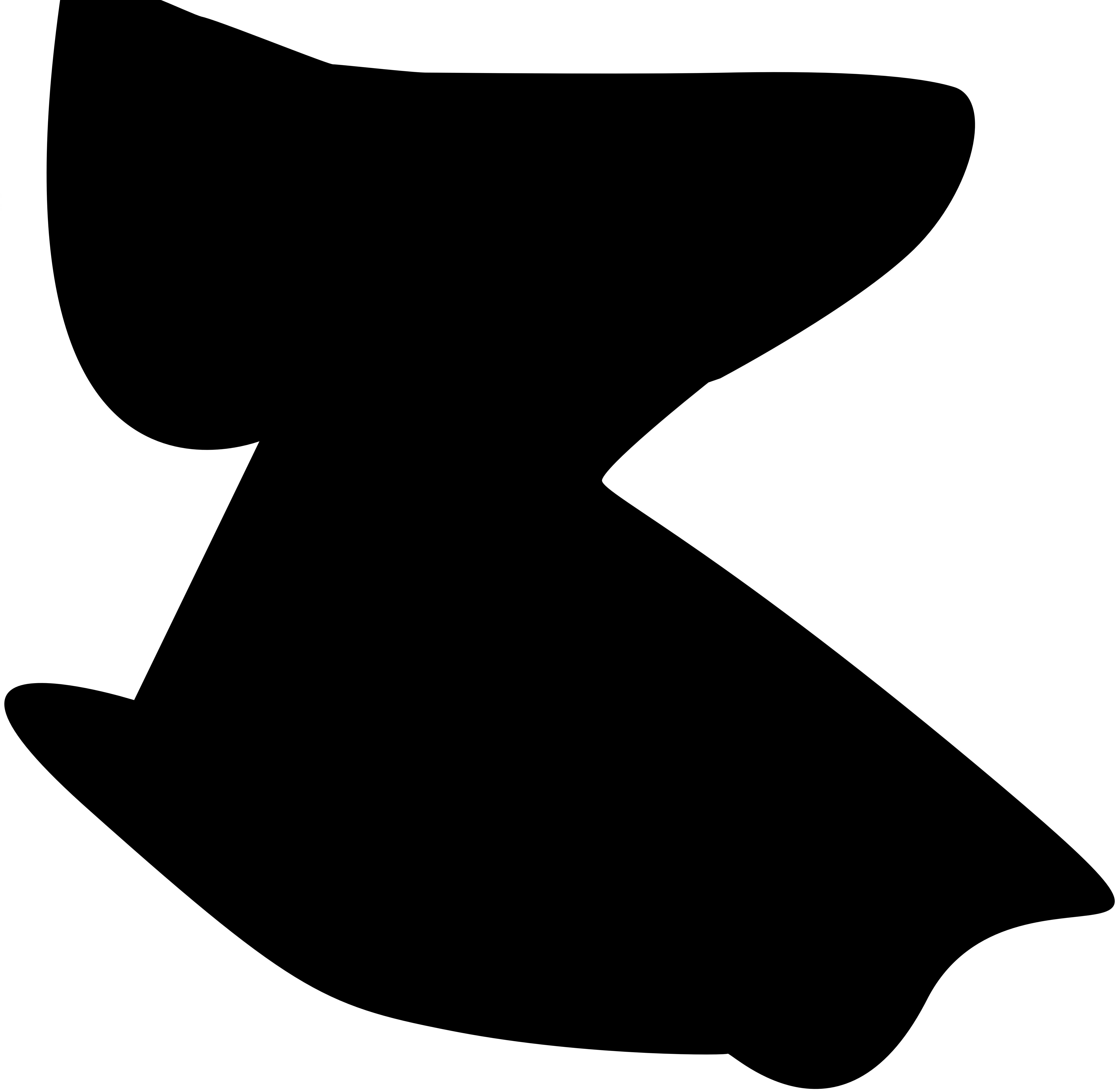
L5 5800

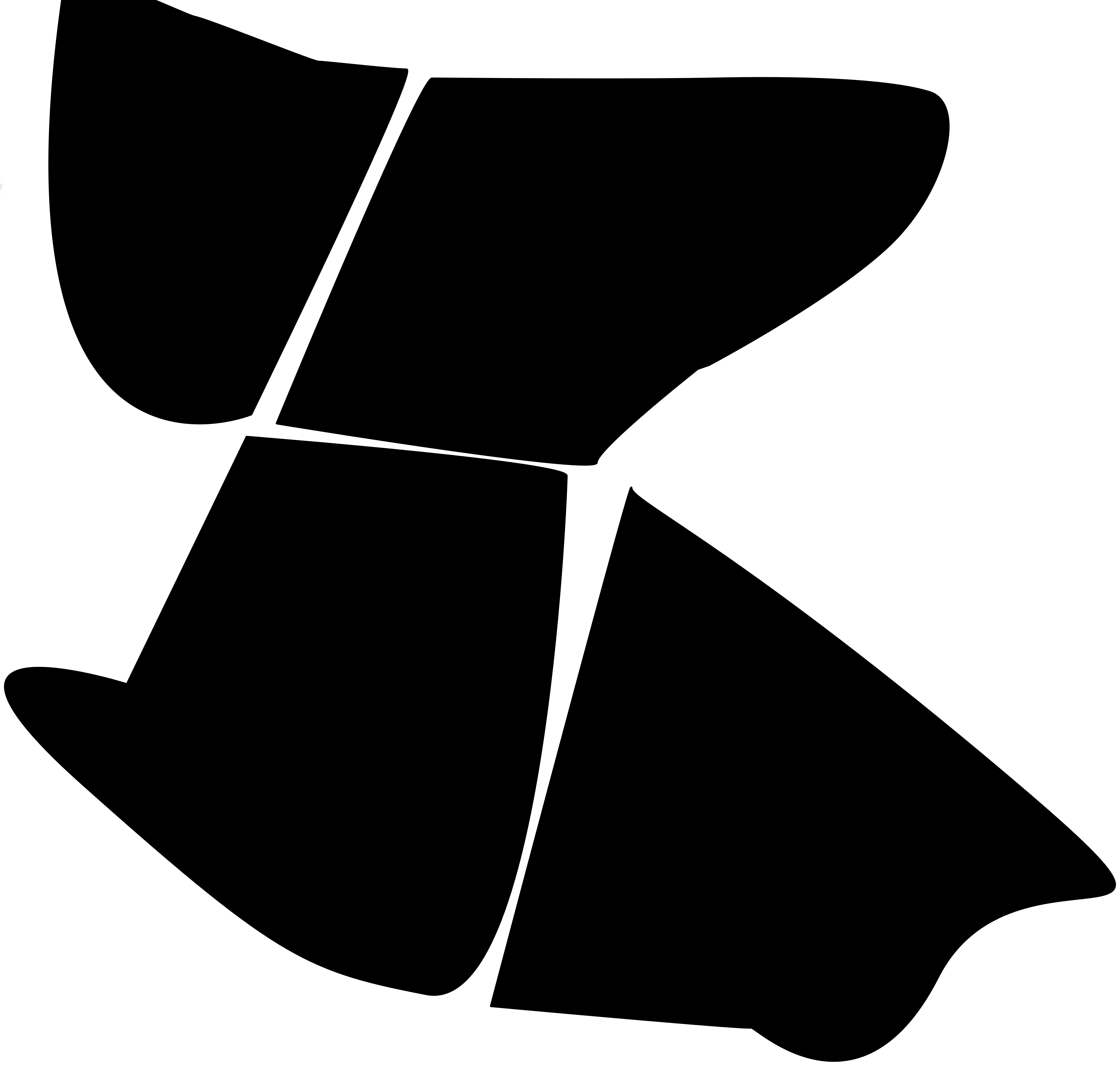
feb 1/3 2022

shelat

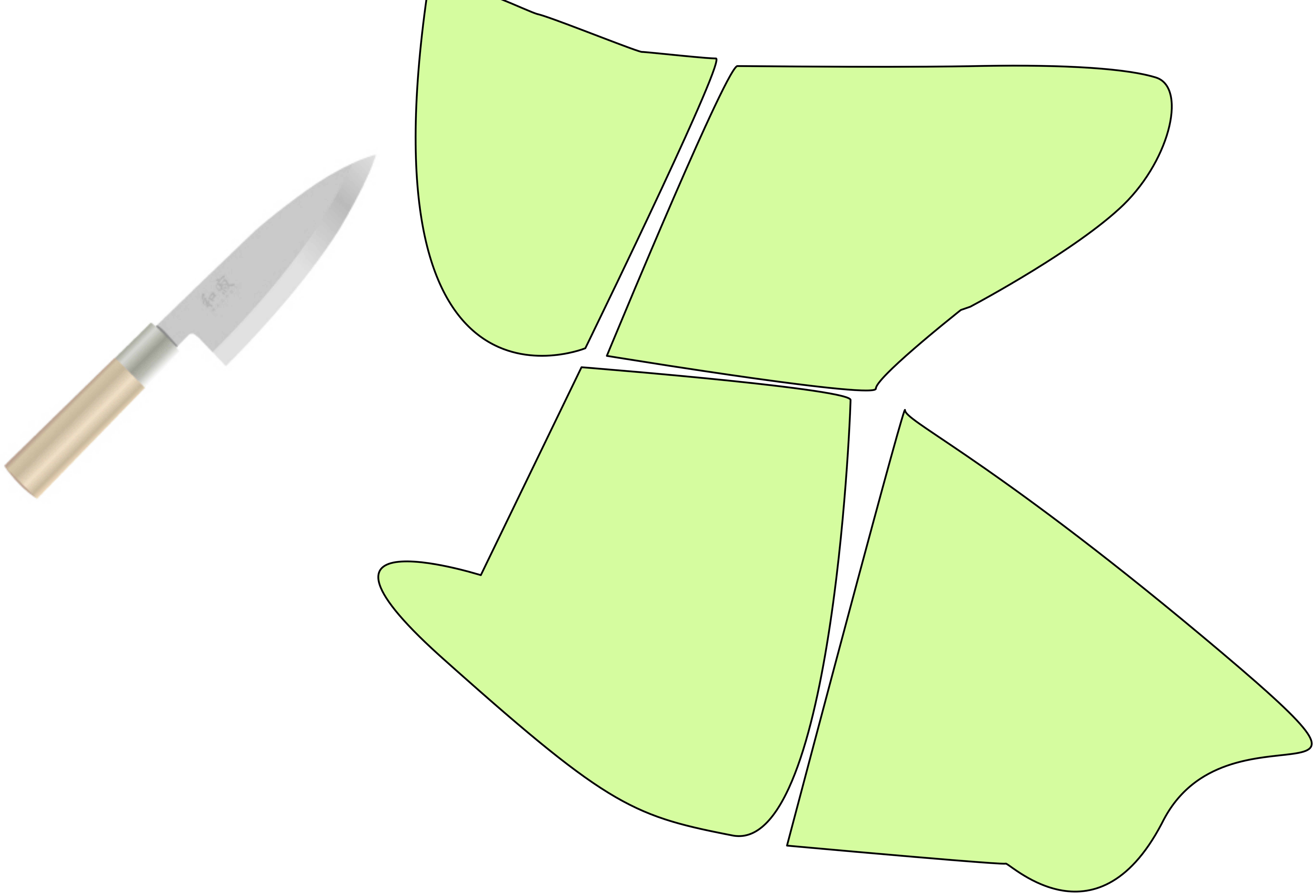
divide
& conquer



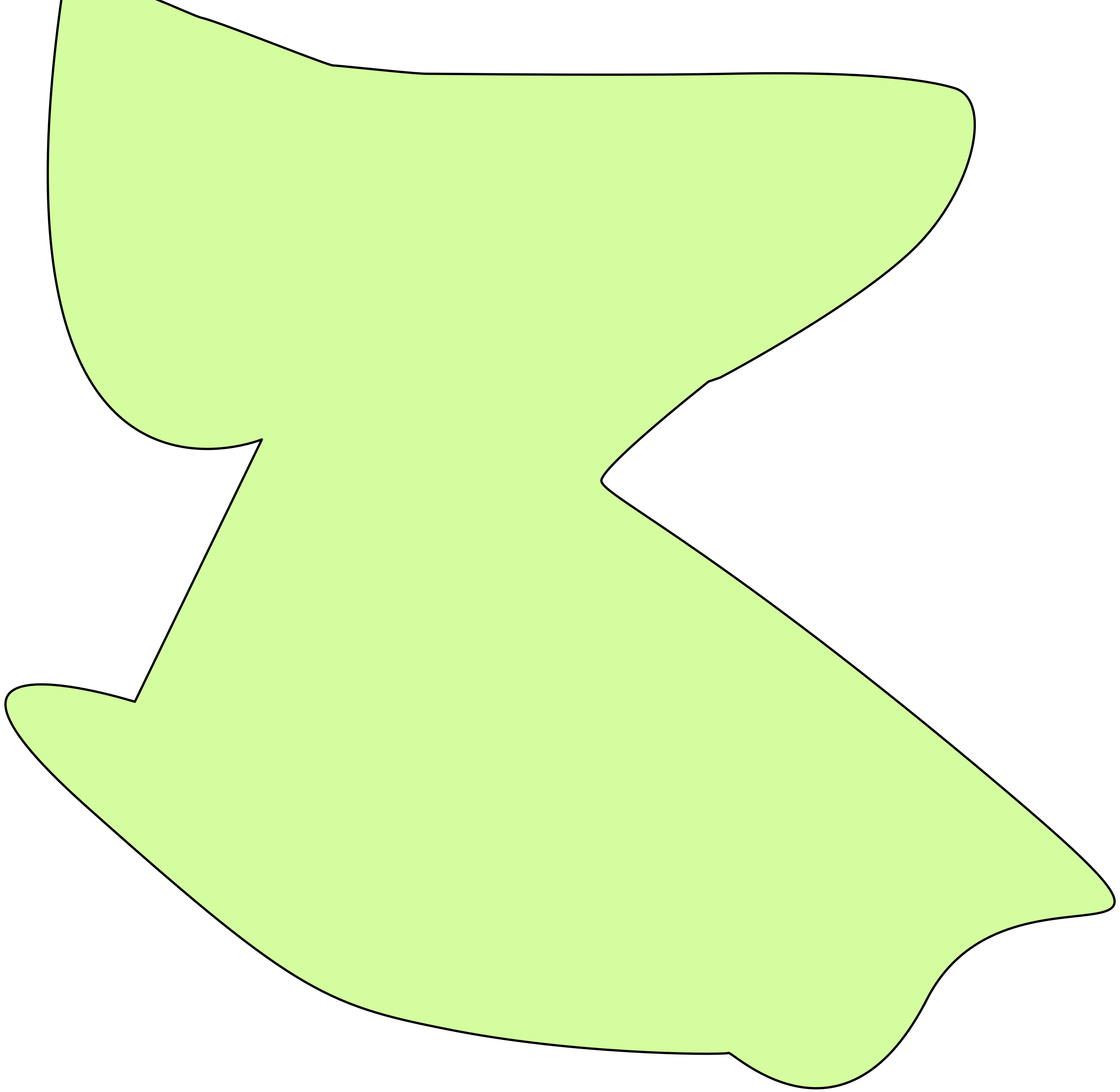
http://www.kitchenknifedrawer.com/files/1696205/uploaded/K6615D.jpg



http://www.kitchenknifedrawer.com/files/1696205/uploaded/K6615D.jpg



http://www.kitchenknifedrawer.com/files/1696205/uploaded/K6615D.jpg



http://www.kitchenknifedrawer.com/files/1696205/uploaded/K6615D.jpg

Examples we will discuss

Merge

merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p + r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q + 1, r)$
        merge(A, p, q, r)

<u>MERGE$(A[1..n], m)$:</u>
    $i \leftarrow 1;\ j \leftarrow m + 1$
    for $k \leftarrow 1$ to n
        if $j > n$
            $B[k] \leftarrow A[i];\ i \leftarrow i + 1$
        else if $i > m$
            $B[k] \leftarrow A[j];\ j \leftarrow j + 1$
        else if $A[i] < A[j]$
            $B[k] \leftarrow A[i];\ i \leftarrow i + 1$
        else
            $B[k] \leftarrow A[j];\ j \leftarrow j + 1$
    for $k \leftarrow 1$ to n
        $A[k] \leftarrow B[k]$

jeff erickson

5	2	4	7	1	3	2	6

merge-sort (A, p, r)
  if $p < r$
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    merge-sort (A, p, q)
    merge-sort $(A, q+1, r)$
    merge(A, p, q, r)

MERGE$(A[1..n], m)$:
  $i \leftarrow 1$; $j \leftarrow m+1$
  for $k \leftarrow 1$ to n
    if $j > n$
      $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
    else if $i > m$
      $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
    else if $A[i] < A[j]$
      $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
    else
      $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
  for $k \leftarrow 1$ to n
    $A[k] \leftarrow B[k]$

jeff erickson

5	2	4	7	1	3	2	6

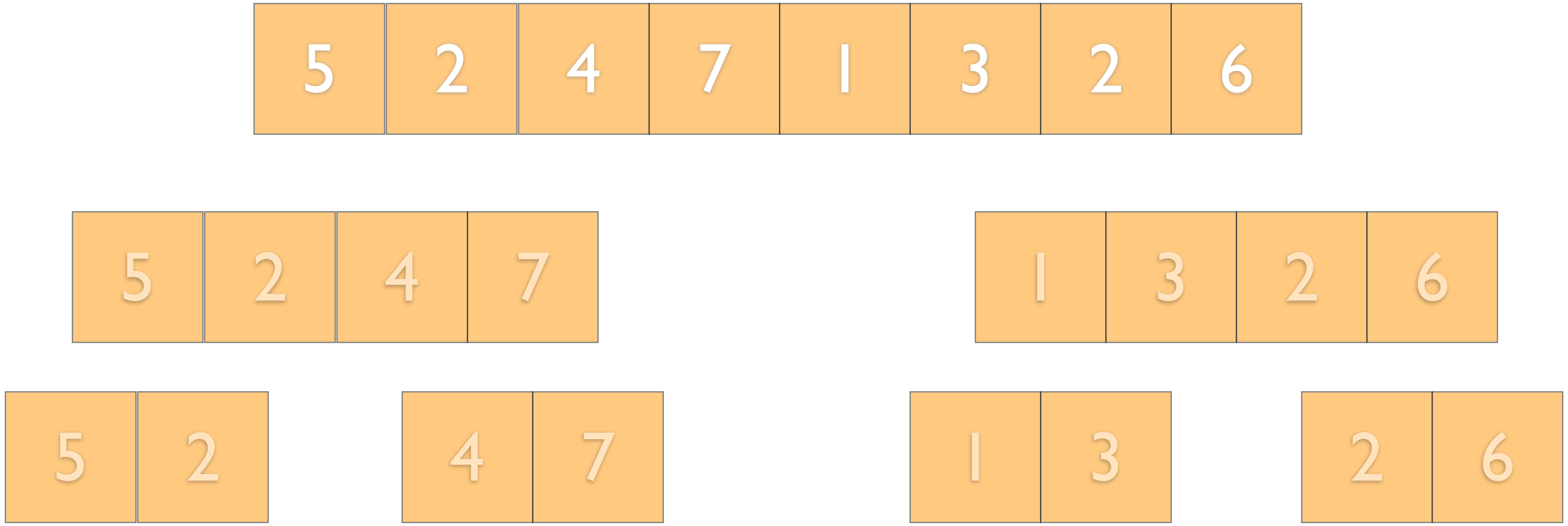
5	2	4	7

1	3	2	6

merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p + r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q+1, r)$
        merge(A, p, q, r)

MERGE$(A[1..n], m)$:
    $i \leftarrow 1$; $j \leftarrow m+1$
    for $k \leftarrow 1$ to n
        if $j > n$
            $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
        else if $i > m$
            $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
        else if $A[i] < A[j]$
            $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
        else
            $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
    for $k \leftarrow 1$ to n
        $A[k] \leftarrow B[k]$

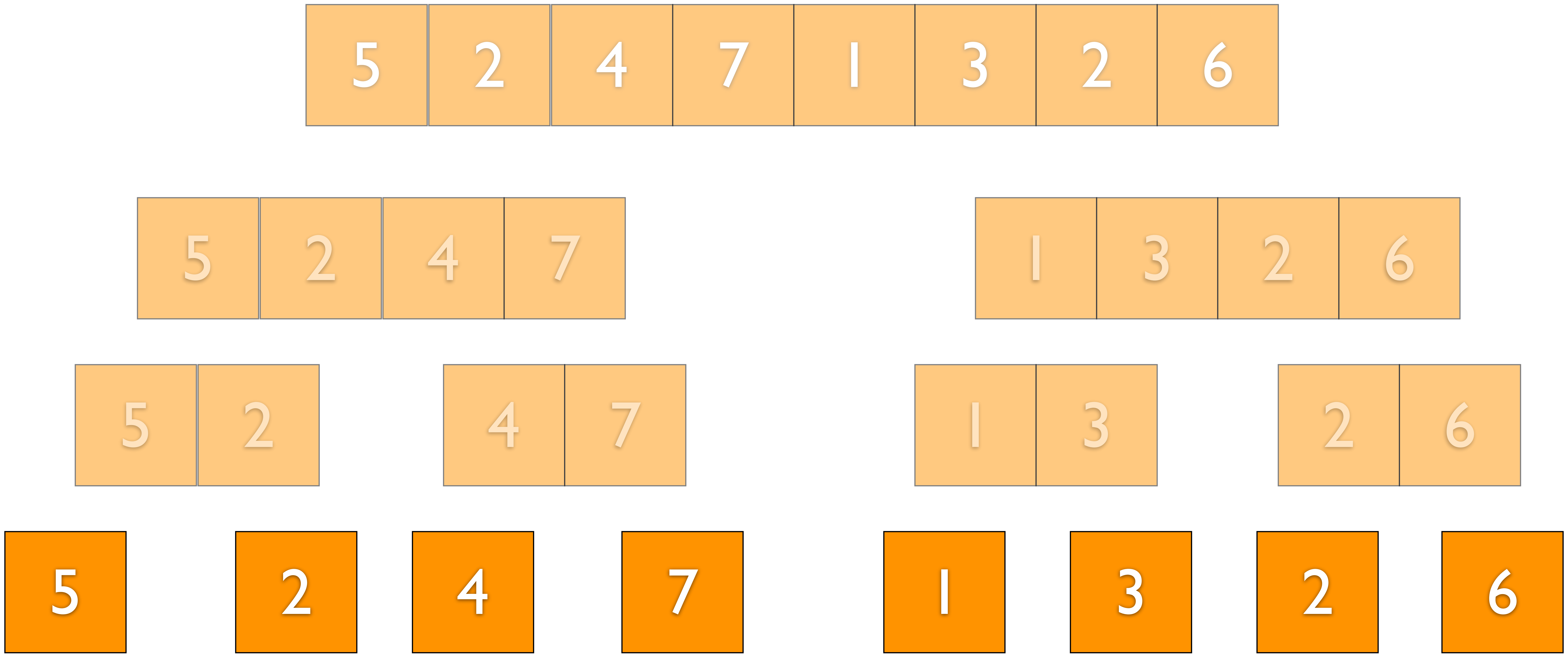
jeff erickson



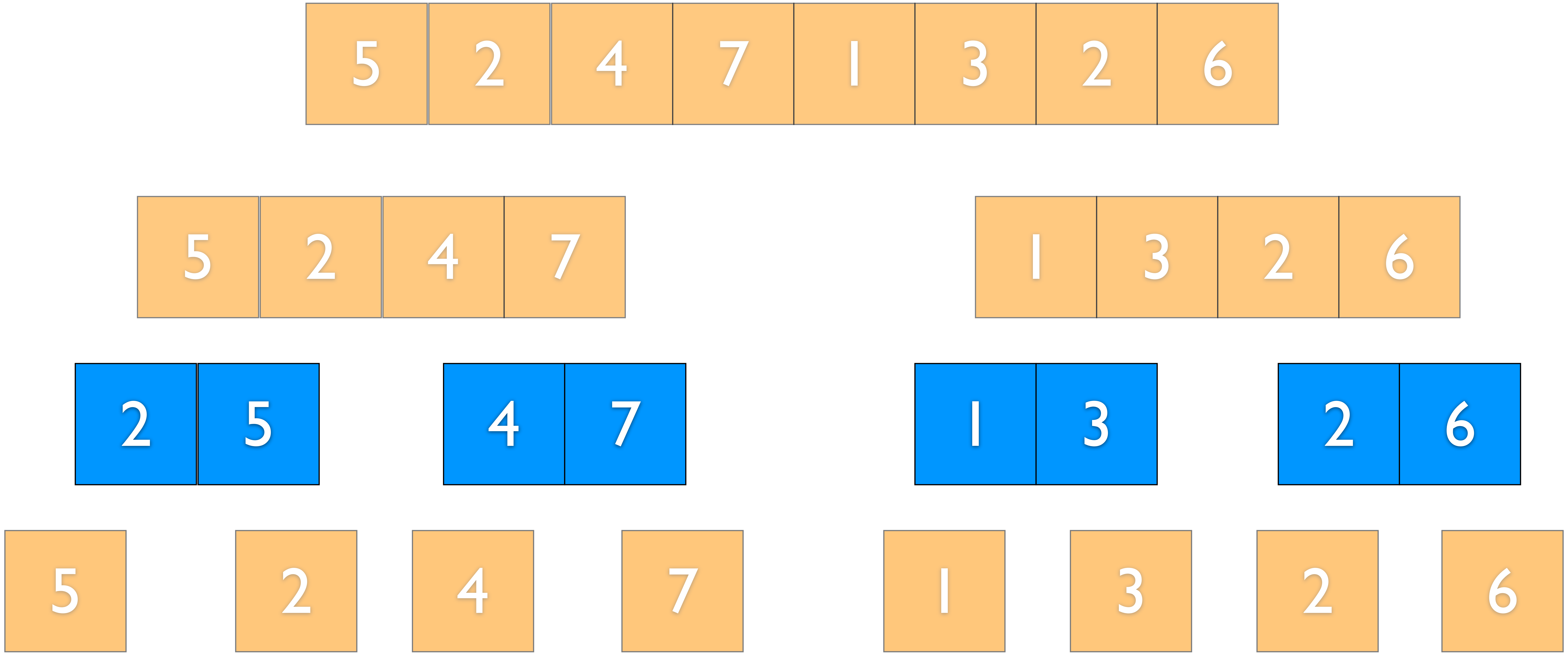
merge-sort (A, p, r)
  if $p < r$
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    merge-sort (A, p, q)
    merge-sort $(A, q+1, r)$
    merge(A, p, q, r)

```
MERGE(A[1..n], m):
  i ← 1; j ← m + 1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i + 1
      else if i > m
          B[k] ← A[j]; j ← j + 1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i + 1
      else
          B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
      A[k] ← B[k]
```

jeff erickson



merge-sort (A, p, r)
  if $p < r$
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    merge-sort (A, p, q)
    merge-sort $(A, q+1, r)$
    merge(A, p, q, r)



merge-sort (A, p, r)

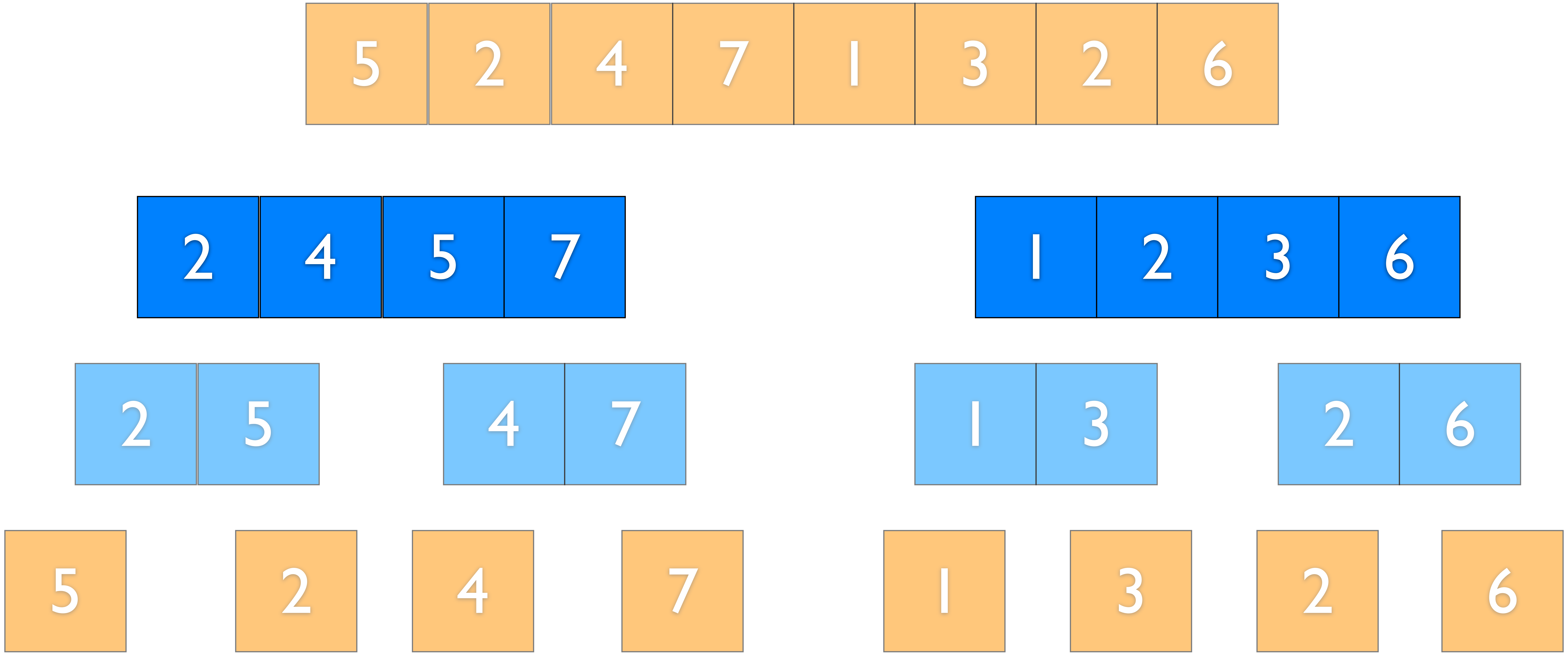
  if $p < r$

    $q \leftarrow \lfloor (p + r)/2 \rfloor$

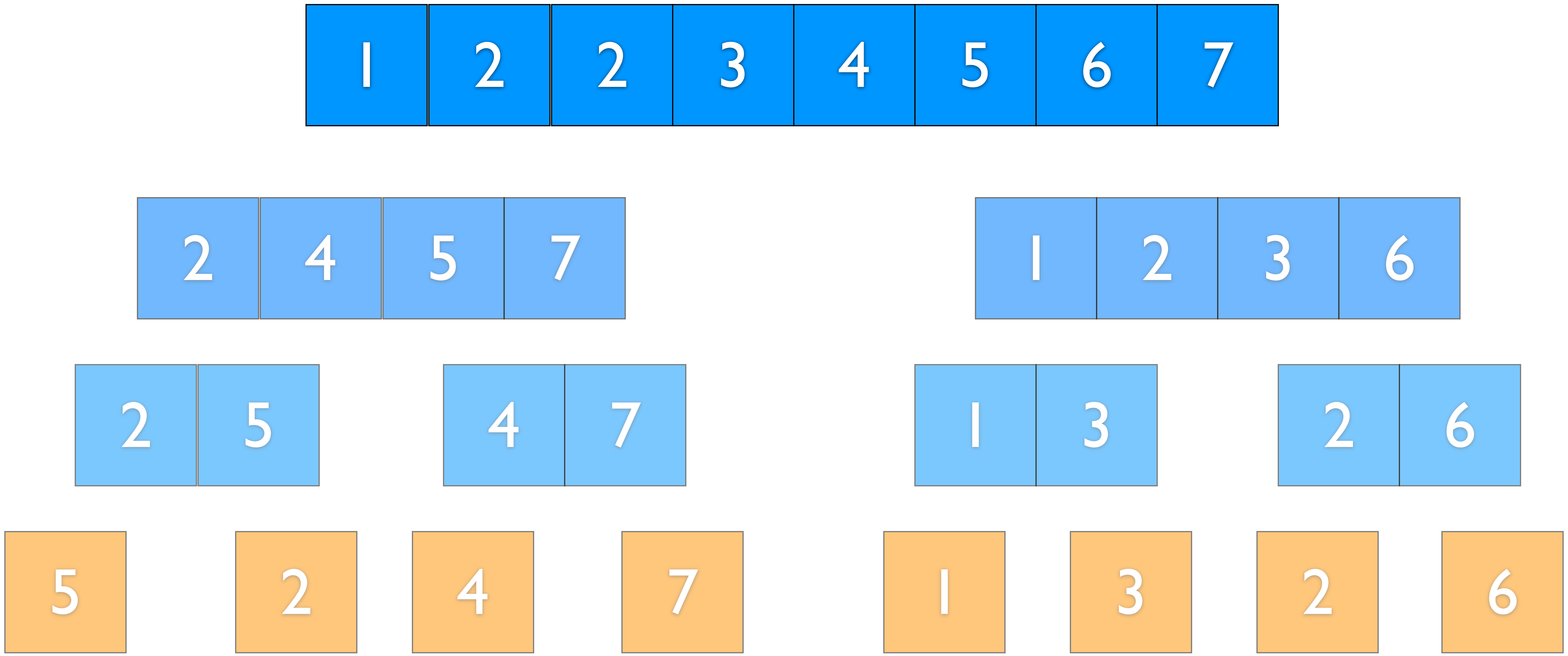
    merge-sort (A, p, q)

    merge-sort $(A, q + 1, r)$

    merge(A, p, q, r)



merge-sort (A, p, r)
if $p < r$
$q \leftarrow \lfloor (p+r)/2 \rfloor$
merge-sort (A, p, q)
merge-sort $(A, q+1, r)$
merge(A, p, q, r)

