Do not distribute course material

You may not and may not allow others to reproduce or distribute lecture notes and course materials publicly whether or not a fee is charged.

Divide and Conquer Continued

A way to design an algorithm

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively.

Base case: If the subproblems are small enough, just solve them by brute force.

Combine the subproblem solutions to give a solution to the original problem.

- Closest pair of points in the plane
- Quick Sort
- Quick Select
- Select: Deterministic Linear Selection

Section 33.4 in CSRS third edition

traffic-control systems.

robotics

Pair share: Solve 1-Dimensional case: $p_1, p_2, \ldots, p_n \in \mathbb{R}$
Find pair to minimize $|p_i - p_j|$

computer vision

integrated circuits verification

Closest Pair of Points in the Plane

page 1039 in CLRS

traffic-control systems.

robotics

computer vision

integrated circuits verification

Pair share: Solve 1-Dimensional case: $p_1, p_2, \ldots, p_n \in \mathbb{R}$
Find pair to minimize $|p_i - p_j|$

# of Points

ane

Classic Problem from Geometry!

$$P = \{p_1, p_2, \ldots, p_n\} \qquad p_i = (x_i, y_i)$$

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

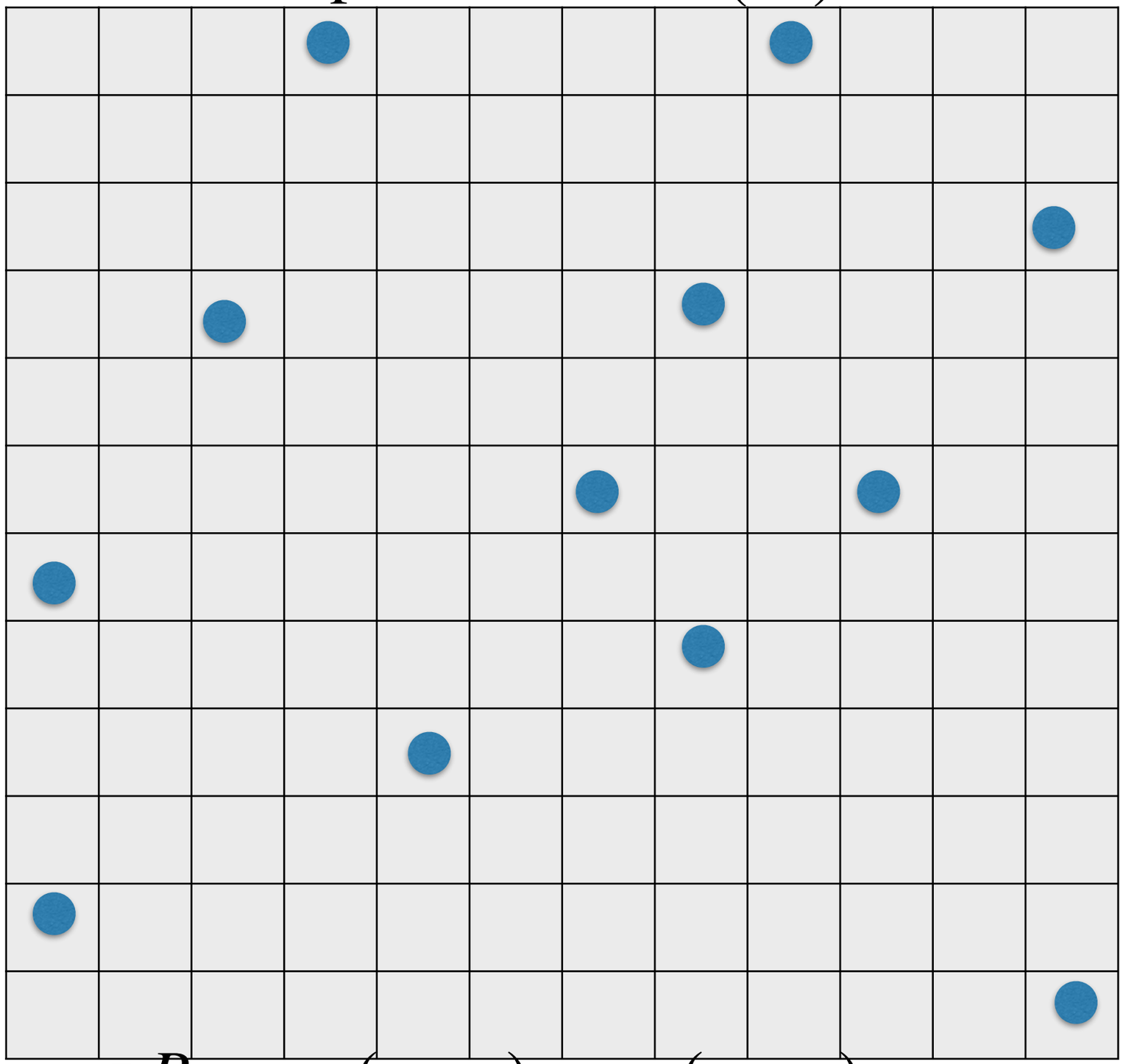
Find $p_i, p_j \in P$ that minimizes $d(p_i, p_j)$

Divide and Conquer!

Finding the closest pair of points

P set of $n \geq 2$ points

$O(n^2)$?



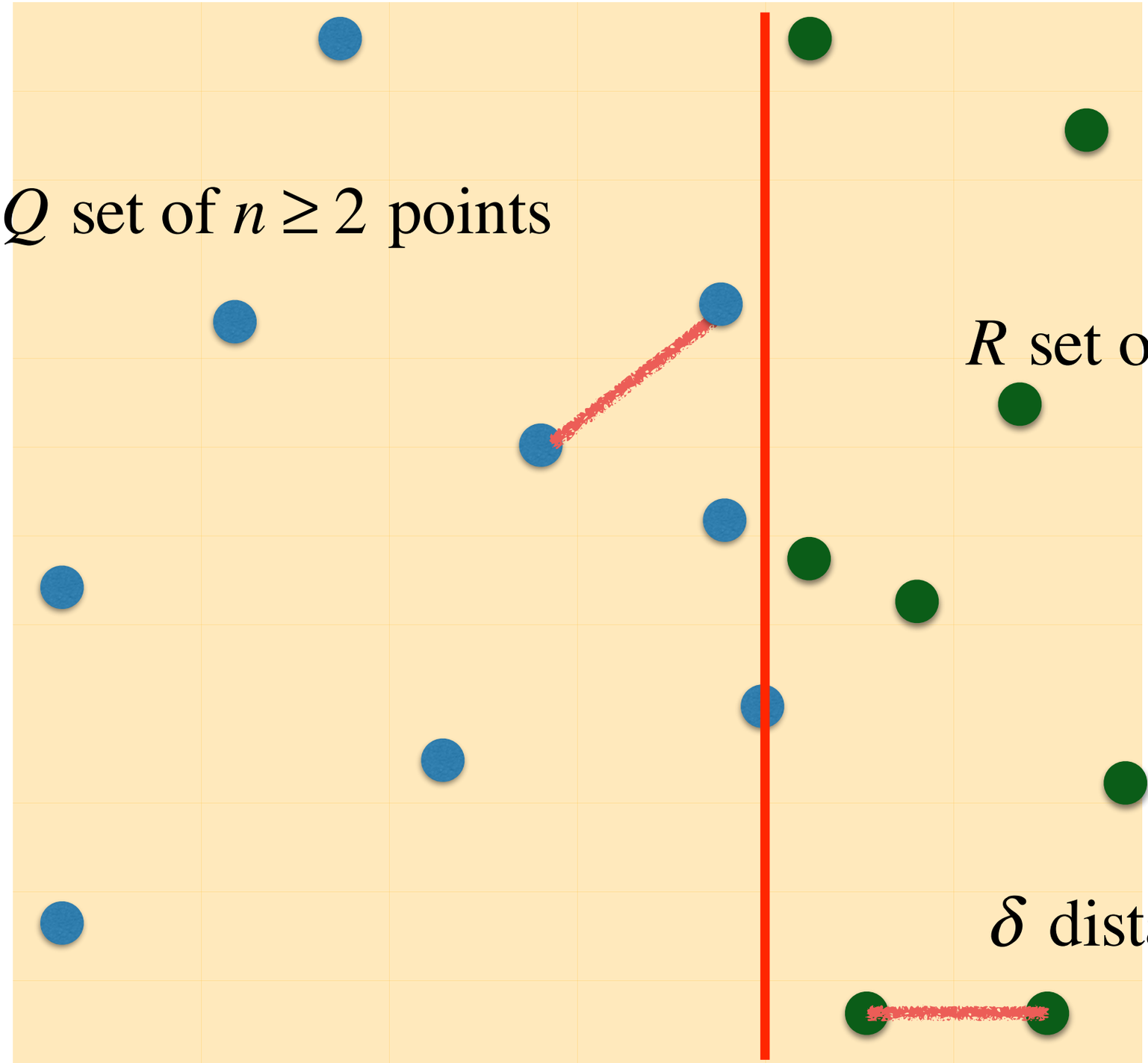
Pair share

$p_1, p_2 \in P,\ p_1 = (x_1, y_1), p_2 = (x_1, y_2)$

and the distance is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Divide and Conquer!

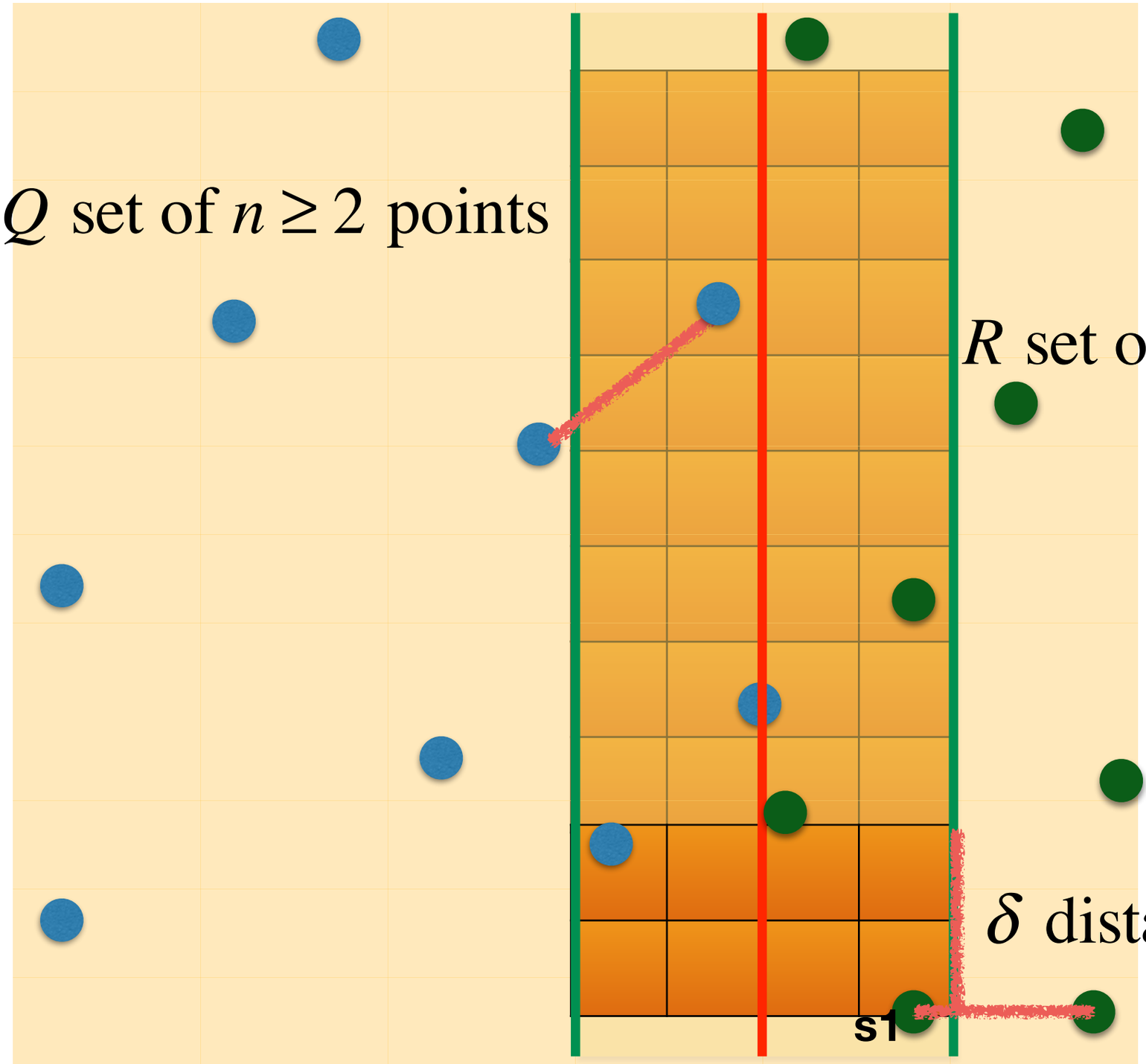
For two points to be closer than δ distance apart they must be closer than δ in x and y coordinates

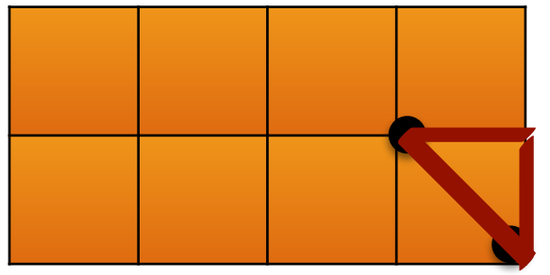


- **Minimum Distance in R**: Every pair of points in set R has a minimum distance of at least δ
- **Minimum Distance in Q**: Similarly, every pair of points in set Q is separated by a distance of at least δ

Divide and Conquer!

