分治法的应用

快速排序

为什么需要快速排序?

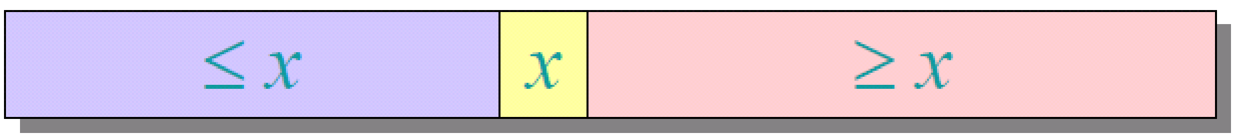
- 插入排序不好吗？
- 归并排序不是改进了吗？
- 问题是什么？

Quicksort

- Proposed by C.A.R. Hoare in 1962.
- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).

快速排序的描述

- **分解**：数组A[p …r]被划分成两个子数组A[p..q-1]和A[q+1…r],且A[q]为下标（标杆）元素。使得
  - A[p..q-1]都≤A[q]， A[q+1…r] ＞A[q]



- **解决**：递归地调用快速排序，对子数组A[p..q-1]和A[q+1…r],排序
- **合并**：因为两个子数组就地排序，将它们的合并不需要操作，整个数组已经排序了。

Pseudocode for quicksort

QUICKSORT(A, p, r)
  if $p < r$
    then $q \leftarrow$ PARTITION(A, p, r)
      QUICKSORT$(A, p, q-1)$
      QUICKSORT$(A, q+1, r)$

Initial call: QUICKSORT$(A, 1, n)$

Example of partitioning

6	10	13	5	8	3	2	11
i	j						

$x \leftarrow A[p]$ ▷ pivot $= A[p]$

$i \leftarrow p$

do if $A[j] \leq x$

then $i \leftarrow i + 1$

exchange $A[i] \leftrightarrow A[j]$

x	$\leq x$	$\geq x$	?
p	i	j	q

Example of partitioning

6	10	13	5	8	3	2	11

i $\longrightarrow$ j

do if $A[j] \le x$

then $i \leftarrow i + 1$

exchange $A[i] \leftrightarrow A[j]$

Example of partitioning

6	10	13	5	8	3	2	11

i $\longrightarrow j$

do if $A[j] \le x$

then $i \leftarrow i + 1$

exchange $A[i] \leftrightarrow A[j]$

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

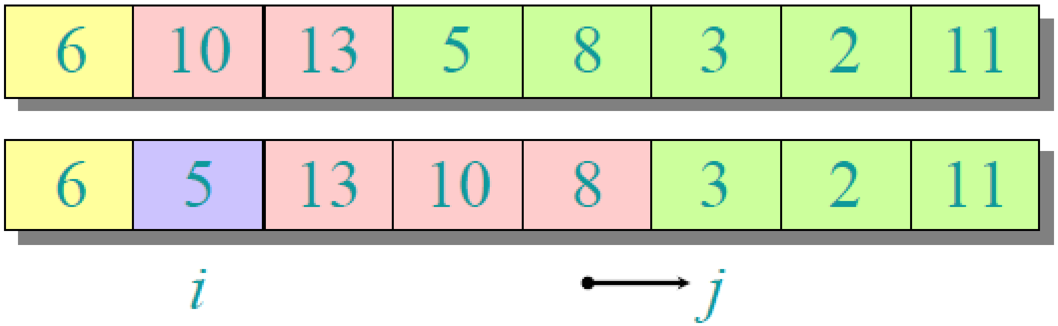
i j

Example of partitioning

6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11
	i			j			

Example of partitioning



Example of partitioning

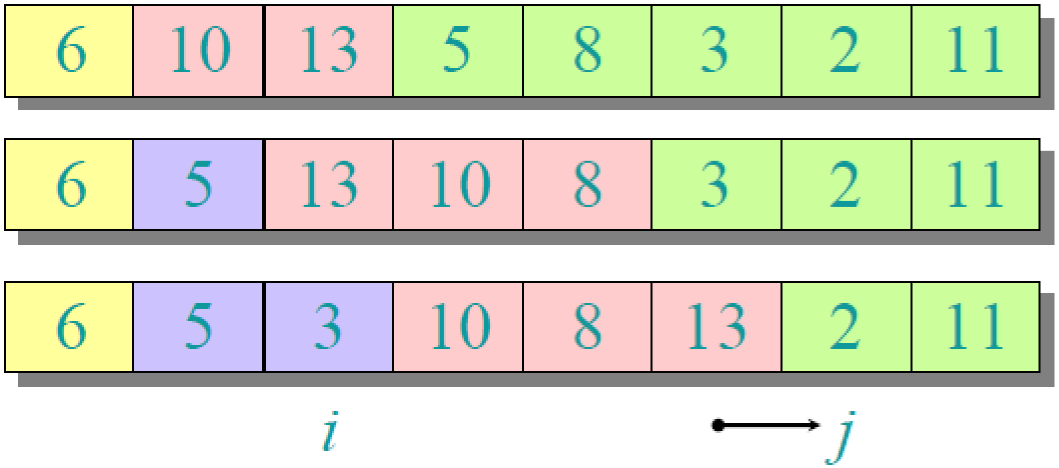
6	10	13	5	8	3	2	11

6	5	13	10	8	3	2	11

6	5	3	10	8	13	2	11

i j

Example of partitioning



Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
			i			j	

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
			i				j

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11

i $\longrightarrow j$

Example of partitioning

6	10	13	5	8	3	2	11
6	5	13	10	8	3	2	11
6	5	3	10	8	13	2	11
6	5	3	2	8	13	10	11
2	5	3	6	8	13	10	11

i

Partitioning subroutine

PARTITION(A, p, q) ▷ $A[p\,.\,.\,q]$

  $x \leftarrow A[p]$ ▷ pivot $= A[p]$

  $i \leftarrow p$

  for $j \leftarrow p + 1$ **to** q

    do if $A[j] \leq x$

      then $i \leftarrow i + 1$

        exchange $A[i] \leftrightarrow A[j]$

  exchange $A[p] \leftrightarrow A[i]$

  return i

Running time $= O(n)$ for n elements.

Invariant:

x	$\leq x$	$\geq x$	?

p i j q

Pseudocode for quicksort

QUICKSORT(A, p, r)
    if $p < r$
        then $q \leftarrow$ PARTITION(A, p, r)
            QUICKSORT(A, p, $q-1$)
            QUICKSORT(A, $q+1$, r)

Initial call: QUICKSORT(A, 1, n)

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- Let $T(n)$ = worst-case running time on an array of n elements.

Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \qquad \textit{(arithmetic series)} \end{aligned}$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$T(n)$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

$$cn$$

$$T(0) \qquad T(n-1)$$

Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$

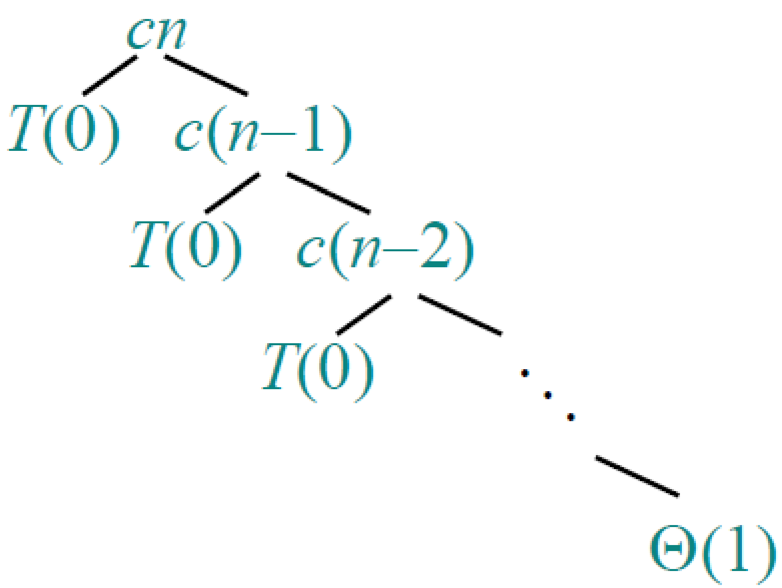
$$cn$$

$$T(0) \qquad c(n-1)$$

$$T(0) \qquad T(n-2)$$

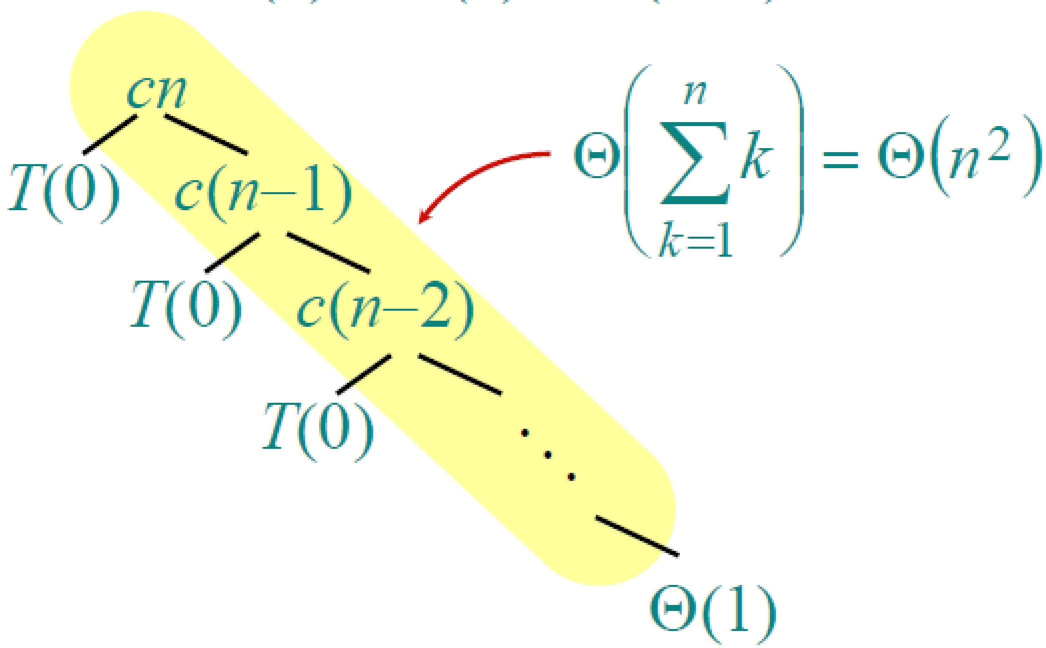
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



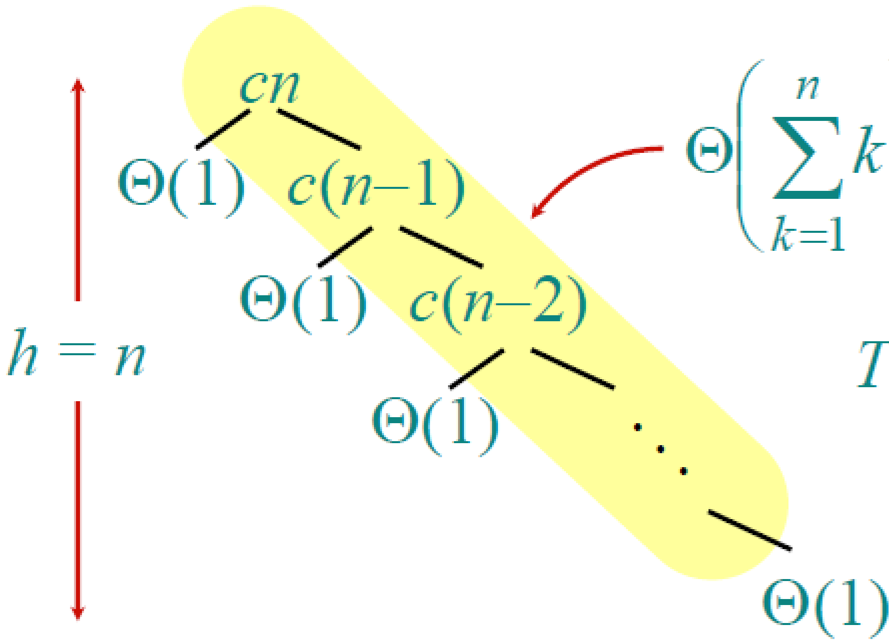
Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



Worst-case recursion tree

$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

$$\begin{aligned} T(n) &= \Theta(n) + \Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

Best-case analysis

(For intuition only!)

If we're lucky, PARTITION splits the array evenly:

$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&= \Theta(n \lg n) \qquad \text{(same as merge sort)}
\end{aligned}
$$

What if the split is always $\frac{1}{10} : \frac{9}{10}$?

$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

What is the solution to this recurrence?

