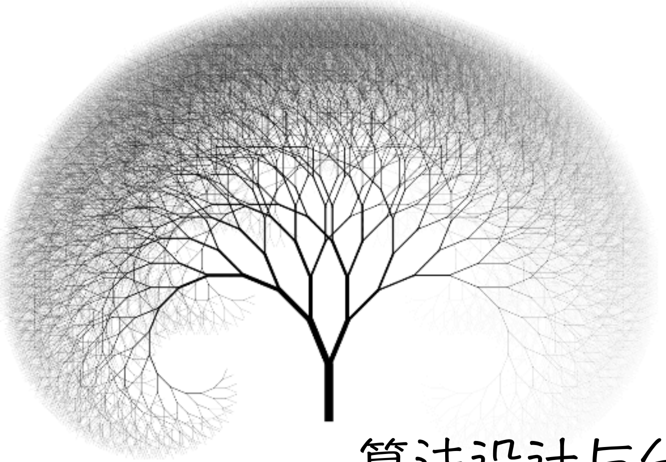


算法设计与分析

刘 安
苏州大学 计算机科学与技术学院
http://web.suda.edu.cn/anliu/

致谢：本课件部分源于Kevin Wayne分享的课件
https://www.cs.princeton.edu/~wayne/kleinberg-tardos/

归约

- 问题X可归约为问题Y，意味着可以使用求解Y的算法A_Y来求解X
  - 解决问题X的算法A_X将A_Y作为黑盒使用，不需要了解A_Y的细节
- 问题X：计算$2k+1$个整数的中位数
- 问题Y：对n个整数排序
  - 算法A_X：使用算法A_Y对$2k+1$个整数排序，然后返回第$k+1$个整数
- 问题X：计算图$G=(V,E)$所有点对之间的最短路径
- 问题Y：计算单源最短路径（即某一个顶点到每个顶点的最短路径）
  - 算法A_X：使用$|V|$次单源最短路径算法，其中每次使用V中的一个不同的顶点作为源结点
- 不要考虑A_Y是如何工作的！即使你知道，也要装作不知道！！

递归

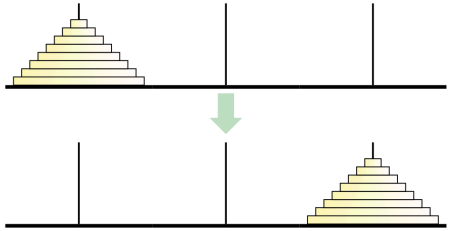
- 递归是一种特殊的归约
  - 如果给定的子问题可以直接求解，那么直接求解
  - 否则将其归约为一个或多个更简单的子问题
    - 子问题的形式和原问题一样，只是规模更小
- 注意
  - 不要考虑这些更简单的子问题是如何求解的，除非可以直接求解
  - 需要保证一定能够归约到一个可以直接求解的子问题（基本情况 base case），否则递归将永远循环下去

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

汉诺塔

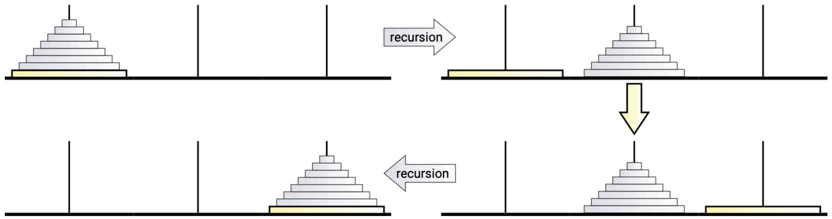
- 如何将一个由n个圆盘组成的塔从一个钉子转移到另一个钉子，可以借助第3个备用钉子，但每次只能移动一个圆盘，且大圆盘不能放在小圆盘的上面



汉诺塔的递归解法

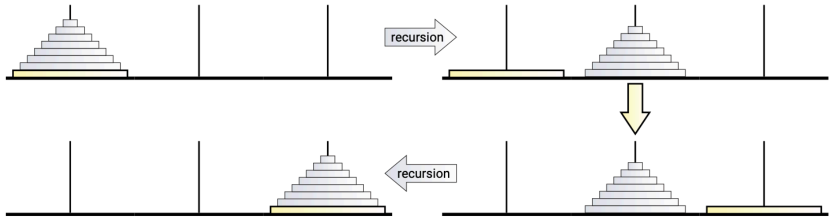
- 当$n > 1$时，最大的圆盘无法移动，所以先将其上的$n-1$个圆盘转移到第3个备用钉子上，然后将其移动至目标钉子，最后再将第3个备用钉子上的$n-1$个圆盘转移到目标钉子

```
def hanoi(n, src, dst, tmp):
    if n > 0:
        hanoi(n - 1, src, tmp, dst)
        move(n, src, dst)
        hanoi(n - 1, tmp, dst, src)
```



运行时间

- 当$n > 1$时，最大的圆盘无法移动，所以先将其上的$n-1$个圆盘转移到第3个备用钉子上，然后将其移动至目标钉子，最后再将第3个备用钉子上的$n-1$个圆盘转移到目标钉子
- 令$T(n)$是转移n个圆盘所需的移动次数
- 那么，$T(n) = 2T(n-1) + 1$，$T(0) = 0$
- 可以证明，$T(n) = 2^n - 1$



递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

归并排序

- 将输入数组分解成大小大致相等的两个子数组
- 对每个子数组进行归并排序（递归求解）
- 将新的有序的子数组合并为一个有序数组

输入数组	A	L	G	O	R	I	T	H	M	S
分解	A	L	G	O	R	I	T	H	M	S
递归求解左子数组	A	G	L	O	R	I	T	H	M	S
递归求解右子数组	A	G	L	O	R	H	I	M	S	T
合并	A	G	H	I	L	M	O	R	S	T

合并两个有序数组

- 将两个有序数组A和B合并为一个有序数组C
  - 自左向右扫描数组A和B
  - 比较a_i和b_j
    - 如果$a_i \le b_j$，将a_i放至C的末尾（a_i不大于任何B中剩余的元素）
    - 如果$a_i > b_j$，将b_j放至C的末尾（b_j小于任何A中剩余的元素）
- 合并算法的运行时间是$O(n)$，更准确的，是$\Theta(n)$

sorted list A

3	7	10	a_i	18

sorted list B

2	11	b_j	20	23

merge to form sorted list C

2	3	7	10	11					

归并排序算法

- 输入：具有n个元素的数组L
- 输出：数组L中的元素升序排列

```
MERGE-SORT(L)

IF (list L has one element)
    RETURN L.

Divide the list into two halves A and B.
A ← MERGE-SORT(A).    ⟵ T(⌈n/2⌉)
B ← MERGE-SORT(B).    ⟵ T(⌊n/2⌋)
L ← MERGE(A, B).      ⟵ Θ(n)
RETURN L.
```

归并排序的Python实现

```python
def merge_sort(L, low, high):
    if low == high:
        return
    else:
        mid = (high + low) // 2
        merge_sort(L, low, mid)
        merge_sort(L, mid + 1, high)
        merge(L, low, high, mid)
```

```python
def merge(L, low, high, mid):
    size = high - low + 1
    C = [0] * size
    i, j = low, mid + 1
    for k in range(size):
        if j > high:
            C[k] = L[i]
            i += 1
        elif i > mid:
            C[k] = L[j]
            j += 1
        elif L[i] <= L[j]:
            C[k] = L[i]
            i += 1
        else:
            C[k] = L[j]
            j += 1
    for k in range(size):
        L[low + k] = C[k]
```


运行时间

- 使用比较操作来衡量排序算法的运行时间
- 令$T(n)$是对长度为n的数组进行归并排序所需的比较次数
  - $T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{if } n > 1 \end{cases}$
- 可以证明，$T(n) = O(n \log n)$

比较次数在$\lfloor n/2 \rfloor$和$n-1$之间

- 一般来说，可以直接忽略取整符号，不影响函数的渐近界
  - $T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ T(n/2) + T(n/2) + n & \text{if } n > 1 \end{cases}$
  - 即，$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$

递归（分治法）

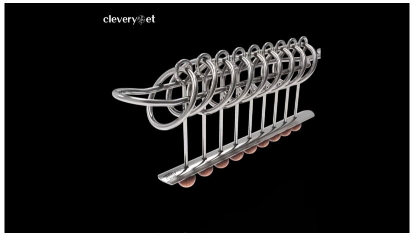
- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

