Design and Analysis of Algorithms
Divide-and-Conquer

Si Wu

School of CSE, SCUT
cswusi@scut.edu.cn

TA: 1684350406@qq.com

Topics

- **Counting Inversions**
- **Matrix Multiplication**
- **Randomized Quick-Sort**

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, …, *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: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: ?
- Combine: ?

input

1 5 4 8 10 2 6 9 3 7

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

Counting Inversions: how to combine two sub-problems?

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

A. Easy if A and B are sorted!

Algorithm:

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

list A

7	10	18	3	14

list B

20	23	2	11	16

sort A

3	7	10	14	18

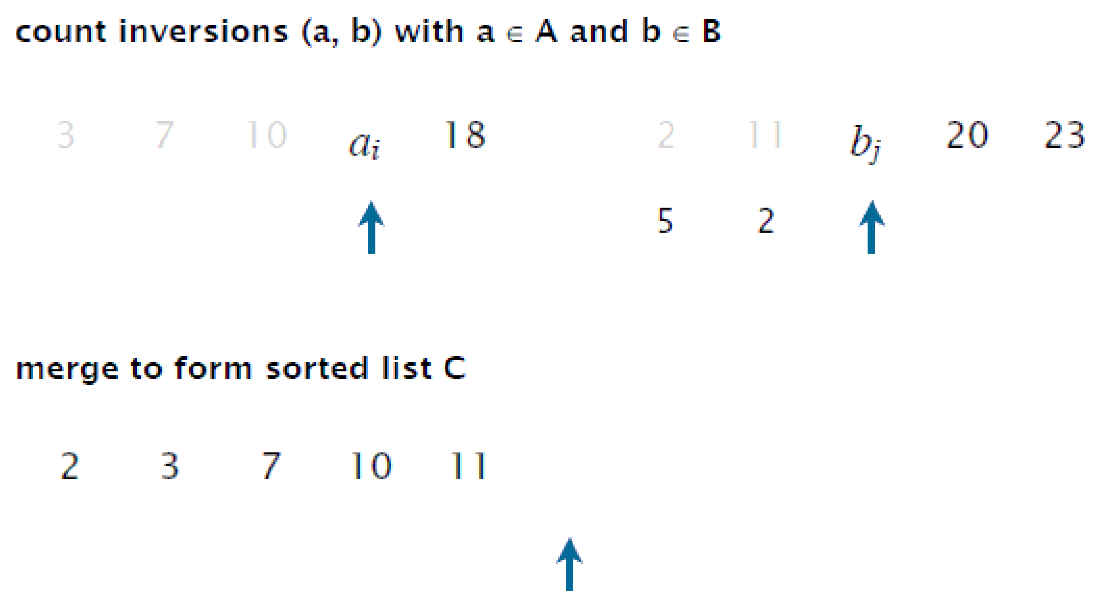
sort B

2	11	16	20	23

Counting Inversions: how to combine two sub-problems?

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.



Counting Inversions: Merge-and-Count

```
Merge-and-Count(A,B)
  Maintain a Current pointer into each list, initialized to
    point to the front elements
  Maintain a variable Count for the number of inversions,
    initialized to 0
  While both lists are nonempty:
    Let a_i and b_j be the elements pointed to by the Current pointer
    Append the smaller of these two to the output list
    If b_j is the smaller element then
      Increment Count by the number of elements remaining in A
    Endif
    Advance the Current pointer in the list from which the
      smaller element was selected.
  EndWhile
Once one list is empty, append the remainder of the other list
    to the output
Return Count and the merged list
```

How about the running time?

Counting Inversions: 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.
$(r_A, A) \leftarrow$ 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).

How about the running time T(n)?

Counting Inversions: algorithm analysis

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

$$T(n) = \ ?$$

Counting Inversions: algorithm analysis

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

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

Proposition. The Sort-and-Count algorithm counts the number of inversions in a permutation of size n in $O(nlogn)$ time.

