Analysis of Algorithms II

Recitation Week-7

2022-2023 Spring

Q1)Count the number of inversions for the given numbers; 1, 20, 6, 7, 5, 8, 11, 3. Use divide and conquer (DnC) approach.

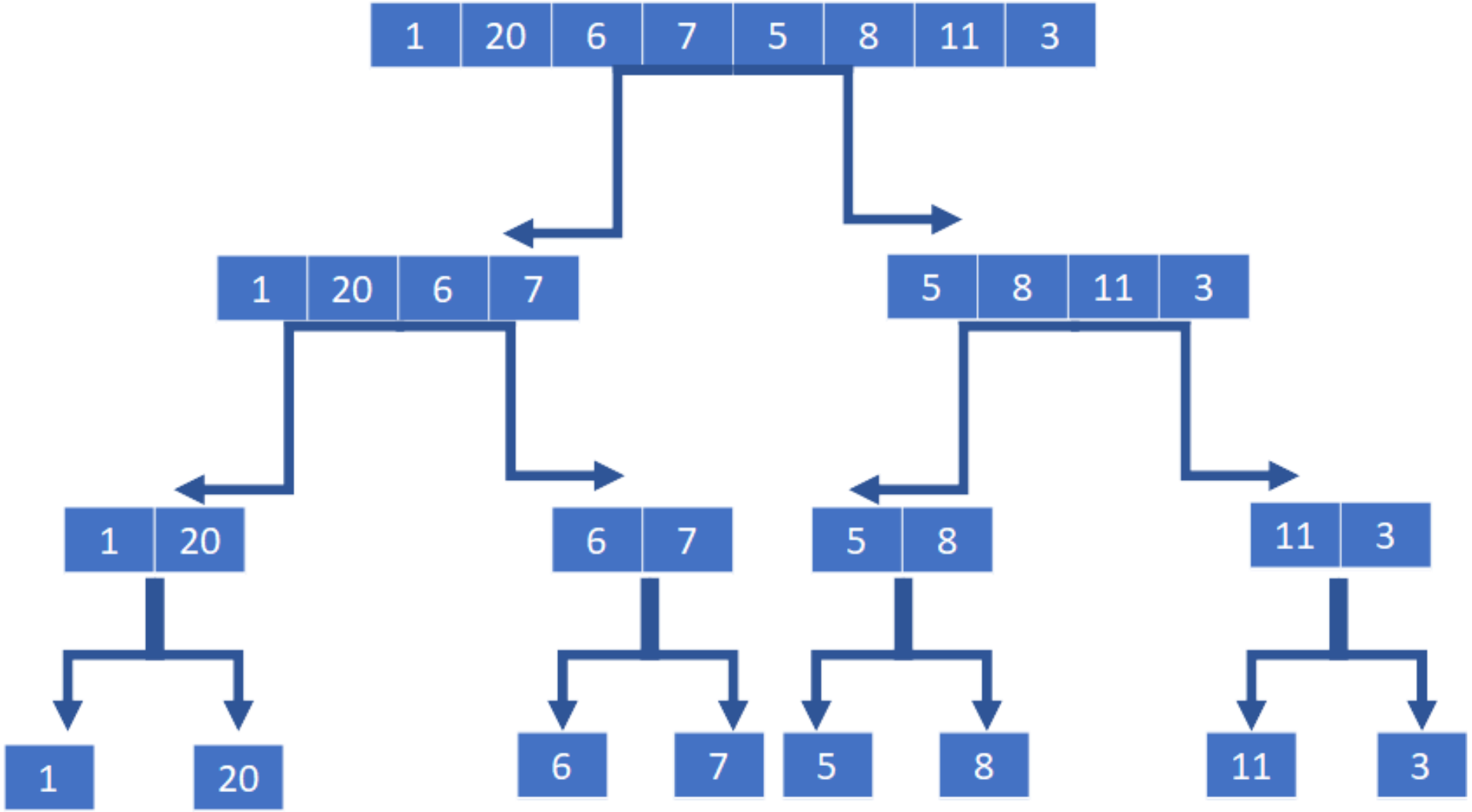
Q1)Count the number of inversions for the given numbers; 1, 20, 6, 7, 5, 8, 11, 3. Use divide and conquer (DnC) approach.

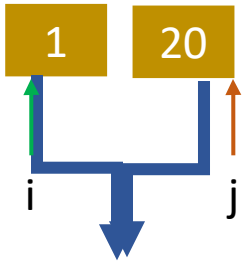
Direct approach -> Modify Bubble-Sort

DnC approach -> Modify Merge-Sort

The numerical example is taken from https://www.youtube.com/watch?v=owZhw-A0yWE

Partition



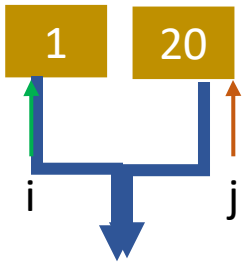


Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed



6 7

5 8

11 3

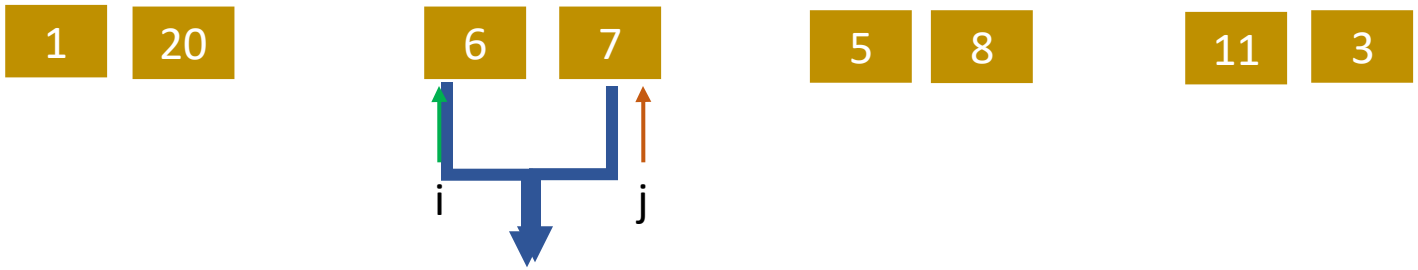
Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed

1 20



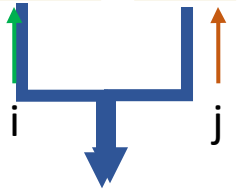
Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed

1 20 6 7 5 8 11 3



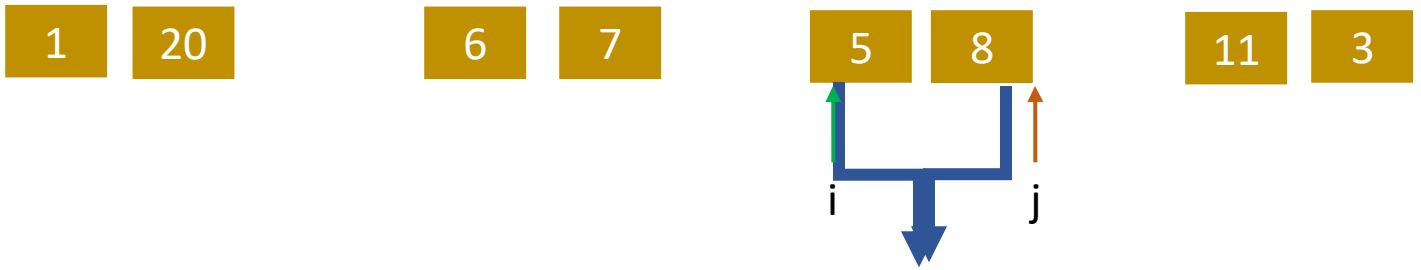
Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed

1 20 6 7

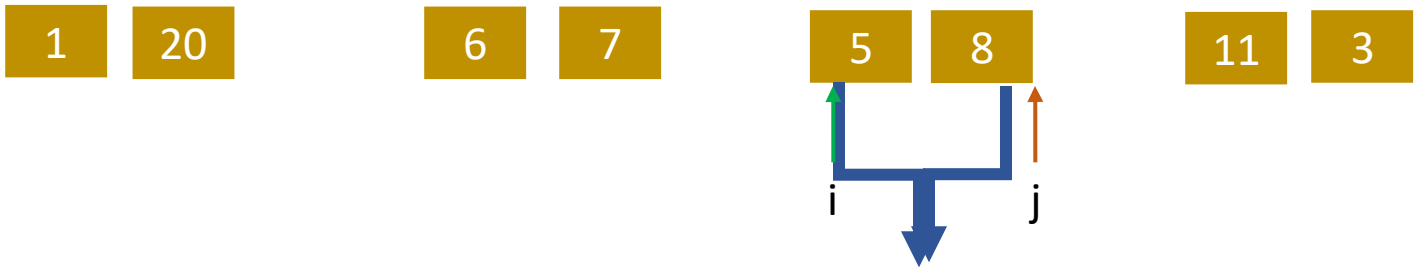


Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed



Merging step

inversion_count = 0

is arr[i] > arr[j] ?

NO → no inversion needed

Merging step

inversion_count = 0

is arr[i] > arr[j] ?

YES → inversion needed!

is arr[i] > arr[j] ?

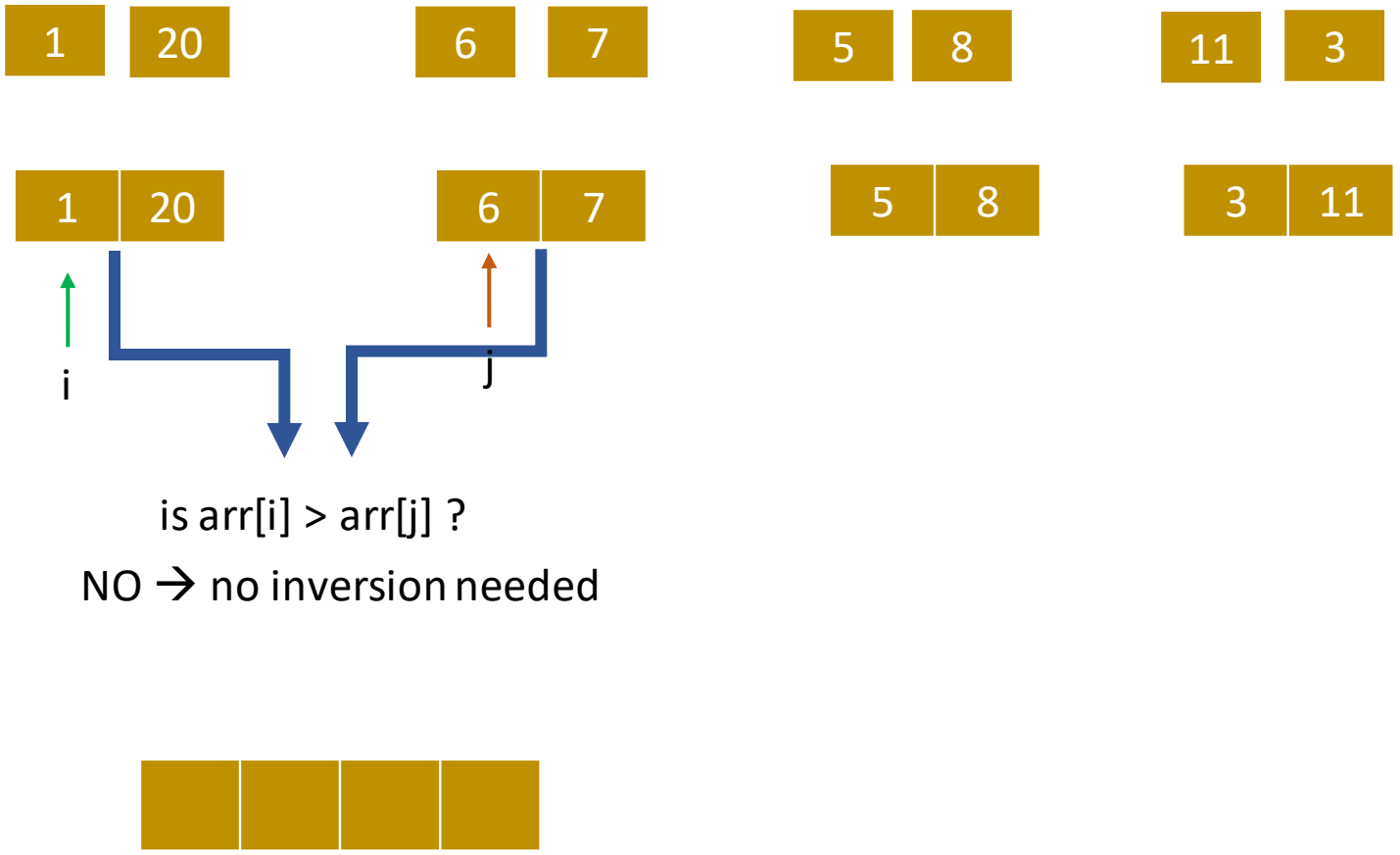
YES → inversion needed!

Merging step

inversion_count = 1

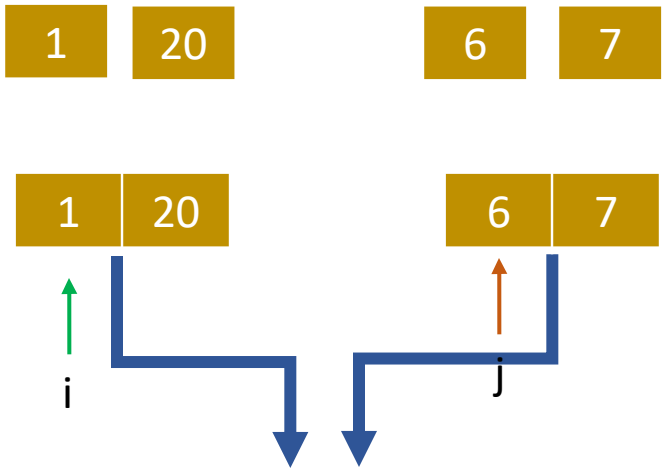
Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 1.



