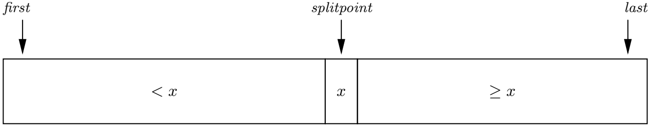
# Lecture 5
# Av. case analysis of quick sort, divide and conquer, mergesort

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

# Pseudocode for Quicksort

```
def quickSort(A,first,last):
    if first < last:
        splitpoint = split(A,first,last)
        quickSort(A,first,splitpoint-1)
        quickSort(A,splitpoint+1,last)
```



# The split step

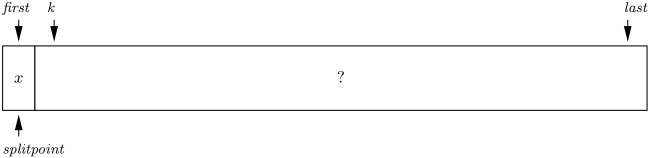
```
def split(A,first,last):
    splitpoint = first
    x = A[first]
    for k = first+1 to last do:
        if A[k] < x:
            A[splitpoint+1] ↔ A[k]
            splitpoint = splitpoint + 1
    A[first] ↔ A[splitpoint]
    return splitpoint
```

Loop invariants:

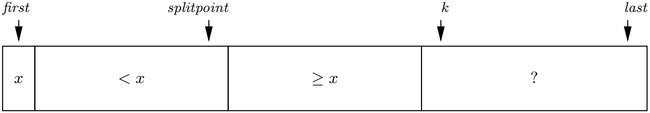
- `A[first+1..splitpoint]` contains keys $< x$.
- `A[splitpoint+1..k-1]` contains keys $\geq x$.
- `A[k..last]` contains unprocessed keys.

# The split step

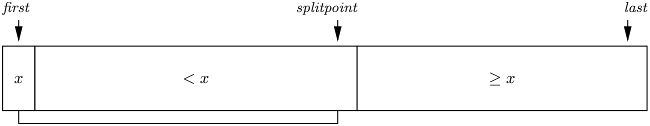
At start:



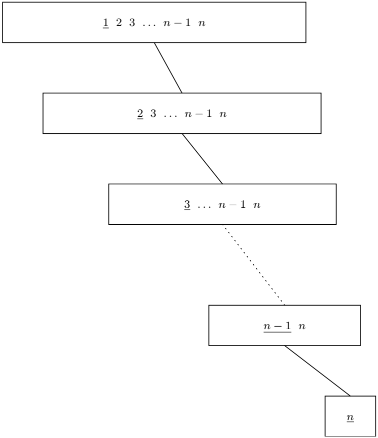

In middle:




At end:

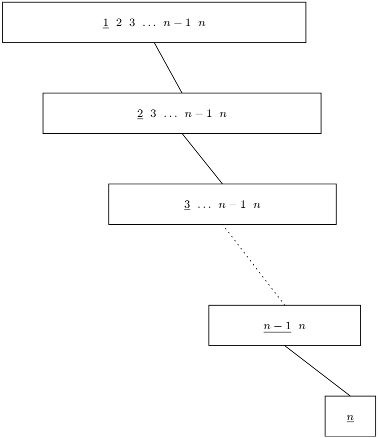


# A bad case case for Quicksort: $1, 2, 3, \ldots, n-1, n$




$\binom{n}{2}$ comparisons required. So the worst-case running time for Quicksort is $\Theta(n^2)$.

# A bad case case for Quicksort: $1, 2, 3, \ldots, n-1, n$




$\binom{n}{2}$ comparisons required. So the worst-case running time for Quicksort is $\Theta(n^2)$. But what about the average case ...?

# Average-case analysis of Quicksort:

Our approach:

# Average-case analysis of Quicksort:

Our approach:

1. Use the binary tree of sorted lists

# Average-case analysis of Quicksort:

Our approach:

1. Use the binary tree of sorted lists
2. Number the items in sorted order

# Average-case analysis of Quicksort:

Our approach:

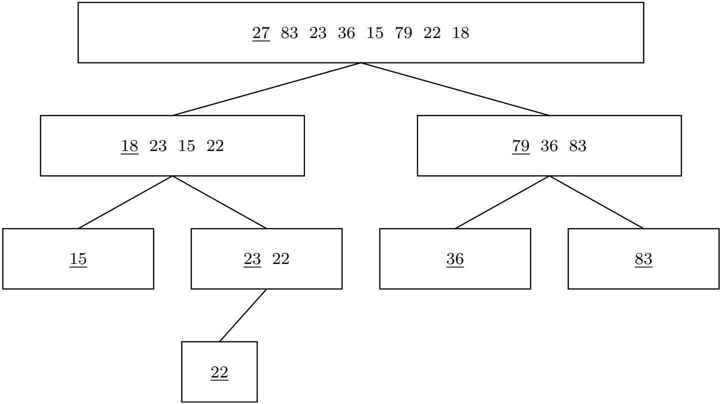
1. Use the binary tree of sorted lists
2. Number the items in sorted order
3. Calculate the probability that two items get compared