For two points to be closer than δ distance apart they must be closer than δ in x and y coordinates





At most one point per box.
The maximum distance between two points in the same box is:

$$\sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \sqrt{2}\frac{\delta}{2} < \delta$$

A point located within the rectangle may not necessarily have a distance smaller than δ from s1.

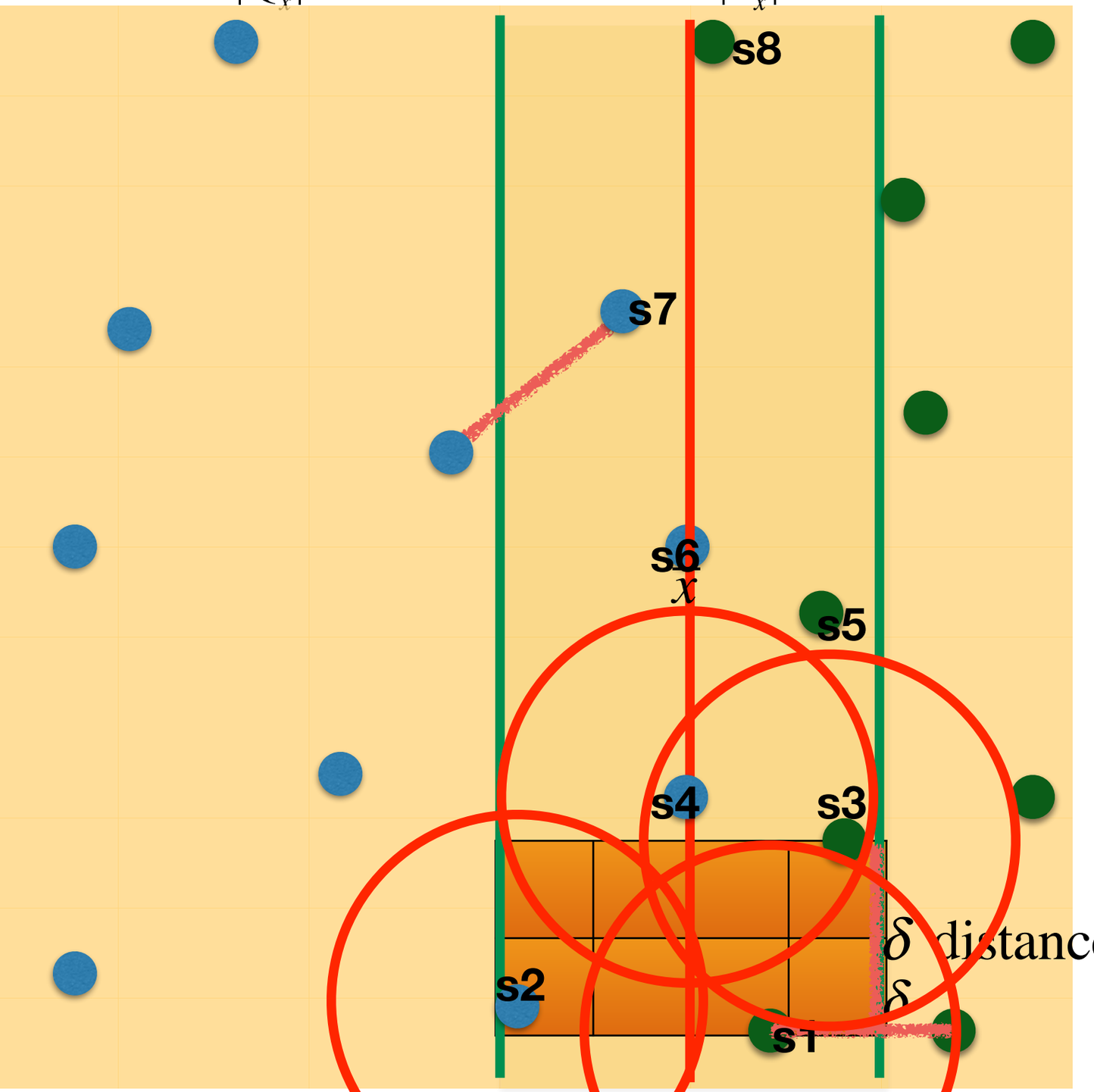
$$P_x = Q_x \cup R_x$$

Q_x set of $n/2$ points

$|Q_x| \geq 2$

R_x set of $n/2$ points

$|R_x| \geq 2$



$T(n) = 2T(n/2) + O(n \log n)$ is $O(n \log^2 n)$

CLOSEST_PAIR_REC(P_x)

if $|P_x| \leq 3$

solve and **return**_____

CONSTRUCT Q_x and R_x

(q_1, q_2) = CLOSEST_PAIR_REC(Q_x)

(r_1, r_2) = CLOSEST_PAIR_REC(R_x)

$\delta = \min\{ d(q_1, q_2), d(r_1, r_2) \}$

$\bar{x}$ = max x-coordinate in Q_x

CONSTRUCT $Y = (s_1, s_2, \ldots, s_m)$

where $s_i = (x_i, y_i) \in P_y$ and x_i is within

δ distance of the line

for $i = 1$ **to** m

for $j = 1$ **to** $\min(7, m-i)$

if $d(s_i, s_{i+j}) < \delta$

$(s^*_1, s^*_2) = (s_i, s_{i+j})$

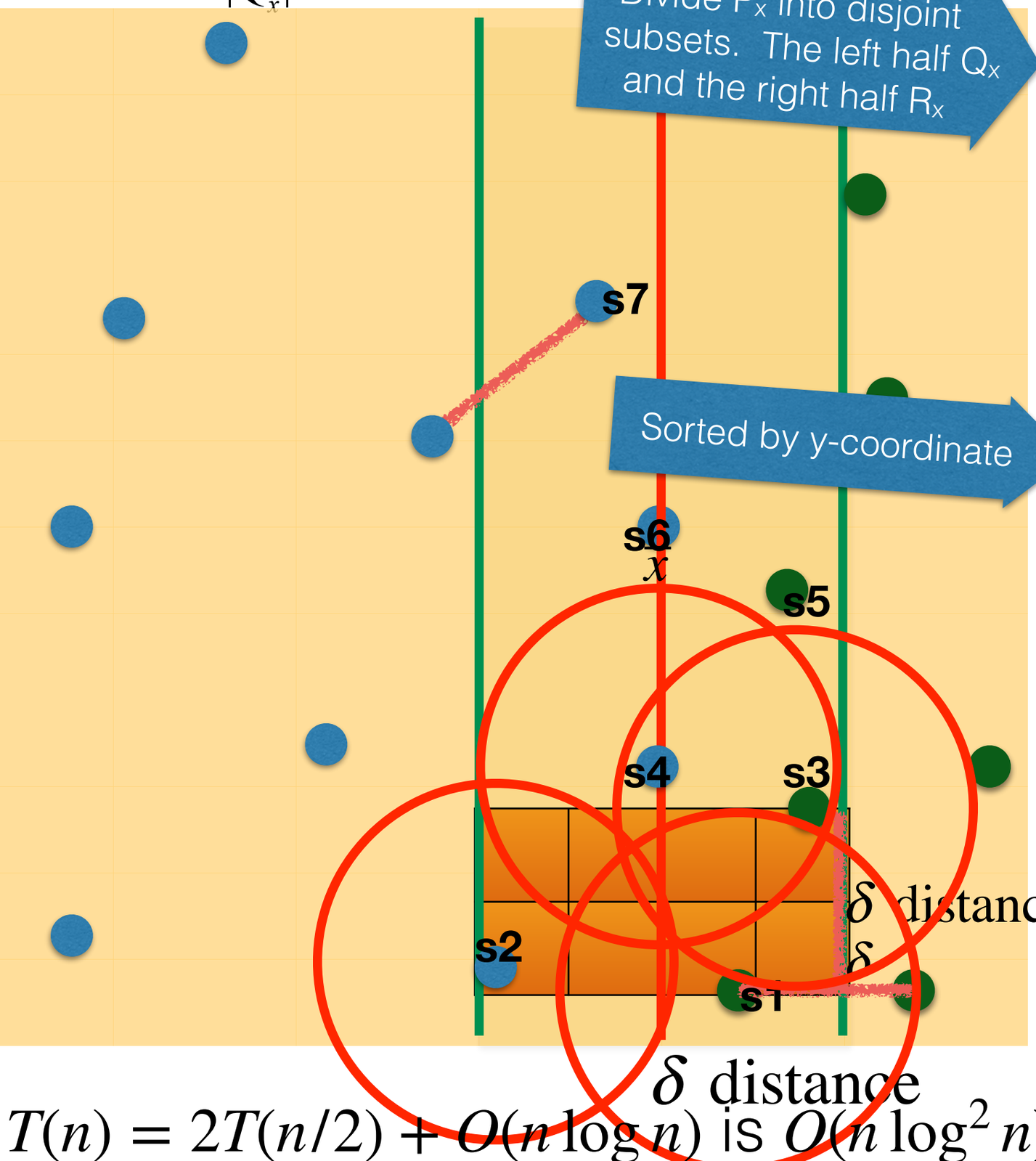
$\delta = d(s^*_1, s^*_2)$

return _____

$$P_x = Q_x \cup R_x$$

Q_x set of $n/2$ points $\quad$ R_x set of $n/2$ points

$|Q_x| \geq 2$




```
CLOSEST_PAIR_REC(Px)
if |Px| ≤ 3 solve directly and return_____
CONSTRUCT Qx and Rx
(q1, q2) = CLOSEST_PAIR_REC(Qx)
(r1, r2) = CLOSEST_PAIR_REC(Rx)
δ = min{ δ(q1, q2), δ(r1, r2) }
x̄ = max x-coordinate in Qx
CONSTRUCT Y = (s1, s2, . . . , sm)
where si = (xi, yi) ∈ Py and xi is within
    δ distance of the line
for i = 1 to m
  for j = 1 to min(7, m-i)
    if d( si, si+j) < δ
      (s*1, s*2) = ( si, si+j)
      δ = d(s*1, s*2)
return _____
```

points sorted by x-value

Points sorted by y-value that are within δ distance of the center

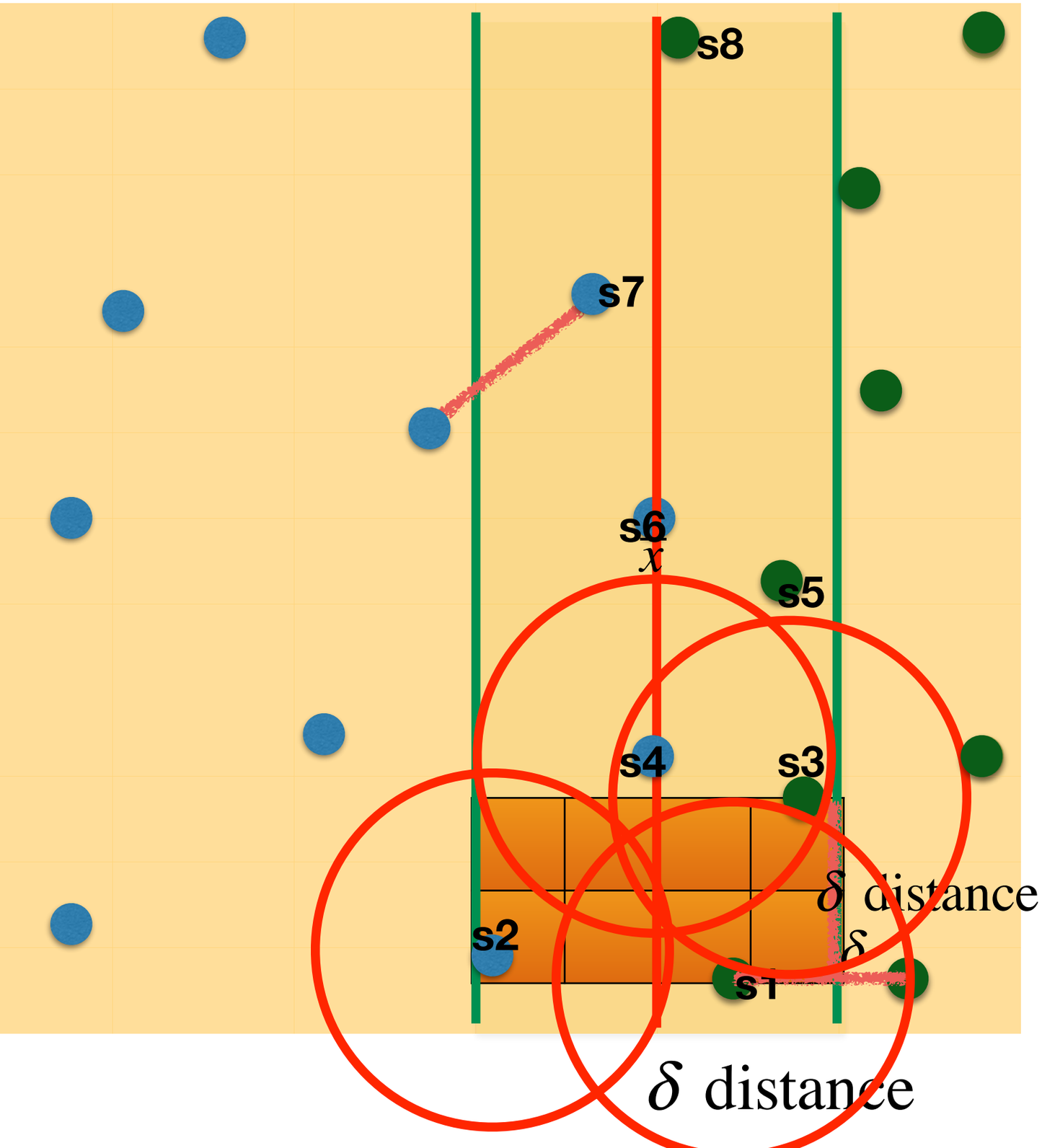
$T(n) = 2T(n/2) + O(n \log n)$ is $O(n \log^2 n)$