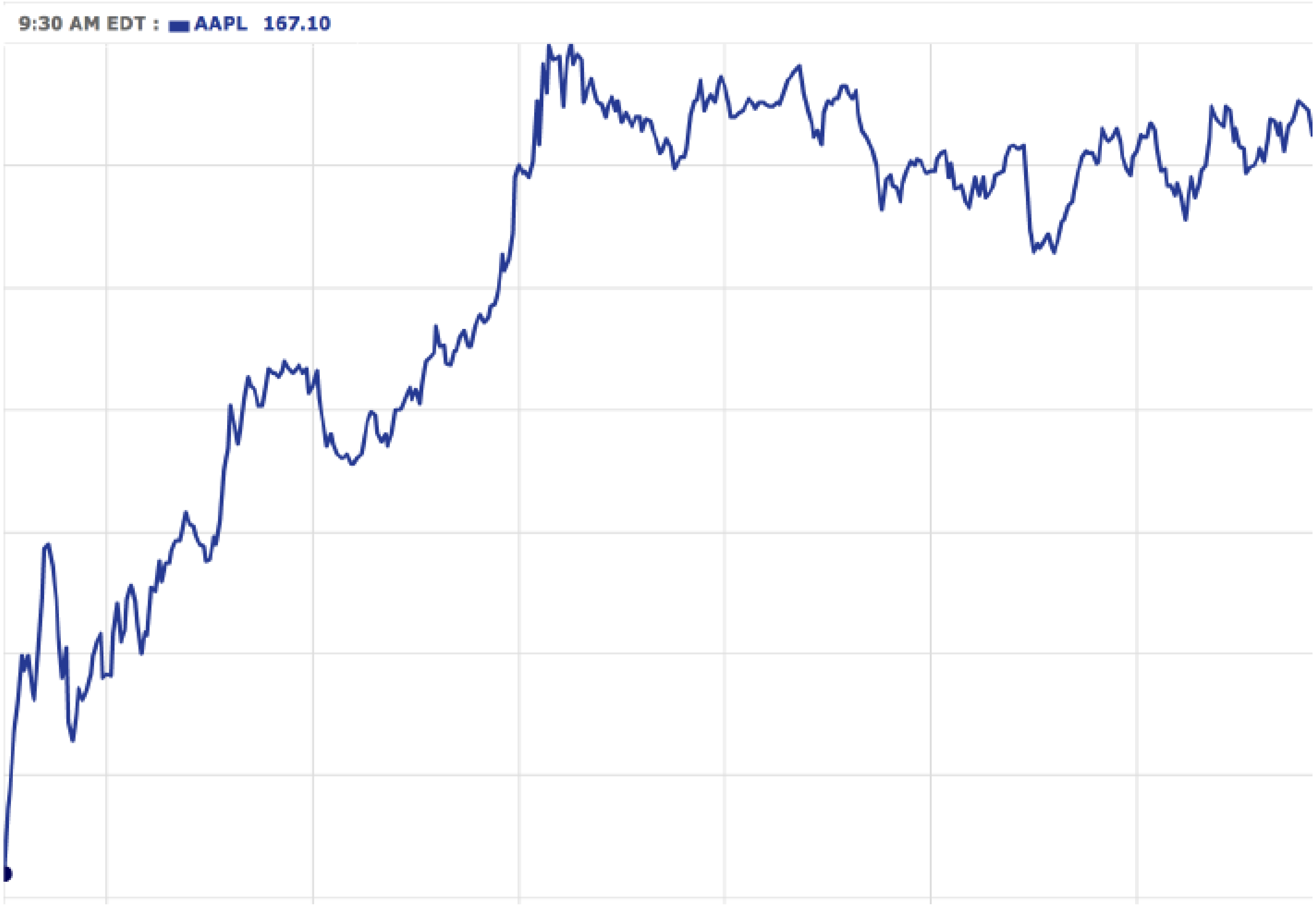


merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p+r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q+1, r)$
        merge(A, p, q, r)

$$T(n) = 2T(n/2) + O(n)$$

$$= \Theta(n \log n)$$

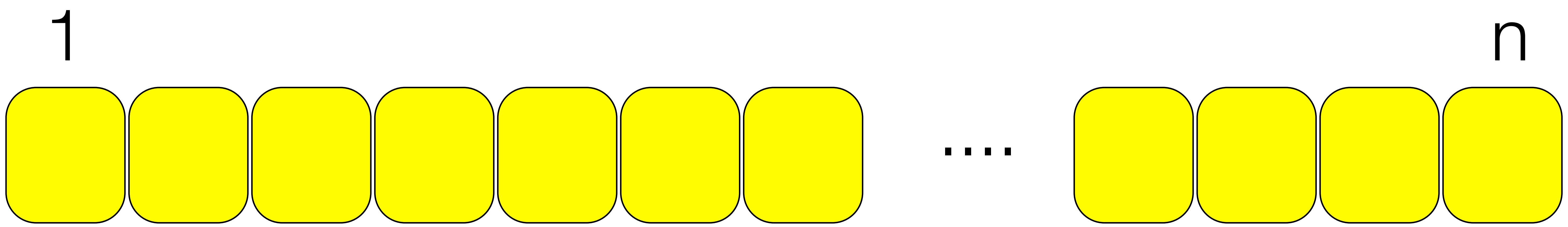
arbitrage


9:30 AM EDT : AAPL 167.10

Open 27.46
Close 27.07
Low 26.65
High 27.69
Vol 33.79K
% Chg -75.68%
BPT 27.07
YAHOO!
FINANCE

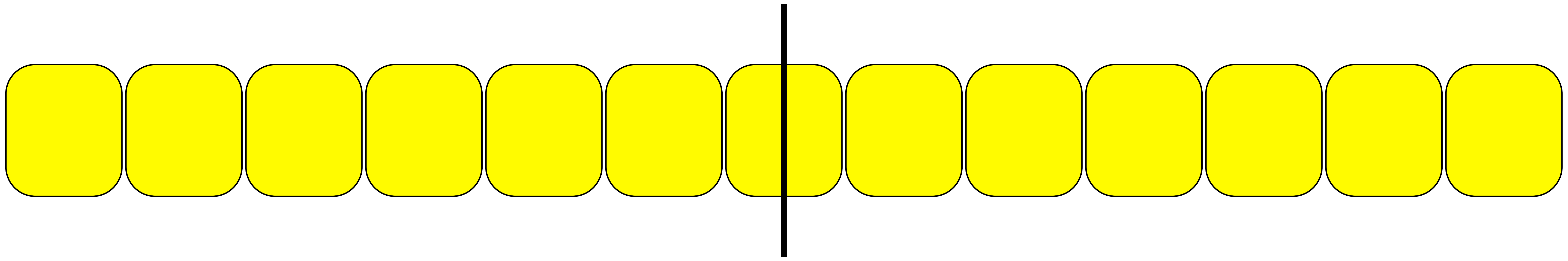

12:38 PM EDT : AIG 40.58
© 2008 Yahoo! Inc.
10:00 AM
11:00 AM
12:00 PM
1:00 PM
2:00 PM
3:00 PM
4:00

input: array of n numbers



goal:

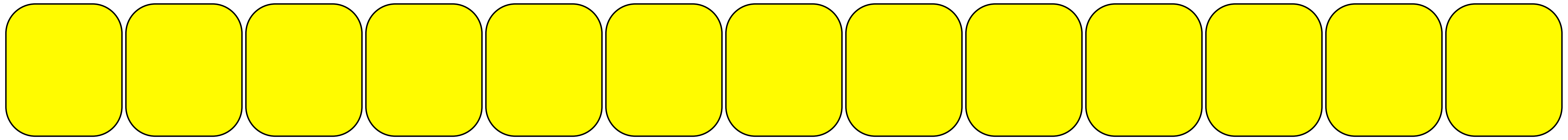
Main idea



Find the best arbitrage opportunity in LEFT and in RIGHT.

Then look for opportunities when you buy on the left and sell on the right.

first attempt



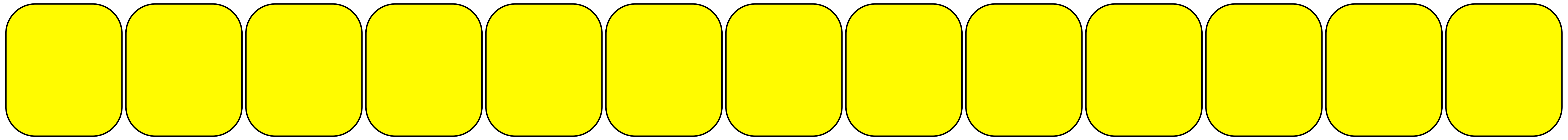
```
arbit(A[1...n])
```

first attempt

```
arbit(A[1...n])
    base case if |A|<=2
    lg = arbit(left(A))
    rg = arbit(right(A))
    minl = min(left(A))
    maxr = max(right(A))
    return max{maxr-minl,lg,rg}
```

T(n) =

first attempt: time $\Theta(n \log n)$



```
arbit(A[1...n])
    base case if |A|<=2
    lg = arbit(left(A))
    rg = arbit(right(A))
    minl = min(left(A))
    maxr = max(right(A))
    return max{maxr-minl,lg,rg}
```

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

better approach

These are the steps that are taking $\Theta(n) time$

better approach

Can we find a solution that has T(n) = 2T(n/2) + O(1) ?

These are the steps that are taking $\Theta(n) time$

better approach

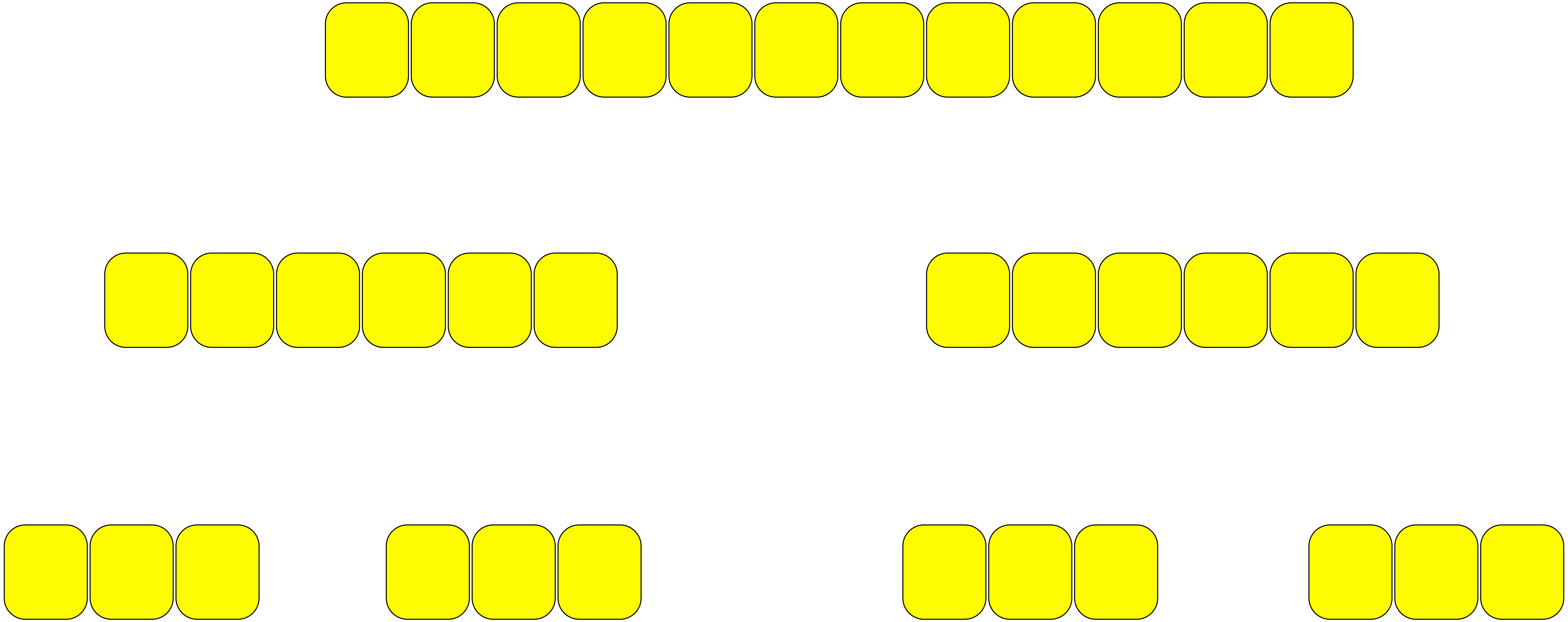
Can we find a solution that has T(n) = 2T(n/2) + O(1) ?

```
minl  = min(left(A))
maxr  = max(right(A))
return max{maxr-minl,lg,rg}
```

These are the steps that are taking $\Theta(n) time$

first attempt

`arbit(A[1...n])`



second attempt

```
arbit2(A[1…n])                    // Returns {best trade,min,max}
 base case if |A|<=2
```

second attempt

```
arbit2(A[1...n])                // Returns {best trade,min,max}
   base case if |A|<=2, …
   (lg,minl,maxl) = arbit2(left(A))
   (rg,minr,maxr) = arbit2(right(A))
   return max{maxr-minl,lg,rg},
          min{minl, minr},
          max{maxl, maxr}
```

second attempt

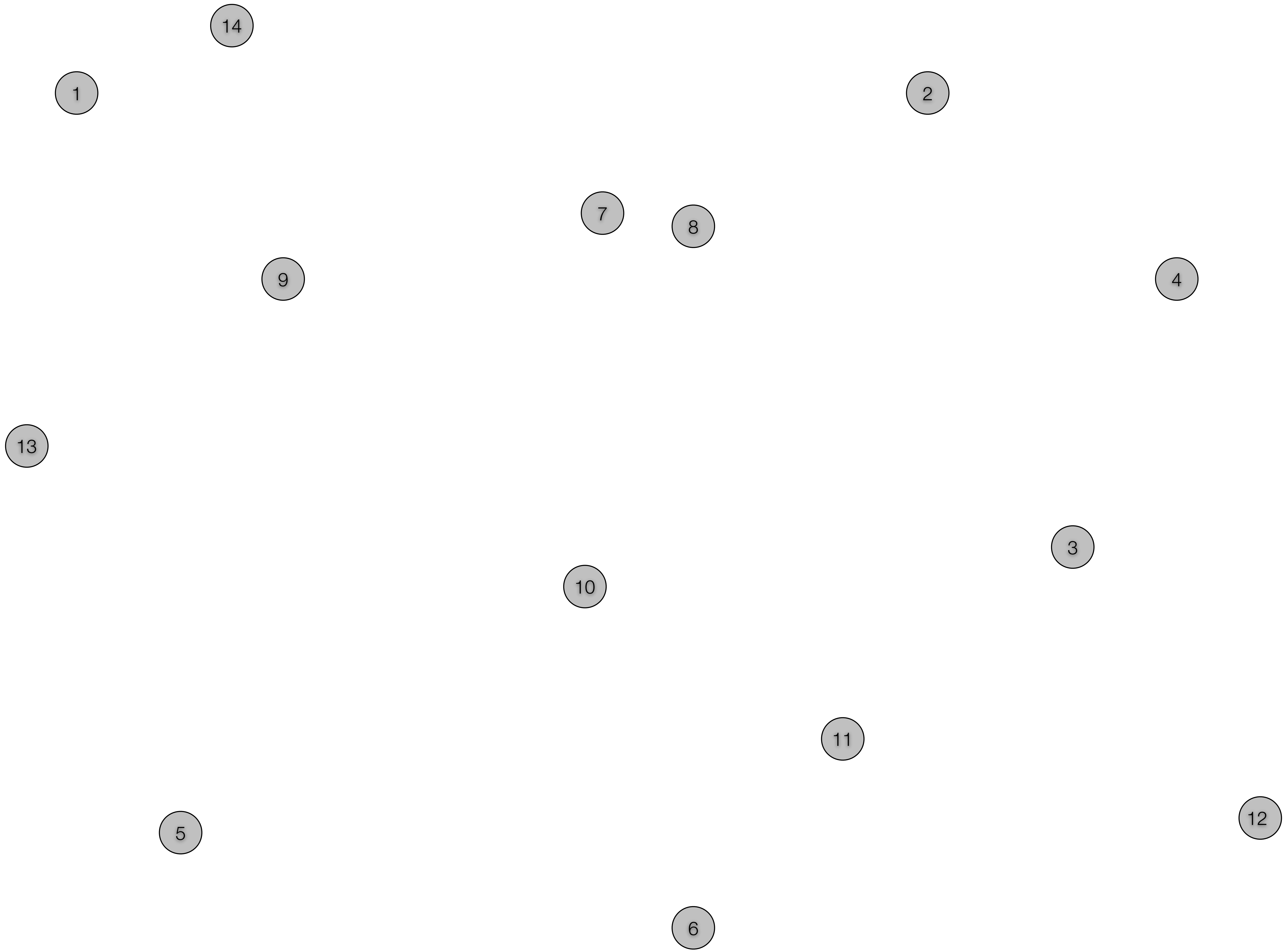
```
arbit2(A[1...n])                    // Returns {best trade,min,max}
    base case if |A|<=2, …
    (lg,minl,maxl) = arbit2(left(A))
    (rg,minr,maxr) = arbit2(right(A))
    return max{maxr-minl,lg,rg},
           min{minl, minr},
           max{maxl, maxr}
```

New runtime is $T(n) = 2T(n/2) + \Theta(1) = \Theta(n)$

closest pair

of points

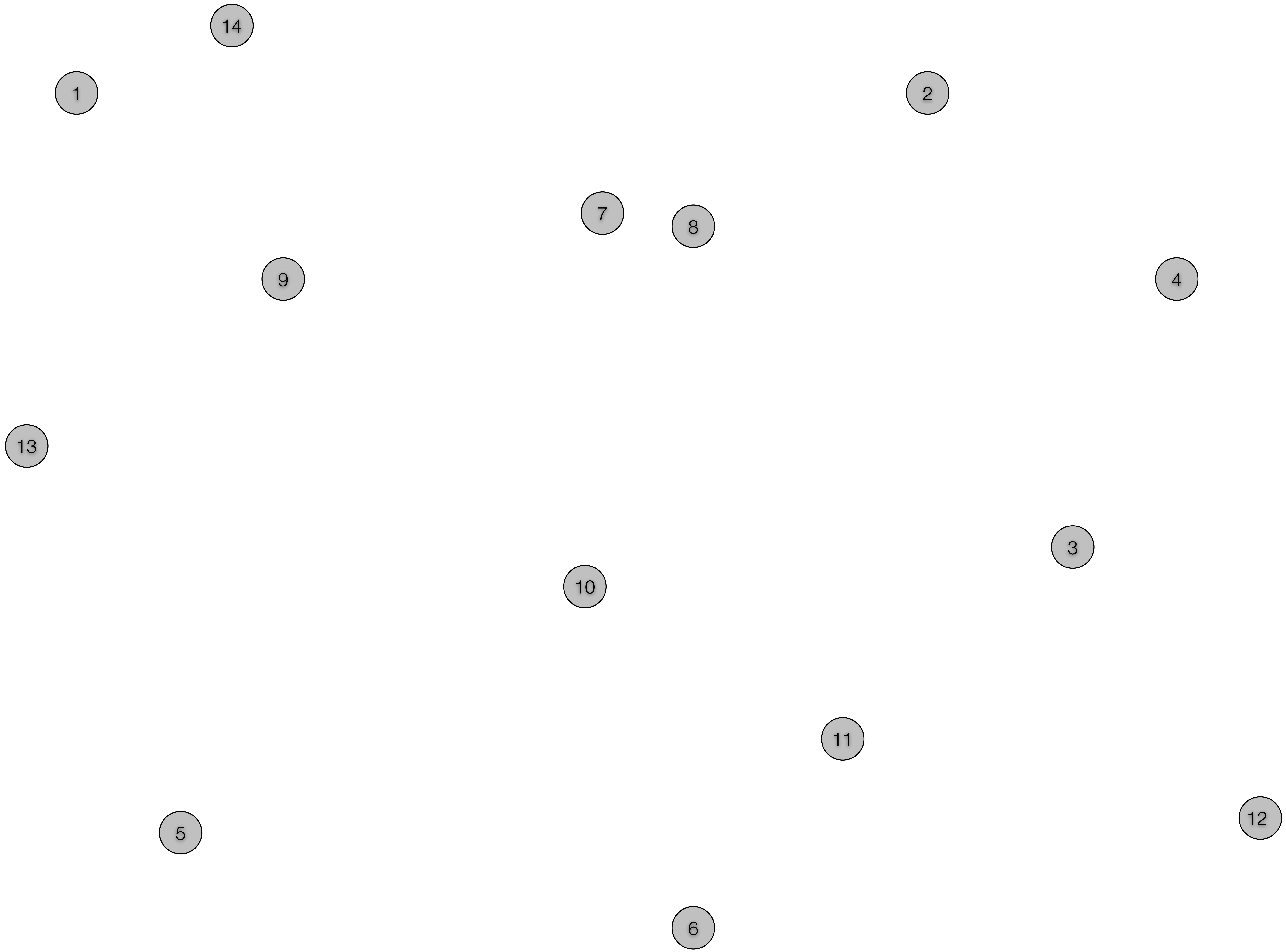
Simple brute force approach takes $\Theta(n^2)$



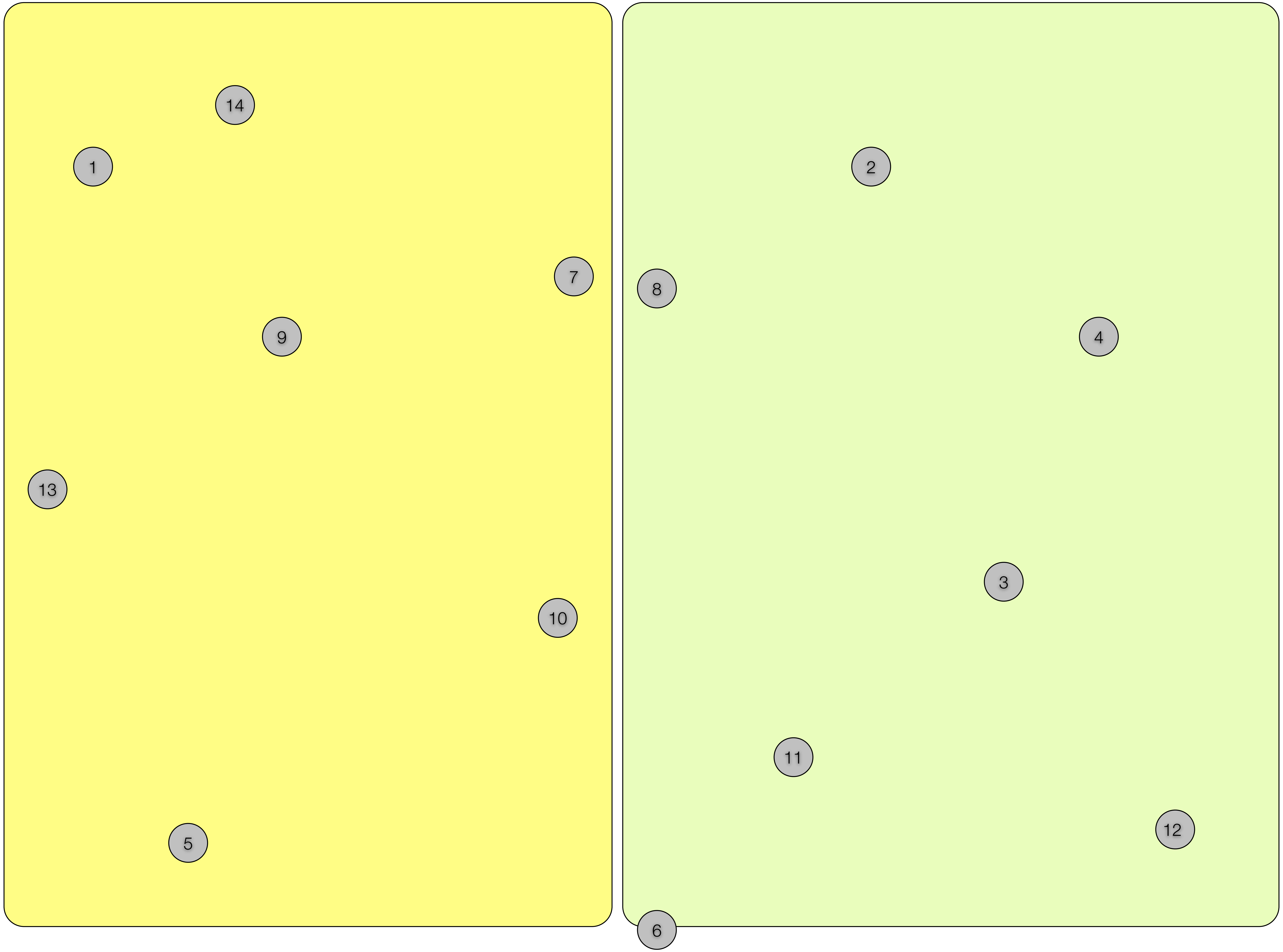
Assume all points have distinct x & y coordinates.

solve the large problem by

solving smaller problems
and combining solutions

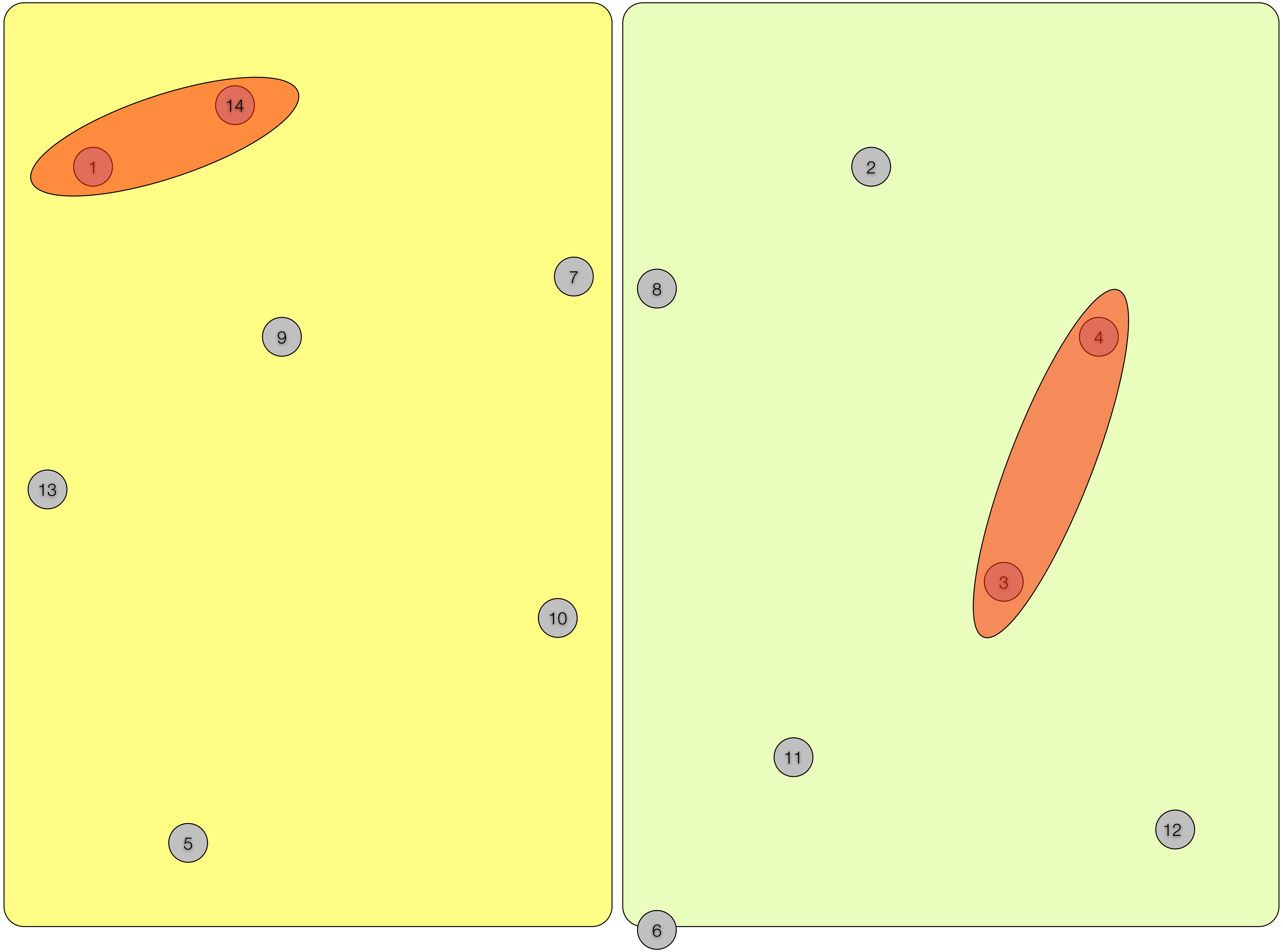

14
1
2
7
8
9
4
13
3
10
11
12
5
6

Divide & Conquer



Divide & Conquer

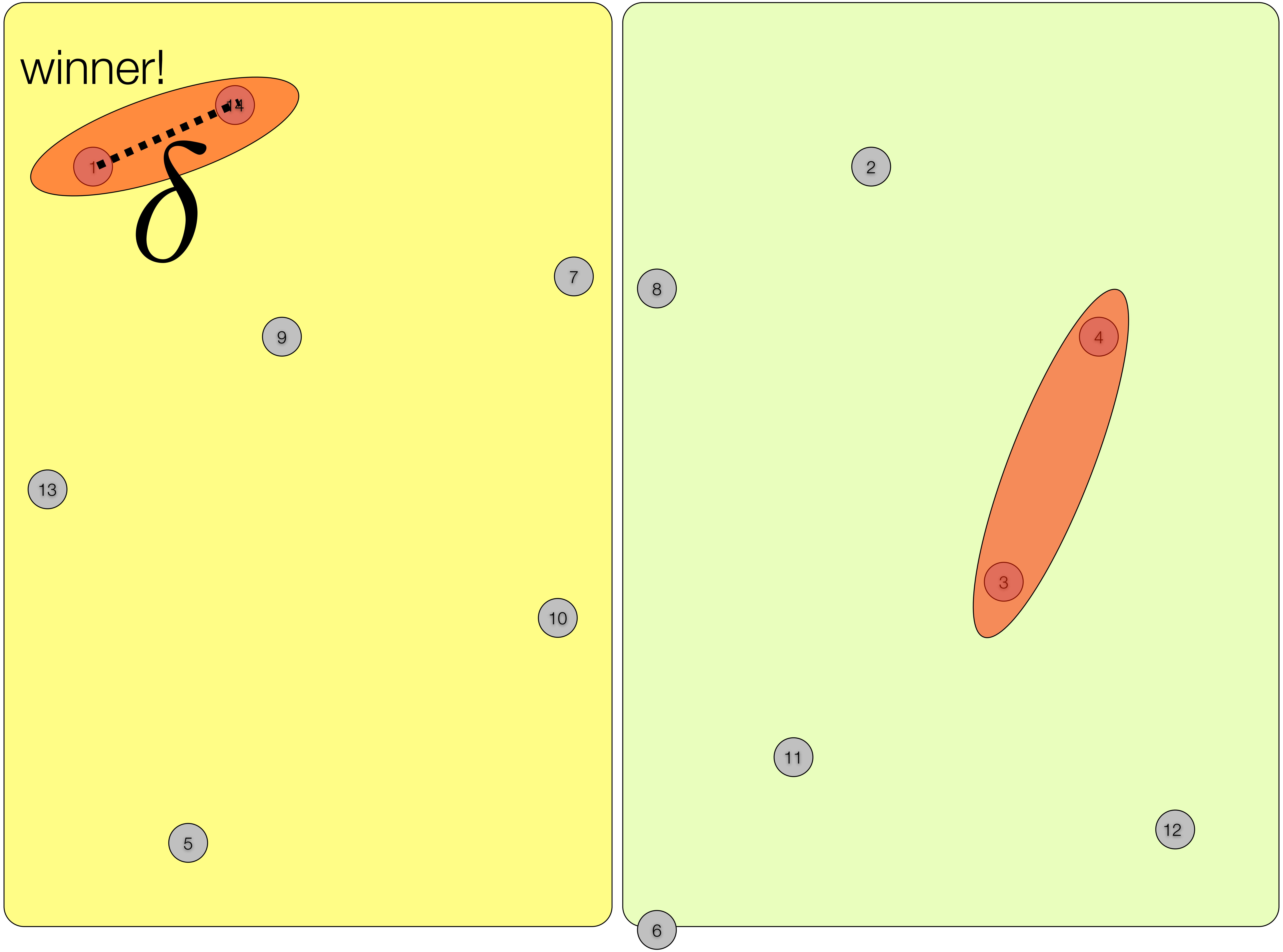
Find closest pair on the left half.



Find closest pair on the right half.

