DIVIDE AND CONQUER ALGORITHMS

Lecture 6, CMSC 142

Last Meeting

- Searching Algorithms

Design of Algorithms

- Brute-force/Exhaustive
- Divide and Conquer
- Dynamic Programming
- Greedy Approach

Design of Algorithms

- Brute-force/Exhaustive
- **Divide and Conquer**
- Dynamic Programming
- Greedy Approach

Divide and Conquer

Divide and Conquer

A class of algorithmic techniques that solves problems in 3 different steps:

Divide and Conquer

A class of algorithmic techniques that solves problems in 3 different steps:

- Divide

Divide and Conquer

A class of algorithmic techniques that solves problems in 3 different steps:

- Divide
- Conquer

Divide and Conquer

A class of algorithmic techniques that solves problems in 3 different steps:

- Divide
- Conquer
- Combine

3 steps

Divide

- Breaking the problem into subproblems that are themselves smaller instances of the same type of problem

Conquer

- Recursively solving the subproblems

3 steps

Combine

- Appropriately combining / merging the answers to form the original problem's answer
- At the tail end of recursion (base case), subproblems are so small that they can be solved outright or trivially.

Divide and Conquer

- Improves on the natural brute-force approach (naive, obvious, not sophisticated) for a given problem by reducing the running time to a lower polynomial

Examples

- Merge Sort
- Quick Sort
- Binary Search (no combine step)

Counting Inversions

Counting Inversions

Counting Inversions

Input: array A containing numbers in some arbitrary order

Counting Inversions

Input: array A containing numbers in some arbitrary order

Output: number of inversions, that is the number of pairs (i,j) of array indices with $i < j$ and $A[i] > A[j]$

Example

Array: [2, 4, 1, 3, 5]

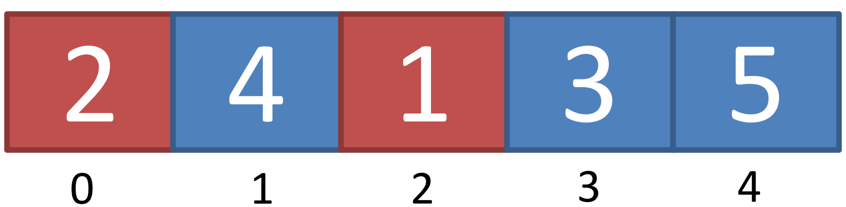
Output (2, 1), (4, 1), (4, 3)

2	4	1	3	5
0	1	2	3	4

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)

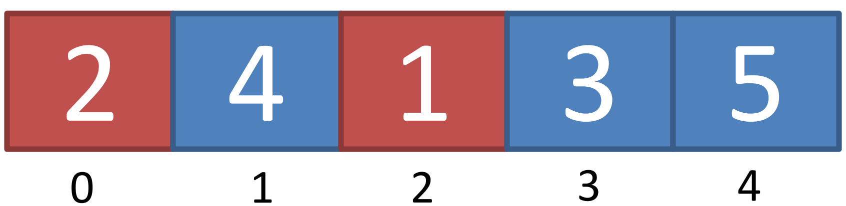


A[i] > A[j]? YES!

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)



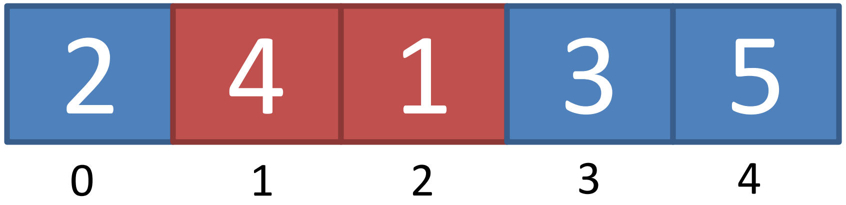
A[i] > A[j]? YES!

i < j? YES!

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)

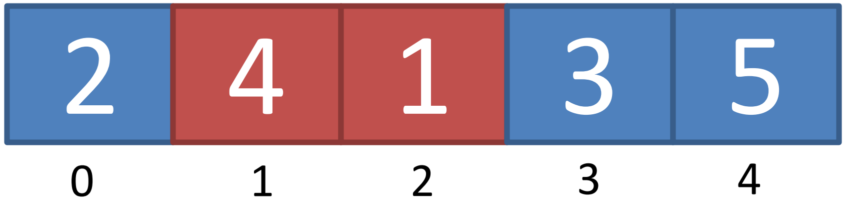


A[i] > A[j]? YES!

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)



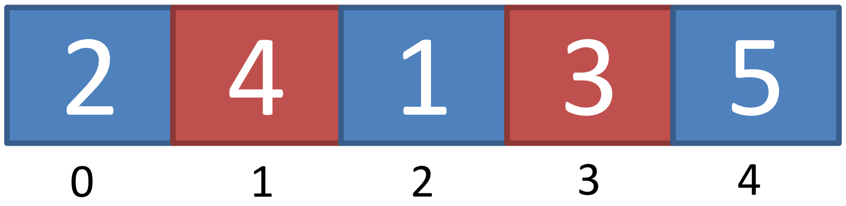
A[i] > A[j]? YES!

i < j? YES!

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)



A[i] > A[j]? YES!

Example

Array: [2, 4, 1, 3, 5]

Output (2, 1), (4, 1), (4, 3)

2	4	1	3	5
0	1	2	3	4

A[i] > A[j]? YES!

i < j? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

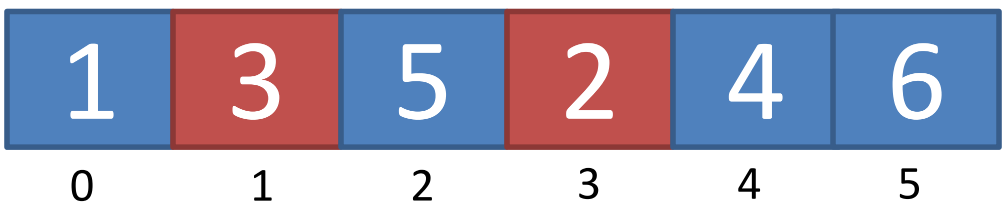
Output: (3, 2), (5, 2), (5, 4)

1	3	5	2	4	6
0	1	2	3	4	5

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

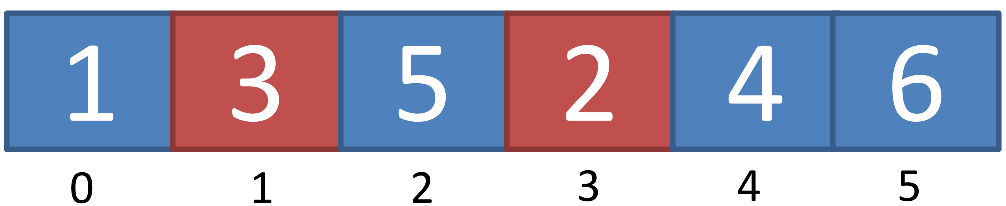


A[i] > A[j]? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)



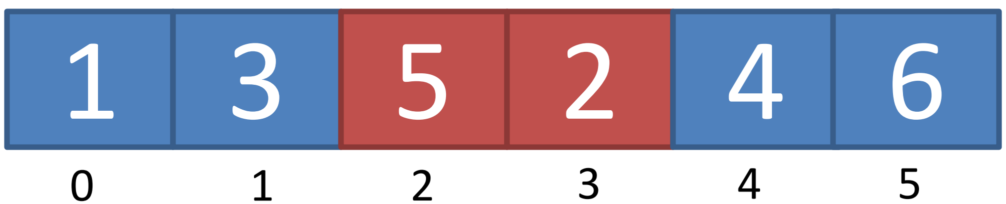
A[i] > A[j]? YES!

i < j? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

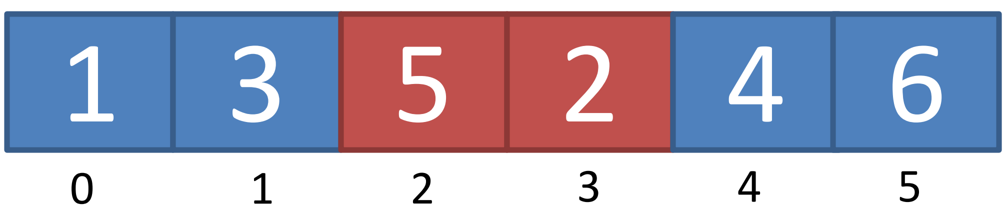


A[i] > A[j]? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)



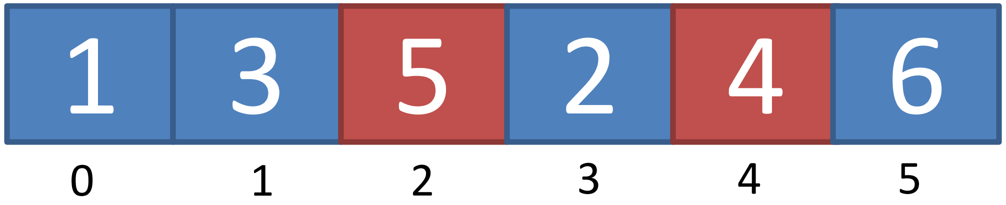
A[i] > A[j]? YES!

i < j? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)



A[i] > A[j]? YES!

Another Example

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

1	3	5	2	4	6
0	1	2	3	4	5

A[i] > A[j]? YES!

i < j? YES!

Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

1	3	5	2	4	6

Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)

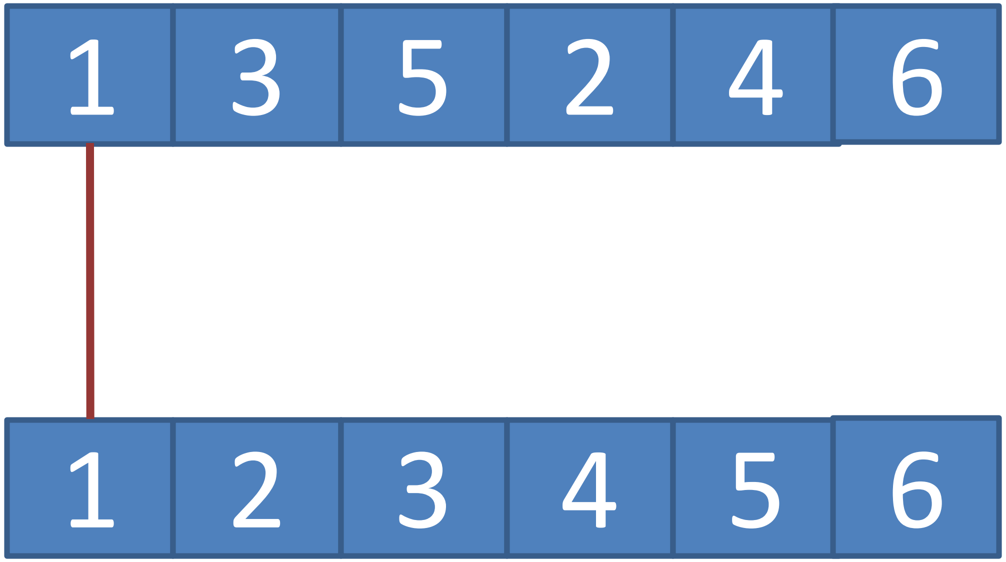
1	3	5	2	4	6

1	2	3	4	5	6

Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

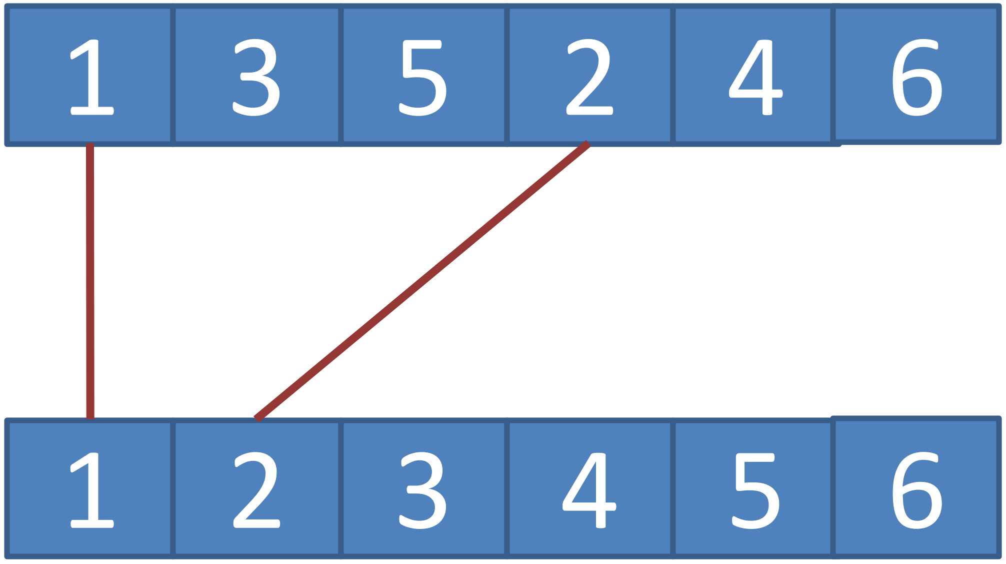
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

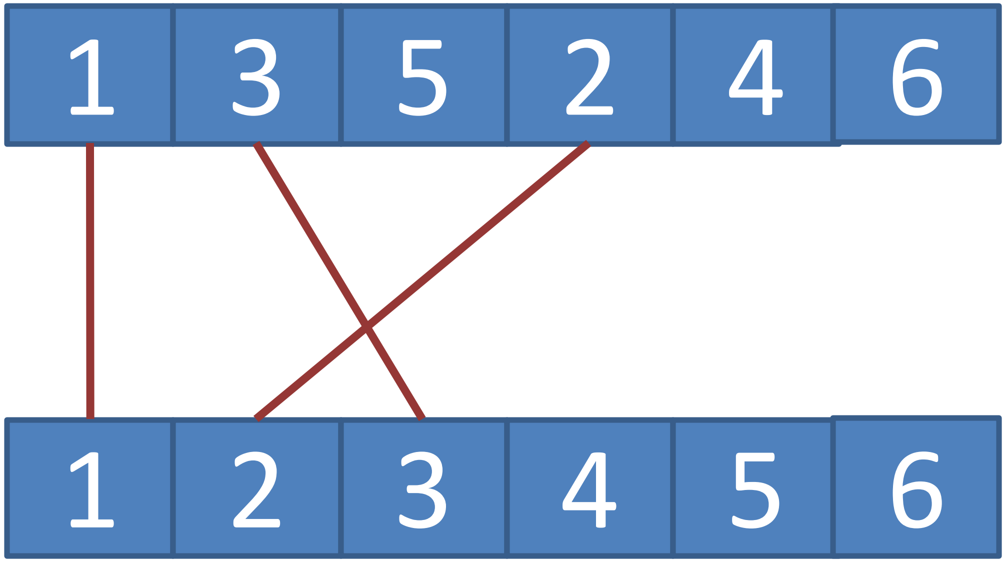
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