Closest Pair Of Points
Running time?

$$T'(n) = 2T(n/2) + O(n \log n)$$

$$T'(n) = O(n \log^2 n)$$

Different parameters - otherwise the same



CLOSEST_PAIR_REC(P_x, P_y)

if $|P_x| \leq 3$

solve and **return**_____

CONSTRUCT Q_x, Q_y and R_x, R_y

(q_1, q_2) = CLOSEST_PAIR_REC(Q_x, Q

(r_1, r_2) = CLOSEST_PAIR_REC(R_x, R

δ = min{ $d(q_1, q_2)$, $d(r_1, r_2)$ }

$\bar{x}$ = max x-coordinate in Q_x

CONSTRUCT $Y = (s_1, s_2, \ldots, s_m)$

where $s_i = (x_i, y_i) \in P_y$ and x_i is within δ distance of $\bar{x}$

for i =1 **to** m

for j = 1 **to** min(7,m-i)

if $d(s_i, s_{i+j}) < \delta$

$(s^*_1, s^*_2) = (s_i, s_{i+j})$

$\delta = d(s^*_1, s^*_2)$

return _____

Q set of $n/2$ points

$|Q| \geq 2$

R set of $n/2$ points

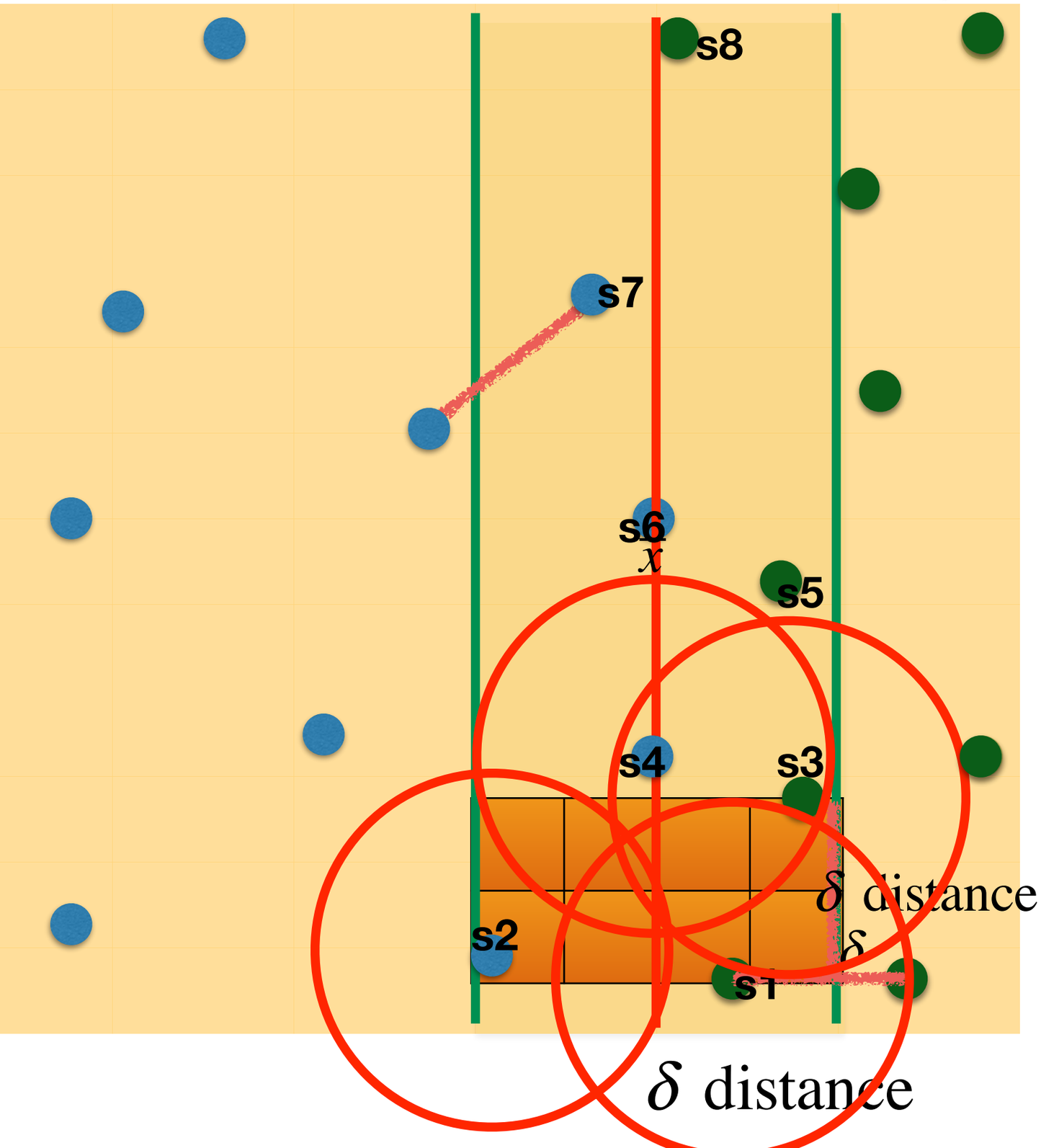
$|R| \geq 2$

The points in Px and Py are the same points just in a different order

points sorted by x-value

points sorted by y-value

points sorted by y-value that are within δ distance of the center



CLOSEST [illegible] EC(P_x, P_y)

if |P[illegible]

solve and [illegible]

CONSTRUCT Q_x, Q_y and R_x, R_y

(q_1, [illegible] CLOSEST_PAIR_REC(Q_x, Q[illegible]

([illegible] OSEST_PAIR_REC(R_x, R_y

δ = [illegible] d(r_1, r_2) }

$\bar{x}$ = max [illegible] te in Q_x

CONSTRUCT $Y = (s_1, s_2, \ldots, s_m)$

where $s_i = (x_i, y_i) \in P_y$ and x_i is within δ distance of $\bar{x}$

for i =1 **to** m

  for j = 1 **to** min(7,m-i)

    if d(s_i, s_{i+j}) < δ

      (s^*_1, s^*_2) = (s_i, s_{i+j})

      δ = d(s^*_1, s^*_2)

return _____

Closest Pair Of Points
Running time?

$$T'(n) = 2T(n/2) + O(n \log n)$$

$$T'(n) = O(n \log^2 n)$$

$$T(n) = \begin{cases} 2T(n/2) + O(n) & \text{if } n > 3 \\ O(1) & \text{if n} \le 3 \end{cases}$$

$$T(n) = O(n \log n)$$

Master theorem

1. $f(n) = O(n^{\log_b a - \epsilon})$ constant $\epsilon > 0$

   $T(n) = \Theta(n^{\log_b a})$

2. $f(n) = \Theta(n^{\log_b a} \log^k n)$ constant k ≥ 0

   $T(n) = \Theta(n^{\log_b a} \log^{k+1}(n))$

3. $f(n) = \Omega(n^{\log_b a + \epsilon})$

   constant $\epsilon > 0$ and

   $f(n)$ satisfies the regularity condition

   $T(n) = \Theta(f(n))$

Quick Sort
Divide and Conquer

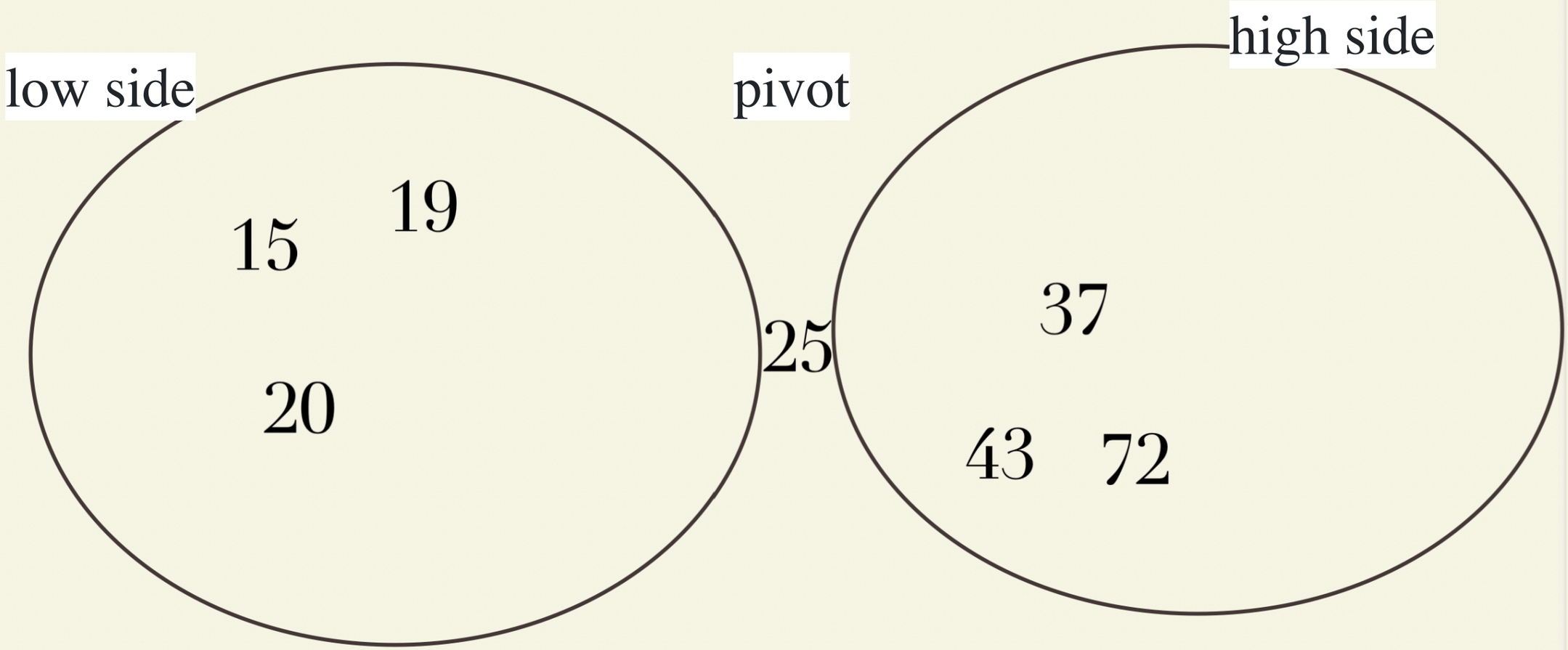
sort A[p .. r]

Divide: Partition A[p .. r], into two (possibly empty) subarrays A[p .. q-1] and A[q+1 .. r] such that each element in the first subarray A[p .. q-1] is at most each element in the second subarray A[q+1 .. r]

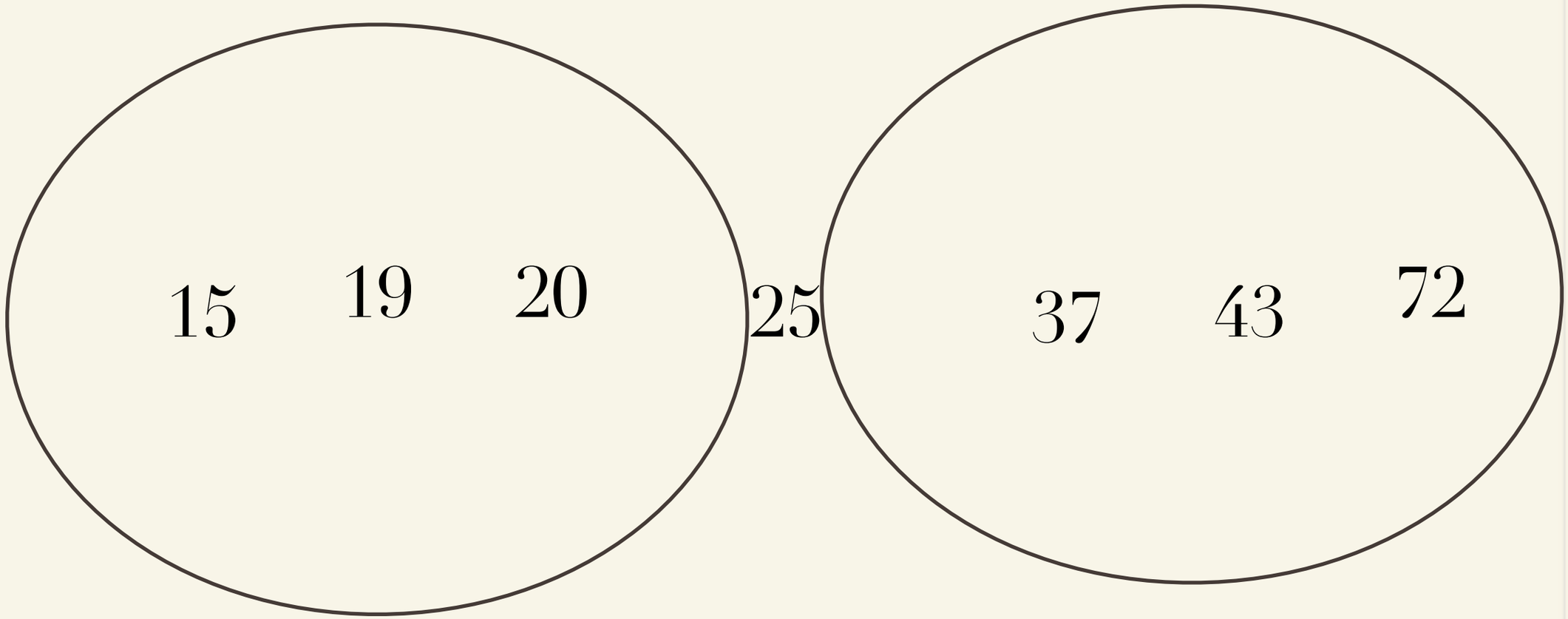
Conquer: Sort the two subarrays by recursive calls to QUICKSORT.