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A					sorted list B				
3	7	10	14	18	2	11	16	20	23

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

compare minimum entry in each list: copy 2 and add x to inversion count

sorted list C

x = 5 ← number of elements remaining in A

inversions = 0

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5

compare minimum entry in each list: copy 3 and decrement x

sorted list C

2

x = 5

inversions = 5

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5

compare minimum entry in each list: copy 7 and decrement x

sorted list C

2 3

x = 4

inversions = 5

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5

compare minimum entry in each list: copy 10 and decrement x

sorted list C

2 3 7

x = 3

inversions = 5

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5

compare minimum entry in each list: copy 11 and add x to increment count

sorted list C

2 3 7 10

x = 2

inversions = 5

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2

compare minimum entry in each list: copy 14 and decrement x

sorted list C

2 3 7 10 11

x = 2

inversions = 7

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2

compare minimum entry in each list: copy 16 and add x to increment count

sorted list C

2 3 7 10 11 14

x = 1

inversions = 7

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2 1

compare minimum entry in each list: copy 18 and decrement x

sorted list C

2 3 7 10 11 14 16

x = 1

inversions = 8

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2 1

list A exhausted: copy 20

sorted list C

2 3 7 10 11 14 16 18

x = 0

inversions = 8

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2 1 0

list A exhausted: copy 23

sorted list C

2 3 7 10 11 14 16 18 20

x = 0

inversions = 8

Merge-and-Count Demo

Given two sorted lists A and B,

- Count number of inversions (a, b) with $a \in A$ and $b \in B$.
- Merge A and B into sorted list C.

sorted list A

3 7 10 14 18

sorted list B

2 11 16 20 23

5 2 1 0 0

done: return 8 inversions

sorted list C

2 3 7 10 11 14 16 18 20 23

x = 0

inversions = 8

Matrix Multiplication

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

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

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

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Block Matrix Multiplication

C_{11} A_{11} A_{12} B_{11}

$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

Block Matrix Multiplication: Warmup

To multiply two n-by-n matrices A and B:

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

n-by-n matrices

$$C = A \times B$$

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

$\frac{n}{2}$-by-$\frac{n}{2}$ matrices

8 matrix multiplications

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

4 matrix additions

Running time. $T(n) = \ ?$

Block Matrix Multiplication: Warmup

To multiply two n-by-n matrices A and B:

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

n-by-n matrices

$$C = A \times B$$

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

$\frac{n}{2}$-by-$\frac{n}{2}$ matrices

8 matrix multiplications

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

4 matrix additions

Running time. $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) \;\Rightarrow\; T(n) = ?$

Block Matrix Multiplication: Warmup

To multiply two n-by-n matrices A and B:

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

n-by-n matrices

$$C = A \times B$$

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

$\frac{n}{2}$-by-$\frac{n}{2}$ matrices

8 matrix multiplications

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

4 matrix additions

Running time. Apply Case 1 of the master theorem.

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) \;\Rightarrow\; T(n) = \Theta(n^3)$$

Strassen's Trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar matrix multiplications (plus 11 additions and 7 subtractions).

scalars

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

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

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

7 scalar multiplications

Pf.
$$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \end{aligned}$$

Strassen's Trick

Key idea. Can multiply two n-by-n matrices via 7 $\frac{n}{2}$-by-$\frac{n}{2}$ multiplications (plus 11 additions and 7 subtractions).

½n-by-½n matrices

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

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

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

7 matrix multiplications (of ½n-by-½n matrices)

Pf. $$\begin{aligned} C_{12} &= P_1 + P_2 \\ &= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\ &= A_{11} \times B_{12} + A_{12} \times B_{22}. \end{aligned}$$

Strassen's Algorithm

Strassen (n, A, B)

If $(n = 1)$ Return $A \times B$.

Partition A and B into $\frac{n}{2}$-by-$\frac{n}{2}$ blocks.

$P_1 \leftarrow$ Strassen (n/2, A_{11}, $B_{12} - B_{22}$).

$P_2 \leftarrow$ Strassen (n/2, $A_{11} + A_{12}$, B_{22}).

$P_3 \leftarrow$ Strassen (n/2, $A_{21} + A_{22}$, B_{11}).

$P_4 \leftarrow$ Strassen (n/2, A_{22}, $B_{21} - B_{11}$).

$P_5 \leftarrow$ Strassen (n/2, $A_{11} + A_{22}$, $B_{11} + B_{22}$).

$P_6 \leftarrow$ Strassen (n/2, $A_{12} - A_{22}$, $B_{21} + B_{22}$).

$P_7 \leftarrow$ Strassen (n/2, $A_{11} - A_{21}$, $B_{11} + B_{12}$).