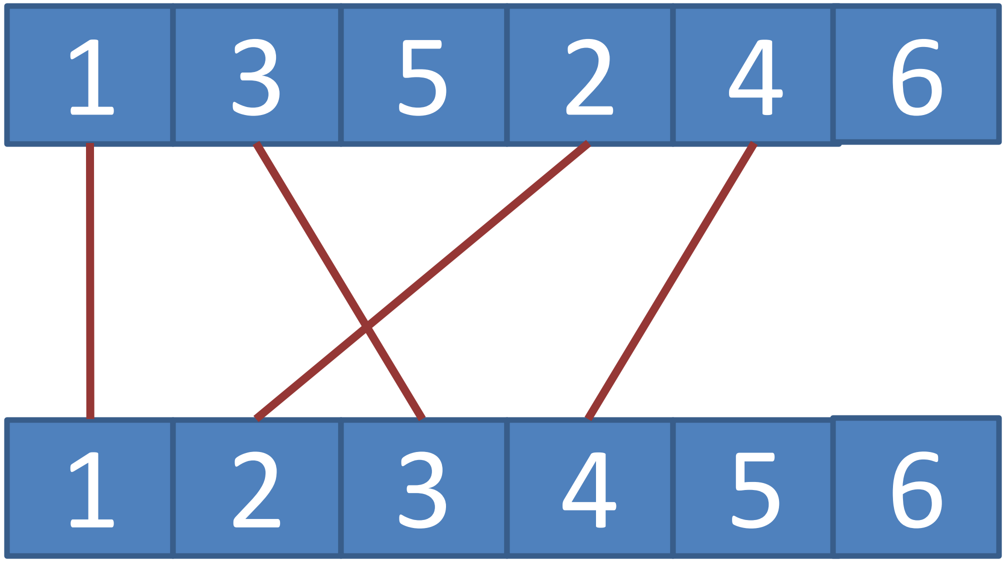
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

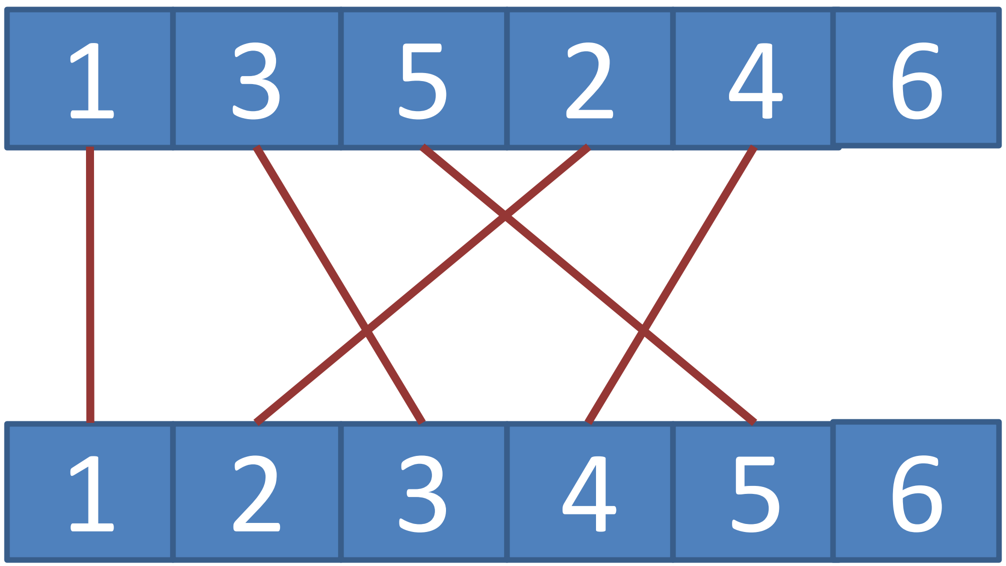
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

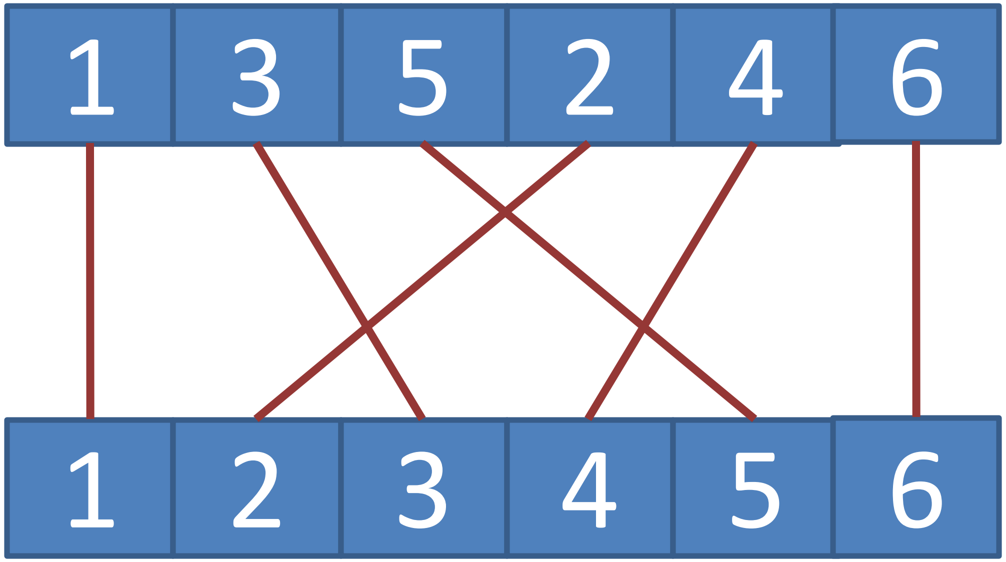
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

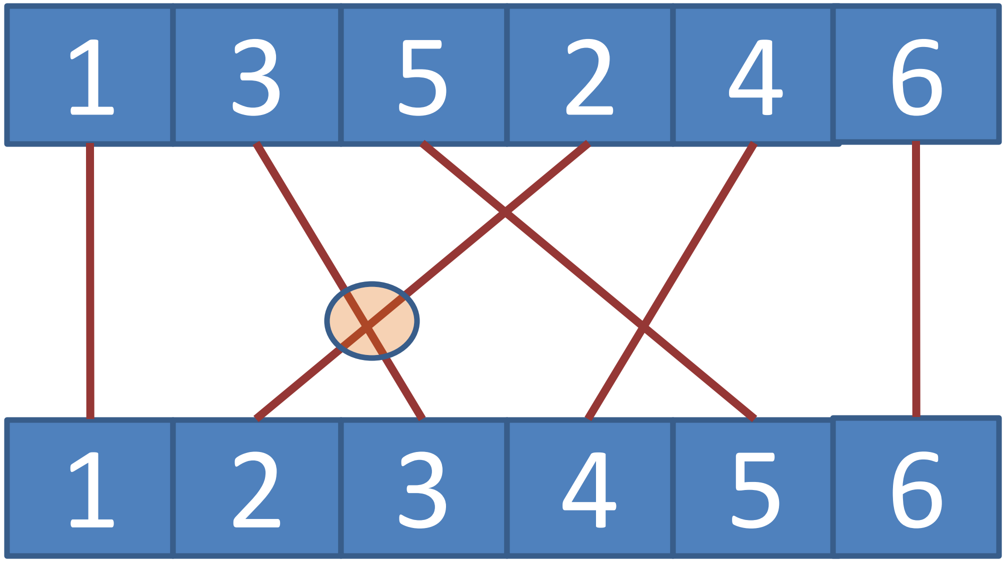
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

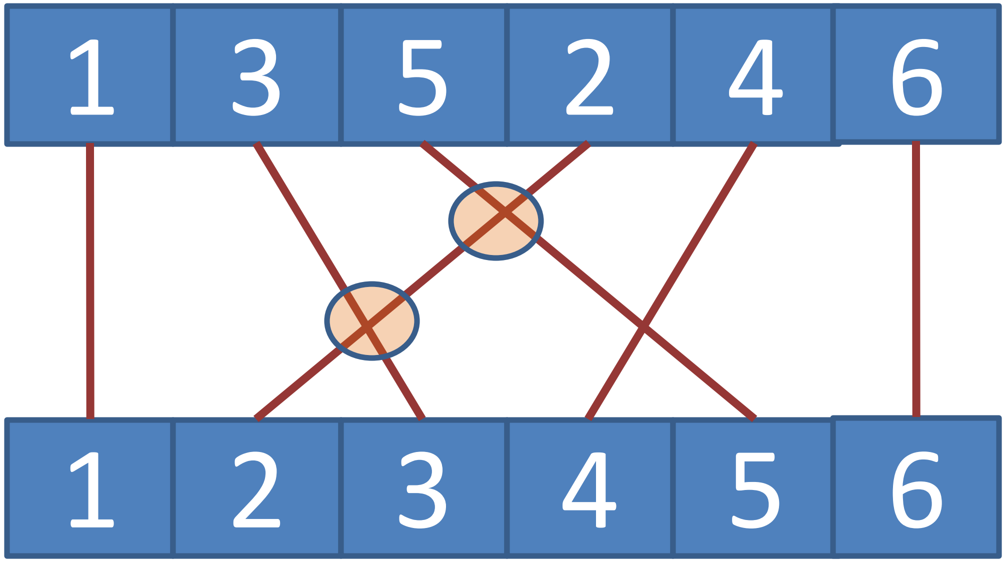
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

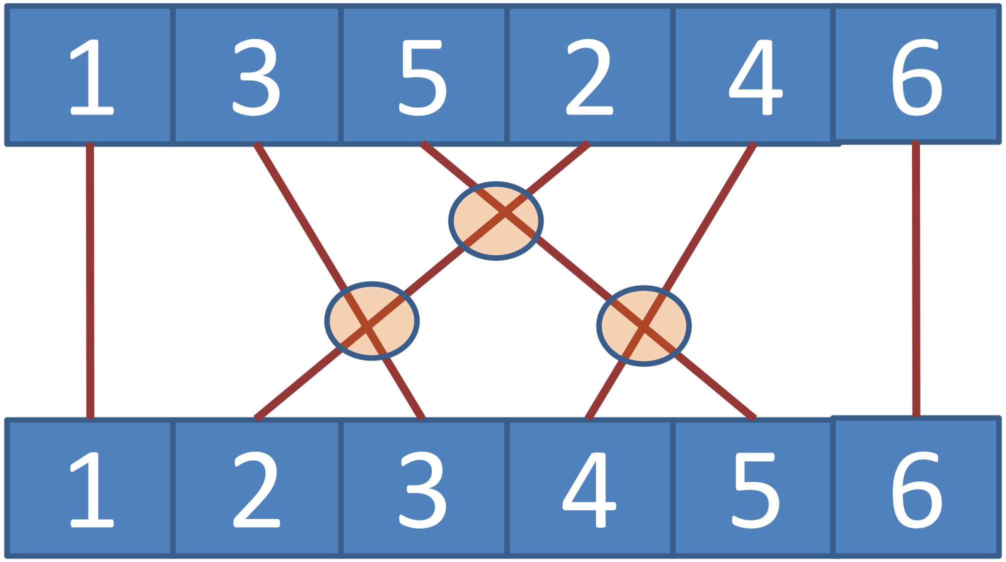
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

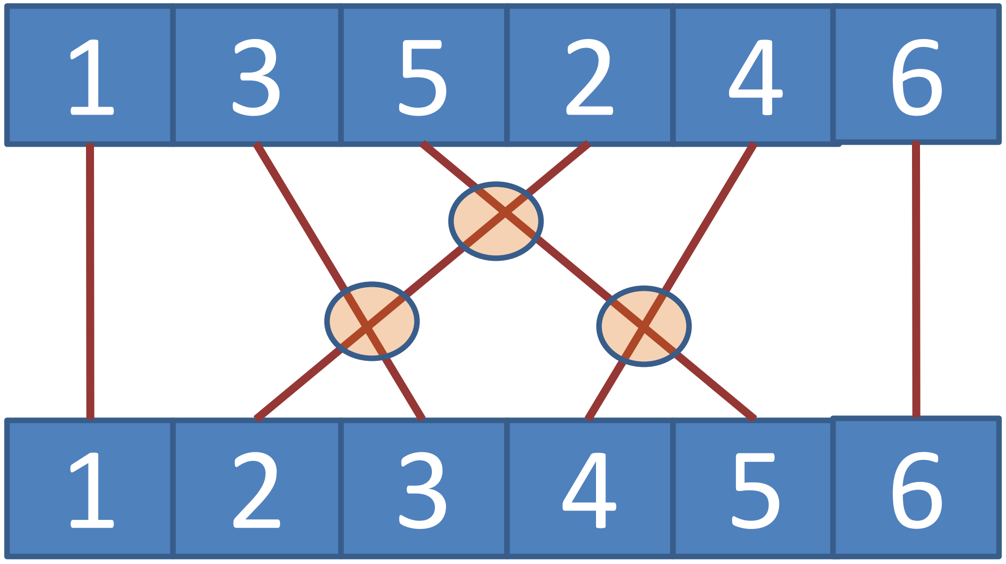
Output: (3, 2), (5, 2), (5, 4)



Counting Inversions

Array: [1, 3, 5, 2, 4, 6]

Output: (3, 2), (5, 2), (5, 4)



Number of Inversions? 3!

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

Formula?

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

Formula? $\boxed{n(n-1)/2}$

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64

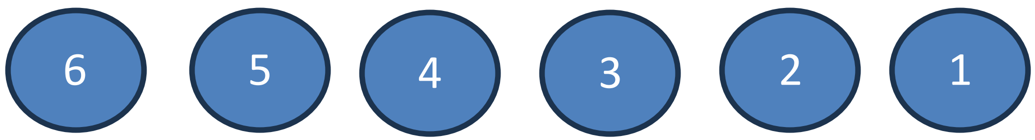
Formula? $\boxed{n(n-1)/2}$

Happens when the array is in the descending order / completely the opposite of other list

Question

What is the largest number of inversions that a 6-element array can have?

- 15
- 21
- 36
- 64



- **6:** (6,5), (6,4), (6,3), (6,2), (6,1) → **5 inversions**
- **5:** (5,4), (5,3), (5,2), (5,1) → **4 inversions**
- **4:** (4,3), (4,2), (4,1) → **3 inversions**
- **3:** (3,2), (3,1) → **2 inversions**
- **2:** (2,1) → **1 inversion**

5+4+3+2+1=15

Activity

Rank the following bands/artist according to your own preferences:

- Taylor Swift, Fallout Boy, All Time Low, MCR, Sabrina Carpenter, Hozier, Bini

Motivation

Why are we interested in such a problem?

- Answer: We want to find the numerical similarity between two ranked list
- Same ranking = 0 inversions
- The more varied the two lists, the more inversions there are.

Motivation

- Used in collaborative filtering
- When buying online, the system tries to suggest other items based on other people’s purchase with similar items

How can we implement the code?

Do you remember…

Tale of NOOBgrammer and PROgrammer

NOOBgrammer says:

Use **Brute-force** approach.

Brute-force Approach

Brute-force Approach

How will you prove the counting inversions problem using Brute-force?

Brute-force Approach

How will you prove the counting inversions problem using Brute-force?

- Answer: Nested for-loops to check each pair if it’s an inversion

Brute-force Approach

How will you prove the counting inversions problem using Brute-force?