Analysis of "almost-best" case

$T(n)$

Analysis of “almost-best” case

$$cn$$

$$T\left(\tfrac{1}{10}n\right) \qquad\qquad T\left(\tfrac{9}{10}n\right)$$

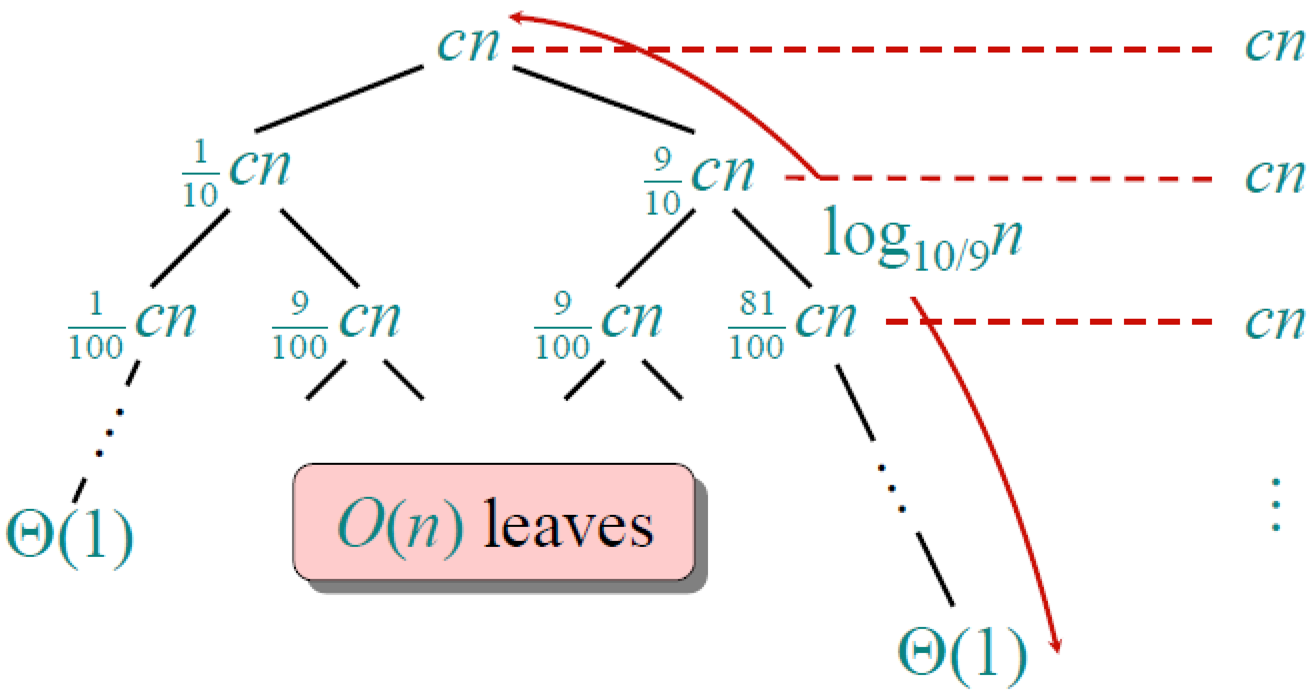
Analysis of "almost-best" case

$$cn$$

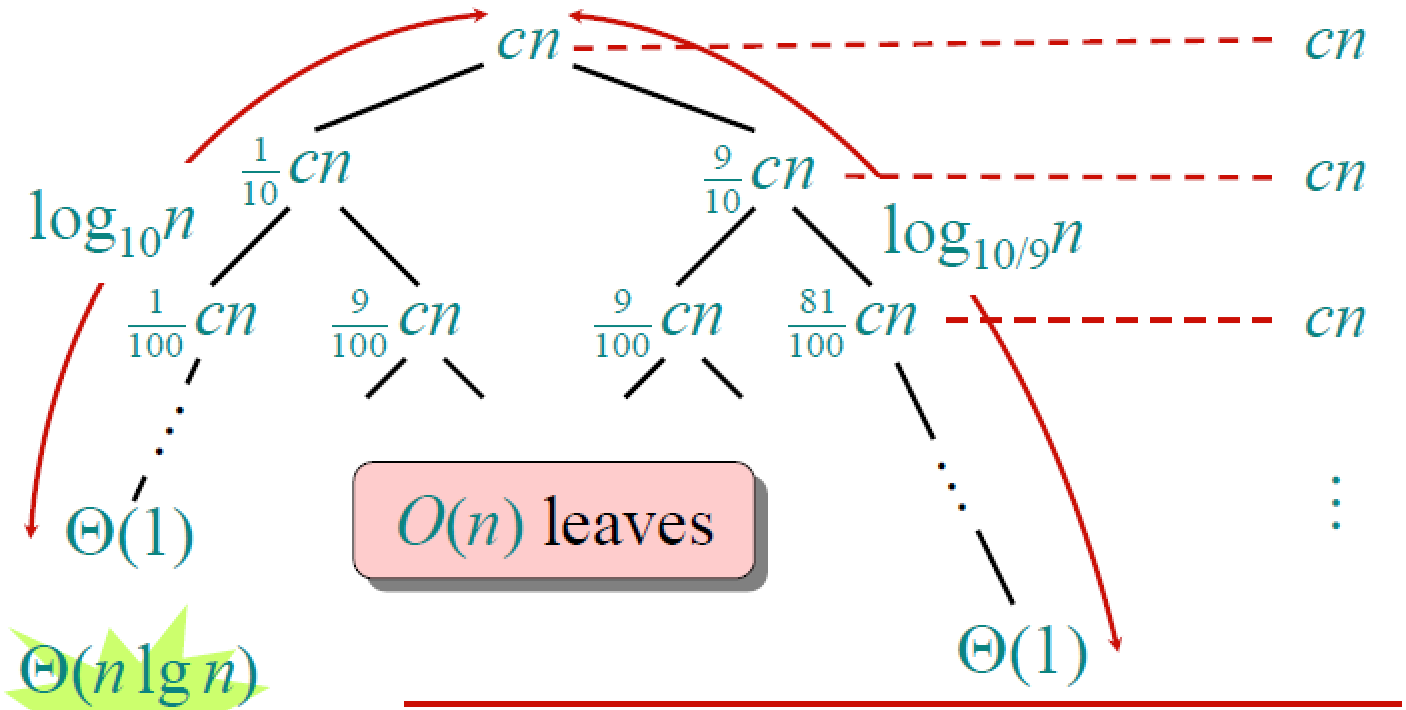
$$\frac{1}{10}cn \qquad \frac{9}{10}cn$$

$$T\left(\frac{1}{100}n\right) \; T\left(\frac{9}{100}n\right) \qquad T\left(\frac{9}{100}n\right) \; T\left(\frac{81}{100}n\right)$$

Analysis of “almost-best” case



Analysis of "almost-best" case



$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$

More intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ….

$$L(n) = 2U(n/2) + \Theta(n) \quad \textit{lucky}$$
$$U(n) = L(n-1) + \Theta(n) \quad \textit{unlucky}$$

Solving:

$$\begin{aligned} L(n) &= 2(L(n/2-1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2-1) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

Lucky!