Merging step

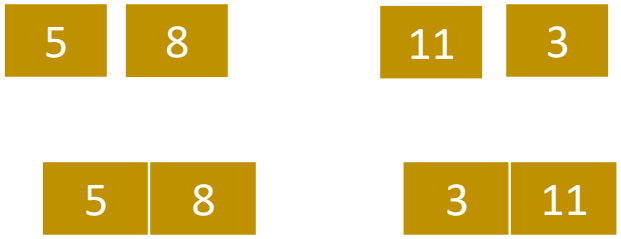
inversion_count = 1



is arr[i] > arr[j] ?

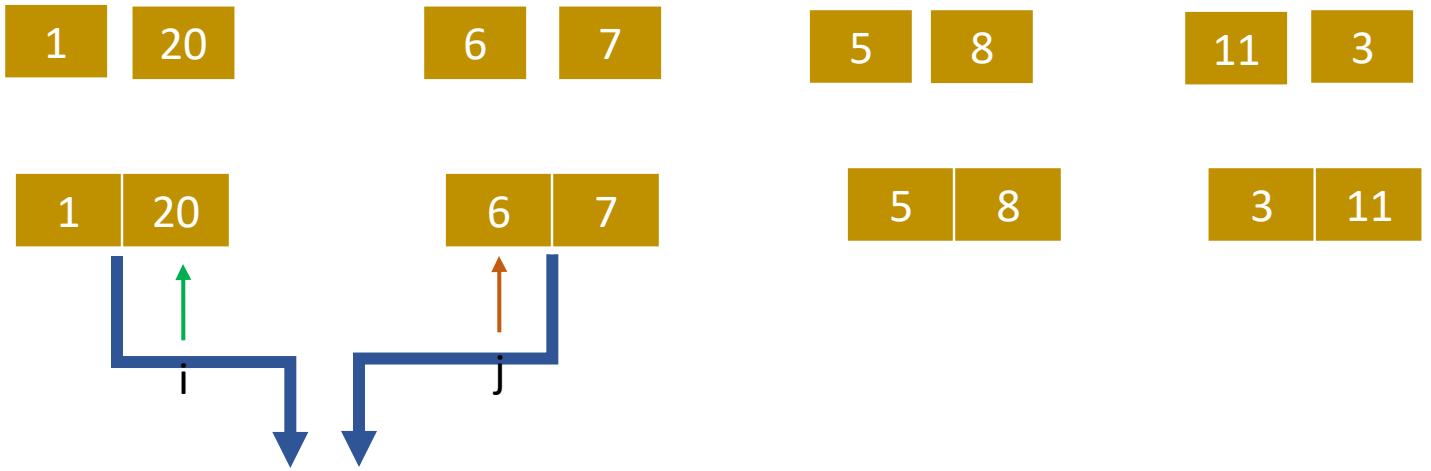
NO → no inversion needed

i = i + 1



Merging step

inversion_count = 1

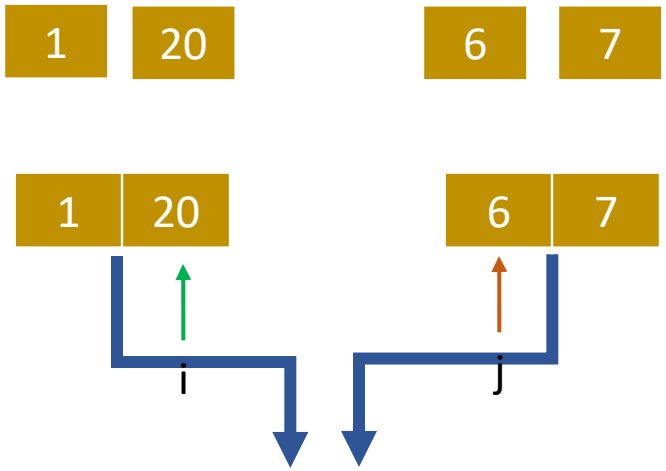


Merging step

inversion_count = 1

is arr[i] > arr[j] ?

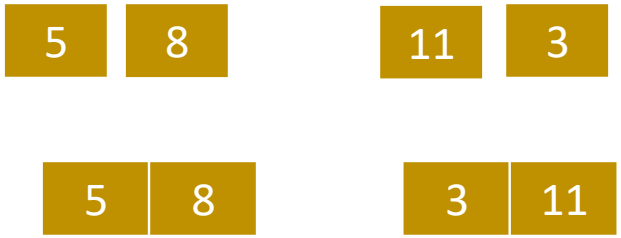
YES → inversion needed!



is arr[i] > arr[j] ?

YES → inversion needed!

j = j + 1

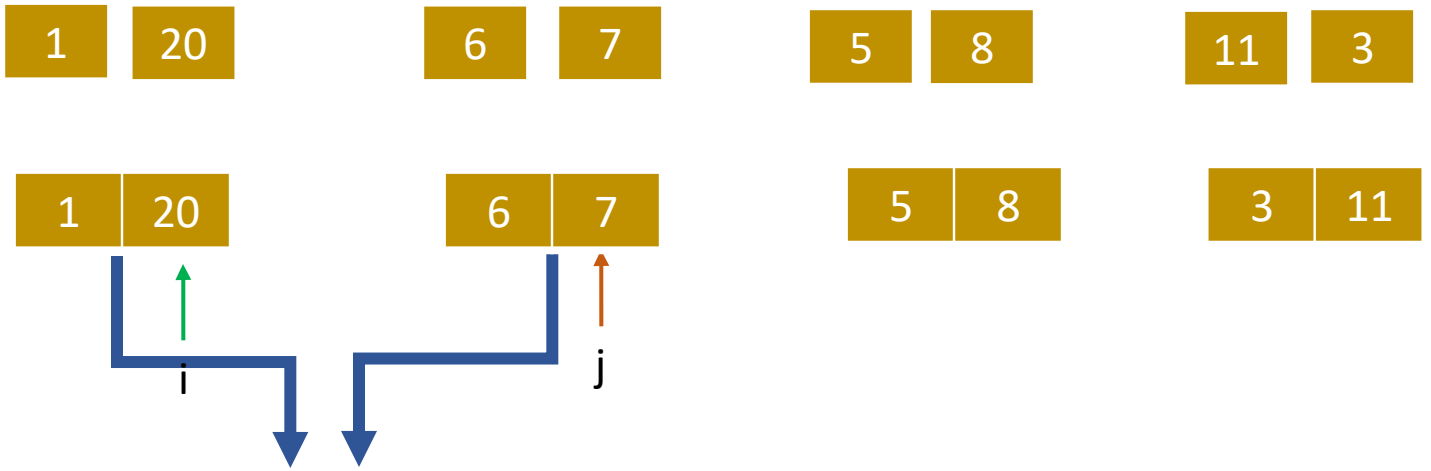


Merging step

inversion_count = 2

Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 1.

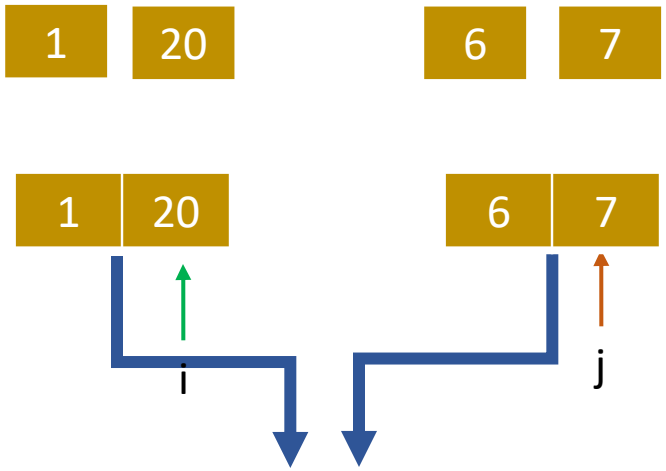


Merging step

inversion_count = 2

is arr[i] > arr[j] ?

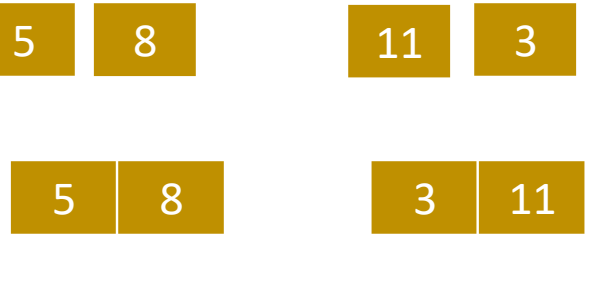
YES → inversion needed!



is arr[i] > arr[j] ?

YES → inversion needed!

j = j + 1

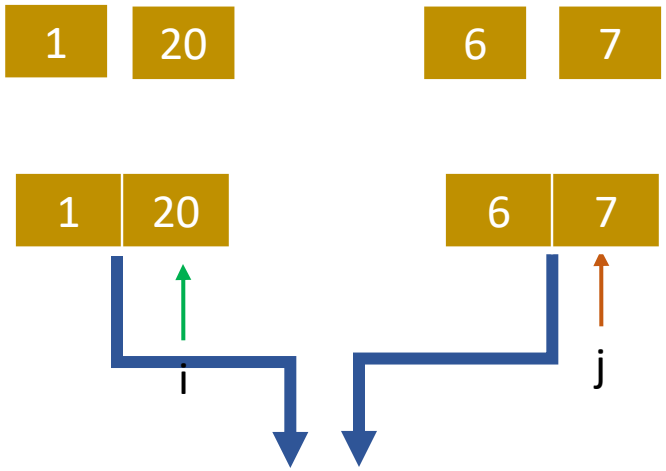


Merging step

inversion_count = 3

Increment this number by how many numbers are there after and including the current value of i.

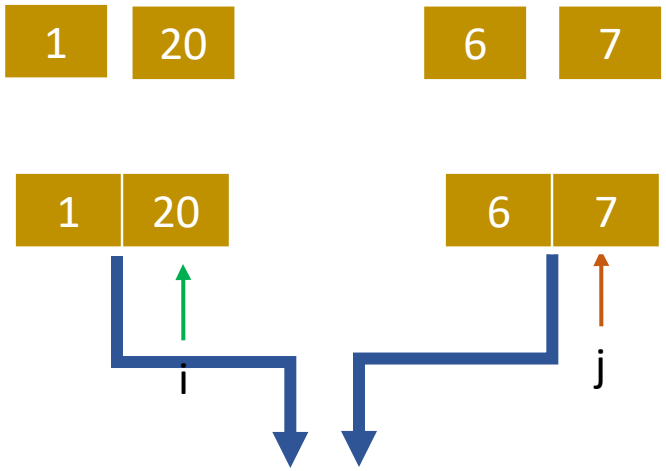
→ Increment by 1.



j is in its maximum value,
put what is left to i.

Merging step

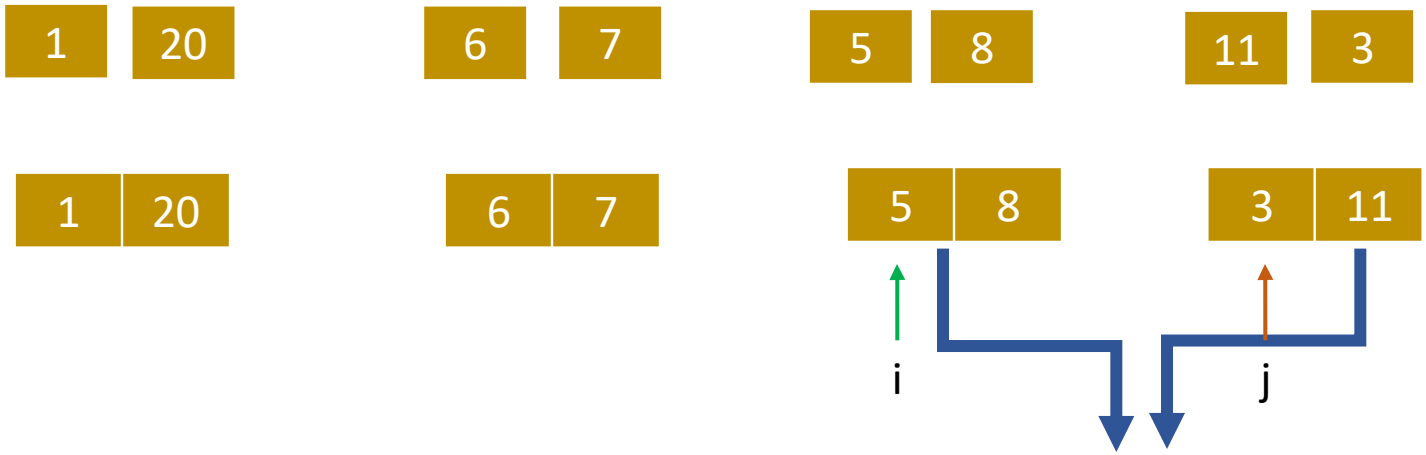
inversion_count = 3



j is in its maximum value,
put what is left to i.