- Answer: Nested for-loops to check each pair if it's an inversion

- Efficiency class: **$O(n^2)$**

Brute-force Approach: Pseudocode

```
CountingInversion(A, n):
        count = 0
        for i=1 to N:
                for j= i+1 to n:
                        if A[i] > A[j] and i<j:
                                count ++;
        return count;
```

i=1

count=0

CountingInversion(A, n):

count = 0

for i=1 to N: i=1

for j= i+1 to n:

if A[i] > A[j] and i<j:

count ++;

return count;

1	2	3	4	5	6
6	5	4	3	2	1

i=1

count=1

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:              i=1
        for j= i+1 to n:           j=2
            if A[i] > A[j] and i<j:     6>5 and 1<2
                count ++;
    return count;
```

1	2	3	4	5	6
6	5	4	3	2	1
A[i]	A[j]				

i=1

count=2

CountingInversion(A, n):

count = 0

for i=1 to N: i=1

for j= i+1 to n: j=3

if A[i] > A[j] and i<j: 6>4 and 1<3

count ++;

return count;

1	2	3	4	5	6
6	5	4	3	2	1
A[i]		*A[j]*			

i=1

count=3

CountingInversion(A, n):

    count = 0
    for i=1 to N: i=1
        for j= i+1 to n: j=4
            if A[i] > A[j] and i<j: 6>3 and 1<4
                count ++;
    return count;

1	2	3	4	5	6
6	5	4	3	2	1
A[i]			*A[j]*		

i=1

count=4

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:          i=1
        for j= i+1 to n:       j=5
            if A[i] > A[j] and i<j:      6>2 and 1<5
                count ++;
    return count;
```

1	2	3	4	5	6
6	5	4	3	2	1
A[i]				*A[j]*	

i=1

count=5

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:            i=1
        for j= i+1 to n:         j=5
            if A[i] > A[j] and i<j:      6>1 and 1<6
                count ++;
    return count;
```

1	2	3	4	5	6
6	5	4	3	2	1
A[i]					*A[j]*

i=2

count=6

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:            i=2
        for j= i+1 to n:         j=3
            if A[i] > A[j] and i<j:     5>4 and 2<3
                count ++;
    return count;
```

1	2	3	4	5	6
6	5	4	3	2	1
	A[i]	*A[j]*			

i=2

count=7

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:            i=2
        for j= i+1 to n:     j=4
            if A[i] > A[j] and i<j:    5>3 and 2<4
                count ++;
    return count;
```

1	2	3	4	5	6
6	5	4	3	2	1
	A[i]		*A[j]*		

i=2

count=8

1	2	3	4	5	6
6	5	4	3	2	1
	A[i]			*A[j]*	

```
CountingInversion(A, n):
    count = 0
    for i=1 to N:            i=2
        for j= i+1 to n:         j=5
            if A[i] > A[j] and i<j:     5>2 and 2<5
                count ++;
    return count;
```

i=2

count=9

CountingInversion(A, n):

count = 0

for i=1 to N: i=2

for j= i+1 to n: j=6

if A[i] > A[j] and i<j: 5>1 and 2<6

count ++;

return count;

1	2	3	4	5	6
6	5	4	3	2	1
	A[i]				A[j]

And so on and so forth….

Can we do better?

PROgrammer says:

YES!

Use **Divide and conquer** Approach!

Divide and Conquer Approach

Divide and Conquer Approach

- Key Idea: Piggyback on the merge sort, have recursive calls both to count inversions and sort

Divide and Conquer Approach

- Key Idea: Piggyback on the merge sort, have recursive calls both to count inversions and sort

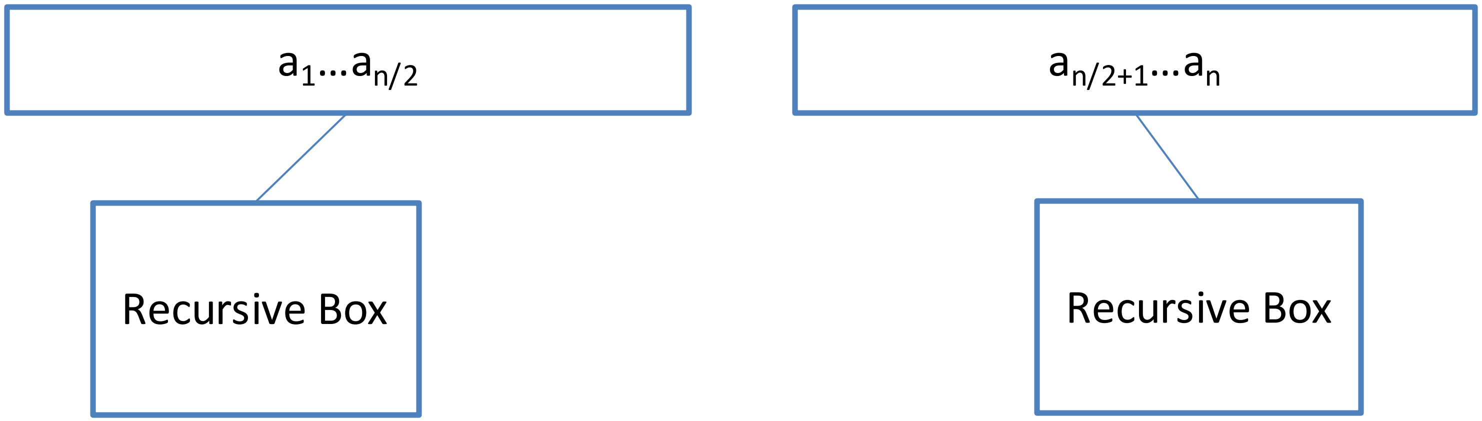
- This is the clever part of the algorithm.

Merge Sort + Count Inversions

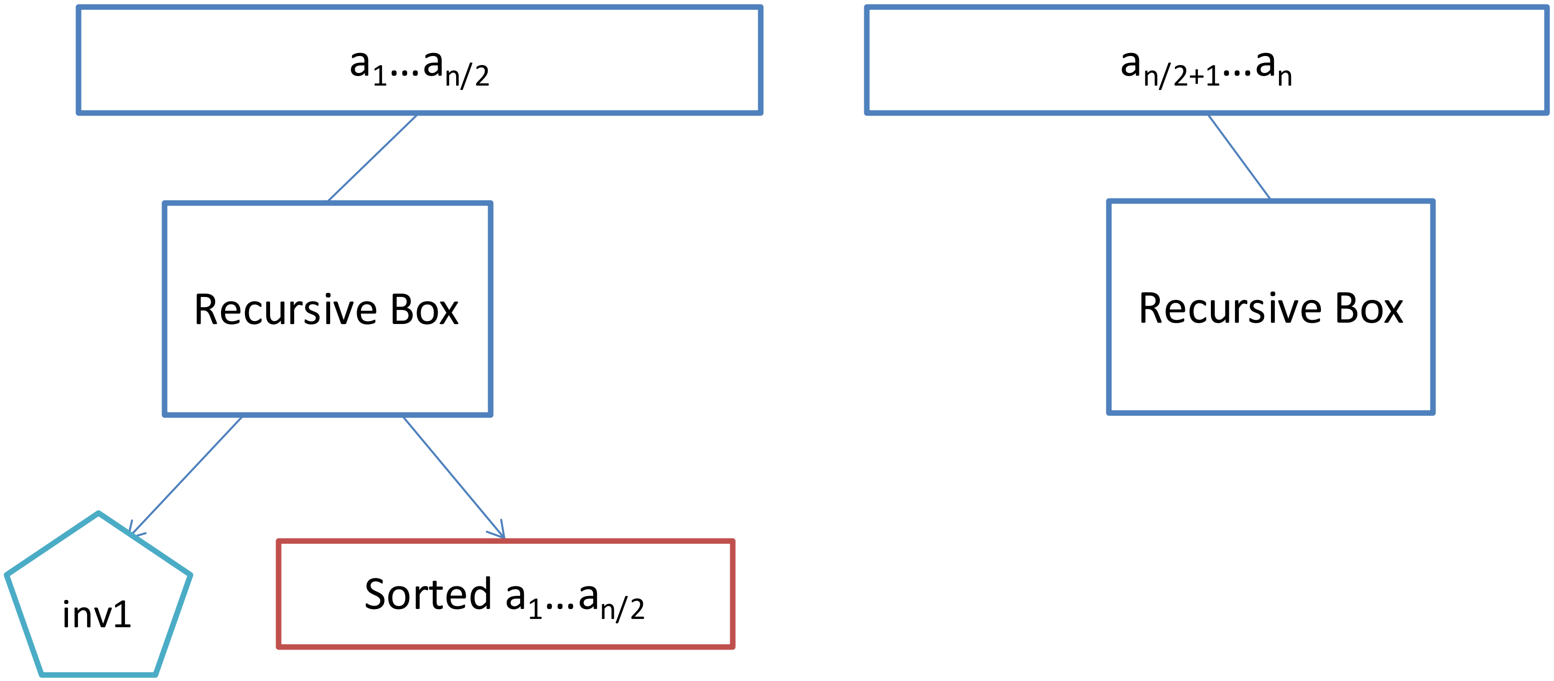
Merge Sort + Count Inversions

$a_1 \ldots a_{n/2}$	$a_{n/2+1} \ldots a_n$

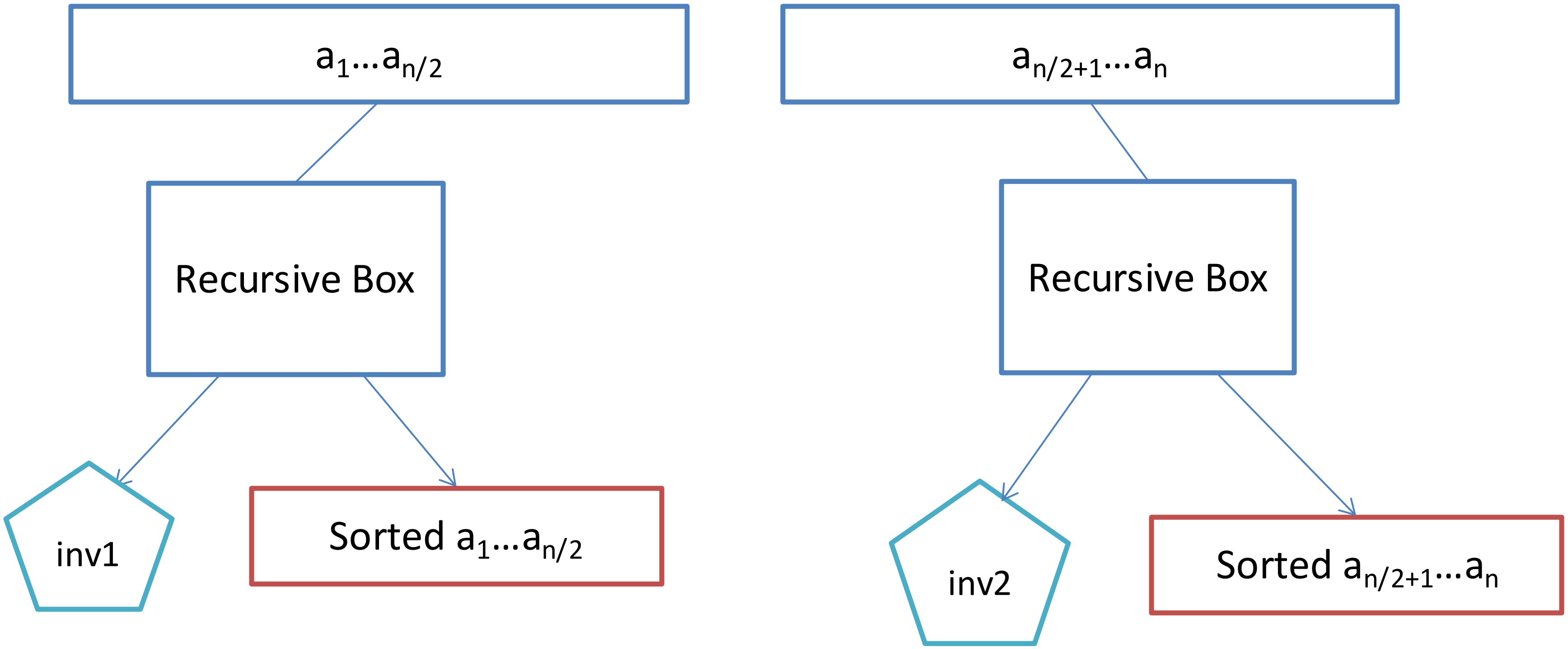
Merge Sort + Count Inversions



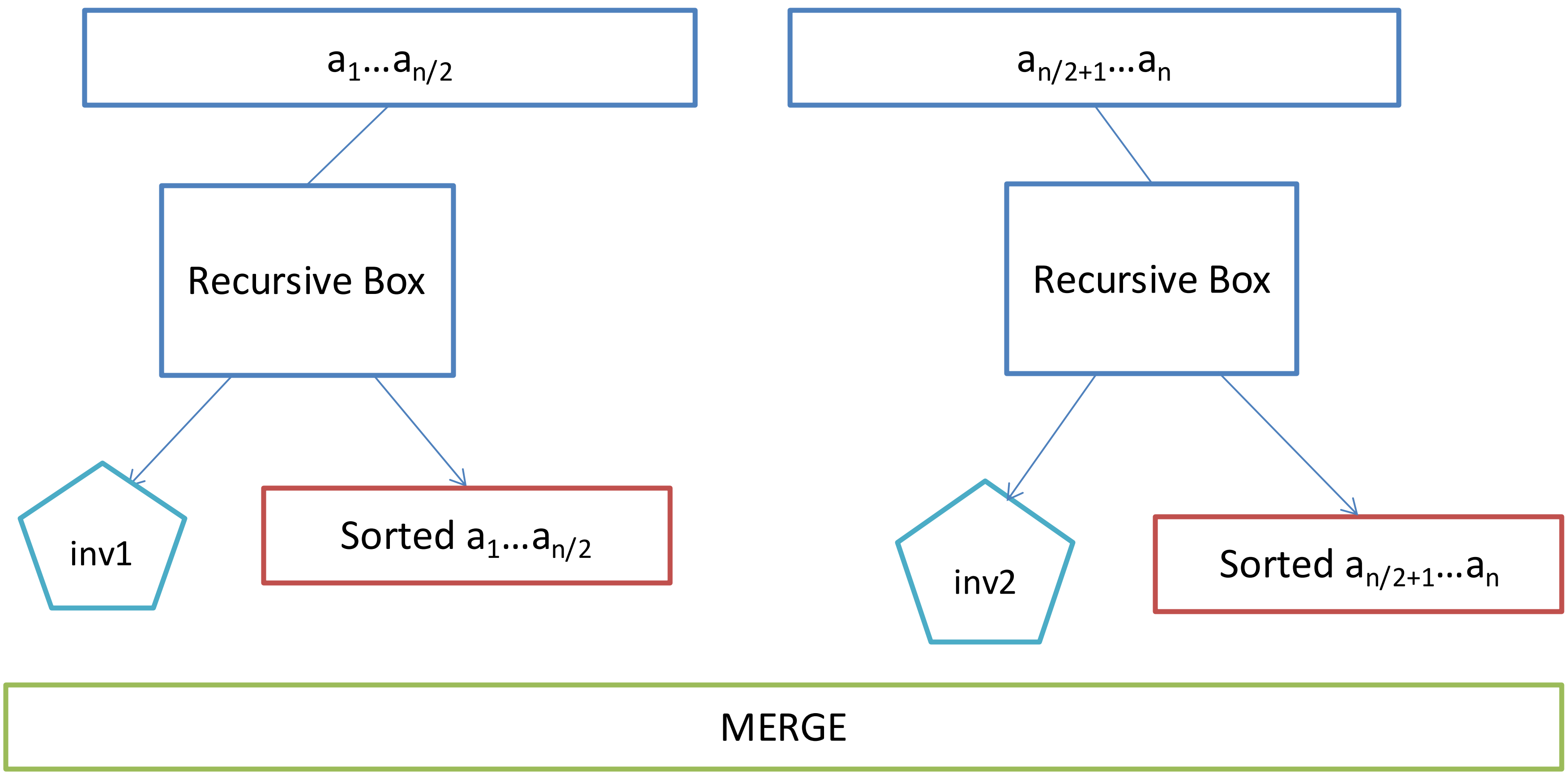
Merge Sort + Count Inversions



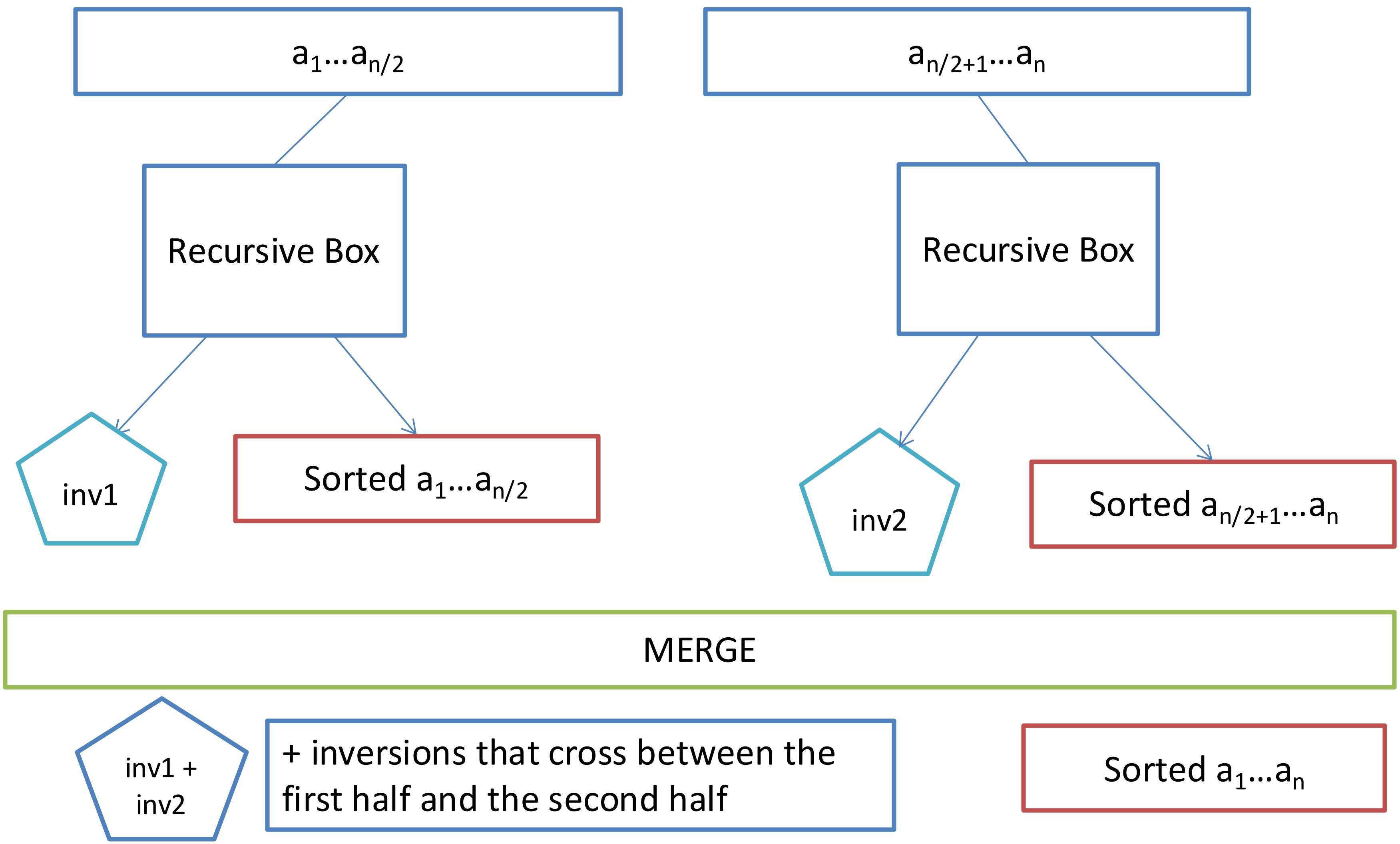
Merge Sort + Count Inversions



Merge Sort + Count Inversions



Merge Sort + Count Inversions



Merge Sort + Count Inversions

Merge Sort + Count Inversions

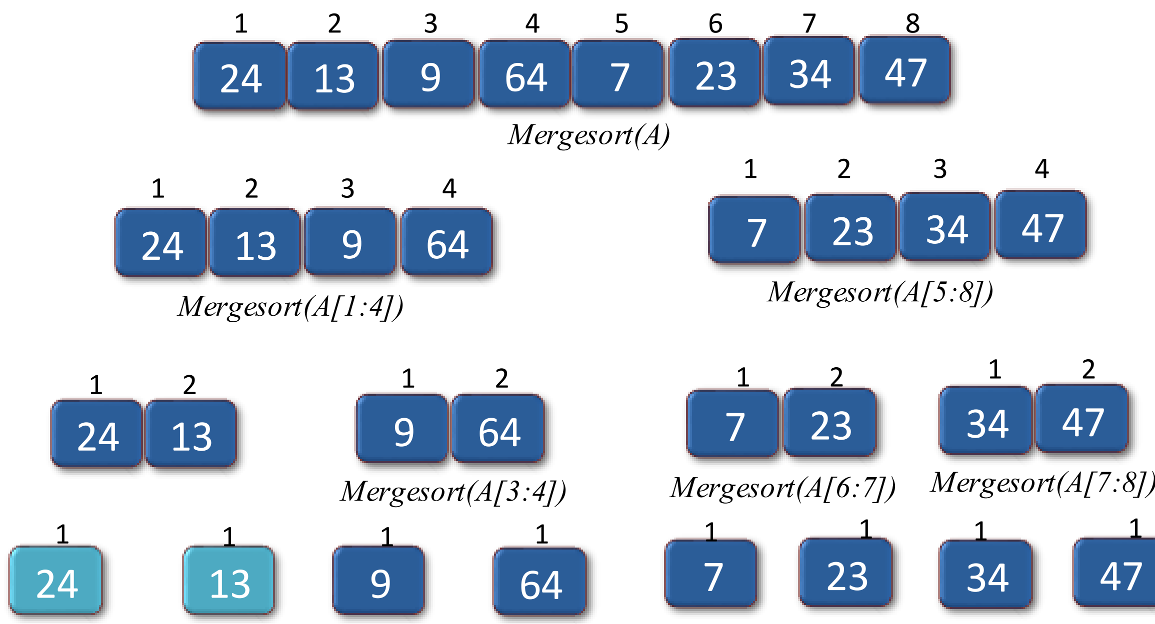
```
sort_count_inversions(array A):
    if N == 1: return A, 0
    M = floor(N/2)
    B, L = sort_count_inversions( A[ 1 : M ] )         | left inversions
    C, R = sort_count_inversions( A[ M+1 : N ] )       | right inversions
    D, S = merge_count_split_inversions(B, C)          | split inversions
    return D, L+R+S
```

Why do this?

- We'll see that the Merge subroutine naturally uncovers split inversions.

B, L = sort_count_inversions(A[1 : 4])

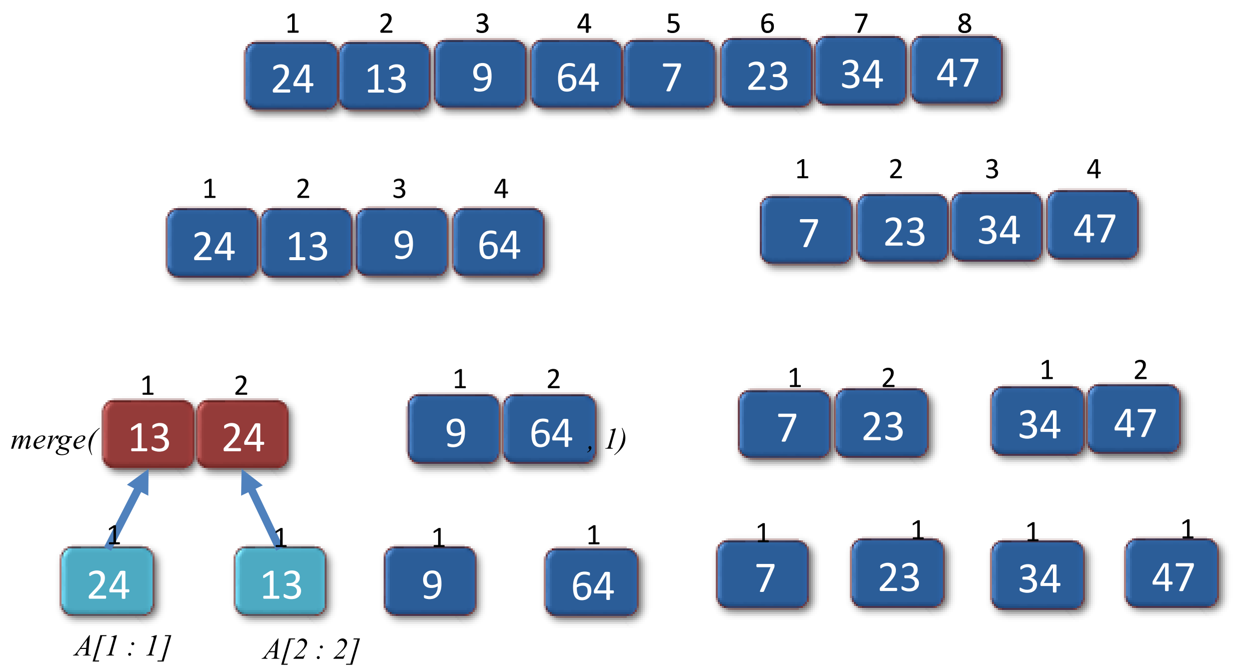
inversion count:0