How can we make sure we are usually lucky?

快速排序

可以看出：快速排序算法的性能取决于划分的对称性。通过修改算法**partition**，可以设计出采用随机选择策略的

还

划

可

- **最坏时间复杂度：$O(n^2)$**
- **平均时间复杂度：$O(n\log n)$**
- **稳定性：不稳定**

```
private static int randomizedPartition (int p, int r) {
    int i = random(p,r);
    MyMath.swap(a, i, p);
    return partition (p, r);
  }
```

Randomized quicksort

IDEA: Partition around a ***random*** element.

- Running time is independent of the input order.
- No assumptions need to be made about the input distribution.
- No specific input elicits the worst-case behavior.
- The worst case is determined only by the output of a random-number generator.

Randomized quicksort analysis

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$, since all splits are equally likely, assuming elements are distinct.

Analysis (continued)

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right).$$

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Take expectations of both sides.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

Linearity of expectation.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k \left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

Independence of X_k from other random choices.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(k) + T(n-k-1) + \Theta(n)\right)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E\left[T(k) + T(n-k-1) + \Theta(n)\right]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

Linearity of expectation; $E[X_k] = 1/n$.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E\left[X_k\big(T(k) + T(n-k-1) + \Theta(n)\big)\right]$$

$$= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)]$$

$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)$$

$$= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Summations have identical terms.

Hairy recurrence

$$E[T(n)] = \frac{2}{n}\sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(The $k = 0, 1$ terms can be absorbed in the $\Theta(n)$.)

Prove: $E[T(n)] \le a n \lg n$ for constant $a > 0$.

- Choose a large enough so that $a n \lg n$ dominates $E[T(n)]$ for sufficiently small $n \ge 2$.

Use fact: $\sum_{k=2}^{n-1} k \lg k \le \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ (exercise).

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n)$$

Substitute inductive hypothesis.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

Use fact.

Substitution method

$$E[T(n)] \le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n)$$

$$\le \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n)$$

$$= an\lg n - \left(\frac{an}{4} - \Theta(n)\right)$$

Express as ***desired*** – ***residual***.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=2}^{n-1} ak\lg k + \Theta(n) \\
&= \frac{2a}{n}\left(\frac{1}{2}n^2\lg n - \frac{1}{8}n^2\right) + \Theta(n) \\
&= an\lg n - \left(\frac{an}{4} - \Theta(n)\right) \\
&\le an\lg n\,,
\end{aligned}
$$

if a is chosen large enough so that $an/4$ dominates the $\Theta(n)$.

Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from ***code tuning***.
- Quicksort behaves well even with caching and virtual memory.

作业1：

- 挪动次数？
  - 插入排序，合并排序，快速排序
- 比较次数？
  - 都有哪些排序方法？比较排序有哪些？效率如何分析？

分治法的应用

选择问题\最小线性表选择

选择问题

- 问题：
  - 给定线性序集中n个元素和一个整数k，$1 \le k \le n$，要求找出这n个元素中第k小的元素。
- 思路1：
  - 先采用一种排序算法先将数组按不降的次序排好，然后从排好序的数组中检出第k小的元素。
- 分析算法复杂性：
  - 最坏情况下至少是O(nlogn).

3个特殊情况

- 找第1小元素
  - 即转换为找最小值元素问题
- 找第n小元素
  - 即转换为找最大值元素问题
- 找中间小元素
  - 找中位数，$k = (n+1)/2$问题

选择问题

Example

Select the $k = 7$th smallest:

6	10	13	5	8	3	2	11

$k = 7$

pivot

Partition:

2	5	3	6	8	13	10	11

$i = 4$

Select the $7 - 4 = 3$rd smallest recursively.

选择问题

给定线性序集中n个元素和一个整数k，1≤k≤n，要求找出这n个元素中第k小的元素

Type RandomizedSelect(Type a[], int p, int r, int k)
{
　　极值情况：数组中只有一个元素；
　　利用二分法进行划分，分成两个数组，左半部和右半部，左半部小于右半部。
　　统计出左半部的元素个数i，如果 **(k<=i) k**一定出现在左半部
　　否则，**k**在右半部，则问题变成了在右半部递归查找第**k-i**小元素。
}

想想这个算法的最坏情况下，是什么样子的？
是否可以借鉴快速排序的思路？

Intuition for analysis

(All our analyses today assume that all elements are distinct.)

Lucky:

$$T(n) = T(9n/10) + \Theta(n) = \Theta(n)$$

$n^{\log_{10/9} 1} = n^0 = 1$

CASE 3

Unlucky:

$$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$$

arithmetic series

Worse than sorting!

选择问题

给定线性序集中n个元素和一个整数k，1≤k≤n，要求找出这n个元素中第k小的元素

```
template<class Type>
Type RandomizedSelect(Type a[], int p, int r, int k)
{
    if (p==r) return a[p];
    int x=RandomizedPartition(a, p, r),
    i=x-p+1;
    if (k<=i) return RandomizedSelect(a, p, x, k);
    else return RandomizedSelect(a,x+1,r,k-i);
}
```

在最坏情况下，算法randomizedSelect需要$O(n^2)$计算时间
但可以证明，算法randomizedSelect可以在O(n)平均时间内找出n个输入元素中的第k小元素。

深入分析选择问题

上述算法的核心步骤是什么？

线性时间选择

如果能在线性时间内找到一个划分基准，使得按这个基准所划分出的2个子数组的长度都至少为原数组长度的ε倍($0<\varepsilon<1$是某个正常数)，那么就可以**在最坏情况下**用O(n)时间完成选择任务，**这是线性时间选择问题。**

例如，若$\varepsilon=9/10$，算法递归调用所产生的子数组的长度至少缩短1/10。所以，在最坏情况下，算法所需的计算时间T(n)满足递归式$T(n)\le T(9n/10)+O(n)$ 。

