Divide and Conquer

COMP 6651 – Algorithm Design Techniques

Denis Pankratov

D&C Strategy

Solve a problem by:

- **Breaking it into subproblems** that are themselves smaller instances of the same problem
- **Recursively solving** the subproblems
- **Appropriately combining** the solutions to subproblems

Integer Multiplication (CLRS Ch 4 notes)

Input: X, Y – two n-digit integers

Output: $X \cdot Y$

Example:

$$X = 4512354$$
$$Y = 1238970$$
$$X \cdot Y = 5590671235380$$
$$(n = 7)$$

X	Y	$X \cdot Y$
9	9	81
99	99	9801
999	999	998001
9999	9999	99980001
99999	99999	9999800001

Observation: if X and Y are n-digit numbers then $X \cdot Y$ is at most $2n$-digit number

High School Method

```
      2 6 4 2
    × 5 8 2 1
    ---------
      2 6 4 2
    5 2 8 4
  2 1 1 3 6
1 3 2 1 0
---------------
1 5 3 7 9 0 8 2
```

$X = [2, 6, 4, 2]$
$Y = [5, 8, 2, 1]$
$Z = [1, 5, 3, 7, 9, 0, 8, 2]$

$Multiply(X[1..n], Y[1..n])$
  $Z[1..2n] \leftarrow 0$
  $\boldsymbol{for}\ i = n\ \boldsymbol{down\ to}\ 1$
    $carry \leftarrow 0$
    $\boldsymbol{for}\ j = n\ \boldsymbol{down\ to}\ 1$
      $m \leftarrow Z[i + j] + carry + X[j] \cdot Y[i]$
      $Z[i + j] \leftarrow m\ mod\ 10$
      $carry \leftarrow \left\lfloor \frac{m}{10} \right\rfloor$
    $Z[i] \leftarrow carry$
  $\boldsymbol{return}\ Z$

$Multiply(X[1..n], Y[1..n])$

  $Z[1..2n] \leftarrow 0$

  $\boldsymbol{for}\ i = n\ \boldsymbol{down\ to}\ 1$

    $carry \leftarrow 0$

    $\boldsymbol{for}\ j = n\ \boldsymbol{down\ to}\ 1$

      $m \leftarrow Z[i+j] + carry + X[j] \cdot Y[i]$

      $Z[i+j] \leftarrow m\ mod\ 10$

      $carry \leftarrow \left\lfloor \frac{m}{10} \right\rfloor$

    $Z[i] \leftarrow carry$

  $\boldsymbol{return}\ Z$

Cost $O(1)$

n

n

Cost measure: number of single-digit multiplications

$M(n)$ = worst-case cost of $Multiply$ on inputs of length n

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

Can we multiply two integers faster?

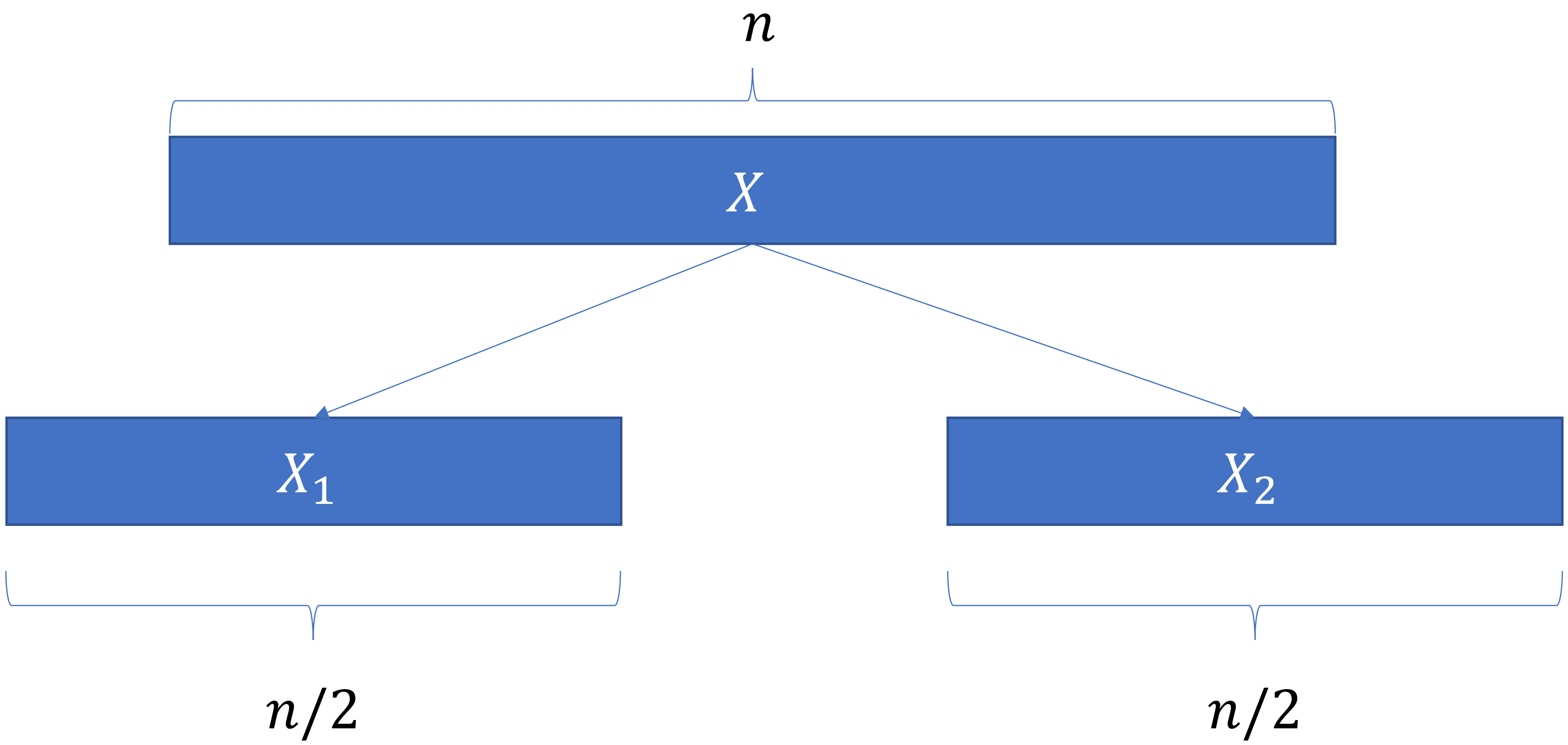
In 1960s Kolmogorov conjectured NO

Karatsuba disproved the conjecture

Karatsuba's idea: divide and conquer!

$$\boxed{X = 10^{n/2} X_1 + X_2}$$

$$\boxed{Y = 10^{n/2} Y_1 + Y_2}$$



$$X \cdot Y = \left(10^{\frac{n}{2}} X_1 + X_2\right) \cdot \left(10^{\frac{n}{2}} Y_1 + Y_2\right)$$
$$= 10^n X_1 \cdot Y_1 + 10^{\frac{n}{2}} (X_1 \cdot Y_2 + X_2 \cdot Y_1) + X_2 \cdot Y_2$$

$Mutliply(X, Y)$:

$\quad \boldsymbol{if}\ n = 1$

$\quad\quad return\ X \cdot Y$

$\quad R_1 \leftarrow Multiply(X_1, Y_1)$

$\quad R_2 \leftarrow Multiply(X_1, Y_2)$

$\quad R_3 \leftarrow Multiply(X_2, Y_1)$

$\quad R_4 \leftarrow Multiply(X_2, Y_2)$

$\quad return\ 10^n R_1 + 10^{\frac{n}{2}} (R_2 + R_3) + R_4$

$M(n) =$ number of single digit multiplications in this procedure

$M(n) = 4M\left(\frac{n}{2}\right)$

$M(1) = 1$

Solves to $M(n) = \Theta(n^2)$ (see Master's theorem)

No improvement ☹

Cool idea:

$$X \cdot Y = 10^n X_1 \cdot Y_1 + 10^{\frac{n}{2}}(X_1 \cdot Y_2 + X_2 \cdot Y_1) + X_2 \cdot Y_2$$

We don't need $X_1 \cdot Y_2$ and $X_2 \cdot Y_1$ separately

We only need $W = X_1 \cdot Y_2 + X_2 \cdot Y_1$

Can we compute W with one extra recursive call?

$$(X_1 - X_2) \cdot (Y_1 - Y_2) = X_1 Y_1 - (X_1 \cdot Y_2 + X_2 \cdot Y_1) + X_2 Y_2$$

$R_1 \leftarrow Multiply(X_1, Y_1)$
$R_2 \leftarrow Multiply(X_2, Y_2)$
$R_3 \leftarrow Multiply(X_1 - X_2, Y_1 - Y_2)$

Then: $\boldsymbol{W = R_1 + R_2 - R_3}$

$Multiply(X, Y)$

$if\ n = 1$

$return\ X \cdot Y$

// Split X and Y in half

$X = 10^{\frac{n}{2}} X_1 + X_2$

$Y = 10^{\frac{n}{2}} Y_1 + Y_2$

$R_1 \leftarrow Multiply(X_1, Y_1)$

$R_2 \leftarrow Multiply(X_2, Y_2)$

$R_3 \leftarrow Multiply(X_1 - X_2, Y_1 - Y_2)$

$W \leftarrow R_1 + R_2 - R_3$

$return\ 10^n R_1 + 10^{\frac{n}{2}} W + R_2$

$M(n) =$ number of single digit multiplications in this procedure

$M(n) = 3M\left(\frac{n}{2}\right)$

$M(1) = 1$

Solves to $M(n) = \Theta\left(n^{\log_2 3}\right) = O(n^{1.585})$ (see Master's theorem)

Comments

(1) Actual runtime also includes additions, copying arrays, and shifting arrays

$$T(n) = \text{worst} - \text{case runtime}$$

$$T(n) = 3\,T\left(\frac{n}{2}\right) + O(n)$$

$$T(1) = O(1)$$

Still solves to $T(n) = O\left(n^{\log_2 3}\right)$ (e.g., Master's Theorem)

(2) What if n is not divisible by 2?