九连环

九连环

- 抽象模型：将一串n个1转换成一串n个0
  - R1：始终可以翻转第一个位（即最右边的位）
  - R2：如果该串恰好以k个0结尾，那么可以翻转第$k+2$个位
- 当$n=4$时，转换过程如下

$$
\begin{aligned}
1111 &\xrightarrow{R2(0)} 1101 \xrightarrow{R1} 1100 \\
&\xrightarrow{R2(2)} 0100 \xrightarrow{R1} 0101 \\
&\xrightarrow{R2(0)} 0111 \xrightarrow{R1} 0110 \\
&\xrightarrow{R2(1)} 0010 \xrightarrow{R1} 0011 \\
&\xrightarrow{R2(0)} 0001 \xrightarrow{R1} 0000
\end{aligned}
$$



九连环的递归解法

- $n=1$：直接使用R1，$1 \xrightarrow{R1} 0$
- $n=2$：先使用R2，$11 \xrightarrow{R2} 01$，然后R1，$01 \xrightarrow{R1} 00$
- $n>2$

remove_all(n)：1 1 1 . . . 1

remove_all($n-2$) ↓

1 1 0 . . . 0

remove(n) ↓

0 1 0 . . . 0

place_all($n-2$) ↓

remove_all($n-1$)：0 1 1 . . . 1

九连环的递归解法

- $n=1$：直接使用R1，$1 \xrightarrow{R1} 0$
- $n=2$：先使用R2，$11 \xrightarrow{R2} 01$，然后R1，$01 \xrightarrow{R1} 00$
- $n>2$

place_all(n)：0 0 0 . . . 0

place_all($n-1$) ↓

0 1 1 . . . 1

remove_all($n-2$) ↓

0 1 0 . . . 0

place(n) ↓

place_all($n-2$)：1 1 0 . . . 0

运行时间

- 令$f(n)$表示remove_all(n)中remove()函数的调用次数，令$g(n)$表示place_all(n)中place()函数的调用次数，显然有$f(n)=g(n)$
- $$f(n)=\begin{cases} 1 & \text{if } n=1 \\ 2 & \text{if } n=2 \\ f(n-1)+2f(n-2)+1 & \text{if } n>2 \end{cases}$$
- 可以证明，$T(n)=\frac{2}{3}2^n-\frac{1}{2}-\frac{1}{6}(-1)^n$

```
def remove_all(n):
    if n == 1: remove(1)
    elif n == 2:
        remove(2), remove(1)
    else:
        remove_all(n - 2)
        remove(n)
        place_all(n - 2)
        remove_all(n - 1)
```

```
def place_all(n):
    if n == 1: place(1)
    elif n == 2:
        place(2), place(1)
    else:
        place_all(n - 1)
        remove_all(n - 2)
        place(n)
        place_all(n - 2)
```

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

代入法

- 先猜测一个解，然后用数学归纳法证明解的正确性
- 汉诺塔：$T(n) = 2T(n-1) + 1$，$T(0) = 0$
- 猜测$T(n) = 2^n - 1$
- 证明
  - $T(0) = 0 = 2^0 - 1$
  - $T(n) = 2T(n-1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$
- 为什么猜测$T(n) = 2^n - 1$
  - $T(0) = 0$，$T(1) = 1$，$T(2) = 3$，$T(3) = 7$，$T(4) = 15$，$T(5) = 31$
  - 展开递归式

展开递归式

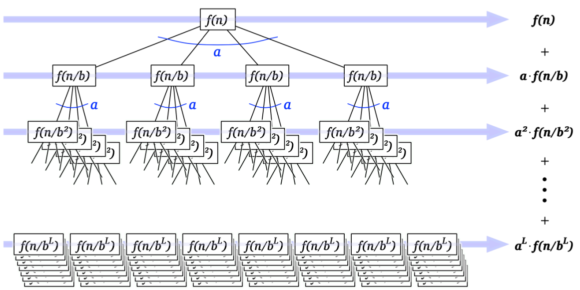
- $k = 1$：$T(n) = 2T(n-1) + 1$
- $k = 2$：$T(n) = 2(2T(n-2) + 1) + 1 = 4T(n-2) + 3$
- $k = 3$：$T(n) = 4(2T(n-3) + 1) + 3 = 8T(n-3) + 7$
- 通过上述观察，猜测$T(n) = 2^kT(n-k) + (2^k - 1)$
- 已知$T(0) = 0$，所以令$n - k = 0$，即将$k = n$代入上式
- $T(n) = 2^kT(n-k) + (2^k - 1) = 2^nT(0) + (2^n - 1) = 2^n - 1$

代入法

- 归并排序，$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n > 1 \end{cases}$
- 先展开递归式
  - $k = 1$：$T(n) \le 2T(n/2) + n$
  - $k = 2$：$T(n) \le 2(2T(n/4) + n/2) + n = 4T(n/4) + 2n$
  - $k = 3$：$T(n) \le 4(2T(n/8) + n/4) + 2n = 8T(n/8) + 3n$
- 通过上述观察，猜测$T(n) \le 2^kT(n/2^k) + kn$
- 已知$T(1) = 0$，所以令$n/2^k = 1$，即将$k = \log_2 n$代入上式
- $T(n) \le 2^kT(n/2^k) + kn = nT(1) + n\log_2 n = n\log_2 n$
- 利用数学归纳法证明$T(n) \le n\log_2 n$（过程省略）
- 所以，$T(n) = O(n\log n)$

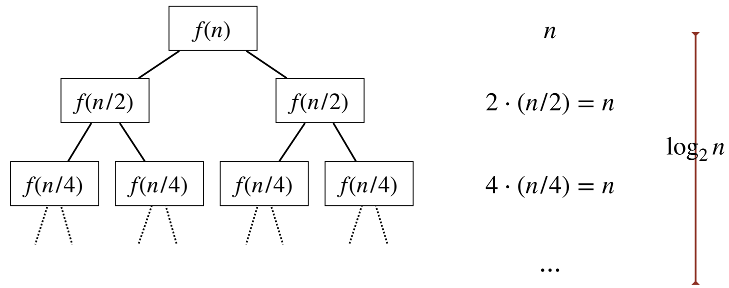
递归树

- $T(n) = aT(n/b) + f(n)$，其中$a \geq 1$，$b > 1$是常量
  - 将输入规模为n的问题分解成a个输入规模为n/b的子问题
  - …
  - 直至规模为1的子问题，即$n/b^L = 1 \Rightarrow L = \log_b n$
- $T(n) = \sum_{i=0}^{L} a^i f(n/b^i)$



递归树

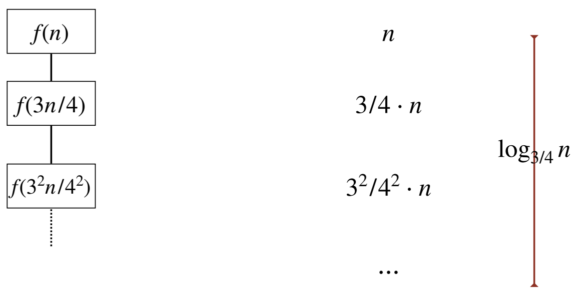
- $T(n) = aT(n/b) + f(n) \Rightarrow T(n) = \sum_{i=0}^{L} a^i f(n/b^i)$，$L = \log_b n$
- 归并排序：$T(n) = 2T(n/2) + n$
  - $f(n) = n, a = 2, b = 2$
  - $T(n) = \sum_{i=0}^{L} a^i f(n/b^i) = \sum_{i=0}^{L} 2^i n/2^i = \sum_{i=0}^{L} n = O(n \log n)$



递归树

- $T(n) = aT(n/b) + f(n) \Rightarrow T(n) = \sum_{i=0}^{L} a^i f(n/b^i)$，$L = \log_b n$
- 随机选择：$T(n) = T(3n/4) + n$
  - $f(n) = n, a = 1, b = 4/3$
  - $T(n) = \sum_{i=0}^{L} a^i f(n/b^i) = \sum_{i=0}^{L} n(3/4)^i = n\sum_{i=0}^{L} (3/4)^i = O(n)$