Combine: No work is needed to combine the subarrays, because they are sorted

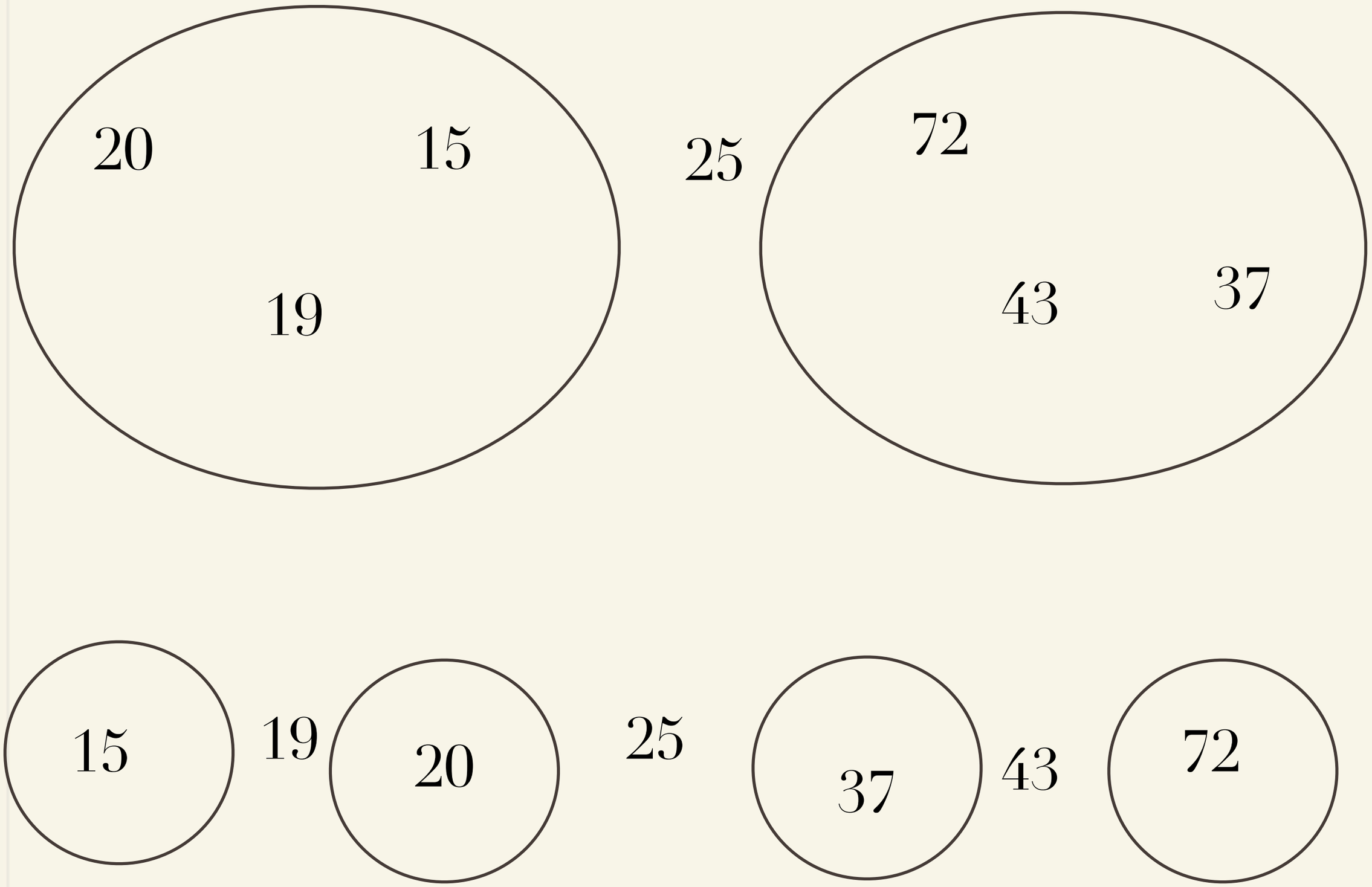
Reduce the original problem into subproblems that are easier to solve.



Solve the subproblems, and the combine to solve the original problem



The subproblems are solved recursively



QuickSort

43	37	25	19	4	72	15	20

SELECT PIVOT; PARTITION

19	4	15

<= pivot

20

pivot

37	72	25	43

>= pivot

} divide

SORT EACH PIECE (Recursively)

4	15	19

20

done!

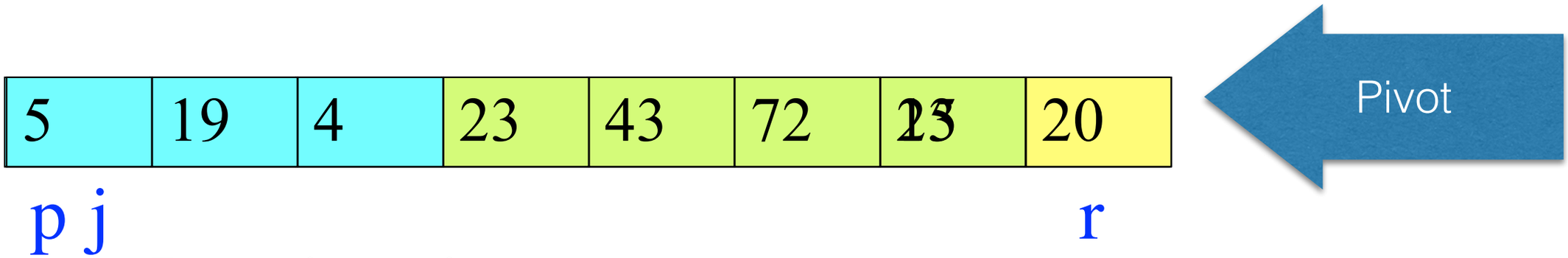
25	37	43	72

} conquer

} combine

QUICKSORT(A, 1, n)

```
QUICKSORT(A, p, r)
  if p < r
    q = PARTITION(A,p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q +1, r)
```



Loop invariant:

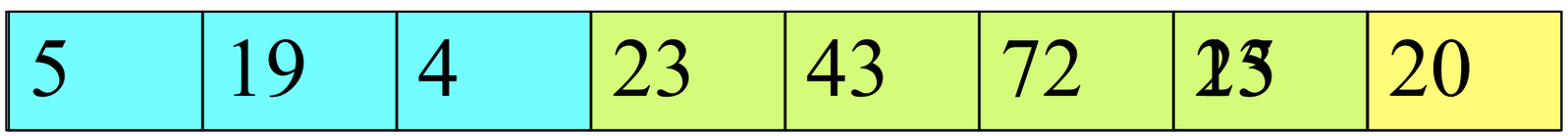
1. All entries in A[p..i] are <= pivot
2. All entries in A[i+1..j-1] are > pivot
3. A[r] is the pivot

QUICKSORT(A, 1, n)

```
QUICKSORT(A, p, r)
  if p < r
    q = PARTITION(A,p, r)
    QUICKSORT(A, p, q - 1)
    QUICKSORT(A, q +1, r)
```

```
PARTITION(A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r -1
     if A[j ] ≤ x
        i = i +1
        exchange A[i ] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

A[p..i] ≤ x,
A[i+1..j-1] >x



Loop invariant:

1. All entries in A[p..i] are <= pivot
2. All entries in A[i+1..j-1] are > pivot
3. A[r] is the pivot

Correctness (sketch) page 173 for a full proof

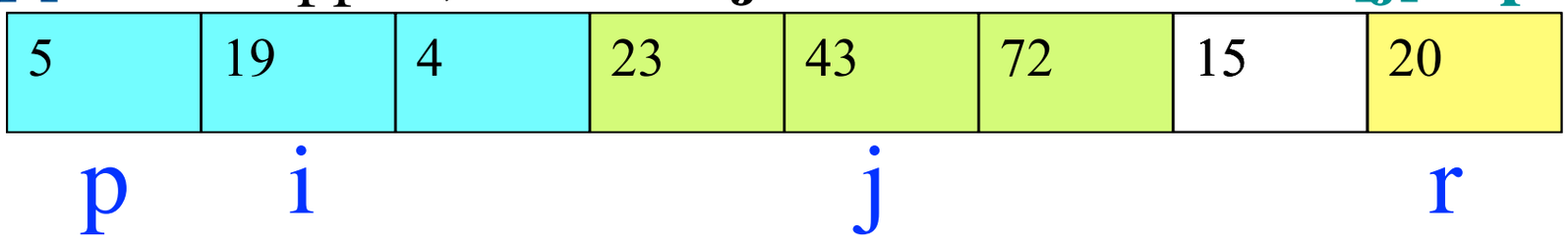
Loop invariant:

1. All entries in A[p..i] are <= pivot.
2. All entries in A[i+1..j-1] are > pivot.
3. A[r] is the pivot.

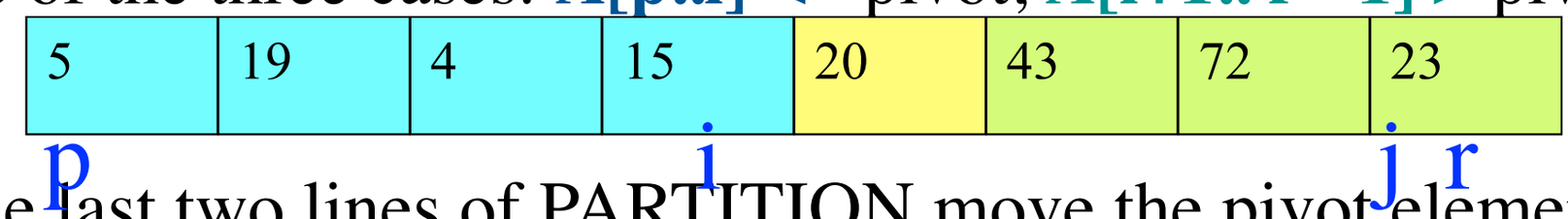
Initialization: Before the loop starts, all the conditions of the loop invariant is satisfied because r is the pivot and the subarrays **A[p..i]** and **A[i+1.. j -1]** are empty.

