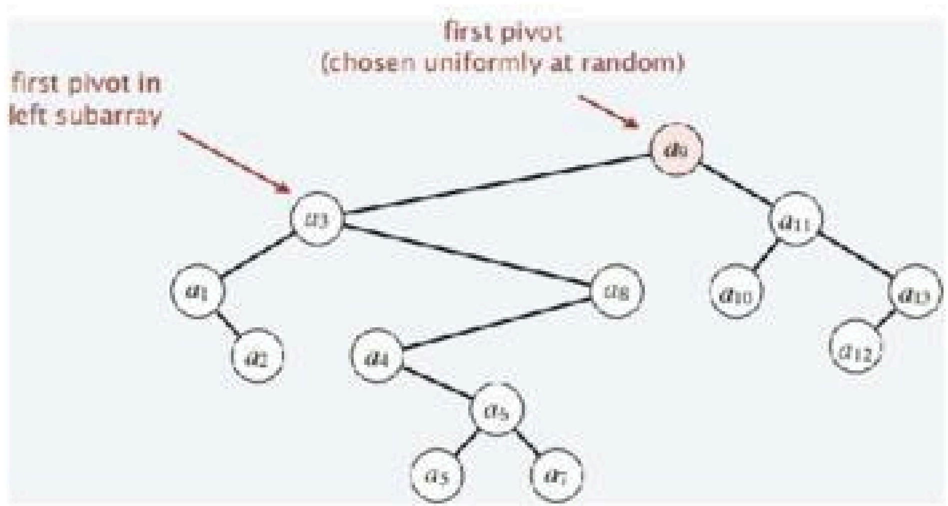


≡

Quiz: Quicksort 2

Given an array of $n \geq 2$ distinct elements $a_1 < a_2 < \ldots < a_n$, 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



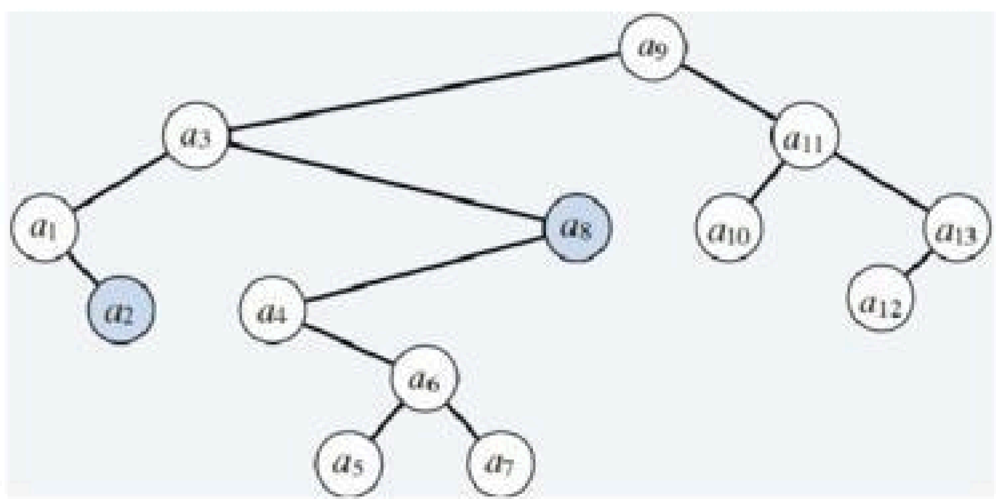
C: compared iff either is chosen as pivot before any of the other

Randomized quicksort: analysis (cont. 1)

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \ldots < a_n$ 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 if either a_2 or a_8 is chosen as pivot before any of $\{a_3, a_4, a_5, a_6, a_7\}$



Randomized quicksort: analysis (cont. 2)

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \ldots < a_n$ 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:

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

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

$$(harmonic) \leq 2n(\ln n + 1)$$

≡

Randomized quicksort: analysis (cont. 2)

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \ldots < a_n$ 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:

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

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

$$(harmonic) \leq 2n(\ln n + 1)$$

≡

Remark. Number of compares only decreases on equal elements.

Closest pair of points

Closest Pair Problem

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

Closest Pair Problem

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.

Closest Pair Problem

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.