There exists $n \le n' \le 2n$ such that n' is a power of 2

$$T(n) \le T(n') = O\left((n')^{\log_2 3}\right) = O\left((2n)^{\log_2 3}\right) = O\left(n^{\log_2 3}\right)$$

Maximum Subarray Problem (CLRS 4.1)

Input: $A[1..n]$ – array of n integers

Output: S – maximum sum of a contiguous subarray, i.e.,

there exists $1 \leq i < j \leq n$ such that

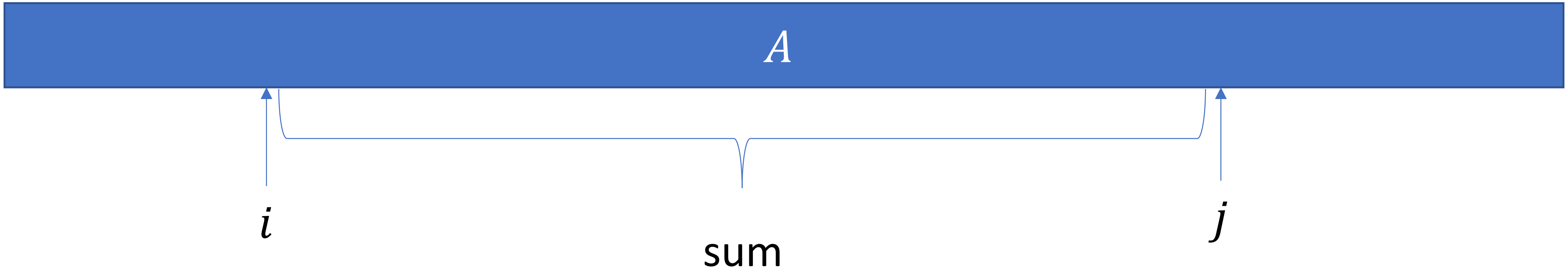
$S = \sum_{k=i}^{j} A[k]$ and S is maximized

Example:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

Trivial Algorithm

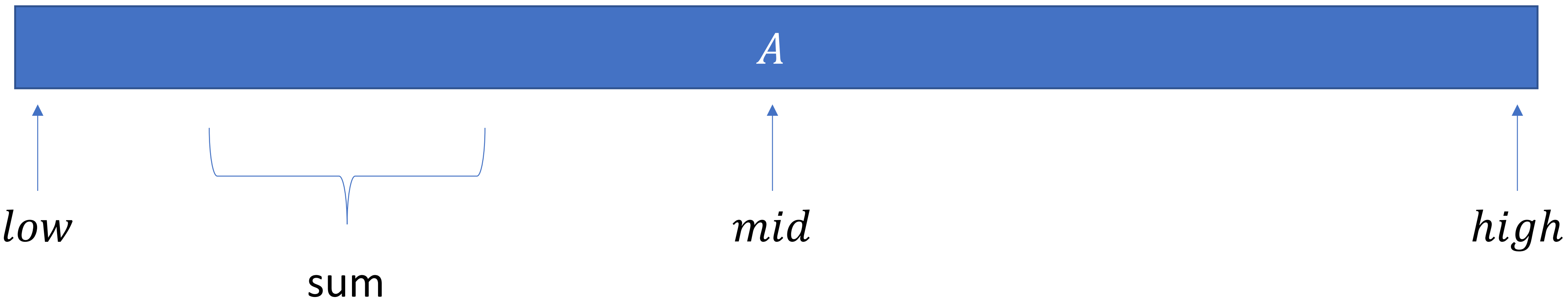


Check every pair of indices $1 \leq i < j \leq n$

Even if we can compute each such sum in constant time
there are still $\binom{n}{2} = \Theta(n^2)$ such pairs of indices i and j

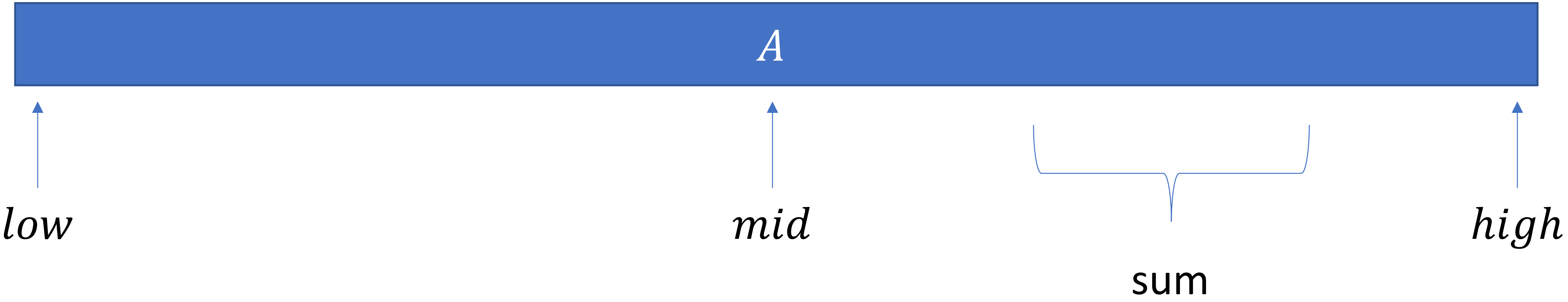
Trivial algorithm runs in time $\Omega(n^2)$

Divide and Conquer



Maximum subarray $A[i..j]$ either doesn't cross mid
then it entirely lies in $A[low..mid]$

Divide and Conquer

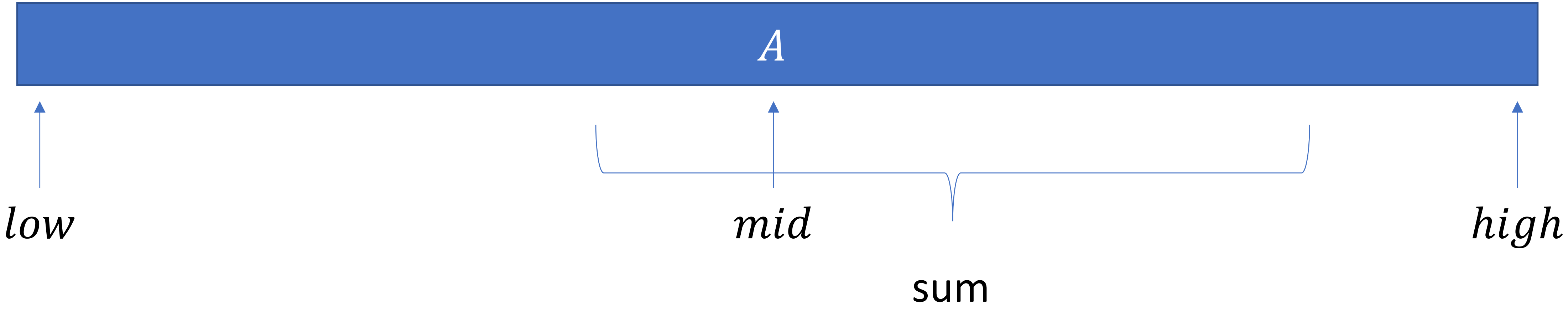


Maximum subarray $A[i..j]$ either doesn't cross mid

then it entirely lies in $A[low..mid]$

or it entirely lies in $A[mid+1..high]$

Divide and Conquer

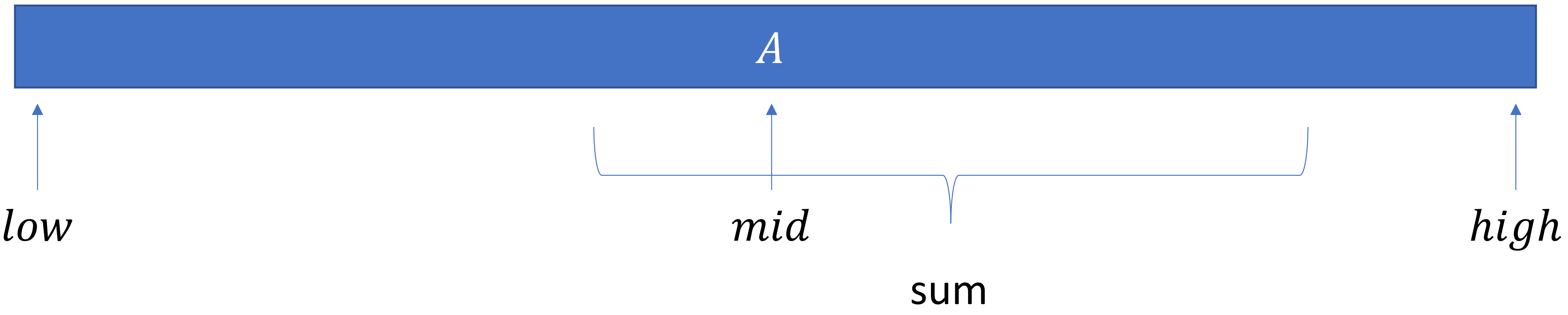


Maximum subarray $A[i..j]$ either doesn't cross mid

then it entirely lies in $A[low..mid]$

or it entirely lies in $A[mid+1..high]$

OR maximum subarray crosses mid



$MaxCrossingSubarray(A, low, mid, high)$

$$
\begin{aligned}
&\quad L \leftarrow -\infty; R \leftarrow -\infty \\
&\quad S \leftarrow 0 \\
&\quad \boldsymbol{for}\ i = mid\ \boldsymbol{down\ to}\ low \\
&\qquad S \leftarrow S + A[i] \\
&\qquad L \leftarrow \max(L, S) \\
&\quad S \leftarrow 0 \\
&\quad \boldsymbol{for}\ i = mid + 1\ \boldsymbol{to}\ high \\
&\qquad S \leftarrow S + A[i] \\
&\qquad R \leftarrow \max(R, S) \\
&\quad \boldsymbol{return}\ L + R
\end{aligned}
$$

Observe: linear amount of work

$MaxSubarray(A, low, high)$

$\boldsymbol{if}\ high == low + 1$

$return\ A[low] + A[high]$

$\boldsymbol{if}\ \ high \leq low$

$return\ -\infty$

$mid \leftarrow \left\lfloor \frac{low+high}{2} \right\rfloor$

$left \leftarrow MaxSubarray(A, low, mid)$

$right \leftarrow MaxSubarray(A, mid + 1, high)$

$cross \leftarrow MaxCrossingSubarray(A, low, mid, high)$

$\boldsymbol{return}\ \max(left, cross, right)$

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

$T(n)$ = worst-case runtime on instances of length n

Runtime

$MaxSubarray$ on input of length n:

makes 2 recursive calls on inputs of length $\frac{n}{2}$

makes additional $O(n)$ amount of work

Thus, we have $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Base cases $T(0), T(1), T(2) = O(1)$

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

Note: using recursion tree to understand recurrence

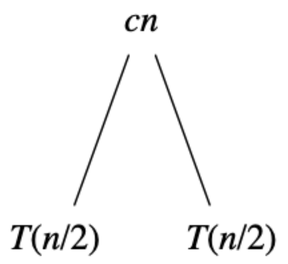
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