```
PARTITION(A, p, r)
  x = A[r]
  i = p - 1
  for j = p to r -1
     if A[j ] ≤ x
        i = i +1
        exchange A[i ] with A[j]
  exchange A[i + 1] with A[r]
  return i + 1
```

Maintenance: While the loop is running, if **A[j] <= pivot**, increment **i,** then **A[j]** and **A[i]** are swapped, and then **j** is incremented. If **A[j] > pivot**, then increment only **j** .



Termination: When the loop terminates, j = r, so all elements in A are partitioned into one of the three cases: **A[p..i] <=** pivot, **A[i+1.. r - 1] >** pivot, and **A[r]** = pivot.



The last two lines of PARTITION move the pivot element from the end of the array to between the two subarrays. This is done by swapping the pivot and the first element of the second subarray, i.e., by swapping **A[i+1]** and **A[r]**

Running Time?

- Assume all elements are distinct
- Depends on the partitioning of the array
  - if each partition is roughly 1/2 of the items
    - quicksort is quick!
  - if one of the partitions has “most of the items” and the other portion has very few items
    - quicksort is not quick - it is slow…

Worst case analysis of quick sort

Unbalanced partition

For example: **0** items on **low side** and **n-1** items on **high side**

- $T(n) = \mathbf{T(0)} + \mathbf{T(n-1)} + \Theta(n)$

  $= \mathbf{\Theta(1)} + \mathbf{T(n-1)} + \Theta(n)$

  $= \mathbf{T(n-1)} + \Theta(n)$

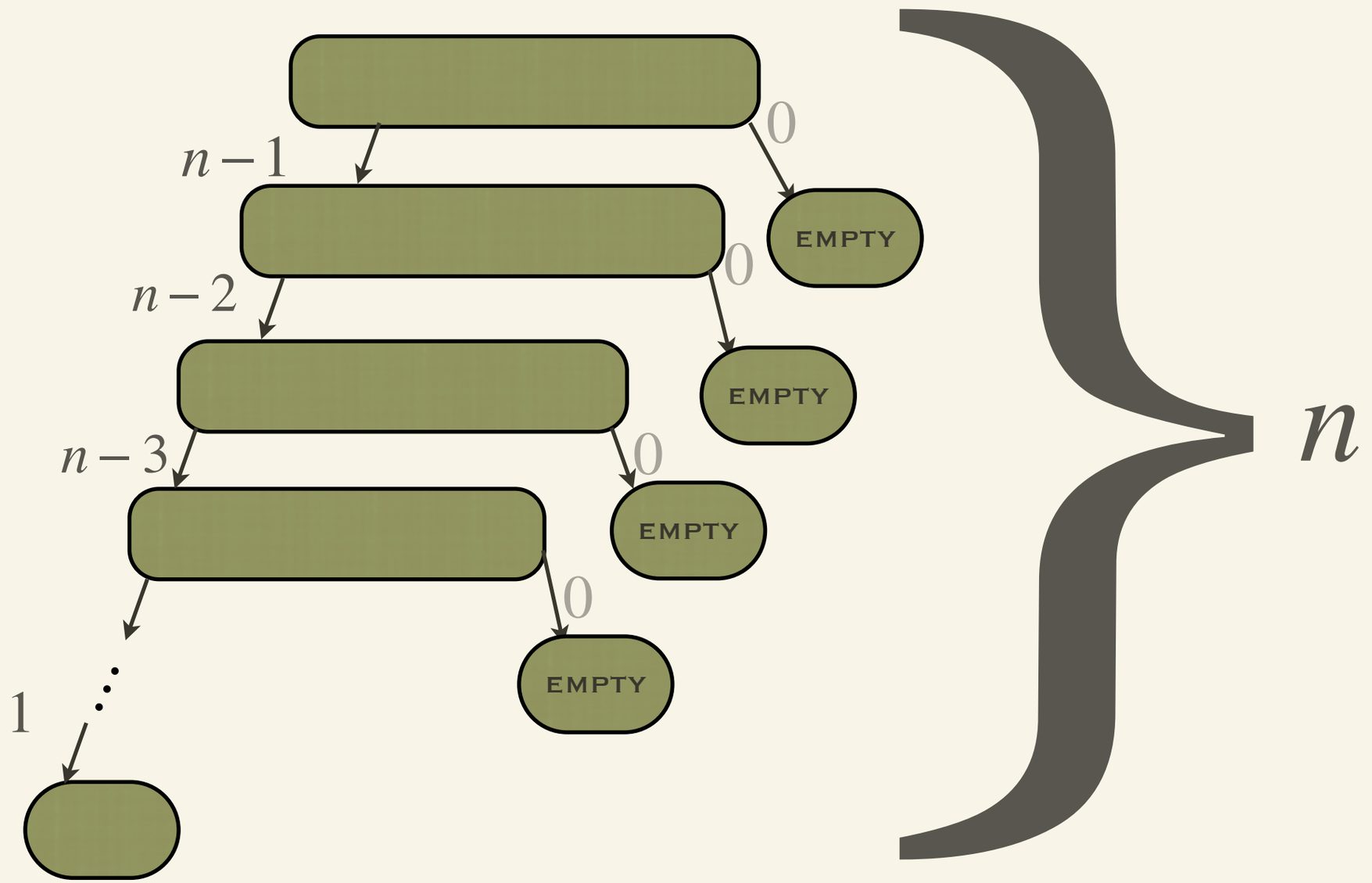
  $= \Theta(n^2)$ // arithmetic series

- Worst case occurs if items are already sorted!

Worst Case: n-1/0 Split...

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

$$= \Theta(n^2)$$



$$T(1) = 1$$

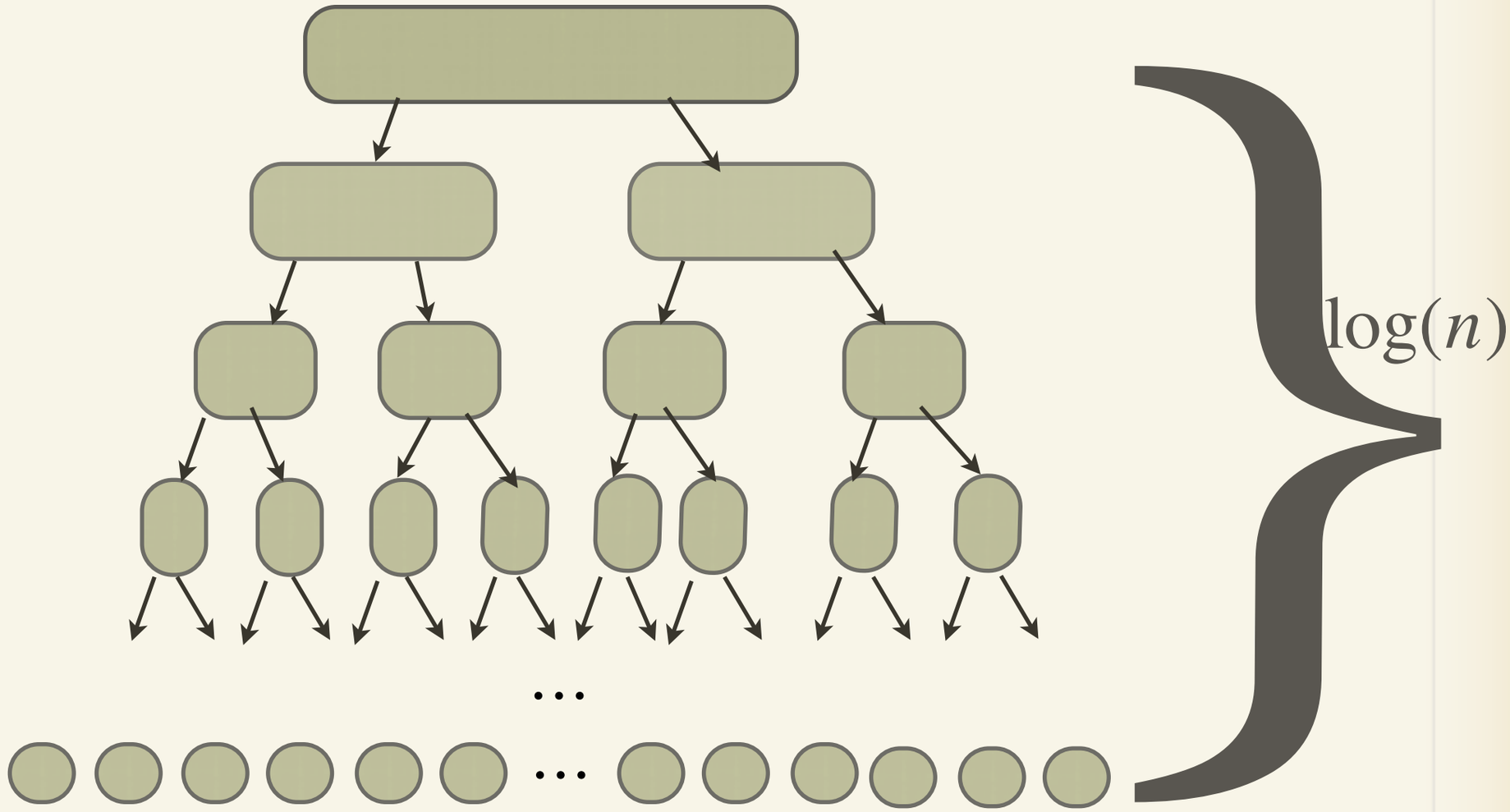
$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$O(n^2)$ time

Best case

- Always pick the median element as the pivot!
- $T(0) = T(1) = \theta(1)$
- $T(n) = 2T(\lfloor n/2 \rfloor]) + \theta(n)$
- $T(n) = \theta(n \log n)$

Best Case: 50/50 Split!



$$T(1) = 1$$

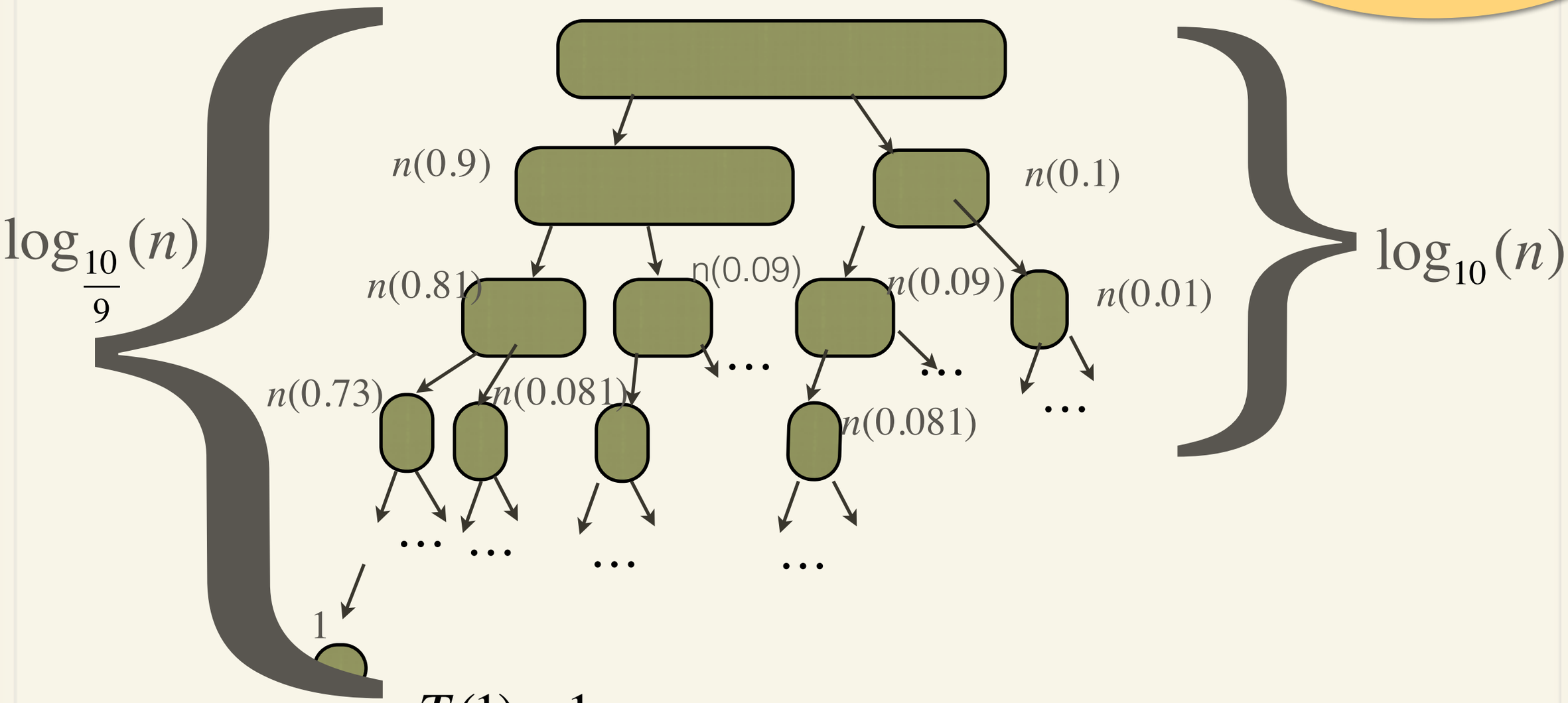
$$T(n) = 2T(n/2) + \Theta(n)$$

Sloppy not using
$T(n)=T(\lceil n/2\rceil-1)+T(\lfloor n/2\rfloor)+\Theta(n)$

$O(n \bullet \log(n))$ time

Suppose: 90/10 Split...

80 % of the time a partition will be at least as good as a 90/10 split



$$T(1) = 1$$

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

$O(n \bullet \log(n))$ time

How to (probably) avoid the bad case in QuickSort

Randomization!

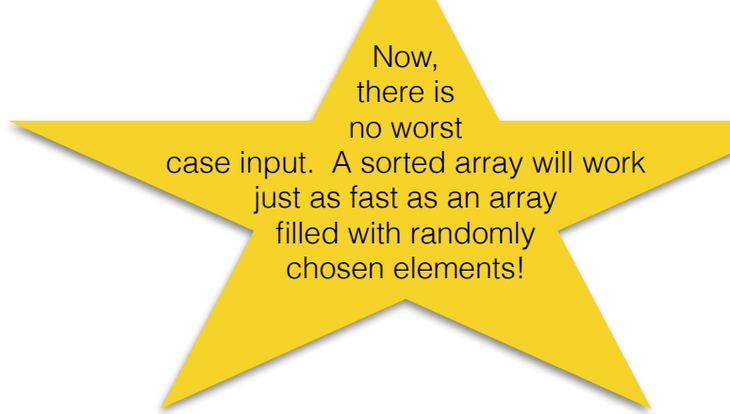
Assume any input - not just the “average” input

Chose a random **pivot**