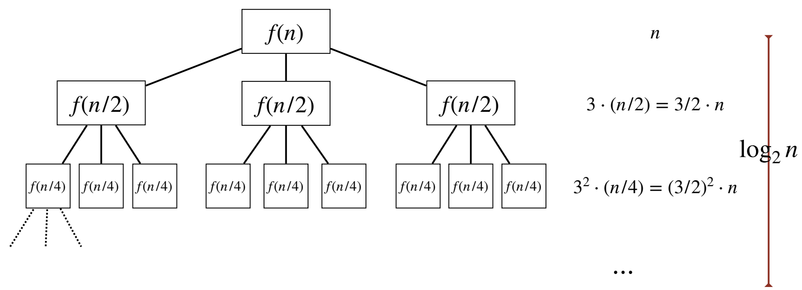
当$r < 1$时，几何级数的和$\sum_k r^k \leq \frac{1}{1-r}$



递归树

- $T(n) = aT(n/b) + f(n) \Rightarrow T(n) = \sum_{i=0}^{L} a^i f(n/b^i)$，$L = \log_b n$
- Karatsuba整数乘法：$T(n) = 3T(n/2) + n$
  - $f(n) = n, a = 3, b = 2$
  - $T(n) = \sum_{i=0}^{L} a^i f(n/b^i) = \sum_{i=0}^{L} n(3/2)^i = n\sum_{i=0}^{L} (3/2)^i = O((3/2)^L n) = O(n^{\log_2 3})$

当$r > 1$时，$\sum_k r^k \leq r^k \frac{r}{r-1}$（最后一项的常数倍）



主定理

- 对于$a \geq 1,\ b \geq 2,\ d \geq 0$，递归式$T(n) = aT(n/b) + O(n^d)$可以如下求解
  - 情况1：如果$d > \log_b a$，那么$T(n) = O(n^d)$
  - 情况2：如果$d = \log_b a$，那么$T(n) = O(n^d \log n)$
  - 情况3：如果$d < \log_b a$，那么$T(n) = O(n^{\log_b a})$
- 归并排序：$T(n) = 2T(n/2) + n \Rightarrow a = 2,\ b = 2,\ d = 1$
  - $\log_b a = 1 = d$，情况2，所以$T(n) = O(n \log n)$
- 随机选择：$T(n) = T(3n/4) + n \Rightarrow a = 1,\ b = 4/3,\ d = 1$
  - $\log_b a = 0 < d$，情况1，所以$T(n) = O(n)$
- Karatsuba整数乘法：$T(n) = 3T(n/2) + n \Rightarrow a = 3,\ b = 2,\ d = 1$
  - $\log_b a = \log_2 3 > d$，情况3，所以$T(n) = O(n^{\log_2 3})$

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

计算逆序数

- 假设$a_1, a_2, \cdots, a_n$是整数$1, 2, \cdots, n$的一个排列，如果$i < j$且$a_i > a_j$，那么称(a_i, a_j)是一个逆序
  - 比如$1, 3, 4, 2, 5$中，有2个逆序$(3, 2)$和$(4, 2)$
- 如何计算$a_1, a_2, \cdots, a_n$中的逆序数
- 穷举法：$\Theta(n^2)$

递归求解逆序数

- 分解：将输入数组分解成大小大致相等的两个子数组A和B
- 解决：递归地求解每个子数组的逆序数
- 合并：计算满足特定条件的逆序(a, b)的数量，其中$a \in A,\ b \in B$
- 返回这些逆序数的和

所有的逆序数=1+3+13=17

输入

1	5	4	8	10	2	6	9	3	7

计算子数组A的逆序数

1	5	4	8	10

(5,4)

计算子数组B的逆序数

2	6	9	3	7

(6,3) (9,3) (9,7)

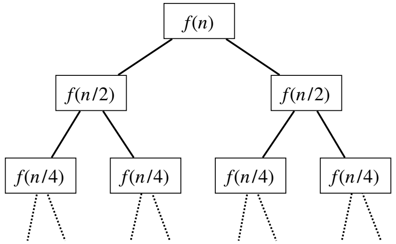
计算逆序(a, b)的数量，其中$a \in A,\ b \in B$

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)

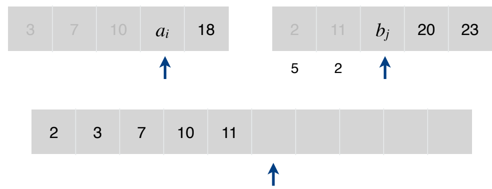
合并算法

- 如何计算逆序(a, b)的数量，其中$a \in A, b \in B$
- 穷举：$T(n) = 2T(n/2) + O(n^2)$
  - $a = b = d = 2,\ \log_b a < d,\ T(n) = O(n^2)$
- 如果子数组A, B均有序，那么对于每个$b \in B$，通过二分查找来确定A中大于b的元素个数
  - $T(n) = 2T(n/2) + O(n \log n)$
  - 第0层：$n \log n$
  - 第1层：$2 \cdot (n/2 \cdot \log(n/2)) = n \log(n/2)$
  - 第2层：$4 \cdot (n/4 \cdot \log(n/4)) = n \log(n/4)$
  - 第i层：$n \log(n/2^i)$
  - 所以，$T(n) = \sum_{i=0}^{\log n} n \log(n/2^i) = n \sum_{i=0}^{\log n} (\log n - i) = O(n \log^2 n)$



改进的合并算法

- 如何计算逆序(a, b)的数量，其中$a \in A, b \in B$，假设A, B均有序
- 自左向右扫描数组A和B
- 比较a_i和b_j
  - 如果$a_i < b_j$，那么a_i不会与任何B中剩余的元素构成逆序
  - 如果$a_i > b_j$，那么b_j与任何A中剩余的元素构成逆序
- 将较小的元素放入有序数组C中
- 运行时间：$O(n)$



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

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
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