Divide & Conquer

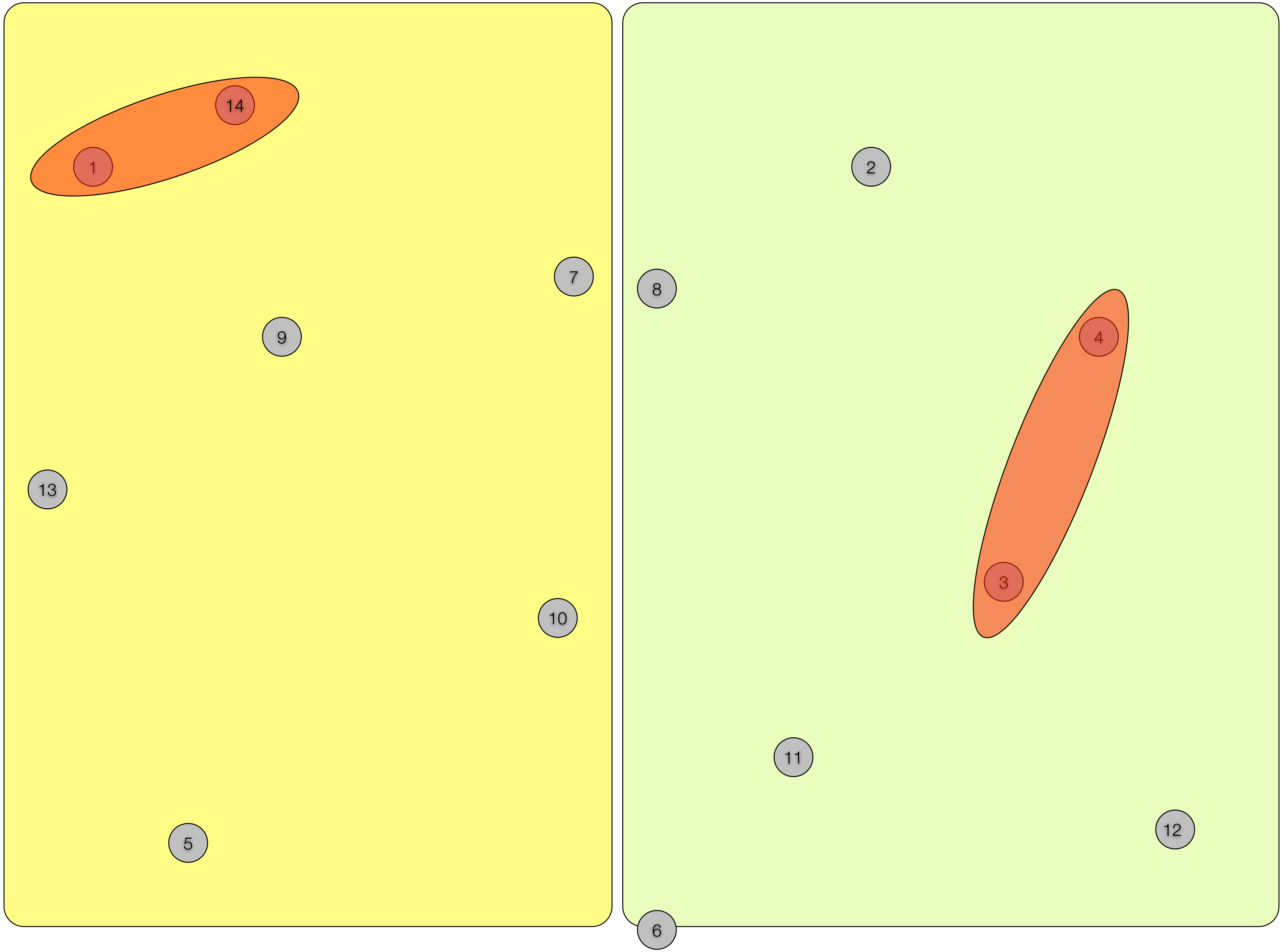
Find closest pair on the left half.



Find closest pair on the right half.

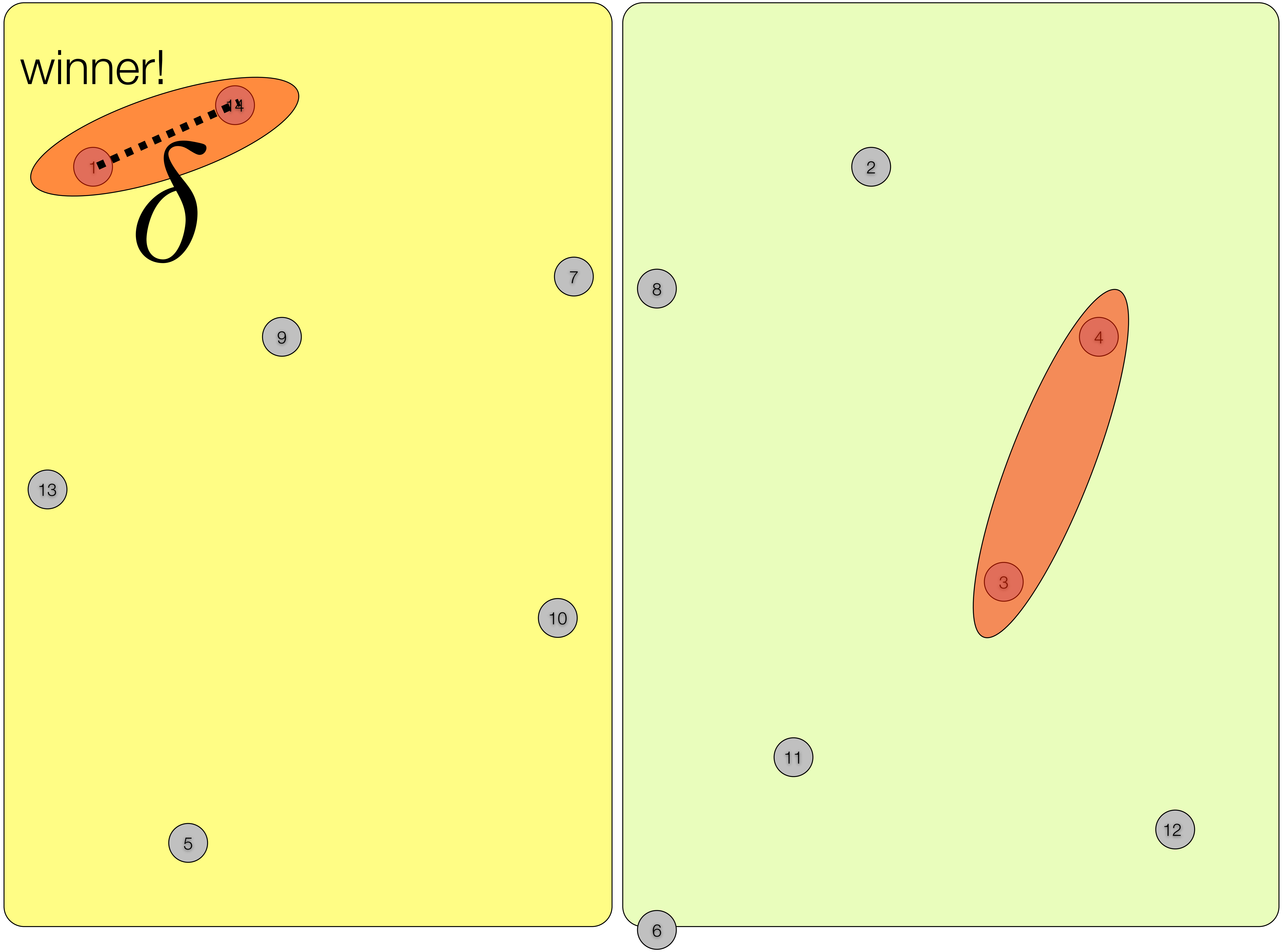
Divide & Conquer

Now look for pairs between the left and right that are closer.



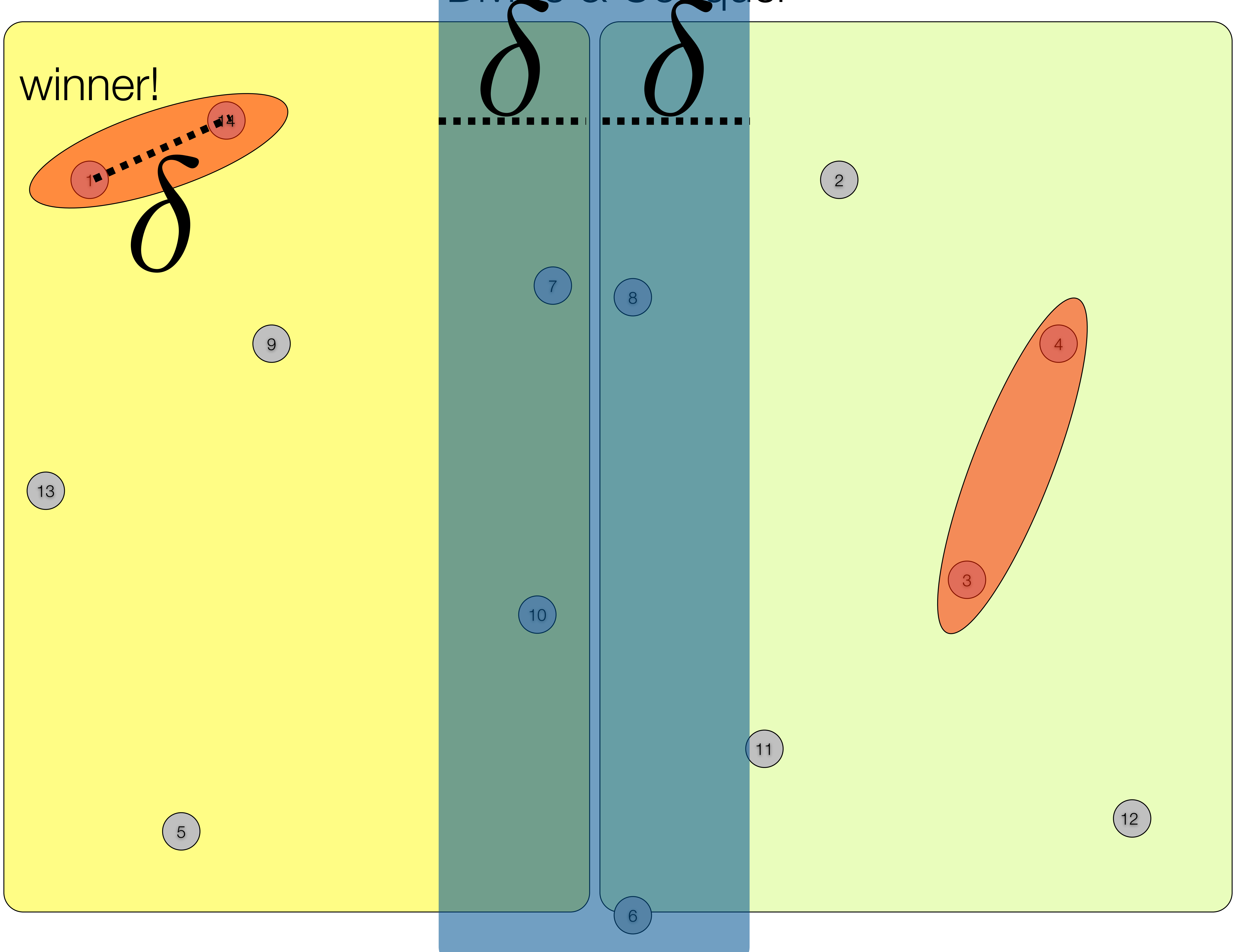
Divide & Conquer

Now look for pairs between the left and right that are closer.

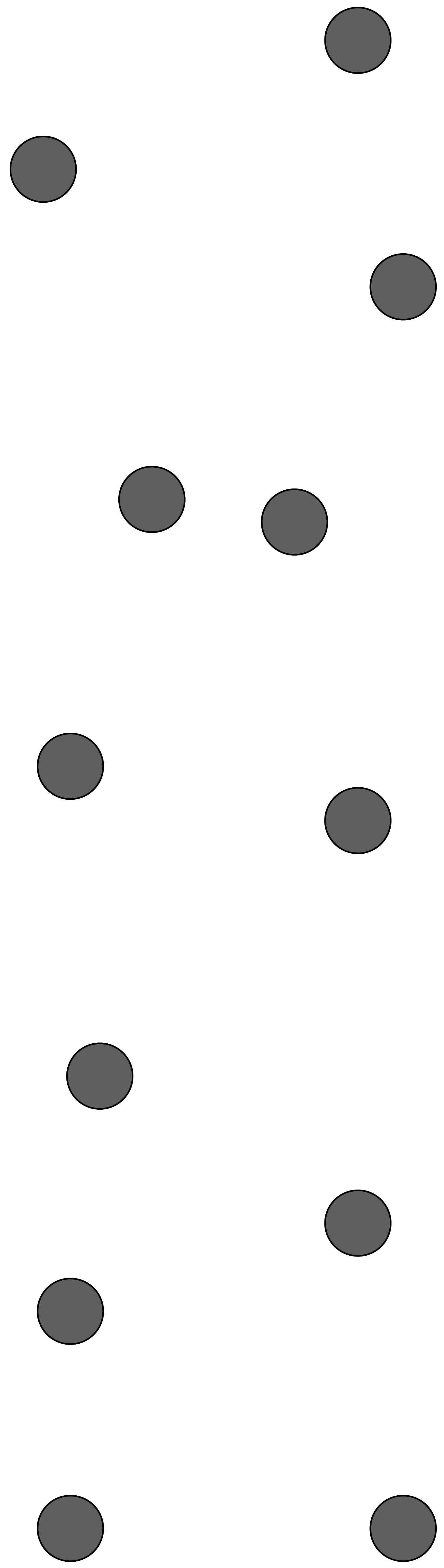


Divide & Conquer

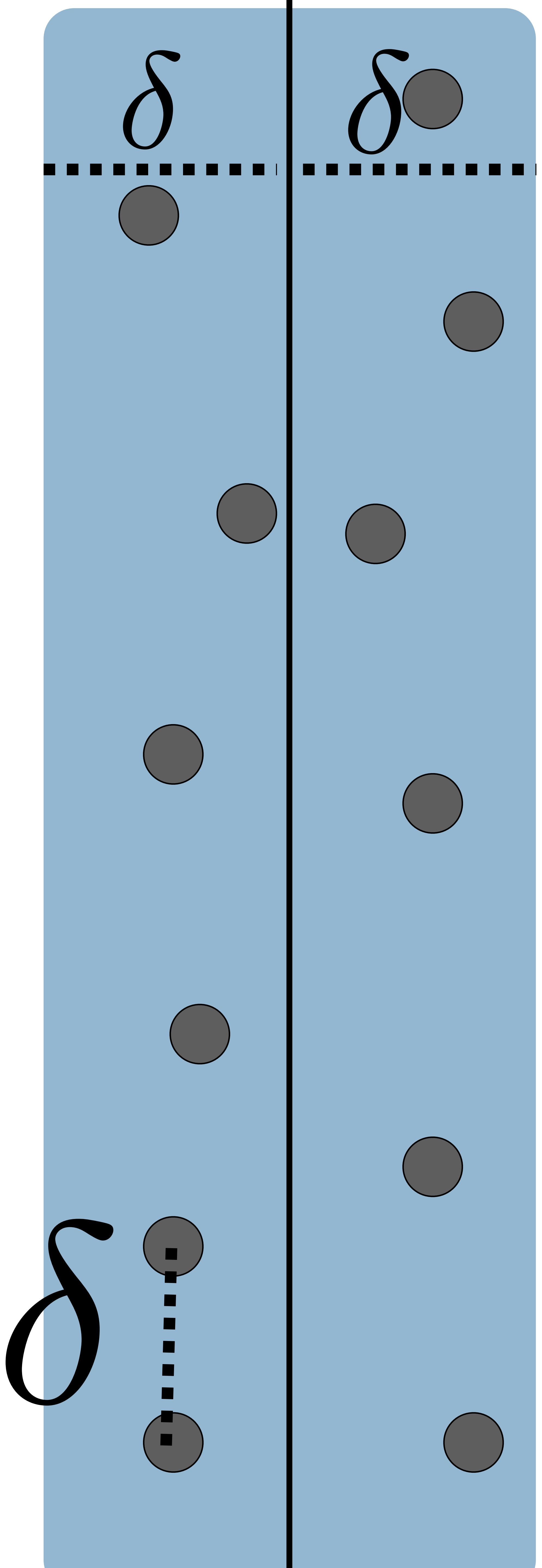
Now look for pairs between the left and right that are closer.



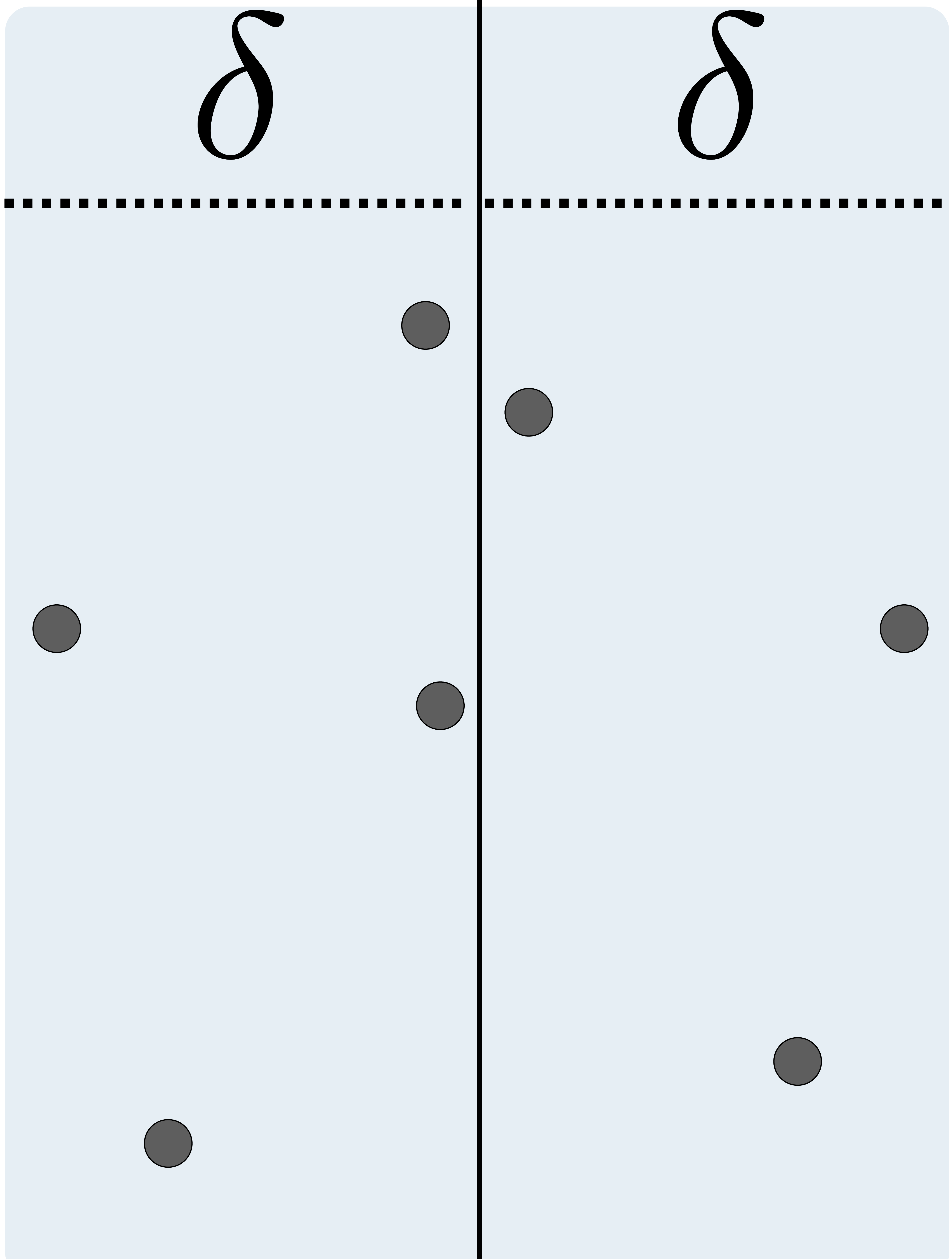
What if the input points are like this?



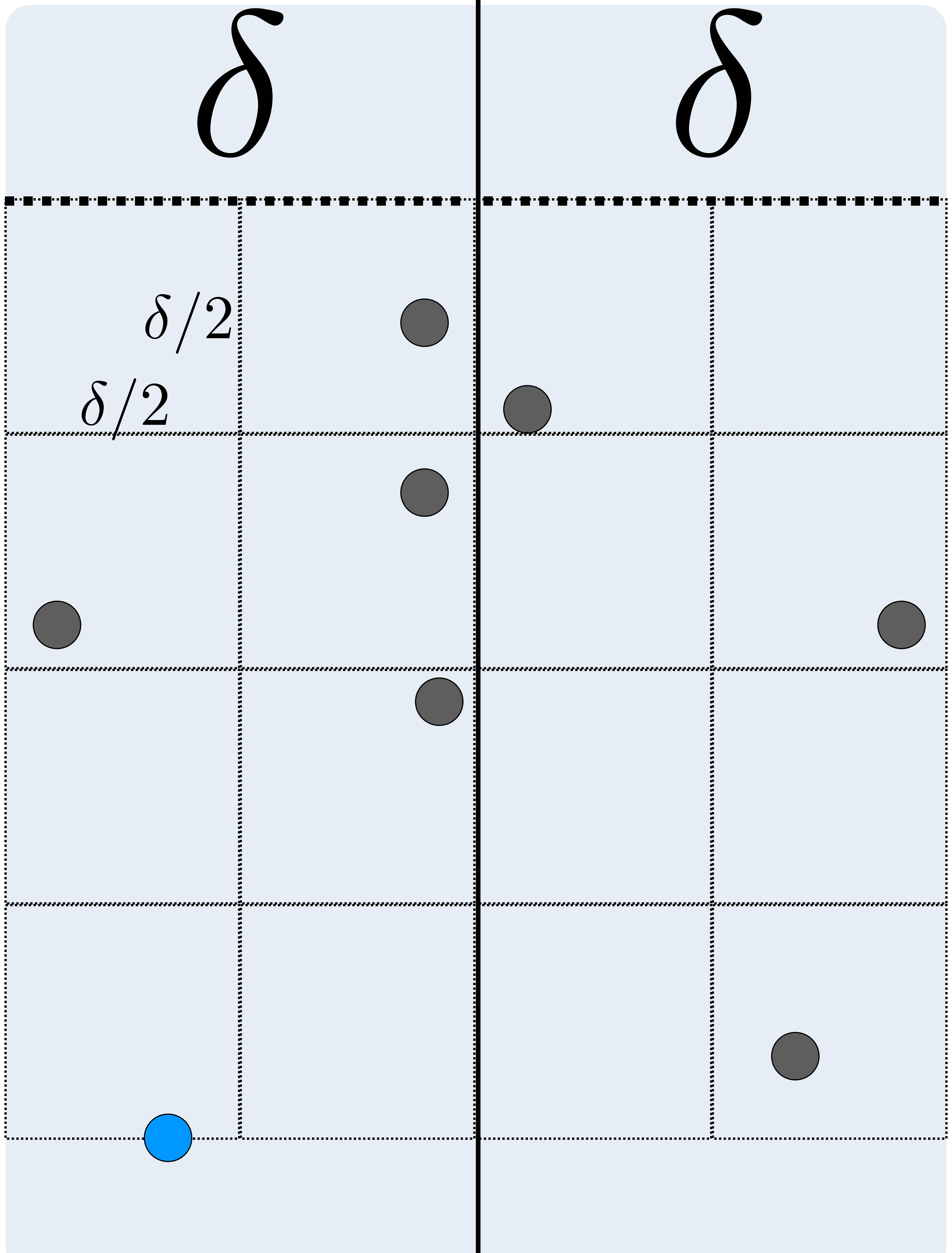
Then all of the points are within δ of the middle. If we need to check all of the points, we are back to $O(n^2)$

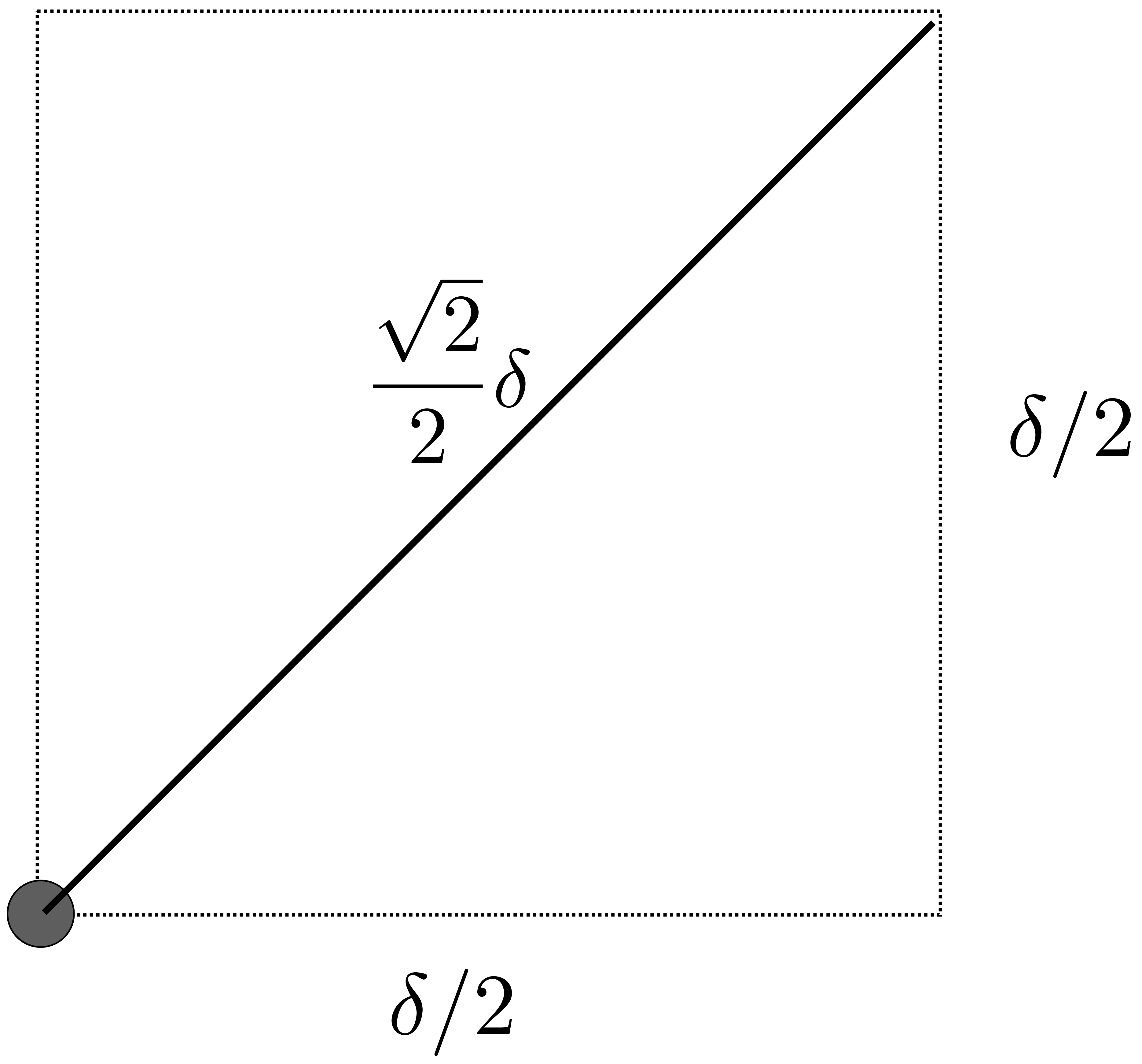


But we have extra information! The only candidates for closest pair are within δ of each other. How can we use this info?

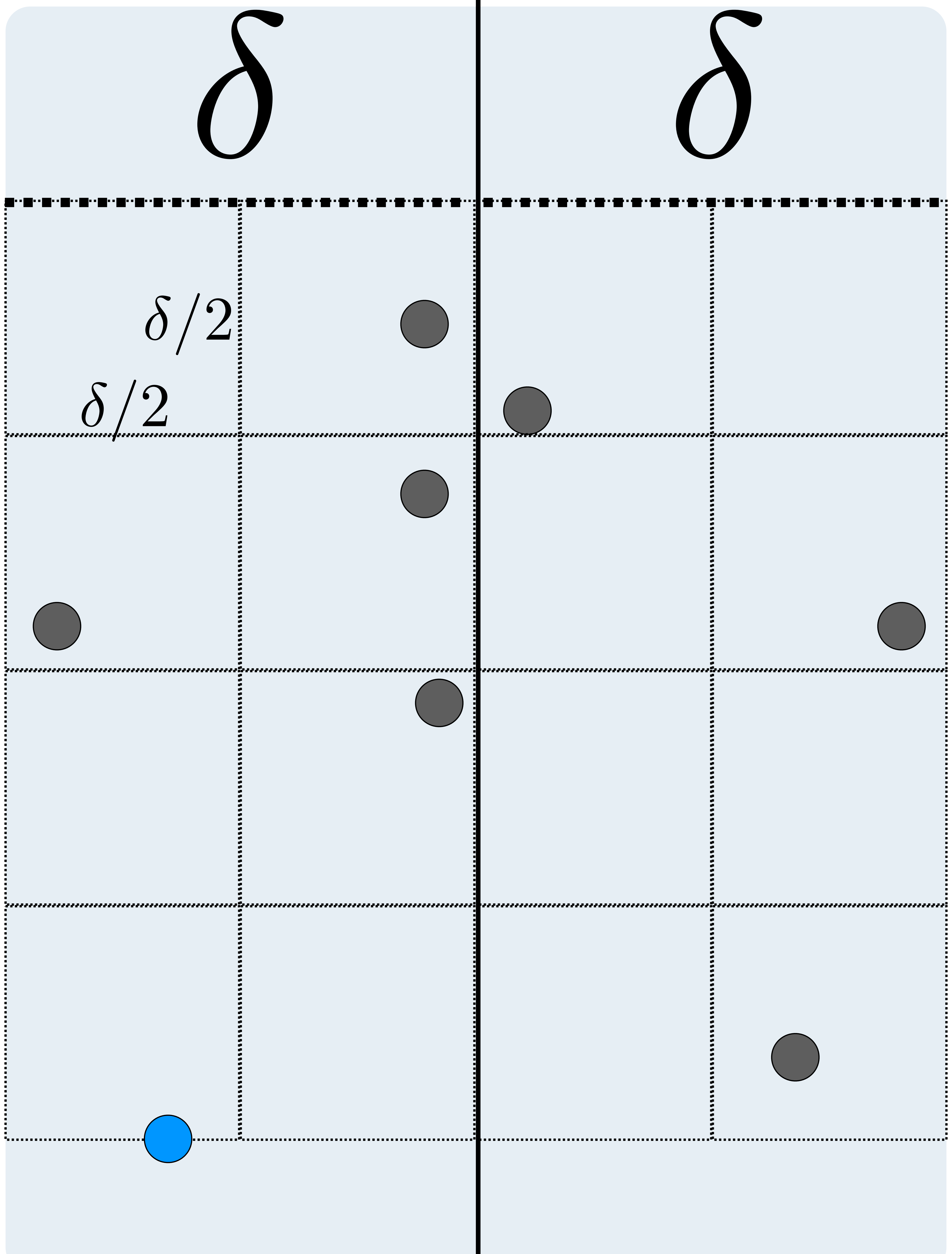


Imagine there is a grid of cubbies starting at the lowest Y point



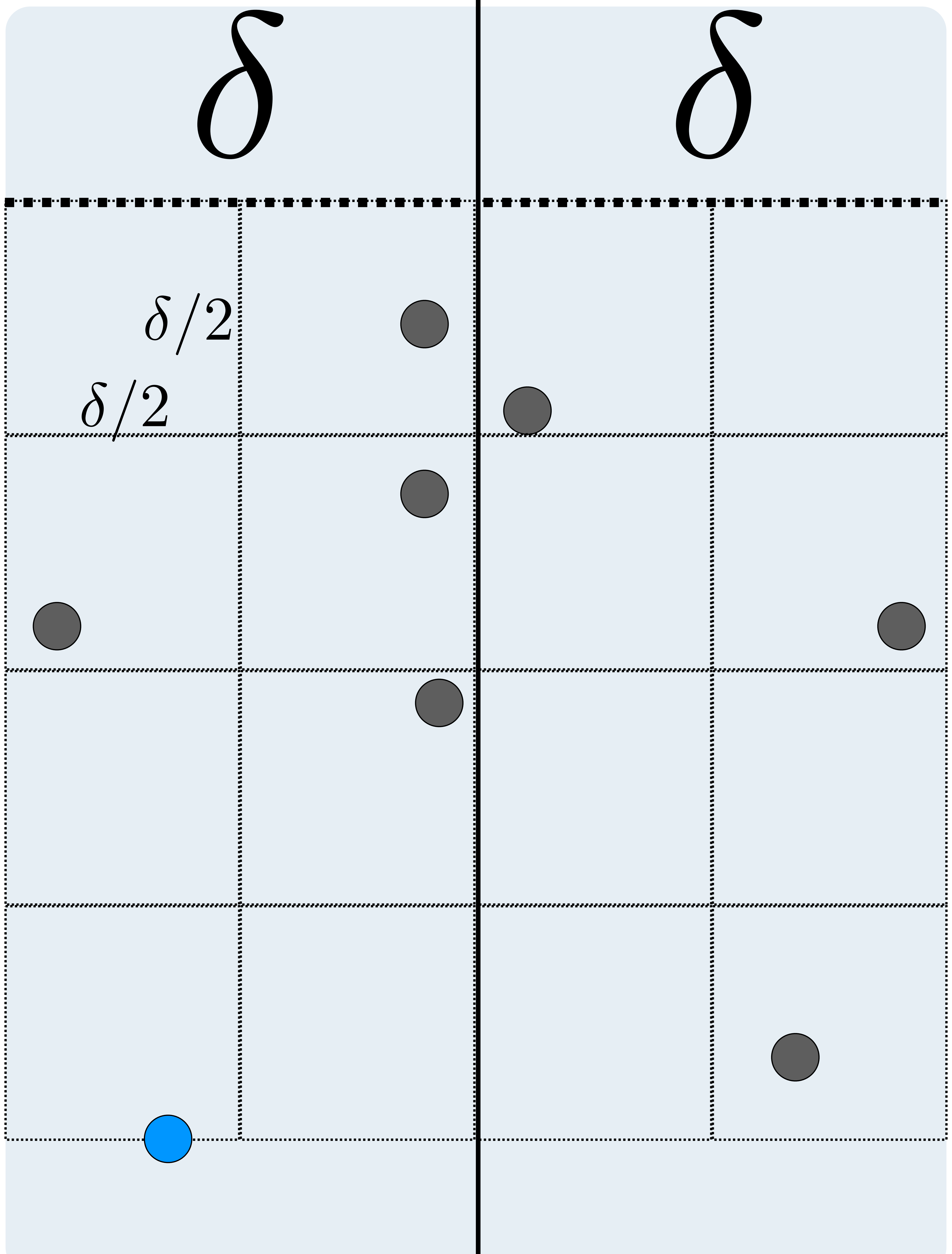


A grid this size has a diagonal that is smaller than delta.That means each grid box can only have 1 point in it.