```
RANDOMIZED-PARTITION(A,p,r)
  i = RANDOM(p,r)
  exchange A[r] with A[i]
  return PARTITION(A, p, r)
```

Now, there is no worst case input. A sorted array will work just as fast as an array filled with randomly chosen elements!

But....



Worst Case Analysis

The book does a formal analysis.

The intuition is that in the worst case the pivot could be the largest (or smallest) item

$$T[1] = 1$$

$$T[n] = T[n-1] + T[0] + \Theta(n)$$

When analyzing an algorithm, sometimes the bad cases are rare, and we analyze the average case

Often easier to do for randomized algorithms

How can we estimate the expected running time of the randomized Quicksort Running algorithm?

What is the expected running time?

How do we analyze a random algorithm?

We won't try to analyze quick sort by writing a recurrence relation. We will use an accounting trick to keep track of how much work is done

Pair share
How many recursive calls are made when running quicksort?

The running time is bounded by:
1) Time to make the recursive calls
2) Time taken to partition the elements

The running time of quick sort is O(n + X) where X is the number of times two items are compared with each other.

What is the average number of times two items are compared?

For one recursive call, the time it takes to partition the items is

$c \cdot$ # comparisons

Observation 1: we only compare z_i and z_j if one of the keys is the pivot

Observation 2: if z_i and z_j are compared, they are never compared again

But, first some math…

$$\ln(n+1) \leq \sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1$$

Log base e

Expected Number of Comparisons

A is not sorted

Ordered based on their **rank**

We add randomness to the algorithm
Randomness is not assumed for the data

Let the elements of A be $z_1, z_2, \ldots, z_n$ in sorted order

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$$

All the elements from z_i to z_j i.e. the **i^{th} smallest through j^{th} smallest** elements

Observe that any two elements will be compared at most one time

$$X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ is compared to } z_j \\ 0 & \text{otherwise} \end{cases}$$

The total number of comparisons $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$

Example: $Z_{3\,7}$={15, 19, 20, 23, 43}

z_1=4 z_2=5 z_3=15 z_4=19 z_5=20 z_6=23 z_7=43 z_8=72

43	23	5	19	4	72	15	20

If 23 is the randomly chosen pivot then 5 will never be compared with 72

Expected Number Comparisons

We add randomness to the algorithm
Randomness is not assumed for the data

Let the elements of A be $z_1, z_2, \ldots, z_n$ in sorted order

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$$

All the elements from z_i to z_j i.e. the **i^{th} smallest through j^{th} smallest** elements

Observe that any two elements will be compared at most one time

$$X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ is compared to } z_j \\ 0 & \text{otherwise} \end{cases}$$

The total number of comparisons $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

Linearity of expectation!

Average Case Analysis Cont

Let the elements of A be $z_1, z_2, \ldots, z_n$ in sorted order $\quad Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

$$\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ or } z_j \text{ is the first chosen pivot from } Z_{ij}\}$$

$$= \Pr\{z_i \text{ is the first chosen pivot from } Z_{ij}\}$$

$$+ \Pr\{z_j \text{ is the first chosen pivot from } Z_{ij}\}$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

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

$z_1=4$ $z_2=5$ $z_3=15$ $z_4=19$ $z_5=20$ $z_6=23$ $z_7=43$ $z_8=72$

43	23	5	19	4	72	15	20

$Z_{2\,7}=\{5, 15, 19, 20, 23, 43\}$

Average Case Analysis Cont

randomness occurs in the algorithm. Randomness is not assumed for the data

Let the elements of A be $z_1, z_2, \ldots, z_n$ in sorted order

$$Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$$

Observe that any two elements will be compared at most one time

$$X_{ij} = \mathrm{I}\{z_i \text{ is compared to } z_j\}$$

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

harmonic series

Renaming $k = j - i$

$$= \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1}\sum_{k=1}^{n} \frac{2}{k} = \sum_{i=1}^{n-1}\left[2\sum_{k=1}^{n}\frac{1}{k}\right] = \sum_{i=1}^{n-1} O(\log(n)) = O(n\log(n))$$

We can solve this problem in n*log(n) time! Just sort the items first.

i^{th} Order Statistic

Input: A (n ***distinct*** numbers), i ($1 \leq i \leq n$)

Output: x such that t i-1 numbers in A are smaller than x

minimum ($i = 1$)

maximum ($i = n$)

median ($i = \lfloor (n+1)/2 \rfloor$)

a graphics program

Pair share

Can we find the min/max faster?

```
MINIMUM(A, n)
min =A[1]
for i = 2 to n
   if min > A[i]
      min = A[i]
return min
```

```
MAXIMUM(A, n)
max =A[1]
for i = 2 to n
   if max < A[i]
      max = A[i]
return max
```

Can we do it faster?

$$\#\,\text{comparisons} = 1 + 3 \cdot \frac{n-2}{2}$$

$$= 1 + 3\frac{n}{2} - 3$$

$$= 3\frac{n}{2} - 2$$

$$\leq 3\frac{n}{2}$$

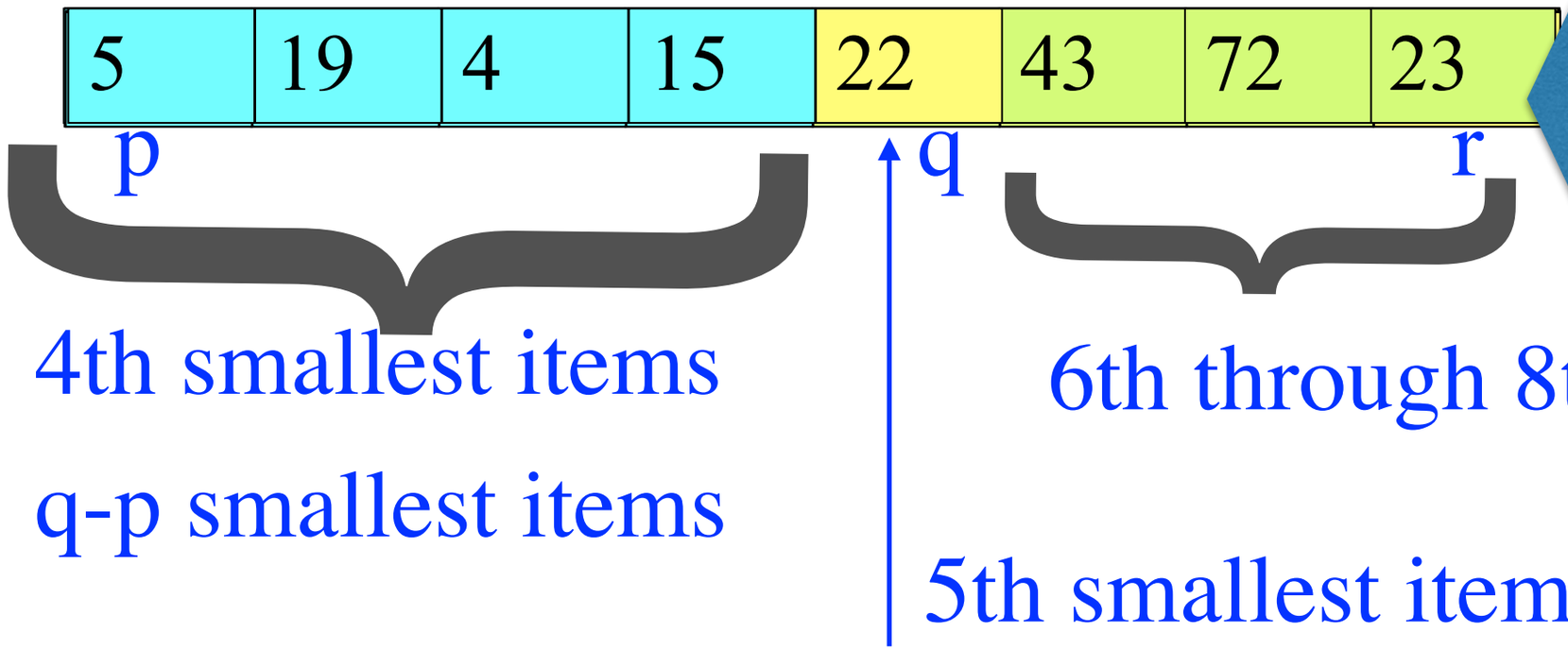
Compare
A[2i] with A[2i+1]
The larger is compared
with max
the smaller with min

For
analysis - assume n is
even.
The case when
n is odd is similar

Pair share

How fast can find the i^{th} smallest item in an array of n random distinct items?

Modify the Quicksort Algo?

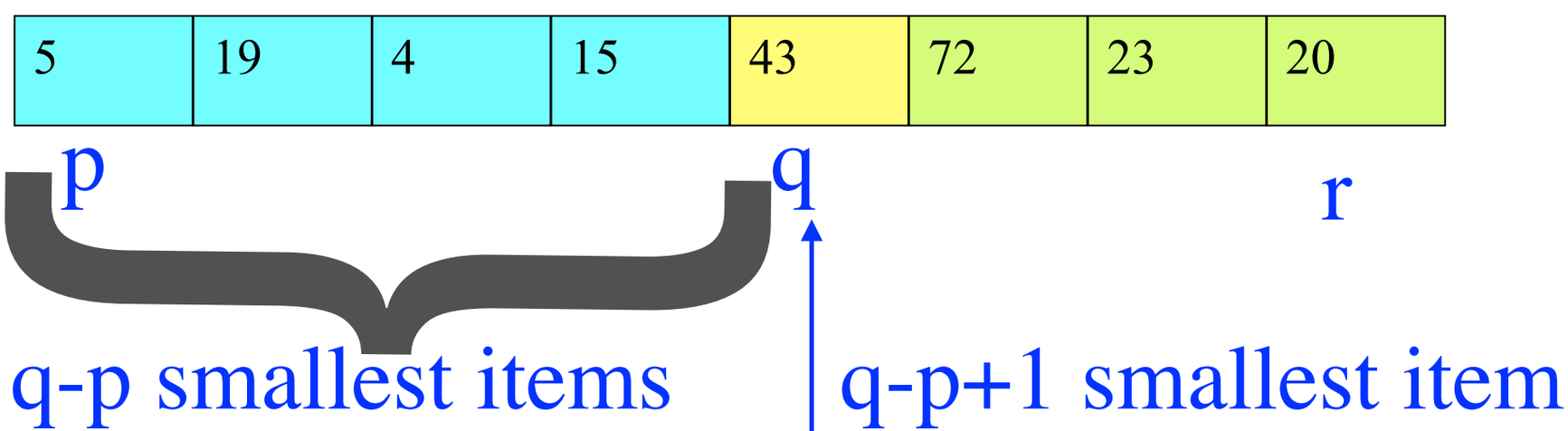


For example, the **3rd** smallest item in this set of **8 items** is item **15.** There are **q-p=4** items smaller than the **pivot** which is the **5th** smallest of these **8** items

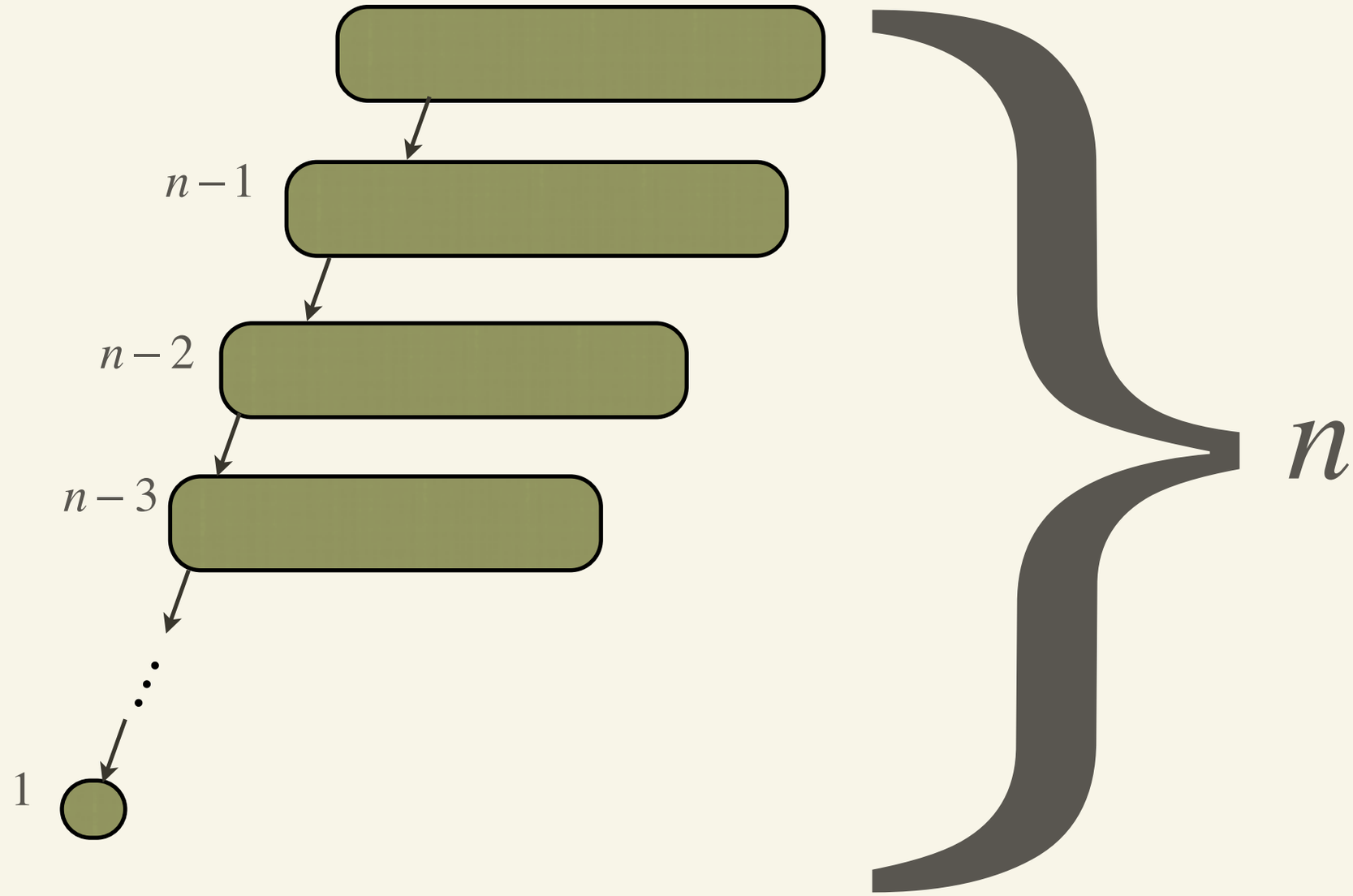
i^{th} Order Statistic

RANDOMIZED-SELECT(A, p, r, i)