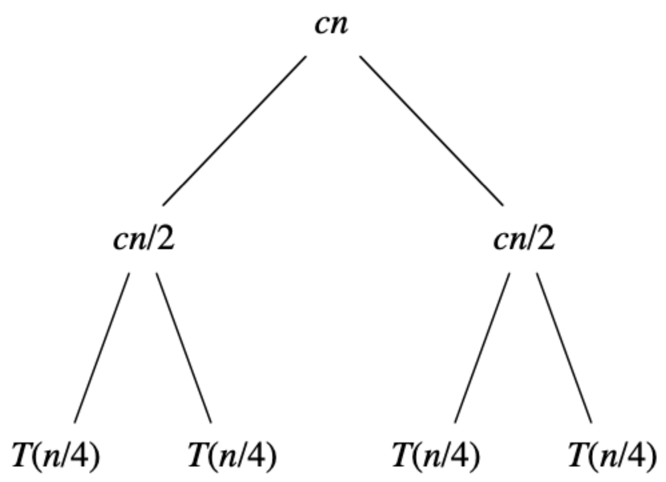
We rewrite the recurrence as

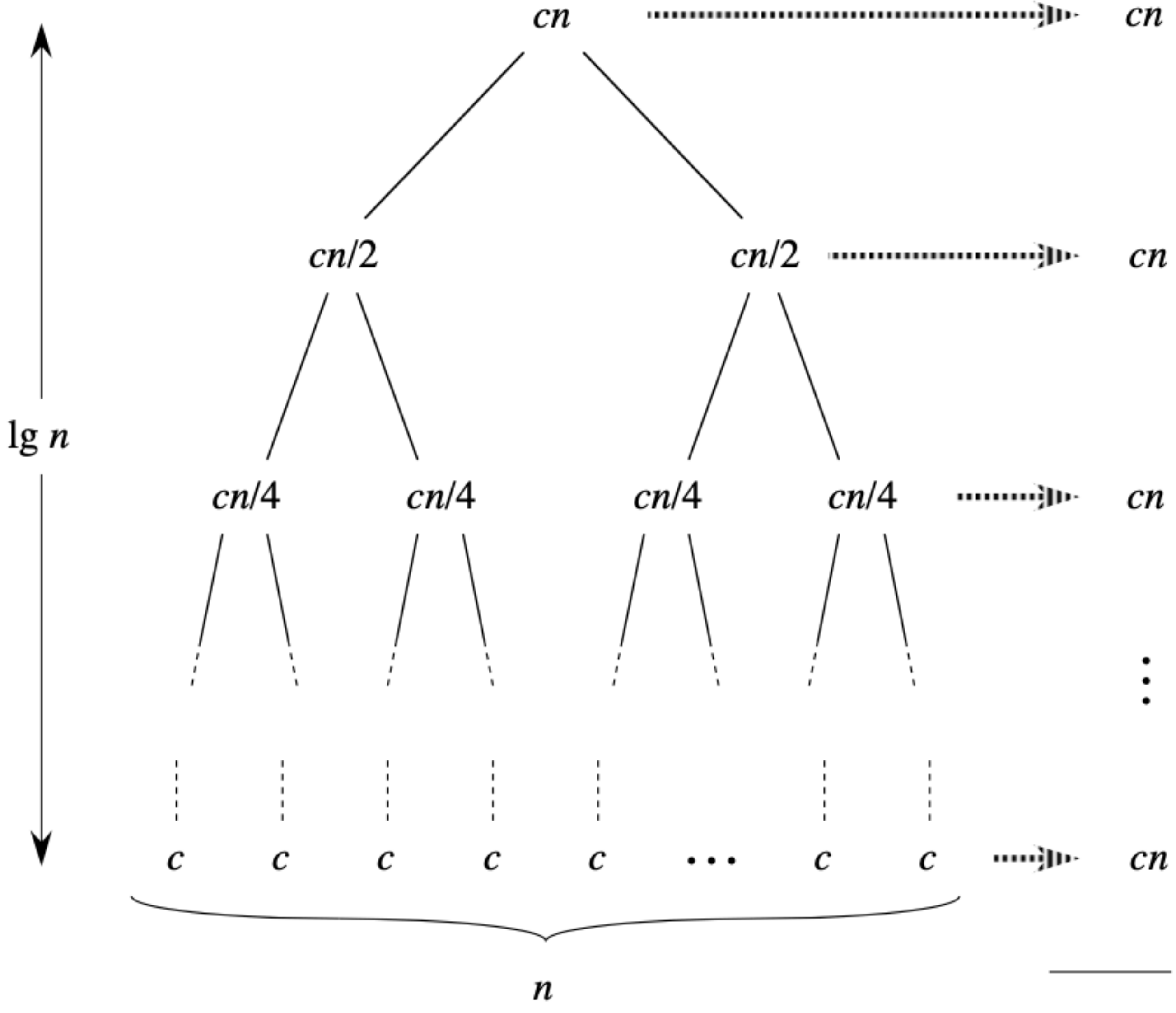
$$T(n) = \begin{cases} c & \text{if } n = 1 , \\ 2T(n/2) + cn & \text{if } n > 1 . \end{cases}$$

2 steps:

1 step:







Total: $cn \lg n + cn$

Merge Sort (CLRS 2.3)

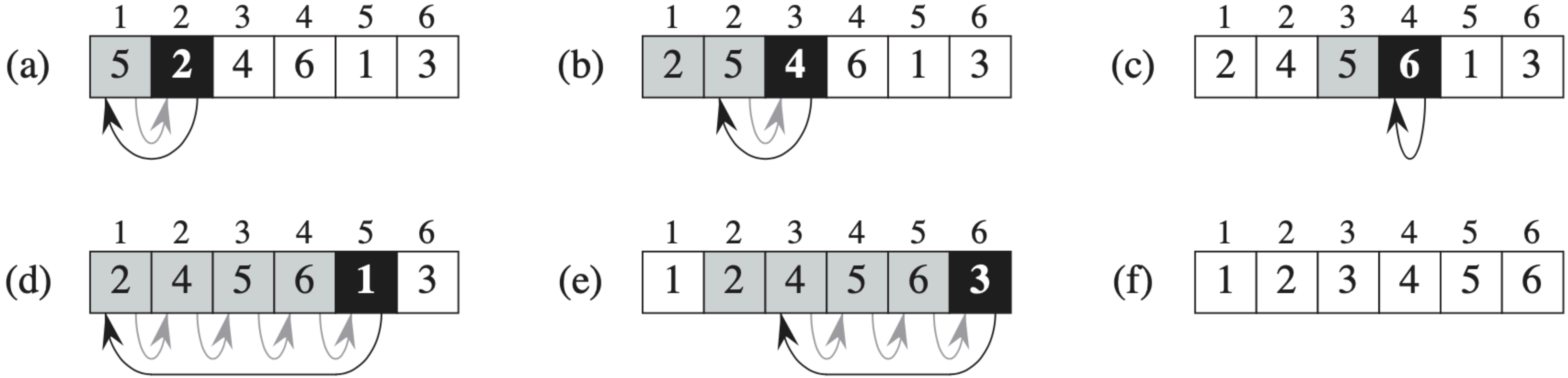
Input: $A[1..n]$ – array of n integers

Output: $A[1..n]$ – reordering of the input array so that elements are in a non-decreasing order

Example:

$$[5, 5, 2, 1, 4, 2, 0, -4]$$
$$[-4, 0, 1, 2, 2, 4, 5, 5]$$

Trivial Algorithm: Insertion Sort Example



$InsertionSort(A)$

$\quad \boldsymbol{for}\ j = 2\ \boldsymbol{to}\ n$

$\quad\quad val \leftarrow A[j]$

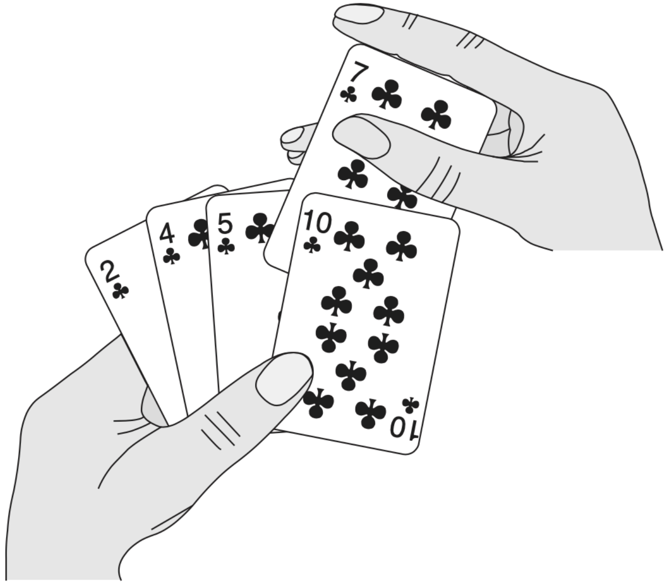
$\quad\quad i \leftarrow j - 1$

$\quad\quad \boldsymbol{while}\ i > 0\ \boldsymbol{and}\ A[i] > val$

$\quad\quad\quad A[i+1] \leftarrow A[i]$

$\quad\quad\quad i \leftarrow i - 1$

$\quad\quad A[i+1] \leftarrow val$



Runtime $T(n) = O(n^2)$

Other simple $O(n^2)$ sorts: $SelectionSort$, $BubbleSort$

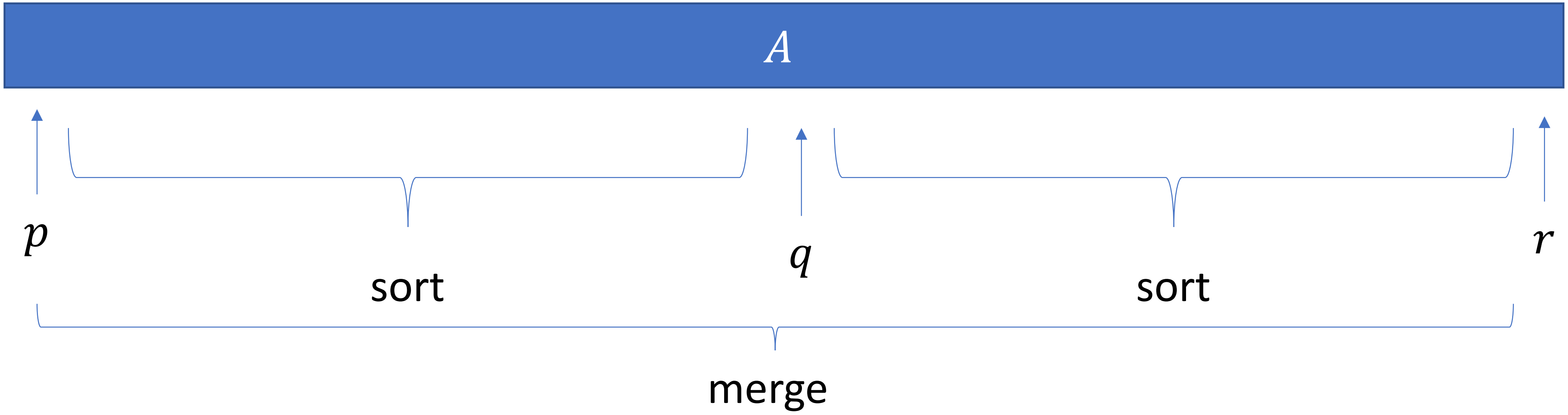
D&C: Merge Sort

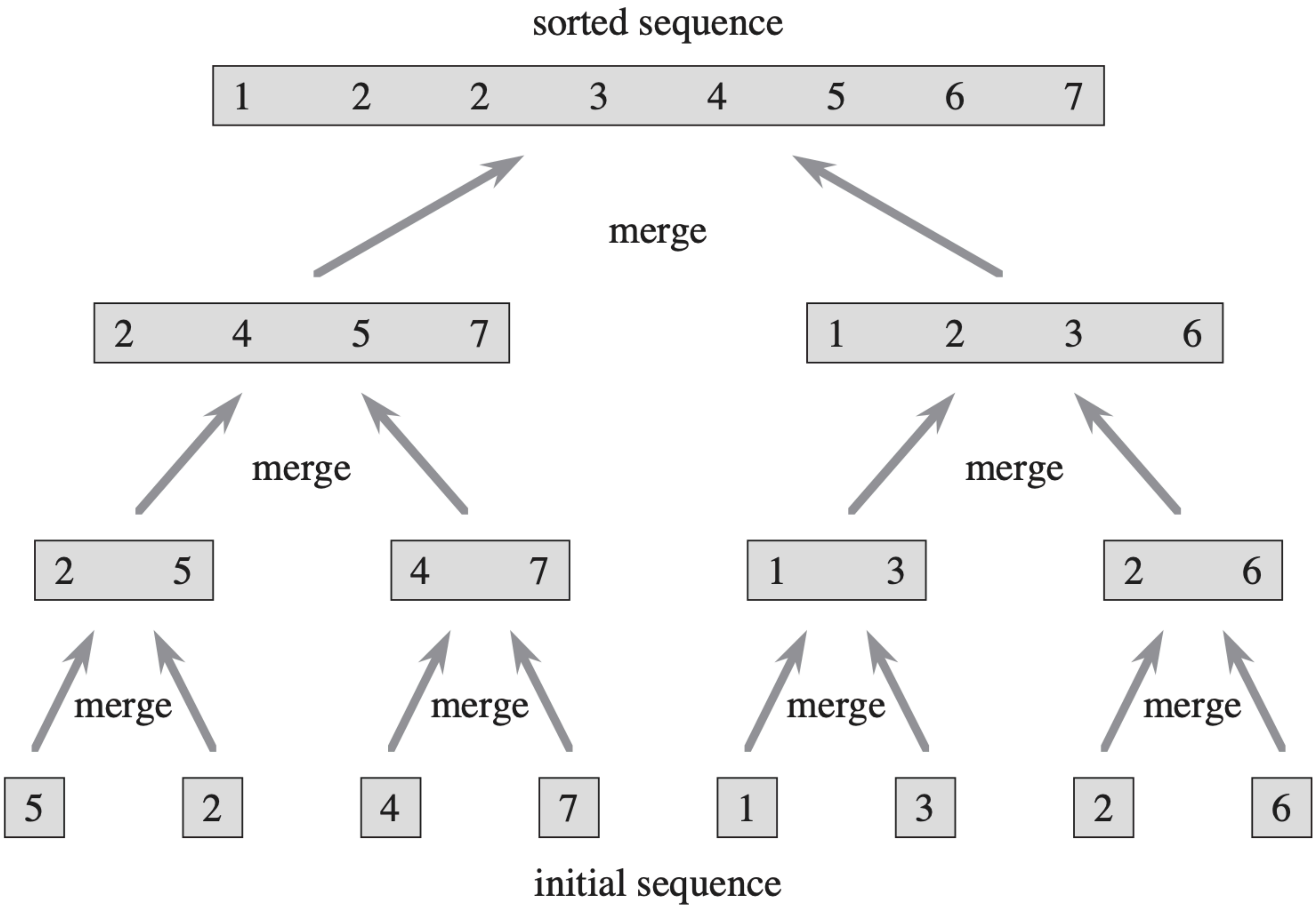
To sort $A[p \mathinner{..} r]$:

Divide by splitting into two subarrays $A[p \mathinner{..} q]$ and $A[q+1 \mathinner{..} r]$, where q is the halfway point of $A[p \mathinner{..} r]$.

Conquer by recursively sorting the two subarrays $A[p \mathinner{..} q]$ and $A[q+1 \mathinner{..} r]$.

Combine by merging the two sorted subarrays $A[p \mathinner{..} q]$ and $A[q+1 \mathinner{..} r]$ to produce a single sorted subarray $A[p \mathinner{..} r]$. To accomplish this step, we'll define a procedure $\text{MERGE}(A, p, q, r)$.




sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

$MergeSort(A, p, r)$

$\quad if\ p < r$

$\qquad q \leftarrow \left\lfloor \frac{p+r}{2} \right\rfloor$

$\qquad MergeSort(A, p, q)$

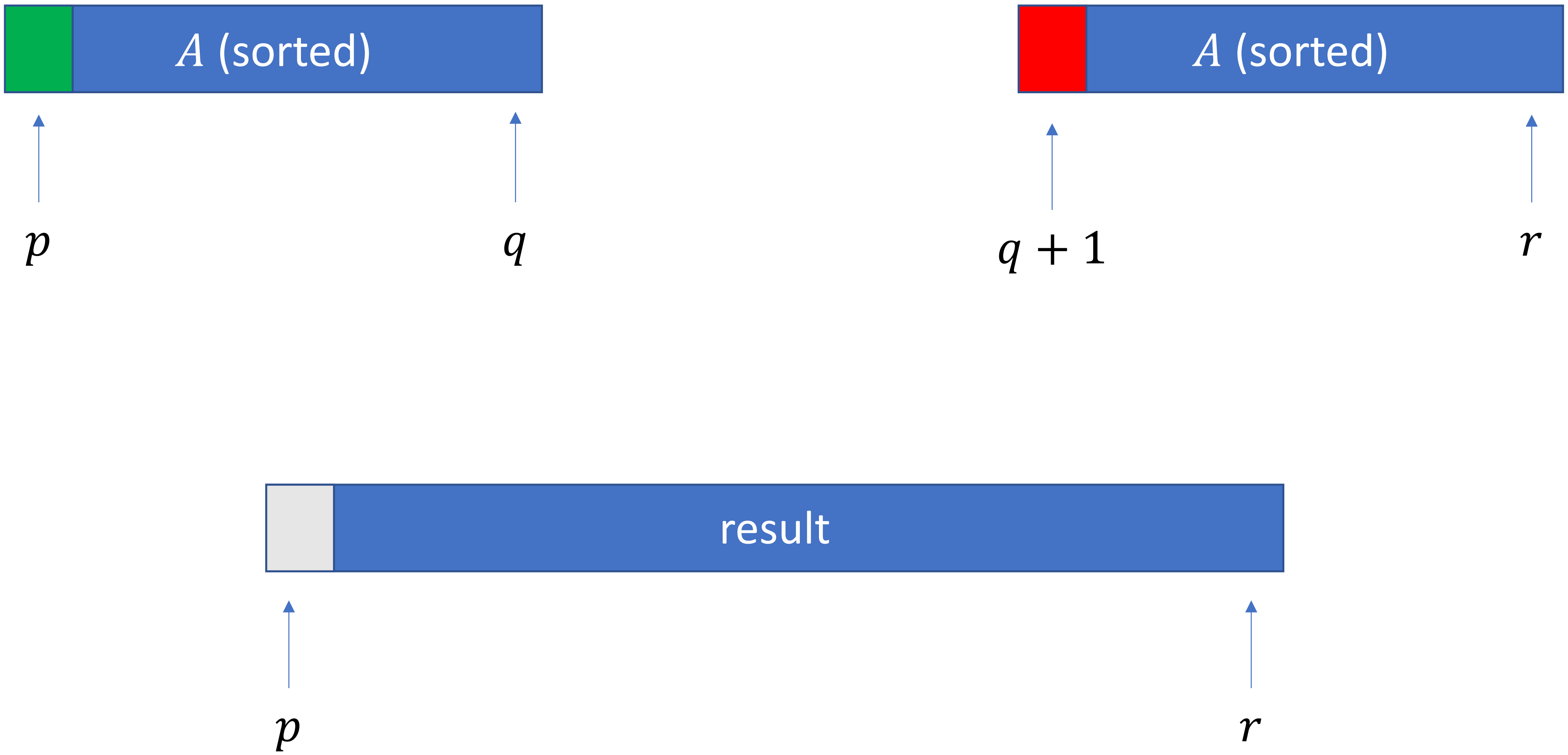
$\qquad MergeSort(A, q + 1, r)$

$\qquad Merge(A, p, q, r)$

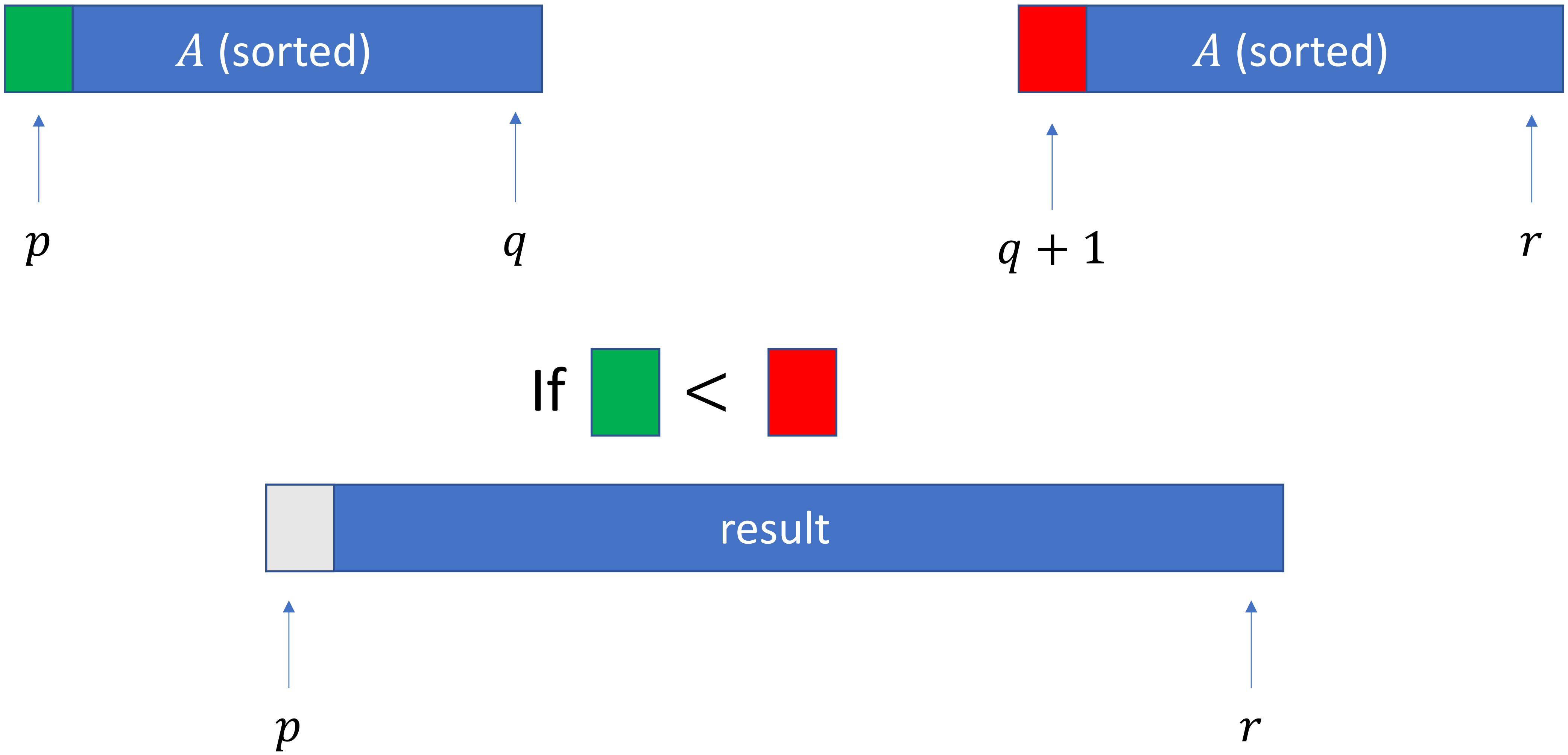
Initial call: $MergeSort(A, 1, n)$

How to do merge efficiently?