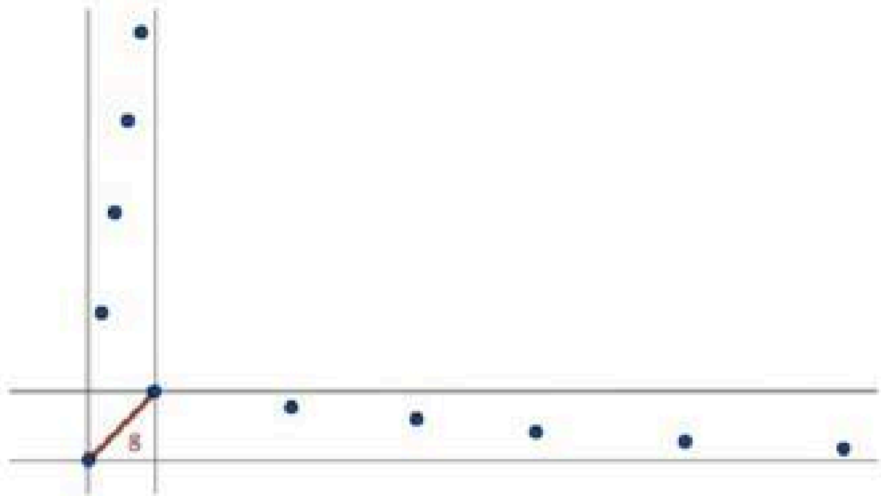
Brute force. Check all pairs with (n^2) distance calculations.

- **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: first attempt

Sorting solution.

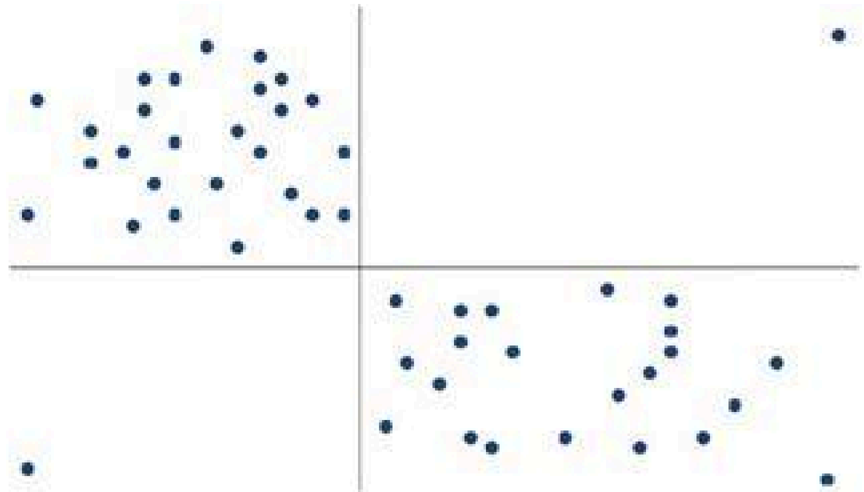
- Sort by x-coordinate and consider nearby points.
- Sort by y-coordinate and consider nearby points.



Closest Pair: second attempt

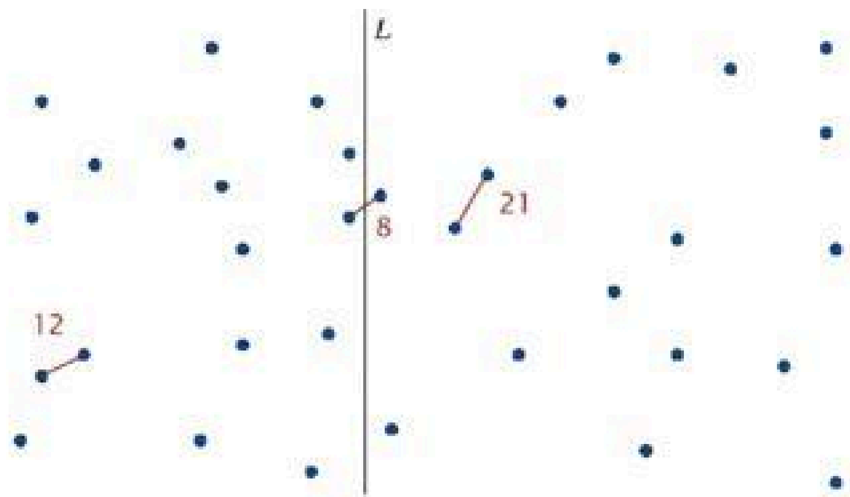
Divide. Subdivide region into 4 quadrants.