Merging step

inversion_count = 3

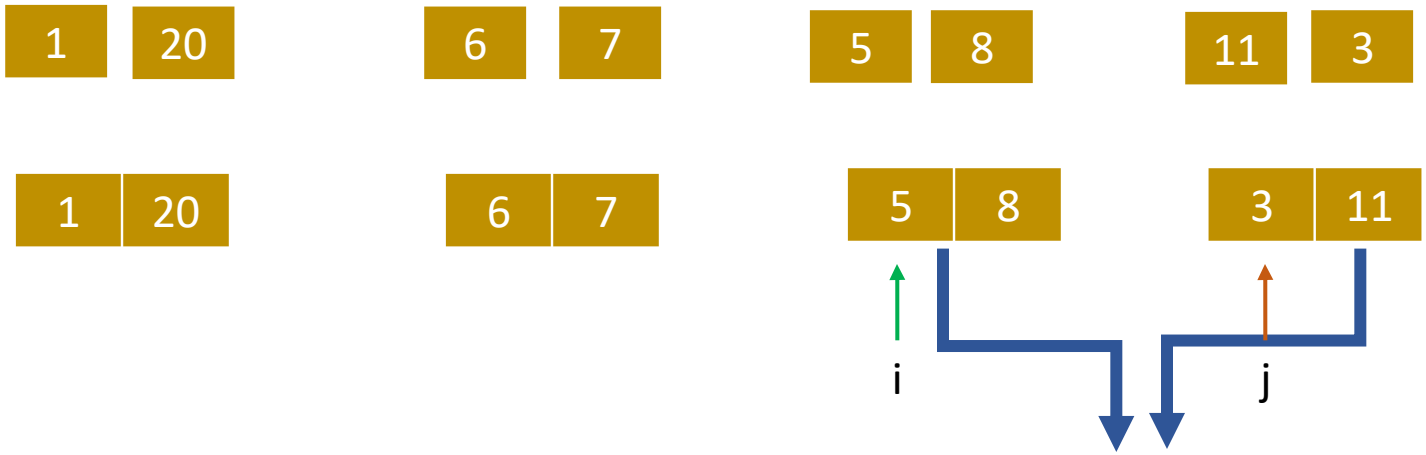


Merging step

inversion_count = 3

is arr[i] > arr[j] ?

YES → inversion needed!

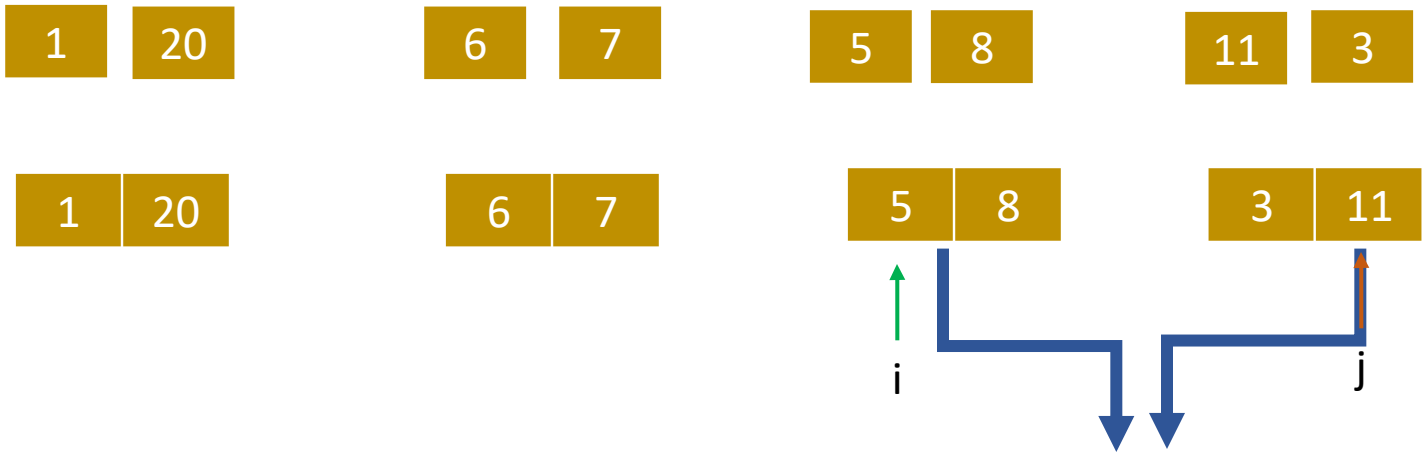


Merging step

inversion_count = 5

Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 2.

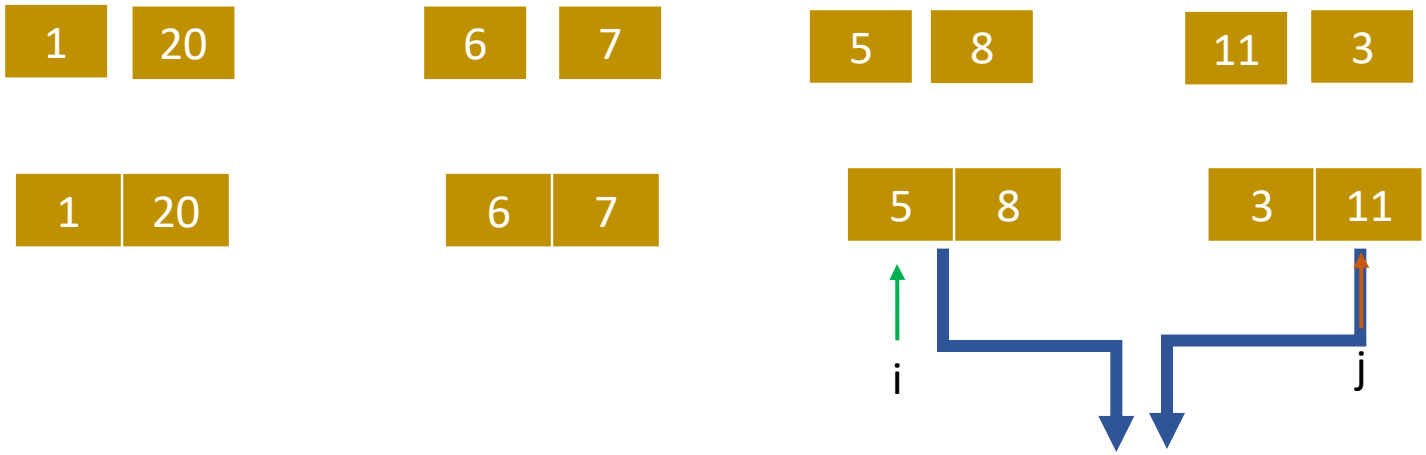


Merging step

inversion_count = 5

is arr[i] > arr[j] ?

NO → no inversion needed



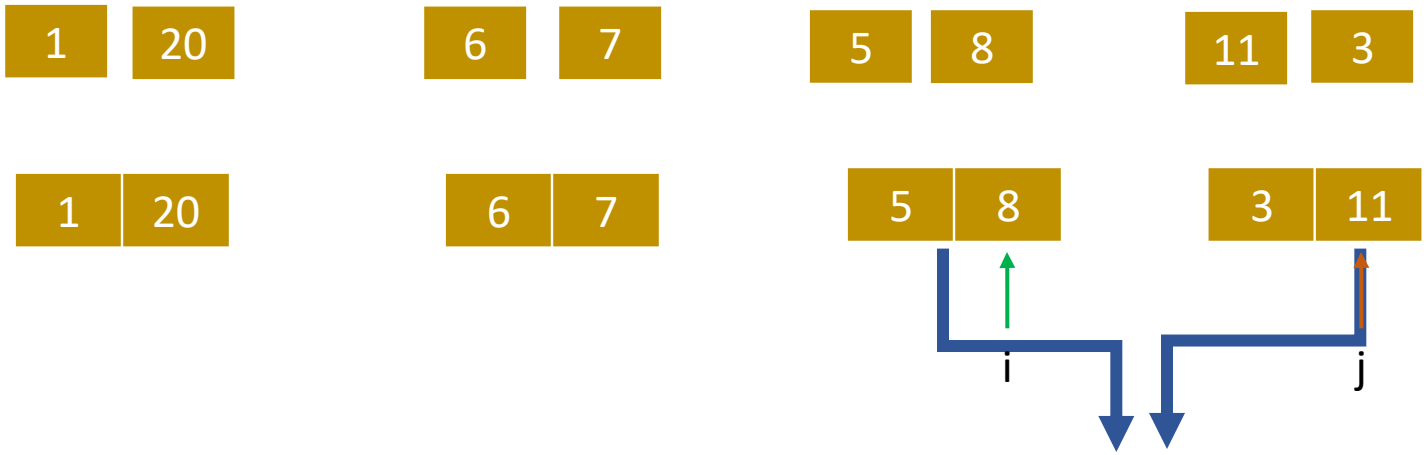
Merging step

inversion_count = 5

is arr[i] > arr[j] ?

NO → no inversion needed

i = i + 1

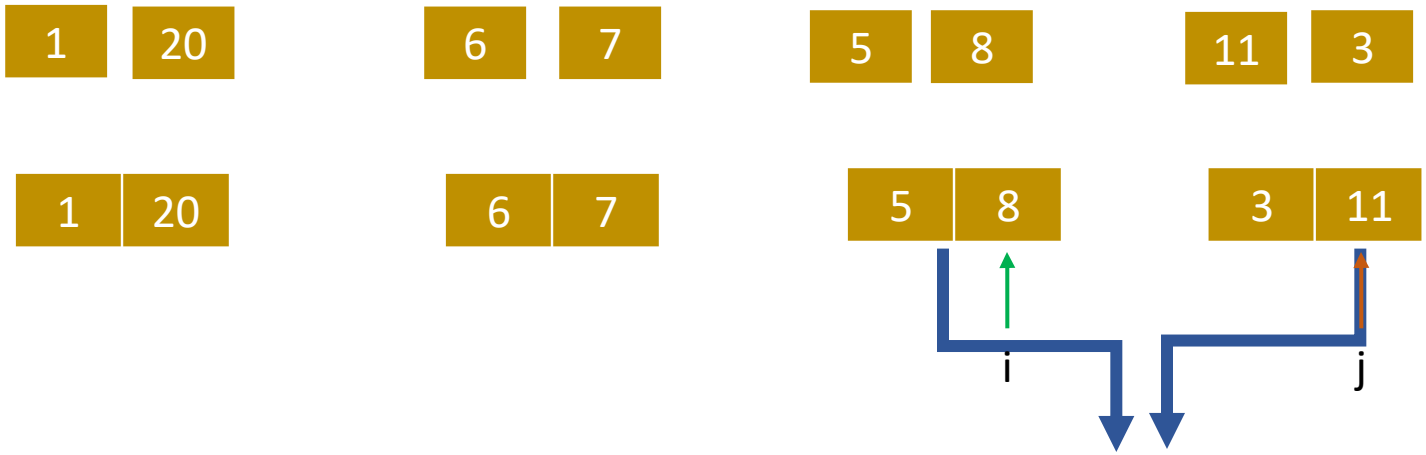


Merging step

inversion_count = 5

is arr[i] > arr[j] ?

NO → no inversion needed



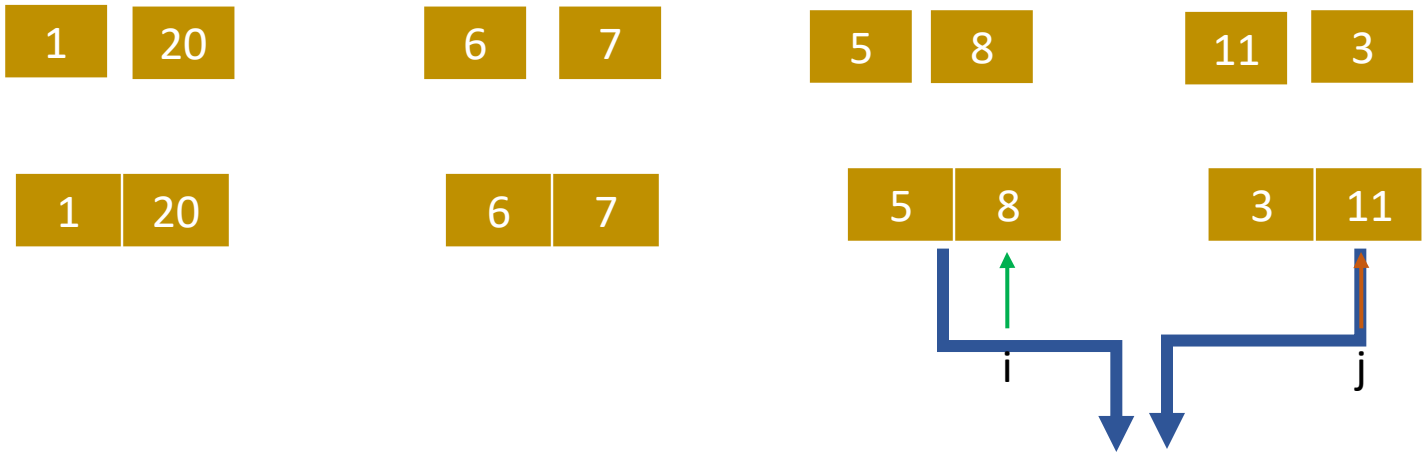
Merging step

inversion_count = 5

is arr[i] > arr[j] ?