分治法计算逆序

- 输入：数组L
- 输入：L中的逆序数（同时也对L进行了排序）
- 运行时间$T(n) = 2T(n/2) + O(n)$
  - $a = b = 2,\ d = 1$
  - $d = \log_b a$
  - $T(n) = O(n^d \log n) = O(n \log n)$

```
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).
(r_B, B)  ← SORT-AND-COUNT(B).
(r_AB, L) ← MERGE-AND-COUNT(A, B).

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


递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

三路划分

- 给定数组A和一个主元p，将A划分成如下3个子数组:
  - 子数组L位于A的左边，由小于p的元素组成
  - 子数组M位于A的中间，由等于p的元素组成
  - 子数组R位于A的右边，由大于p的元素组成
- 挑战：运行时间O(n)，空间需求O(1)

数组A

7	6	12	3	11	8	9	1	4	10	2

$p = 6$

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

Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

lo lt		i											gt hi
16_1	14	12	24	23	16_2	16_3	22	16_4	11	16_5	13	25	26
equal	unknown												

Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

	lt	i											gt
14	16_1	12	24	23	16_2	16_3	22	16_4	11	16_5	13	25	26
less	equal	unknown											

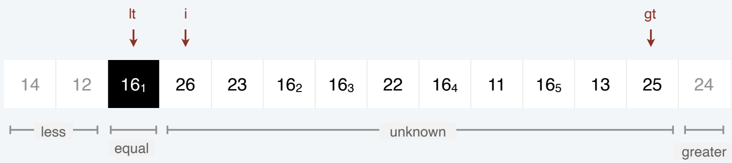
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

		lt	i										gt
14	12	16_1	24	23	16_2	16_3	22	16_4	11	16_5	13	25	26
less		equal	unknown										

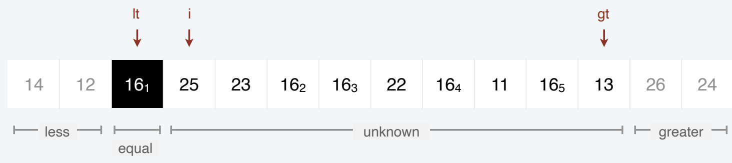
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



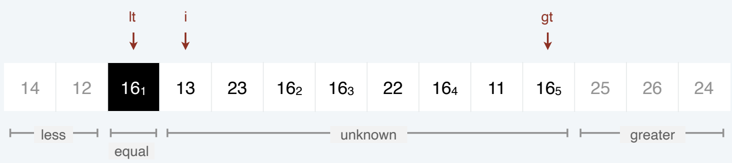
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



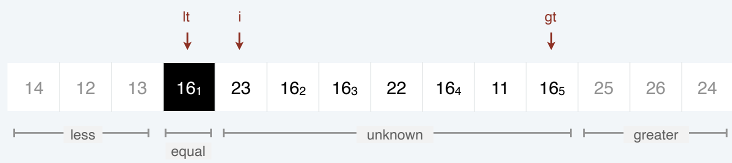
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

			lt	i					gt				
14	12	13	16_1	16_5	16_2	16_3	22	16_4	11	23	25	26	24
less			equal	unknown						greater			

Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

			lt		i				gt				
14	12	13	16_1	16_5	16_2	16_3	22	16_4	11	23	25	26	24
less			equal		unknown					greater			

Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

			lt			i			gt				
14	12	13	16_1	16_5	16_2	16_3	22	16_4	11	23	25	26	24
less			equal			unknown				greater			

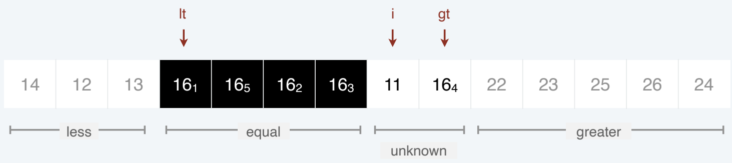
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i

			lt				i		gt				
14	12	13	16_1	16_5	16_2	16_3	22	16_4	11	23	25	26	24
less			equal				unknown			greater			

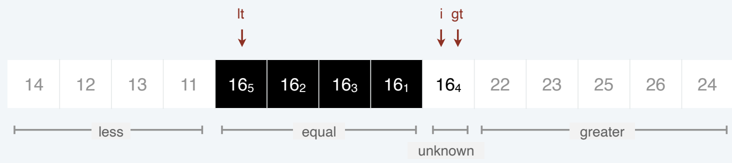
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



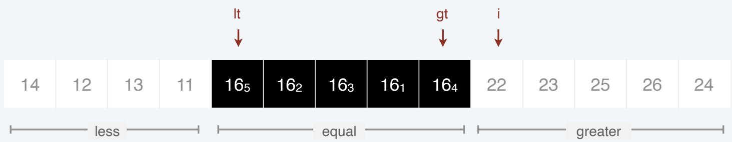
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



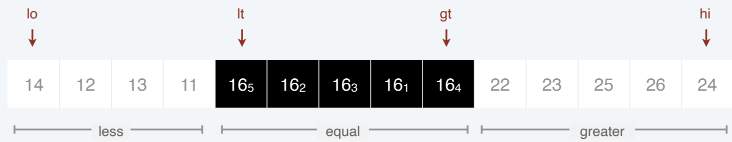
Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



Dijkstra 3-way partitioning demo

- Let p be pivot item.
- Swap p to index lo.
- Scan i from left to right.
  - $(A[i] < p)$: exchange $A[lt]$ with $A[i]$; increment both lt and i
  - $(A[i] > p)$: exchange $A[gt]$ with $A[i]$; decrement gt
  - $(A[i] = p)$: increment i



随机快速排序

- 从A中随机选择一个主元p
- 根据p对A进行三路划分，得到子数组L, M, R
- 递归地对子数组L和R进行排序

the array A	7	6	12	3	11	8	9	1	4	10	2
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

随机快速排序

- 从A中随机选择一个主元p
- 根据p对A进行三路划分，得到子数组L, M, R
- 递归地对子数组L和R进行排序

RANDOMIZED-QUICKSORT(A)

IF (array A has zero or one element)
    RETURN.
Pick pivot $p \in A$ uniformly at random.
$(L, M, R) \leftarrow$ PARTITION-3-WAY(A, p). ⟵ $\Theta(n)$
RANDOMIZED-QUICKSORT(L). ⟵ $T(i)$
RANDOMIZED-QUICKSORT(R). ⟵ $T(n-i-1)$

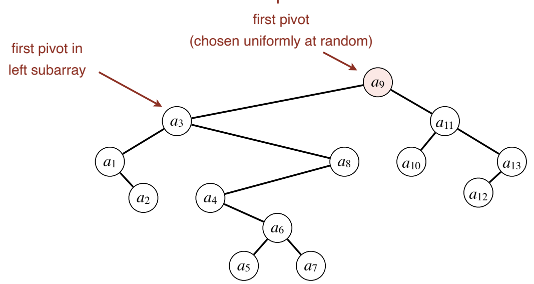
子问题的规模取决于变量i

运行时间分析

- 对具有n个不同元素$a_1 < a_2 < \cdots < a_n$的数组进行快速排序，预期的比较次数是$O(n \log n)$
  - 根据主元的选择顺序构造一棵二叉树

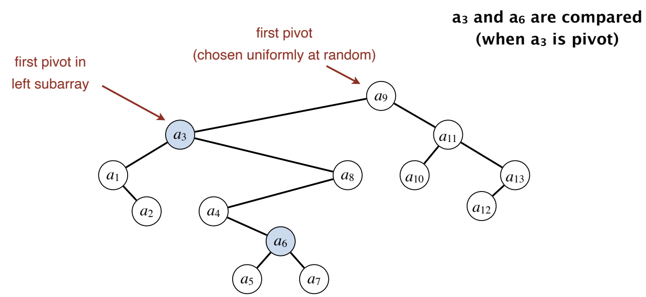
the original array of elements A

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



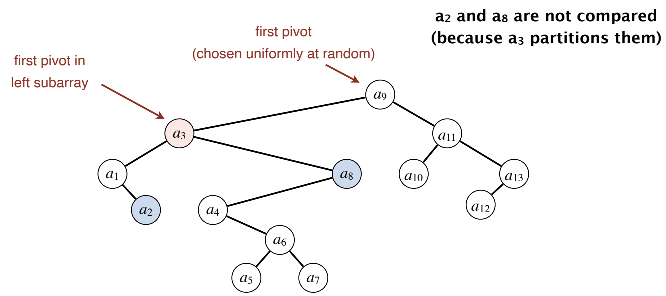
运行时间分析

- 对具有n个不同元素$a_1 < a_2 < \cdots < a_n$的数组进行快速排序，预期的比较次数是$O(n \log n)$
  - 根据主元的选择顺序构造一棵二叉树
  - a_i会与a_j比较一次当且仅当一个是另一个的祖先



运行时间分析

- 对具有n个不同元素$a_1 < a_2 < \cdots < a_n$的数组进行快速排序，预期的比较次数是$O(n \log n)$
  - 根据主元的选择顺序构造一棵二叉树
  - a_i会与a_j比较一次当且仅当一个是另一个的祖先



Divide-and-conquer: quiz

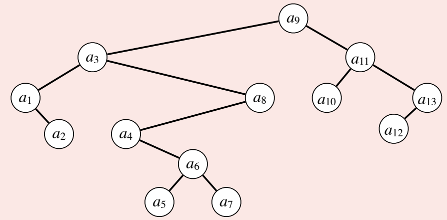
Given an array of $n \ge 8$ distinct elements $a_1 < a_2 < \cdots < a_n$, what is the probability that a_7 and a_8 are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

D. 1



Divide-and-conquer: quiz

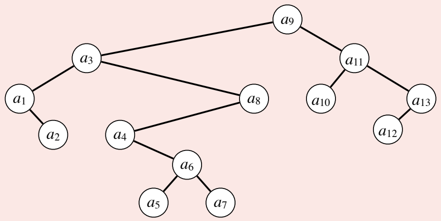
Given an array of $n \ge 2$ distinct elements $a_1 < a_2 < \cdots < a_n$, what is the probability that a_1 and a_n are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

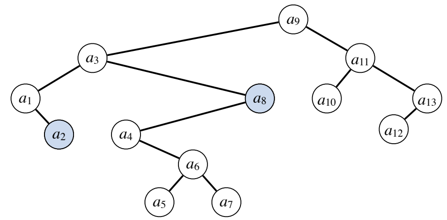
D. 1



运行时间分析

- 对具有n个不同元素$a_1 < a_2 < \cdots < a_n$的数组进行快速排序，预期的比较次数是$O(n \log n)$
  - a_i会与a_j比较一次当且仅当一个是另一个的祖先
  - a_i与a_j比较的概率 $= \dfrac{2}{j - i + 1}$， 其中$i < j$

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}



运行时间分析

- 对具有n个不同元素$a_1 < a_2 < \cdots < a_n$的数组进行快速排序，预期的比较次数是$O(n \log n)$
  - a_i会与a_j比较一次当且仅当一个是另一个的祖先
  - a_i与a_j比较的概率 $= \dfrac{2}{j-i+1}$，其中$i<j$
  - 期望比较次数 $= \displaystyle\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{2}{j-i+1} = \sum_{i=1}^{n}\sum_{j=2}^{n-i+1}\frac{1}{j}$

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

$$\le 2n(\ln n + 1)$$ 调和级数

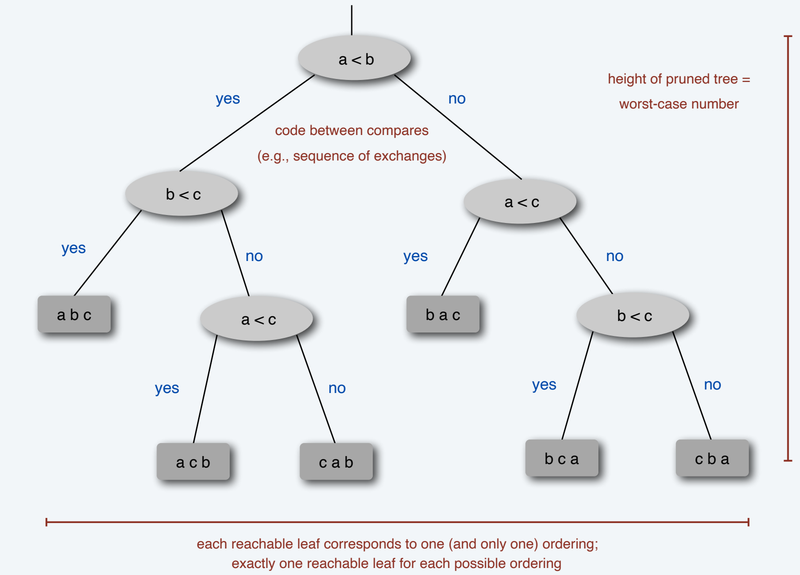
题外话：排序算法的下界

- 挑战：如何证明一类算法（比如所有可能的排序算法）的下界？
- 计算模型：比较树
  - 只能通过“成对比较”来访问元素
  - 所有其他操作（比如流程控制、数据移动等）都是免费的
- 通过比较次数来衡量算法的运行时间
- 上面的计算模型现实吗？
  - 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)

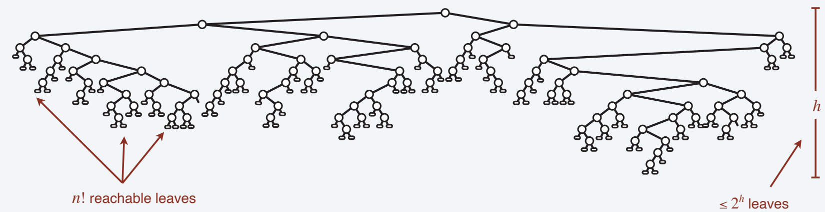


Sorting lower bound

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 $\le 2^h$ leaves.
- n ! different orderings $\Rightarrow$ n ! reachable leaves.

$$2^h \ \ge \ \text{\# reachable leaves} \ = \ n\,!$$

$$\begin{aligned}
\Rightarrow h &\ge \log n! \\
&= \log 1 + \log 2 + \cdots + \log(n/2) + \cdots + \log n \\
&\ge \log(n/2) + \log(n/2+1) + \cdots + \log n \\
&\ge \log(n/2) + \log(n/2) + \cdots + \log(n/2) \\
&= n/2 \cdot \log(n/2) \\
&= 1/2 \cdot n \log n - n/2
\end{aligned}$$

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

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

选择问题

- 给定n个互异的数，找到第k小的数
  - $k = 1$（最小值），$k = n$（最大值），$k = \lfloor (n+1)/2 \rfloor$（中位数）
  - 计算最小值或者最大值：$O(n)$次比较
  - 基于排序：$O(n \log n)$次比较
  - 基于堆
    - $O(n + k \log n)$次比较：构建n个元素的小顶堆+k次extract-min操作
    - $O(k + n \log k)$次比较：构建k个元素的大顶堆+最多n次insert操作
      - 大顶堆存放最小的k个元素
      - 第二种方法更适合流数据
- 挑战：可以通过$O(n)$次比较解决选择问题吗？

随机快速选择

- 从A中随机选择一个主元p
- 根据p对A进行三路划分，得到子数组L, M, R
- 递归地对包含第k小元素的数组进行选择

