选择
Selection

钮鑫涛
Nanjing University
2023 Fall

The slides are mainly adapted from the original ones shared by Chaodong Zheng and Kevin Wayne. Thanks for their supports!

Order Statistics and Selection

- Given a set of n items, the i^{th} **order statistic**（顺序统计量）of it is the i^{th} smallest element of it.
  - Minimum, maximum, median, …
- The Selection Problem: given a set A of n distinct numbers and an integer i, find the i^{th} order statistic of A.

Find Min/Max

- So easy, sequential scan and keep *min/max* till now…
- Make n - 1 comparisons, but is this the best we can do?
- Yes! Otherwise at least two elements could be the minimum.
  - Initially each element could be the minimum.
  - An adversary answers queries like "compare x with y".
  - Each comparison eliminates at most one element.

FindMin(A):

```
min := A[1]
for i := 2 to A.length
    if A[i] < min
        min := A[i]
return min
```

What if we want min *and* max?

- Go through the list twice, one for min and another for max.
- Can we do better? Surprisingly, yes!
  - Group items into pairs. (The first item becomes a "pair" if n is odd.)
  - For each of $\lceil n/2 \rceil$ pairs, find "local" min and max.
  - Among $\lceil n/2 \rceil$ "local" min, find global min; similarly find global max.

$\lfloor n/2 \rfloor$ comparisons

$\leq 2 \cdot \lfloor n/2 \rfloor$ comparisons

Total number of comparisons is at most $3 \cdot \lfloor n/2 \rfloor$

What if we want min *and* max?

- Is $3 \cdot \lfloor n/2 \rfloor$ the best we can do? Remarkably, yes!
  - An item has + mark if it can be max, and has - mark if it can be min.
  - Initially each item has both + and -.
  - An adversary answers queries like “compare x with y”.
  - The adversary can find input such that: at most $\lfloor n/2 \rfloor$ comparisons each removes two marks;
  - Every other comparison removes at most one mark.
  - In total need to remove $2n - 2$ marks.

So $\geq 2n - 2 - 2 * \lfloor n/2 \rfloor + \lfloor n/2 \rfloor = 2n - 2 - \lfloor n/2 \rfloor$ comparisons needed, which can be $3 \cdot \lfloor n/2 \rfloor$

General Selection Problem

- Find i^{th} smallest element (i.e., i^{th} order statistic)?
- Err… Sort them then return the i^{th} entry?
- Sure but this takes $\Omega(n \log n)$ time…

Can we be faster?

RndQuickSort(A):

if $A.size > 1$

  $q := RandomPartition(A)$

  $RndQuickSort(A[1, \ldots (q - 1)])$

  $RndQuickSort(A[(q + 1), \ldots, n])$

General Selection Problem

- What if $i = q$?
  - $A[q]$ is what we need.
- What if $i < q$?

Notice $A[1 \ldots (q\text{-}1)]$ contains the smallest q - 1 elements in A.

  - Find i^{th} order statistic in $A[1 \ldots (q - 1)]$.
- What if $i > q$?

Find $(i - q)^{th}$ order statistic in $A[(q+1) \ldots n]$.

RndQuickSort(A):

if $A.size > 1$

$q := RandomPartition(A)$

$RndQuickSort(A[1, \ldots (q - 1)])$

$RndQuickSort(A[(q + 1), \ldots, n])$

This is Reduce-and-Conquer!

Randomized Selection

A Reduce-and-Conquer Algorithm

```
RndSelect(A, i):
if A.size = 1
    return A[1]
else
    q := RandomPartition(A)
    if i = q
        return A[q]
    else if i < q
        return RndSelect(A[1 … (q-1)], i)
    else
        return RndSelect(A[(q + 1) … A.size], i - q)
```

- **Best-case** runtime? Choose the answer as the pivot in the first call (unlikely to happen).
  - $\Theta(n)$
- **Worst-case** runtime? Partition reduces array size by one each time (unlikely to happen).
  - $\geq cn + c(n-1) + \ldots + c(2) = \Theta(n^2)$
- What is the **average case**?

Average performance of Randomized Selection

- What's unlikely to happen is either get the exactly right pivot or reduces the size just by one. Instead, what's likely to happen is: partition process reduces problem size by a **constant** factor.