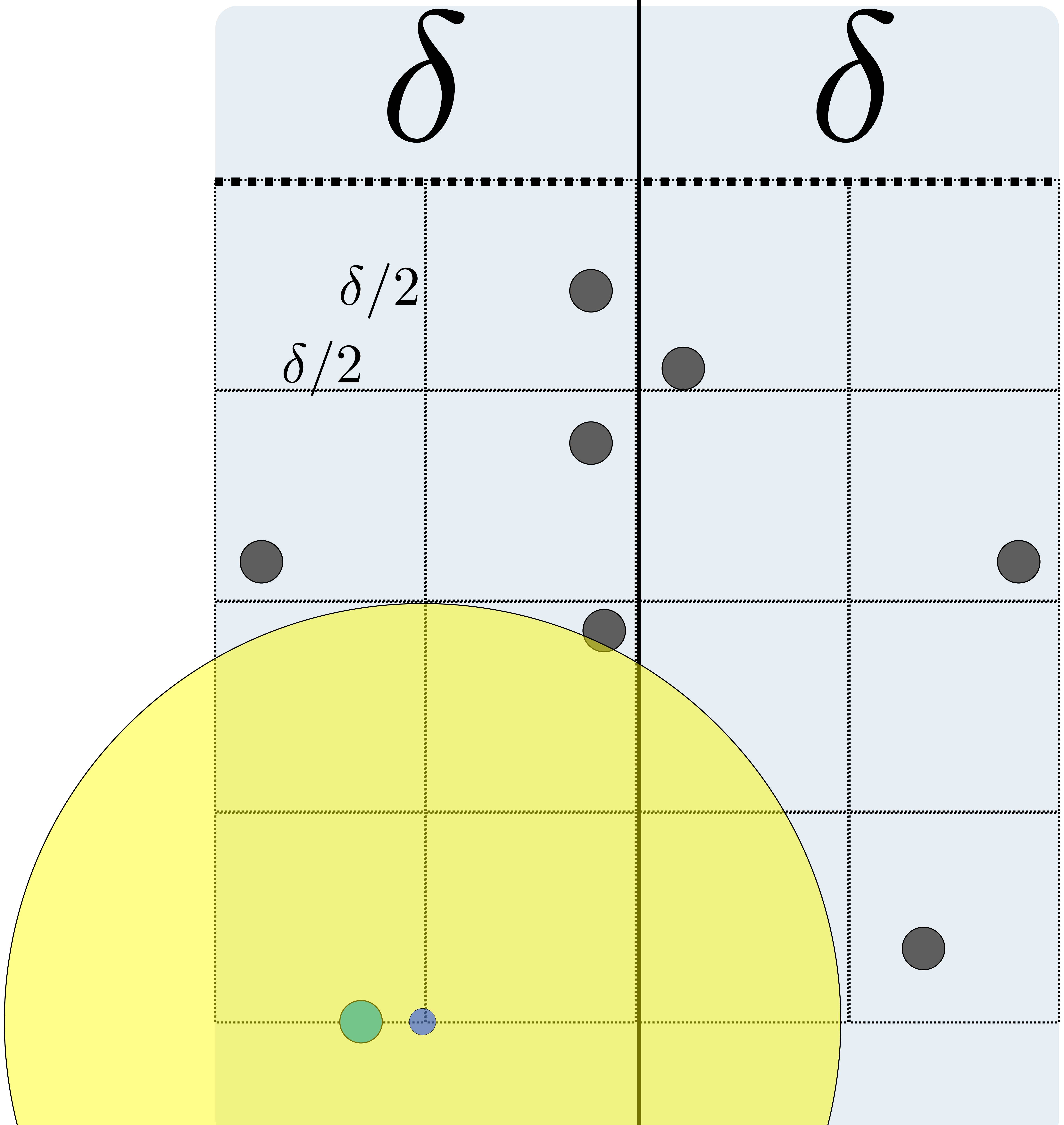


FACT: At most 1 point in each cubby

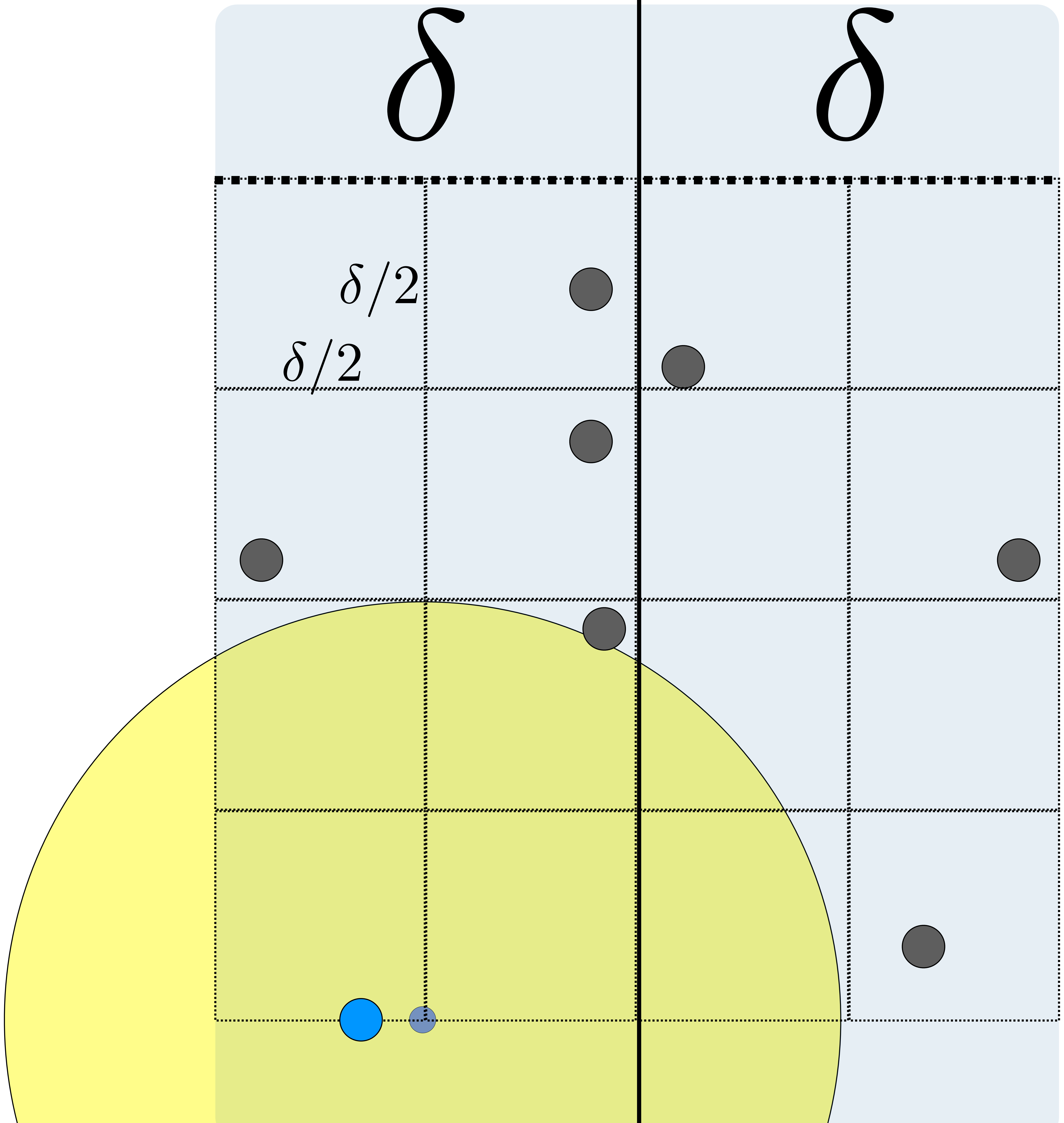
Claim: If there is another point closer than δ, then it must be among the next 15 points sorted by y-coordinate.



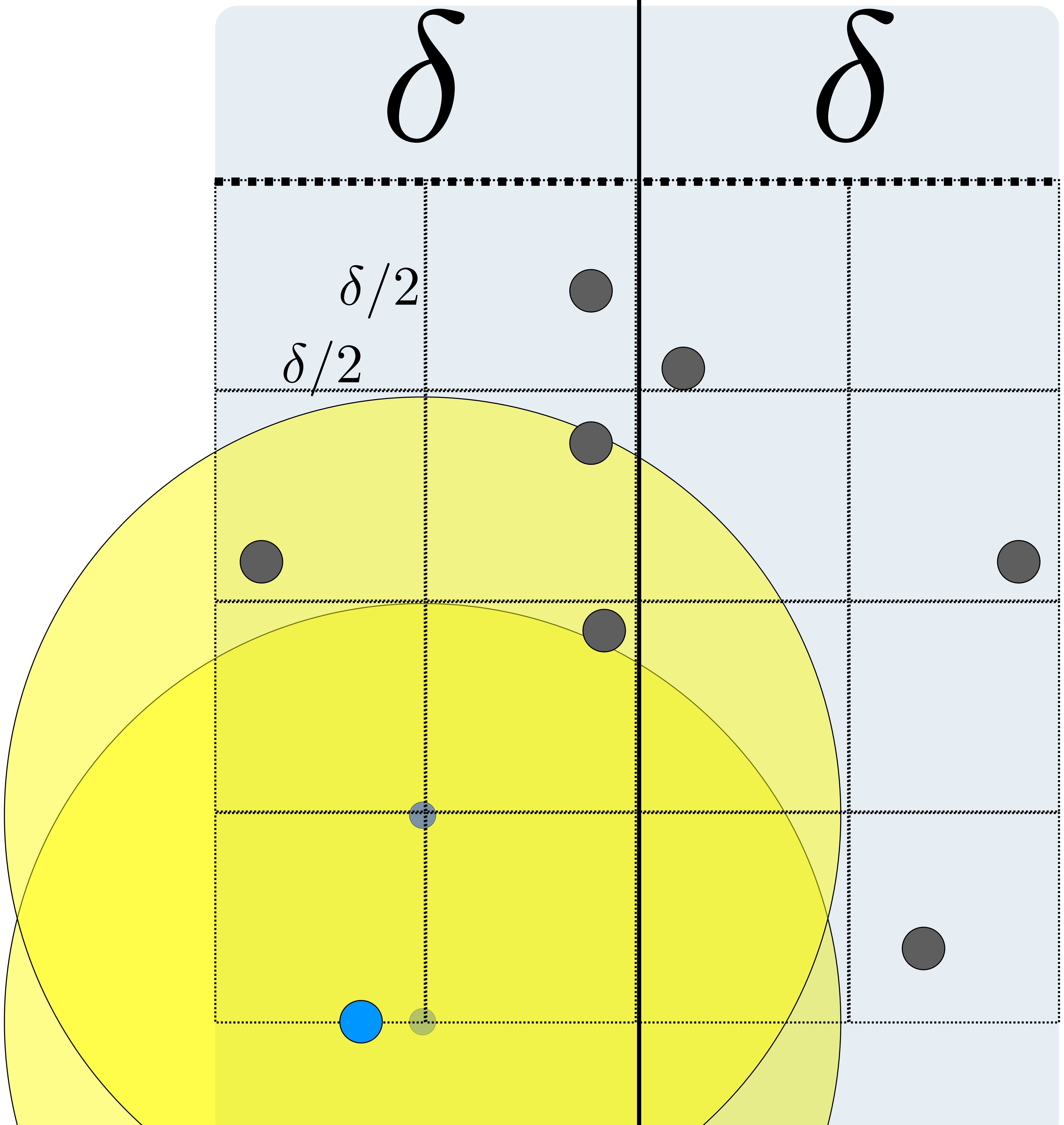
FACT: At most 1 point in each cubby



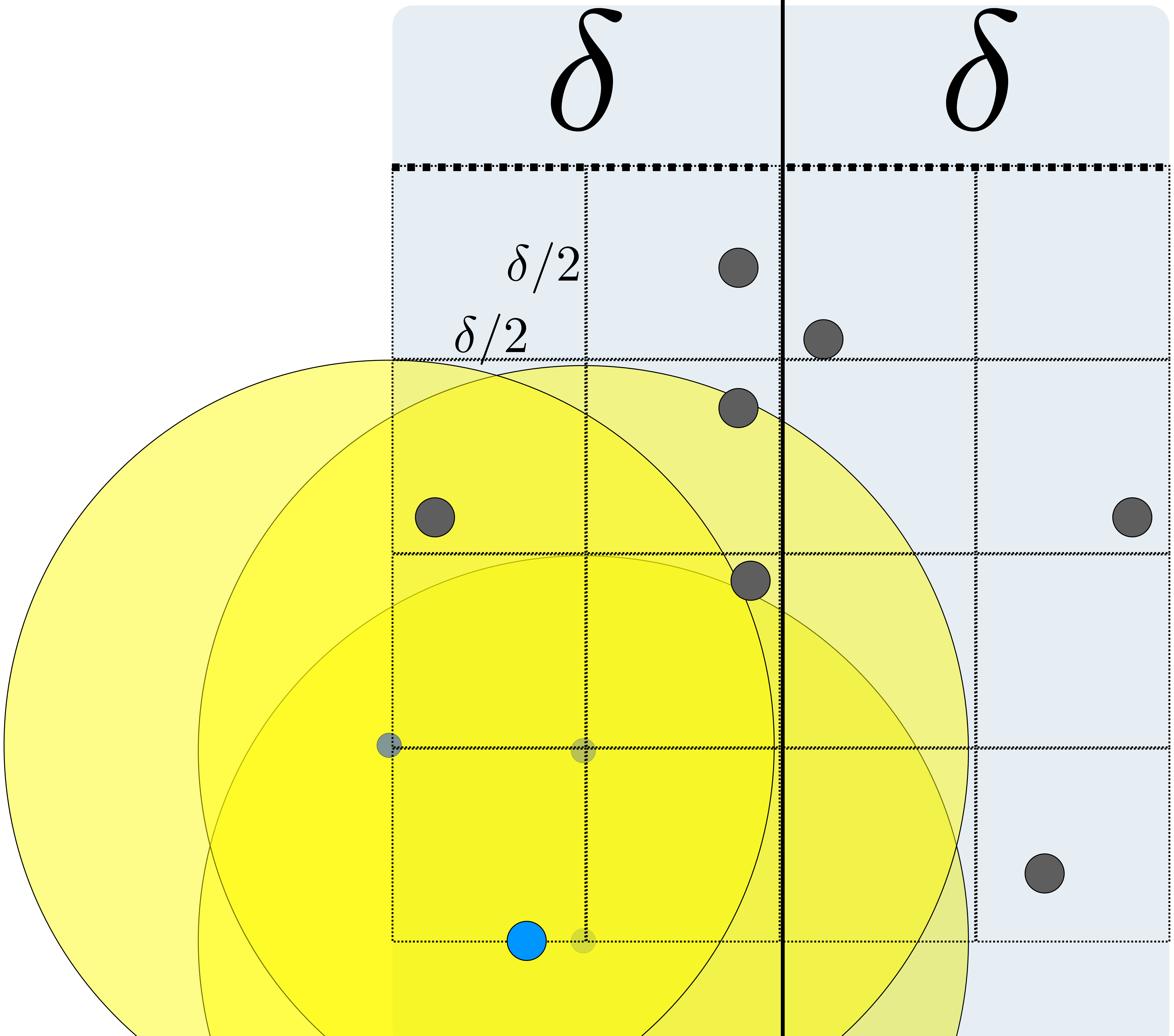
FACT: <=1 point per cubby



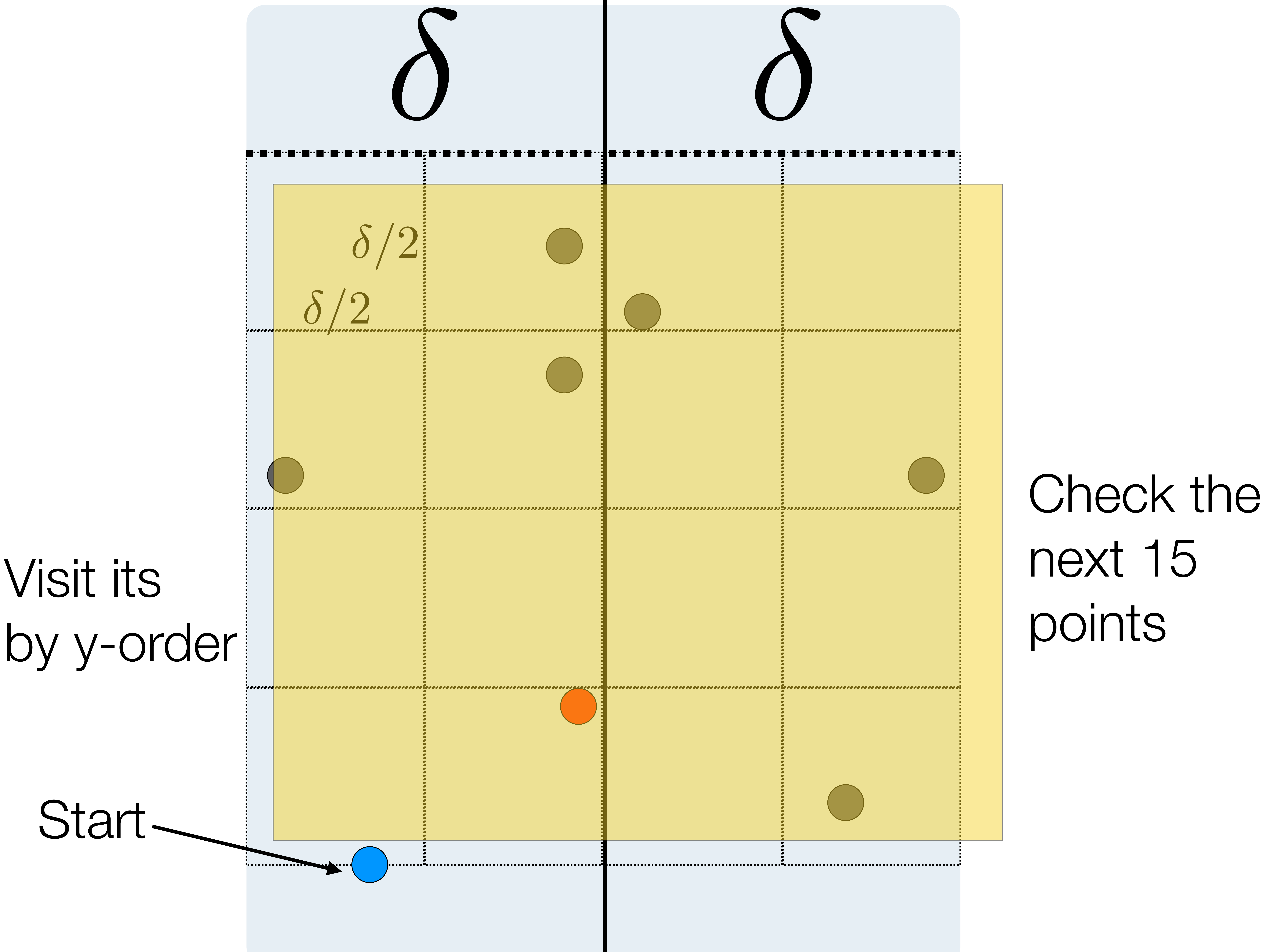
FACT: <=1
point per
cubby



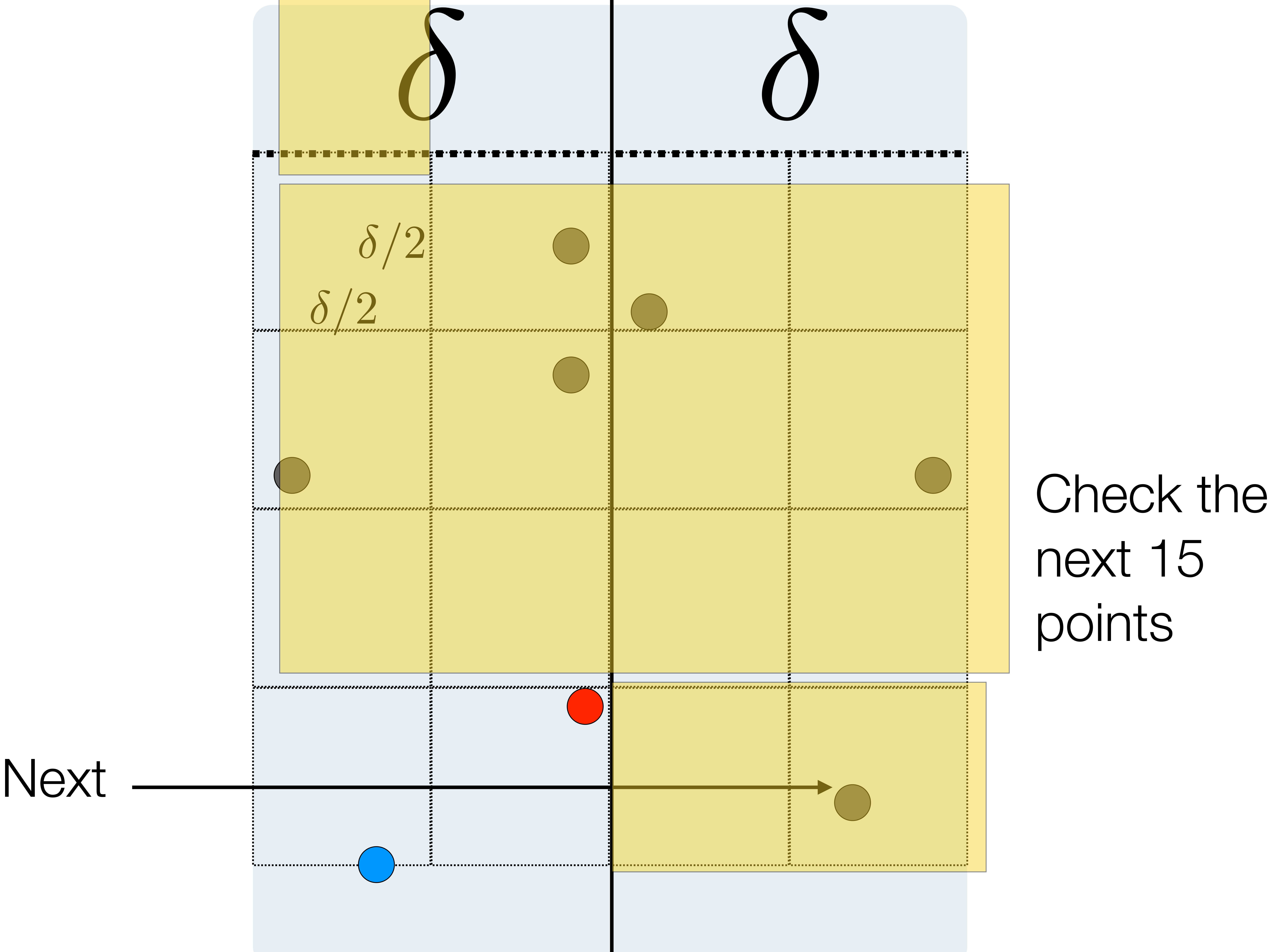
FACT: <=1 point per cubby



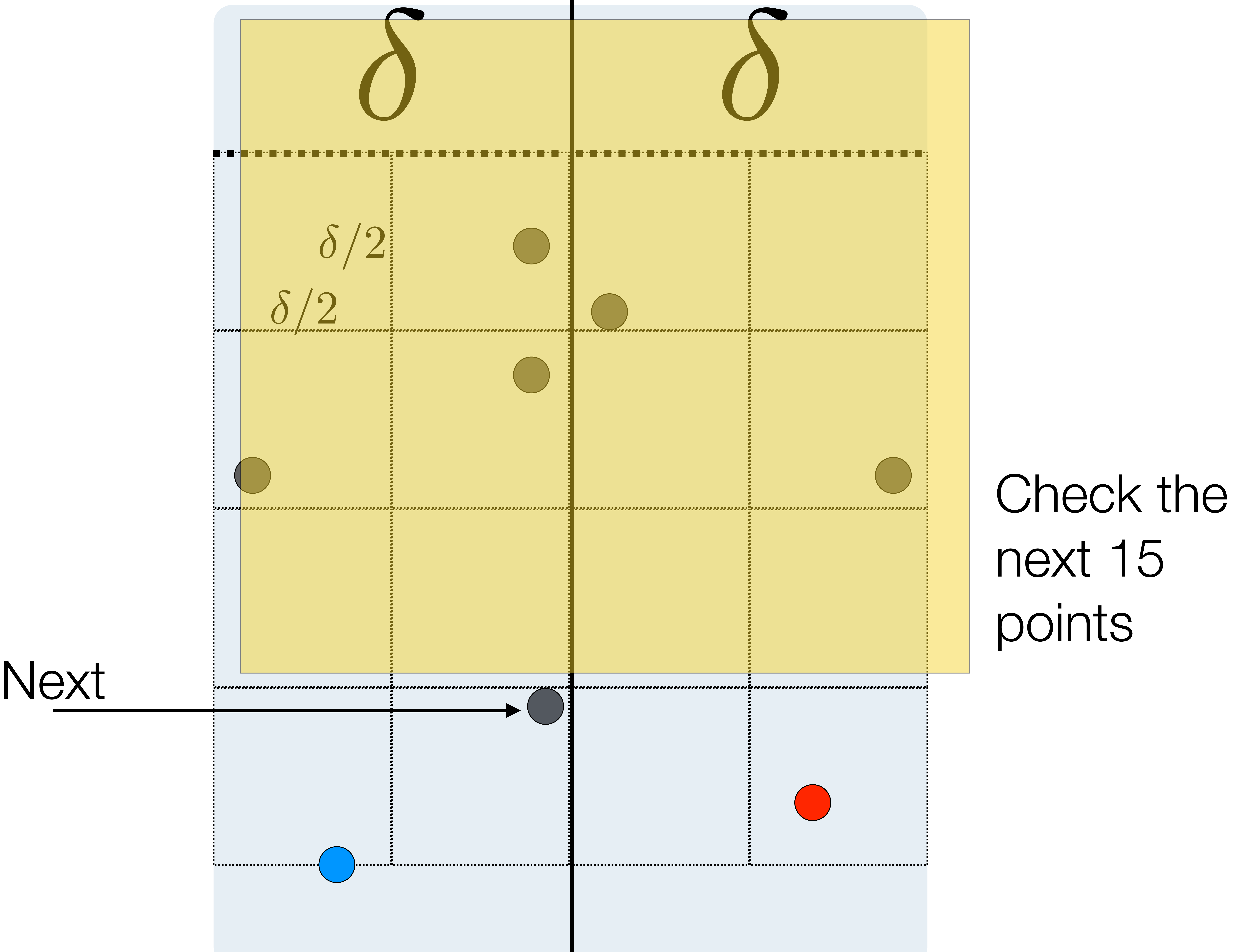
FACT: <=1 point per cubby



δ
δ
$\delta/2$
$\delta/2$
Check the next 15 points
Visit its by y-order
Start



δ
δ
$\delta/2$
$\delta/2$
Check the next 15 points
Next


δ
δ
δ/2
δ/2
Next
Check the next 15 points

Closest(P)

)

Closest(P)

// returns the minimum distance delta
// and the closest pair Romeo, Juliet

Base Case: If <8 points, brute force.

1. Let q be the “middle-element” of points

2. Divide P into Left, Right according to q

3. delta,r,j = MIN(Closest(Left) ,   Closest(Right))

4. Mohawk = { Scan P, add pts that are <delta from q.x }

5. For each point p in Mohawk (in y-order):

    Compute distance between p and its next 15 neighbors
    Update delta,r,j if any pair (x,y) is < delta

6. Return (delta,r,j)

Closest(P)

// returns the minimum distance delta
// and the closest pair Romeo, Juliet

Base Case: If <8 points, brute force.

1. Let q be the “middle-element” of points

2. Divide P into Left, Right according to q

3. delta,r,j = MIN(Closest(Left) , Closest(Right))

4. Mohawk = { Scan P, add pts that are <delta from q.x }

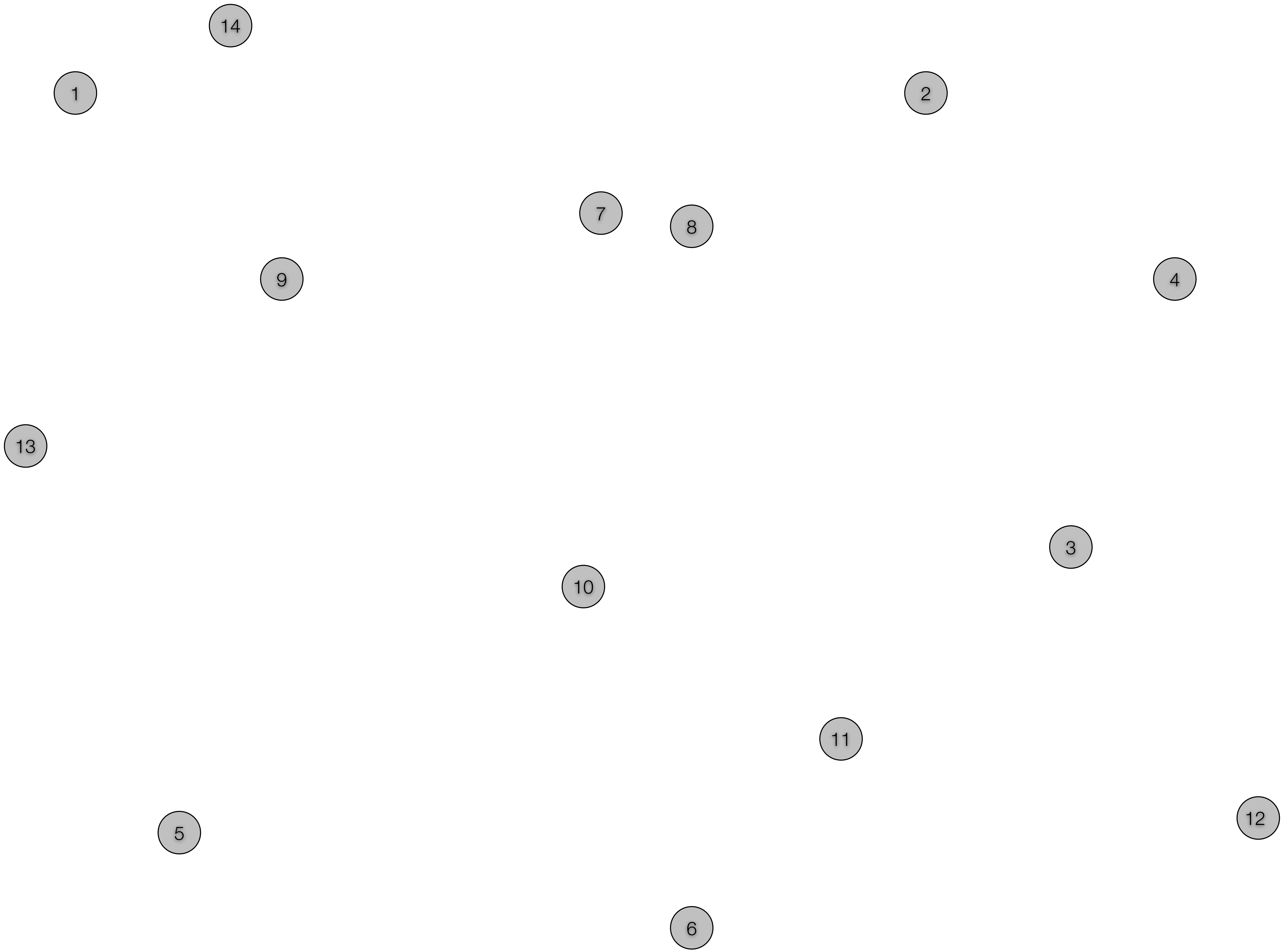
5. For each point p in Mohawk (in y-order):

   Compute distance between p and its next 15 neighbors
   Update delta,r,j if any pair (x,y) is < delta

6. Return (delta,r,j)

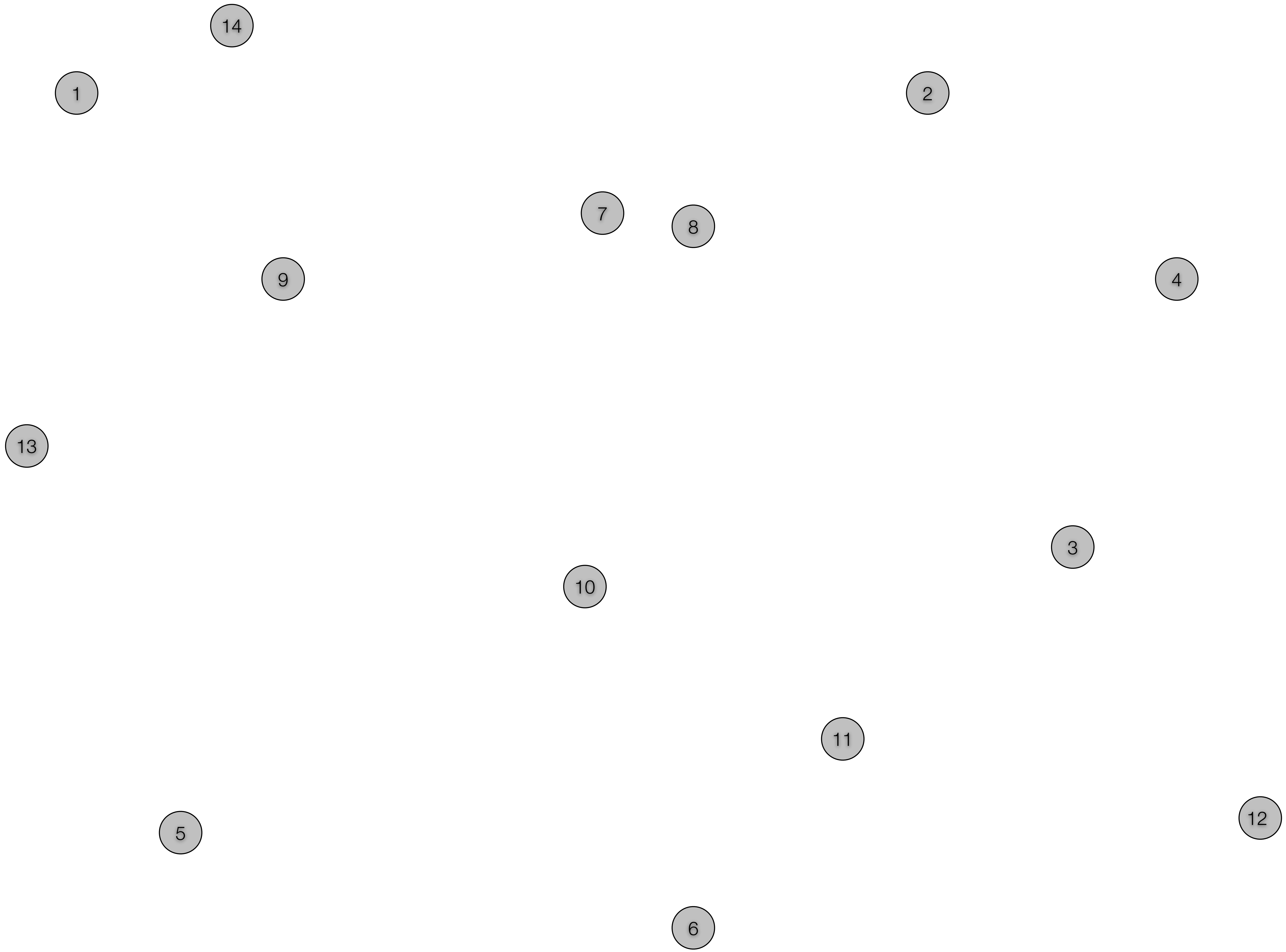
Can be reduced to 7!