```
QUICK-SELECT(A, k)

Pick pivot p ∈ A uniformly at random.
(L, M, R) ← PARTITION-3-WAY(A, p).                 ← Θ(n)
IF      (k <= |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                          RETURN p.
```

Quickselect demo

- 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.

select the k = 8th smallest

65	28	59	33	21	56	22	95	50	12	90	53	28	77	39

k = 8th smallest

Quickselect demo

- 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.

choose a pivot element at random and partition

65	28	59	33	21	56	22	95	50	12	90	53	28	77	39

k = 8th smallest

Quickselect demo

- 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.

partitioned array

28	33	21	56	22	50	12	53	28	39	59	65	95	90	77

L R

k = 8th smallest

Quickselect demo

- 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.

recursively select 8th smallest element in left subarray

28	33	21	56	22	50	12	53	28	39	59	65	95	90	77

k = 8th smallest

Quickselect demo

- 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.

choose a pivot element at random and partition

28	33	21	56	22	50	12	53	28	39	59	65	95	90	77

k = 8^{th} smallest

Quickselect demo

- 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.

partitioned array

21	22	12	28	28	33	56	50	53	39	59	65	95	90	77
	L		M				R							

k = 8^{th} smallest

Quickselect demo

- 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.

recursively select the 3^{rd} smallest element in right subarray

21	22	12	28	28	33	56	50	53	39	59	65	95	90	77

k = 3^{rd} smallest

Quickselect demo

- 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.

choose a pivot element at random and partition

21	22	12	28	28	33	56	50	53	39	59	65	95	90	77

k = 3^{rd} smallest

Quickselect demo

- 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.

partitioned array

21	22	12	28	28	33	39	50	53	56	59	65	95	90	77
					L			R						

k = 3rd smallest

Quickselect demo

- 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.

stop: desired element is in middle subarray

21	22	12	28	28	33	39	50	53	56	59	65	95	90	77

随机快速选择的运行时间

- $T(n,k)$：在n个元素中选择第k小元素的期望比较次数
- i：主元的位置

$$T(n,k) \le \begin{cases} \max\{T(0), T(n-1)\} + cn & \text{if } i = 1 \text{ or } i = n \\ \max\{T(1), T(n-2)\} + cn & \text{if } i = 2 \text{ or } i = n-2 \\ \max\{T(\cdots), T(\cdots)\} + cn & \cdots \\ \max\{T(n/2), T(n/2)\} + cn & \text{if } i = n/2 \end{cases}$$

- 从上式可知，$$\begin{aligned} T(n,k) &\le 1/n \cdot (n \cdot cn + 2T(n/2) + \cdots + 2T(n-2) + 2T(n-1)) \\ &\le cn + 1/n \cdot (2T(n/2) + \cdots + 2T(n-2) + 2T(n-1)) \end{aligned}$$
- 令$T(n) = \max_k T(n,k)$，猜测$T(n) \le 4cn$
- 当$n=0$和$n=1$时，显然成立

$$\begin{aligned} T(n) &\le cn + 1/n \cdot (2T(n/2) + \cdots + 2T(n-2) + 2T(n-1)) \\ &\le cn + 1/n \cdot (8c(n/2) + \cdots + 8c(n-2) + 8c(n-1)) \\ &\le cn + c/n \cdot (3n^2 - 2n) \le cn + 3cn = 4cn \end{aligned}$$

最坏情况为线性时间的选择算法

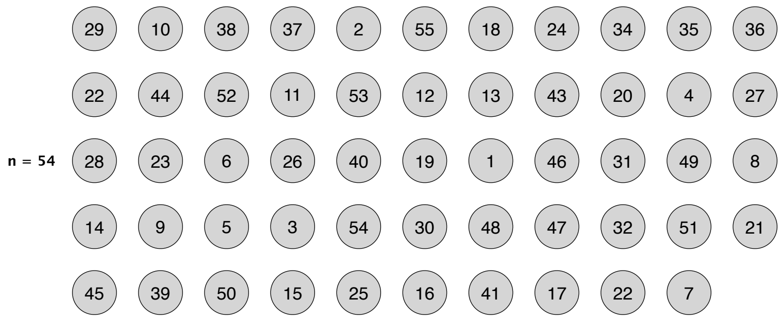
- 目标：找到将n个元素分成两部分的主元p，使得每部分的元素个数一定小于等于$7/10 \cdot n$
- 如何在线性时间内找到近似中位数？
  - 聪明地选取$2/10 \cdot n$个元素，然后递归地求解这些元素的中位数
- $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(7/10 \cdot n) + T(2/10 \cdot n) + \Theta(n) & \text{otherwise} \end{cases}$$

两个不同规模的子问题

- 可以证明$T(n) = \Theta(n)$

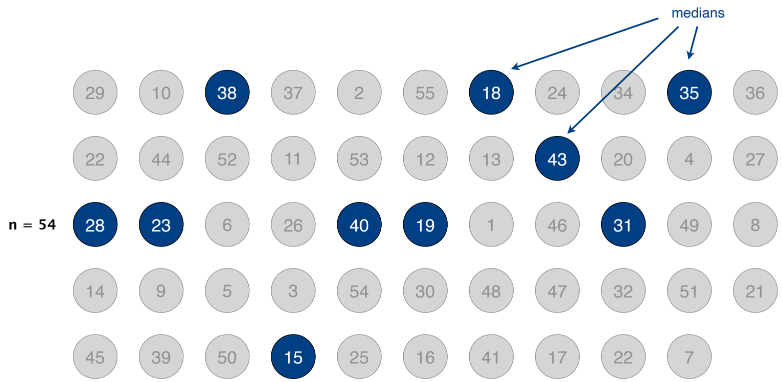
主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\,\%\,5$个元素组成



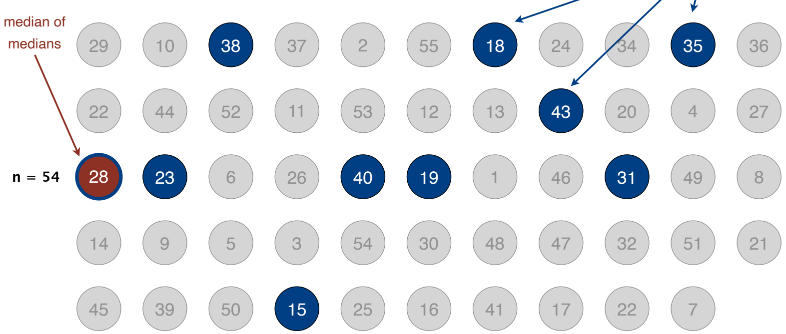
主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\,\%\,5$个元素组成
- 计算每一组的中位数（额外那一组除外）



主元选择

- 将n个元素分成$\lfloor n/5 \rfloor$组，每组5个元组，可能还有额外的一组，由剩下的$n\,\%\,5$个元素组成
- 计算每一组的中位数（额外那一组除外）
- 递归地计算$\lfloor n/5 \rfloor$个中位数的中位数，将其作为主元

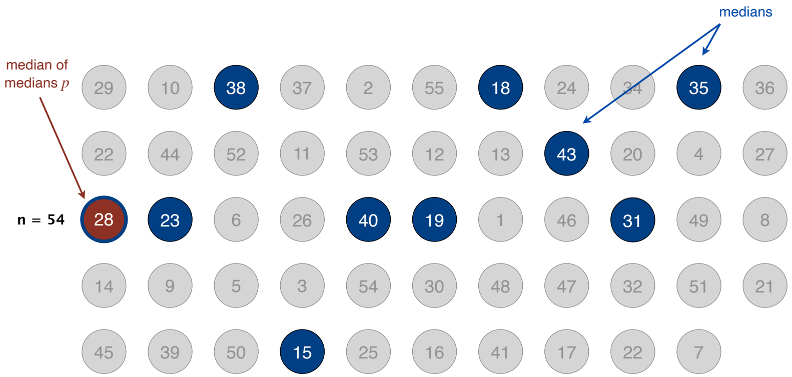


基于中位数的中位数的选择算法MOM-Select