由此可得$T(n)=O(n)$。

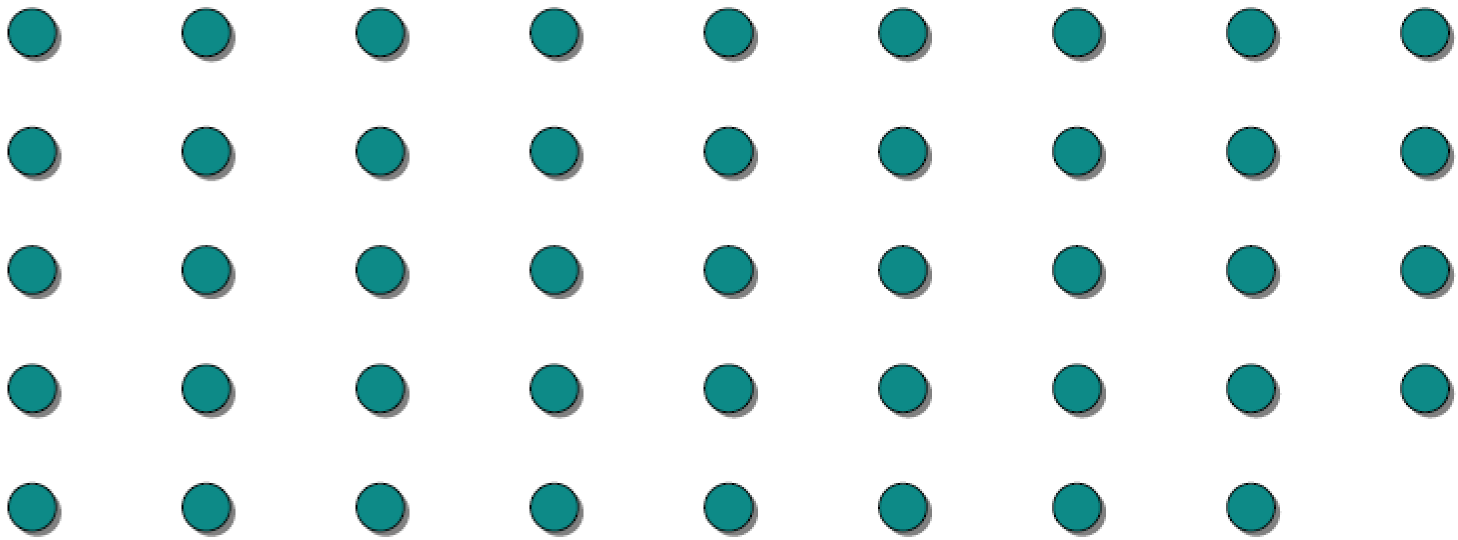
Pivot/划分基准的选择

1. 将n个输入元素划分成 n/5 个组，每组5个元素，只可能有一个组不是5个元素。用任意一种排序算法，将每组中的元素排好序，并取出每组的中位数，共 n/5 个。

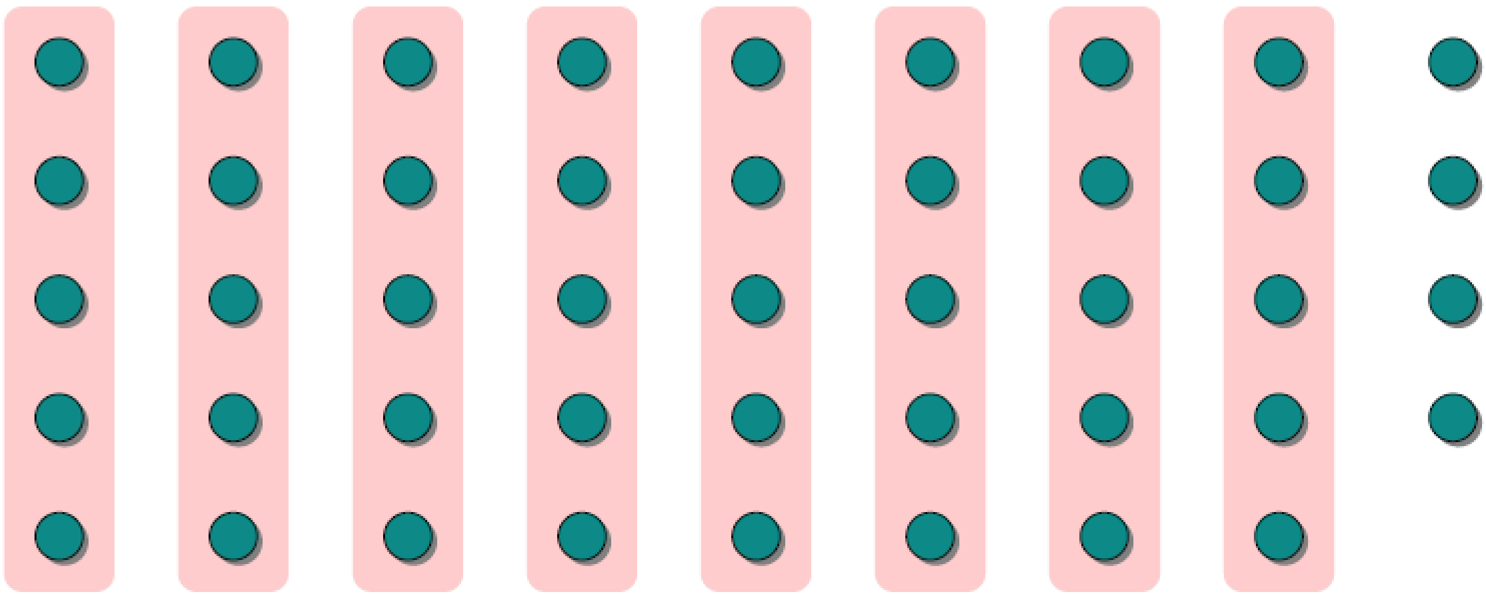
2. 递归调用select来找出这 n/5 个元素的中位数。如果 n/5 是偶数，就找它的2个中位数中较大的一个。