Details: How to do step 1?



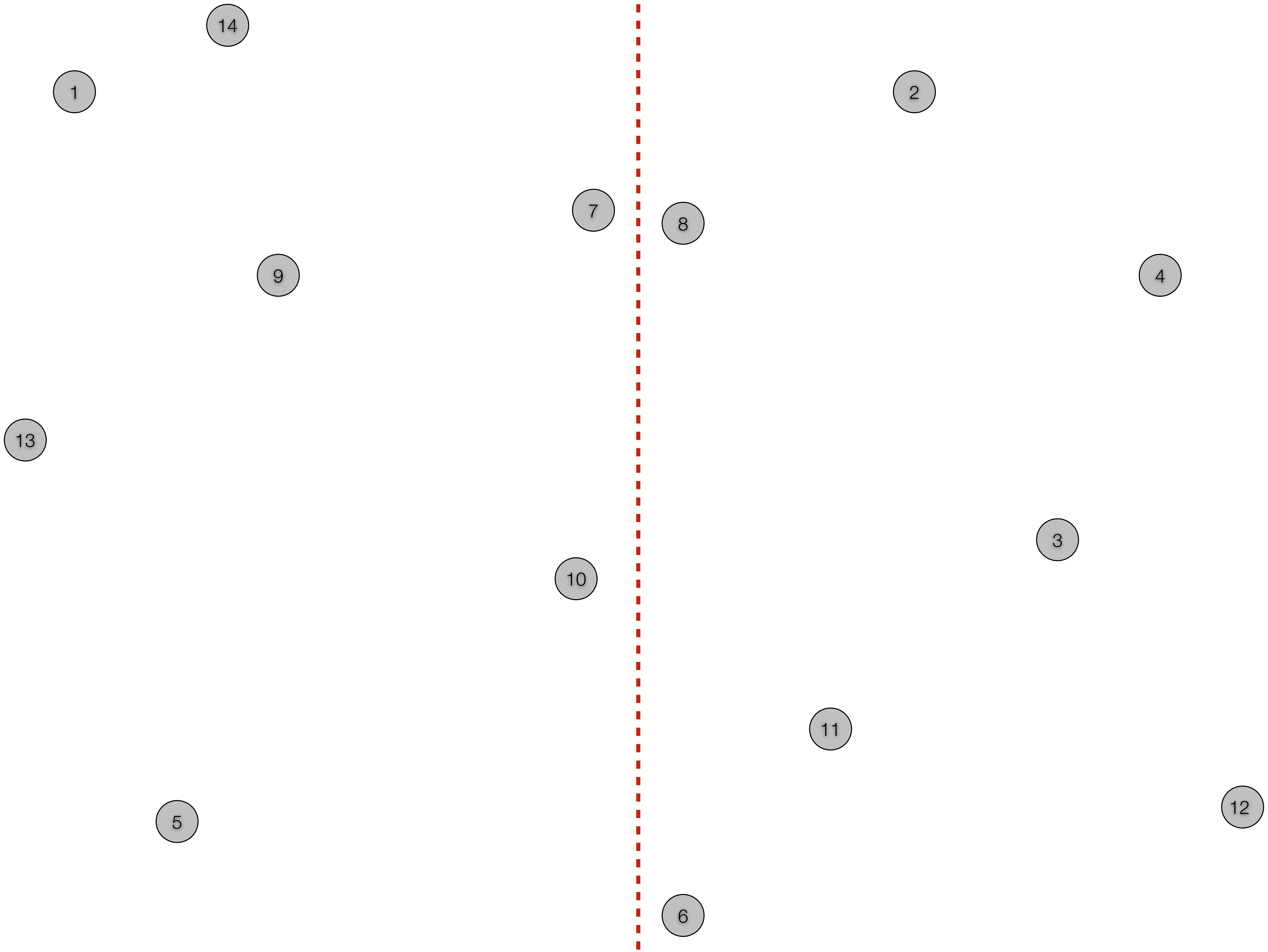
Points sorted in X: 13 1 5 14 9 10 7 6 8 11 2 3 4 12

Points sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



sorted in X: 13 1 5 14 9 10 7 9 8 11 2 3 4 12

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



ClosestPair(P)

Compute Sorted-in-X list SX

Compute Sorted-in-Y list SY

Closest(P,SX,SY)

Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q

delta,r,j = MIN(Closest(Left, LX, LY) Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q

delta,r,j = MIN(Closest(Left, LX, LY) Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

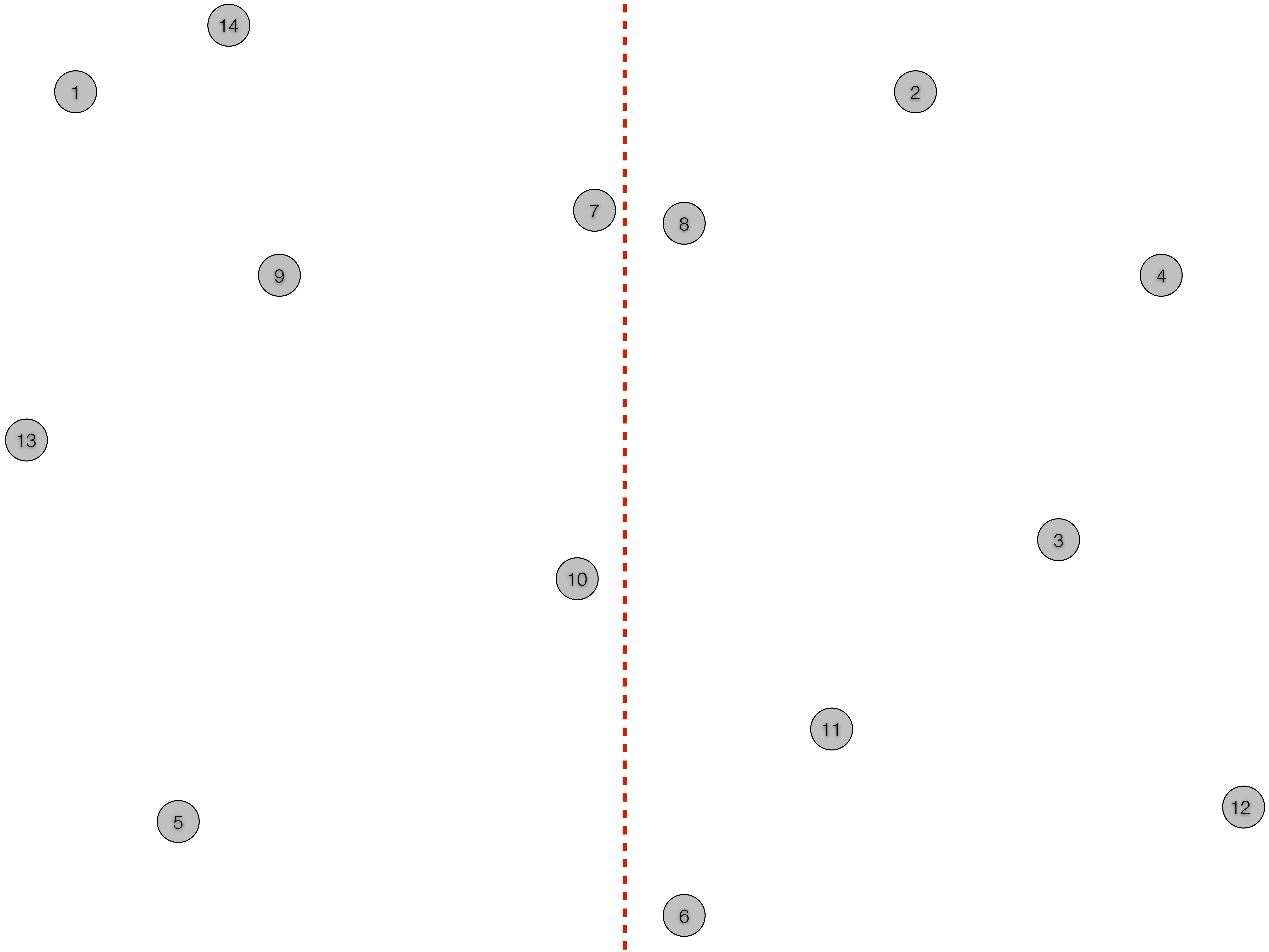
Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Can be reduced to 7!

sorted in X: 13 1 5 14 9 10 7 9 8 11 2 3 4 12

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q. Scan to get LY, RY.

delta,r,j = MIN(Closest(Left, LX, LY)   Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q. Scan to get LY, RY.

delta,r,j = MIN(Closest(Left, LX, LY)   Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

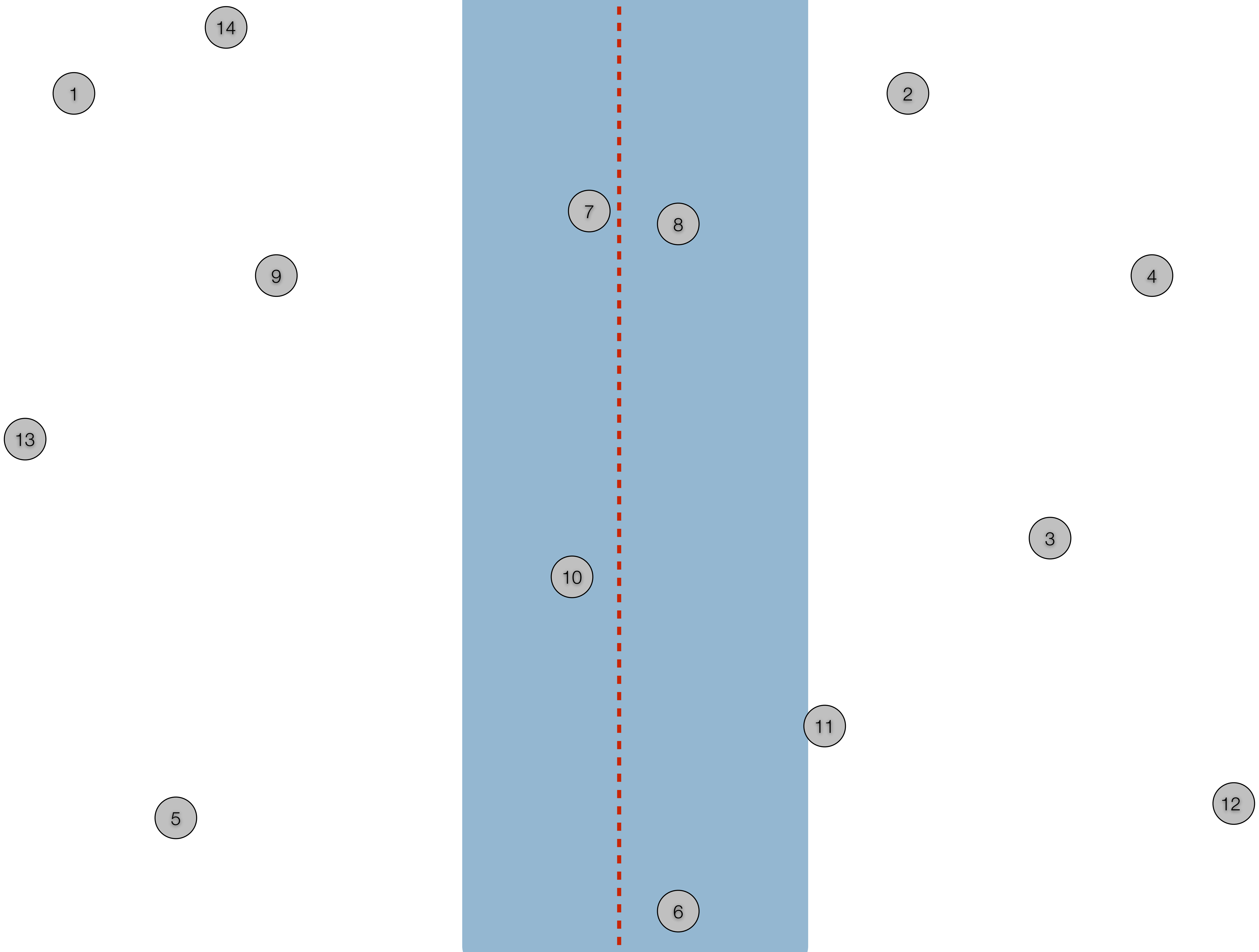
Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Can be reduced to 7!

sorted in X: 13 1 5 14 9 10 7 9 8 11 2 3 4 12

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q. Scan to get LY, RY.

delta,r,j = MIN(Closest(Left, LX, LY)   Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Closest(P,SX,SY)

Let q be the middle-element of SX

Divide P into Left, Right according to q. Scan to get LY, RY.

delta,r,j = MIN(Closest(Left, LX, LY)   Closest(Right, RX, RY))

Mohawk = { Scan SY, add pts that are delta from q.x }

For each point p in Mohawk (in order):

Compute distance between p and its next 15 neighbors
Update delta,r,j if any pair (x,y) is < delta

Return (delta,r,j)

Can be reduced to 7!

Running time for Closest pair algorithm

$$T(n) =$$

Running time for Closest pair algorithm

$$T(n) =$$

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

@author Robert Sedgewick
@author Kevin Wayne

```
public ClosestPair(Point2D[] points) {
    int N = points.length;
    if (N <= 1) return;

    // sort by x-coordinate (breaking ties by y-coordinate)
    Point2D[] pointsByX = new Point2D[N];
    for (int i = 0; i < N; i++)
        pointsByX[i] = points[i];
    Arrays.sort(pointsByX, Point2D.X_ORDER);

    // check for coincident points
    for (int i = 0; i < N-1; i++) {
        if (pointsByX[i].equals(pointsByX[i+1])) {
            bestDistance = 0.0;
            best1 = pointsByX[i];
            best2 = pointsByX[i+1];
            return;
        }
    }

    // sort by y-coordinate (but not yet sorted)
    Point2D[] pointsByY = new Point2D[N];
    for (int i = 0; i < N; i++)
        pointsByY[i] = pointsByX[i];

    // auxiliary array
    Point2D[] aux = new Point2D[N];

    closest(pointsByX, pointsByY, aux, 0, N-1);
}
```

```
// find closest pair of points in pointsByX[lo..hi]
// precondition:  pointsByX[lo..hi] and pointsByY[lo..hi] are the same sequence of points, sorted by x,y-coord
private double closest(Point2D[] pointsByX, Point2D[] pointsByY, Point2D[] aux, int lo, int hi) {
    if (hi <= lo) return Double.POSITIVE_INFINITY;

    int mid = lo + (hi - lo) / 2;
    Point2D median = pointsByX[mid];

    // compute closest pair with both endpoints in left subarray or both in right subarray
    double delta1 = closest(pointsByX, pointsByY, aux, lo, mid);
    double delta2 = closest(pointsByX, pointsByY, aux, mid+1, hi);
    double delta = Math.min(delta1, delta2);

    // merge back so that pointsByY[lo..hi] are sorted by y-coordinate
    merge(pointsByY, aux, lo, mid, hi);

    // aux[0..M-1] = sequence of points closer than delta, sorted by y-coordinate
    int M = 0;
    for (int i = lo; i <= hi; i++) {
        if (Math.abs(pointsByY[i].x() - median.x()) < delta)
            aux[M++] = pointsByY[i];
    }

    // compare each point to its neighbors with y-coordinate closer than delta
    for (int i = 0; i < M; i++) {
        // a geometric packing argument shows that this loop iterates at most 7 times
        for (int j = i+1; (j < M) && (aux[j].y() - aux[i].y() < delta); j++) {
            double distance = aux[i].distanceTo(aux[j]);
            if (distance < delta) {
                delta = distance;
                if (distance < bestDistance) {
                    bestDistance = delta;
                    best1 = aux[i];
                    best2 = aux[j];
                    // StdOut.println("better distance = " + delta + " from " + best1 + " to " + best2);
                }
            }
        }
    }
    return delta;
}
```

http://algs4.cs.princeton.edu/99hull/ClosestPair.java.html

Matrix

multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bigstar \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \star \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5+14 & 6+16 \\ 15+28 & 18+32 \end{bmatrix}$$

$$= \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & & & \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & & & \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & & & \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & & & \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix}$$

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & & & \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & & & \\ c_{n,1} & c_{n,2} & \cdots & c_{n,n} \end{bmatrix}$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$$T(n) = 8T(n/2) + \Theta(n^2)$$

$$\Theta(n^3)$$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

[Strassen]

$$P_1 = A(F - H)$$

$$P_2 = (A + B)H$$

$$P_3 = (C + D)E$$

$$P_4 = D(G - E)$$

$$P_5 = (A + D)(E + H)$$

$$P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

$R = P_5 + P_4 - P_2 + P_6$ $\quad$ $S = P_1 + P_2$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$T = P_3 + P_4$ $\quad$ $U = P_5 + P_1 - P_3 - P_7$

[strassen]

$$P_1 = A(F - H)$$

$$P_2 = (A + B)H$$

$$P_3 = (C + D)E$$

$$P_4 = D(G - E)$$

$$P_5 = (A + D)(E + H)$$

$$P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

$$= \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$$R = P_5 + P_4 - P_2 + P_6 \qquad S = P_1 + P_2$$

$$T = P_3 + P_4 \qquad U = P_5 + P_1 - P_3 - P_7$$

[strassen]

$$P_1 = A(F - H)$$

$$P_2 = (A + B)H$$

$$P_3 = (C + D)E$$

$$P_4 = D(G - E)$$

$$P_5 = (A + D)(E + H)$$

$$P_6 = (B - D)(G + H)$$

$$P_7 = (A - C)(E + F)$$

$$M(n) = 7M(n/2) + 18n^2$$

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

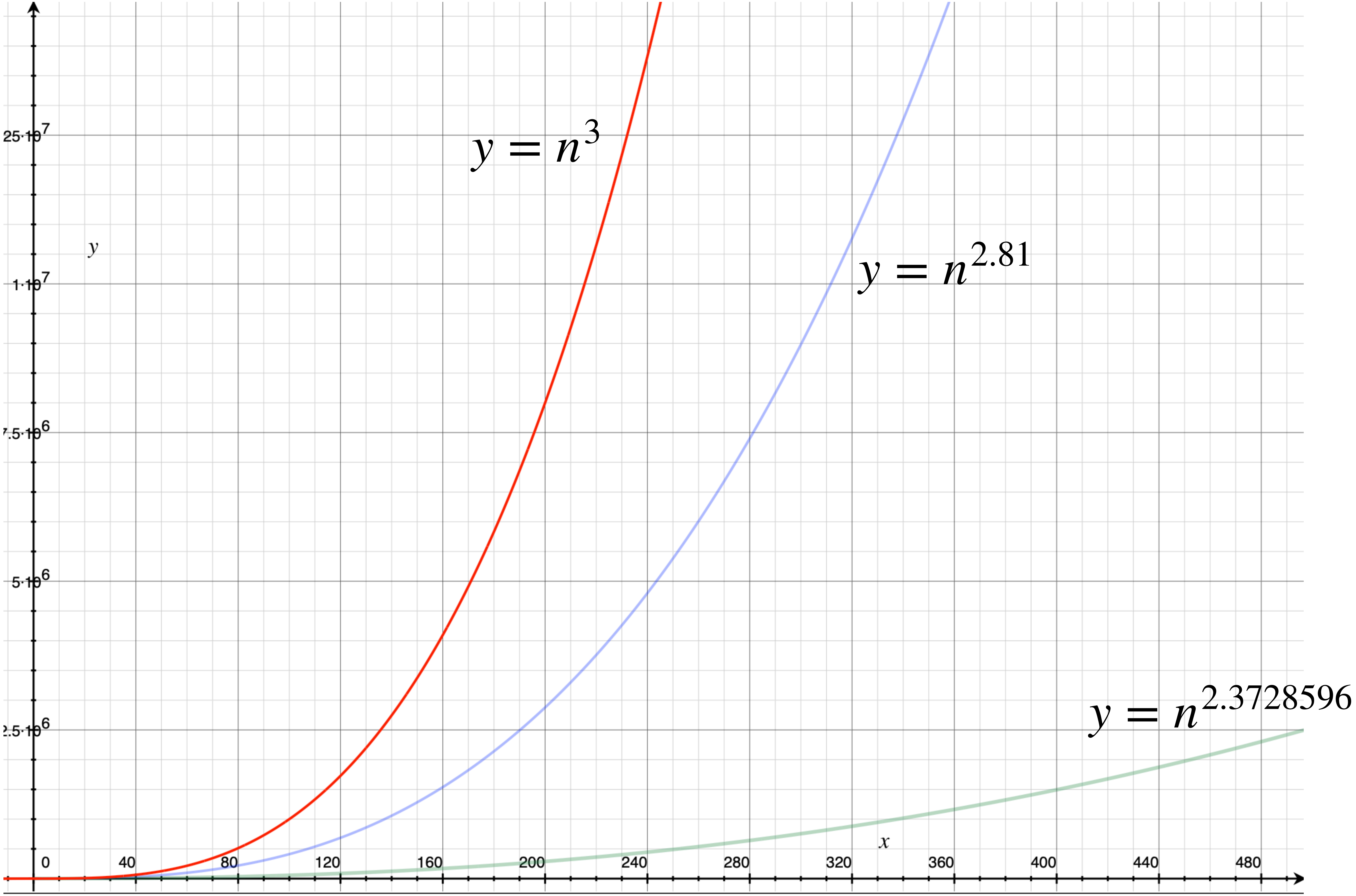
taking this idea further

3x3 matricies [Laderman'75] in 23 multe

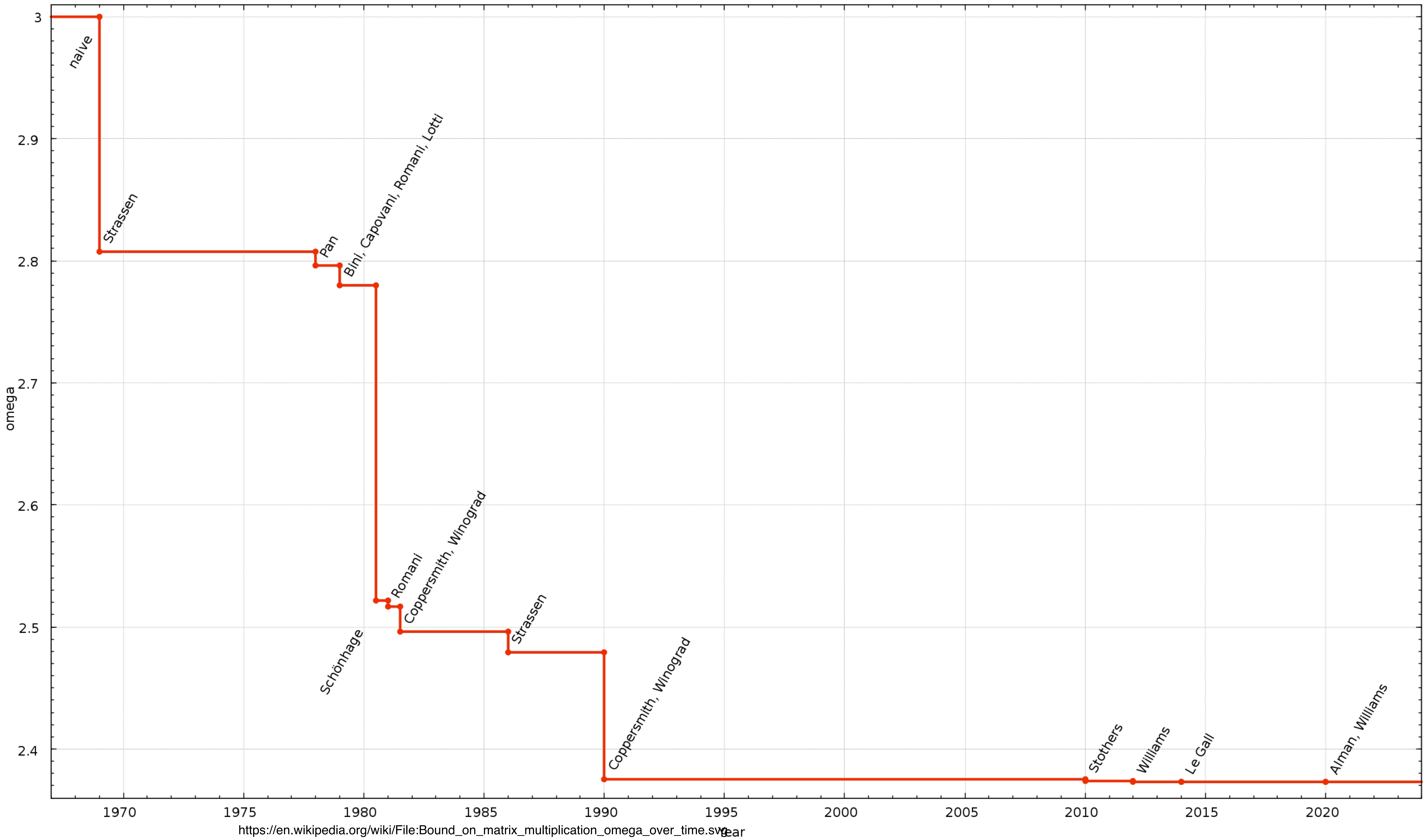
1978 victor pan method

70x70 matrix using 143640 mults

what is the recurrence:

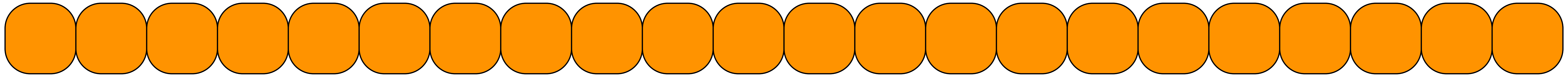


$y = n^3$
$y = n^{2.81}$
$y = n^{2.3728596}$
y
x
25·10^7
1·10^7
7.5·10^6
5·10^6
2.5·10^6
0
40
80
120
160
200
240
280
320
360
400
440
480

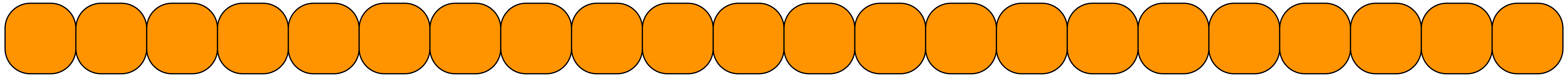


https://en.wikipedia.org/wiki/File:Bound_on_matrix_multiplication_omega_over_time.svg

MEDIAN



problem: given a list of **n** elements, find the element of rank **n**/2. (half are larger, half are smaller)

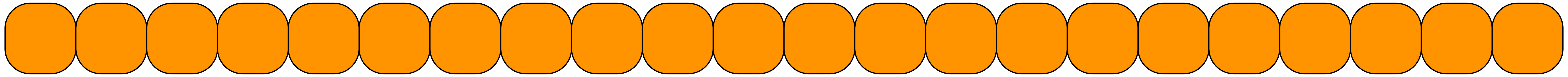


problem: given a list of **n** elements, find the element of rank n/2. (half are larger, half are smaller)

can generalize to i

first solution: sort and pluck.

$$O(n \log n)$$

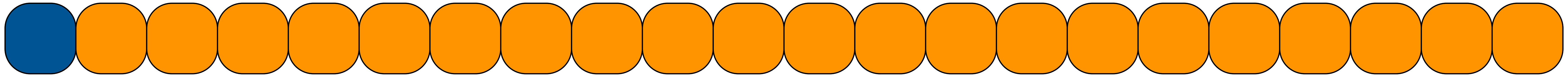


problem: given a list of **n** elements, find the element of rank **i**.

key insight:

we do not have to “fully” sort.

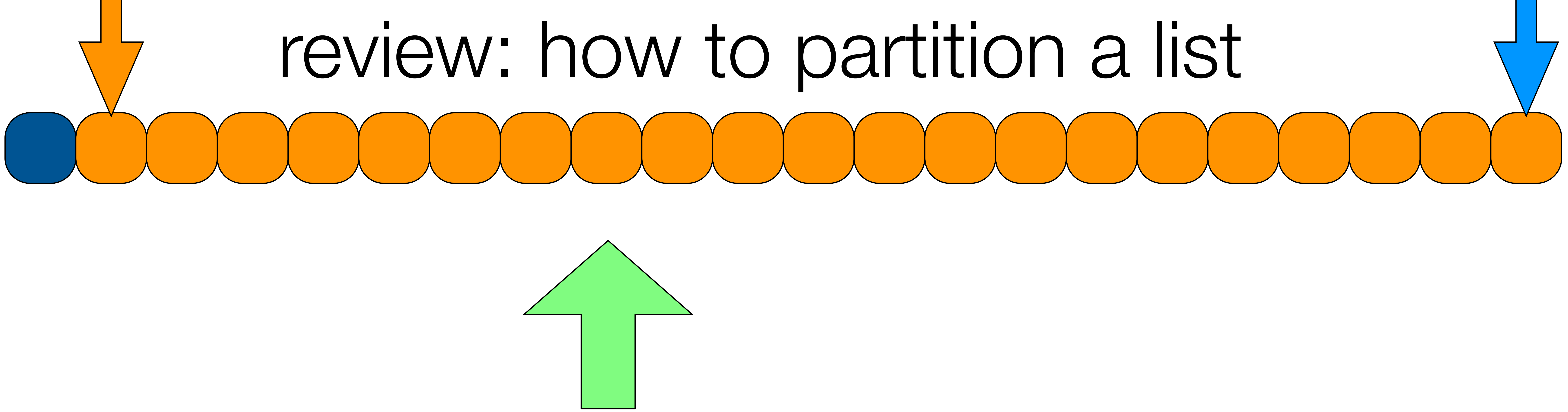
semi sort can suffice.