```
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    k = q - p + 1
    if k == i
        return A[q]
    elseif i < k
        return RANDOMIZED-SELECT(A, p, q-1, i)
    else
        return RANDOMIZED-SELECT(A, q+1, r, i-k)
```



Worst Case: n-1/0 Split...

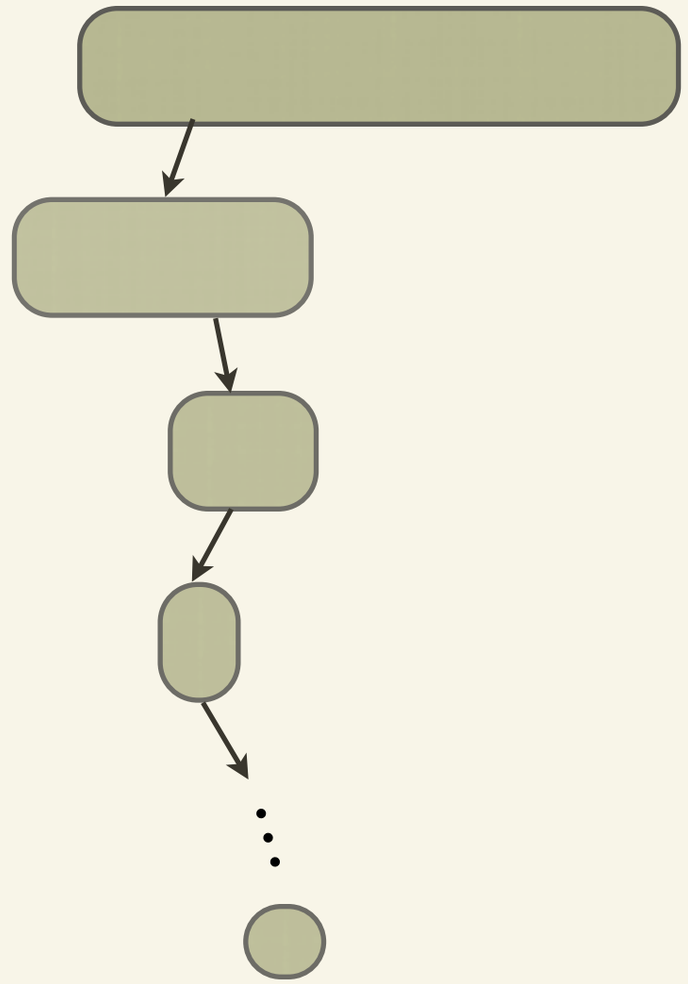


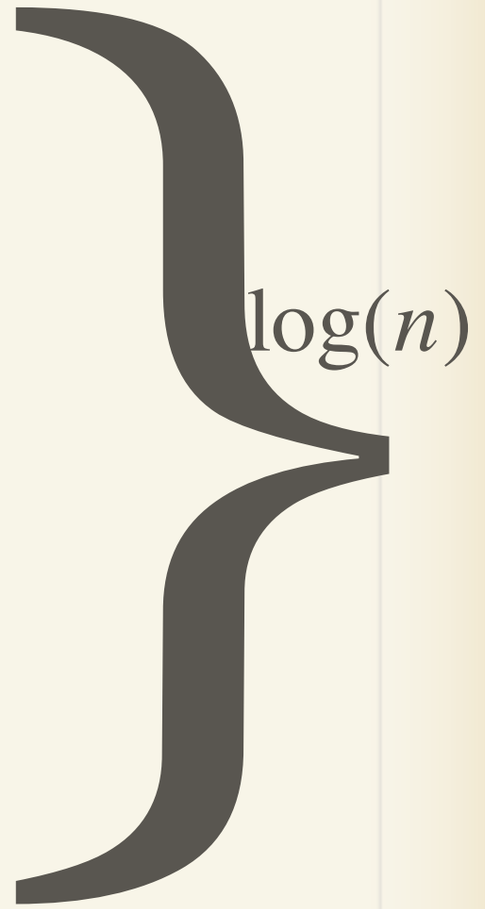
$$T[1] = 1$$

$$T[n] = T[n-1] + \Theta(n)$$

$O(n^2)$ time

50/50 split Case:

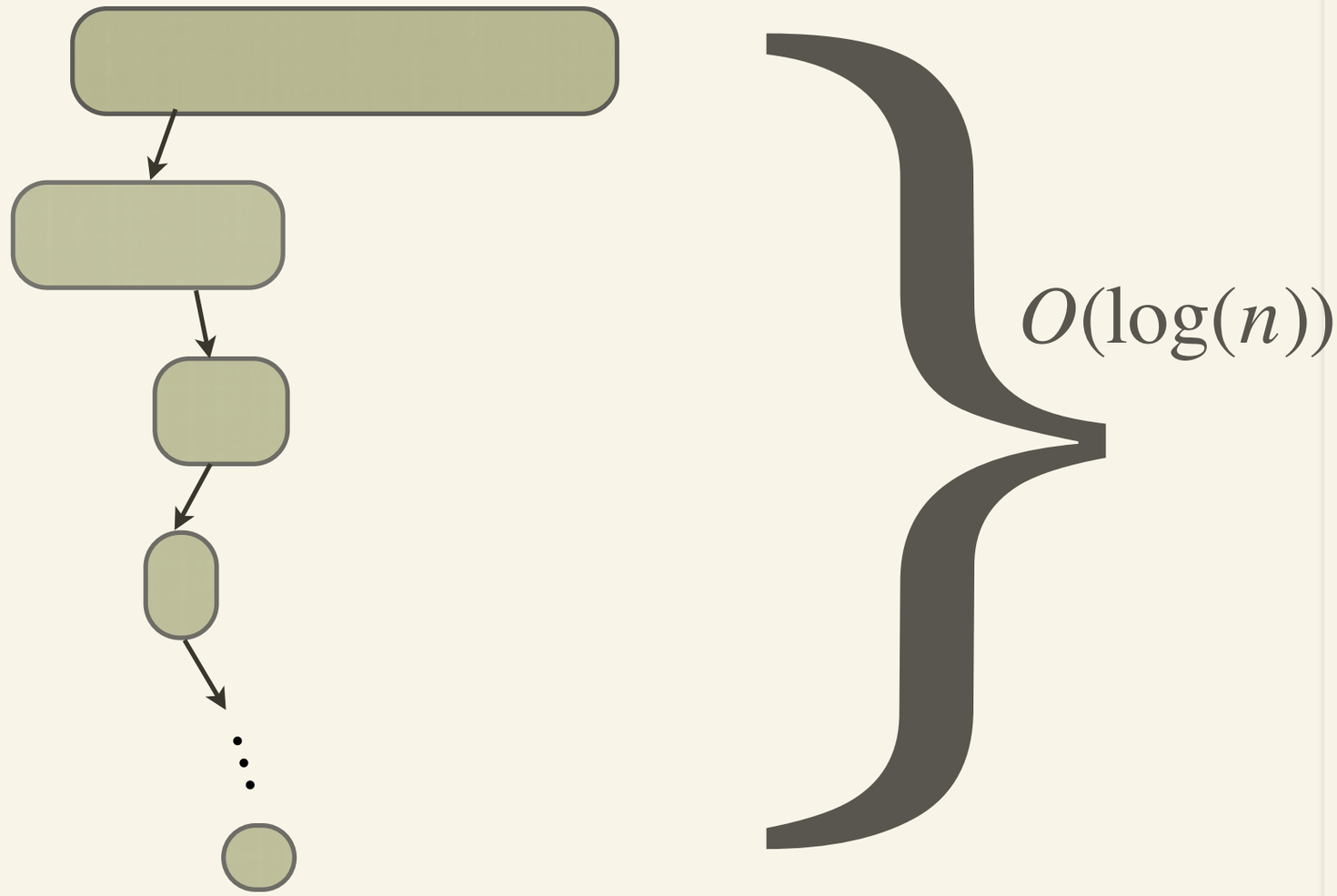




$$T[1] = 1$$

$$T[n] = T[n/2] + \Theta(n)$$

OK Case



$$T[1] = 1$$

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

Order selection

- linear expected time
- works in practice
- bad worst case
- How could we get linear *worst* case time?

The algorithm we design is more for theoretical interest. The constants are large.

Pair share

Can we do this better?

If we could ensure our split was "in the middle", or "close" to being in the middle…

Deterministic Linear Selection

Basic Algorithm

1. Recursively find a “good pivot”
2. Partition items using “good pivot”
3. Recursive call partition containing i’th smallest (if the pivot isn’t the i’th smallest)

1. Recursively find a “good pivot”

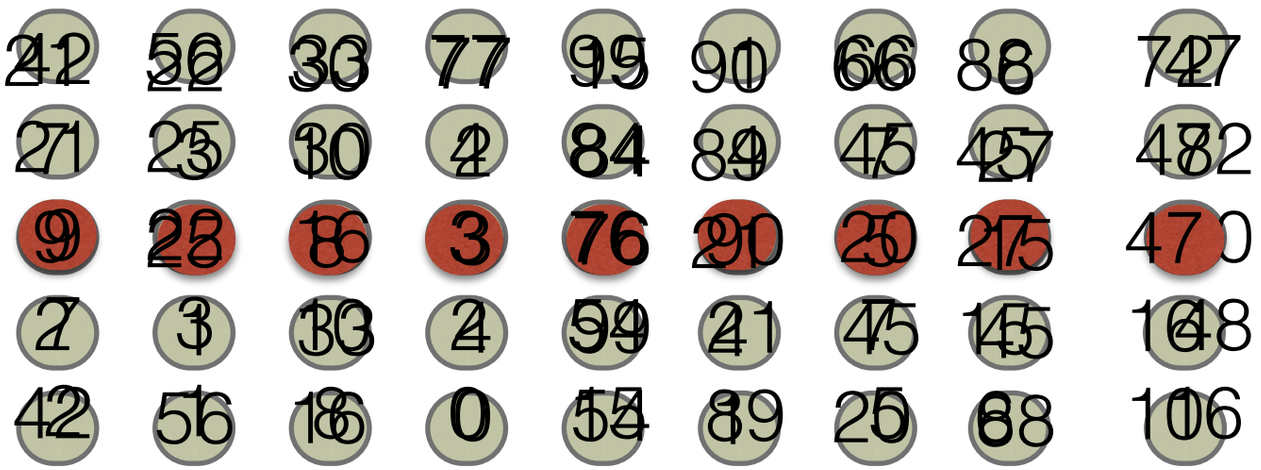
- Partition the items into groups of 5

21	17	30	77	15	1	66	6	47
7	3	10	2	84	4	7	27	72
9	25	8	3	76	90	5	15	10
2	1	33	4	99	21	45	45	48
42	56	16	0	54	89	20	88	16

We won’t code it up this way - this is just an intuitive look at what we are doing

21, 7, 9, 2, 42, 17, 3, 25, 1, 56, …,
6, 27, 15, 45, 88, 47, 72,10, 48,16

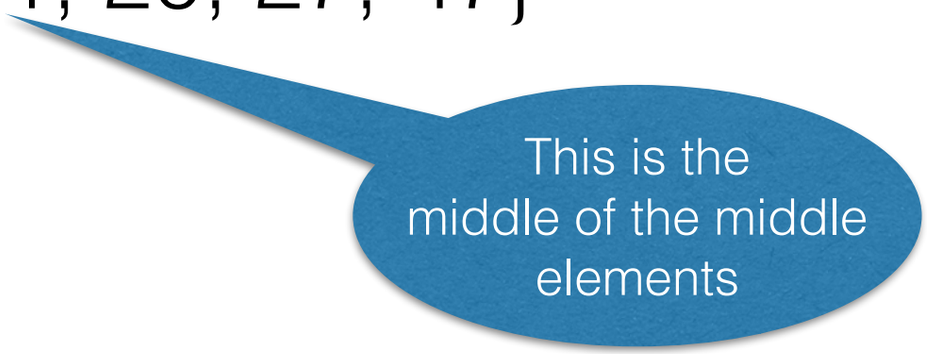
1. Recursively find a “good pivot”
   - Partition the items into groups of 5
   - Sort each group of 5



B={9, 22, 16, 3, 76, 21, 20, 27, 47}

1. Recursively find a “good pivot”
   - Partition the items into groups of 5
   - Sort each group of 5
   - Recursively call DETERMINISTICSELECT to find the middle of the middle elements

B={9, 22, 16, 3, 76, 21, 20, 27, 47}



2. Partition items using “good pivot”

The recursion returns one item

9, 22, 16, 3, 76, 21, 20, 27, 47

This is the middle of the middle elements

7, 9, 2, . . . , 3, 1, 10, 8, 6, 15, 21, 25, 21, 42, 22, 56, 30, 33, 16, . . . 27,

k^{th} smallest

partition based on x, (A[k] = x)

Among all the original items x is the k^{th} smallest

3. Recursive call partition containing i'th smallest (if the pivot isn't the i'th smallest)

k^{th} smallest

if $i = k$, return x

if $i < k$, return i^{th} smallest in A[1..(k-1)]

if $i > k$, return $(i-k)^{th}$ smallest in A[(k+1)..n]

DETERMINISTICSELECT(A, n, i)
// find the i^{th} smallest of the items in A

New CLRS has pseudo code and calls this algorithm SELECT

Simplify recursion/analysis, by reducing the problem to:
- a simple case and