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



Closest Pair: divide-and-conquer

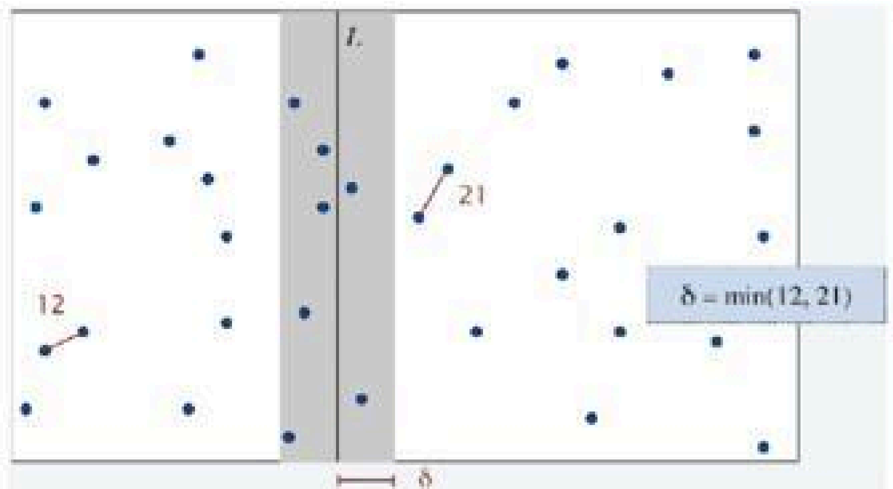
- **Divide**: draw vertical line L so that $n/2$ points on each side.
- **Conquer**: find closest pair in each side recursively.
- **Combine**: find closest pair with one point in each side.
  - looks like $\Theta(n^2)$?
- Return best of 3 solutions.



Closest pair: one point in each side?

Find closest pair with one point in each side, assuming that distance $< \delta$.

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

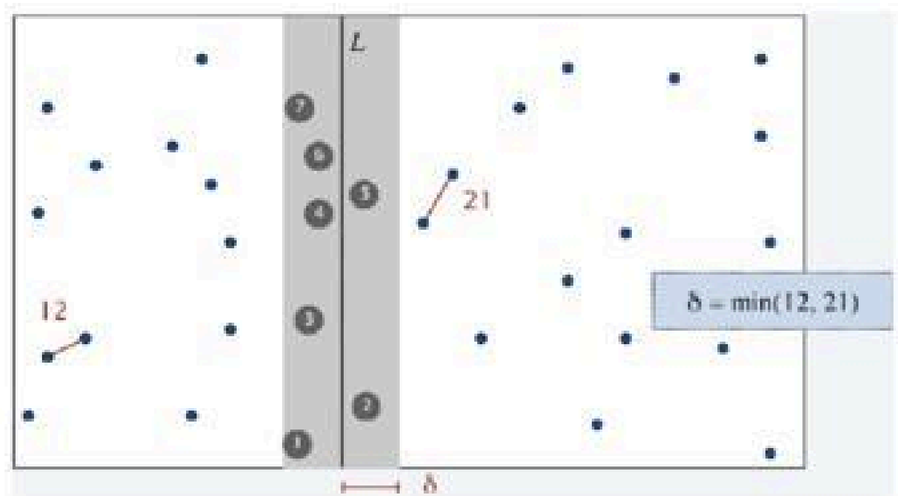


Closest pair: one point in each side? (cont.)

Find closest pair with one point in each side, assuming that distance $< \delta$.

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

- Sort points in 2 δ-strip by their y-coordinate.
- Check distances of only those points within 7 positions in sorted list!
  - But, why?



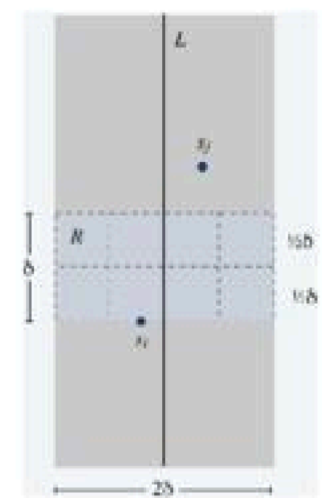
Closest pair: one point in each side

Def. Let s_i be the point in the 2 δ-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 outside R is $\geq \delta$.
- Subdivide R into 8 squares.
  - At most 1 point per square.
    - otherwise, $\delta * \sqrt{2}/2 < \delta$.
  - At most 7 other points can be in R.



Closest pair: algorithm

Input. n points $P = p_1, p_2, \ldots, p_n$.
Output. distance δ.

1. Compute vertical line L such that half the points are on each side of the line: $O(n)$;
2. δ_1 = `CLOSEST-PAIR`(points in left half): $T(n/2)$;
3. δ_2 = `CLOSEST-PAIR`(points in right half): $T(n/2)$;
4. $\delta = \min\{\delta_1, \delta_2\}$;
5. Delete all points further than δ from line L: $O(n)$;
6. Sort remaining points by y-coordinate: $O(n \log n)$;
7. 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.
8. RETURN δ.

Quiz: Closest pair

What is the solution to the following recurrence?

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

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

Quiz: Closest pair

What is the solution to the following recurrence?

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

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

C

Closest pair: Refined 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.

Closest pair: Refined 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.

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

Closest pair: Computational complexity

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.

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

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	$O(n^2)$	$O(n \log n)$