# Average-case analysis of Quicksort:

Our approach:

1. Use the binary tree of sorted lists
2. Number the items in sorted order
3. Calculate the probability that two items get compared
4. Use this to compute the expected number of comparisons performed by Quicksort.

# Average-case analysis of Quicksort:




Sorted order: 15 18 22 23 27 36 79 83

# Average-case analysis of Quicksort

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
  - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
    - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.
    - If $S_i$ is chosen first, it is compared against all the other keys in the set in the split step (including $S_j$).

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
  - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.
  - If $S_i$ is chosen first, it is compared against all the other keys in the set in the split step (including $S_j$).
  - Similar if $S_j$ is chosen first.

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
  - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.
  - If $S_i$ is chosen first, it is compared against all the other keys in the set in the split step (including $S_j$).
  - Similar if $S_j$ is chosen first.

Examples:

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
    - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.
    - If $S_i$ is chosen first, it is compared against all the other keys in the set in the split step (including $S_j$).
    - Similar if $S_j$ is chosen first.

Examples:

- 23 and 22 (both statements true)

# Average-case analysis of Quicksort

- Number the keys in sorted order: $S_1 < S_2 < \cdots < S_n$.
- Fact about comparisons: During the run of Quicksort, two keys $S_i$ and $S_j$ get compared if and only if the first key from the set of keys $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot is either $S_i$ or $S_j$.
  - If some key $S_k$ is chosen first with $S_i < S_k < S_j$, then $S_i$ goes in the left half, $S_j$ goes in the right half, and $S_i$ and $S_j$ never get compared.
  - If $S_i$ is chosen first, it is compared against all the other keys in the set in the split step (including $S_j$).
  - Similar if $S_j$ is chosen first.

Examples:

- 23 and 22 (both statements true)
- 36 and 83 (both statements false)

# Average-case analysis of Quicksort

# Average-case analysis of Quicksort

Assume:

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct
- All permutations are equally likely

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct
- All permutations are equally likely
- The keys in sorted order are $S_1 < S_2 < \cdots < S_n$.

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct
- All permutations are equally likely
- The keys in sorted order are $S_1 < S_2 < \cdots < S_n$.

Let $P_{i,j} =$ The probability that keys $S_i$ and $S_j$ are compared with each other during the invocation of quicksort

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct
- All permutations are equally likely
- The keys in sorted order are $S_1 < S_2 < \cdots < S_n$.

Let $P_{i,j}$ = The probability that keys $S_i$ and $S_j$ are compared with each other during the invocation of quicksort

Then by Fact about comparisons on previous slide:

$P_{i,j}$ = The probability that the first key from $\{S_i, S_{i+1}, \ldots, S_j\}$ to be chosen as a pivot value is either $S_i$ or $S_j$

# Average-case analysis of Quicksort

Assume:

- All $n$ keys are distinct
- All permutations are equally likely
- The keys in sorted order are $S_1 < S_2 < \cdots < S_n$.

Let $P_{i,j} =$ The probability that keys $S_i$ and $S_j$ are compared with each other during the invocation of quicksort

Then by Fact about comparisons on previous slide:

$$
\begin{aligned}
P_{i,j} &= \text{The probability that the first key from } \{S_i, S_{i+1}, \ldots, S_j\} \text{ to be chosen as a pivot value is either } S_i \text{ or } S_j \\
&= \frac{2}{j-i+1}
\end{aligned}
$$

# Average-case analysis of Quicksort

Define indicator random variables $\{X_{i,j} : 1 \leq i < j \leq n\}$

$$X_{i,j} = \begin{cases} 1 & \text{if keys } S_i \text{ and } S_j \text{ get compared} \\ 0 & \text{if keys } S_i \text{ and } S_j \text{ do } \underline{\text{not}} \text{ get compared} \end{cases}$$

# Average-case analysis of Quicksort

Define indicator random variables $\{X_{i,j} : 1 \leq i < j \leq n\}$

$$X_{i,j} = \begin{cases} 1 & \text{if keys } S_i \text{ and } S_j \text{ get compared} \\ 0 & \text{if keys } S_i \text{ and } S_j \text{ do } \underline{\text{not}} \text{ get compared} \end{cases}$$