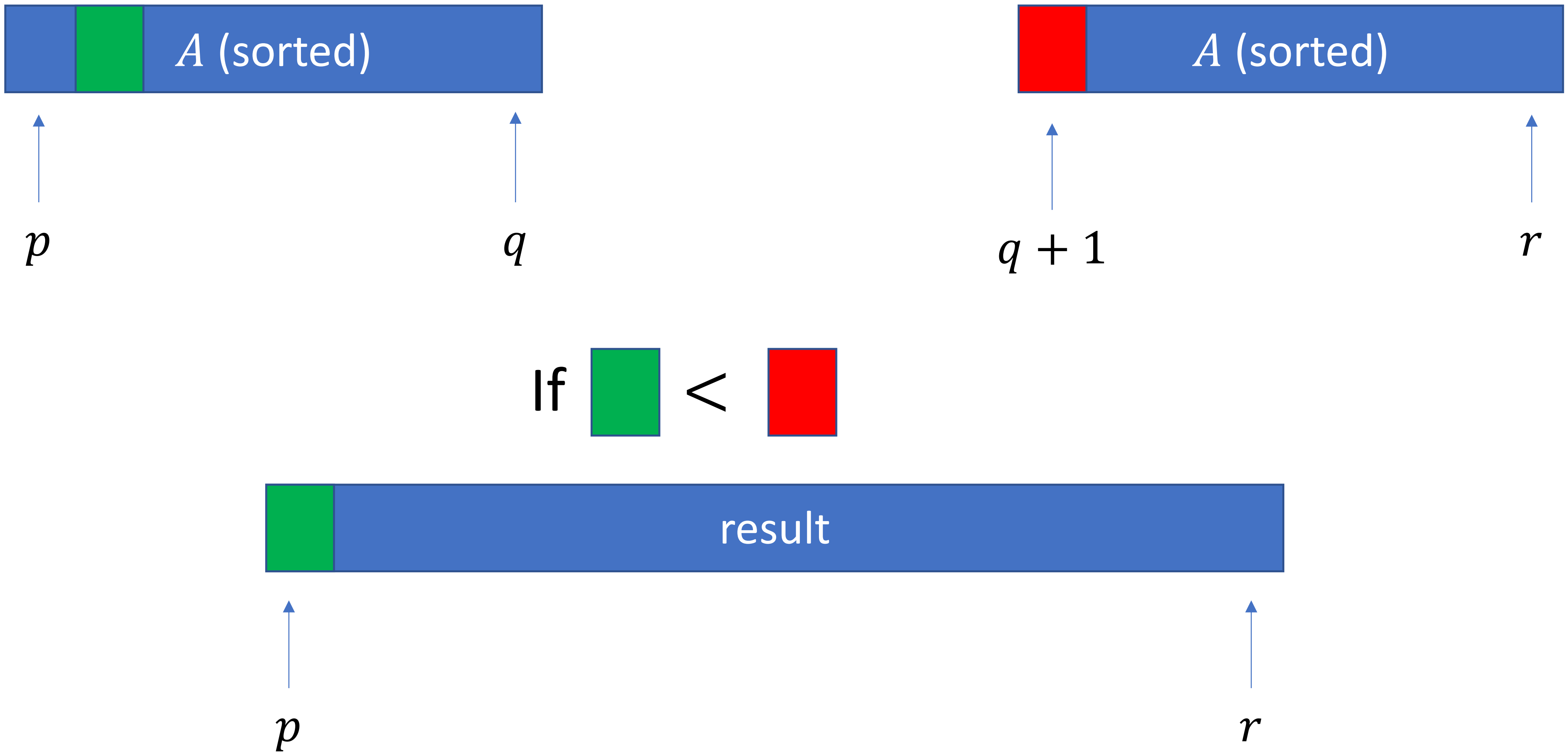
Merge Idea



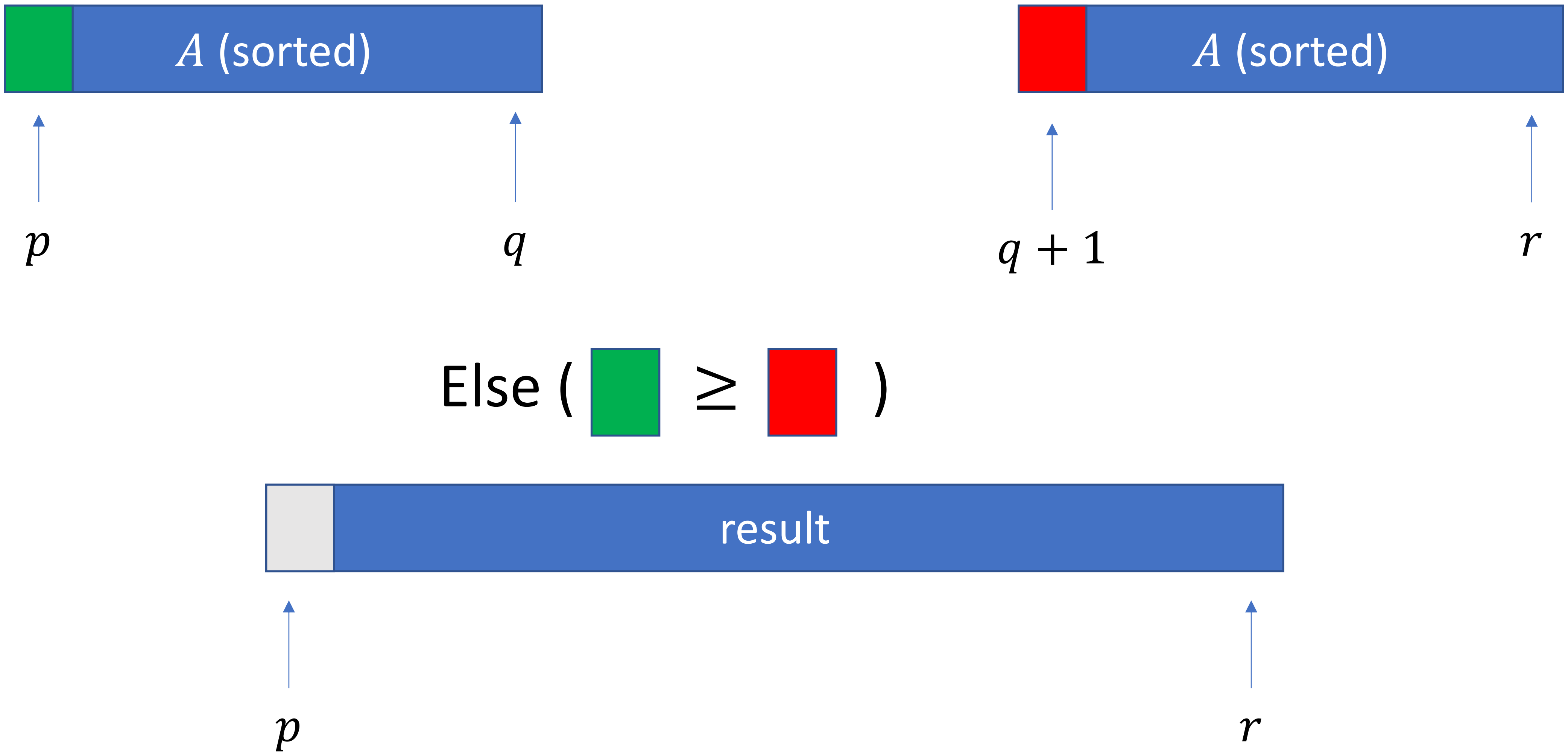
Merge Idea



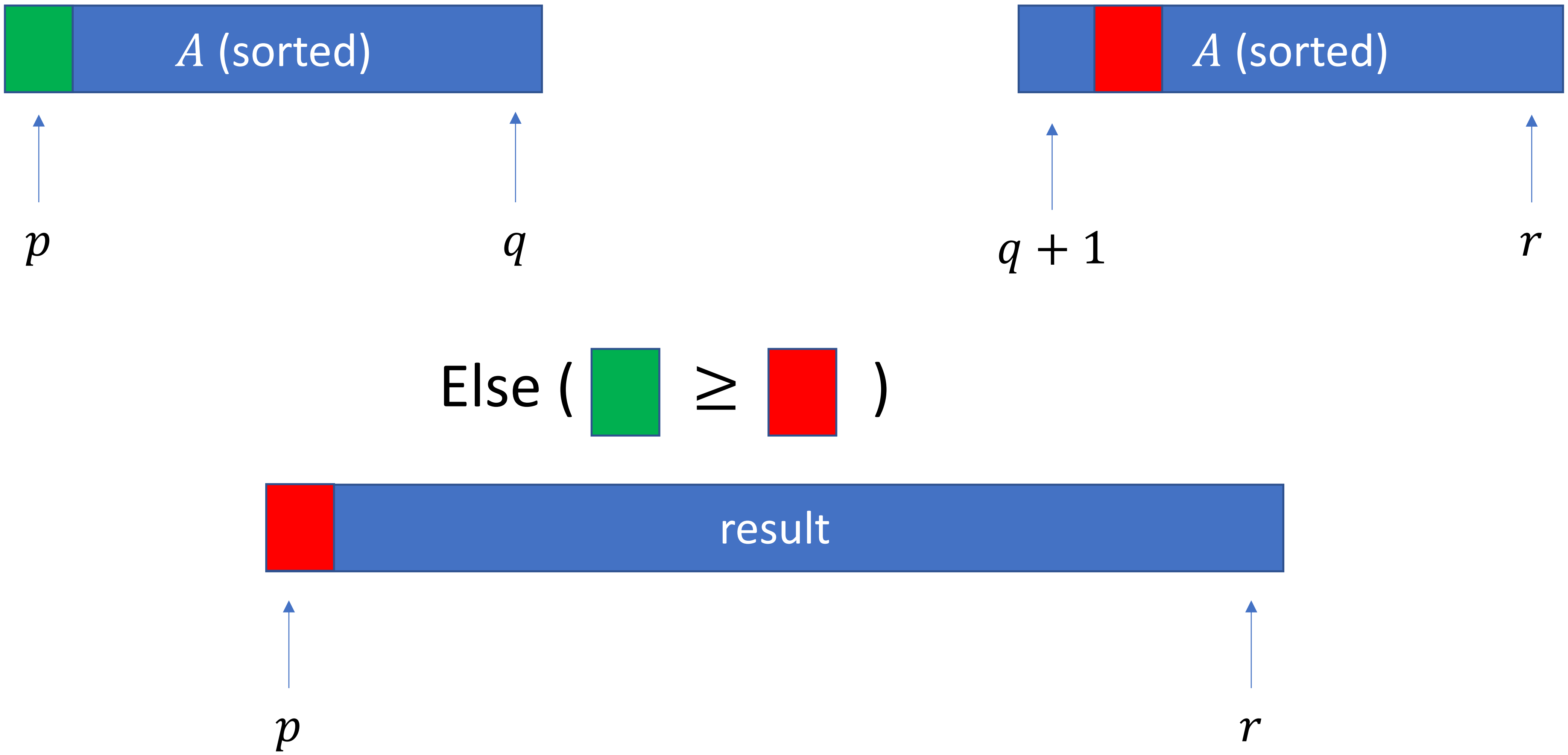
Merge Idea



Merge Idea

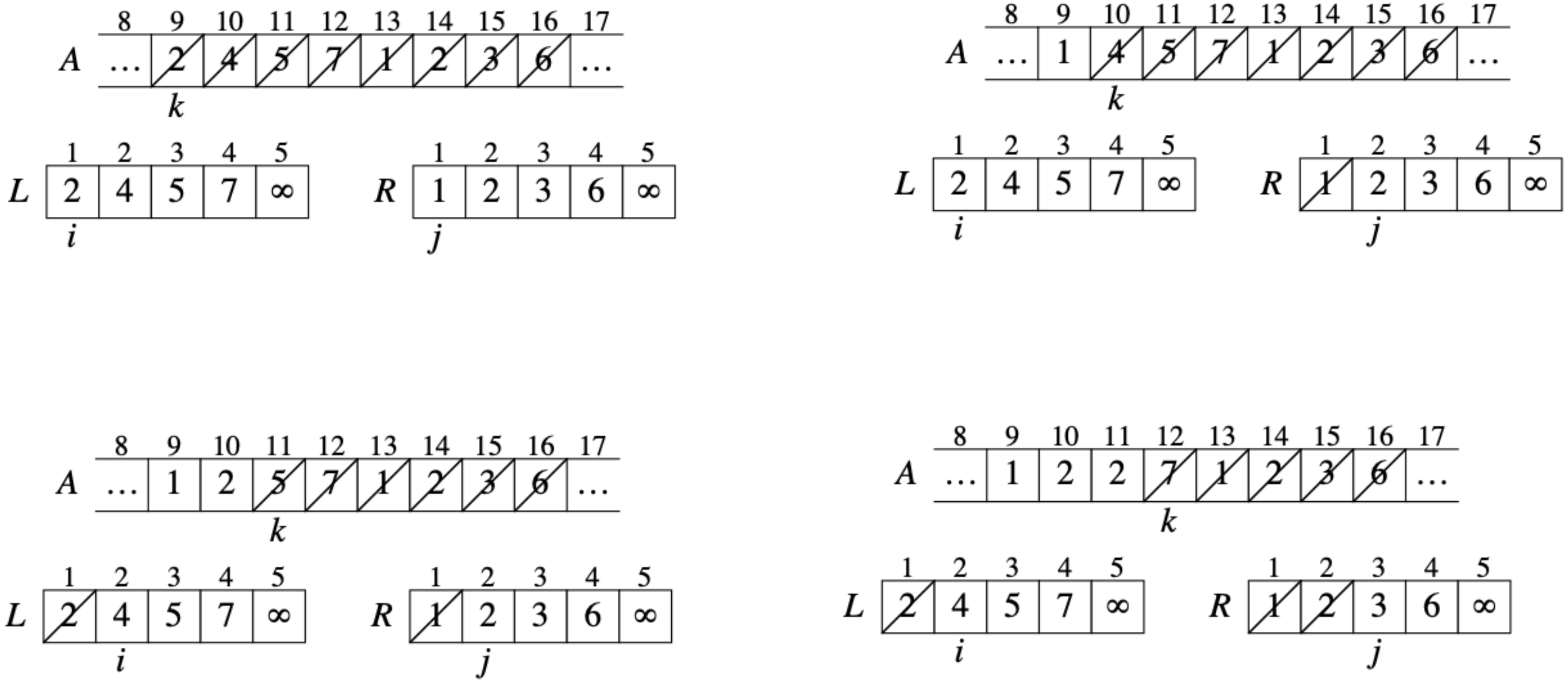


Merge Idea



MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```



Analysis of Runtime

$MergeSort$ consists of

two recursive calls to instances of roughly half the size

linear amount of additional work to $Merge$ the results

$T(n)$ = worst-case runtime of $MergeSort$ on inputs of length n

$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

$T(0), T(1), T(2) = O(1)$