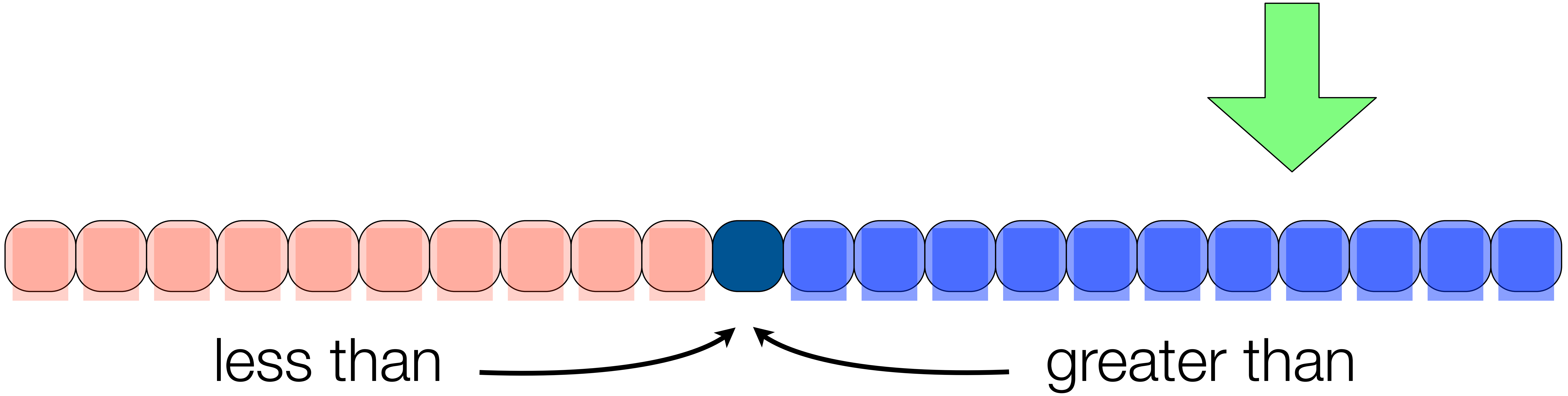
pick first element
partition list about this one
see where we stand

review: how to partition a list

review: how to partition a list

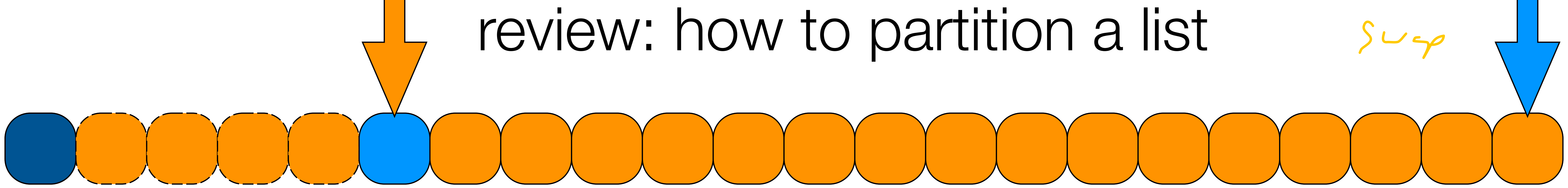


GOAL: start with THIS LIST and END with THAT LIST

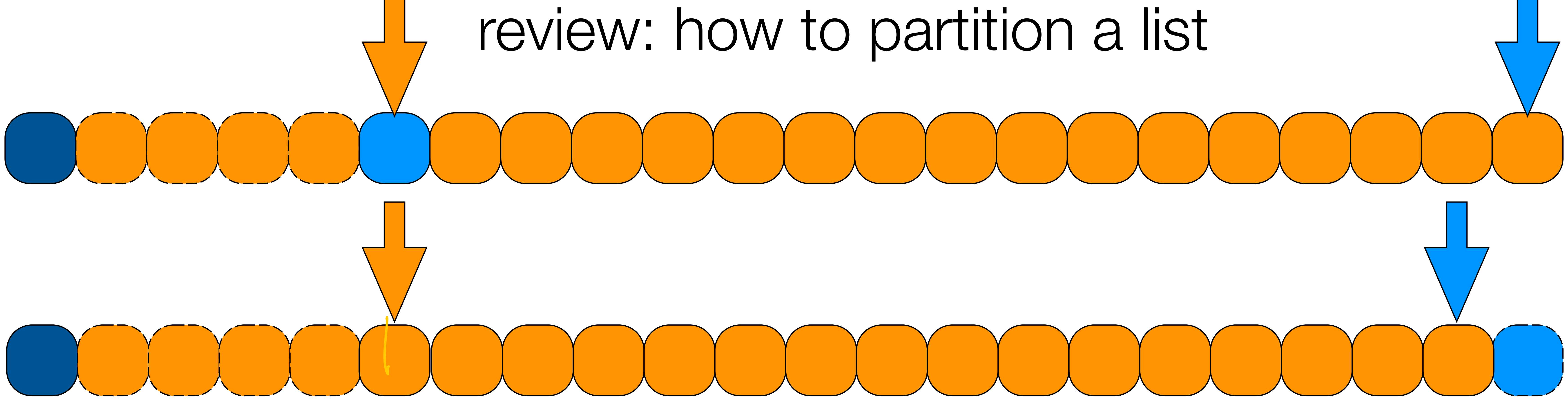


review: how to partition a list

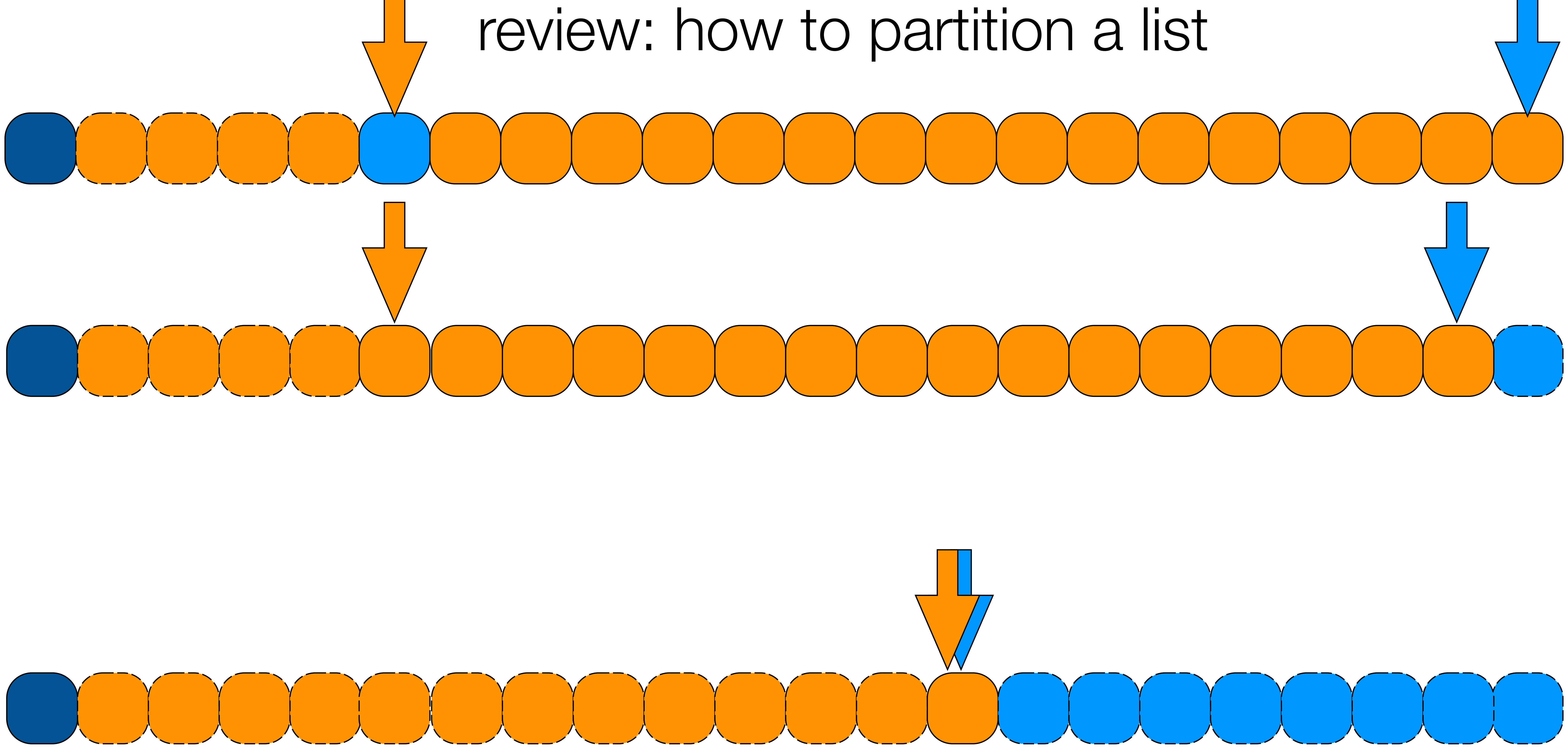
review: how to partition a list



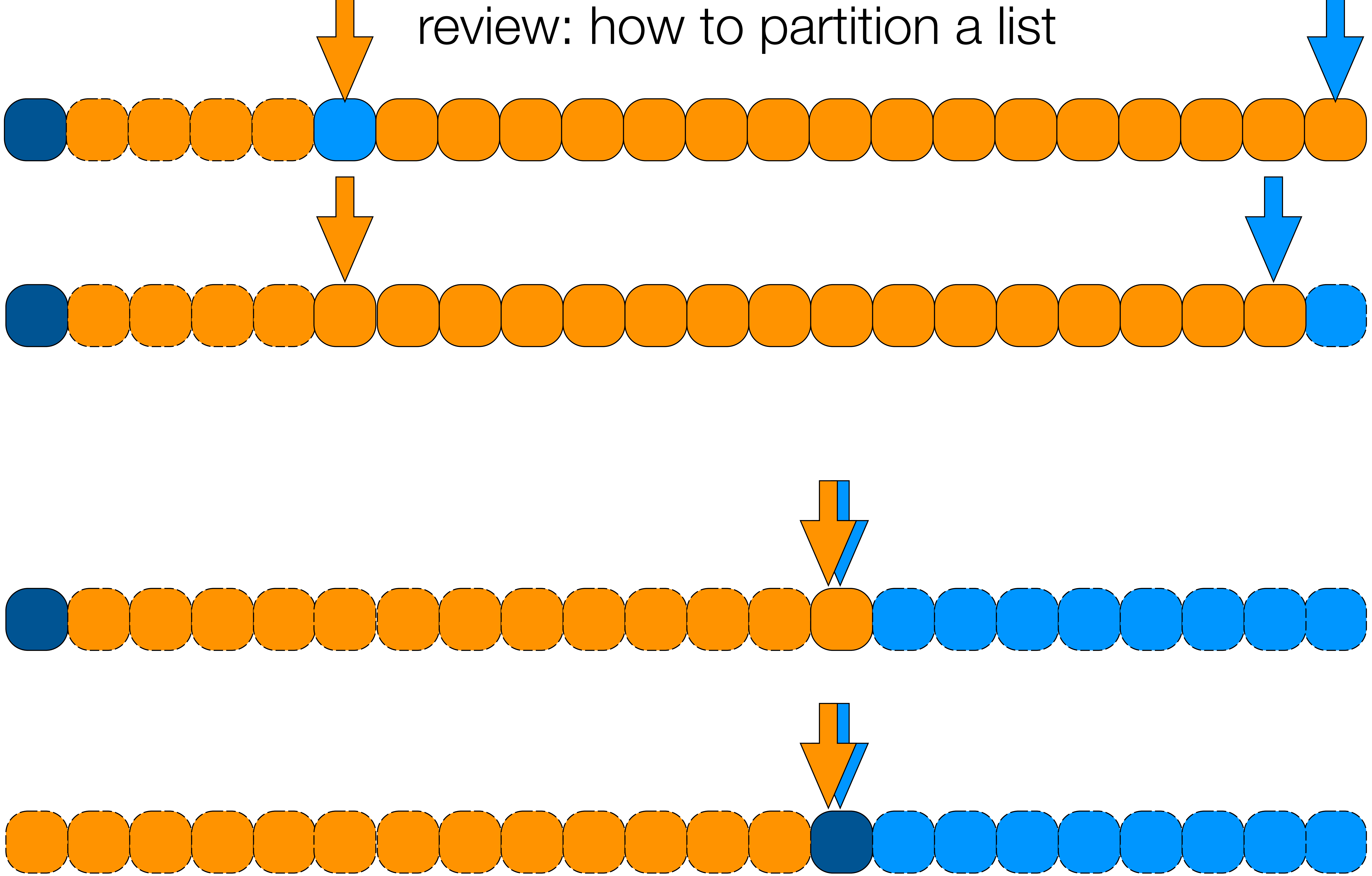
review: how to partition a list



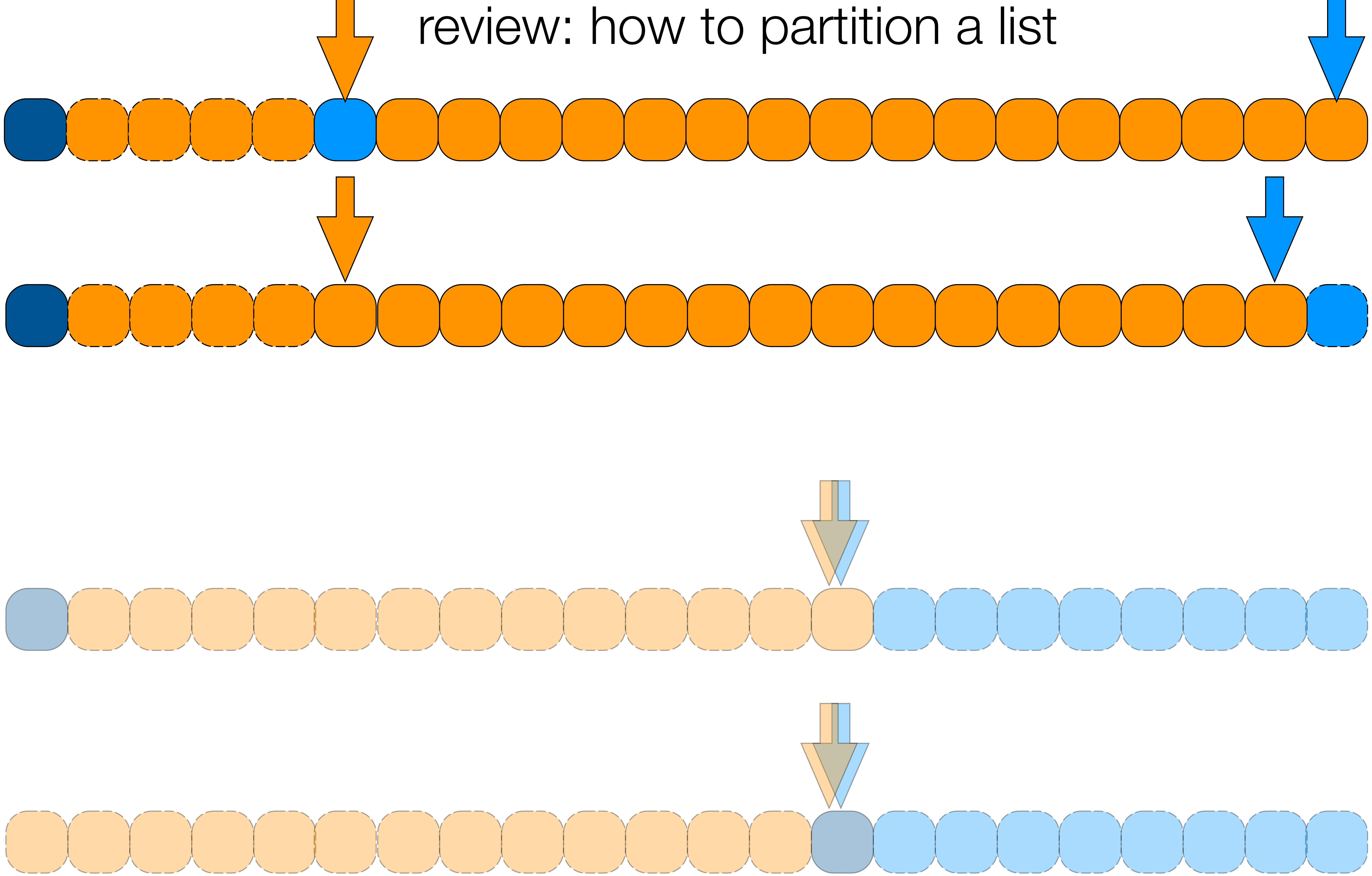
review: how to partition a list



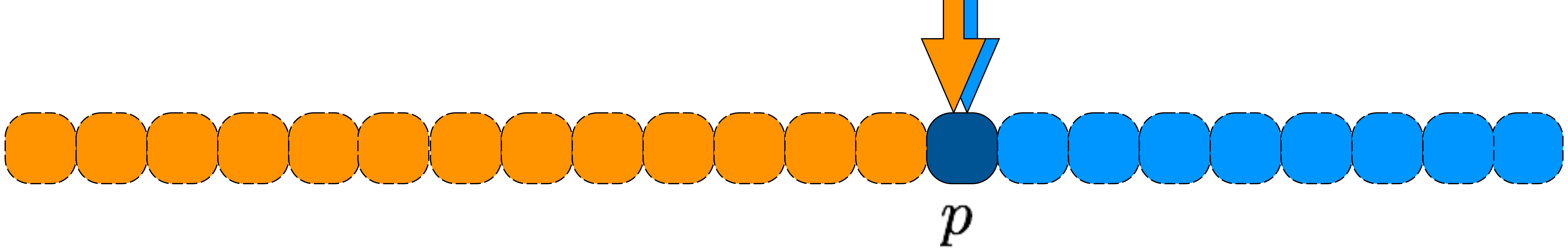
review: how to partition a list



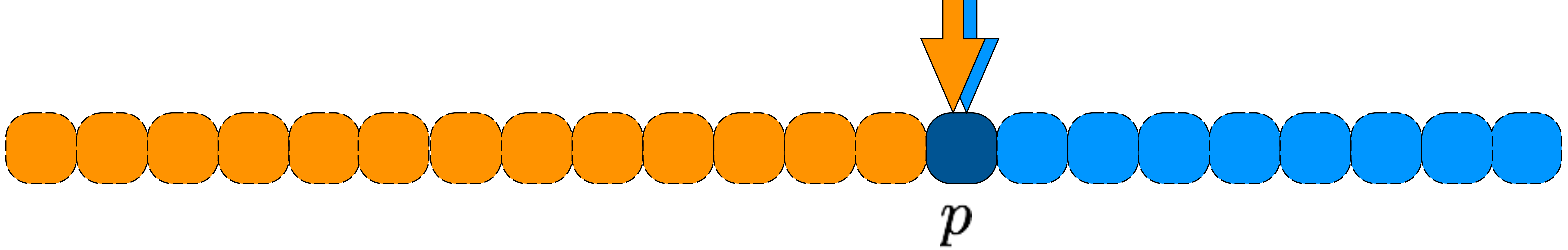
review: how to partition a list



partitioning a list about an element takes linear time.



select $(i, A[1, \ldots, n])$



select $(i, A[1, \ldots, n])$

handle base case of 1 element.

partition list about first element

if pivot p is position **i**, return pivot

else if pivot p is in position > **i** select $(i, A[1, \ldots, p-1])$

else select $((i-p-1), A[p+1, \ldots, n])$

select $(i, A[1, \ldots, n])$

Assume our partition always splits list into two eql parts

handle base case.

partition list about first element

if pivot is position **i**, return pivot

else if pivot is in position > i   select $(i, A[1, \ldots, p-1])$

else select $((i-p-1), A[p+1, \ldots, n])$

select $(i, A[1, \ldots, n])$

Assume our partition always splits list into two eql parts

handle base case.

partition list about first element

if pivot is position **i**, return pivot

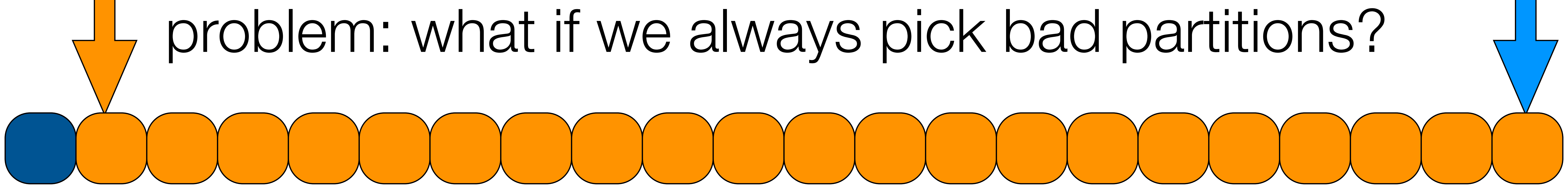
else if pivot is in position > i select $(i, A[1, \ldots, p-1])$

else select $((i-p-1), A[p+1, \ldots, n])$

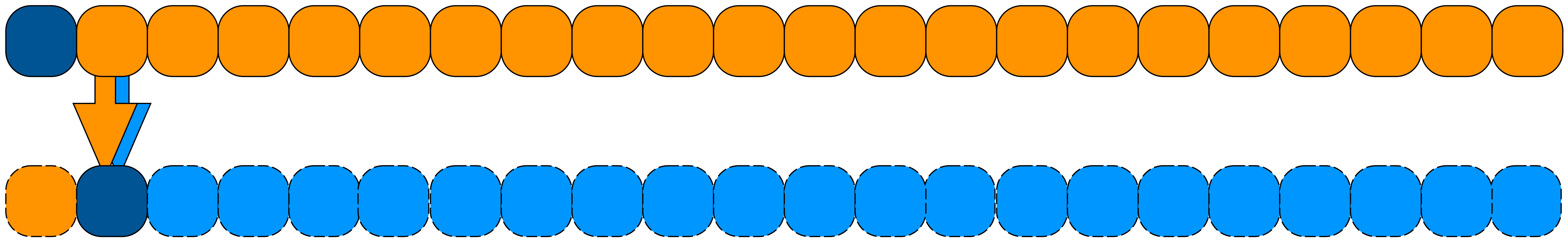
$$T(n) = T(n/2) + O(n)$$

$$\Theta(n)$$

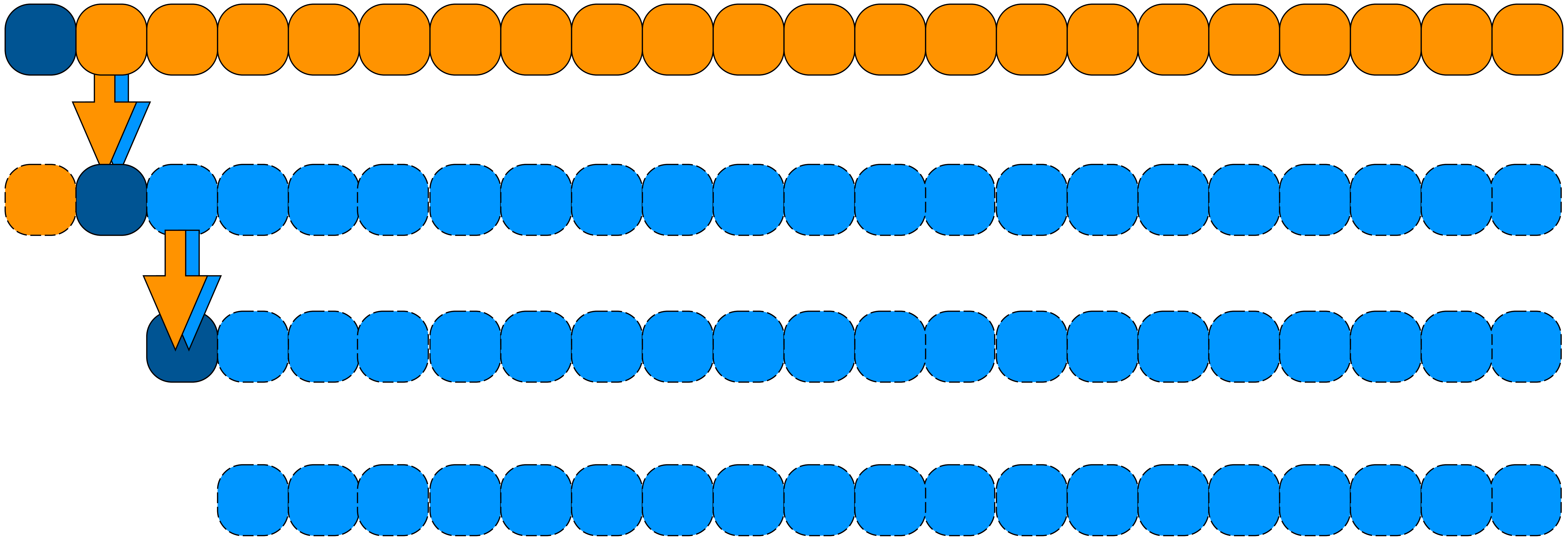
problem: what if we always pick bad partitions?

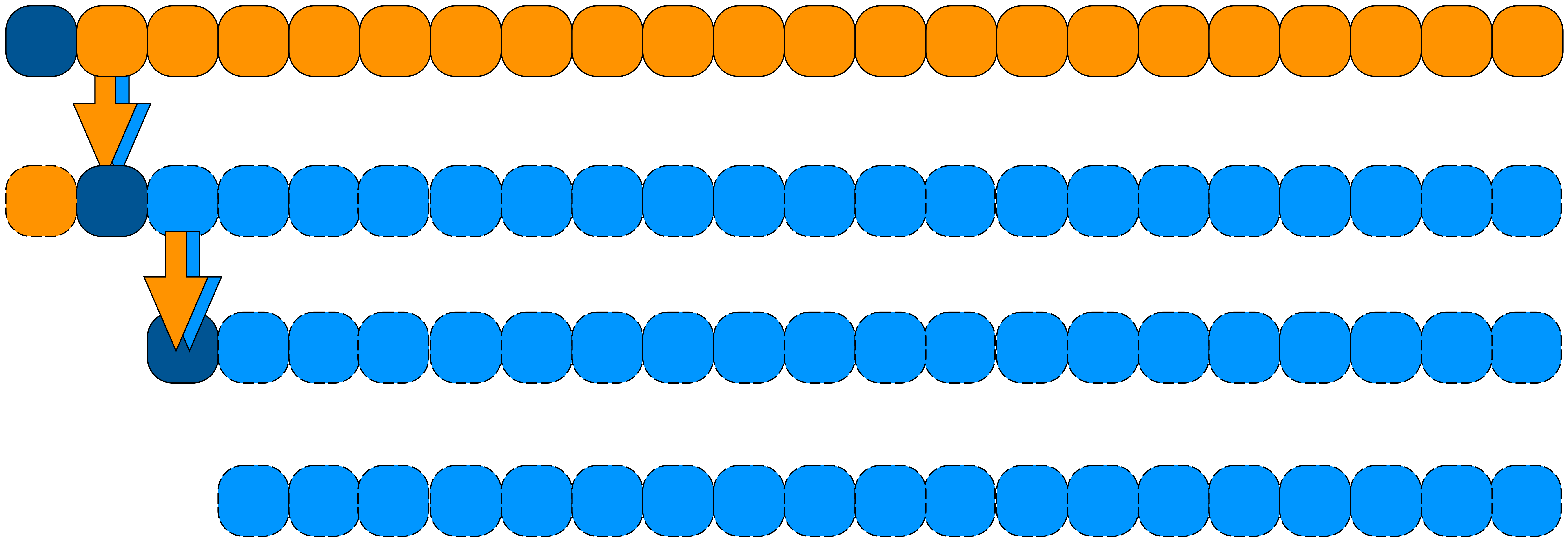


problem: what if we always pick bad partitions?



problem: what if we always pick bad partitions?





problem: what if we always pick bad partitions?

select $(i, A[1, \ldots, n])$

handle base case.

partition list about first element

if pivot is position **i**, return pivot

else if pivot is in position > i   select $(i, A[1, \ldots, p-1])$

else select $((i-p-1), A[p+1, \ldots, n])$

select $(i, A[1, \ldots, n])$

handle base case.
partition list about first element
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

$$T(n) = T(n-1) + O(n)$$

$$\Theta(n^2)$$

Needed:

a good partition element

partition $(A[1, \ldots, n])$

Needed:

a good partition element

partition $(A[1, \ldots, n])$

produce an element where
30% smaller, 30% larger

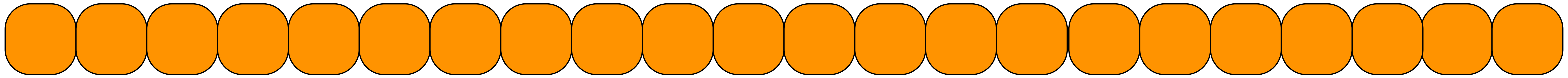
solution: bootstrap

image: gucci

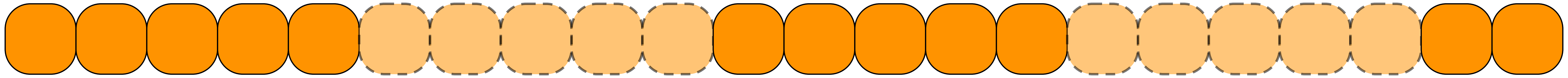
image: mark nason

image: d&g

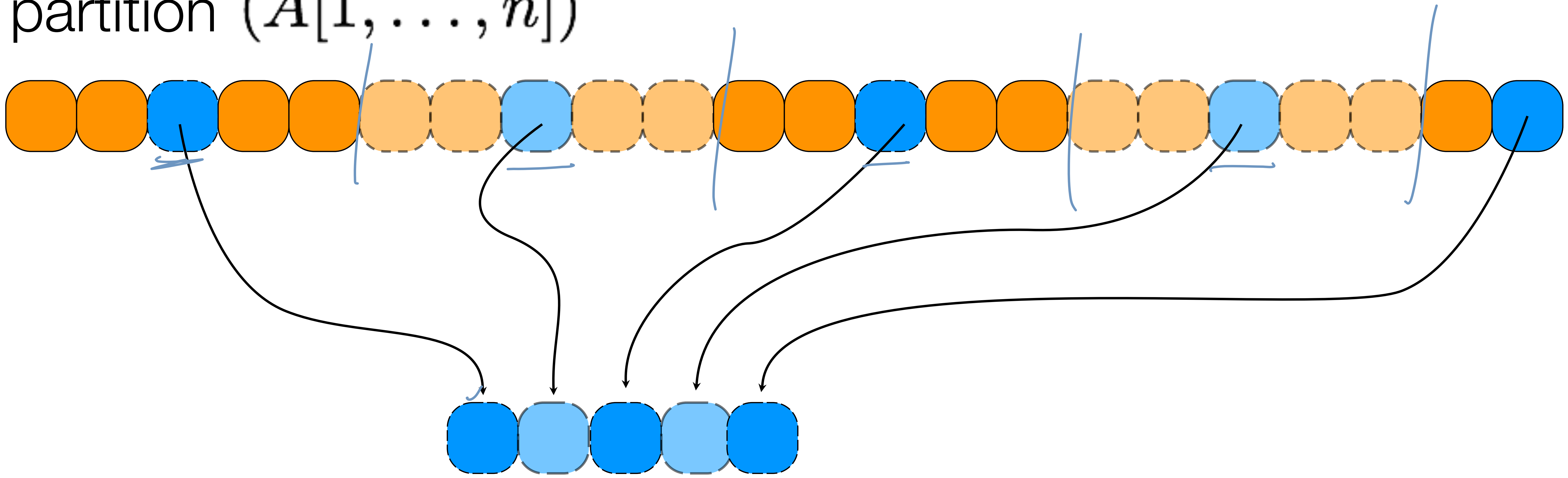
partition $(A[1, \ldots, n])$



partition $(A[1, \ldots, n])$



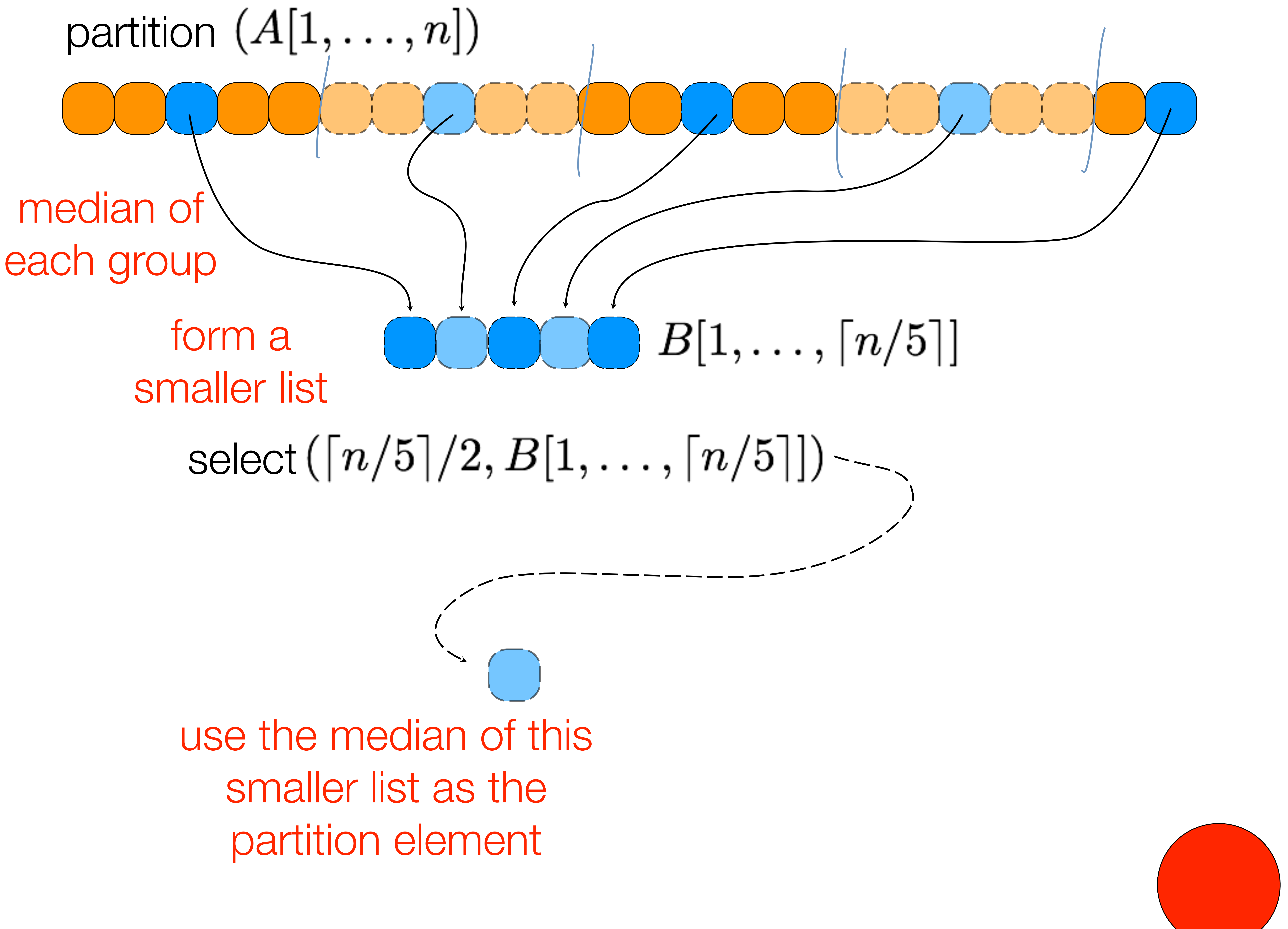
partition $(A[1,\ldots,n])$



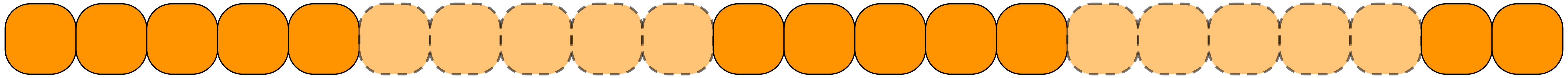
divide list into groups of 5 elements

find median of each small list using brute force

gather all medians


partition $(A[1, \ldots, n])$
median of
each group
form a
smaller list
$B[1, \ldots, \lceil n/5 \rceil]$
select $(\lceil n/5 \rceil / 2, B[1, \ldots, \lceil n/5 \rceil])$
use the median of this
smaller list as the
partition element