- the case where the number of elements is a multiple of 5

Θ(n) 3. Divide the remaining elements of the input array **A** into $g = n/5$ groups of **5**.

Θ(n) 4. Find the medium of each group of **5** items and put them into another array **B**

T(n/5) 5. **x = DETERMINISTICSELECT(B,** g**,** $\lceil g/2 \rceil$**)** // finding the medium of the mediums

Θ(n) 6. Partition **A[j..n] - {x}** into two sets **A_1, A_2** such that

1. **A_1 = {k | k < x}**
2. **A_2 = {k | x < k}**

T(?) 7. if **i = $|A_1|$+1**, return **x**

1. otherwise if **i < $|A_1|$+1** return **DETERMINISTICSELECT(A_1, $|A_1|$, i)**
2. else return **DETERMINISTICSELECT(A_2, $|A_2|$, i - $|A_1|$ - 1)**

New CLRS has pseudo code and calls this algorithm SELECT

DETERMINISTICSELECT(A, n, i)
// find the i^{th} smallest of the items in A

Θ(n) 1. Let $j = n \bmod 5$. Move the smallest j elements (in sorted order to the front of A)

Θ(1) 2. If $i \leq j$, return A[i], else $i = i - j$

Θ(n) 3. Divide the remaining elements of the input array **A** into $g = n/5$ groups of **5**.

Θ(n) 4. Find the medium of each group of **5** items and put them into another array **B**

T(n/5) 5. **x = DETERMINISTICSELECT(B, g, $\lceil g/2 \rceil$)** // finding the medium of the mediums

Θ(n) 6. Partition **A[j..n] - {x}** into two sets **A_1, A_2** such that

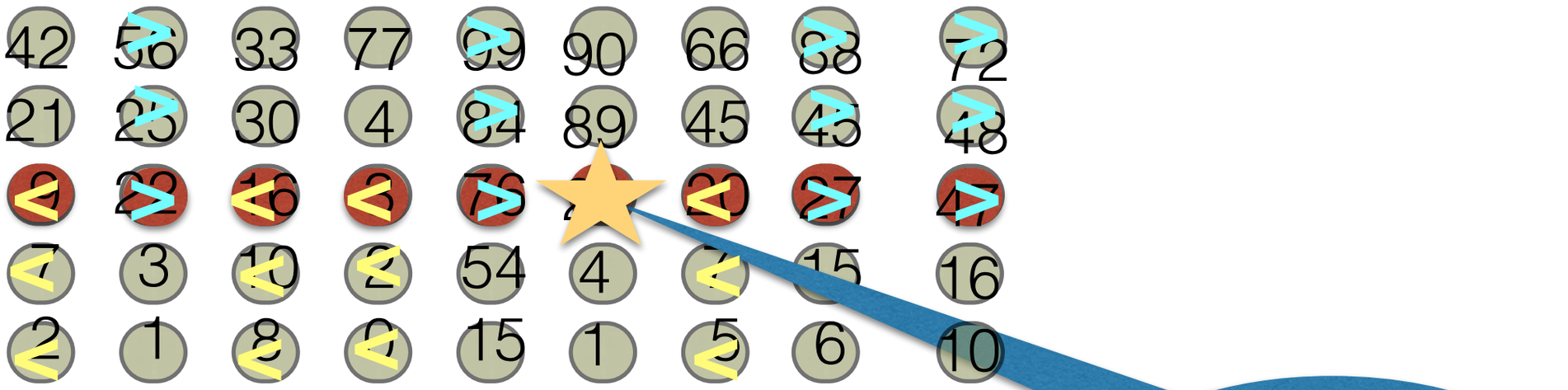
1. **A_1 = {k | k < x}**
2. **A_2 = {k | x < k}**

T(?) 7. if **i = |A_1|+1**, return **x**

1. otherwise if **i < |A_1|+1** return **DETERMINISTICSELECT(A_1, |A_1|, i)**
2. else return **DETERMINISTICSELECT(A_2, |A_2|, i - |A_1| - 1)**

Observation about the partition

The middle of the middle elements is the pivot



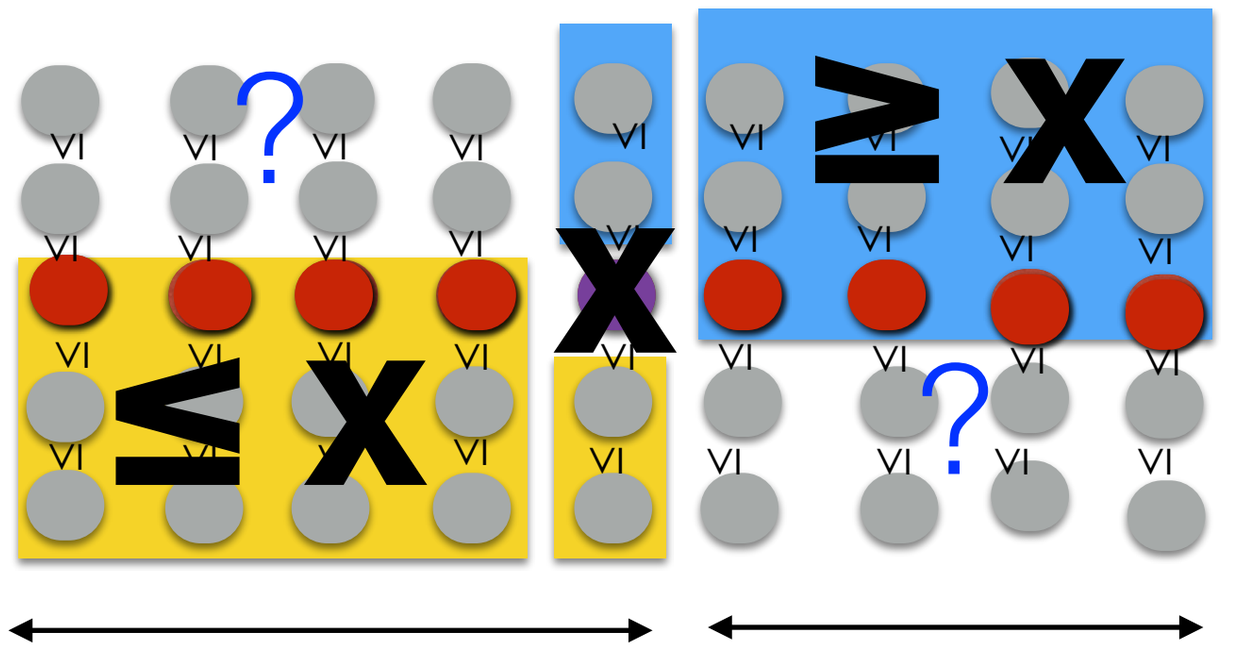
9, 22, 16, 3, 76, 21, 20, 27, 47

Analysis

There are $g = \frac{n}{5}$ groups of 5

$\lceil g/2 \rceil$ of these groups have medians less than or equal to x

$\lfloor g/2 \rfloor + 1$ of these groups have medians are greater than or equal to x

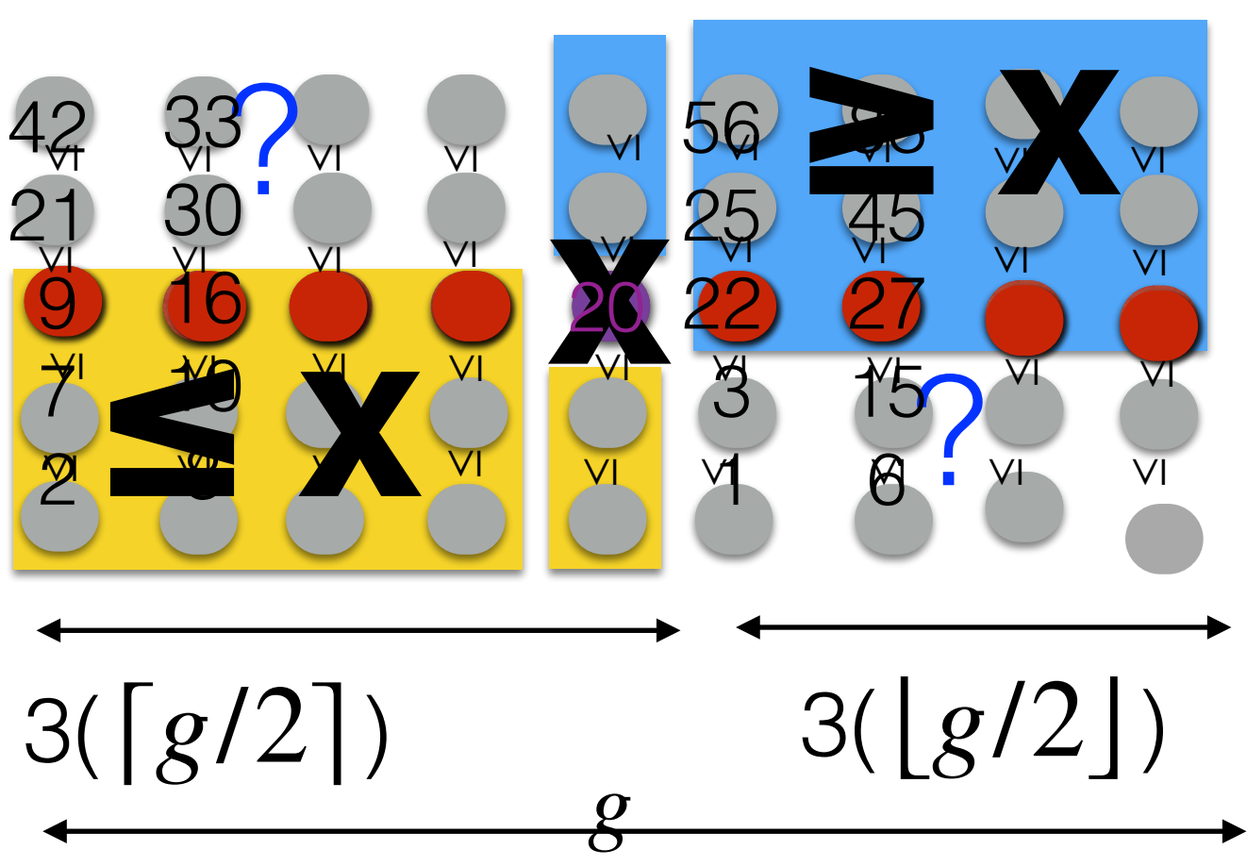


Analysis

Pair share:
How many items are in the recursive call?

Each of the groups to the **left** of x has a medium of at most x.
Thus we know at least $3\lceil g/2 \rceil \geq 3g/2$ elements at most x

Each of the groups to the **right** of x has a medium of at least x.
Thus we know at least $3\left(\lfloor g/2 \rfloor + 1\right) \geq 3g/2$ elements at least x



Analysis

Pair share:
What is the recurrence relation?

The number of items: $5g = n$

Thus there are *at most* $5g - 3g/2 = 7g/2 \leq 7n/10$ items in a partition

Finding the median of the medians was a recursive call of $g = n/5$

Homework: solve this recurrence using the substitution method (guess and check)

$$T(n) = T(7n/10) + T(n/5) + \theta(n)$$