```
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.
```

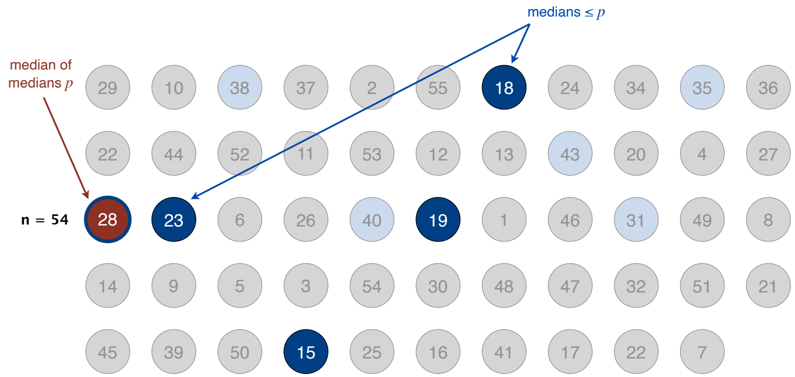
MOM-Select的运行时间

- 至少有一半中位数小于等于主元p



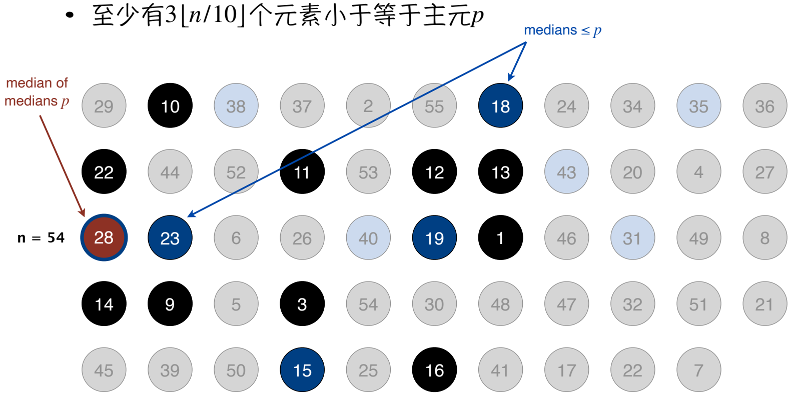
MOM-Select的运行时间

- 至少有一半中位数小于等于主元p
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数小于等于主元$p$



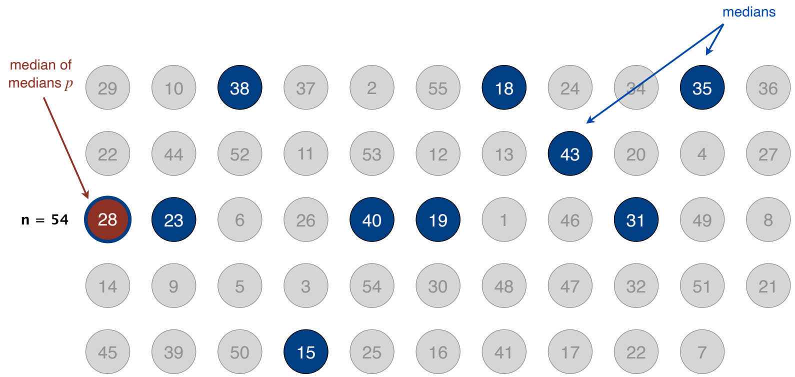
MOM-Select的运行时间

- 至少有一半中位数小于等于主元p
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数小于等于主元$p$
- 至少有$3\lfloor n/10 \rfloor$个元素小于等于主元p



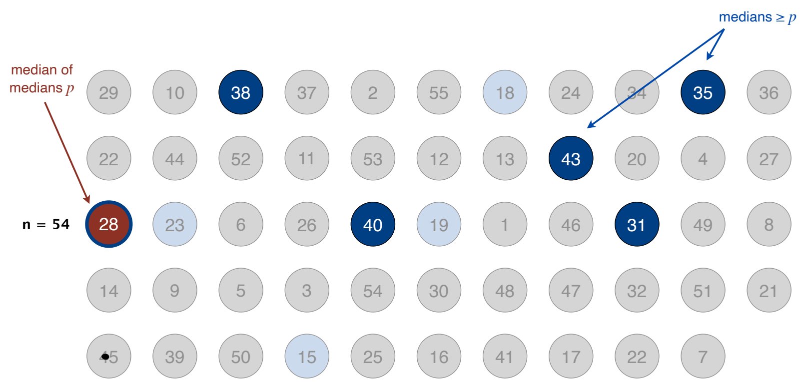
MOM-Select的运行时间

- 至少有一半中位数大于等于主元p



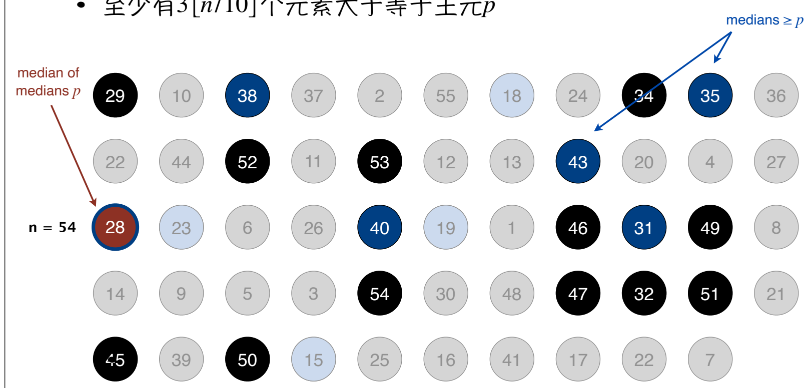
MOM-Select的运行时间

- 至少有一半中位数大于等于主元p
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数大于等于主元$p$



MOM-Select的运行时间

- 至少有一半中位数大于等于主元p
- 至少有$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$个中位数大于等于主元$p$
- 至少有$3\lfloor n/10 \rfloor$个元素大于等于主元p



MOM-Select运行时间的递归式

- 从$\lfloor n/5 \rfloor$个中位数中递归地选择中位数，即主元p
- 至少有$3\lfloor n/10 \rfloor$个元素小于等于主元p
- 至少有$3\lfloor n/10 \rfloor$个元素大于等于主元p
- 从最多$n - 3\lfloor n/10 \rfloor$个元素中递归地选择
- 令$T(n)$是算法MOM-Select从n个元素中选择第k小元素所需的比较次数
  - $T(n) \le \underbrace{T(\lfloor n/5 \rfloor)}_{\text{median of medians}} + \underbrace{T(n - 3\lfloor n/10 \rfloor)}_{\text{recursive select}} + 11/5 \cdot n$

    computing median of 5 (≤ 6 compares per group)

    partitioning (≤ n compares)

MOM-Select的运行时间

- $$T(n) \le \begin{cases} 6n & \text{if } n < 50 \\ T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n & \text{if } n \ge 50 \end{cases}$$
- 猜测：$T(n) \le 44n$
- 数学归纳法证明
  - 基本情况：当$n < 50$时使用归并排序，所以$T(n) \le 6n$
  - 当$n \ge 50$时，

$$\begin{aligned} T(n) &\le T(\lfloor n/5 \rfloor) + T(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n \\ &\le 44(\lfloor n/5 \rfloor) + 44(n - 3\lfloor n/10 \rfloor) + 11/5 \cdot n \\ &\le 44(n/5) + 44n - 44(n/4) + 11/5 \cdot n \\ &= 44n \end{aligned}$$

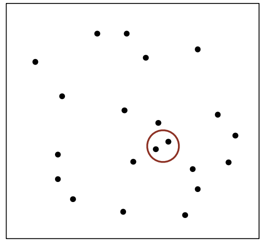
当$n \ge 50$时，$3\lfloor n/10 \rfloor \ge n/4$

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

最近点对问题

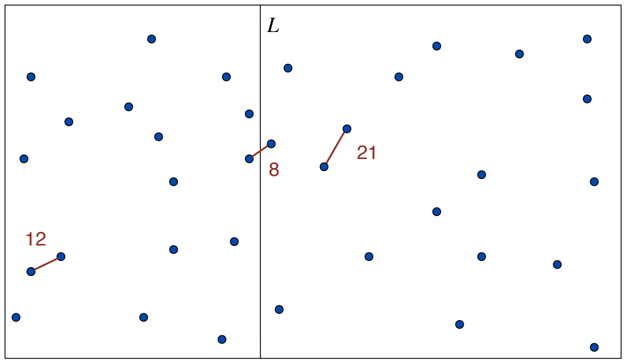
- 给定二维平面上的n个点，找到距离最近的一对点
- 穷举法：$\Theta(n^2)$
- 一维情况：数轴上的n个点，显然有$O(n \log n)$的算法



分治法求解最近点对

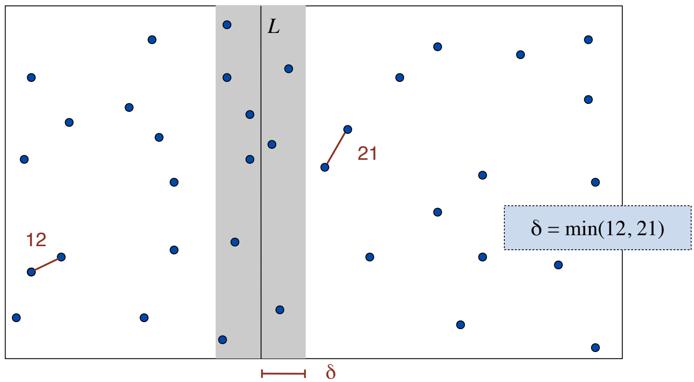
- 分解：构造垂线L使得L的两侧各有$n/2$个点
- 解决：递归地求解每侧的最近点对
- 合并：找到最近的特殊点对，即一个点在L的左侧，另一个点在L的右侧
- 返回三者之间的最近点对

seems like $\Theta(n^2)$



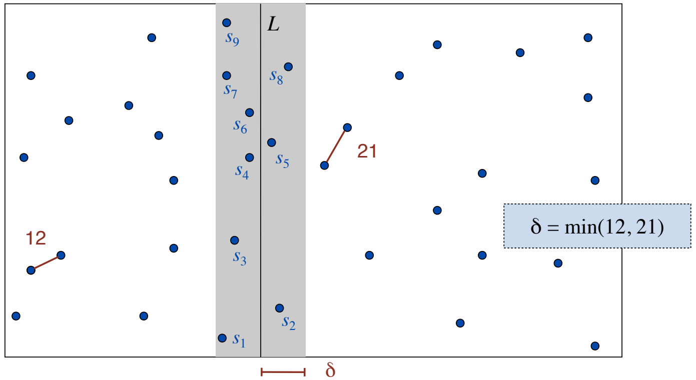
如何找到特殊的最近点对

- 假设L左侧的最近点对的距离是δ_1，L右侧的最近点对的距离是δ_2，令$\delta = \min\{\delta_1, \delta_2\}$
- 仅仅需要考虑距离L小于δ的点



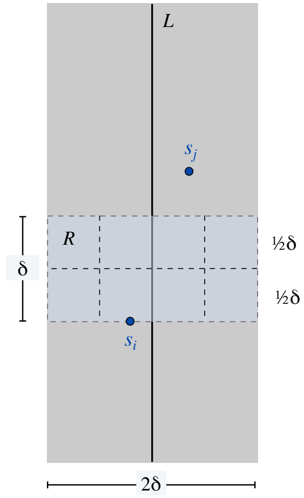
如何找到特殊的最近点对

- 将宽为2δ的带状区域内的点根据y坐标从小到大排序
- 对于每个点，只需计算它与7个点之间的距离
- $7n$次计算就可以找到特殊的最近点对



如何找到特殊的最近点对

- 令s_i是带状区域中y坐标第i小的点
- 如果$|j-i|>7$，那么$d(s_i,s_j)\ge\delta$
  - 考虑带状区域中一个长为2δ宽为δ的矩形R，其下底边恰好经过点s_i
  - 令s_j是任意位于矩形R上方的点，其到s_i的距离显然大于等于δ
  - 将矩形R分成8个小正方形
  - 每个小正方形中最多只有一个点
    - 小正方形的对角线长为$\delta/\sqrt{2}<\delta$
    - L两侧的最近点对的距离是δ
  - 除了s_i，R中最多只有7个点