3. 以这个元素作为划分基准。

Choosing the pivot



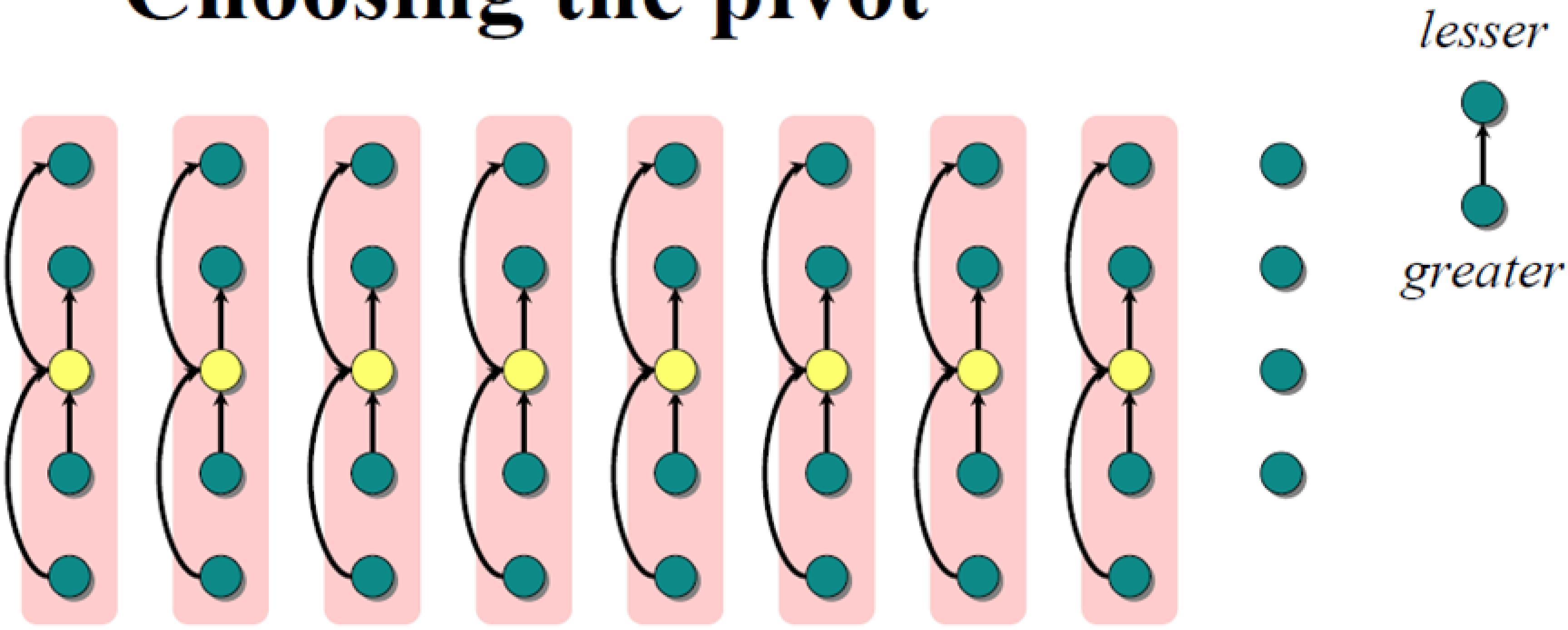
Choosing the pivot



1. Divide the n elements into groups of 5.

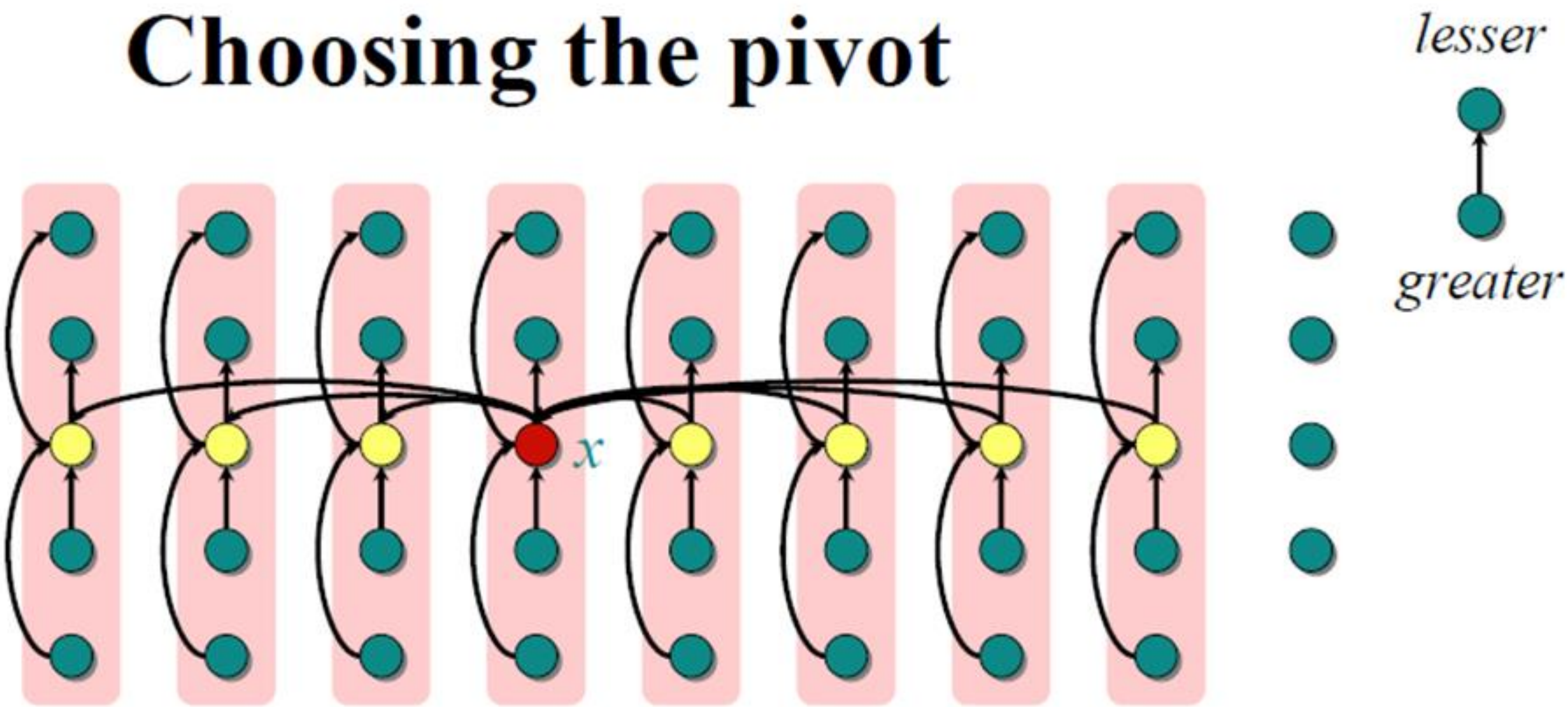
1. 将n个输入元素划分成 n/5 个组，每组5个元素，只可能有一个组不是5个元素。