```
sort_count_inversions(array A):
    if N == 1: return A, 0
    M = floor(N/2)
    B, L = sort_count_inversions( A[ 1 : M ] )
    C, R = sort_count_inversions( A[ M+1 : N ] )
    D, S = merge_count_split_inversions(B, C)
    return D, L+R+S
```

(24,13)

inversion count:
1



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

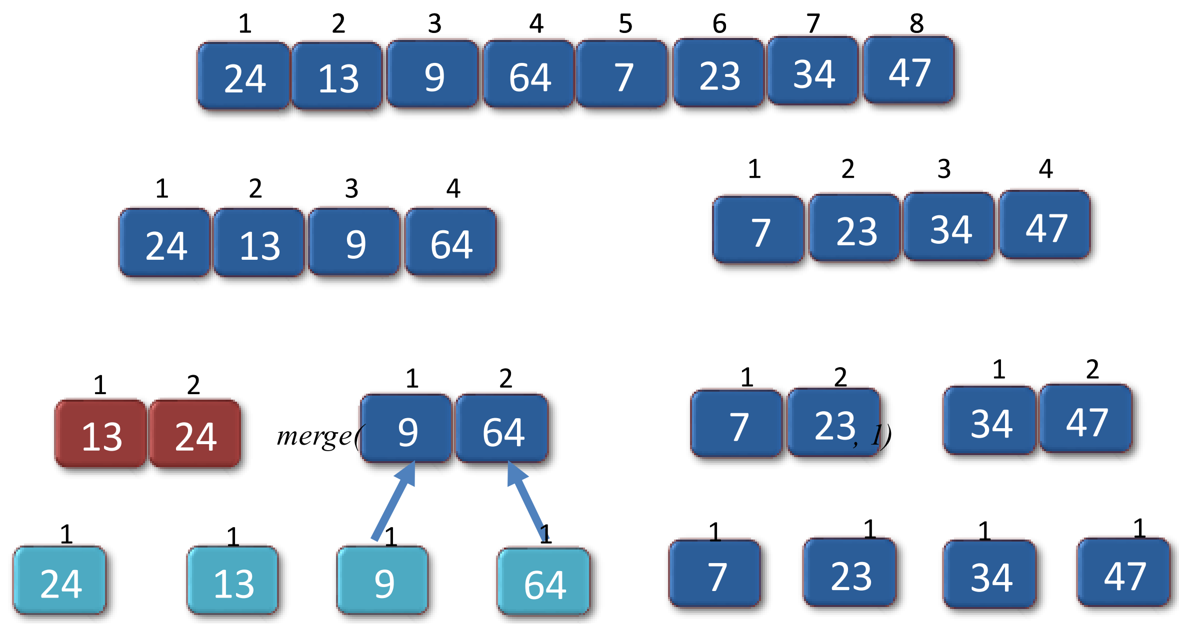
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13)

inversion count:
1



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

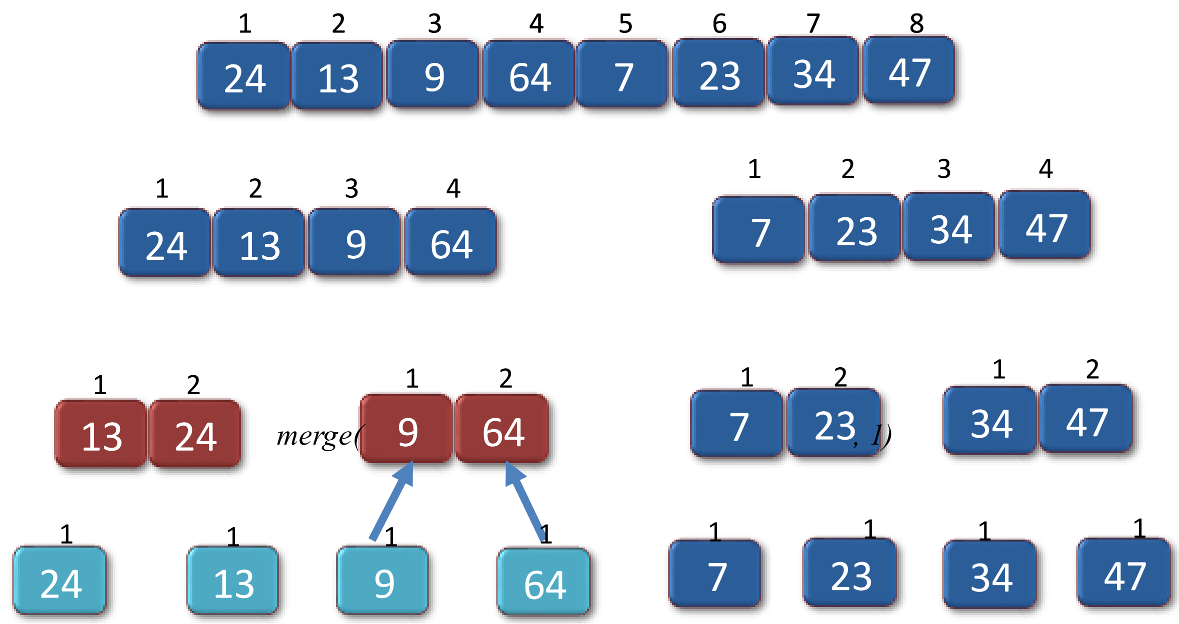
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13)

inversion count:
1+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

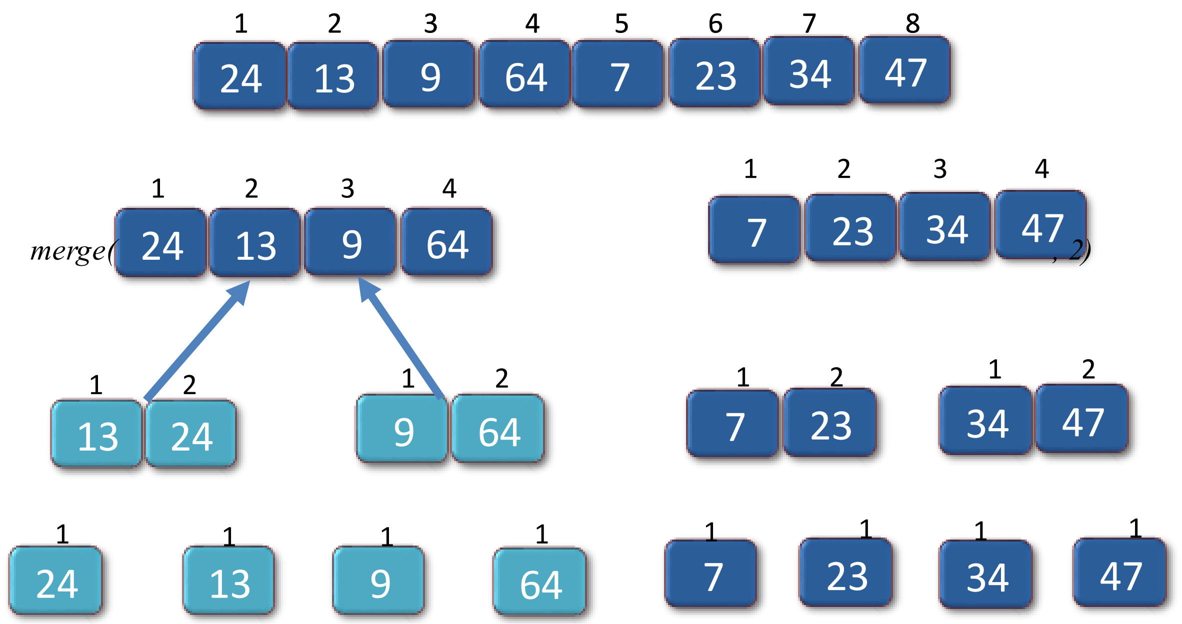
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13)

inversion count:
1+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

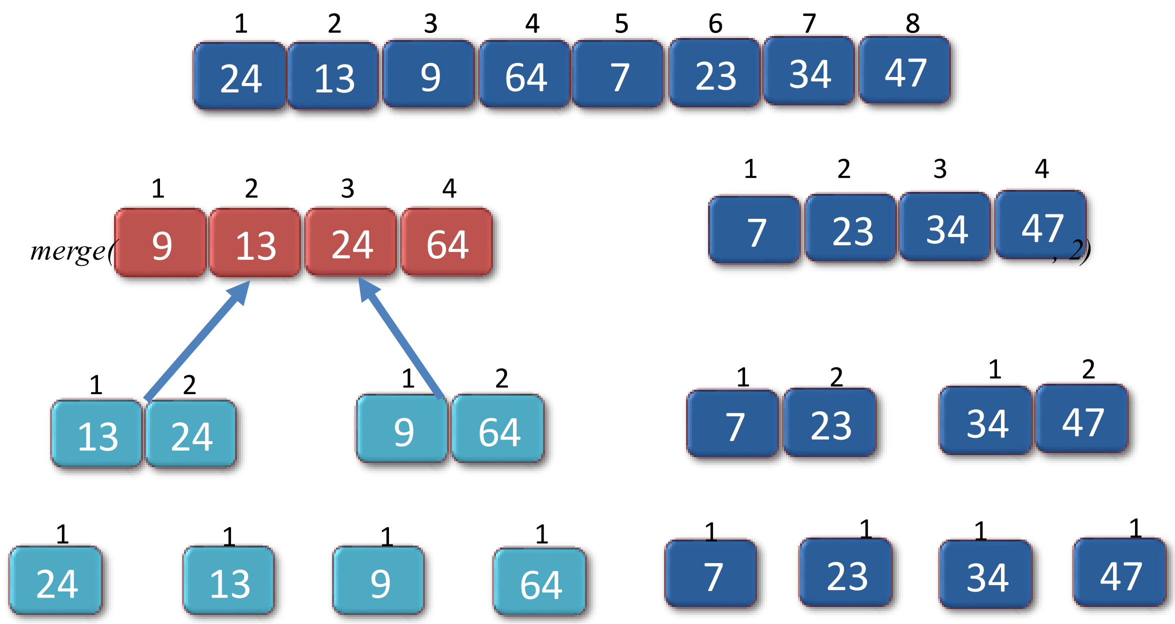
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

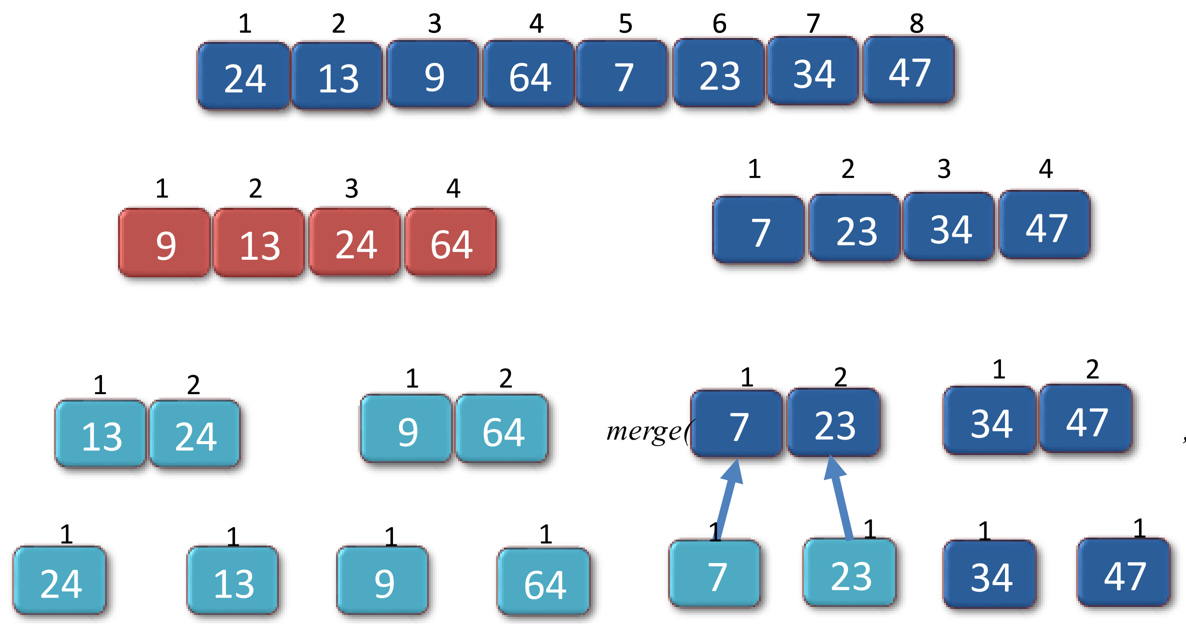
inversion count:
1+0+2