1. The total number of comparisons is:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$$

# Average-case analysis of Quicksort

Define indicator random variables $\{X_{i,j} : 1 \leq i < j \leq n\}$

$$X_{i,j} = \begin{cases} 1 & \text{if keys } S_i \text{ and } S_j \text{ get compared} \\ 0 & \text{if keys } S_i \text{ and } S_j \text{ do } \underline{\text{not}} \text{ get compared} \end{cases}$$

1. The total number of comparisons is:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$$

2. The expected (average) total number of comparisons is:

$$E\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}\right) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right)$$

# Average-case analysis of Quicksort

Define indicator random variables $\{X_{i,j} : 1 \leq i < j \leq n\}$

$$X_{i,j} = \begin{cases} 1 & \text{if keys } S_i \text{ and } S_j \text{ get compared} \\ 0 & \text{if keys } S_i \text{ and } S_j \text{ do } \underline{\text{not}} \text{ get compared} \end{cases}$$

1. The total number of comparisons is:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}$$

2. The expected (average) total number of comparisons is:

$$E\left(\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j}\right) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right)$$

3. The expected value of $X_{i,j}$ is:

$$E(X_{i,j}) = P_{i,j} = \frac{2}{j-i+1}$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right)$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} E\left(X_{i,j}\right) \quad = \quad \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n}\sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j-i+1)\end{aligned}$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k}
\end{aligned}
$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \qquad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k}
\end{aligned}
$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \qquad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \sum_{i=1}^{n} H_n
\end{aligned}
$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \sum_{i=1}^{n} H_n = 2nH_n
\end{aligned}$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \sum_{i=1}^{n} H_n = 2nH_n \in O(n \lg n).
\end{aligned}
$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j-i+1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \sum_{i=1}^{n} H_n = 2nH_n \in O(n \lg n).
\end{aligned}
$$

# Average-case analysis of Quicksort

Hence the expected number of comparisons is

$$
\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} E\left(X_{i,j}\right) &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad (k = j - i + 1) \\
&< \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k} \\
&= 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2 \sum_{i=1}^{n} H_n = 2nH_n \in O(n \lg n).
\end{aligned}
$$

So the average time for Quicksort is $O(n \lg n)$.

# Divide and Conquer

Divide and conquer paradigm

1. Split problem into subproblem(s)
2. Solve each subproblem (usually via recursive call)
3. Combine solution of subproblem(s) into solution of original problem

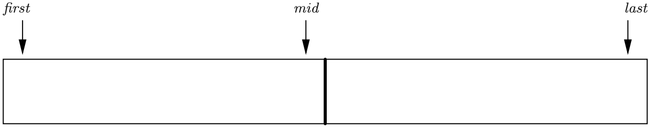
We will discuss two sorting algorithms based on this paradigm:

- Quicksort (done)
- Mergesort

# MergeSort

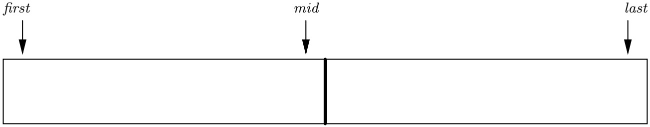
# MergeSort

- Split array into two equal subarrays



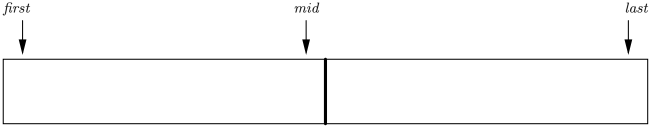
# MergeSort

- Split array into two equal subarrays
- Sort both subarrays (recursively)



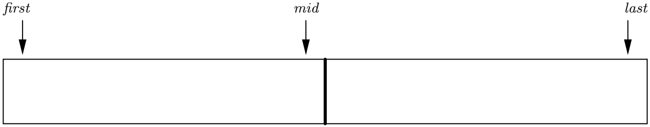
# MergeSort

- Split array into two equal subarrays
- Sort both subarrays (recursively)
- Merge two sorted subarrays



# MergeSort

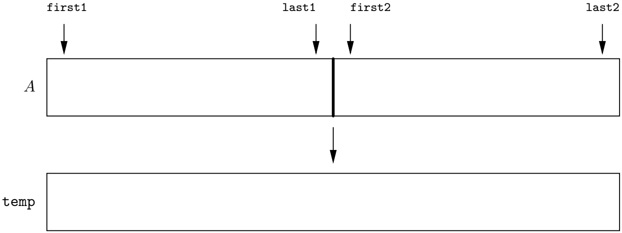
- Split array into two equal subarrays
- Sort both subarrays (recursively)
- Merge two sorted subarrays