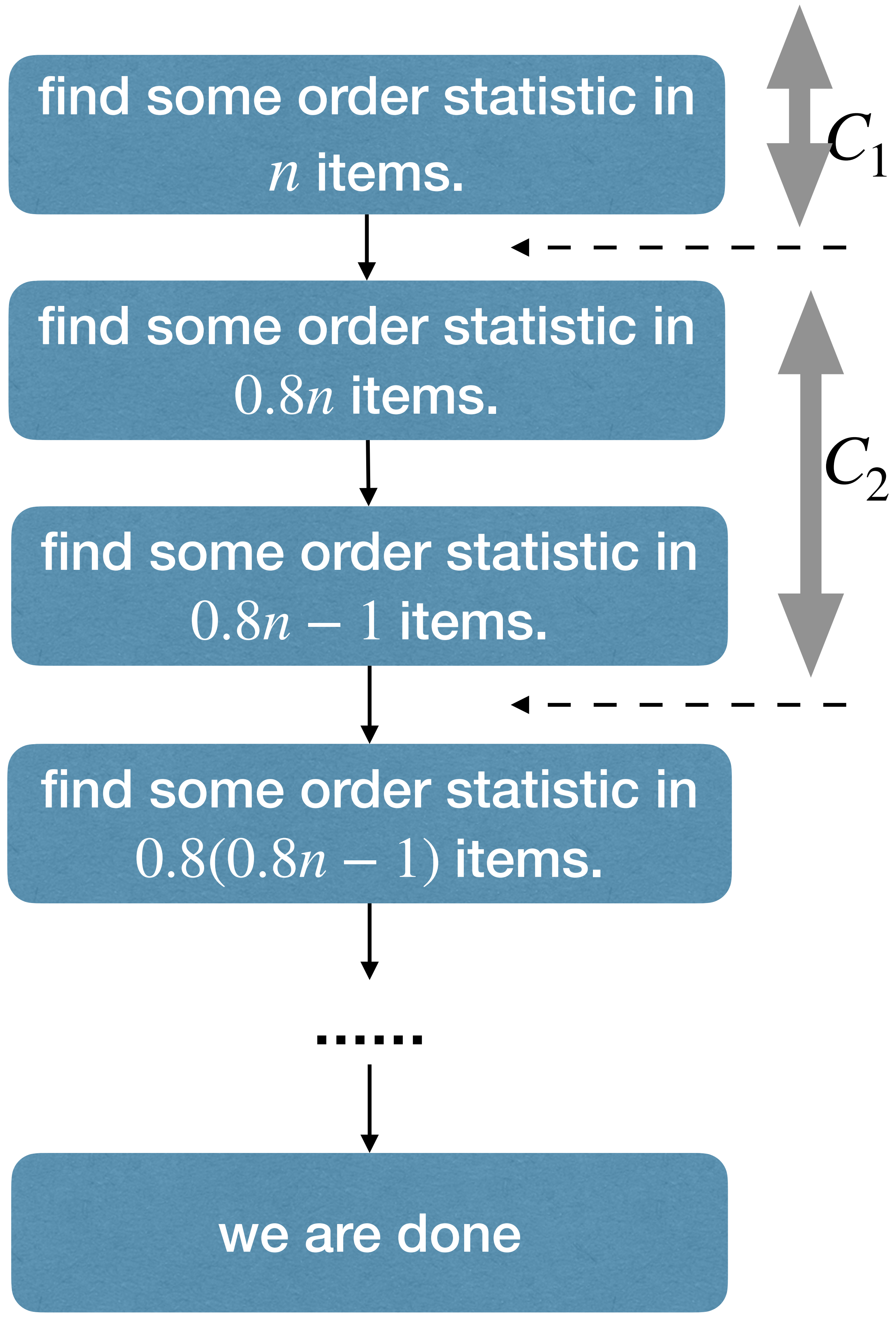
- Call a partition **good** if it reduces problem size to at most 0.8*input_size.

- Let the random variable C_i be the cost since the last good partition to the i^{th} good partition.

- At most $\log_{1.25} n$ good partitions can occur.