partition $(A[1, \ldots, n])$



divide list into groups of 5 elements

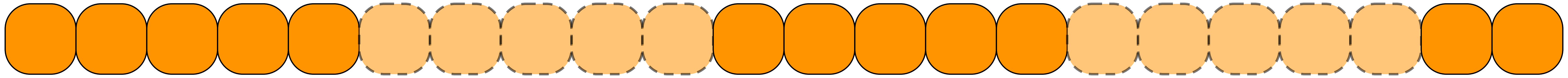
find median of each small list using brute force

gather all medians

call select(...) on this sublist to find median

return the result

partition $(A[1,...,n])$



divide list into groups of 5 elements

find median of each small list

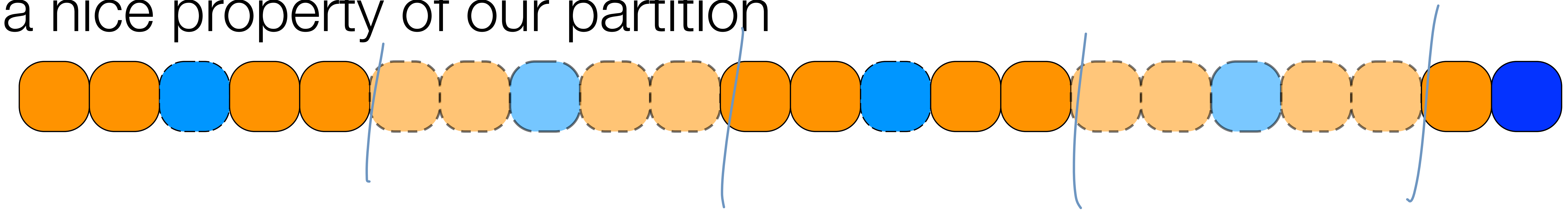
gather all medians

call select(...) on this sublist to find median

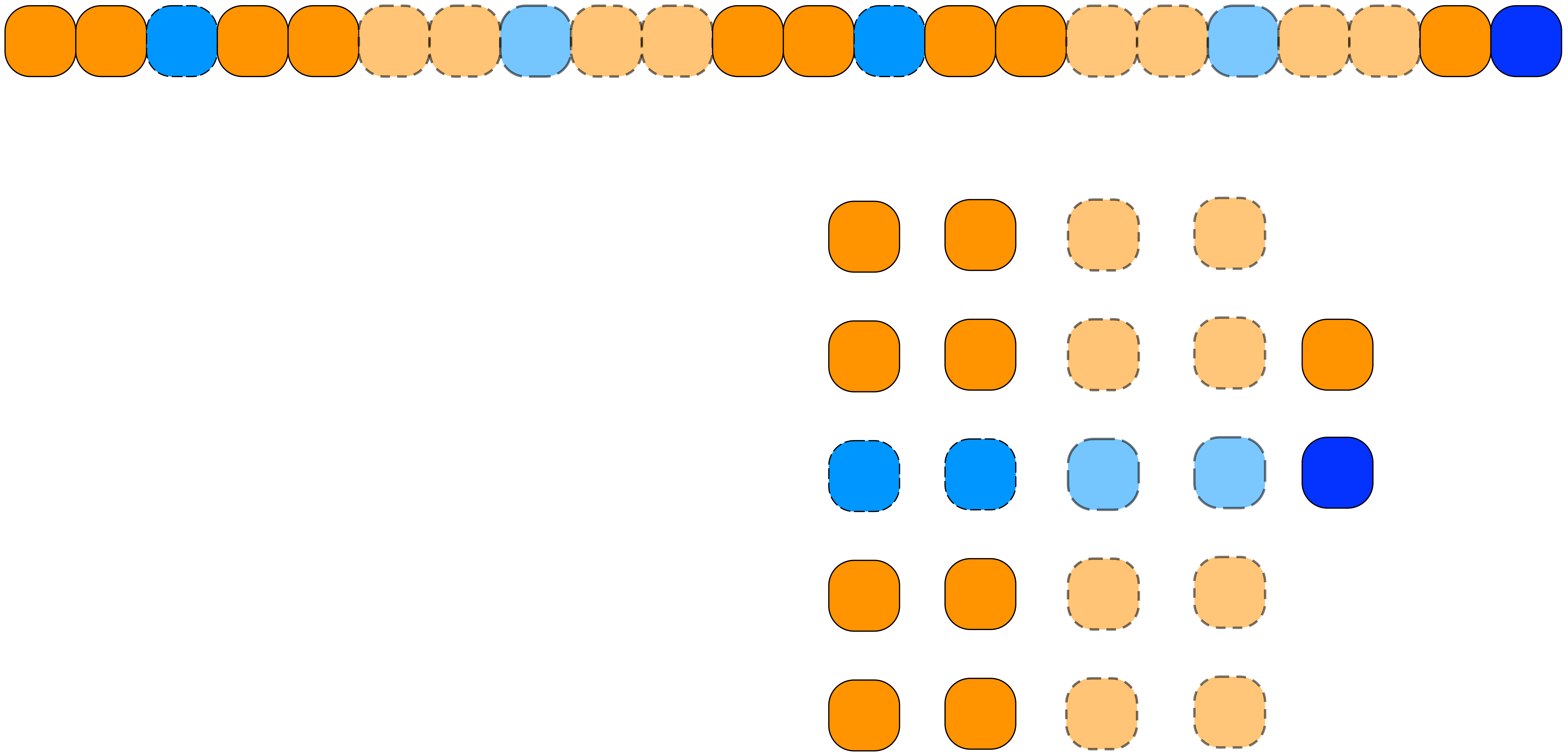
return the result

$$P(n) = S(\lceil n/5 \rceil) + O(n)$$

a nice property of our partition

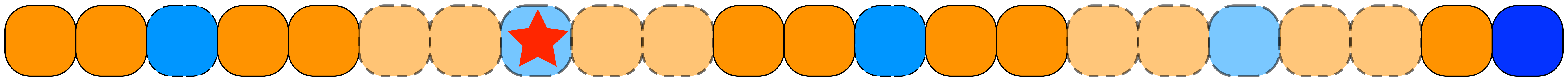


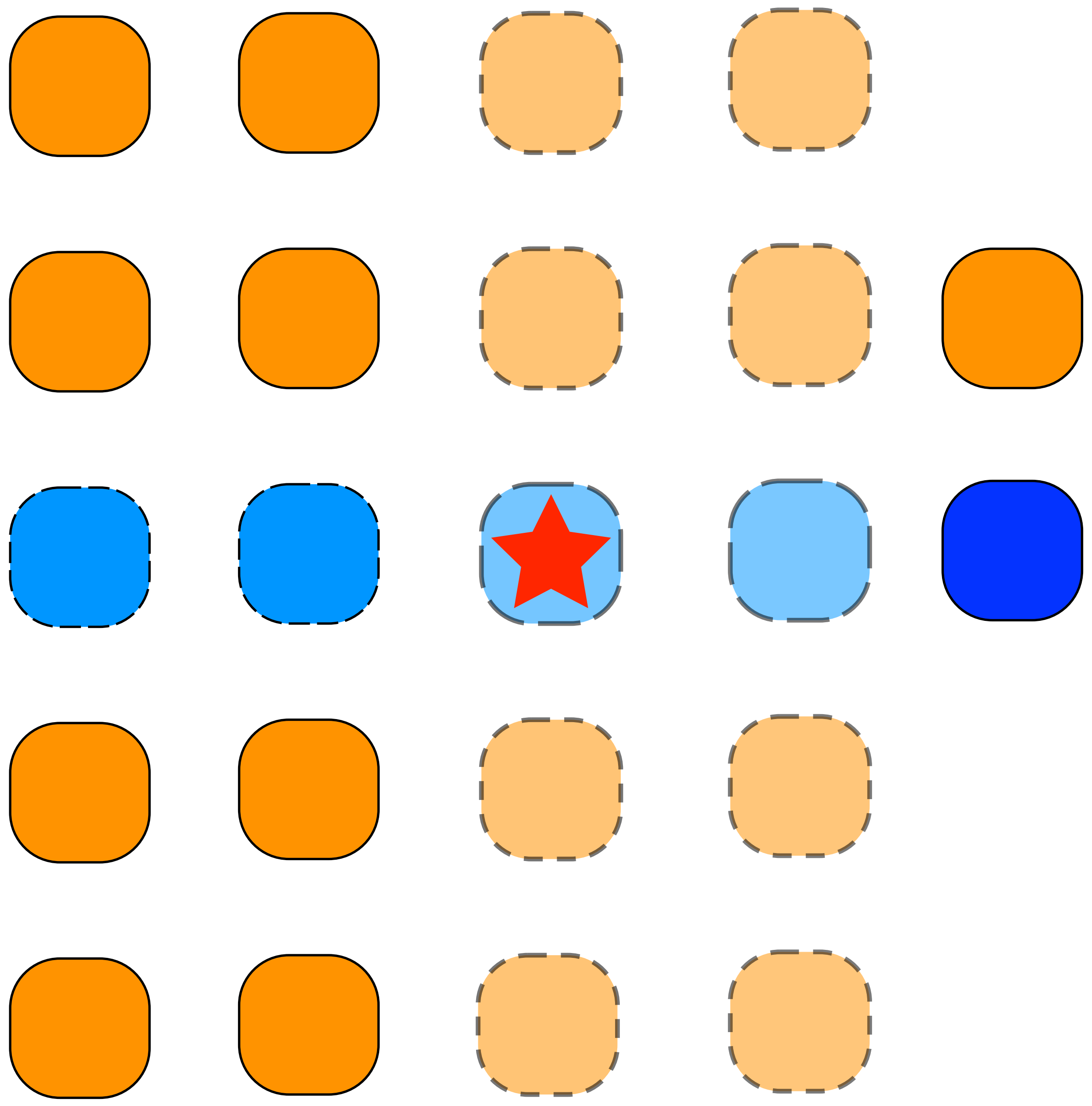
a nice property of our partition



Imagine rearranging the elements by sorting each column and then also sorting the medians.

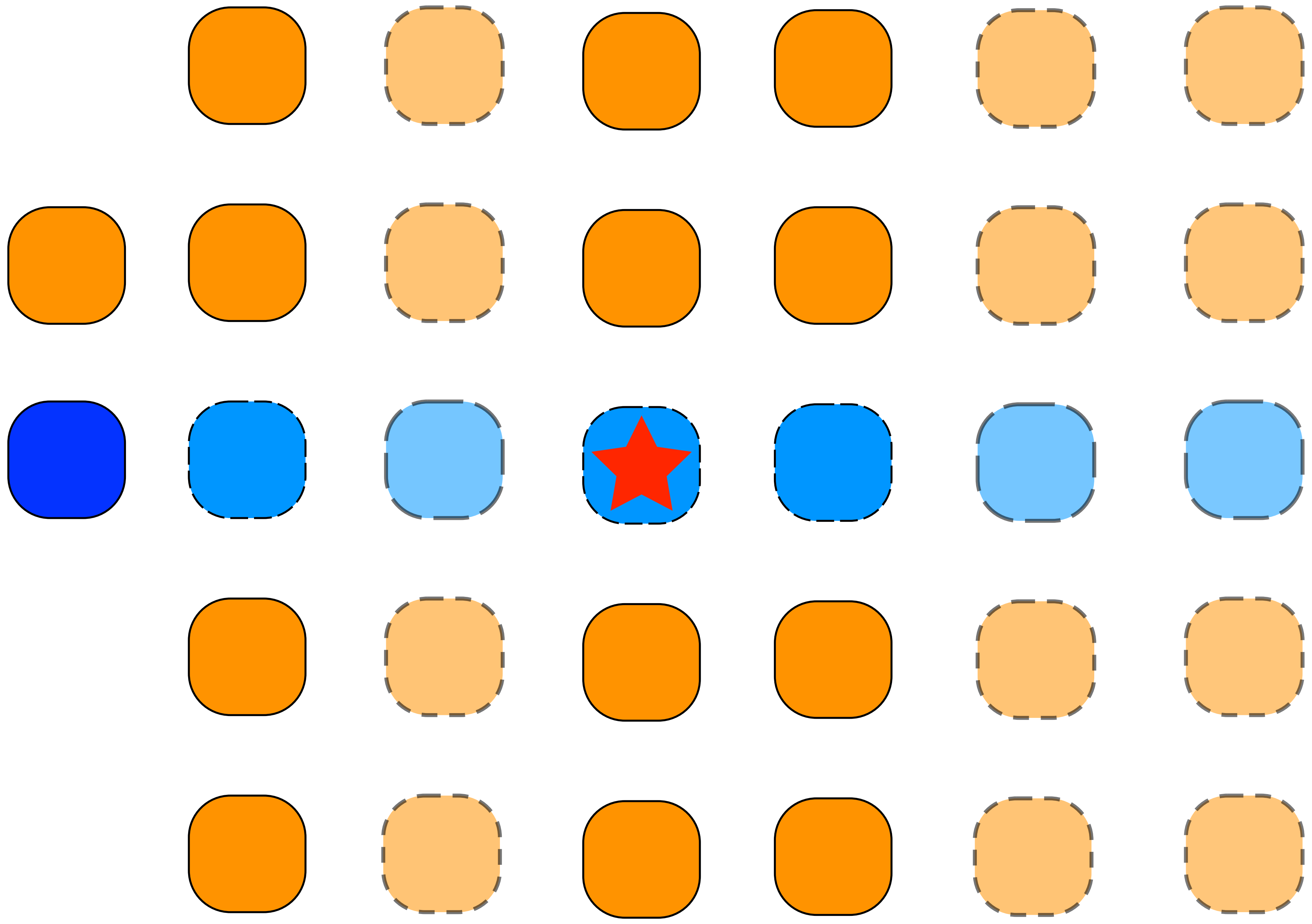
a nice property of our partition





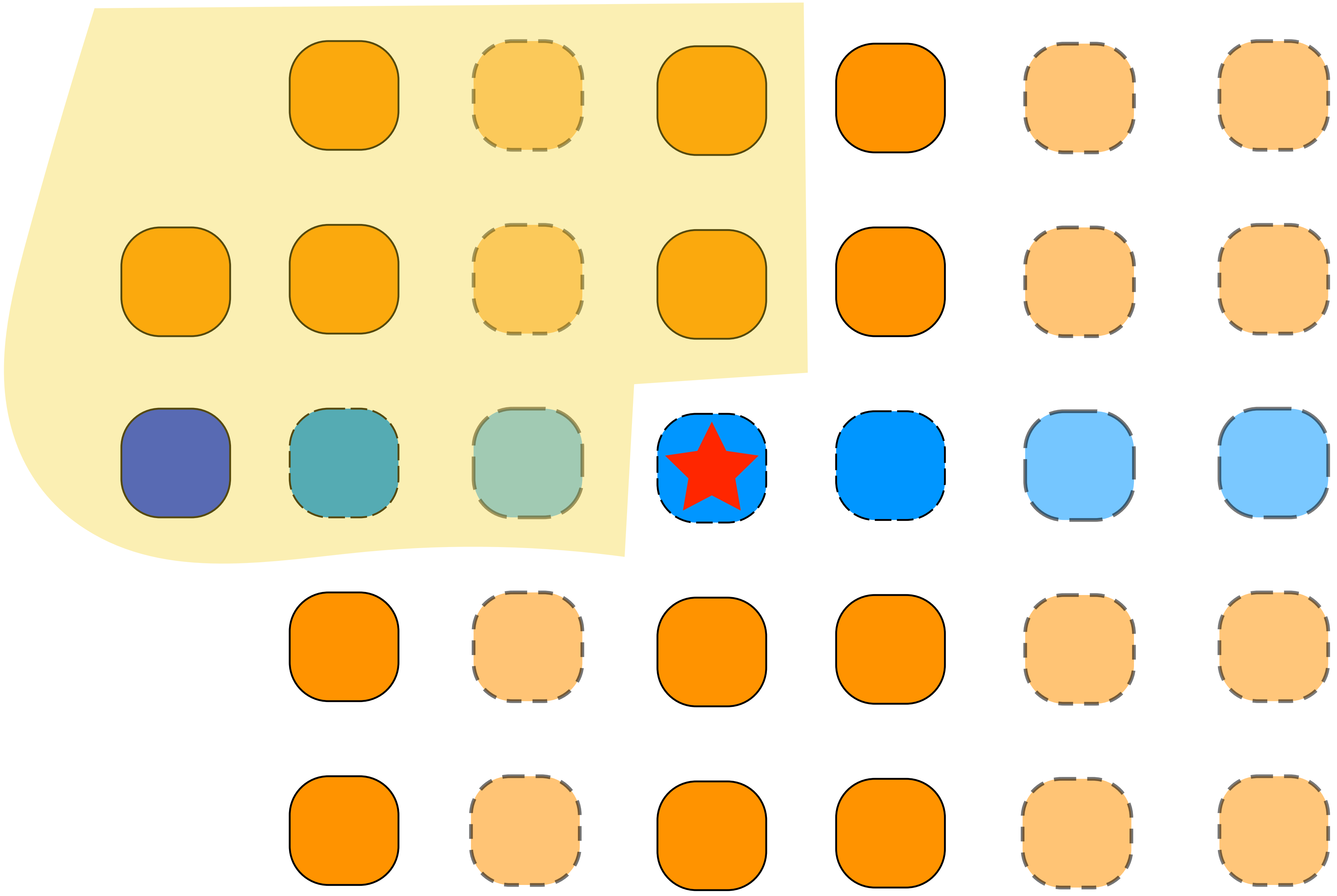
Imagine rearranging the elements by sorting each column and then also sorting the medians.

SWITCH TO A BIGGER EXAMPLE



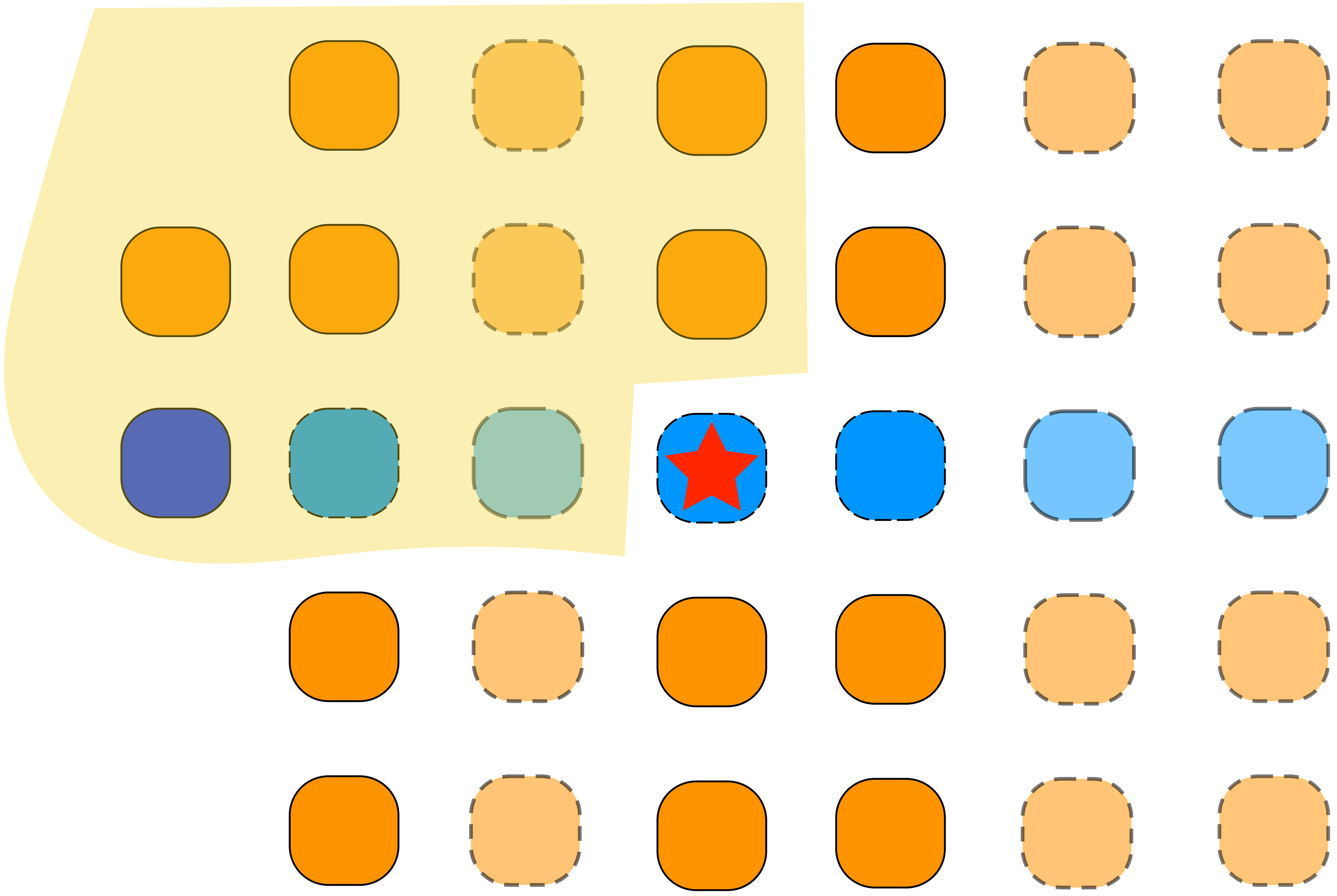
SWITCH TO A BIGGER EXAMPLE

These yellow elements are all smaller than the median. How many are there?



SWITCH TO A BIGGER EXAMPLE

These yellow elements are all smaller than the median. How many are there?



$$3\left(\left\lceil\frac{1}{2}\lceil n/5\rceil\right\rceil - 2\right)$$

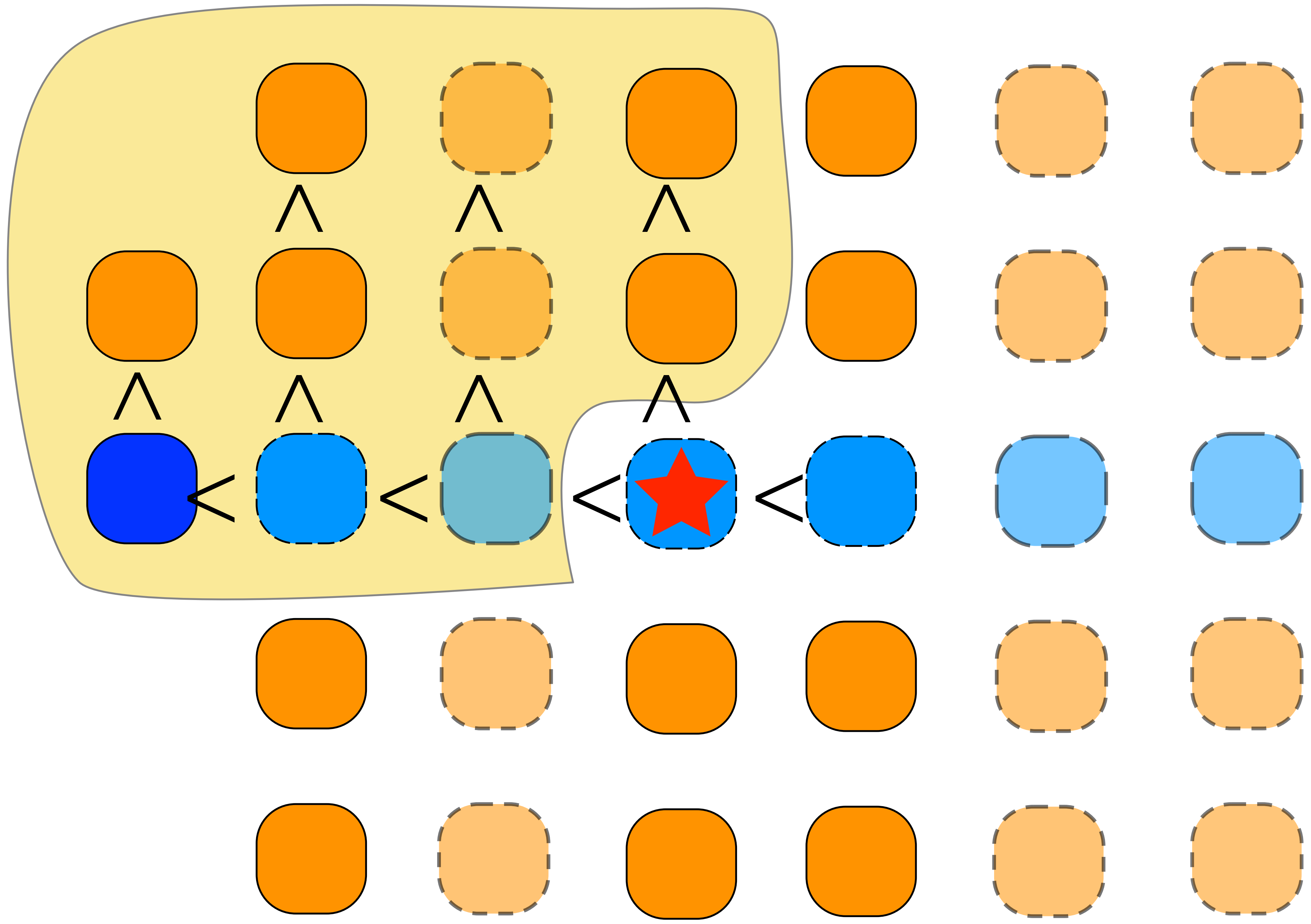
$$\geq \frac{3n}{10} - 6$$

There are $\lceil n/5 \rceil / 2$ columns. Ignoring the first and last, each column has 3 elements in it that are smaller than the median.

a nice property of our partition

$$3\left(\left\lceil \frac{1}{2}\lceil n/5 \rceil \right\rceil - 2\right)$$

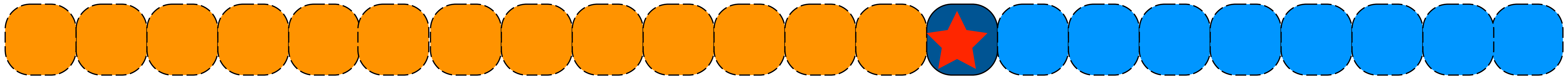
$$\geq \frac{3n}{10} - 6$$

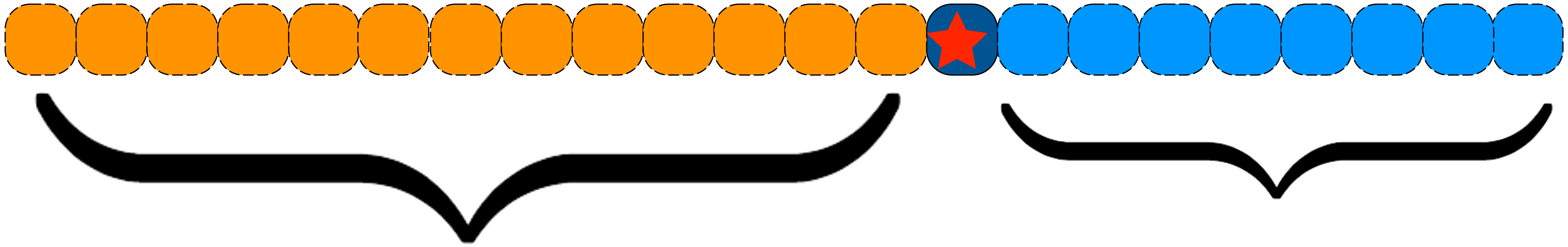


this implies there are
at most $\frac{7n}{10} + 6$ numbers

larger than ★
/smaller

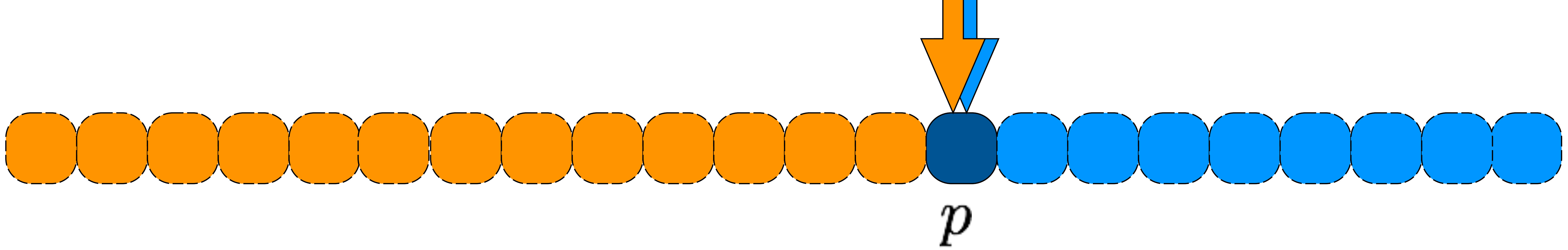
a nice property of our partition





$$\leq \frac{7n}{10} + 6 \qquad\qquad \leq \frac{7n}{10} + 6$$

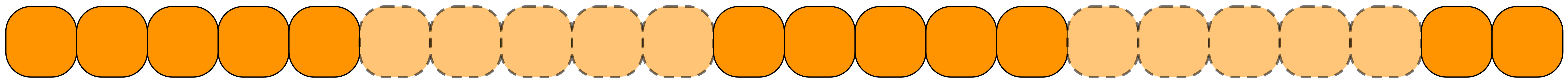
The median-of-medians is guaranteed to have a **linear fraction** of the input that is smaller and larger than it.



select $(i, A[1, \ldots, n])$

handle base case for small list
else pivot = FindPartitionValue(A,n)
partition list about pivot
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

FindPartition $(A[1,\ldots,n])$



divide list into groups of 5 elements

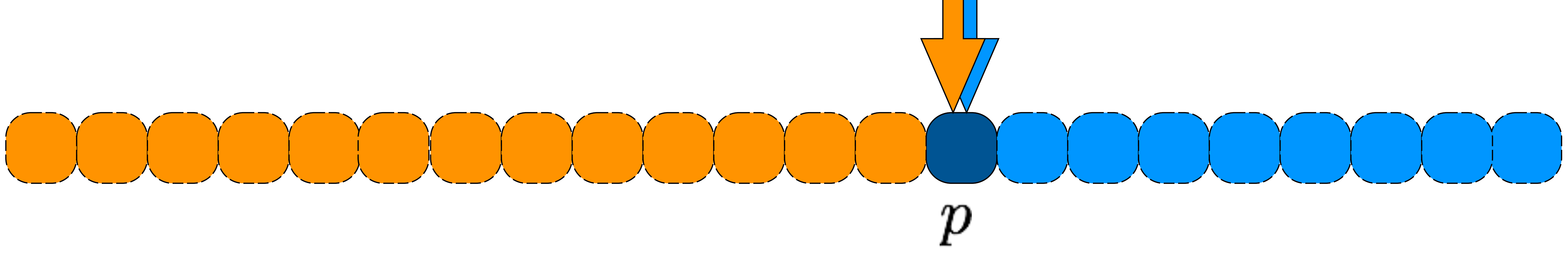
find median of each small list

gather all medians

call select(...) on this sublist to find median

return the result

$$P(n) = S(\lceil n/5 \rceil) + O(n)$$



select $(i, A[1, \ldots, n])$

handle base case for small list
else pivot = FindPartitionValue(A,n)
partition list about pivot
if pivot is position **i**, return pivot
else if pivot is in position > **i** select $(i, A[1, \ldots, p-1])$
else select $((i-p-1), A[p+1, \ldots, n])$

$$S(n) = S(\lceil n/5 \rceil) + \Theta(n) + S(\lceil 7n/10 + 6 \rceil)$$

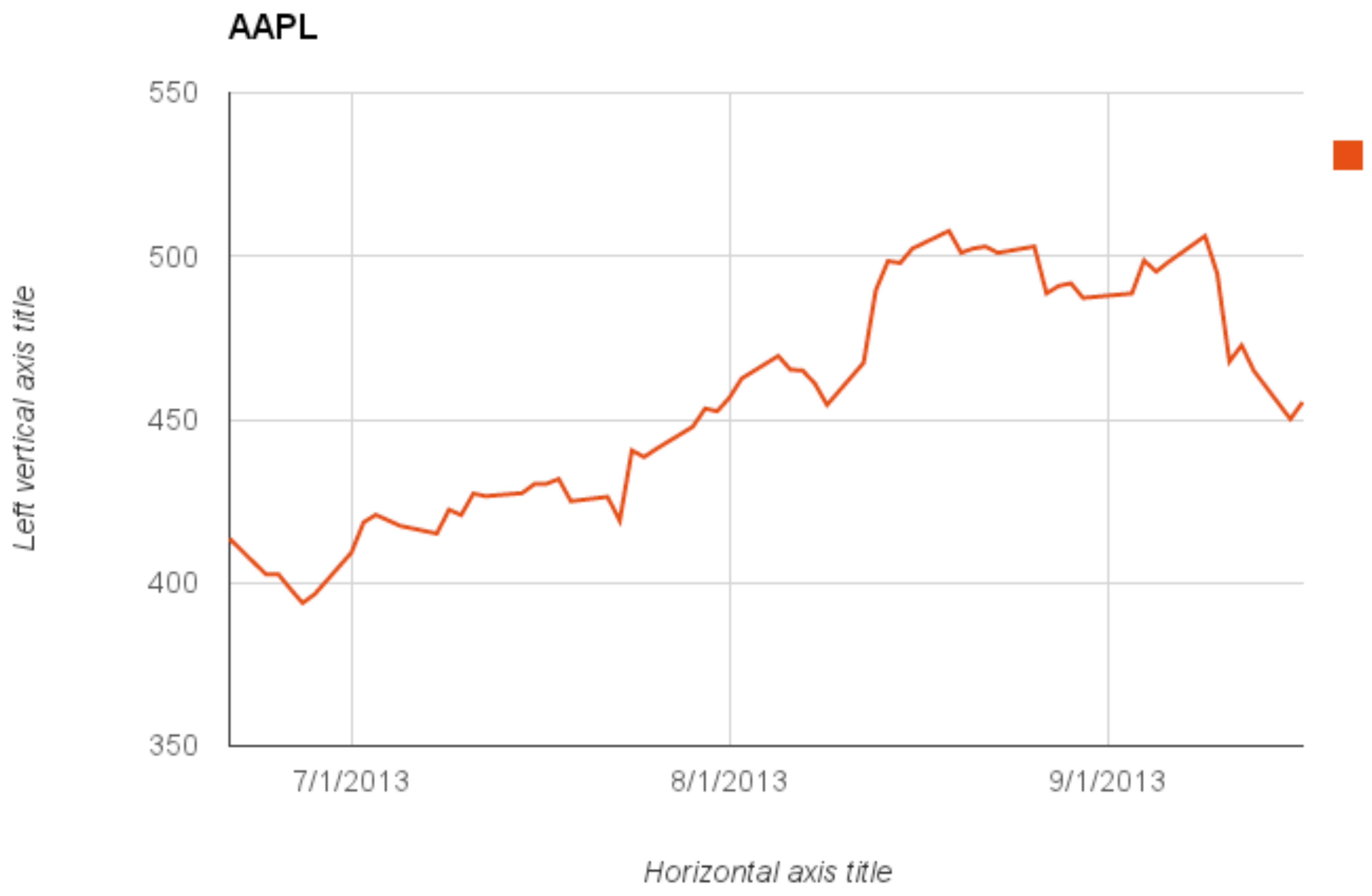
$$\Theta(n)$$

You can use induction like in the homework problem.