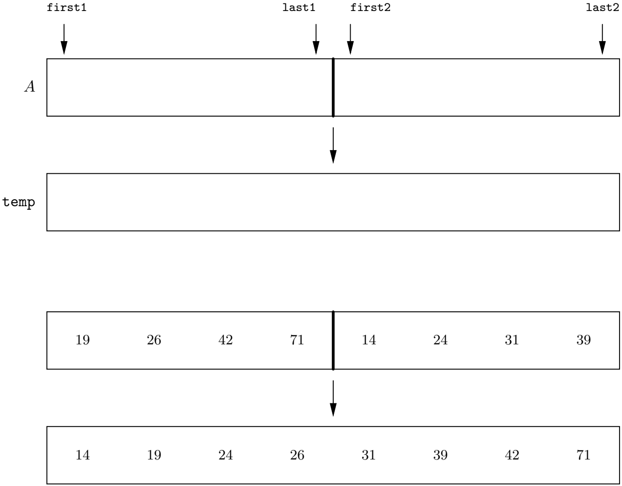

```
def mergeSort(A,first,last):
    if first < last:
        mid = ⌊(first + last)/2⌋
        mergeSort(A,first,mid)
        mergeSort(A,mid+1,last)
        merge(A,first,mid,mid+1,last)
```

# The merge step

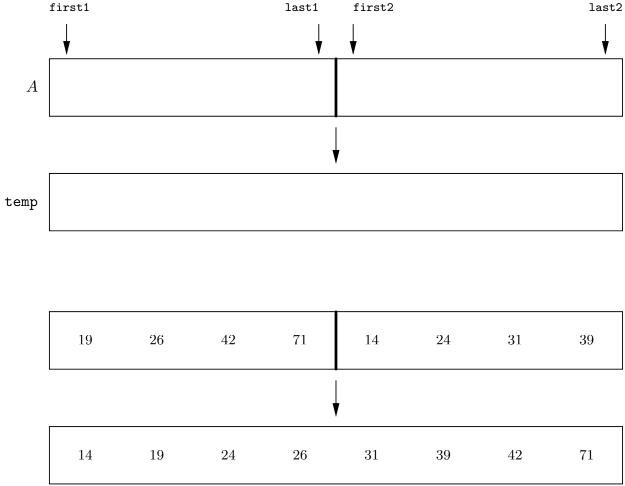
# The merge step



# The merge step



# The merge step




Merging two lists of total size $n$ requires at most $n-1$ comparisons.

# Code for the merge step

```
def merge(A,first1,last1,first2,last2):
    index1 = first1; index2 = first2; tempIndex = 0
   // Merge into temp array until one input array is exhausted
    while (index1 <= last1) and (index2 <= last2)
        if A[index1] <= A[index2]:
            temp[tempIndex++] = A[index1++]
        else:
            temp[tempIndex++] = A[index2++]
   // Copy appropriate trailer portion
    while (index1 <= last1):  temp[tempIndex++] = A[index1++]
    while (index2 <= last2):  temp[tempIndex++] = A[index2++]
   // Copy temp array back to A array
    tempIndex = 0; index = first1
    while (index <= last2):  A[index++] = temp[tempIndex++]
```

# Analysis of Mergesort

# Analysis of Mergesort

$T(n) =$ number of comparisons required to sort $n$ items in the worst case

# Analysis of Mergesort

$T(n) =$ number of comparisons required to sort $n$ items in the worst case

$$T(n) = \begin{cases} T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n - 1, & n > 1 \\ 0, & n = 1 \end{cases}$$

# Analysis of Mergesort

$T(n) =$ number of comparisons required to sort $n$ items in the worst case

$$T(n) = \begin{cases} T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n - 1, & n > 1 \\ 0, & n = 1 \end{cases}$$

The asymptotic solution of this recurrence equation is

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

# Analysis of Mergesort

$T(n) =$ number of comparisons required to sort $n$ items in the worst case

$$T(n) = \begin{cases} T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n - 1, & n > 1 \\ 0, & n = 1 \end{cases}$$

The asymptotic solution of this recurrence equation is

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

The exact solution of this recurrence equation is

$$T(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$$

# Geometrical Application: Counting line intersections

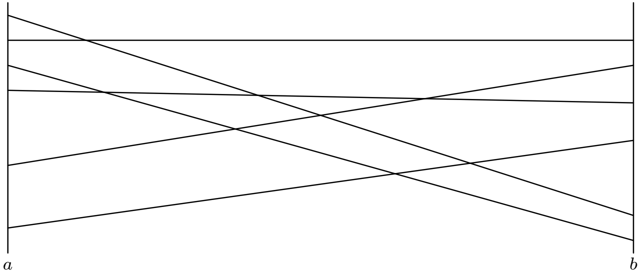
# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).

# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).

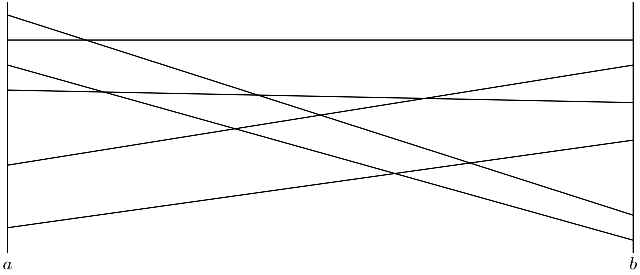
Example: $n = 6$



# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).
- Problem: Count/report all pairs of lines that intersect between the two vertical lines $x = a$ and $x = b$.

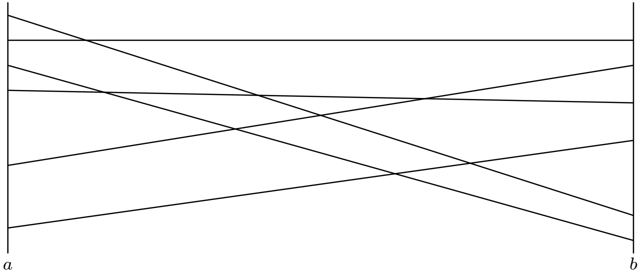
Example: $n = 6$



# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).
- Problem: Count/report all pairs of lines that intersect between the two vertical lines $x = a$ and $x = b$.

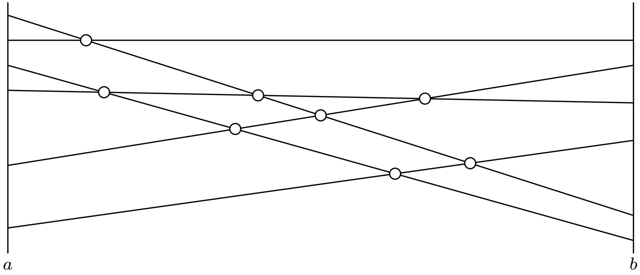
Example: $n = 6$ 8 intersections



# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).
- Problem: Count/report all pairs of lines that intersect between the two vertical lines $x = a$ and $x = b$.

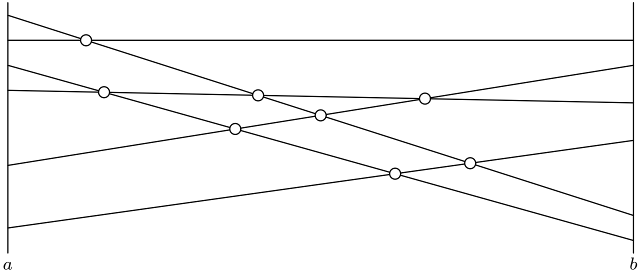
Example: $n = 6$ 8 intersections



# Geometrical Application: Counting line intersections

- Input: $n$ lines in the plane, none of which are vertical; two vertical lines $x = a$ and $x = b$ (with $a < b$).
- Problem: Count/report all pairs of lines that intersect between the two vertical lines $x = a$ and $x = b$.

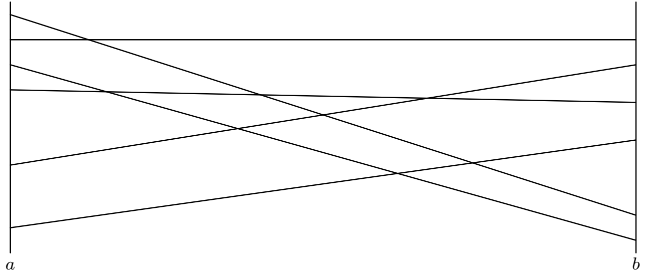
Example: $n = 6$ 8 intersections




Checking every pair of lines takes $\Theta(n^2)$ time. We can do better.

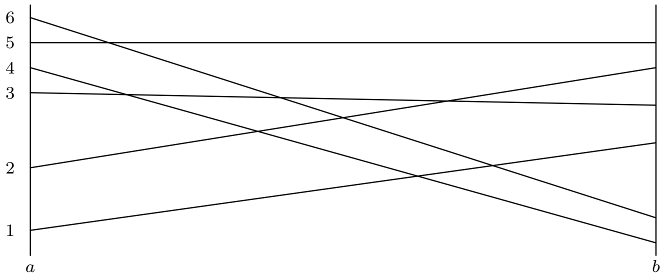
# Geometrical Application: Counting line intersections

# Geometrical Application: Counting line intersections



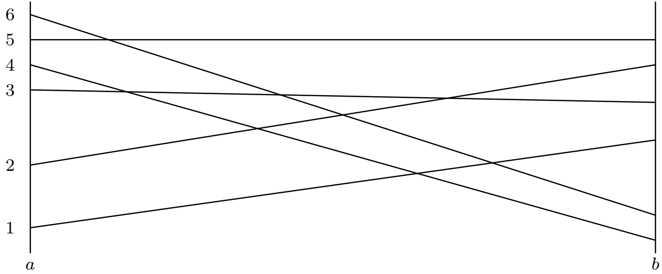
# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order.