最近点对的分治算法

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.	← $O(n)$
δ_1 ← CLOSEST-PAIR(points in left half).	← $T(\lfloor n/2\rfloor)$
δ_2 ← CLOSEST-PAIR(points in right half).	← $T(\lceil n/2\rceil)$
δ ← min { δ_1 , δ_2 }.	
A ← list of all points closer than δ to line L.	← $O(n)$
Sort points in A by y-coordinate.	← $O(n \log n)$
Scan points in A 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

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

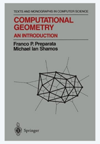
Q. How to improve to $O(n \log n)$?

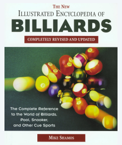
A. Don't sort points in strip from scratch each time.

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





最近点对的分治算法-优化版

CLOSEST-PAIR(P_x, P_y)
INPUT: two copies P_x and P_y of n points, sorted by x- and y-coordinate, respectively $O(n \log n)$
OUTPUT: a pair p_i, p_j of points with the smallest Euclidean distance

$L_x \leftarrow$ first half of P_x, sorted by x-coordinate $O(1)$
$R_x \leftarrow$ second half of P_x, sorted by x-coordinate $O(1)$
// scan P_y and add each point to the end of either L_y or R_y according to its x-coordinate
$L_y \leftarrow$ first half of P_x, sorted by y-coordinate
$R_y \leftarrow$ second half of P_x, sorted by y-coordinate $O(n)$
$l_1, l_2 \leftarrow$ CLOSEST-PAIR(L_x, L_y) $T(\lfloor n/2 \rfloor)$
$r_1, r_2 \leftarrow$ CLOSEST-PAIR(R_x, R_y) $T(\lceil n/2 \rceil)$
$\delta \leftarrow \min\{ d(l_1, l_2), d(r_1, r_2) \}$ $O(1)$
Compute vertical line L such that half the points are on each side of the line $O(1)$
$A \leftarrow$ sorted list of all points closer than δ to line L // scan P_y and remove some points $O(n)$
Scan points in A in y-order and compare distance between each point and next 7 neighbors $O(n)$
If any of these distances is less than δ, update δ and p_i, p_j $O(1)$
RETURN p_i, p_j

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- **整数乘法**
- 矩阵乘法

整数的加减法

- 加法：给定两个n比特的数a和b，计算$a + b$
- 减法：给定两个n比特的数a和b，计算$a - b$
- 小学算法：$\Theta(n)$次比特操作
  - 而不是随机访问机RAM模型中的基本操作数目

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

注意：小学加减算法是最优的

整数乘法

- 乘法：给定两个n比特的数a和b，计算$a \times b$
- 小学算法：$\Theta(n^2)$次比特操作

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

猜想【柯尔莫哥洛夫 Kolmogorov 1956】小学乘法算法是最优的

定理【卡拉楚巴 Karatsuba 1960】上述猜想是错误的

基于分治的整数乘法

- 给定两个n比特的数x和y:
  - 将x和y拆分：记a是x的高阶$n/2$位，b是x的低阶$n/2$位，c是y的高阶$n/2$位，d是y的低阶$n/2$位
  - 递归地进行4次两个$n/2$位整数的乘法计算
  - 通过加法和移位操作获得最终结果

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

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

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

use bit shifting to compute 4 terms

$$xy = (2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

1 2 3 4

Ex. $x = \underbrace{1000}_{a}\underbrace{1101}_{b} \quad y = \underbrace{1110}_{c}\underbrace{0001}_{d}$

基于分治的整数乘法