Choosing the pivot



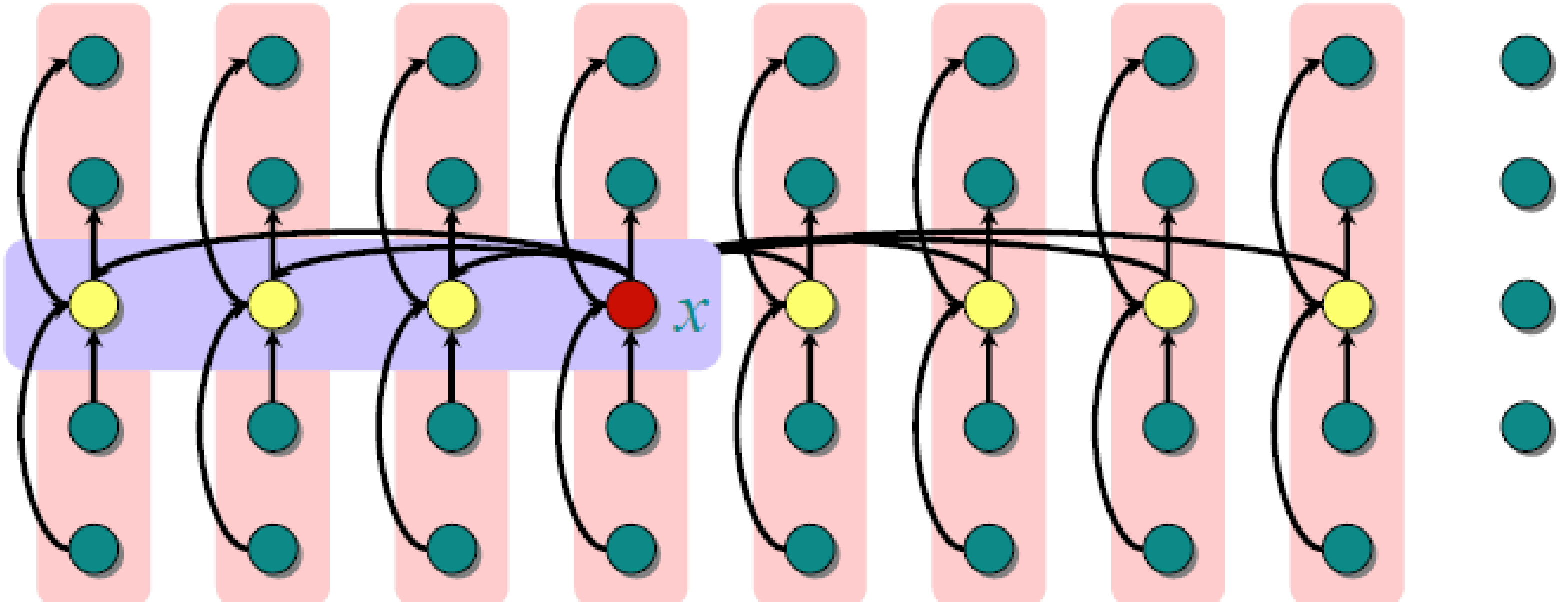
1. 将n个输入元素划分成 n/5 个组，每组5个元素. 用任意一种排序算法，将每组中的元素排好序，并取出每组的中位数，共 n/5 个。

Choosing the pivot



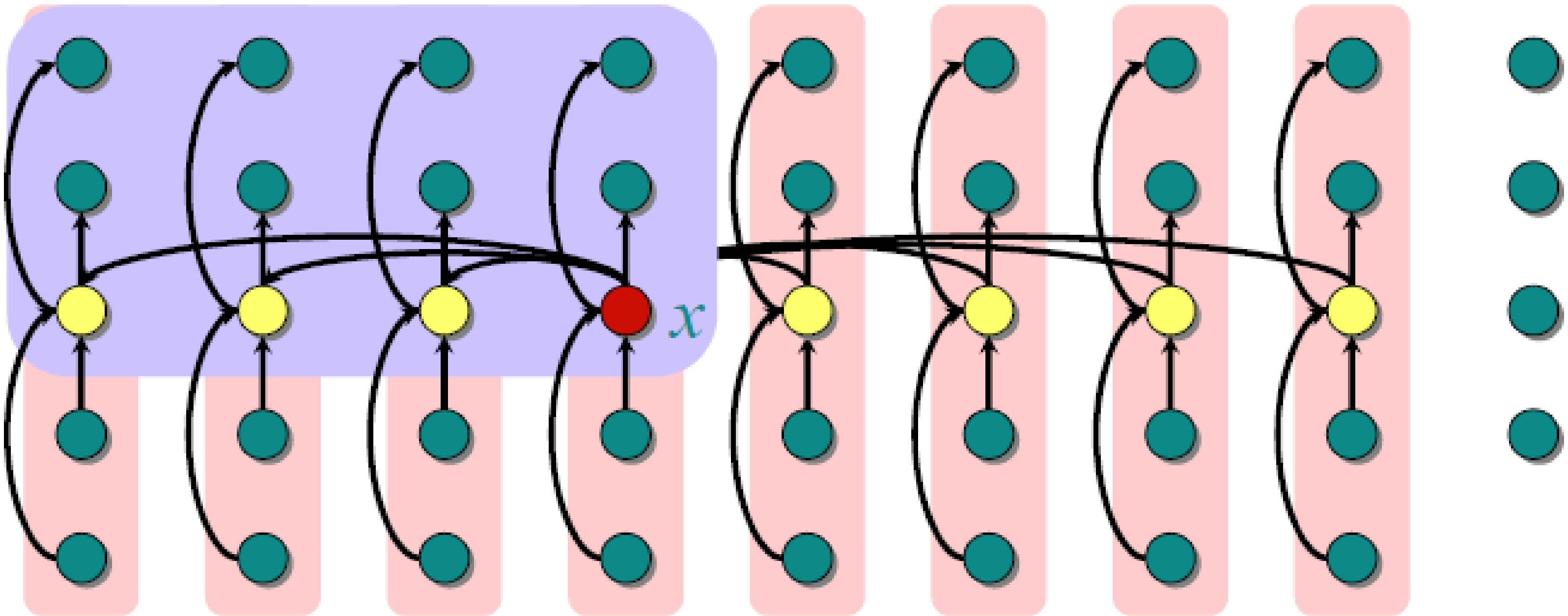
1. 将n个输入元素划分成 n/5 个组，每组5个元素 . 用任意一种排序算法，将每组中的元素排好序，并取出每组的中位数，共 n/5 个。
2. 递归调用select来找出这 n/5 个元素的中位数。如果 n/5 是偶数，就找它的2个中位数中较大的一个。
3. 以这个元素作为划分基准。

Analysis



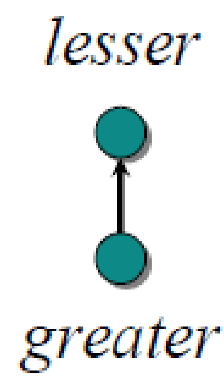
At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

Analysis (Assume all elements are distinct.)