- $\mathbb{E}[C_i] \le \Theta(1) \cdot 0.8^{i-1} n$ **Why?**

- $\mathbb{E}[T(n)] \le \mathbb{E}\left[\sum_{i=1}^{log_{1.25}n} C_i\right] = \sum_{i=1}^{log_{1.25}n} \mathbb{E}[C_i] = O(n)$



RndQuickSort vs RndSelect

RndQuickSort(A):

if $A.size > 1$

  $q := RandomPartition(A)$

  $RndQuickSort(A[1, \ldots (q - 1)])$

  $RndQuickSort(A[(q + 1), \ldots, n])$

RndSelect(A, i):

if $A.size = 1$

  return $A[1]$

else

  $q := RandomPartition(A)$

  if $i = q$

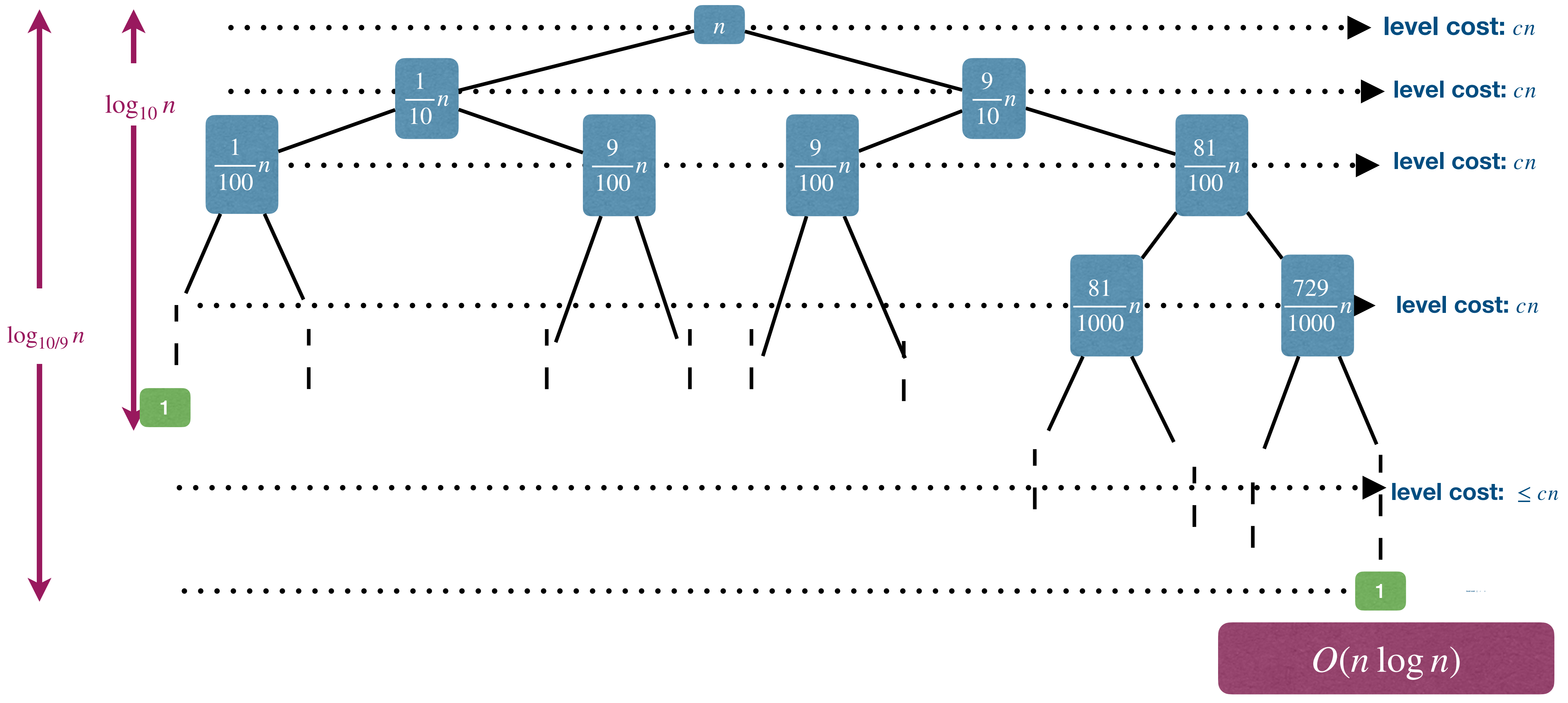
    return $A[q]$

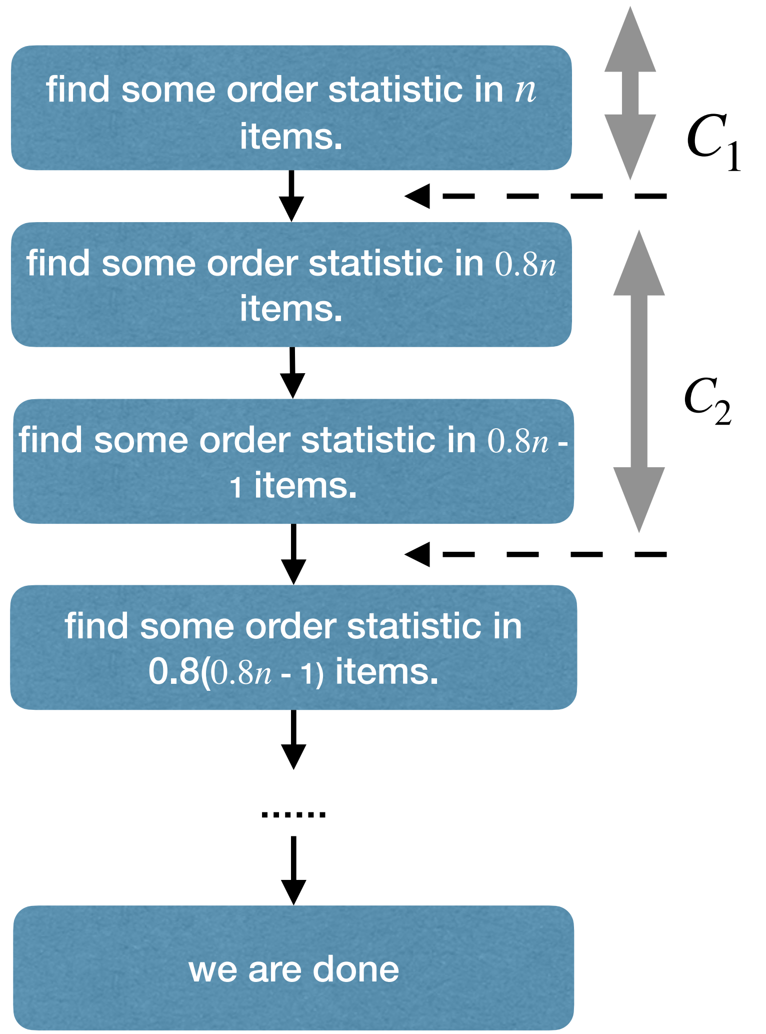
  else if $i < q$

    return $RndSelect(A[1 \ldots (q\text{-}1)], i)$

  else

    return $RndSelect(A[(q + 1) \ldots A.size], i - q)$





We are not done with selection…

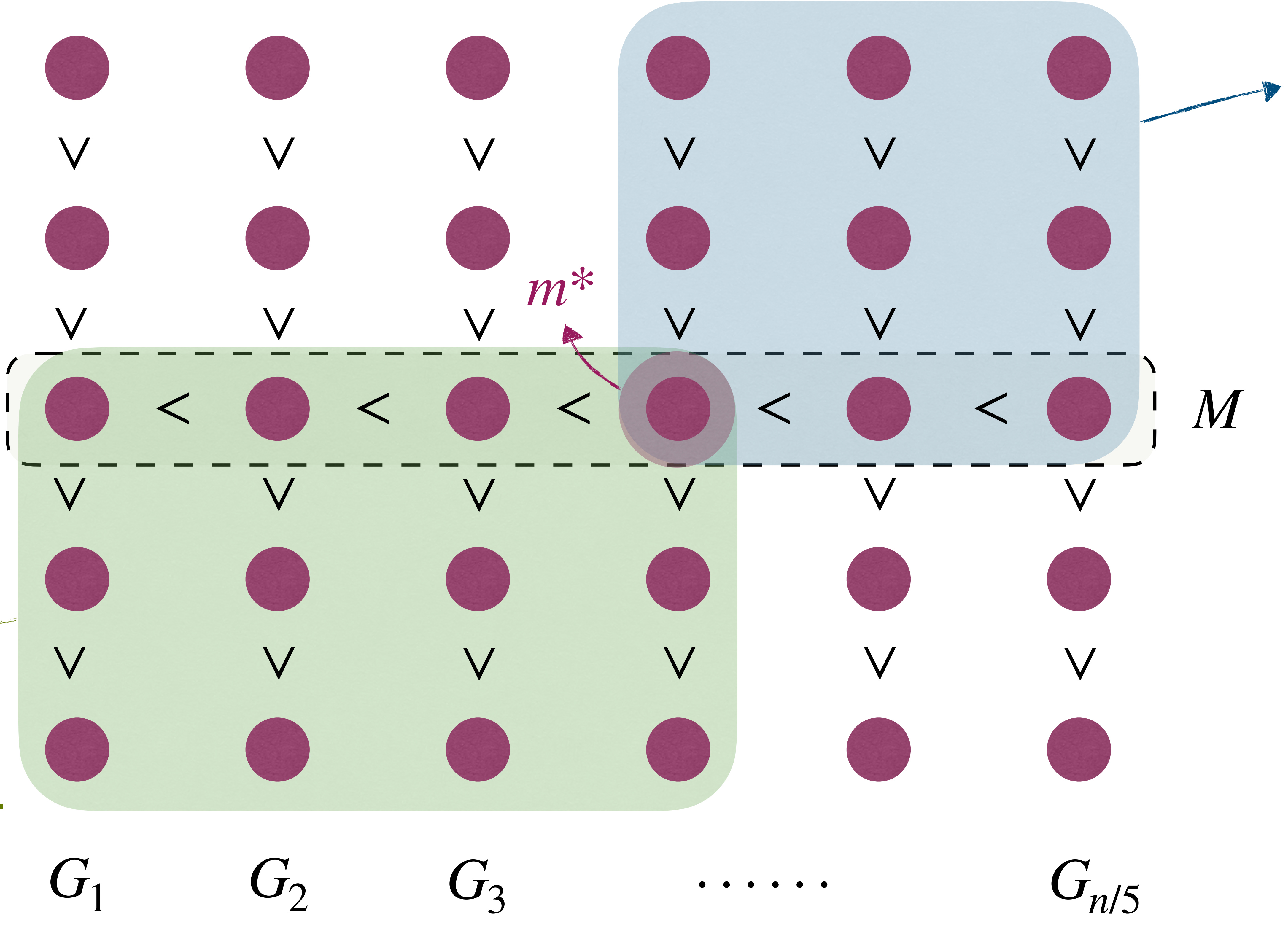
- Can we guarantee **worst-case** runtime of $O(n)$?
- The reason that `RndSelect` could be slow is that `RandomPartition` might return an **unbalanced** partition.