NO → no inversion needed

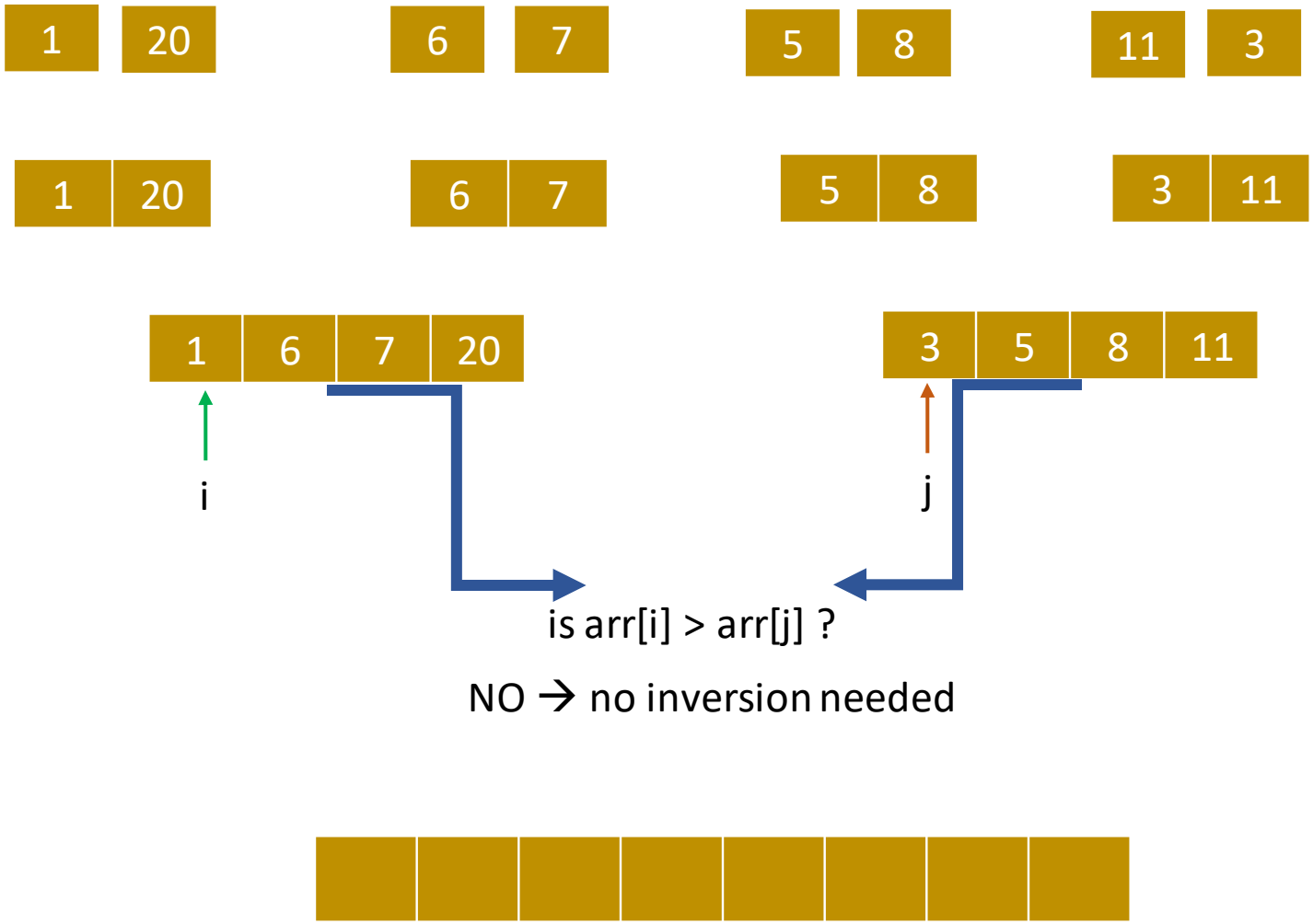
i = i + 1



Merging step

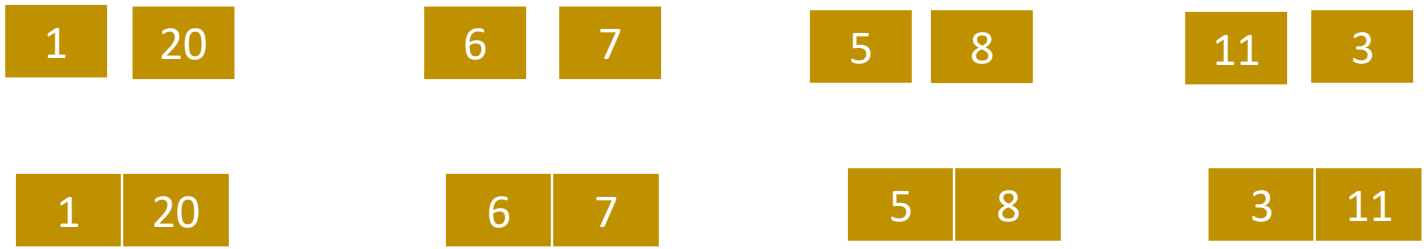
inversion_count = 5

i is in its maximum value,
put what is left to j.



Merging step

inversion_count = 5



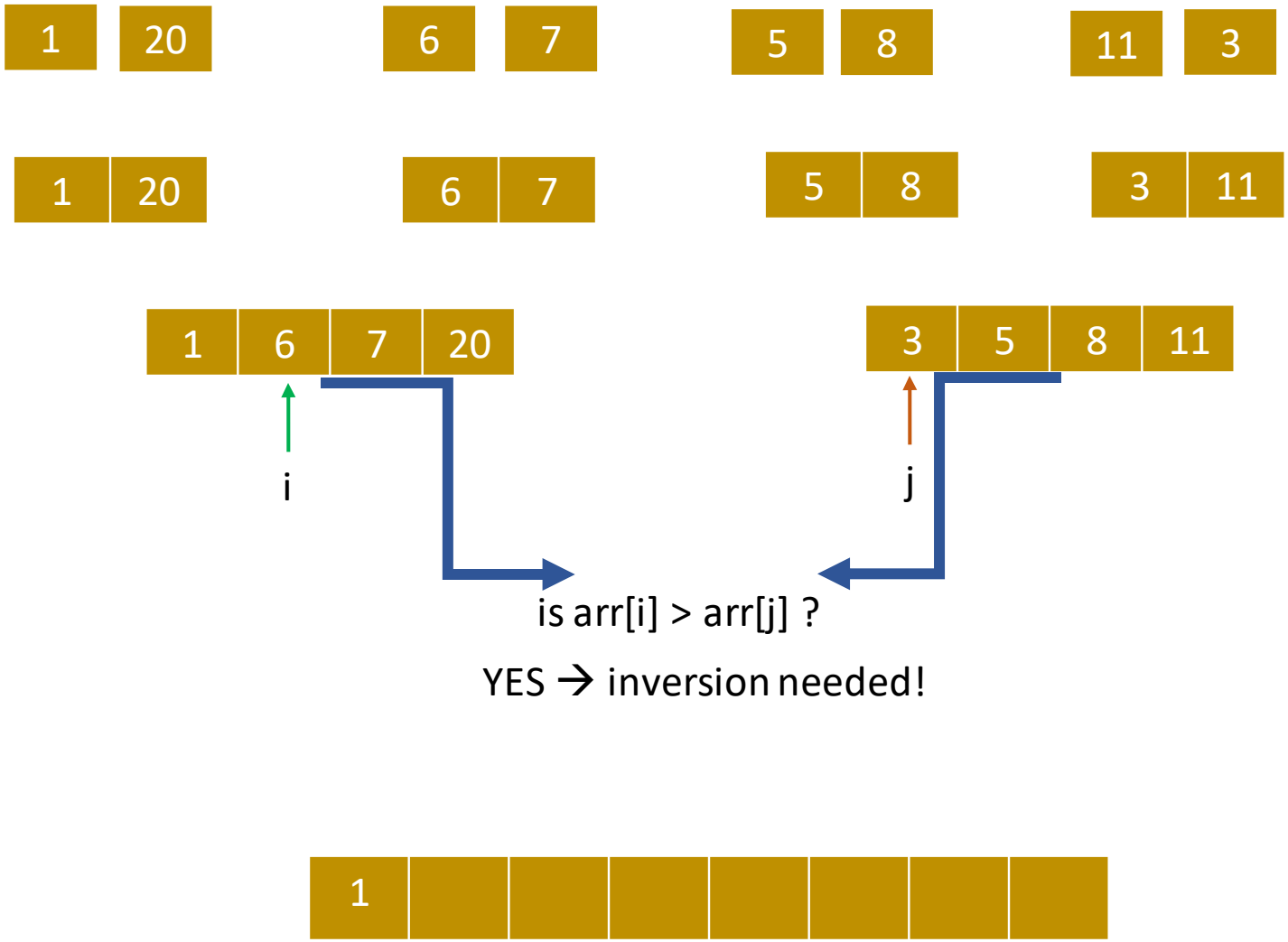
Merging step

inversion_count = 5

is arr[i] > arr[j] ?

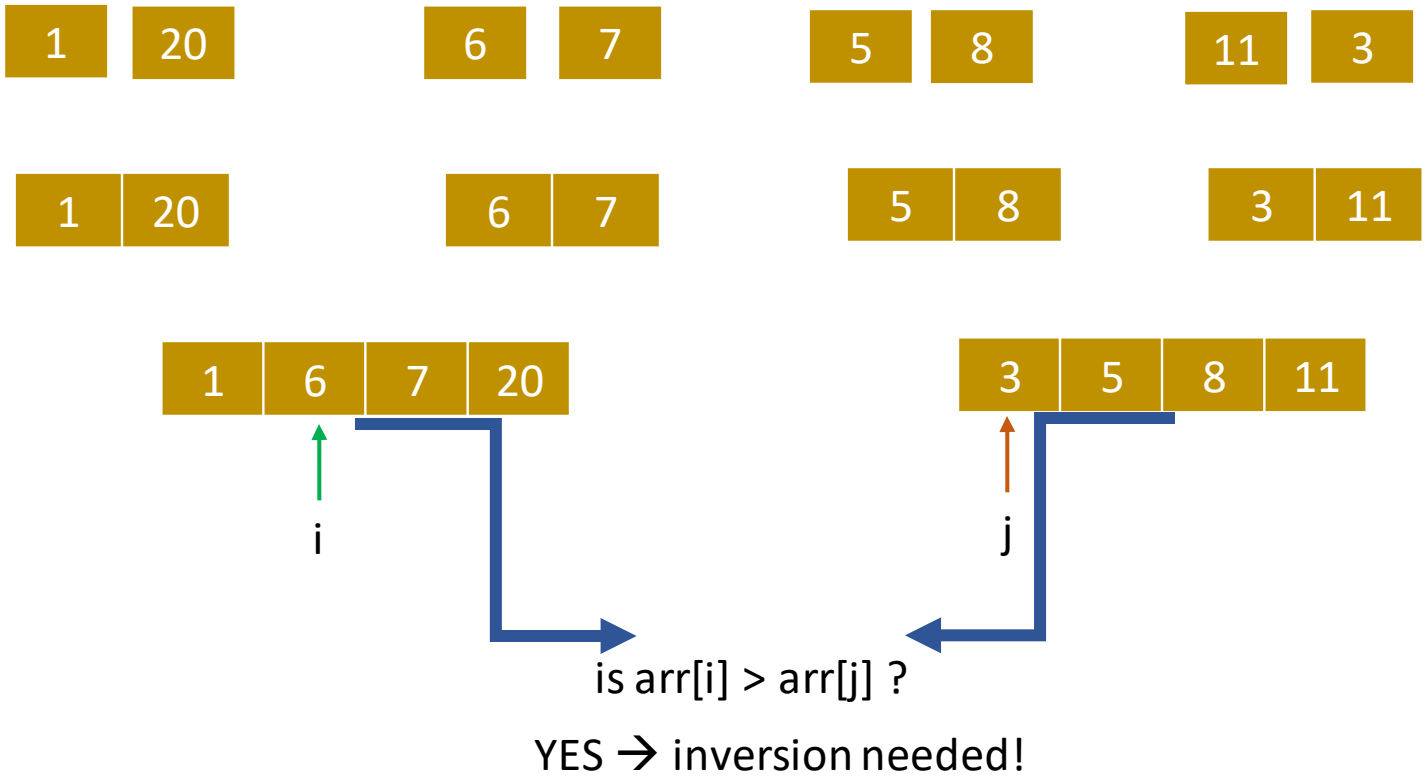
NO → no inversion needed

i = i + 1



Merging step

inversion_count = 5

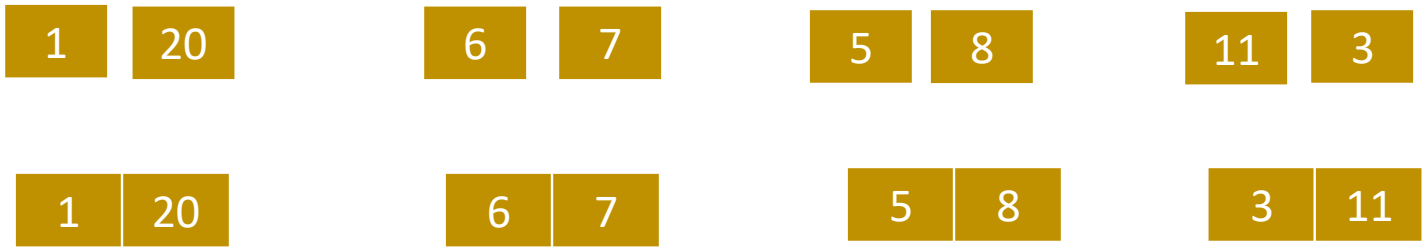


Merging step

inversion_count = 8

Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 3.