```
sort_count_inversions(array A):
    if N == 1: return A, 0
    M = floor(N/2)
    B, L = sort_count_inversions( A[ 1 : M ] )
    C, R = sort_count_inversions( A[ M+1 : N ] )
    D, S = merge_count_split_inversions(B, C)
    return D, L+R+S
```

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

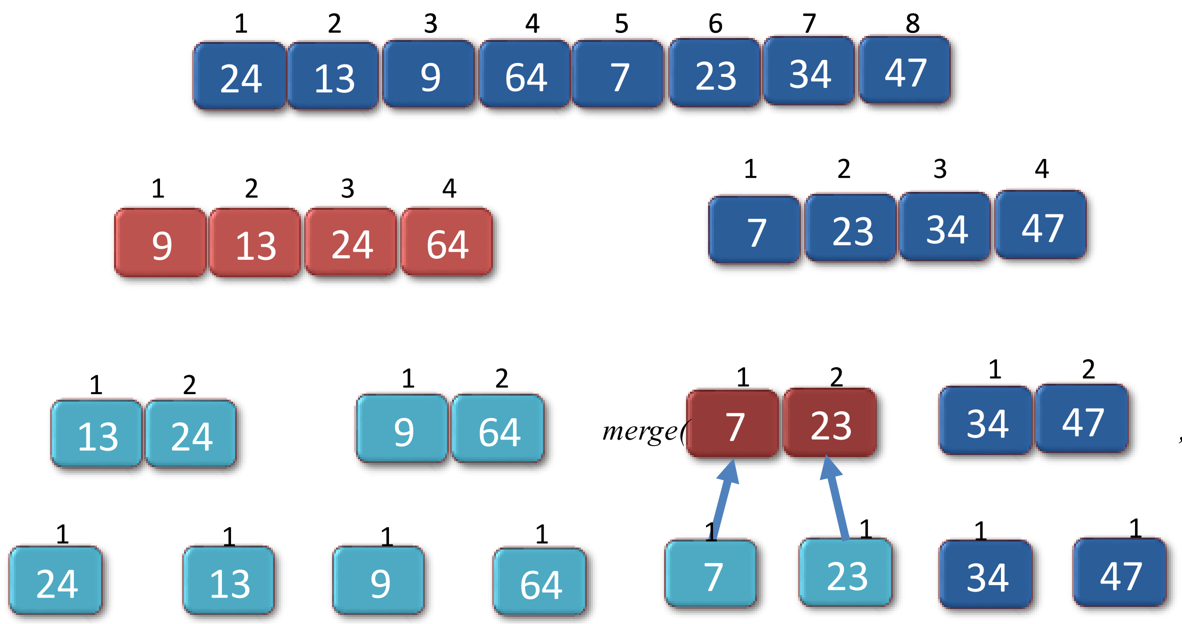
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

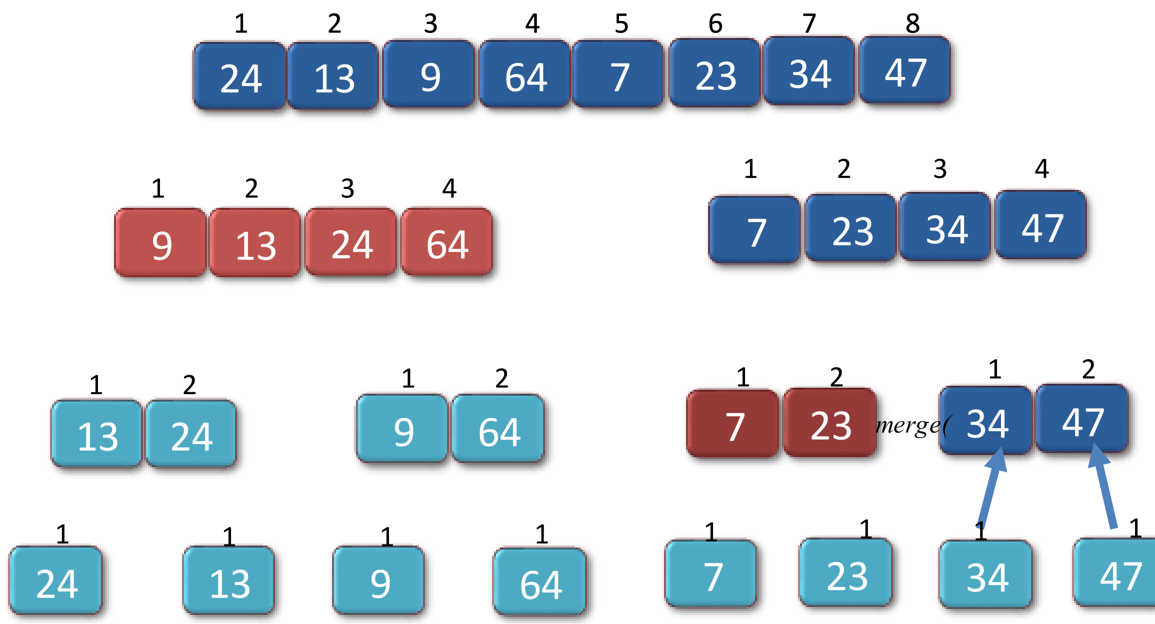
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

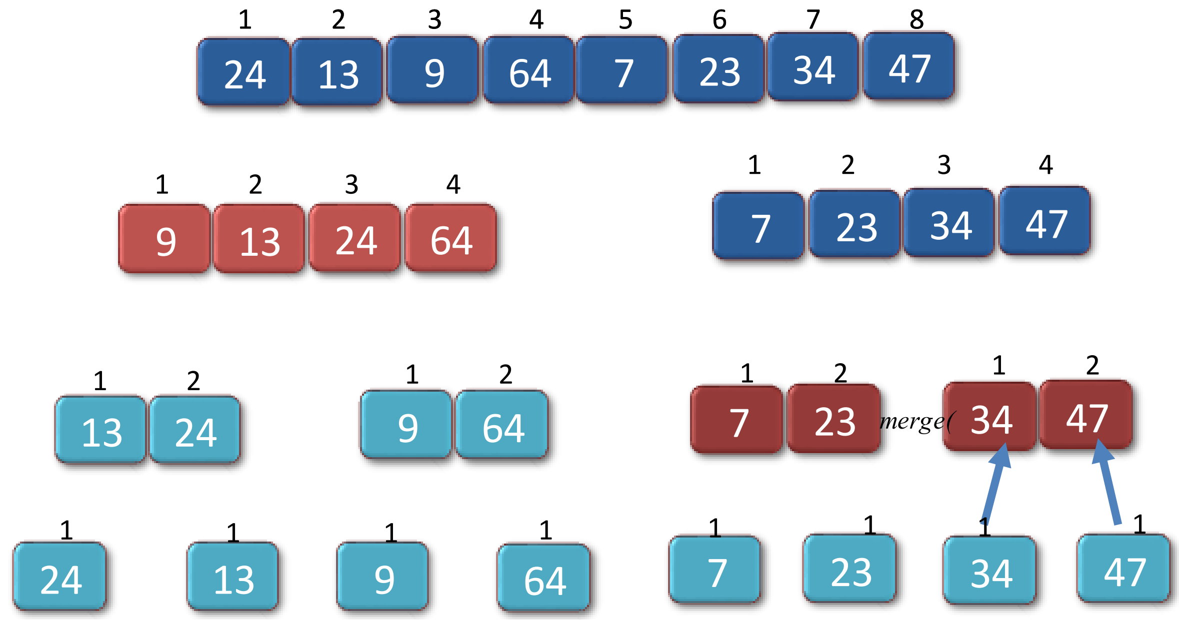
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2+0+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

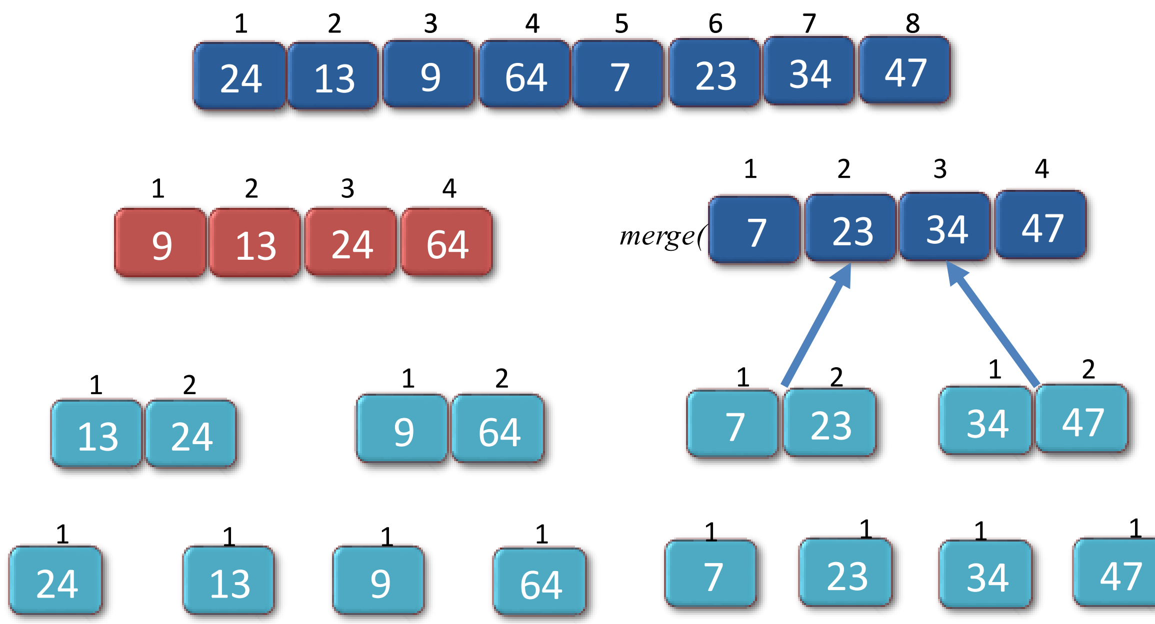
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

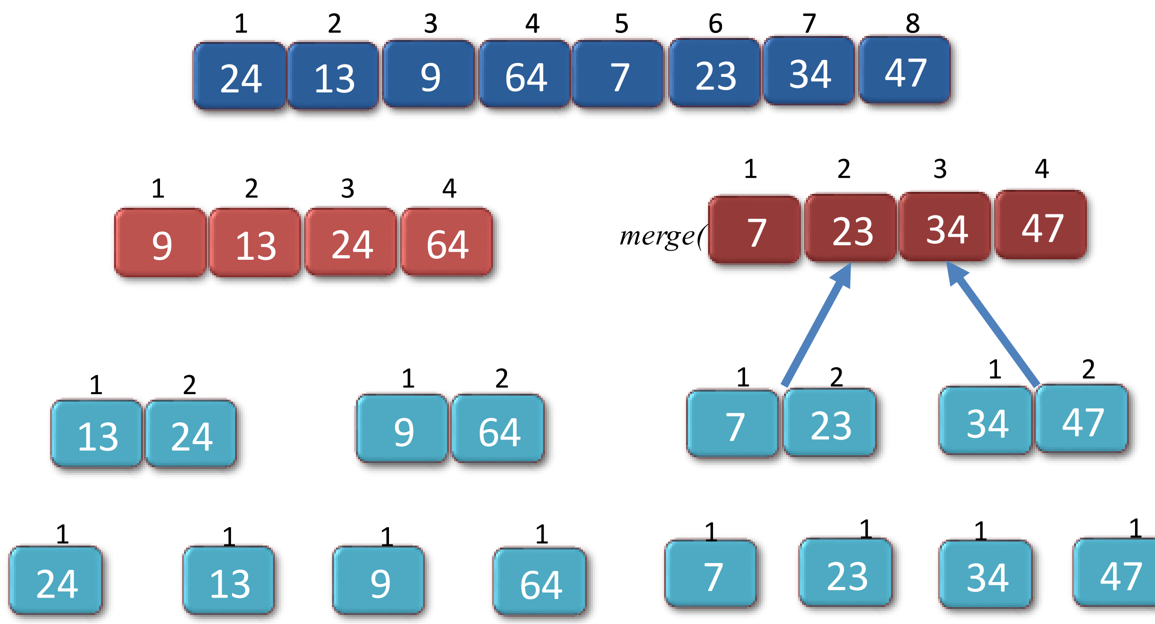
inversion count:
1+0+2+0+0



sort_count_inversions(array A):
if N == 1: return A, 0
M = floor(N/2)
B, L = sort_count_inversions(A[1 : M])
C, R = sort_count_inversions(A[M+1 : N])
D, S = merge_count_split_inversions(B, C)
return D, L+R+S

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2+0+0+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

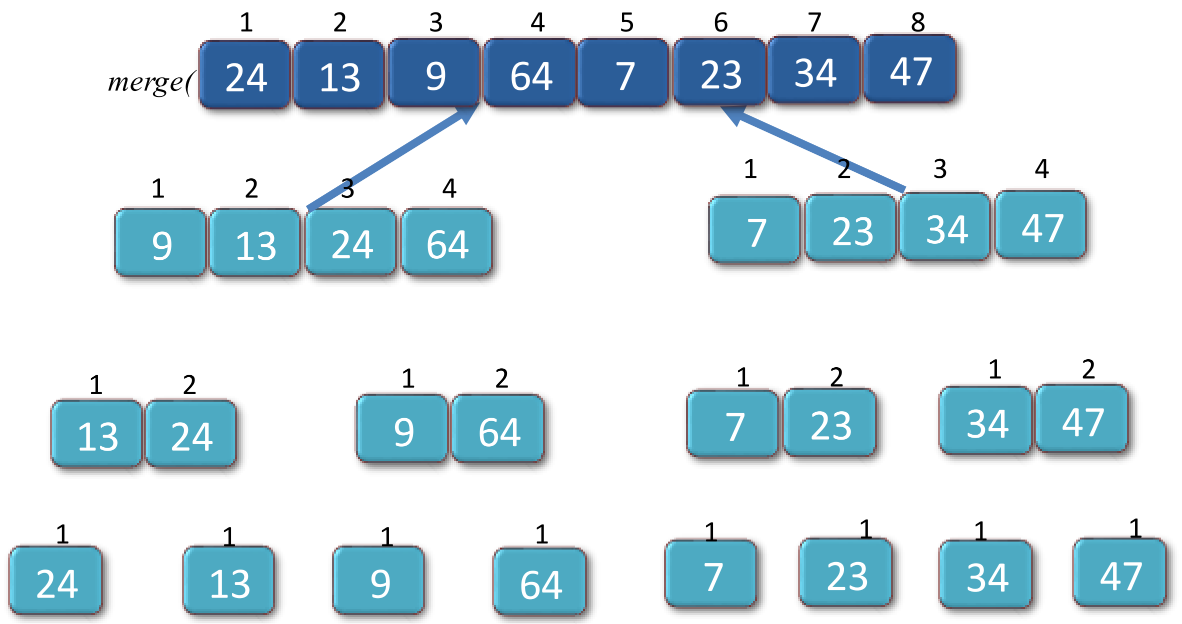
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9)

inversion count:
1+0+2+0+0



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

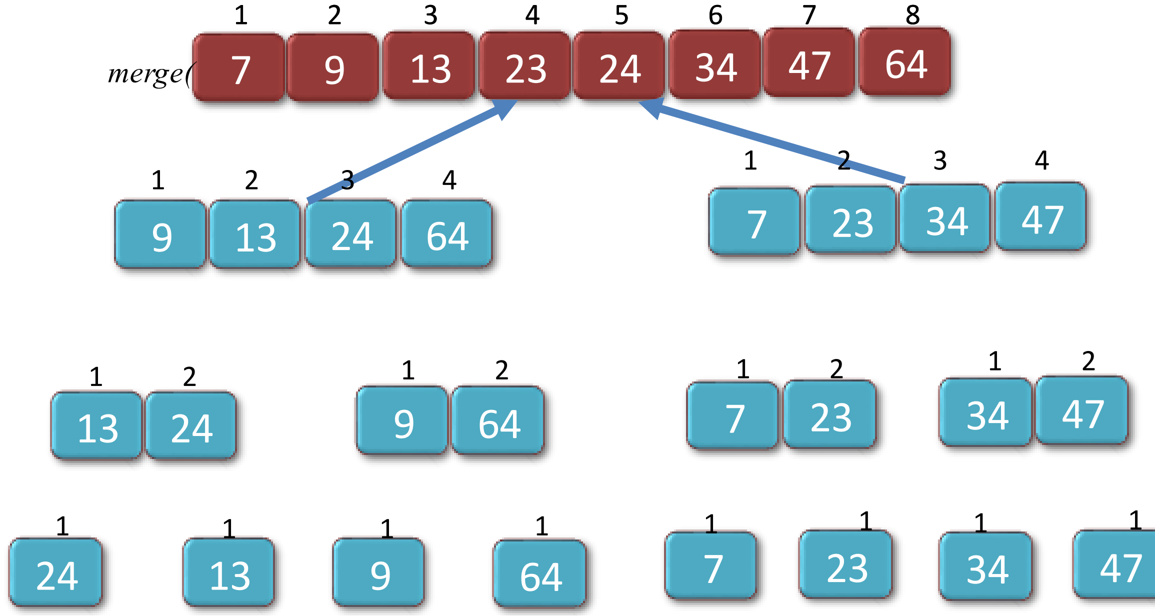
C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

(24,13), (24, 9), (13, 9), (9,7), (13, 7), (24, 7), (64, 7), (24, 23), (64, 23), (64, 34), (64, 47)

inversion count:
1+0+2+0+0+4+2+1+1



sort_count_inversions(array A):

if N == 1: return A, 0

M = floor(N/2)

B, L = sort_count_inversions(A[1 : M])

C, R = sort_count_inversions(A[M+1 : N])

D, S = merge_count_split_inversions(B, C)

return D, L+R+S

Merge Sort + Count Inversions

How to get inversions in merge()?

Merge Sort + Count Inversions

How to get inversions in merge()?

- In merge process, let *i* is used for indexing left sub-array and *j* for right sub-array.

Merge Sort + Count Inversions

How to get inversions in merge()?

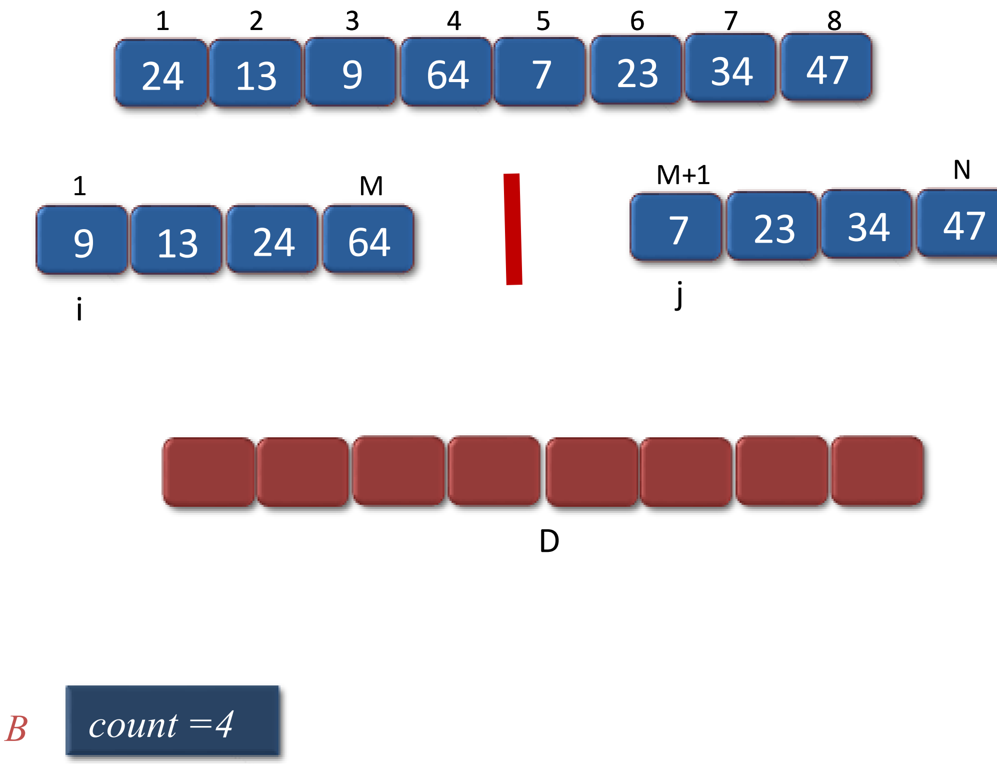
- In merge process, let *i* is used for indexing left sub-array and *j* for right sub-array.
- At any step in merge(), if *a[i]* is greater than *a[j],* then there are (mid-i) inversions.

Merge Sort + Count Inversions

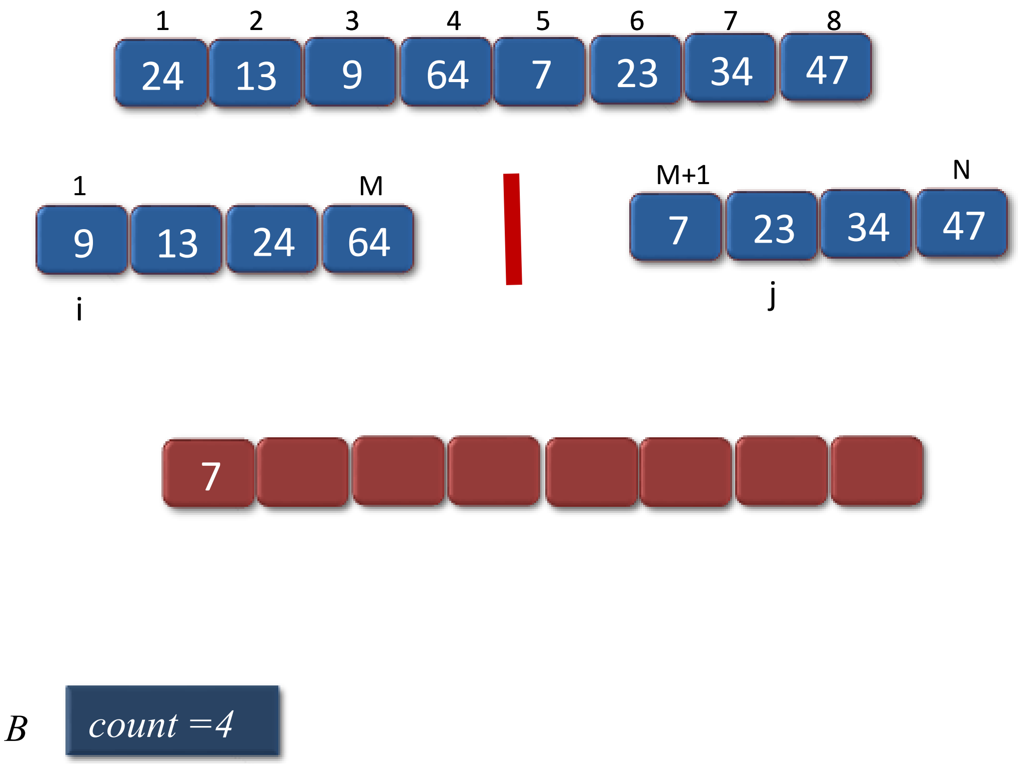
How to get inversions in merge()?

- In merge process, let *i* is used for indexing left sub-array and *j* for right sub-array.
- At any step in merge(), if *a[i]* is greater than *a[j],* then there are (mid-i) inversions.
- Because left and right subarrays are sorted, so all the remaining elements in the left-subarray *(a[i+1], a[i+2]..a[mid])* will be greater than a[j])

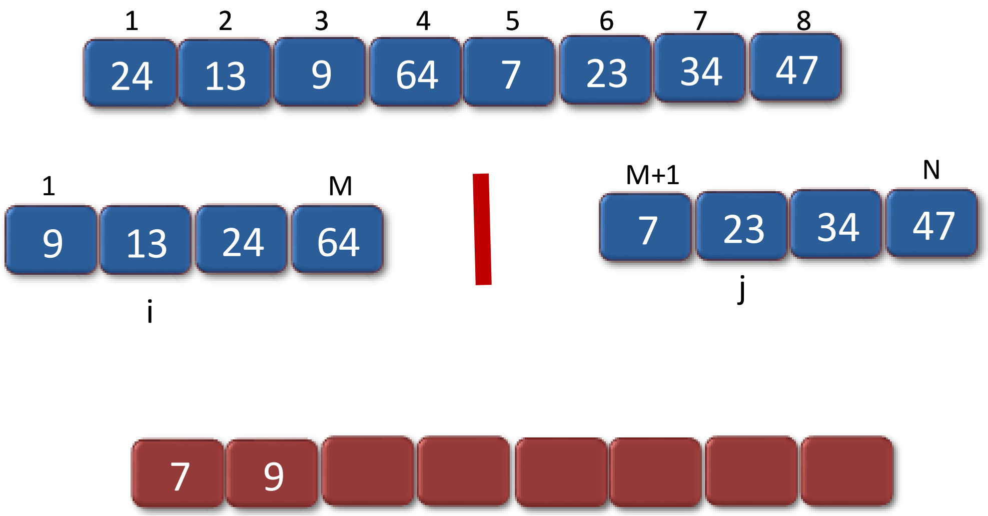
```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```