- Needs a partition procedure that guarantees to be **balanced**. (without using too much time; $O(n)$ time to be specific).

RndSelect(A, i):

```
if A.size = 1
    return A[1]
else
    q := RandomPartition(A)
    if i = q
        return A[q]
    else if i < q
        return RndSelect(A[1 … (q-1)], i)
    else
        return RndSelect(A[(q + 1) … A.size], i - q)
```

Median of medians

- Divide elements into $n/5$ groups, each containing 5 elements, call these groups $G_1, G_2, \ldots G_{n/5}$.
- Find the medians of these $n/5$ groups, let M be this set of medians.
- Find the median of M, call it m^*.



Partition using m^* as pivot is good: the smaller split has $\geq 0.3n$ items.

Finding median of medians

- Divide elements into $n/5$ groups, each containing 5 elements, call these groups $G_1, G_2, \ldots G_{n/5}$.

  Trivial, $O(n)$ time

- Find the medians of these $n/5$ groups, let M be this set of medians.

  Sort each group, then find the medians.
  Cost is $(n/5) \cdot \Theta(1) = \Theta(n)$.

- Find the median of M, call it $m*$.
  - Idea: Use `QuickSelect`, recursively.

Finding median of medians

QuickSelect(A, i):

```
if A.size = 1
    return A[1]
else
    m := MedianOfMedians(A)
    q := PartitionWithPivot(A, m)
    if i = q
        return A[q]
    else if i < q
        return QuickSelect(A[1…(q-1)], i)
    else
        return QuickSelect(A[(q+1)…A.size, i - q])
```

MedianOfMedians(A):

```
if A.size =1
    return A[1]
<G_1, G_2, . . . G_{n/5}> := CreateGroups(A)
for i := 1 to n/5
    Sort(G_i)
M := GetMediansFromSortedGroups(G_1, G_2, . . . G_{n/5})
return QuickSelect(M, (n/5)/2)
```

$O(n)$ — the cost of MedianOfMedians(A) together with the non-recursive steps of MedianOfMedians