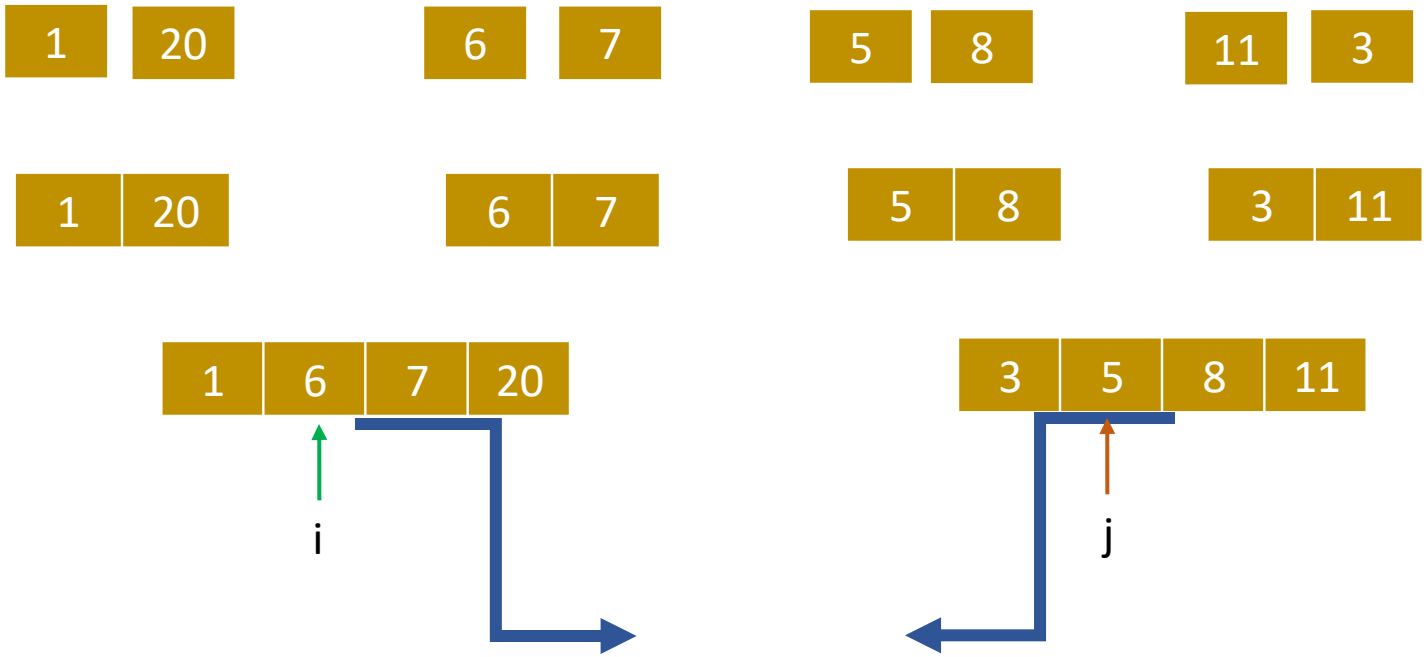


Merging step

inversion_count = 8

is arr[i] > arr[j] ?

YES → inversion needed!

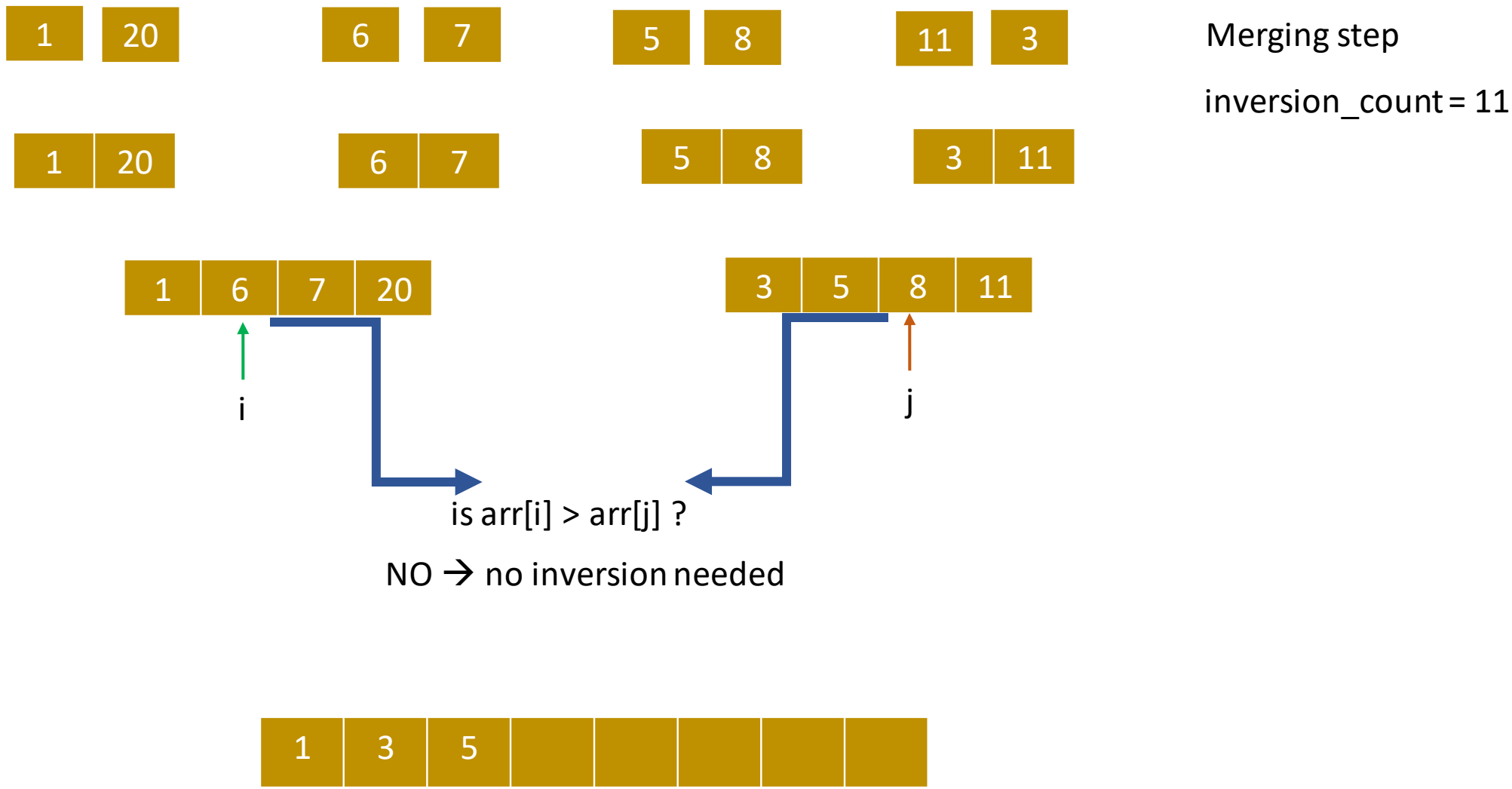


Merging step

inversion_count = 11

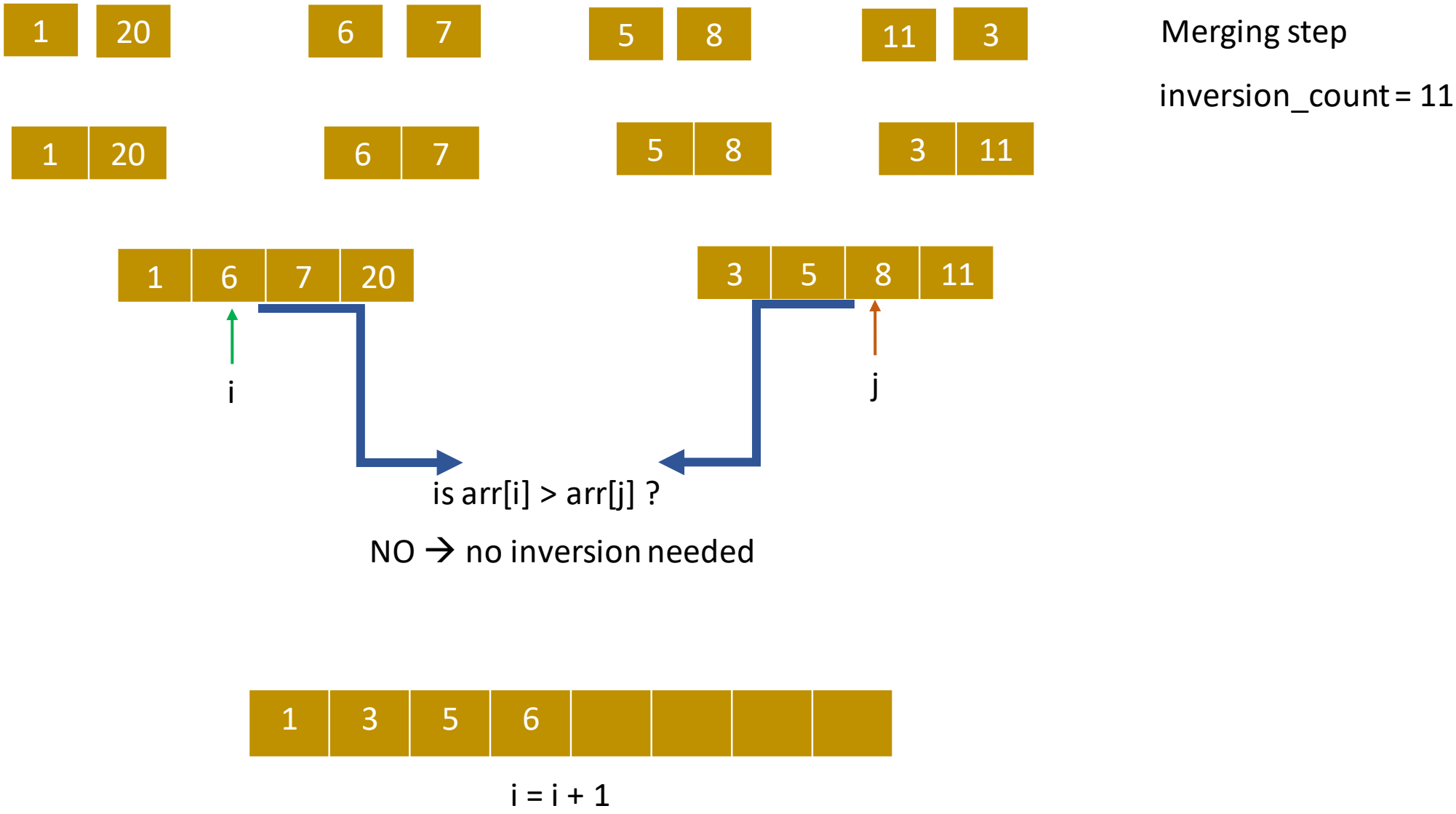
Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 3.