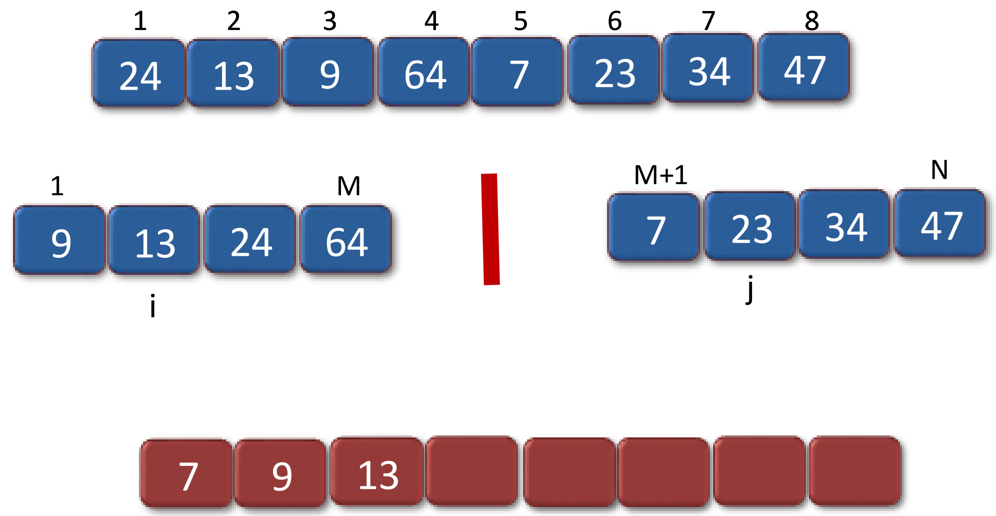


```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



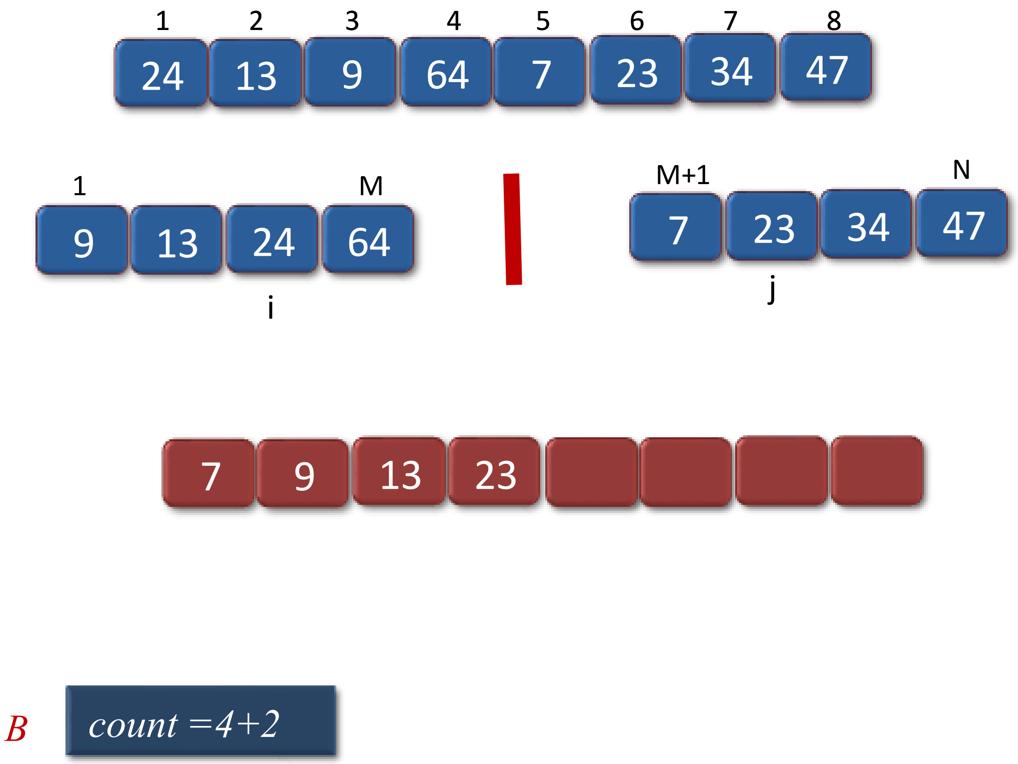
count =4

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```

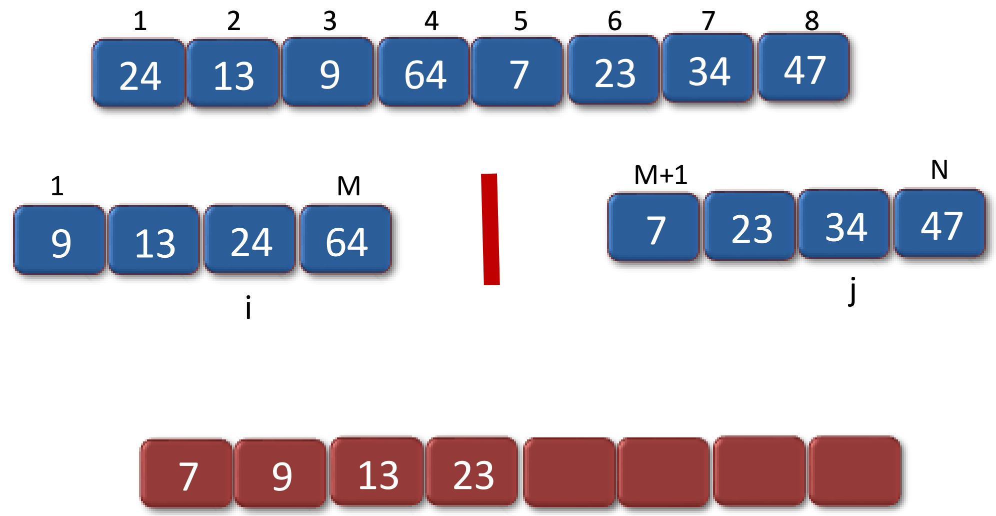


count =4

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```

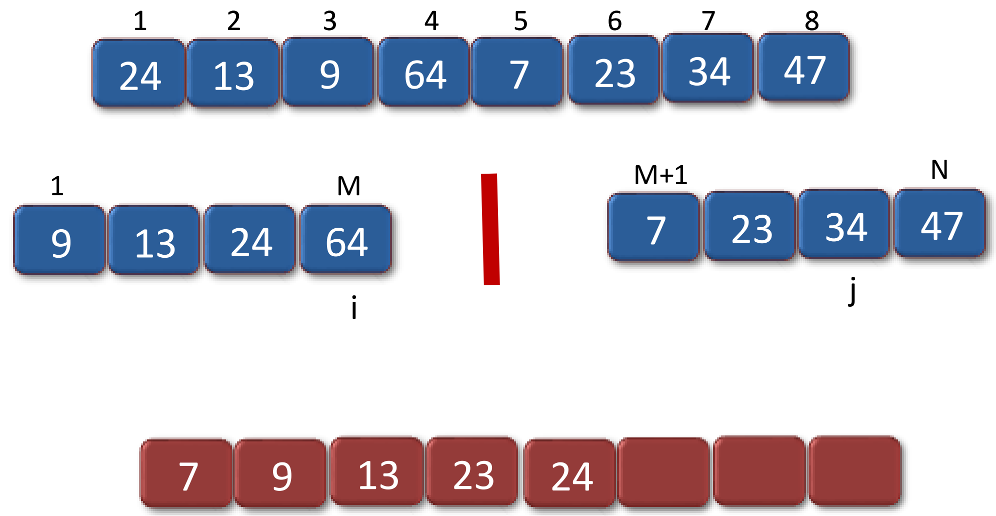