This solves to $T(n) = O(n \log n)$ (same as $MaxSubarray$)

Closest Pair of Points (CLRS 33.4)

Input: $X[1..n], Y[1..n]$ – two arrays of n real numbers each specifying n points in Euclidean $2D$ space. Point i has coordinates $(X[i], Y[i])$

Output: δ – distance between closest pair of distinct points

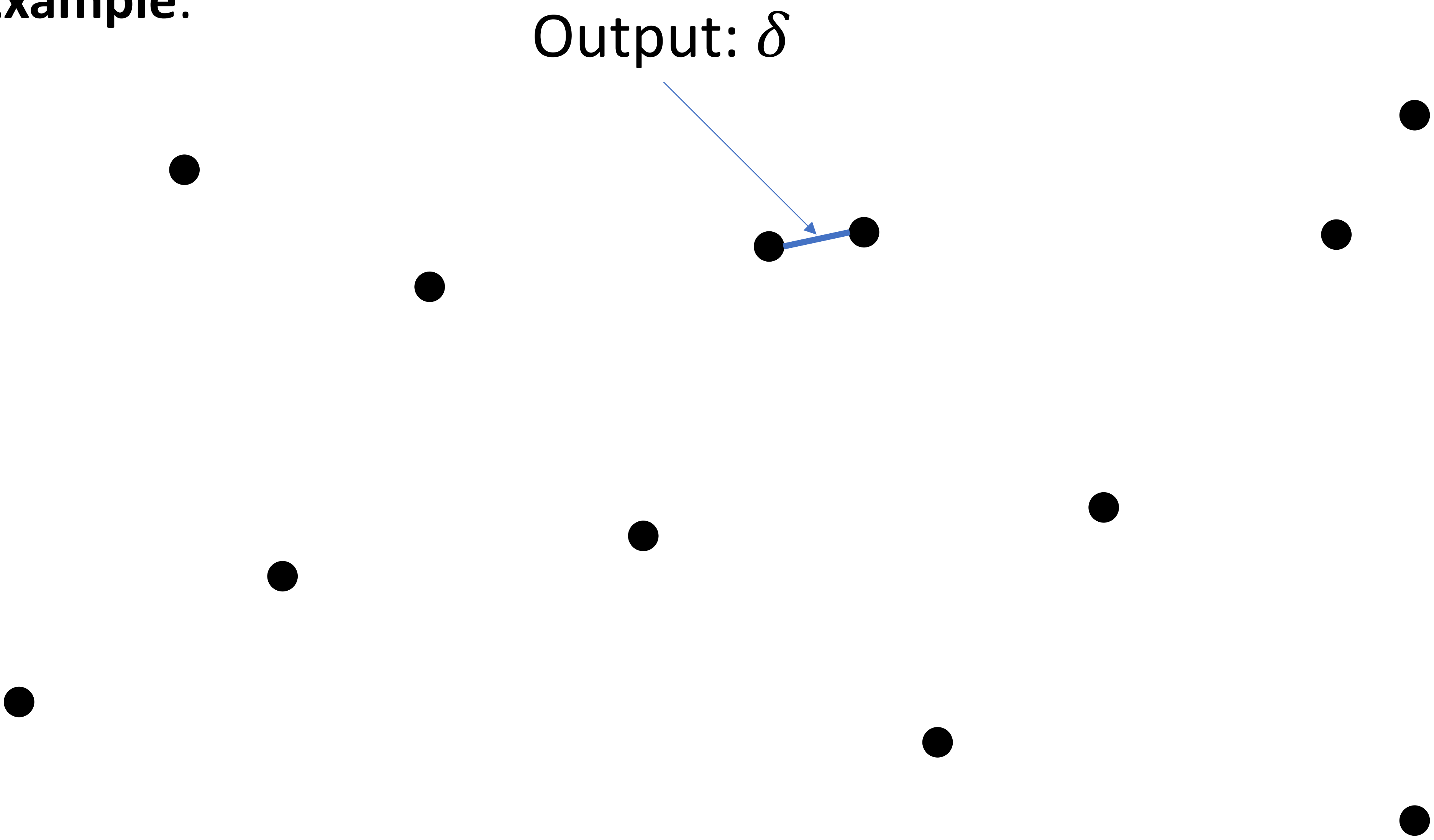
Note: closest is with respect to Euclidean distance

$$d\big((x_1, y_1), (x_2, y_2)\big) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Observe: $d\big((x_1, y_1), (x_2, y_2)\big) \geq |x_1 - x_2|, |y_1 - y_2|$

For simplicity assume that all X- and all Y-coordinates are different

Example:



Simple Algorithm

For each pair of points i and j compute

$$d\big((X[i], Y[i]), (X[j], Y[j])\big)$$

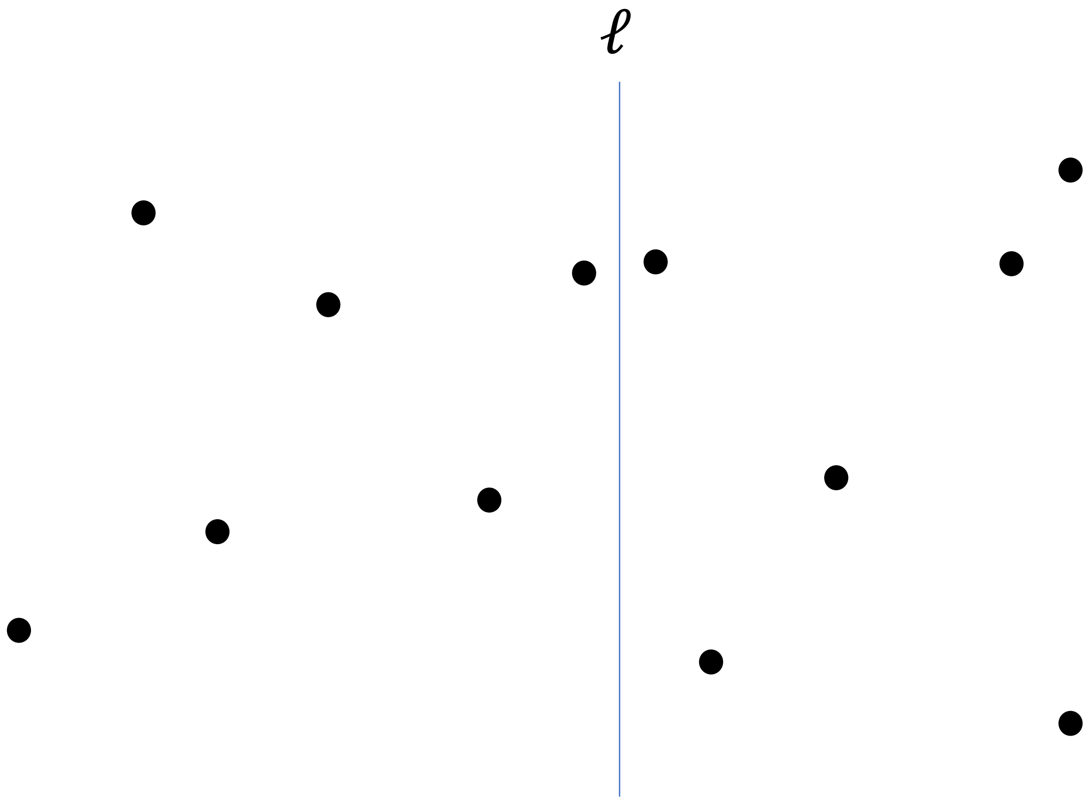
Keep track of the minimum value

Number of distinct pairs i, j is $\binom{n}{2} = \Theta(n^2)$

Therefore, runtime of this simple algorithm is $\Theta(n^2)$

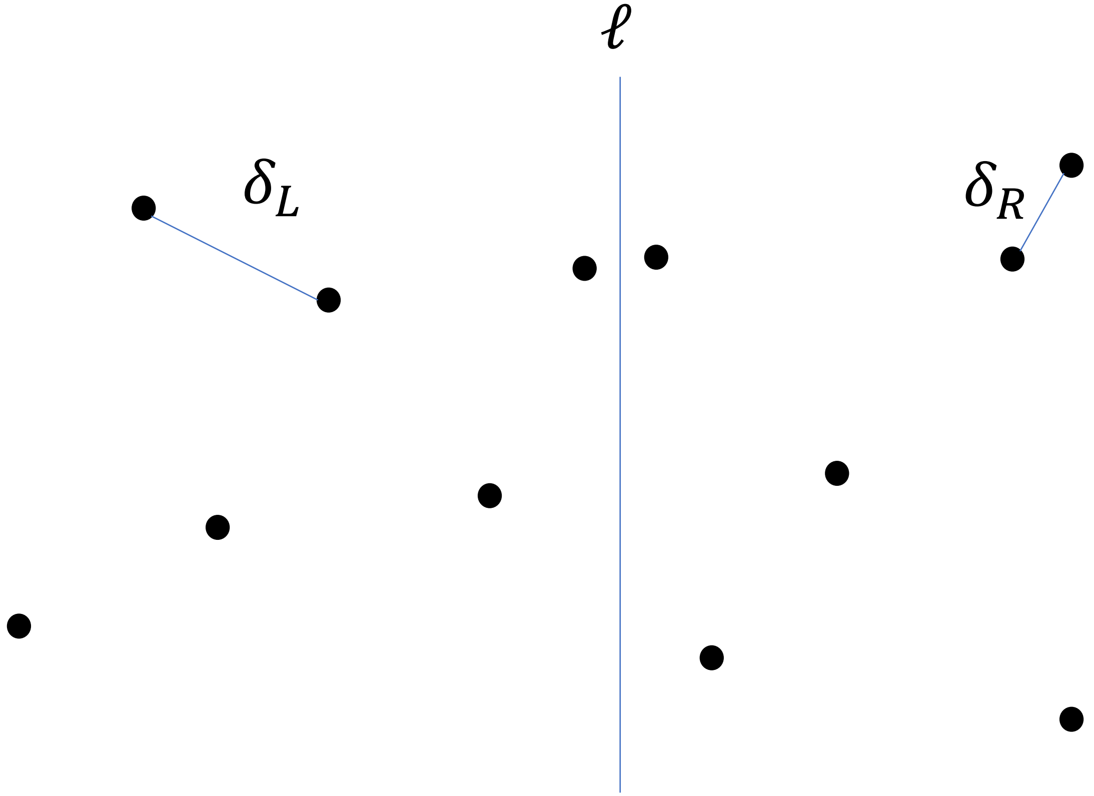
D&C Algorithm

Divide: find vertical line ℓ splitting points into two roughly equal sizes



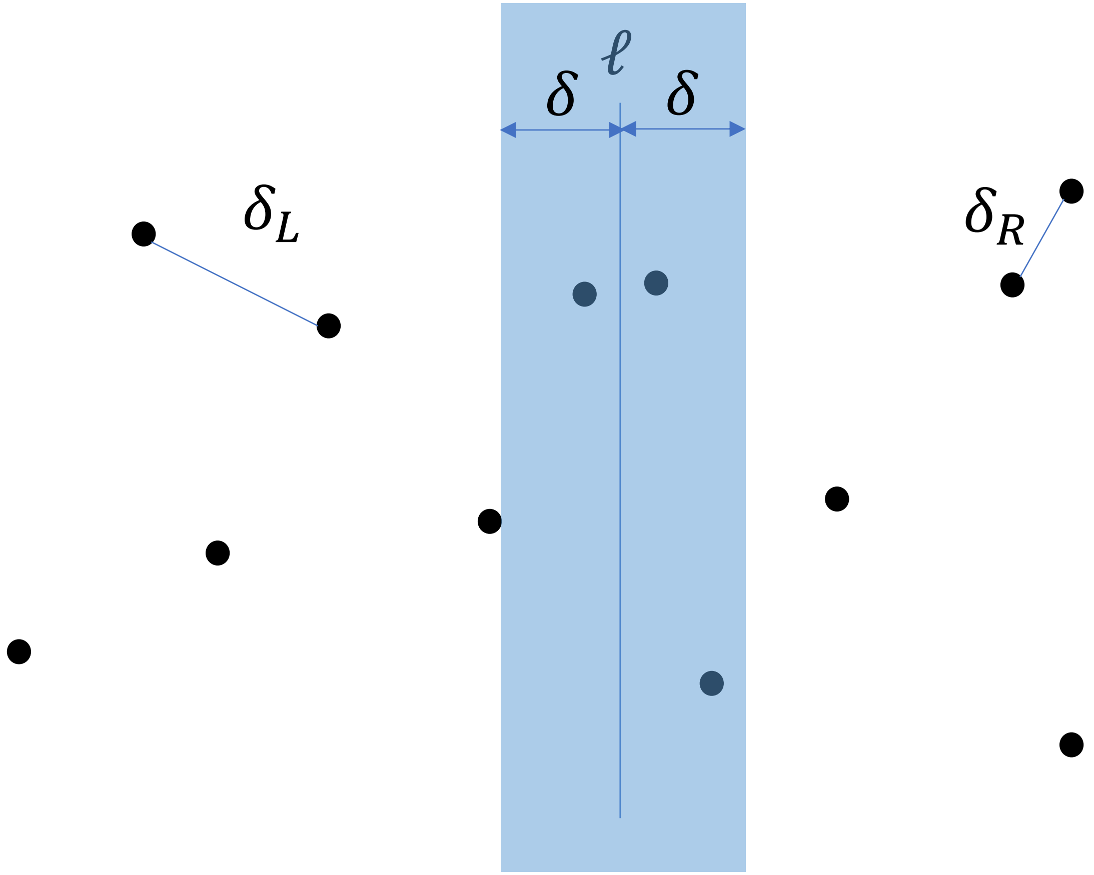
D&C Algorithm

Conquer: find $\delta_L(\delta_R)$ - the minimum distance between a pair of points on the left-hand side (right-hand side)



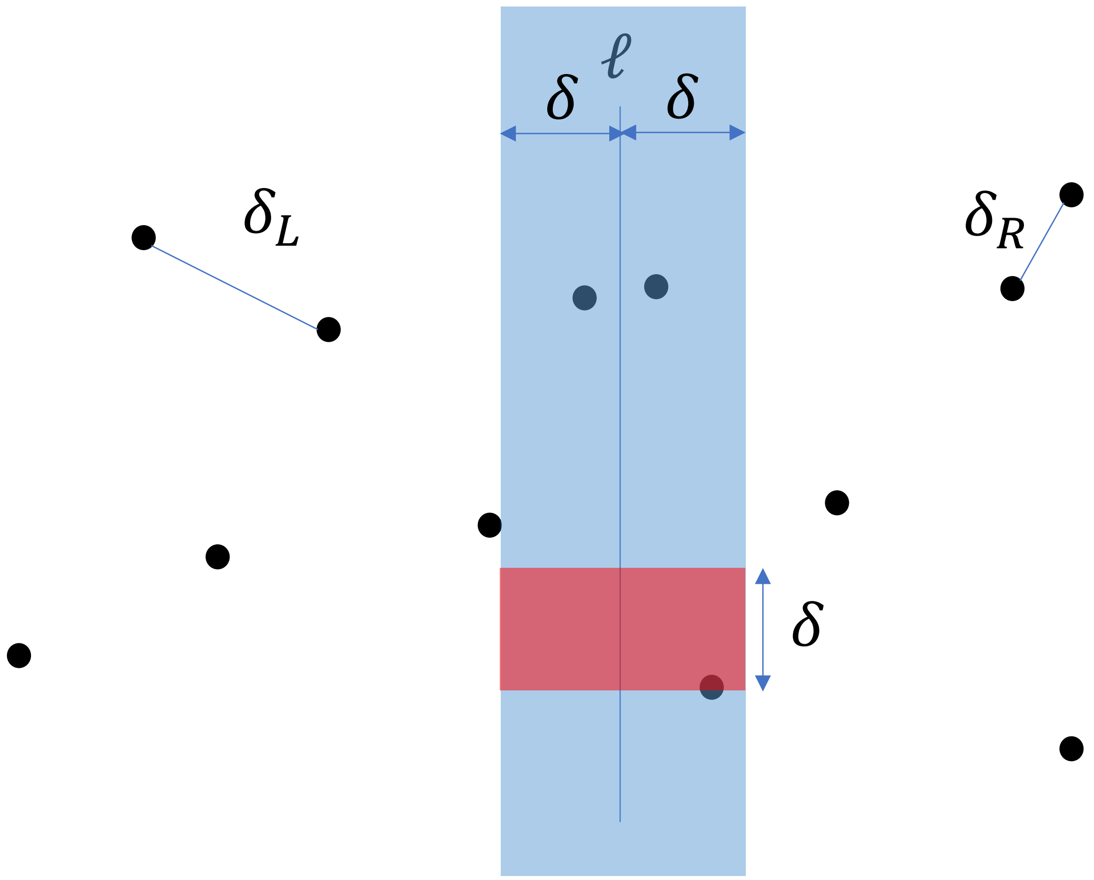
D&C Algorithm

Let $\delta = \min(\delta_L, \delta_R)$ and consider vertical strip of width 2δ centered around ℓ