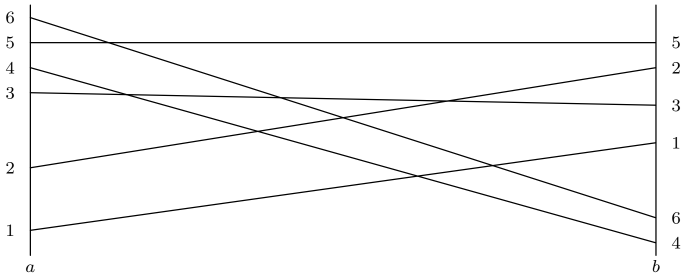
# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order. [$O(n \log n)$ time]



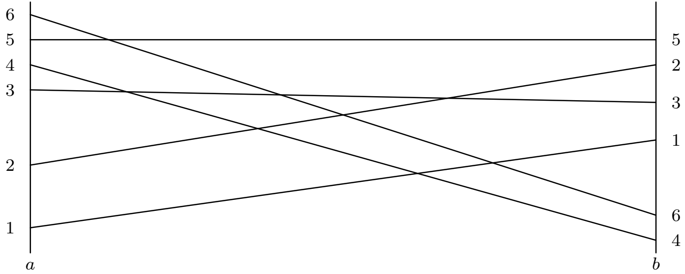
# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order. [$O(n \log n)$ time]
2. Produce the sequence of line numbers sorted according to the $y$-coordinate of their intersection with the line $x = b$



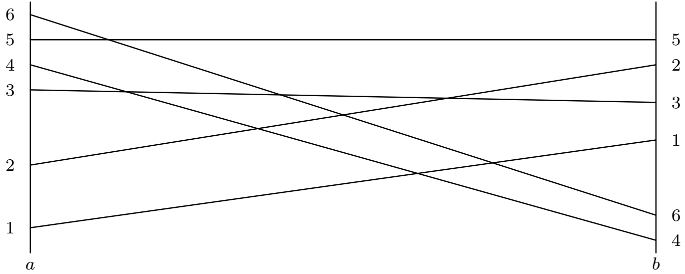
# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order. [$O(n \log n)$ time]
2. Produce the sequence of line numbers sorted according to the $y$-coordinate of their intersection with the line $x = b$ [$O(n \log n)$ time]



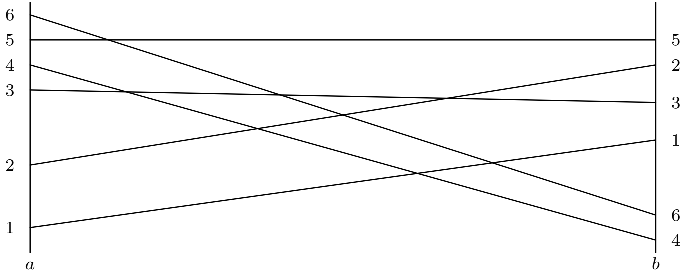
# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order. [$O(n \log n)$ time]
2. Produce the sequence of line numbers sorted according to the $y$-coordinate of their intersection with the line $x = b$ [$O(n \log n)$ time]
3. Count/report inversions in the sequence produced in step 2.



# Geometrical Application: Counting line intersections

1. Sort the lines according to the $y$-coordinate of their intersection with the line $x = a$. Number the lines in sorted order. [$O(n \log n)$ time]
2. Produce the sequence of line numbers sorted according to the $y$-coordinate of their intersection with the line $x = b$ [$O(n \log n)$ time]
3. Count/report inversions in the sequence produced in step 2.




So the problem reduces to counting/reporting inversions.

# Counting Inversions: An Application of Mergesort

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$$

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$$

In a list of size $n$, there can be as many as $\binom{n}{2}$ inversions.

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$$

In a list of size $n$, there can be as many as $\binom{n}{2}$ inversions.

Problem: Given a list, compute the number of inversions.

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$$

In a list of size $n$, there can be as many as $\binom{n}{2}$ inversions.

Problem: Given a list, compute the number of inversions.

Brute force solution: Check each pair $i, j$ with $i < j$ to see if $L[i] > L[j]$.