```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



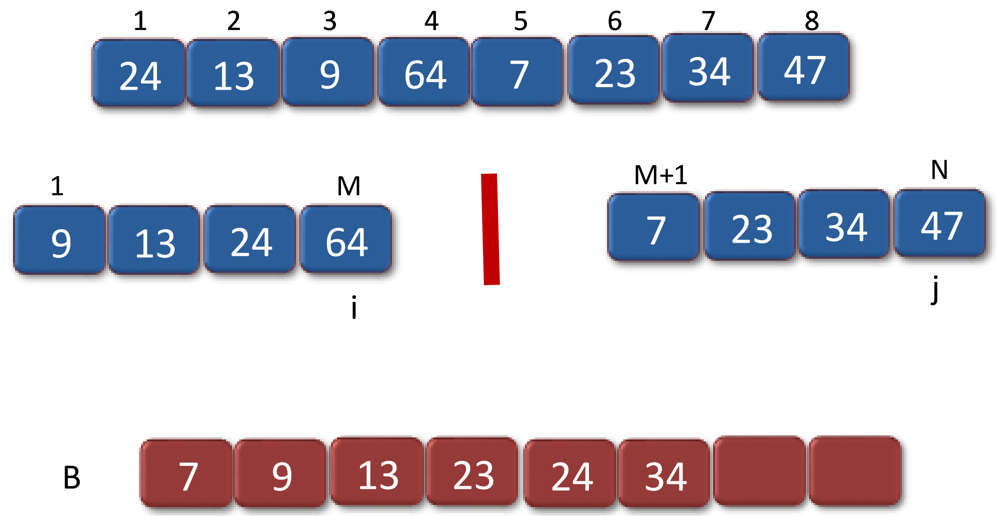
count =4+2

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



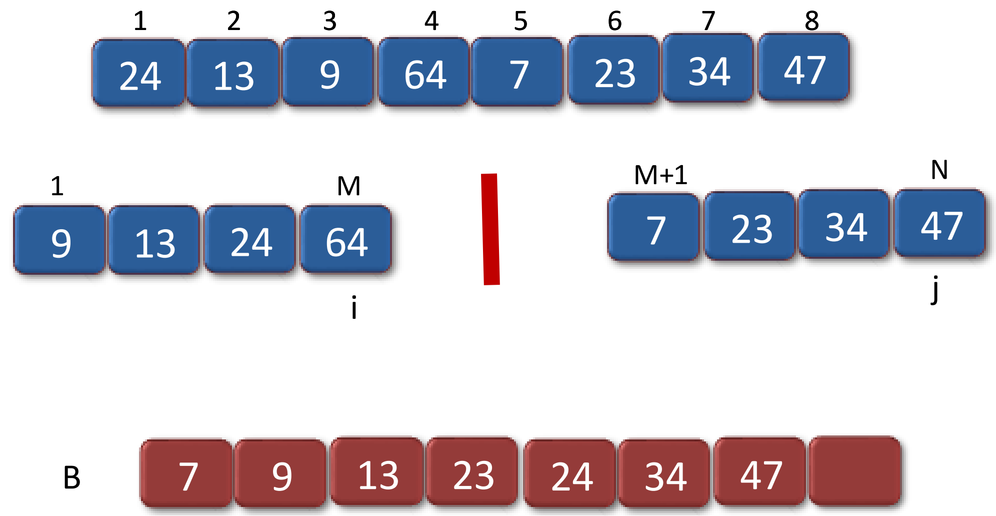
count =4+2+1

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



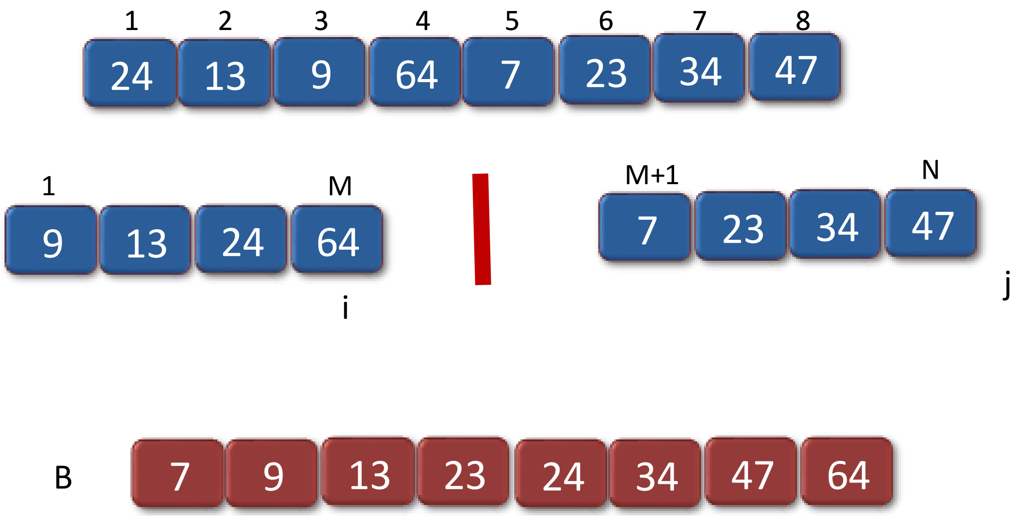
count =4+2+1

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



count =4+2+1+1

```
merge_count_split_inversions(array B, array C):
    i = 1, j = 1
    count = 0
    D = [ ]
    for z = 1 to N:
        if i > size(B):
            D[z] = C[j];
            j++
        else if j > size(C):
            D[z] = B[i];
            i++
        else if B[i] ≤ C[j]:
            D[z] = B[i];
            i++
        else if C[j] < B[i]:
            D[z] = C[j];
            j++
            count += no. of remaining items in B
    return D, count
```



count =4+2+1+1

Goal

- Implement MergeAndCountSplitInv in linear time to have SortAndCount run in O(n log n)

Question

Suppose the input array A has no split inversions, what is the relationship between the sorted subarrays *leftA* and *rightA*?

a. *leftA* has the smallest element of A, *rightA* the second smallest, the third-smallest, and so on
b. All elements of *leftA* are less than all elements of *rightA*
c. All elements of *leftA* are greater than all elements of *rightA*
d. There is not enough information to answer this question

Question

Suppose the input array A has no split inversions, what is the relationship between the sorted subarrays *leftA* and *rightA*?

a. *leftA* has the smallest element of A, *rightA* the second smallest, the third-smallest, and so on
b. All elements of *leftA* are less than all elements of *rightA*
c. All elements of *leftA* are greater than all elements of *rightA*
d. There is not enough information to answer this question

Count Inversions in an Array

Method	Time Complexity
Brute-force	$O(n^2)$
Divide and Conquer	O(nlogn)

Finding the Median

Median

- The median of a list of numbers is its 50^{th} percentile
- Half of the numbers are bigger than it, half are smaller

Median

Median

- Purpose: Summarize a set of numbers by a single, typical value

Median

- Purpose: Summarize a set of numbers by a single, typical value
- Mean is also used for this purpose (average) but median is more typical of data

Median

- Purpose: Summarize a set of numbers by a single, typical value
- Mean is also used for this purpose (average) but median is more typical of data
- Median is always one of the values; it's less sensitive to outliers

Example

Example

- List of hundred 1s : Mean and Median = 1

Example

- List of hundred 1s : Mean and Median = 1
- If one 1 gets corrupted to 10,000, mean shoots up above 100, median is unaffected

Mean vs Median

13, 18, 13, 14, 13, 16, 14, 21, 13

Mean: usual average

(13 + 18 + 13 + 14 + 13 + 16 + 14 + 21 + 13) / 9 = 15

Median: middle value

Sort the list in numerical order:

13, 13, 13, 13, 14, 14, 16, 18, 21

Finding the median

- Input: array of integers

- Output: median of array

Question

- How do you find the median of a list of numbers?