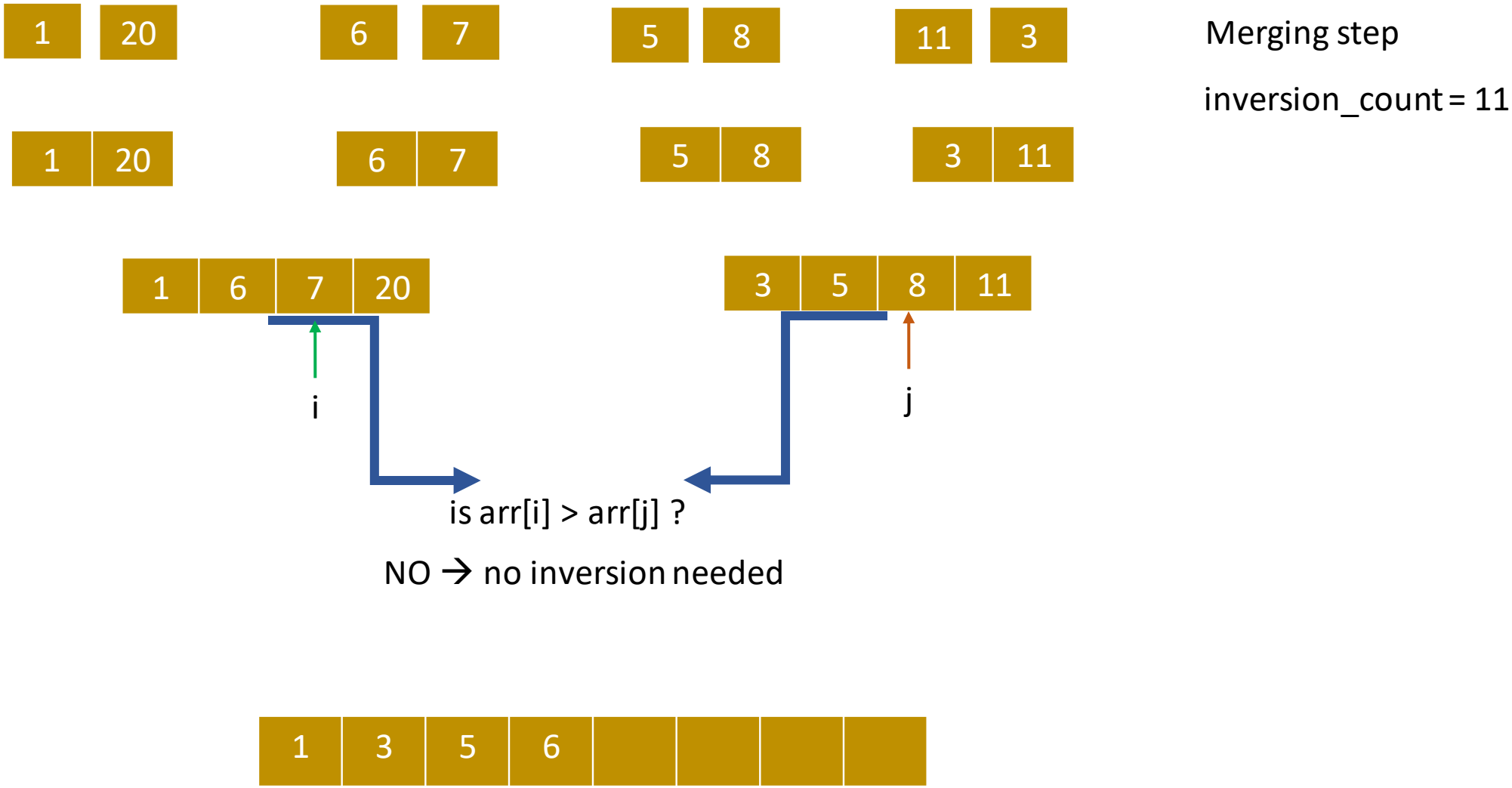
Merging step

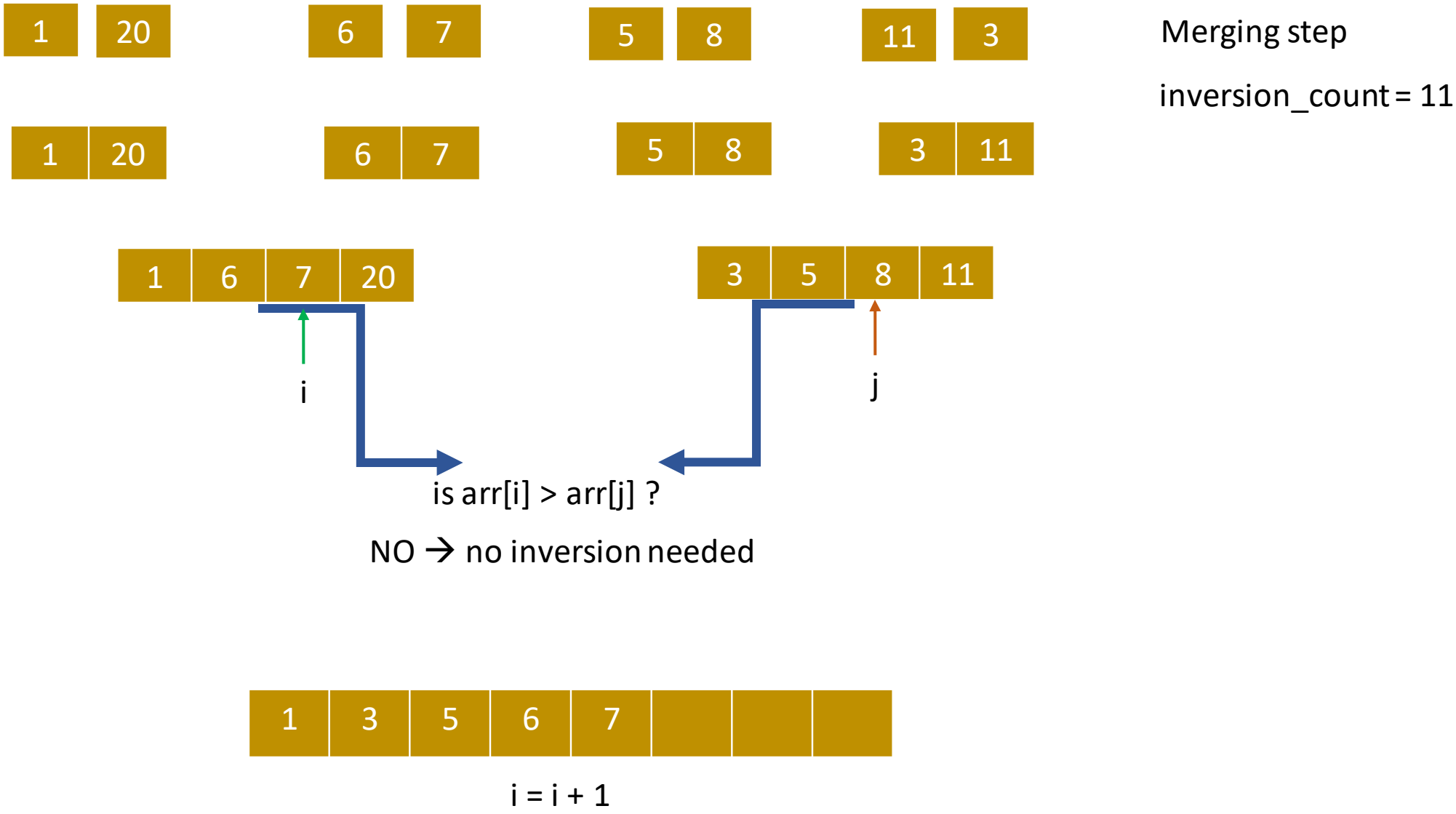
inversion_count = 11

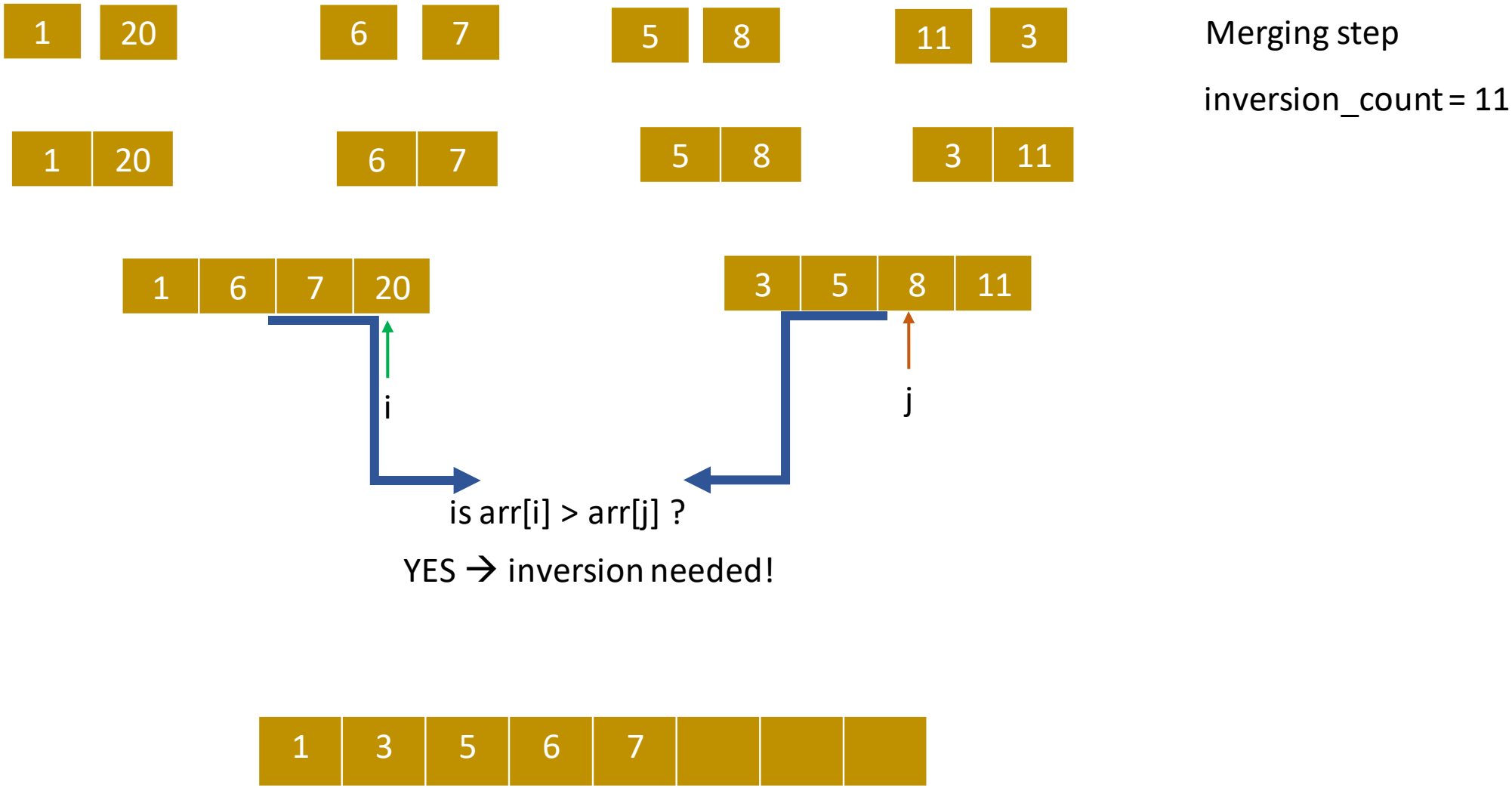


Merging step

inversion_count = 11

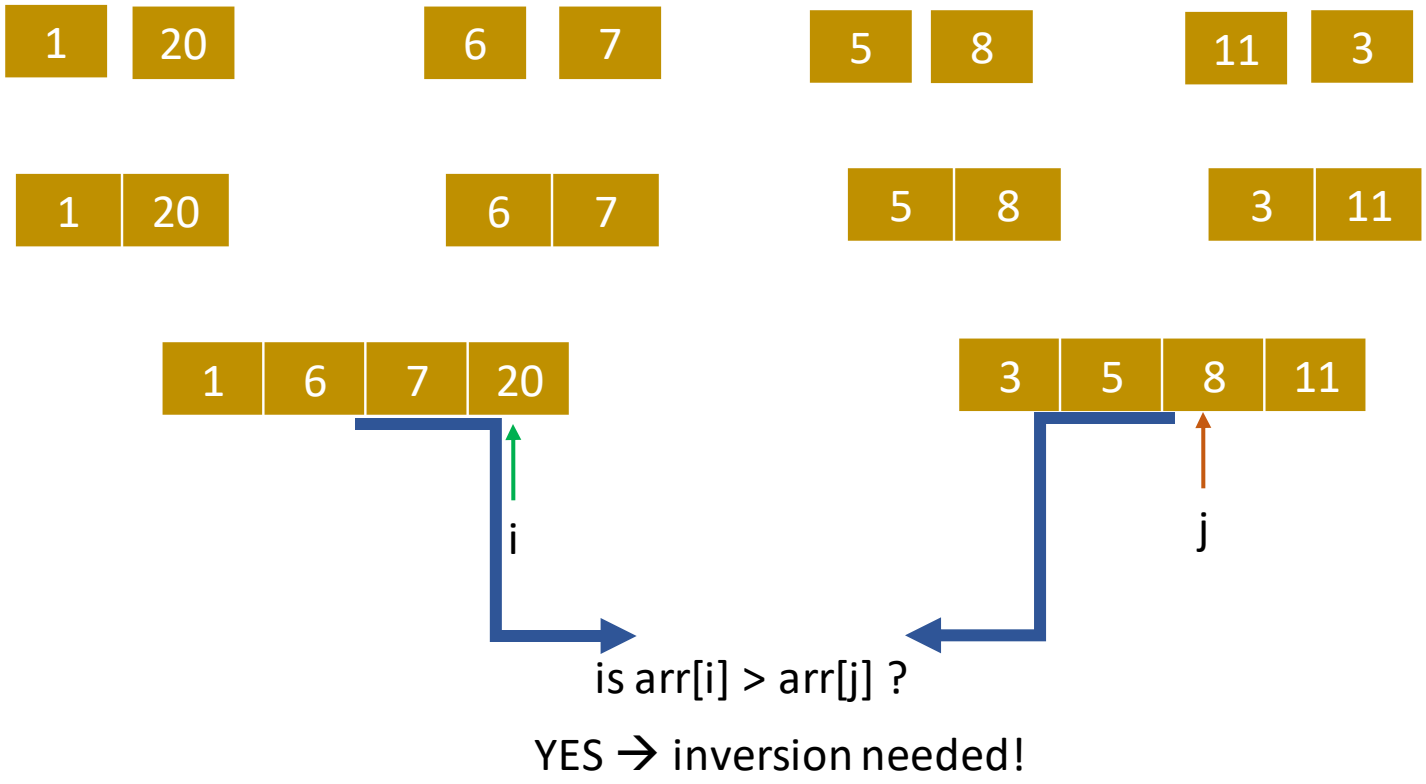

1
20
6
7
5
8
11
3
Merging step
inversion_count = 11
1
20
6
7
5
8
3
11
1
6
7
20
3
5
8
11
i
j
is arr[i] > arr[j] ?
NO → no inversion needed
1
3
5
6


1
20
6
7
5
8
11
3
Merging step
inversion_count = 11
1
20
6
7
5
8
3
11
1
6
7
20
3
5
8
11
i
j
is arr[i] > arr[j] ?
NO → no inversion needed
1
3
5
6
7
i = i + 1