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$

In a list of size $n$, there can be as many as $\binom{n}{2}$ inversions.

Problem: Given a list, compute the number of inversions.

Brute force solution: Check each pair $i, j$ with $i < j$ to see if $L[i] > L[j]$. This gives a $\Theta(n^2)$ algorithm.

# Counting Inversions: An Application of Mergesort

An inversion in a sequence or list is a pair of items such that the larger one precedes the smaller one.

Example: The list [18, 29, 12, 15, 32, 10] has 9 inversions:

$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10), (12, 10), (15, 10), (32, 10)$

In a list of size $n$, there can be as many as $\binom{n}{2}$ inversions.

Problem: Given a list, compute the number of inversions.

Brute force solution: Check each pair $i, j$ with $i < j$ to see if $L[i] > L[j]$. This gives a $\Theta(n^2)$ algorithm. We can do better.

# Inversion Counting

# Inversion Counting

Sorting is the process of removing inversions. So to count inversions:

# Inversion Counting

Sorting is the process of removing inversions. So to count inversions:

- Run a sorting algorithm

# Inversion Counting

Sorting is the process of removing inversions. So to count inversions:

- Run a sorting algorithm
- Every time data is rearranged, keep track of how many inversions are being removed.

# Inversion Counting

Sorting is the process of removing inversions. So to count inversions:

- Run a sorting algorithm
- Every time data is rearranged, keep track of how many inversions are being removed.

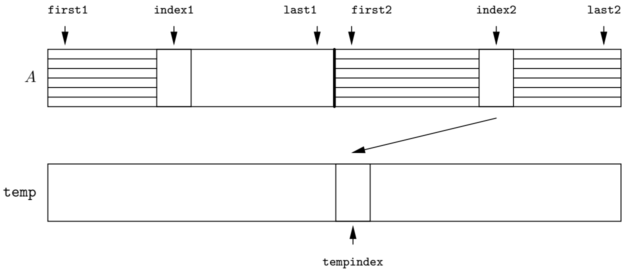
In principle, we can use any sorting algorithm to count inversions. Mergesort works particularly nicely.

# Inversion Counting with MergeSort

In Mergesort, the only time we rearrange data is during the merge step.

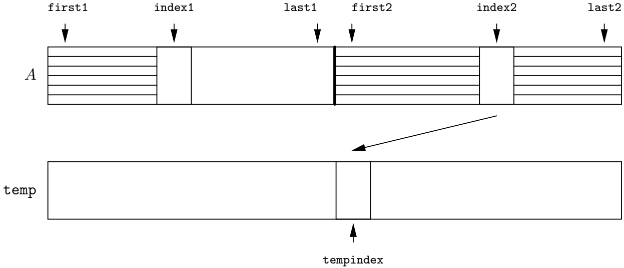
# Inversion Counting with MergeSort

In Mergesort, the only time we rearrange data is during the merge step.



# Inversion Counting with MergeSort

In Mergesort, the only time we rearrange data is during the merge step.

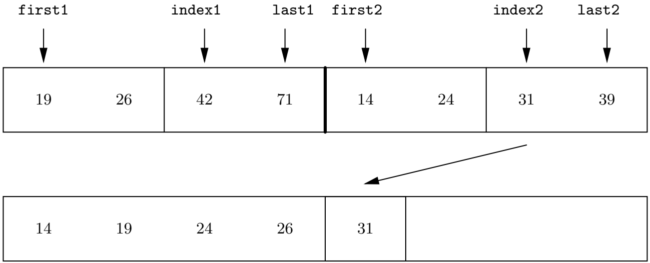



The number of inversions removed is:

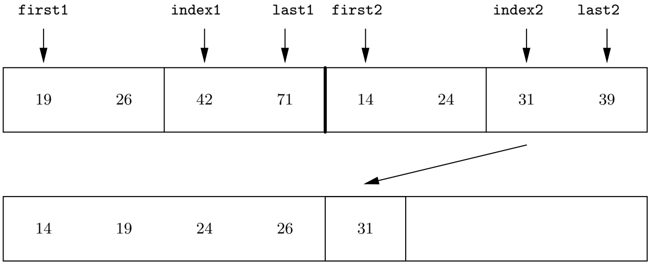
$$\texttt{last1} - \texttt{index1} + 1$$