NOOBgrammer says..

Sort the array and pick the middle element

Takes O(nlogn) if you use merge sort

Can we do better?

PROgrammer says..

YES!

Sorting is far more work than what we really need.

We just want the middle element & don't care about relative order of the rest.

Improvement

Finding the median is a specialized or specific version of the more general **Selection Problem**

- Input: List of numbers, integer *k*
- Output: The *kth* smallest element in the list

Selection

- Finding the median is finding $k = n/2$ in the Selection Problem, where n is the size of the list

Selection

Selection

For any number p, imagine splitting the array A into three categories:

- A_L : elements smaller than p
- A_P : elements equal to p (there might be duplicates)
- A_R : elements greater than p

Ring a bell?

Ring a bell?

- Omg guys, it’s quicksort.

Example

x_1							x_8
2	34	1	6	8	5	3	7

8/2 = 4

Is x_1 the median? Are there 4 values less than x_1?

Example

x_1 ... x_8

2	34	1	6	8	5	3	7

x_1 ... x_8

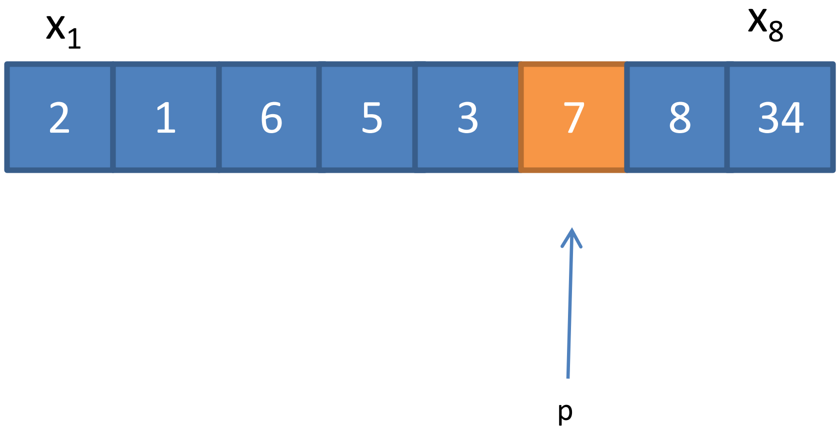
1	2	3	5	6	7	8	34

$k = n/2$

Example

x_1							x_8
2	34	1	6	8	5	3	7

We will divide the array into 3 parts.

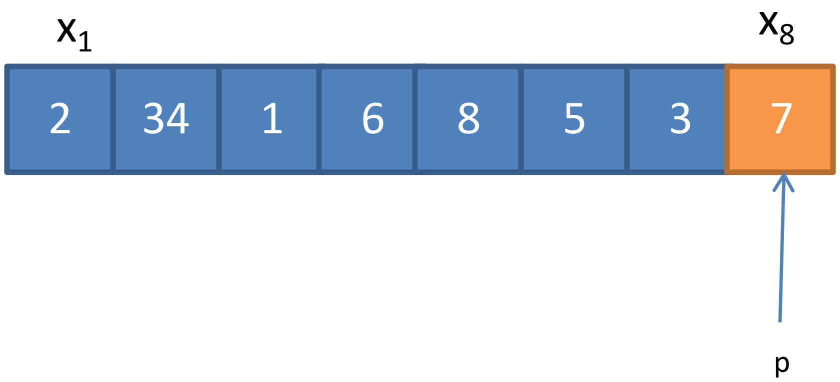


A_p = [7]
A_L = [2, 1, 6, 5, 3]
A_R = [8, 34]

- 7 is 6th the smallest number. But we need the 4th smallest number.
- So median will not be in A_R.

select([2, 1, 6, 5, 3, 7, 8, 34], 4)   pivot = 7, k = 4

split_array(array A):
    L, P, R = [ ], [ ], [ ]
    pivot = A[N]
    for i = 1 to N:
        if A[i] < pivot:
            add A[i] to L
        else if A[i] == pivot:
            add A[i] to P
        else if A[i] > pivot:
            add A[i] to R
    return L, P, R



select([2, 1, 6, 5, 3, 7, 8, 34], 4)   pivot = 7, k = 4

select(array A, integer K):

AL, AP, AR = split_array(A)

NL, NP, NR = lengths of AL, AP, AR

if K ≤ NL:

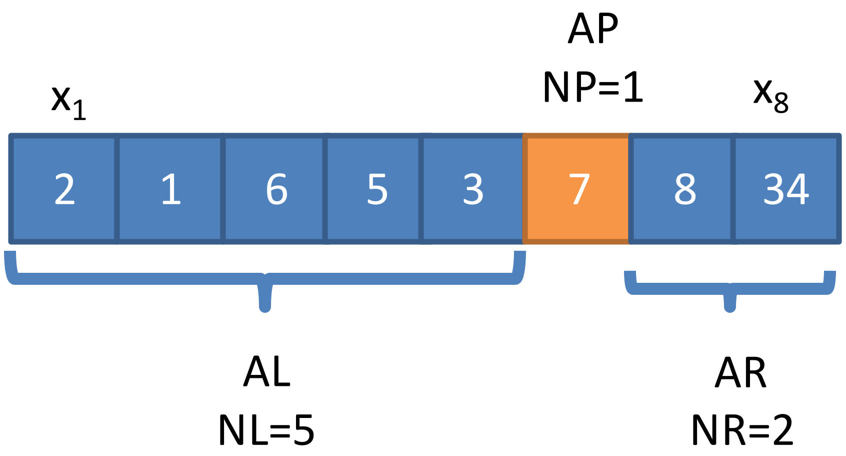
return select(AL, K)

else if NL < K ≤ (NL + NP):

return Ap[0] # pivot

else if K > (NL + NP):

return select(AR, K - (NL + NP))



select([2, 1, 6, 5, 3], 4)   pivot = 3, k = 4

select(array A, integer K):

AL, AP, AR = split_array(A)

NL, NP, NR = lengths of AL, AP, AR

if K ≤ NL:

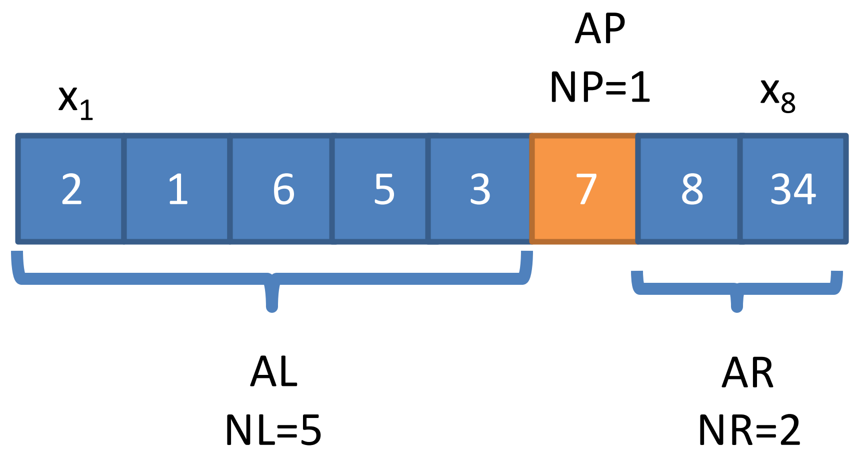
return select(AL, K)

else if NL < K ≤ (NL + NP):

return Ap[0]

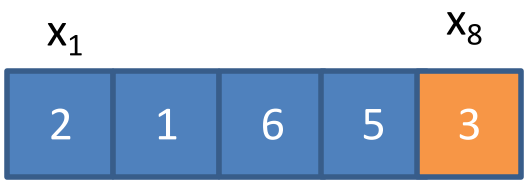
else if K > (NL + NP):

return select(AR, K - (NL + NP))

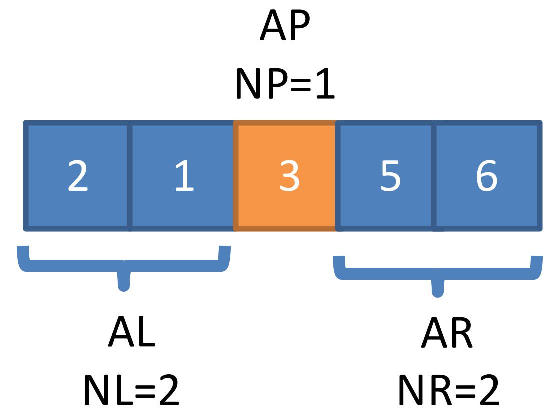


select([2, 1, 6, 5, 3], 4)   pivot = 3, k = 4

```
split_array(array A):
        L, P, R = [ ], [ ], [ ]
        pivot = A[N]
        for i = 1 to N:
                if A[i] < pivot:
                                add A[i] to L
                else if A[i] == pivot:
                                add A[i] to P
                else if A[i] > pivot:
                                add A[i] to R
        return L, P, R
```



select([2, 1, 6, 5, 3], 4)   pivot = 3, k = 4



select(array A, integer K):

AL, AP, AR = split_array(A)

NL, NP, NR = lengths of AL, AP, AR

if $K \leq NL$:

return select(AL, K)

else if $NL < K \leq (NL + NP)$:

return Ap[0]

else if $K > (NL + NP)$:

return select(AR, K - (NL + NP))

select([5, 6], 1)   pivot = 6, k = 1

select([5, 6], 1)   pivot = 6, k = 1


```
split_array(array A):
    L, P, R = [ ], [ ], [ ]
    pivot = A[N]
    for i = 1 to N:
        if A[i] < pivot:
            add A[i] to L
        else if A[i] == pivot:
            add A[i] to P
        else if A[i] > pivot:
            add A[i] to R
    return L, P, R
```

select([5, 6], 1) pivot = 6, k = 1

select(array A, integer K):

AL, AP, AR = split_array(A)

NL, NP, NR = lengths of AL, AP, AR

if K ≤ NL:

return select(AL, K)

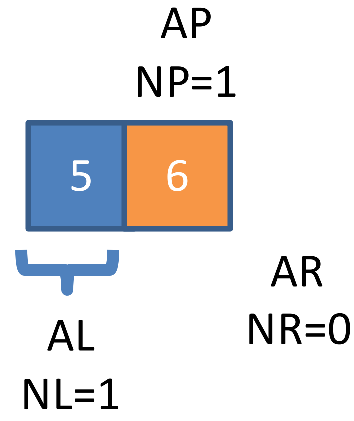
else if NL < K ≤ (NL + NP):

return Ap[0]

else if K > (NL + NP):

return select(AR, K - (NL + NP))

select([5], 1)



select([5], 1)   pivot = 5, k = 1

```
select(array A, integer K):
    AL, AP, AR = split_array(A)
    NL, NP, NR = lengths of AL, AP, AR
    if K ≤ NL:
        return select(AL, K)
    else if NL < K ≤ (NL + NP):
        return Ap[0]
    else if K > (NL + NP):
        return select(AR, K - (NL + NP))
```

return 5

AP
NP=1

5

AL
NL=0

AR
NR=0

Selection

Selection

- After determining on which sublist the item is located, recurse on the appropriate sublist

Selection

- After determining on which sublist the item is located, recurse on the appropriate sublist
- The effect of the split is to shrink the number of elements from $|A|$ to at most $\max(|AL|, |AR|)$

Critical Question

How do we choose *p*?

- *p* should be picked quickly and should shrink the array substantially (ideally, half)

Answer

- We pick p randomly from S, same reason as in Quick Sort

Efficiency

- $T(n) \leq (n-1) \ + T(n\text{-}1)$
- $T(n) \leq (n-1) \ + (n\text{-}2) + T(n\text{-}2)$
- $T(n) \leq (n-1) \ + (n\text{-}2) + (n\text{-}3) + T(n\text{-}3)$
- $T(n) \leq (n-1) \ + (n\text{-}2) + (n\text{-}3) + (n\text{-}3) + T(n\text{-}4)$
- $T(n) \leq (n-1) \ + (n\text{-}2) + (n\text{-}3) + (n\text{-}3) + \ldots + 1$

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

Efficiency

- How can we speed up the running time?

- Quicksort algorithm will speed up via randomization.

Efficiency Analysis

- RT of algorithm depends on the random choices of p
- Possible to have bad luck and keep picking p to be the largest / smallest element of the array, shrinking the array only one element at a time

Efficiency Analysis

- **Worst Case:** $O(n^2)$, but extremely unlikely to occur
- **Best case:** when random p chosen splits the array perfectly in half, RT: $O(n)$

  1 iteration * O(N) work per iteration → O(N)

  One iteration only → find in AP; but uses O(N) for split_array

Efficiency Analysis

- So where in the spectrum of $O(n)$ to $O(n^2)$ does average case lie?
- Very close to Best Case: **$O(n)$**

References

- Algorithm Design, Kleinberg & Tardos
- MIT Opencourseware

Assignment

- Find the median using Selection (DAC) where v = 2nd element
  A = [7, 8, 11, 3, 9, 1, 4, 10, 3, 8, 6, 12]

- Count and list (in ascending order) the inversions in

  A = [1, 5, 4, 8, 10, 2, 6, 9, 3, 7]

  using merge sort.