Merging step

inversion_count = 11

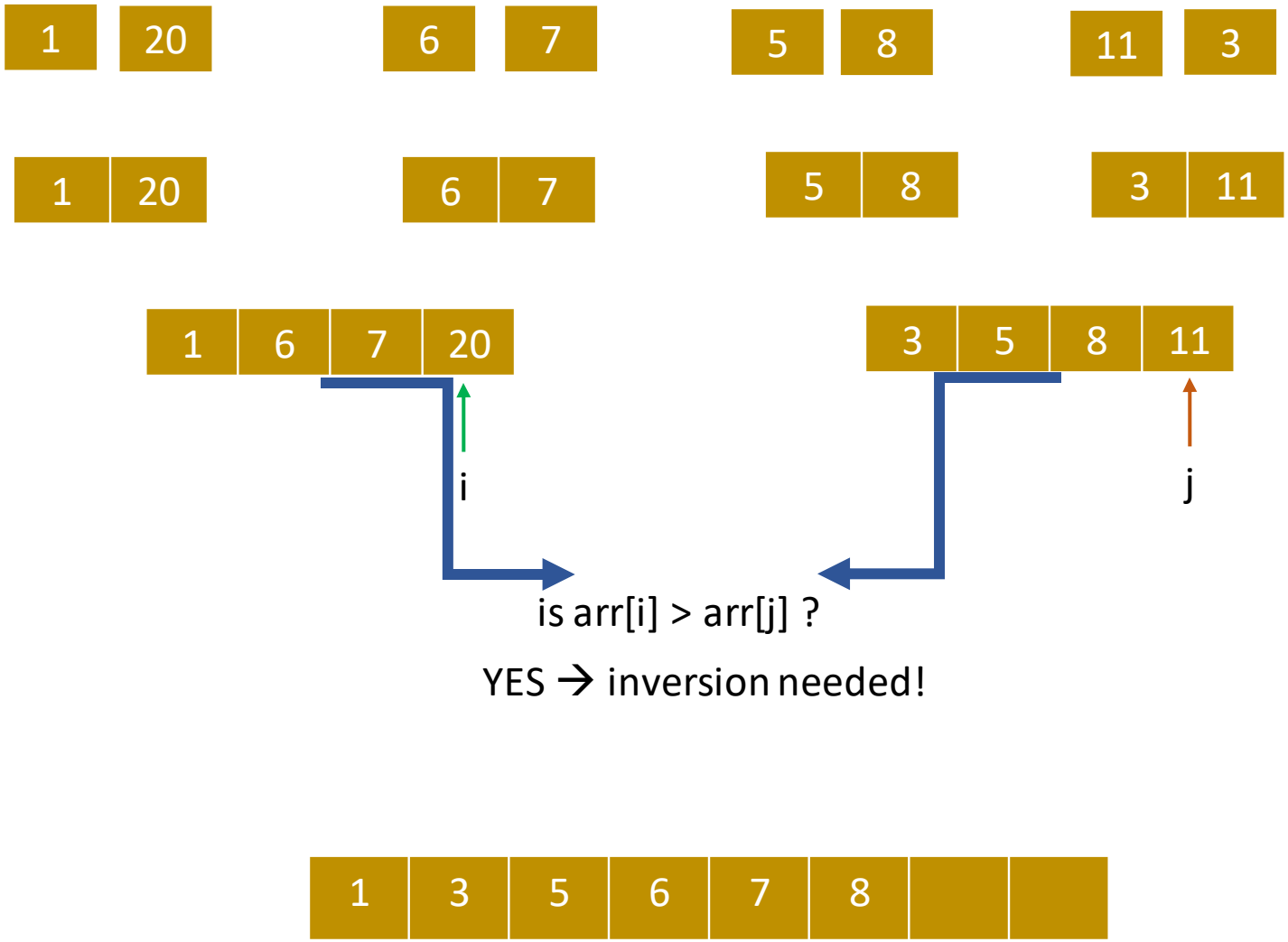


Merging step

inversion_count = 12

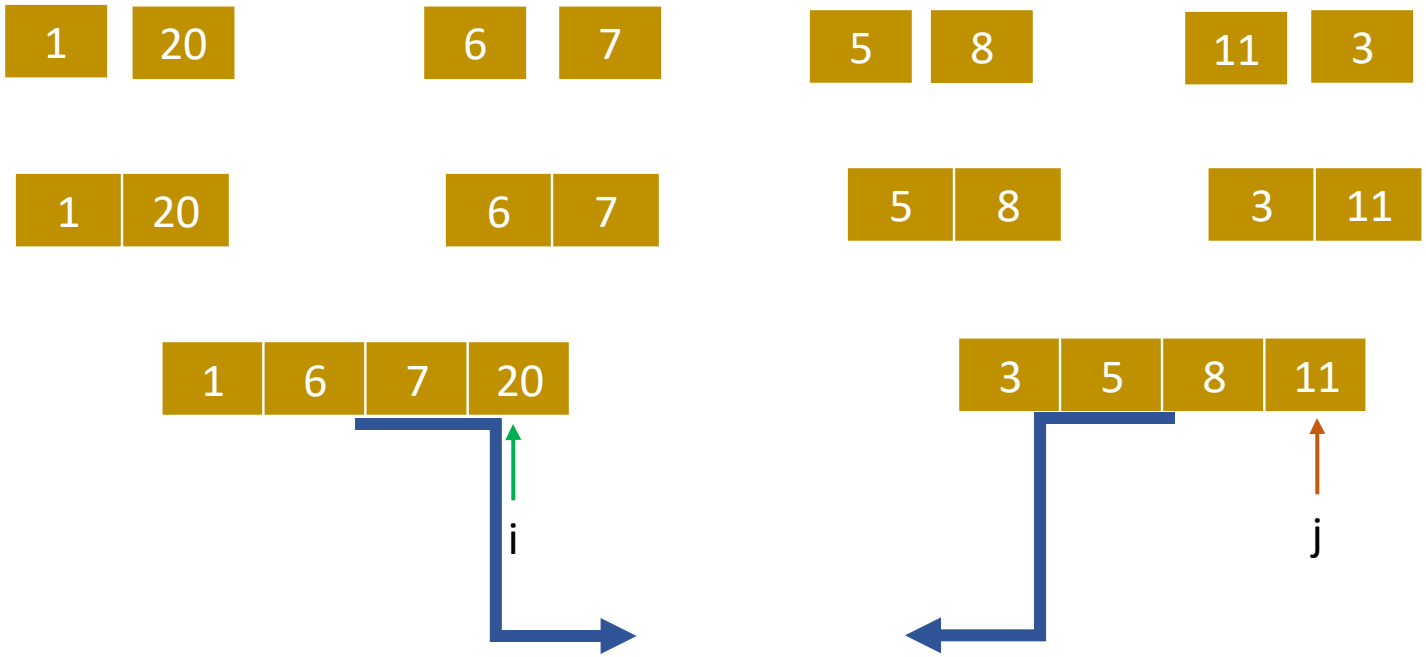
Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 1.



Merging step

inversion_count = 12

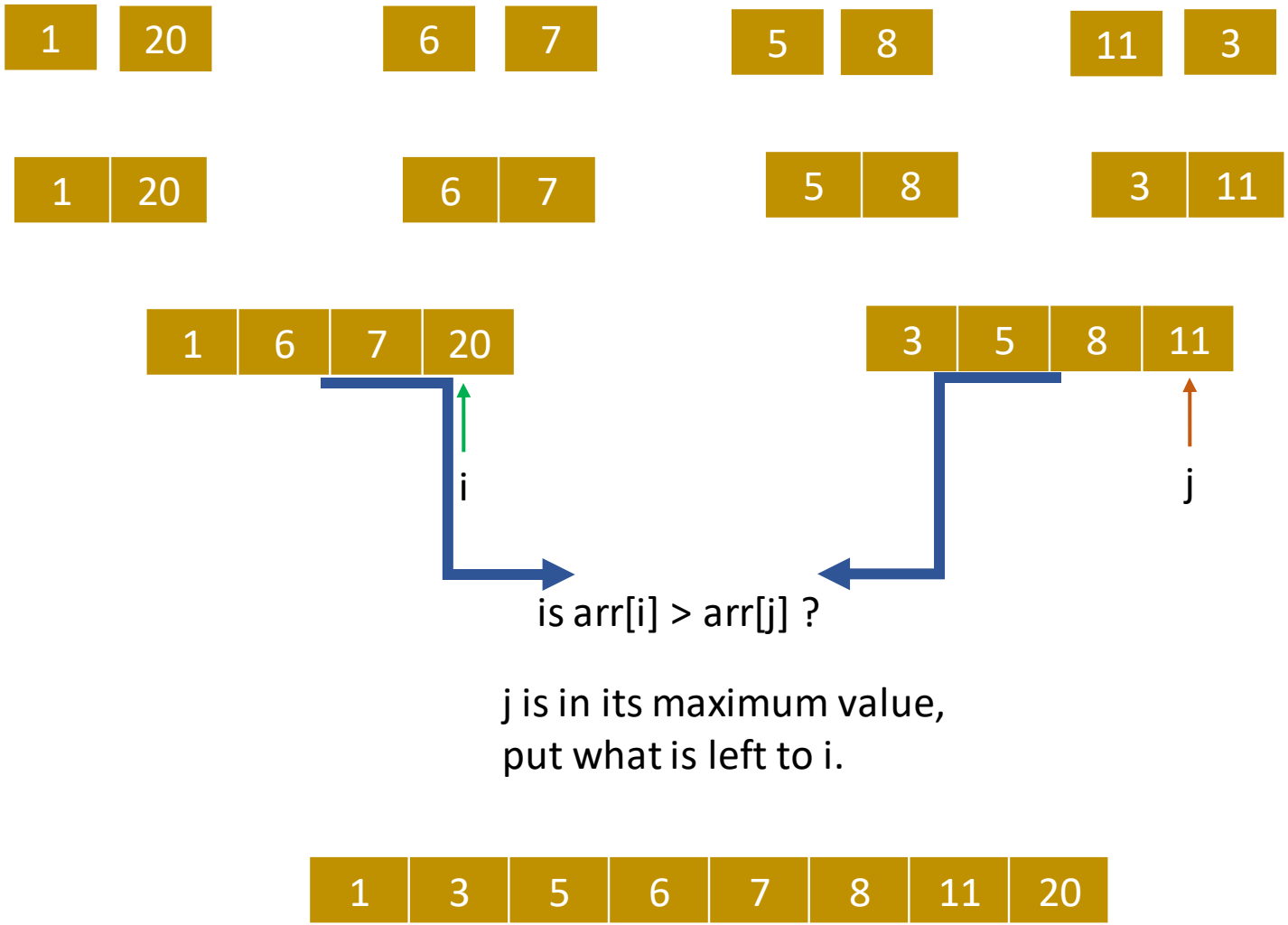


Merging step

inversion_count = 13

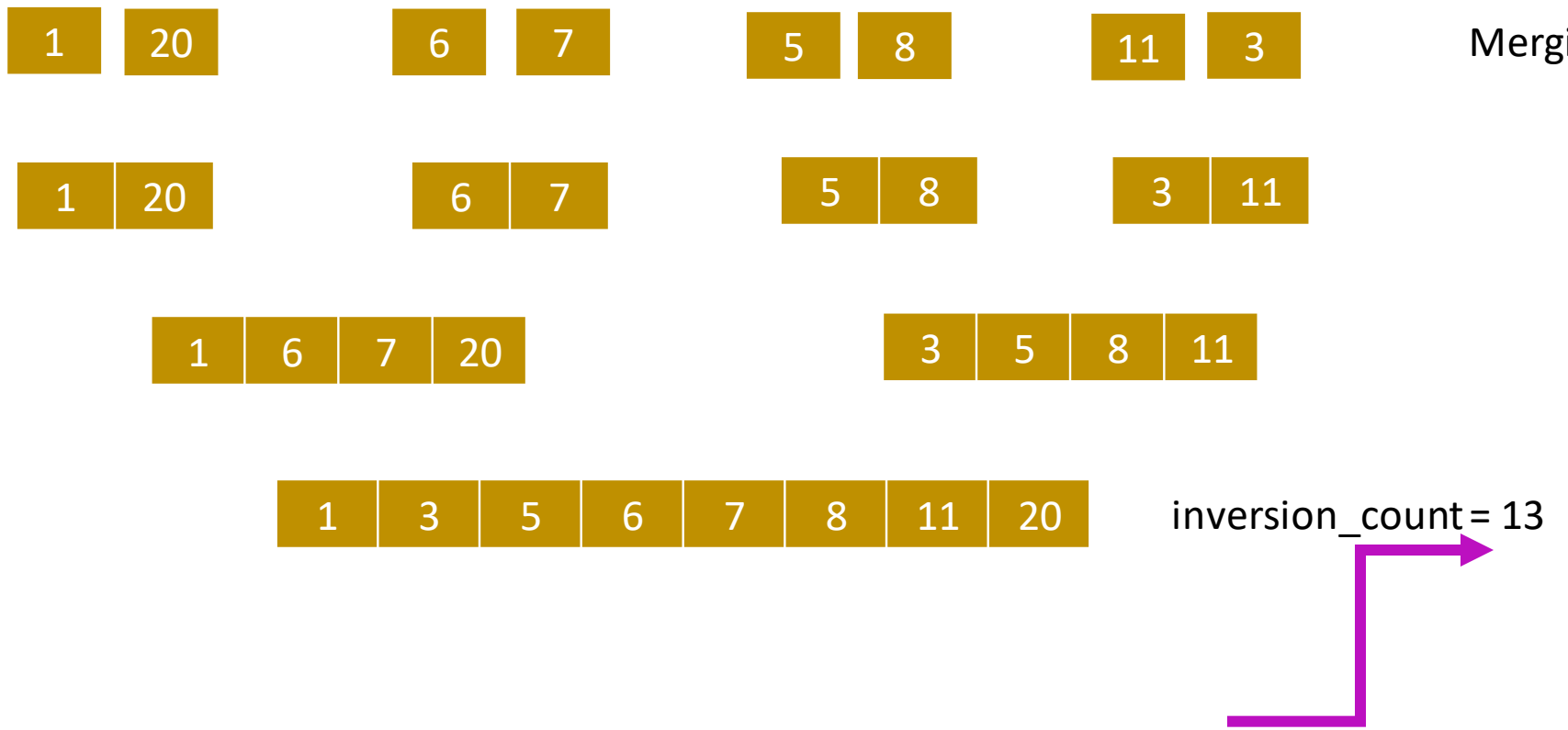
Increment this number by how many numbers are there after and including the current value of i.

→ Increment by 1.