D&C Algorithm

Sort points within the strip by their Y-coordinates
For each point p in the strip consider the next 7 points in the sorted order
Maintain the minimum distance δ'
Return $\min(\delta, \delta')$



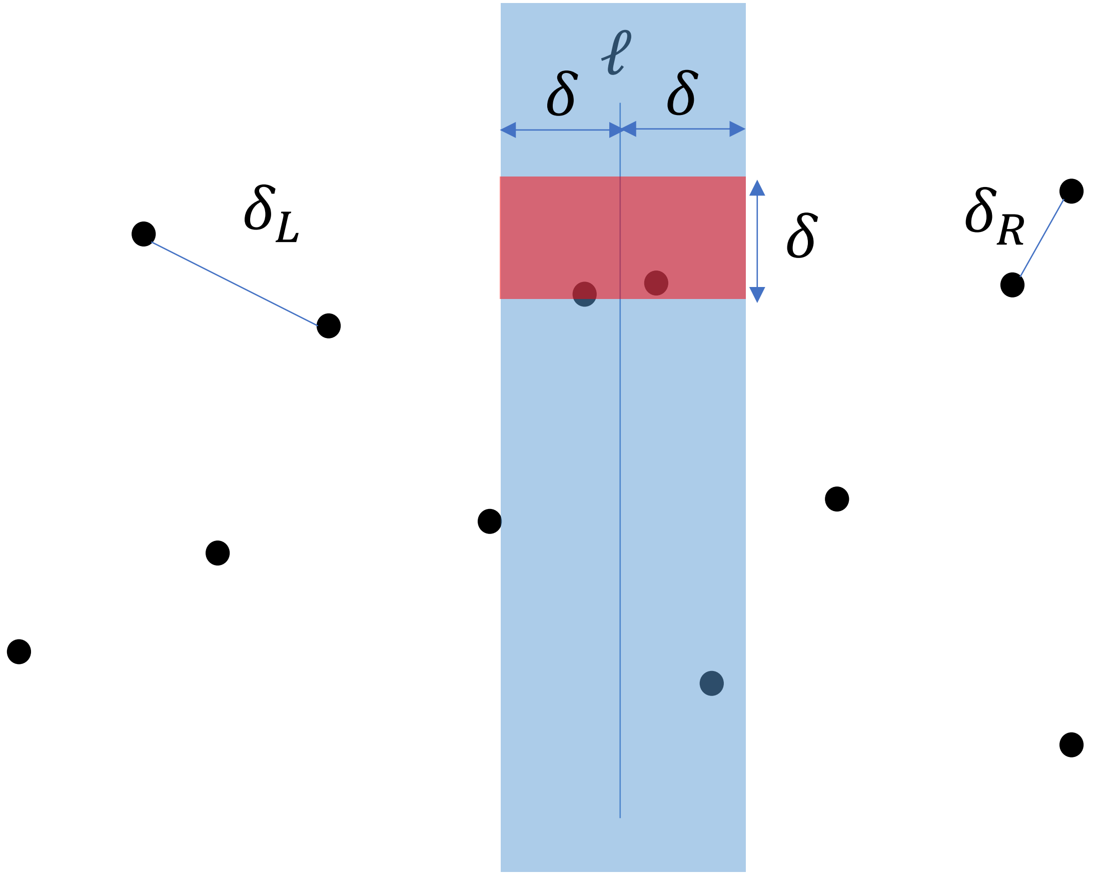
D&C Algorithm

Sort points within the strip by their Y-coordinates

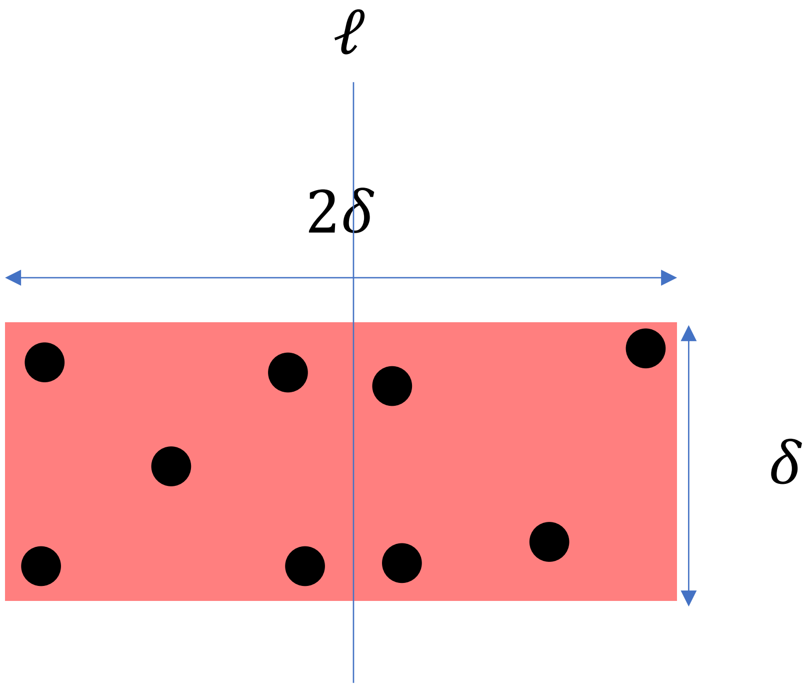
For each point p in the strip consider the next 7 points in the sorted order

Maintain the minimum distance δ'

Return $\min(\delta, \delta')$



Last step

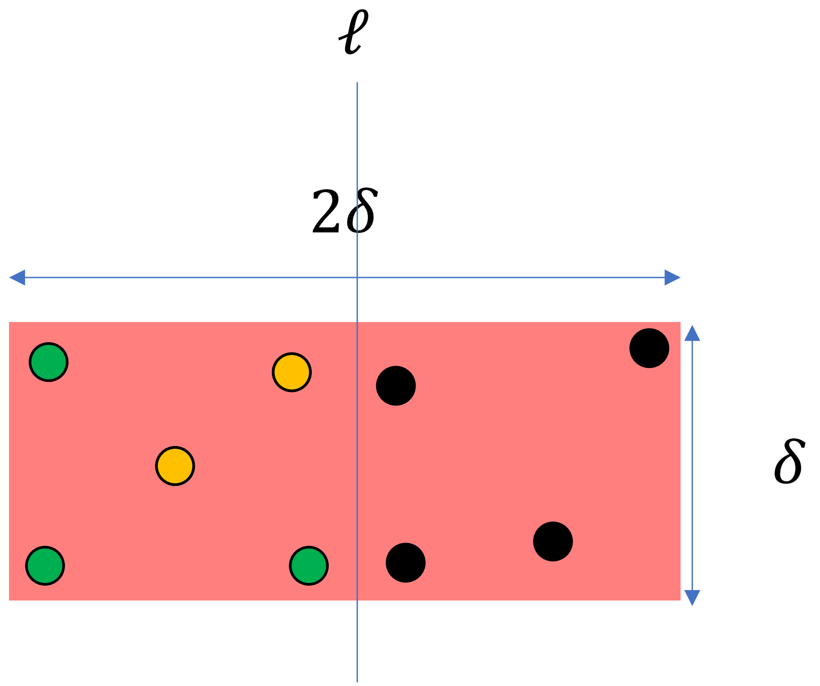


Why can there be no more than 8 points in this rectangle?

Suppose there are 9 points (for contradiction)

Then at least 5 points would fall either to the left of ℓ or to the right of ℓ

Last step



Why can there be no more than 8 points in this rectangle?

Suppose there are 9 points (for contradiction)

Then at least 5 points would fall either to the left of ℓ or to the right of ℓ

5 points within $\delta \times \delta$ square contain at least 2 points at distance $\leq \frac{\delta}{\sqrt{2}} < \delta$

They should have been detected by recursive call on the left. Contradiction.

$ClosestPair(X, Y)$

$\boldsymbol{if}\ n \leq 3$

run simple algorithm

sort points by their X-coordinates (in-place)

$q \leftarrow \lfloor \frac{n}{2} \rfloor$

$\ell \leftarrow X[q]$ // record X-coordinate of line ℓ

$\delta_L \leftarrow ClosestPair(X[1..q], Y[1..q])$

$\delta_R \leftarrow ClosestPair(X[q+1..n], Y[q+1, ..n])$

$\delta \leftarrow \min(\delta_L, \delta_R)$

resort points by their Y-coordinates (in-place)

$X' \leftarrow \emptyset, Y' \leftarrow \emptyset$

$$
\begin{aligned}
&\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ n \\
&\quad \boldsymbol{if}\ |X[i] - \ell| \le \delta \\
&\qquad X'.push(X[i]) \\
&\qquad Y'.push(Y[i]) \\
&\delta' \leftarrow \infty \\
&\boldsymbol{for}\ i = 1\ \boldsymbol{to}\ X'.size() \\
&\quad \boldsymbol{for}\ j = i + 1\ \boldsymbol{to}\ \min(i + 8, X'.size()\ ) \\
&\qquad \delta' \leftarrow \min\Big(\delta', d\big((X'[i], Y'[i]), (X'[j], Y'[j])\big)\Big) \\
&\boldsymbol{return}\ \min(\delta, \delta')
\end{aligned}
$$

Runtime analysis

Let $T(n)$ denote the worst-case runtime on inputs of length n

In $ClosestPair()$ we:

make 2 recursive calls on instances of size roughly $\frac{n}{2}$

sort points twice – can be done in time $O(n \log n)$ ($MergeSort$)

combine the solution by scanning array X' - $O(n)$

We get recurrence:

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n \log n) \text{ and } T(0), T(1), T(2), T(3) = O(1)$$

Solves to $T(n) = O(n \log^2 n)$

Notes on $ClosestPair$

Possible to improve running time to $O(n \log n)$

sort the points once at the beginning

pass sorted points to recursive calls

Possible to drop the assumption of distinct values of X and Y

See CLRS for more details

You should now be able to…

- Explain the divide and conquer strategy
- Use D&C to approach new problems
- Write down pseudocode for D&C algorithms for Integer Multiplication, Maximum Subarray, Sorting, and Closest Pair problems
- Analyze the runtime of D&C solutions and argue their correctness
- Explain how D&C solutions compare to simple algorithms

Review Questions

- What is the runtime of the high-school algorithm for integer multiplication?
- What is the decomposition of integer multiplication and the main trick behind Karatsuba's algorithm?
- What is the runtime of Karatsuba's algorithm? What is the recursion giving rise to this runtime?

Review Questions

- Name 4 sorting algorithms and their runtimes.

- Use recursion tree technique to argue that $T(n) = 2T\left(\frac{n}{2}\right) + O(n), T(1) = O(1)$ solves to $T(n) = O(n \log n)$.

- Write down pseudocode for $MaxCrossingSubarray$ function.

- Explain **divide**, **conquer** and **combine** steps of $ClosestPair$.