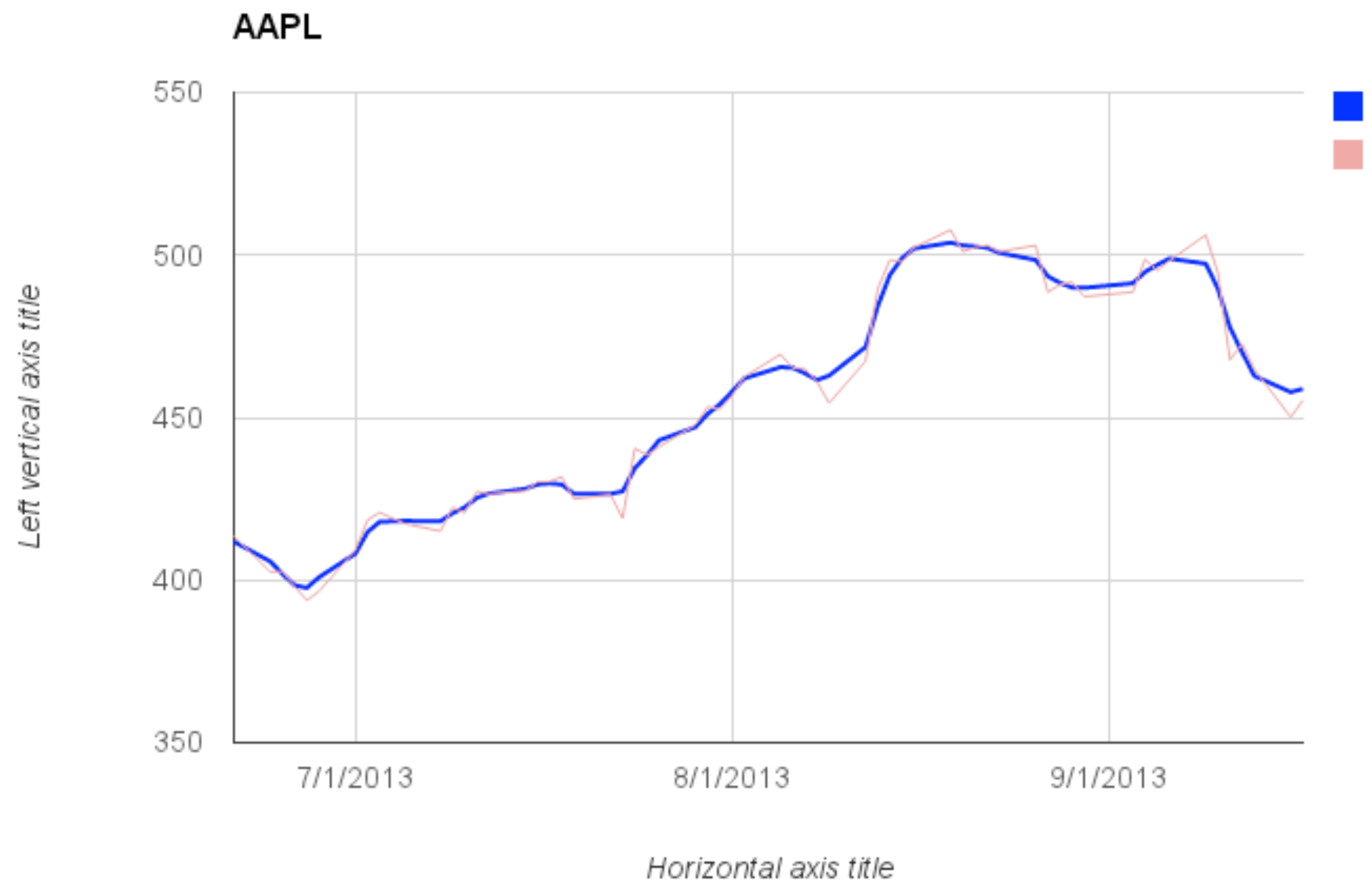
How to get intuition for S(n)

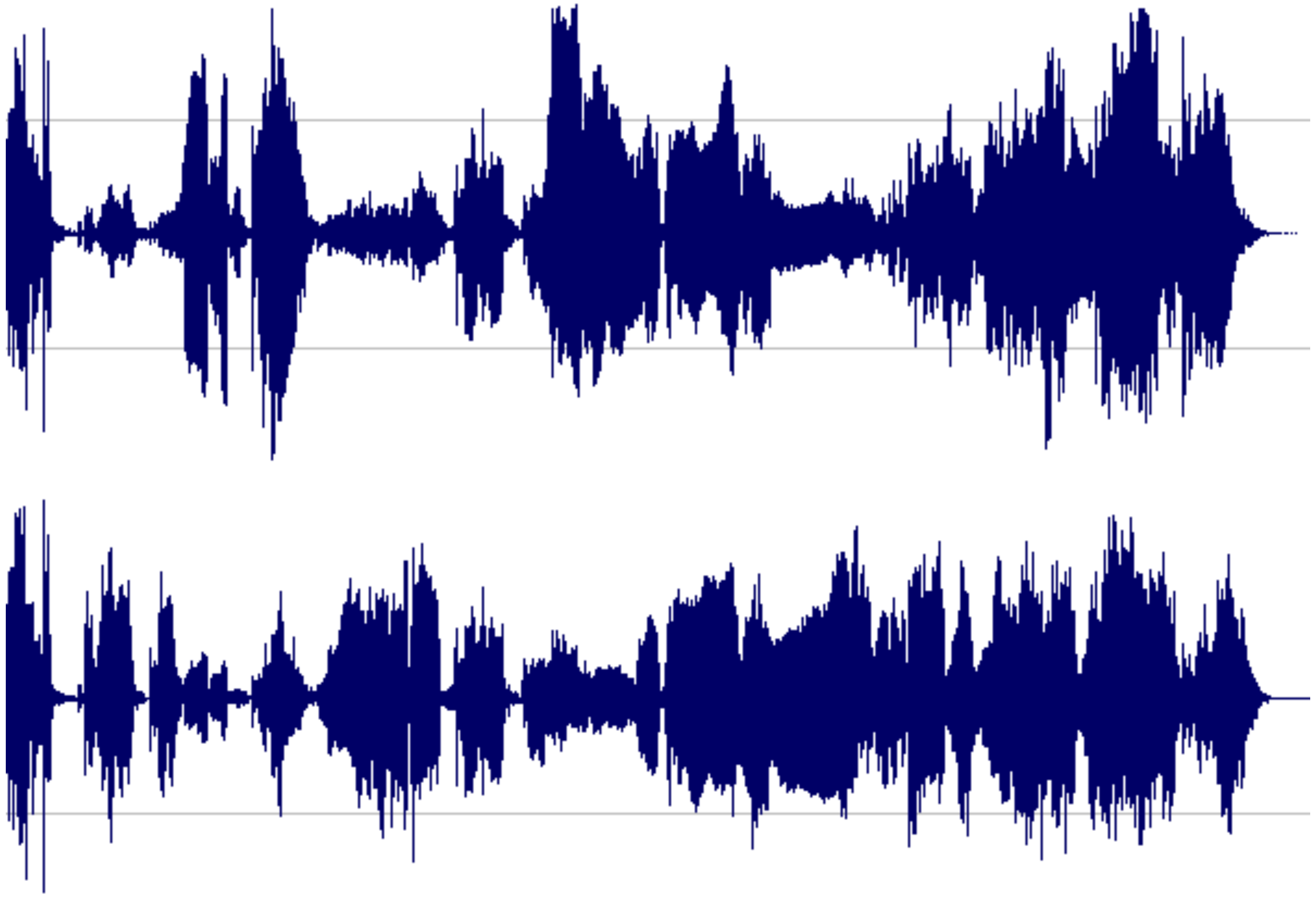
Fast Fourier Transform

© Jim Hatch Illustration / www.khulsey.com



AAPL
550
500
450
400
350
Left vertical axis title
7/1/2013
8/1/2013
9/1/2013
Horizontal axis title


AAPL
550
500
450
400
350
7/1/2013
8/1/2013
9/1/2013
Left vertical axis title
Horizontal axis title



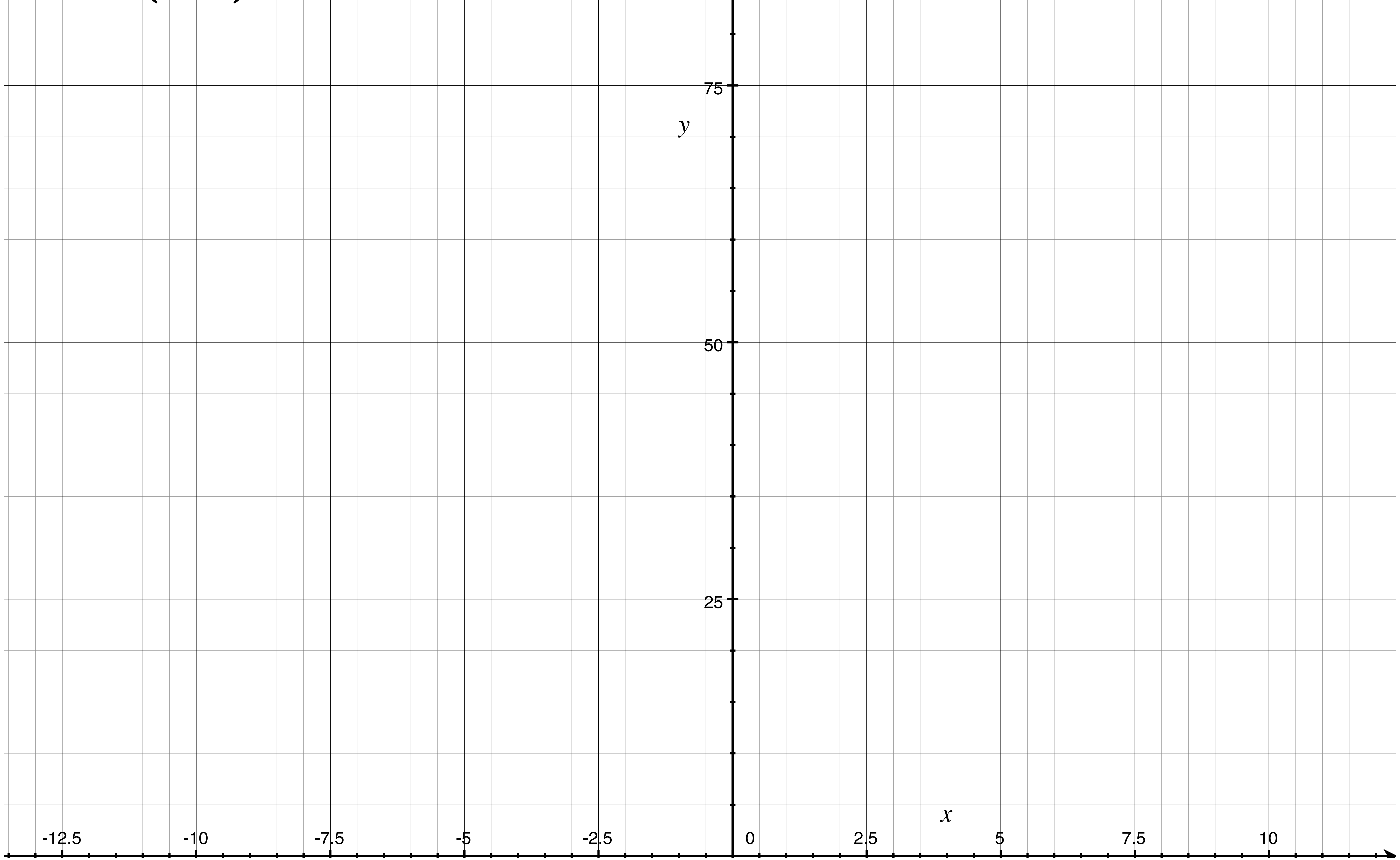
big ideas:

big ideas:

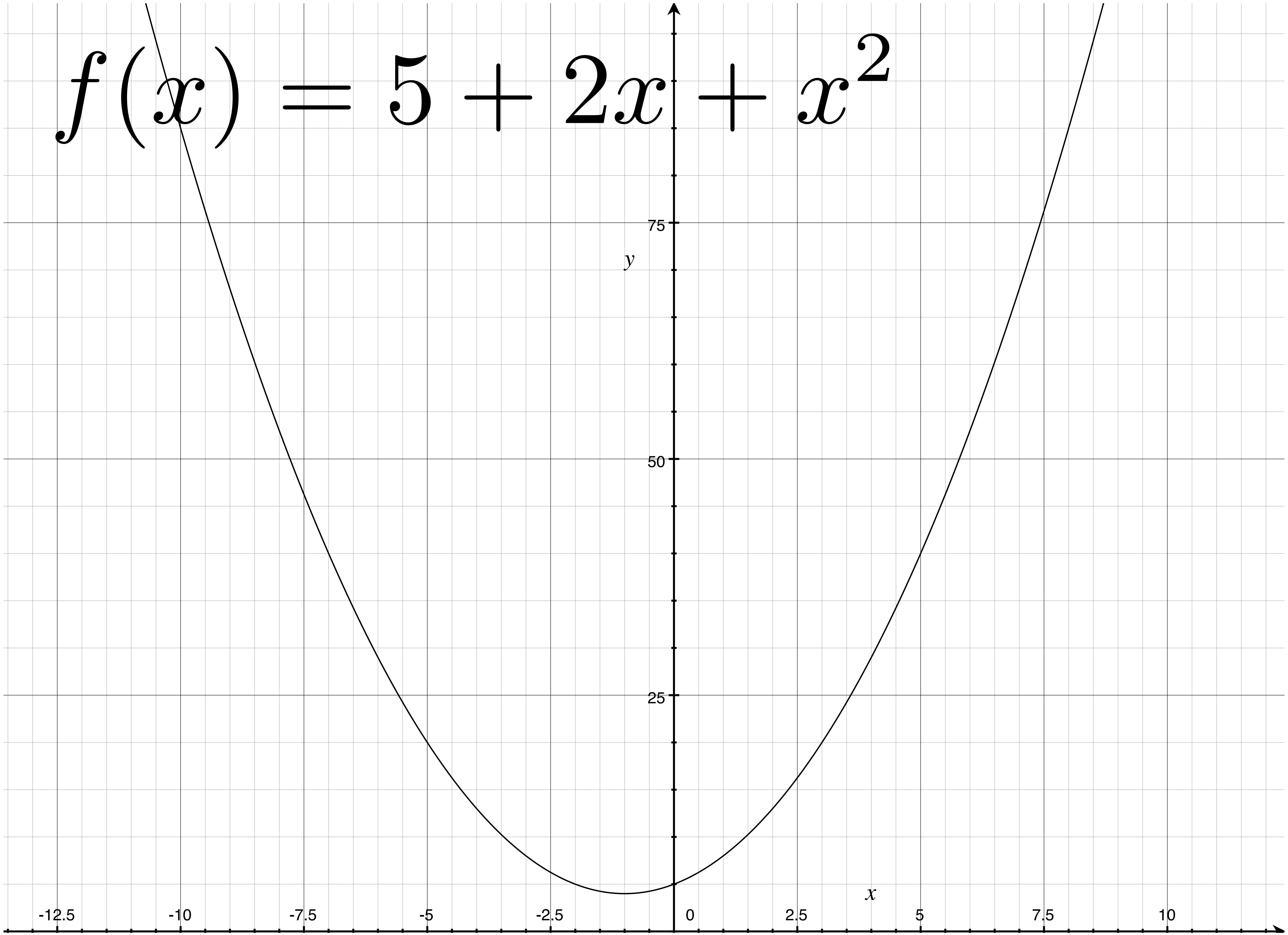
1. Changing representation from polynomial (coefficient form) into polynomial (point-wise form)

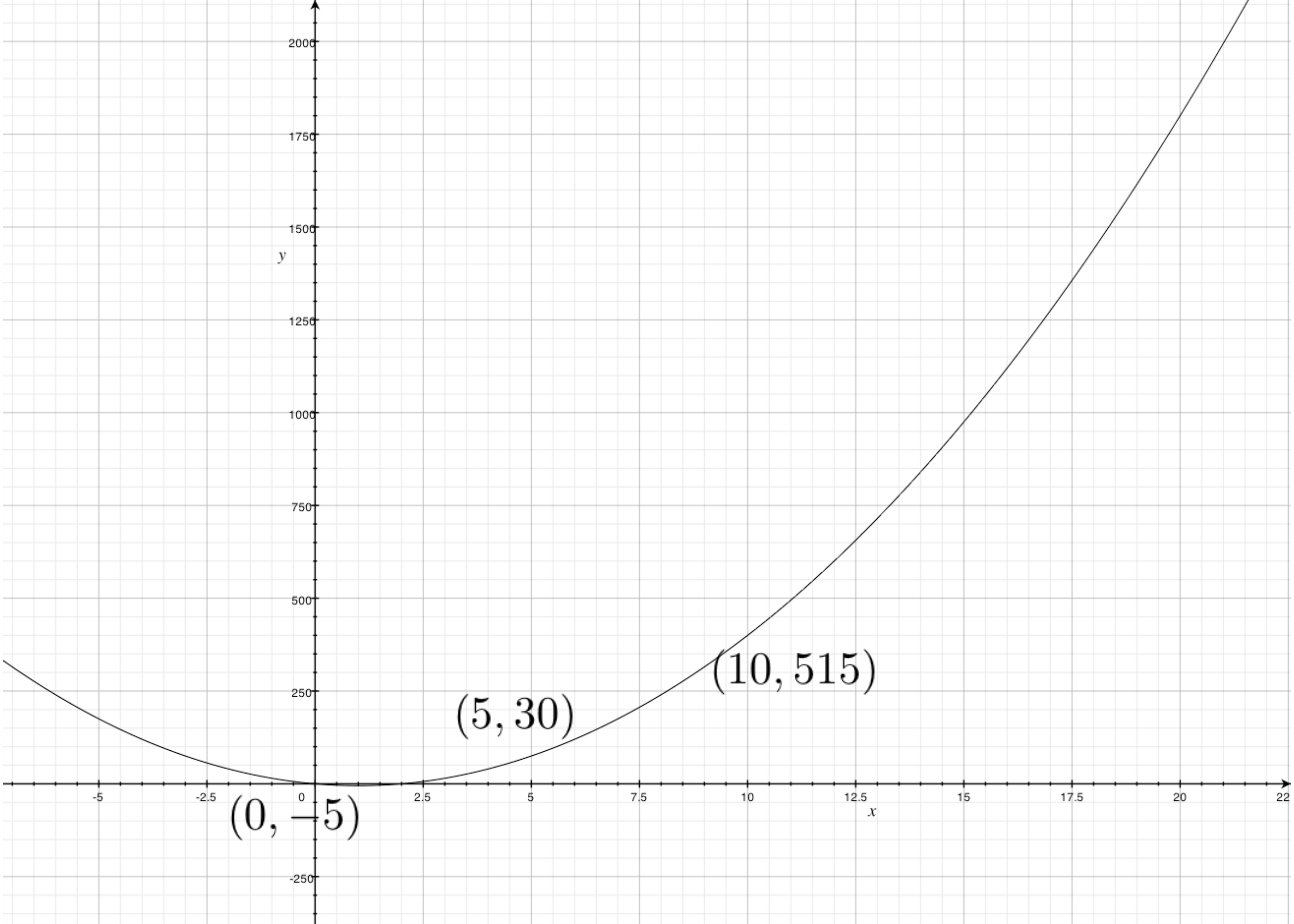
2. Clever divide and conquer

$$f(x) = 5 + 2x + x^2$$



$$f(x) = 5 + 2x + x^2$$





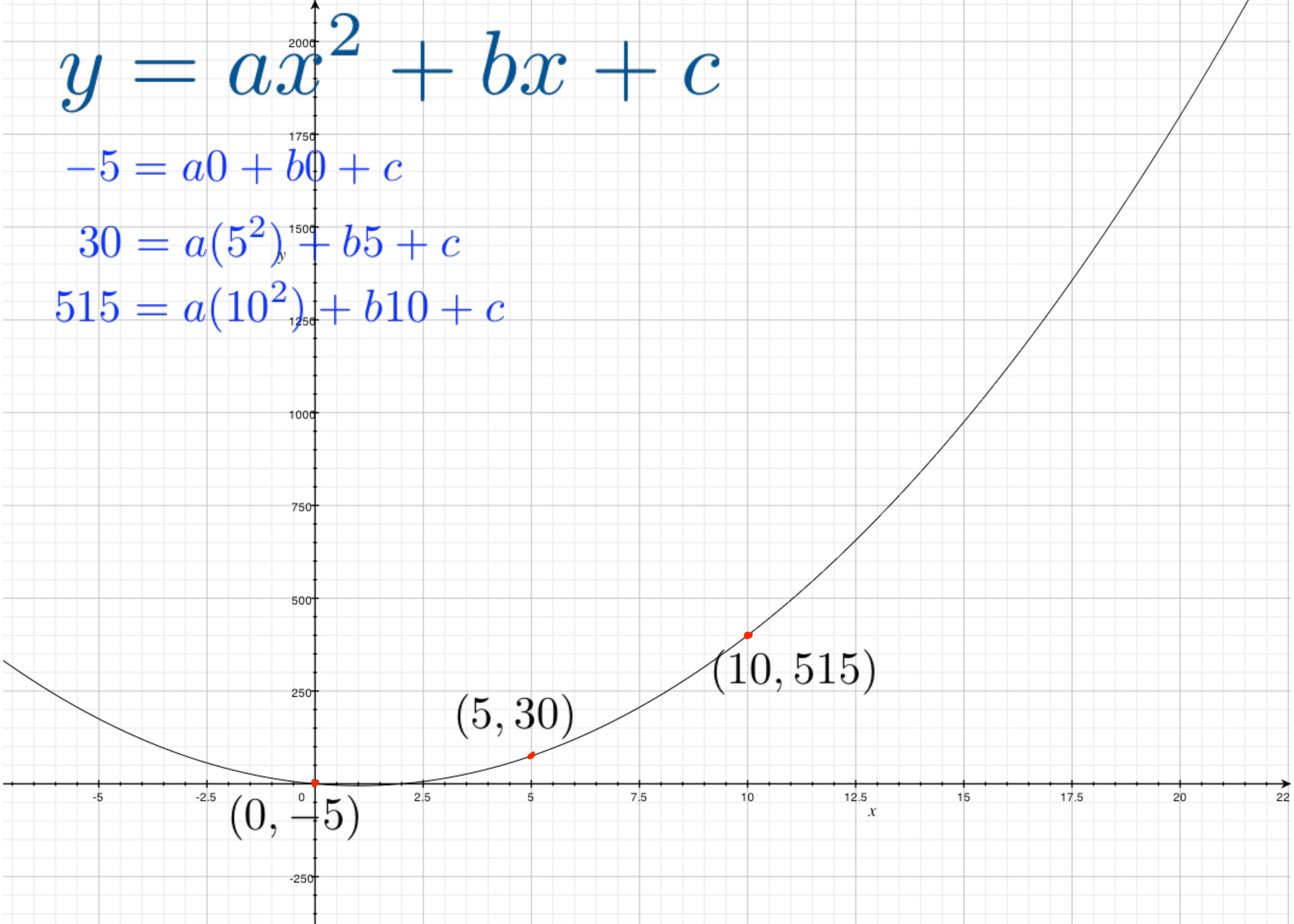
2000
1750
1500
y
1250
1000
750
500
250
-250
-5
-2.5
0
2.5
5
7.5
10
12.5
15
17.5
20
22
x
(0, −5)
(5, 30)
(10, 515)

$$y = ax^2 + bx + c$$

$$-5 = a0 + b0 + c$$

$$30 = a(5^2) + b5 + c$$

$$515 = a(10^2) + b10 + c$$

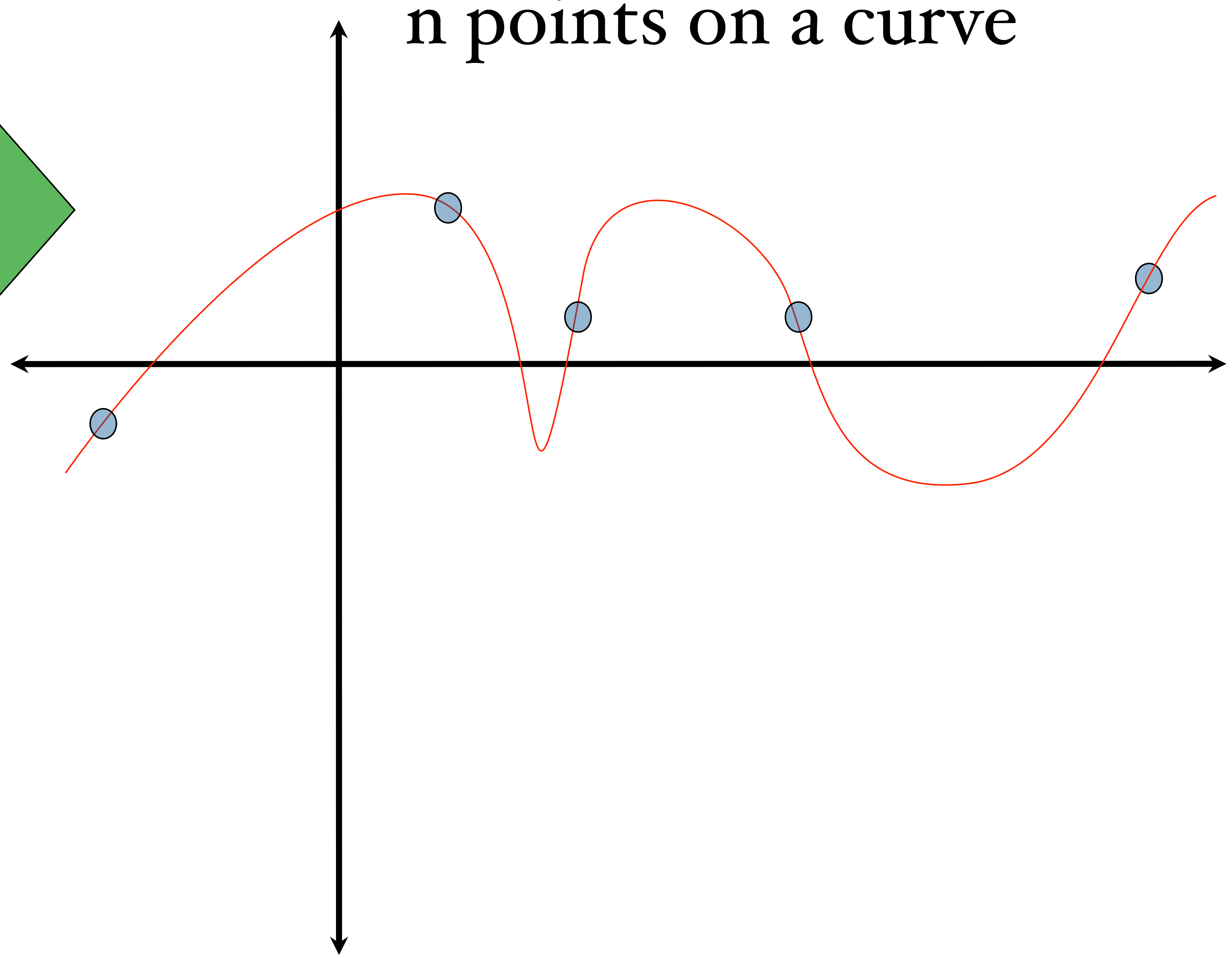


$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

degree $n-1$ polynomial

$$A(x)$$

n points on a curve



FFT

input: $a_0, a_1, a_2, \ldots, a_{n-1}$

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

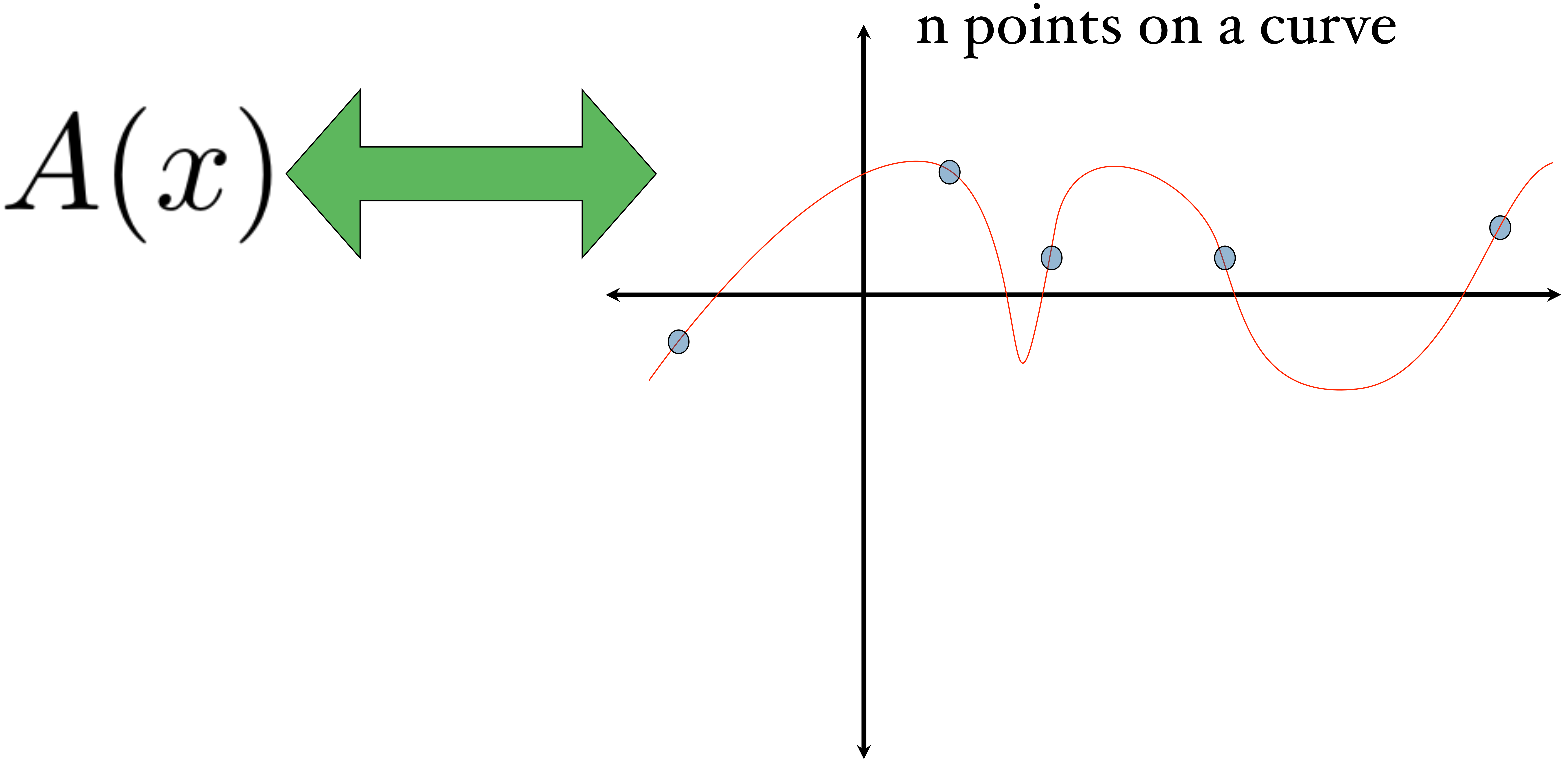
output:

FFT

input: $a_0, a_1, a_2, \ldots, a_{n-1}$

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

output: evaluate polynomial A at (any) n different points.



Later, we shall see that the same ideas for FFT can be used to implement Inverse-FFT.

Inverse FFT: Given n-points,

Later, we shall see that the same ideas for FFT can be used to implement Inverse-FFT.

Inverse FFT: Given n-points,

$$y_0, y_1, \cdots, y_{n-1}$$

find a degree n polynomial A such that

$$y_i = A(\omega_i)$$