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_1$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

Return C.

Analysis of Strassen's Algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two *n*-by-*n* matrices.

Pf.

Apply Case 1 of the master theorem to the recurrence:

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

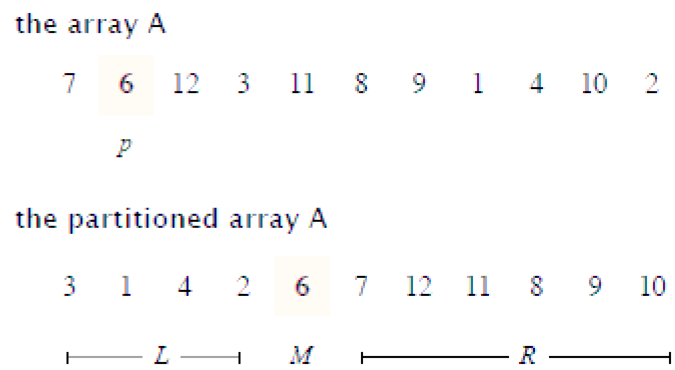
$$\Rightarrow \quad T(n) = \Theta(n^{\log_2 7})$$

If *n* is not a power of 2, could pad matrices with zeros.

Randomized Quick-Sort

3-way partition array so that:

- Pivot element *p* is in place.
- Smaller elements in left subarray L.
- Equal elements in middle subarray M.
- Larger elements in right subarray R.



Recur in both left and right subarrays.

Randomized-Quick-Sort (A)
if list A has zero or one element
    Return.
Pick pivot $p \in A$ uniformly at random.
$(L, M, R) \leftarrow$ Partition-3-Way (A, p).
Randomized-Quick-Sort (L).
Randomized-Quick-Sort (R).

3-way partitioning can be done in-place (using n compares)

Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of n distinct elements is $O(nlogn)$.

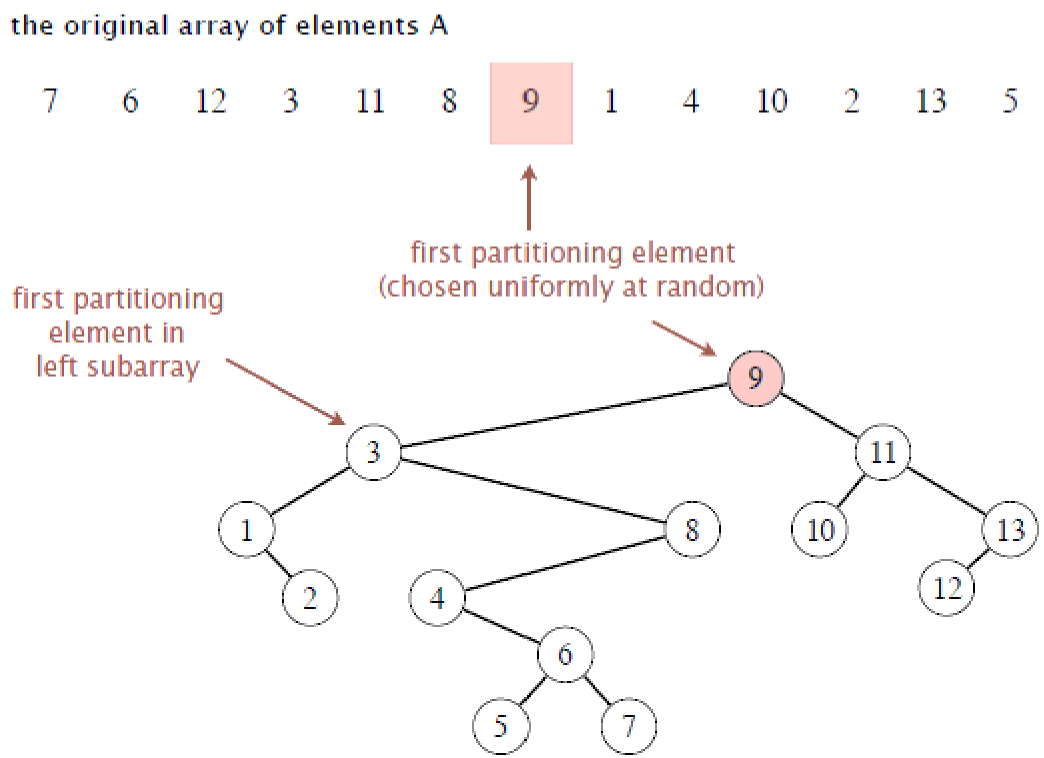
7 6 12 3 11 8 9 1 4 10 2 13 5

Randomized-Quick-Sort (A)
if list A has zero or one element
  Return.
Pick pivot $p \in A$ uniformly at random.
$(L, M, R) \leftarrow$ Partition-3-Way (A, p).
Randomized-Quick-Sort (L).
Randomized-Quick-Sort (R).

Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of *n* distinct elements is $O(nlogn)$.

Pf. Consider BST representation of partitioning elements.

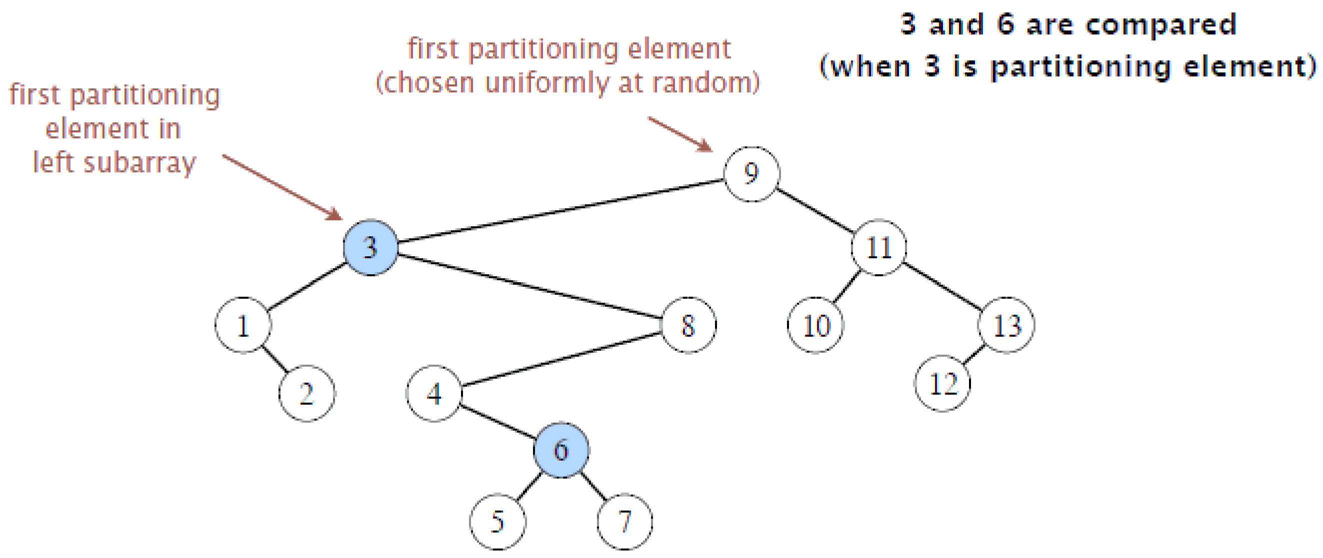


Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of n distinct elements is $O(nlogn)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