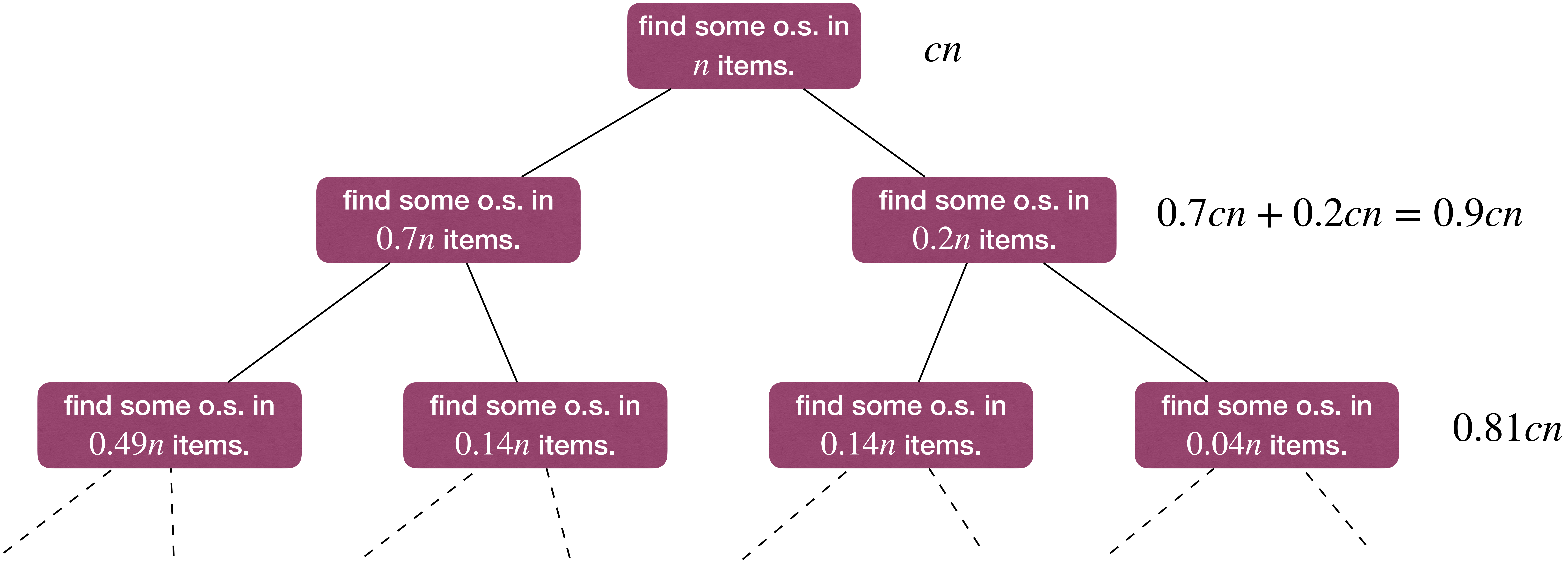
$T(0.2n)$ — return $QuickSelect(M, (n/5)/2)$

M is $\frac{1}{5}$ of A !

$T(0.7n)$ — the recursive QuickSelect calls

$$T(n) \leq T(0.7n) + T(0.2n) + O(n)$$

Time complexity



$O(n)$ in total

Time complexity

QuickSelect(A, i):

```
if A.size = 1
    return A[1]
else
    m := MedianOfMedians(A)
    q := PartitionWithPivot(A, m)
    if i = q
        return A[q]
    else if i < q
        return QuickSelect(A[1…(q-1)], i)
    else
        return QuickSelect(A[(q+1)…A.size], i - q])
```

MedianOfMedians(A):

```
if A.size =1
    return A[1]
<G_1, G_2, ... G_{n/5}> := CreateGroups(A)
for i := 1 to n/5
    Sort(G_i)
M := GetMediansFromSortedGroups(G_1, G_2, ... G_{n/5})
return QuickSelect(M, (n/5)/2)
```

- $T(n) \leq T(0.7n) + T(0.2n) + O(n)$
- $T(n) = O(n)$

You can verify this by the substitution method. (I.e., assume $T(n) \leq cn$ and then verify.)

Complexity of general selection

- `QuickSelect` uses $O(n)$ time/comparisons.
- Solving general selection needs at least n - 1 comparisons.
  - Since finding min/max needs at least n - 1 comparisons.
- So the lower and upper bounds match asymptotically.
- But if we care about constants, needs (much) more efforts.

Further reading

- [CLRS] Ch.9