# Example

# Example


first1
index1
last1
first2
index2
last2
19
26
42
71
14
24
31
39
14
19
24
26
31

# Example




2 inversions removed: $(42, 31)$ and $(71, 31)$

# Pseudocode for the merge step with inversion counting

# Pseudocode for the merge step with inversion counting

```
def merge(A,first1,last1,first2,last2):
    index1 = first1; index2 = first2; tempIndex = 0
    invCount = 0
  // Merge into temp array until one input array is exhausted
    while (index1 <= last1) and (index2 <= last2)
        if A[index1] <= A[index2]:
            temp[tempIndex++] = A[index1++]
        else:
            temp[tempIndex++] = A[index2++]
            invCount += last1 - index1 + 1;
  // Copy appropriate trailer portion
    while (index1 <= last1):  temp[tempIndex++] = A[index1++]
    while (index2 <= last2):  temp[tempIndex++] = A[index2++]
  // Copy temp array back to A array
    tempIndex = 0; index = first1
    while (index <= last2):  A[index++] = temp[tempIndex++]
    return invCount
```

# Pseudocode for MergeSort with inversion counting

# Pseudocode for MergeSort with inversion counting

```
def mergeSort(A,first,last):
    invCount = 0
    if first < last:
        mid = ⌊(first + last)/2⌋
        invCount += mergeSort(A,first,mid)
        invCount += mergeSort(A,mid+1,last)
        invCount += merge(A,first,mid,mid+1,last)
    return invCount
```

# Pseudocode for MergeSort with inversion counting

```
def mergeSort(A,first,last):
    invCount = 0
    if first < last:
        mid = ⌊(first + last)/2⌋
        invCount += mergeSort(A,first,mid)
        invCount += mergeSort(A,mid+1,last)
        invCount += merge(A,first,mid,mid+1,last)
    return invCount
```

Running time is the same as standard mergeSort: $O(n \log n)$

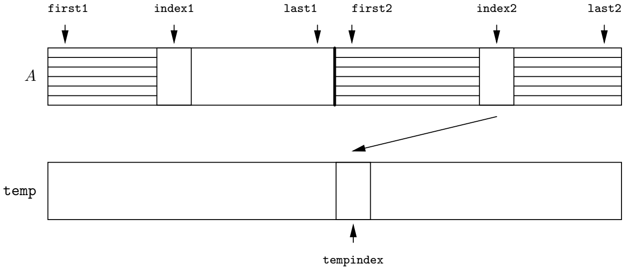
# Listing inversions

# Listing inversions

We have just seen that we can count inversions without increasing the asymptotic running time of Mergesort. Suppose we want to list inversions.

# Listing inversions

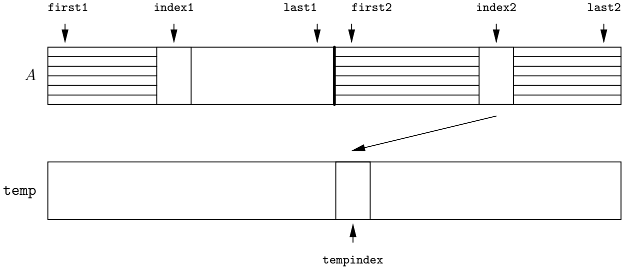
We have just seen that we can count inversions without increasing the asymptotic running time of Mergesort. Suppose we want to list inversions. When we remove inversions, we list all inversions removed:




$(A[\texttt{index1}], A[\texttt{index2}])$, $(A[\texttt{index1+1}], A[\texttt{index2}])$, $\ldots$, $(A[\texttt{last1}], A[\texttt{index2}])$.

# Listing inversions

We have just seen that we can count inversions without increasing the asymptotic running time of Mergesort. Suppose we want to list inversions. When we remove inversions, we list all inversions removed:




$(A[\texttt{index1}], A[\texttt{index2}])$, $(A[\texttt{index1+1}], A[\texttt{index2}])$, ..., $(A[\texttt{last1}], A[\texttt{index2}])$.

The extra work to do the reporting is proportional to the number of inversions reported.

# Inversion counting summary

# Inversion counting summary

Using a slight modification of Mergesort, we can ...

# Inversion counting summary

Using a slight modification of Mergesort, we can ...

- Count inversions in $O(n \log n)$ time.

# Inversion counting summary

Using a slight modification of Mergesort, we can ...

- Count inversions in $O(n \log n)$ time.
- Report inversions in $O(n \log n + k)$ time, where $k$ is the number of inversions.

# Inversion counting summary

Using a slight modification of Mergesort, we can ...

- Count inversions in $O(n \log n)$ time.
- Report inversions in $O(n \log n + k)$ time, where $k$ is the number of inversions.

The same results hold for the line-intersection counting problem.

# Inversion counting summary

Using a slight modification of Mergesort, we can . . .

- Count inversions in $O(n \log n)$ time.
- Report inversions in $O(n \log n + k)$ time, where $k$ is the number of inversions.

The same results hold for the line-intersection counting problem.

The reporting algorithm is an example of an output-sensitive algorithm. The performance of the algorithm depends on the size of the output as well as the size of the input.