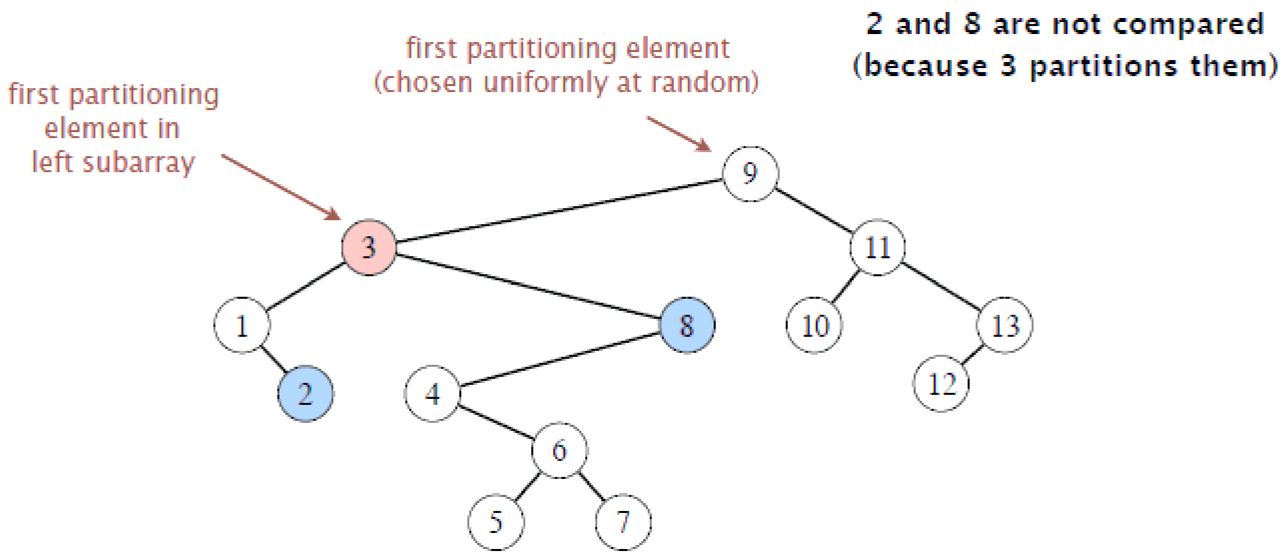


Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of n distinct elements is $O(nlogn)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

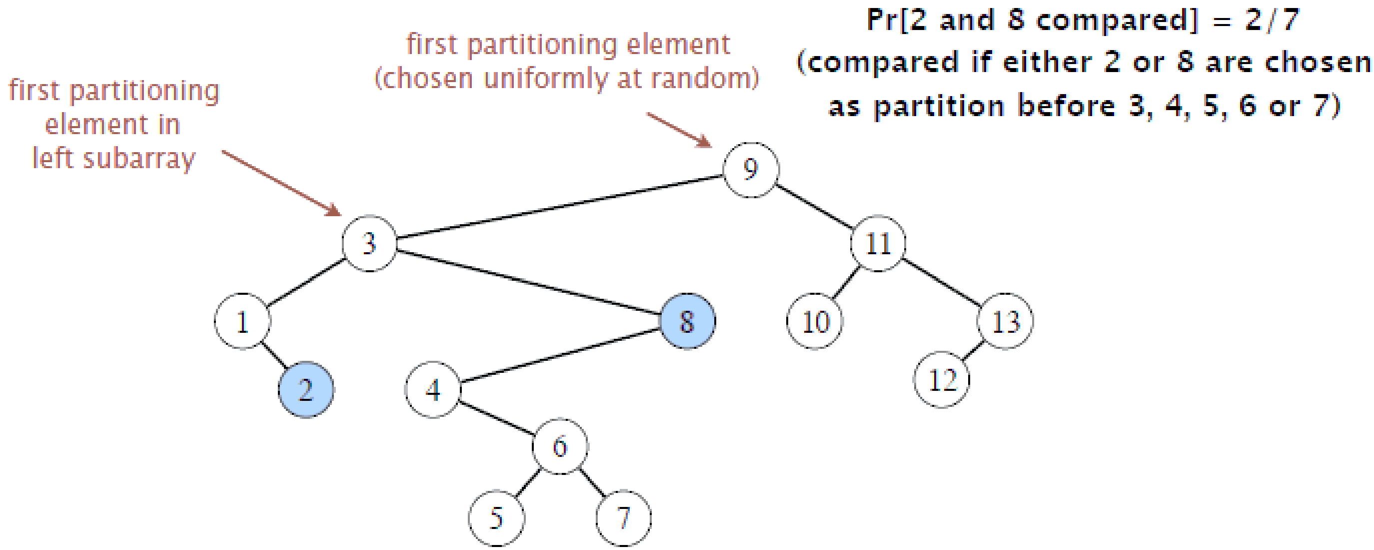


Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of n distinct elements is $O(nlogn)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- **Pr**[a_i and a_j are compared] = 2/(j-i+1), where i<j.



Analysis of Randomized Quick-Sort

Proposition. The expected number of compares to Quick-Sort an array of n distinct elements is $O(nlogn)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- **Pr**[a_i and a_j are compared] = 2/(j-i+1), where i<j.
- Expected number of compares $= \sum_{i}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$

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

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

$$\sim 2n \int_{x=1}^{n} \frac{1}{x} dx = 2nlnn$$