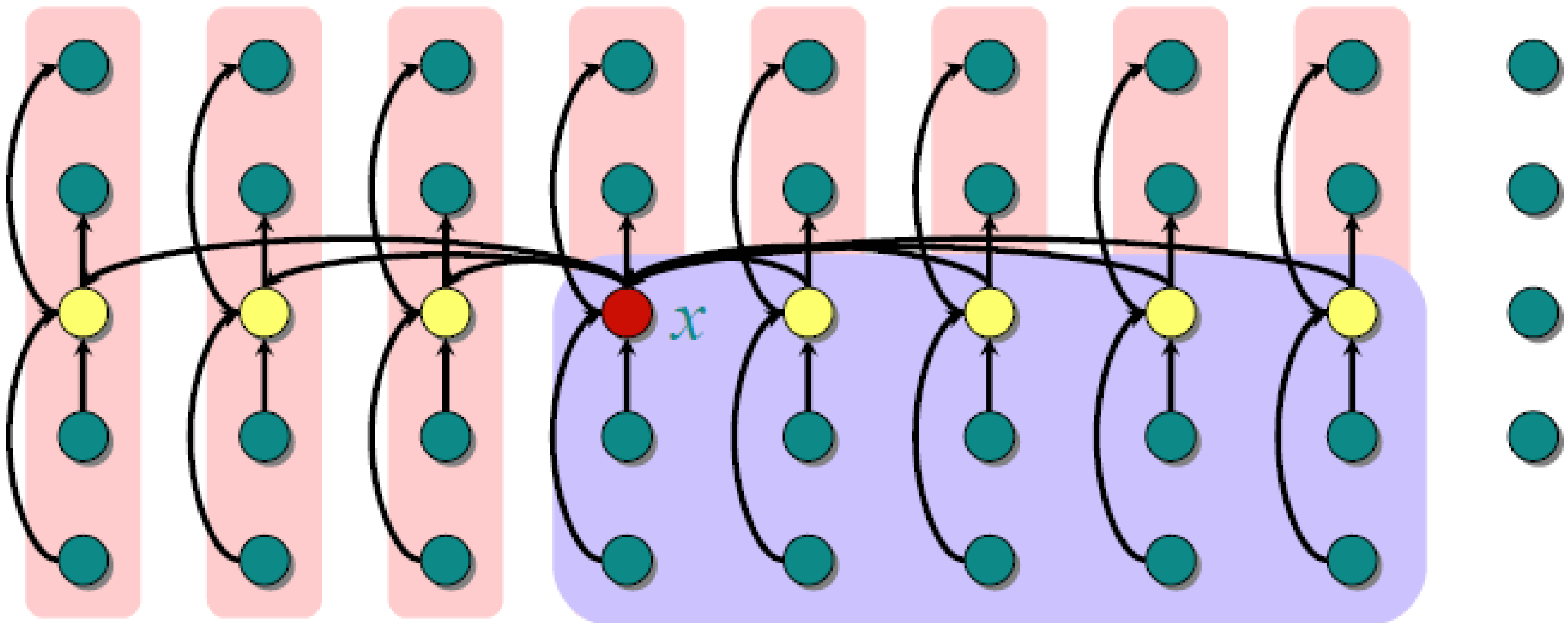


At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\leq x$.



Analysis (Assume all elements are distinct.)



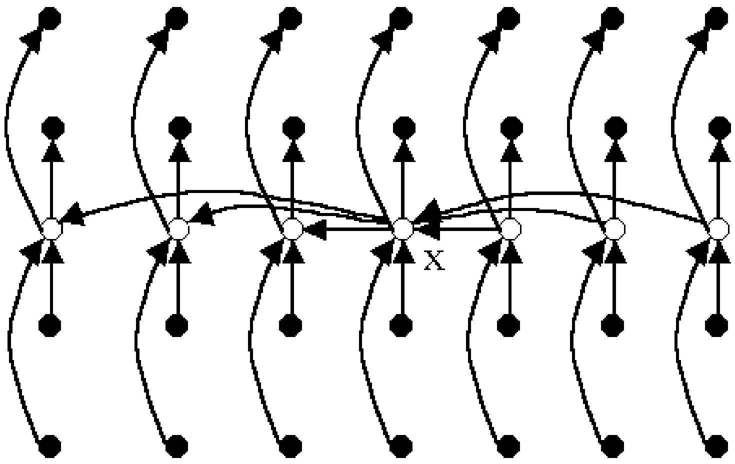
At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\le x$.
- Similarly, at least $3\lfloor n/10 \rfloor$ elements are $\ge x$.

lesser

greater

线性时间选择



设所有元素互不相同。在这种情况下，找出的基准x至少比3(n-5)/10个元素大，因为在每一组中有2个元素小于本组的中位数，而n/5个中位数中又有(n-5)/10个小于基准x。同理，基准x也至少比3(n-5)/10个元素小。而当n≥75时，3(n-5)/10≥n/4所以按此基准划分所得的2个子数组的长度都至少缩短1/4。

线性时间选择伪代码

```
Type Select(Type a[], int p, int r, int k)
{
      if (r-p<75) {
         用某个简单排序算法对数组a[p:r]排序;
         return a[p+k-1];
       };
      for ( int i = 0; i <= (r-p-4)/5; i++ )
         将a[p+5*i] 至 a[p+5*i+4]的第3小元素
         与a[p+i]交换位置;
      //找中位数的中位数，r-p-4即上面所说的n-5
      Type x = Select(a, p, p+(r-p-4)/5, (r-p-4)/10);
      int i=Partition(a, p, r, x),
      j=i-p+1;
      if (k<=j) return Select(a, p, i, k);
      else return Select(a, i+1, r, k-j);
}
```

复杂度分析

$$T(n) \le \begin{cases} C_1 & n < 75 \\ C_2 n + T(n/5) + T(3n/4) & n \ge 75 \end{cases}$$

$$T(n) = \boldsymbol{O}(\boldsymbol{n})$$

上述算法将每一组的大小定为5，并选取75作为是否作递归调用的分界点。这2点保证了T(n)的递归式中2个自变量之和n/5+3n/4=19n/20=εn，0<ε<1。这是使T(n)=O(n)的关键之处。当然，除了5和75之外，还有其他选择。

理论作业（DDL: 10.24）

- 分析以下排序算法的挪动次数和比较次数?
  - 插入排序，合并排序，快速排序

编程作业（DDL: 10.31）

- 归并排序 （迭代实现）
- 快速排序