```
MULTIPLY(x, y, n)

IF (n = 1)
   RETURN x × y.
ELSE
   m ← ⌈ n / 2 ⌉.
   a ← ⌊ x / 2^m ⌋;  b ← x mod 2^m.         ← Θ(n)
   c ← ⌊ y / 2^m ⌋;  d ← y mod 2^m.
   e ← MULTIPLY(a, c, m).
   f ← MULTIPLY(b, d, m).                   ← 4 T(⌈n / 2⌉)
   g ← MULTIPLY(b, c, m).
   h ← MULTIPLY(a, d, m).
   RETURN 2^(2m) e + 2^m (g + h) + f.       ← Θ(n)
```


Divide-and-conquer: quiz

How many bit operations to multiply two n-bit integers using the divide-and-conquer multiplication algorithm?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases}$$

A. $T(n) = \Theta(n^{1/2})$.

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

C. $T(n) = \Theta(n^{\log_2 3}) = O(n^{1.585})$.

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

Karatsuba算法的技巧

- 给定两个n比特的数x和y:
  - 将x和y拆分成4个$n/2$比特的数：记a是x的高阶$n/2$位，b是x的低阶$n/2$位，c是y的高阶$n/2$位，d是y的低阶$n/2$位
  - 通过下式来计算$bc + ad$

$$bc + ad \;=\; ac + bd \;-\; (a-b)\,(c-d) \qquad \cancel{bc + ad = (a+c)(b+d) - ac - bd}$$

  - 递归地进行3次两个$n/2$位整数的乘法计算

$$m = \lceil n/2 \rceil$$
$$a = \lfloor x/2^m \rfloor \quad b = x \bmod 2^m$$
$$c = \lfloor y/2^m \rfloor \quad d = y \bmod 2^m$$

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

middle term

$$x\,y \;=\; (2^m\,a + b)\,(2^m\,c + d) \;=\; 2^{2m}\,ac + 2^m\,\overline{(bc + ad)} \;+\; bd$$
$$=\; 2^{2m}\,ac + 2^m\,(ac + bd \;-\; (a-b)(c-d)) + bd$$

1 1 3 2 3

Karatsuba算法

KARATSUBA-MULTIPLY(x, y, n)

IF $(n = 1)$
  RETURN $x \times y$.
ELSE
  $m \leftarrow \lceil n/2 \rceil$.
  $a \leftarrow \lfloor x/2^m \rfloor$; $b \leftarrow x \bmod 2^m$. ← $\Theta(n)$
  $c \leftarrow \lfloor y/2^m \rfloor$; $d \leftarrow y \bmod 2^m$.
  $e \leftarrow$ KARATSUBA-MULTIPLY(a, c, m).
  $f \leftarrow$ KARATSUBA-MULTIPLY(b, d, m). ← $3\,T(\lceil n/2 \rceil)$
  $g \leftarrow$ KARATSUBA-MULTIPLY$(|a-b|, |c-d|, m)$.
  Flip sign of g if needed.
  RETURN $2^{2m}\,e + 2^m\,(e + f - g) + f$. ← $\Theta(n)$

$$T(n) \;=\; \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3T(\lceil n/2 \rceil) \;+\; \Theta(n) & \text{if } n > 1 \end{cases}$$

$$\Longrightarrow \quad T(n) \;=\; \Theta(n^{\log_2 3}) \;=\; O(n^{1.585})$$

递归（分治法）

- 汉诺塔
- 归并排序
- 九连环
- 求解递归式
- 计算逆序数
- 随机快速排序
- 选择问题
- 最近点对
- 整数乘法
- 矩阵乘法

Dot product

Dot product. Given two length-n vectors a and b, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b \;=\; \sum_{i=1}^{n} a_i\, b_i$$

$$a = [\,.70 \quad .20 \quad .10\,]$$
$$b = [\,.30 \quad .40 \quad .30\,]$$
$$a \cdot b \;=\; (.70 \times .30) + (.20 \times .40) + (.10 \times .30) \;=\; .32$$

Remark. "Grade-school" dot product algorithm is asymptotically optimal.

Matrix multiplication

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

Grade-school. $\Theta(n^3)$ arithmetic operations.

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

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

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

Q. Is "grade-school" matrix multiplication algorithm asymptotically optimal?

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 ½n-by-½n blocks.
- Conquer: multiply 8 pairs of ½n-by-½n matrices, recursively.
- Combine: add appropriate products using 4 matrix additions.

n-by-n matrices

$$C = A \times B$$

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

½n-by-½n matrices

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

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

4 matrix additions (of ½n-by-½n matrices)

Running time. Apply Case 1 of the master theorem.

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

Strassen's trick

Key idea. Can multiply two 2-by-2 matrices via 7 scalar 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. $C_{12} = P_1 + P_2$

$= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$

$= A_{11} \times B_{12} + A_{12} \times B_{22}$. ✔

Strassen's trick

Key idea. Can multiply two ~~2-by-2~~ n-by-n matrices via 7 ~~scalar~~ ½n-by-½n matrix multiplications (plus 11 additions and 7 subtractions).

½n-by-½n matrices

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

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

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

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

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

Strassen's algorithm

assume n is a power of 2

STRASSEN(n, A, B)

IF $(n = 1)$ RETURN $A \times B$.

Partition A and B into ½n-by-½n 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$(n/2, (A_{11} - A_{21}), (B_{11} + B_{12}))$.

$7\,T(n/2) + \Theta(n^2)$

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

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

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

$\Theta(n^2)$

RETURN C.

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

Analysis of Strassen's algorithm

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

Pf.

- When n is a power of 2, apply Case 1 of the master theorem:

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

- When n is not a power of 2, pad matrices with zeros to be n'-by-n', where $n \le n' < 2n$ and n' is a power of 2.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Divide-and-conquer II: quiz 4

Suppose that you could multiply two 3-by-3 matrices with 21 scalar multiplications. How fast could you multiply two n-by-n matrices?

A. $\Theta(n^{\log_3 21})$

B. $\Theta(n^{\log_2 21})$

C. $\Theta(n^{\log_9 21})$

D. $\Theta(n^2)$