Merging step

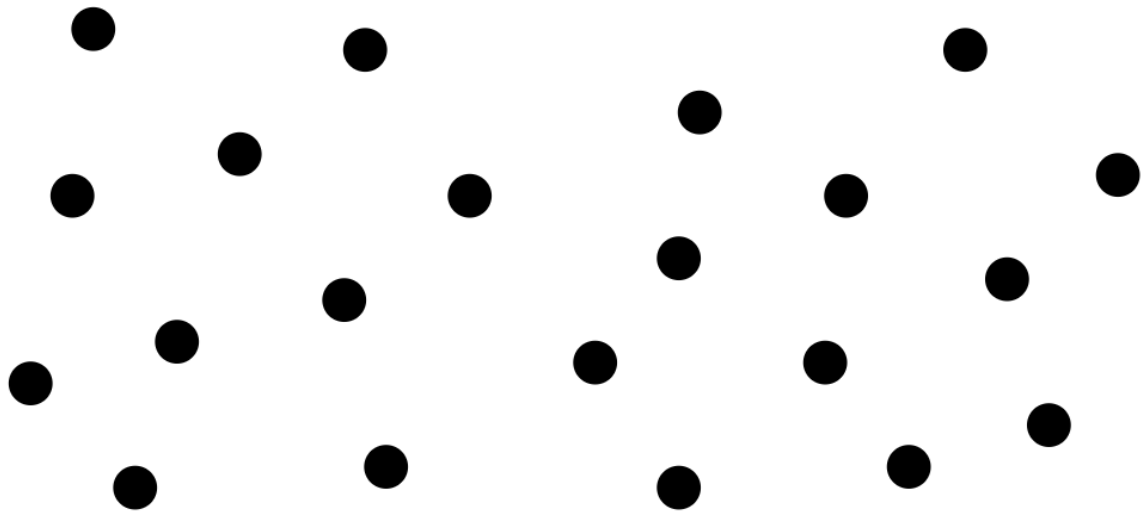
inversion_count = 13


1
20
6
7
5
8
11
3
Merging step
1
20
6
7
5
8
3
11
1
6
7
20
3
5
8
11
1
3
5
6
7
8
11
20
inversion_count = 13

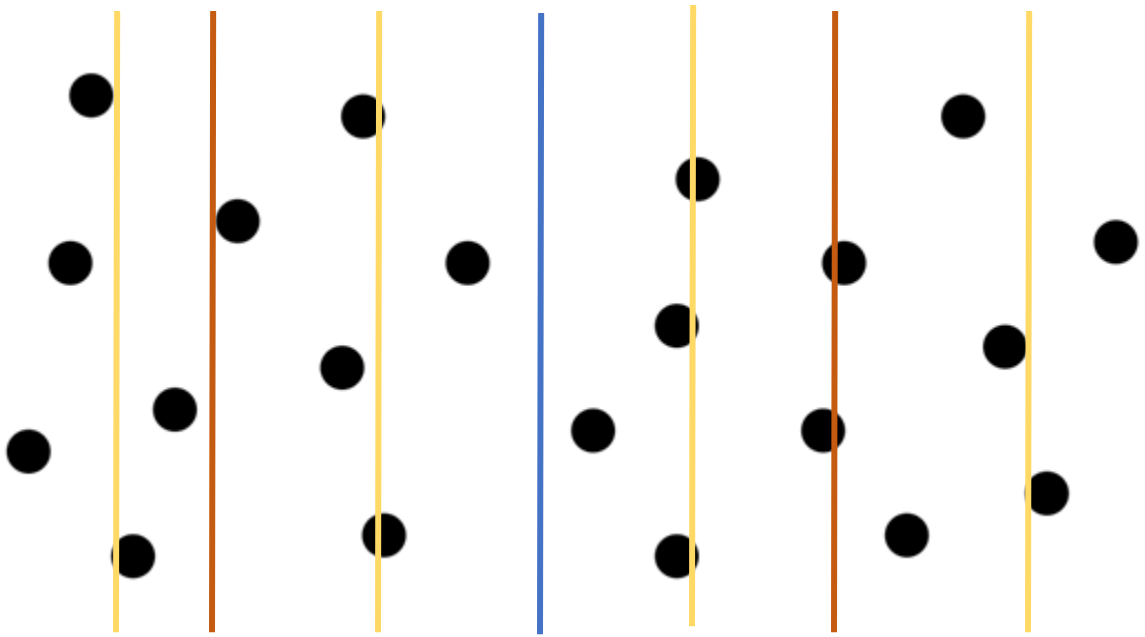
Convex-Hull Problem

- For a given point cloud, the goal is to find the polygon with minimum sides that contains all points on or within the polygon.
- Informally you can think of it as you pound some nails into some wood. You use a rubber band to wrap all the nails from the outside. The rubber band forms the asked polygon.
- There are both iterative (Graham Scan, Jarvis March) and DnC approaches (normal, Quick-Hull).

Q2) Analyze (normal) DnC approach using point cloud below

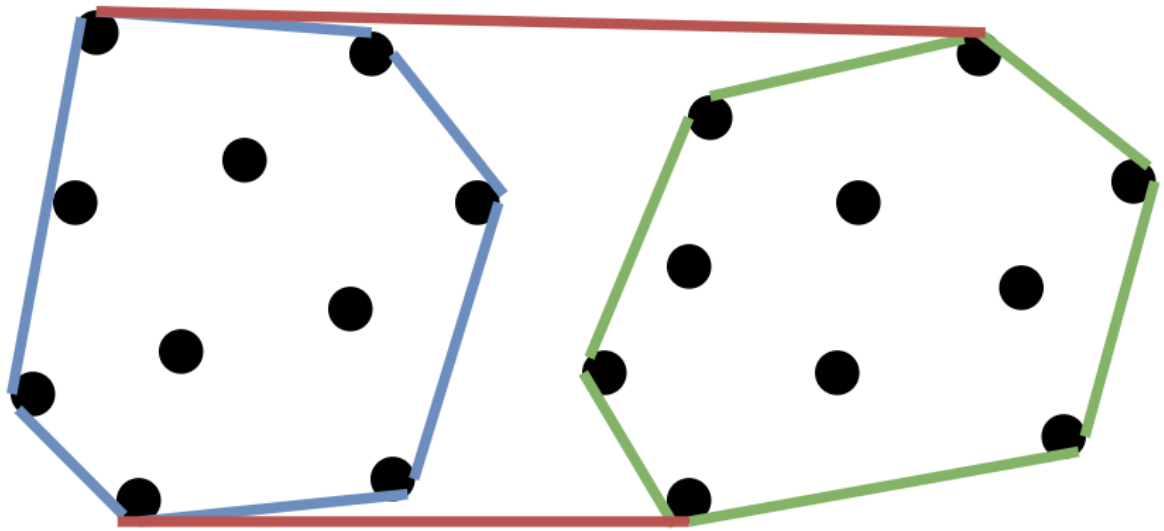


Q2-A) Normal Approach



Partition like Merge-Sort;

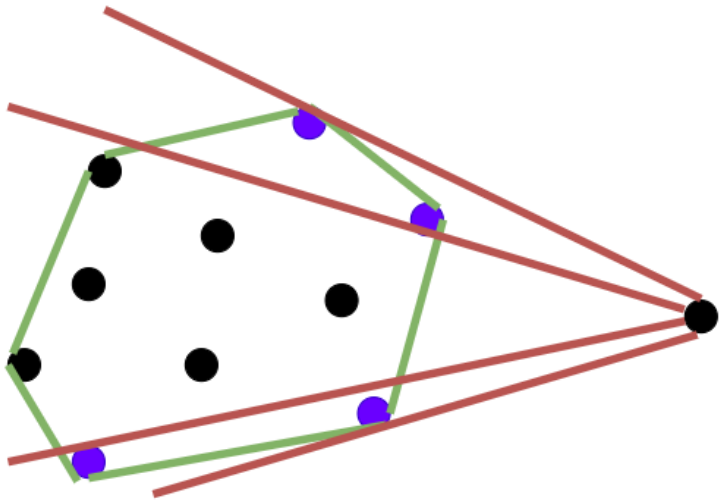
- Sort points regarding x-coordinates
- To find division lines, use median of medians algorithm



Merge two convex-hulls by finding the upper and lower tangent lines.

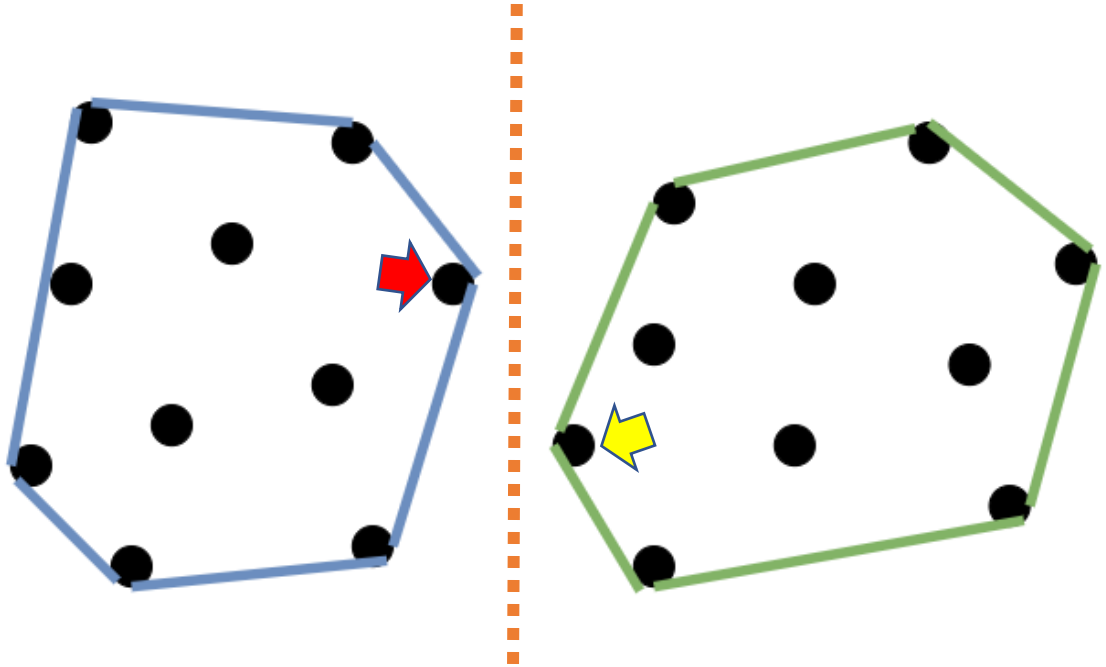
How to find them?

For now, just add one point to a convex-hull



If you look at the relationship between the line, previous and next vertices (points on the current polygon); you can observe;

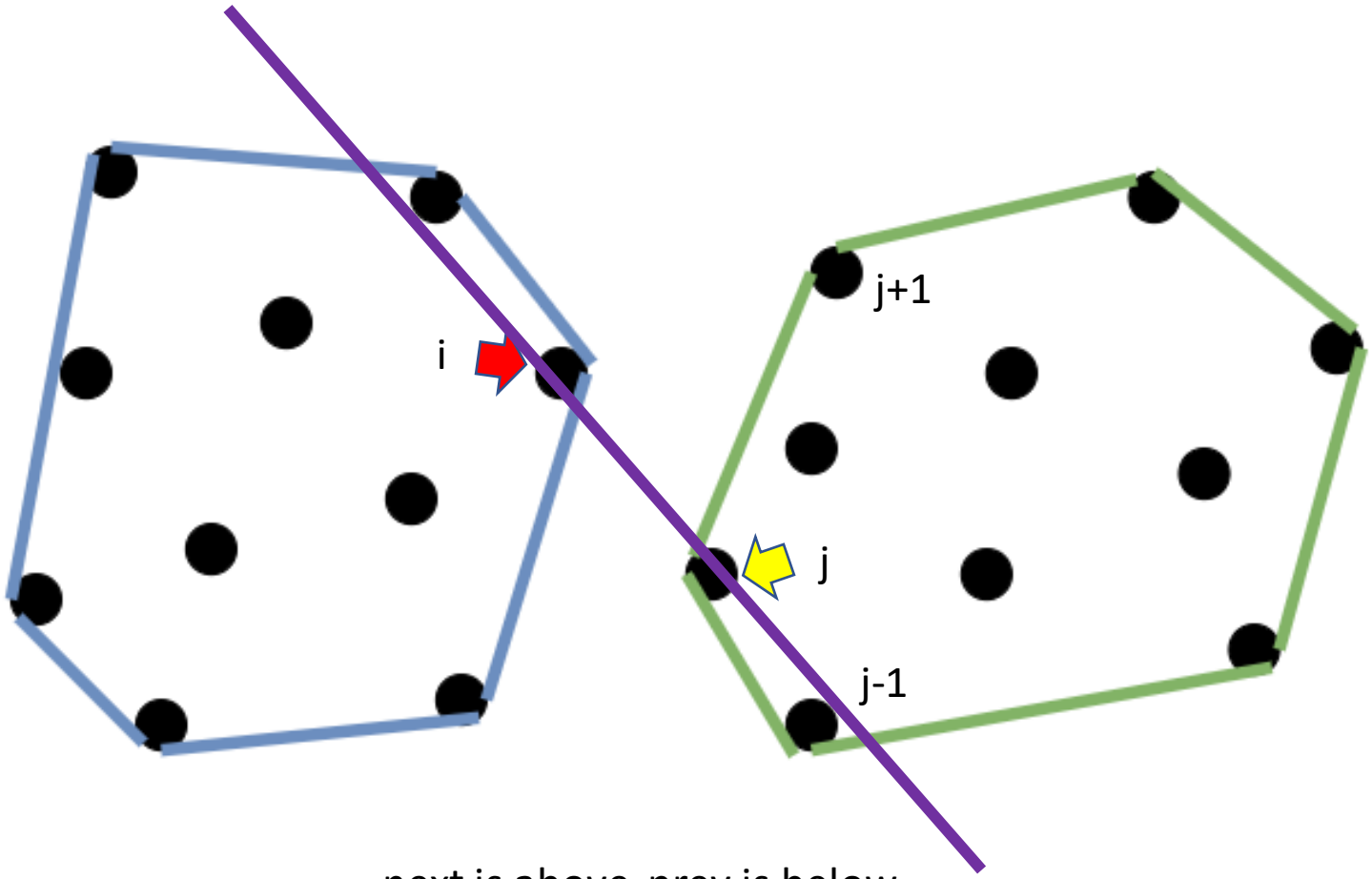
- If both previous and next vertices stay below to the line -> upper tangent
- If both previous and next vertices stay upper to the line -> lower tangent



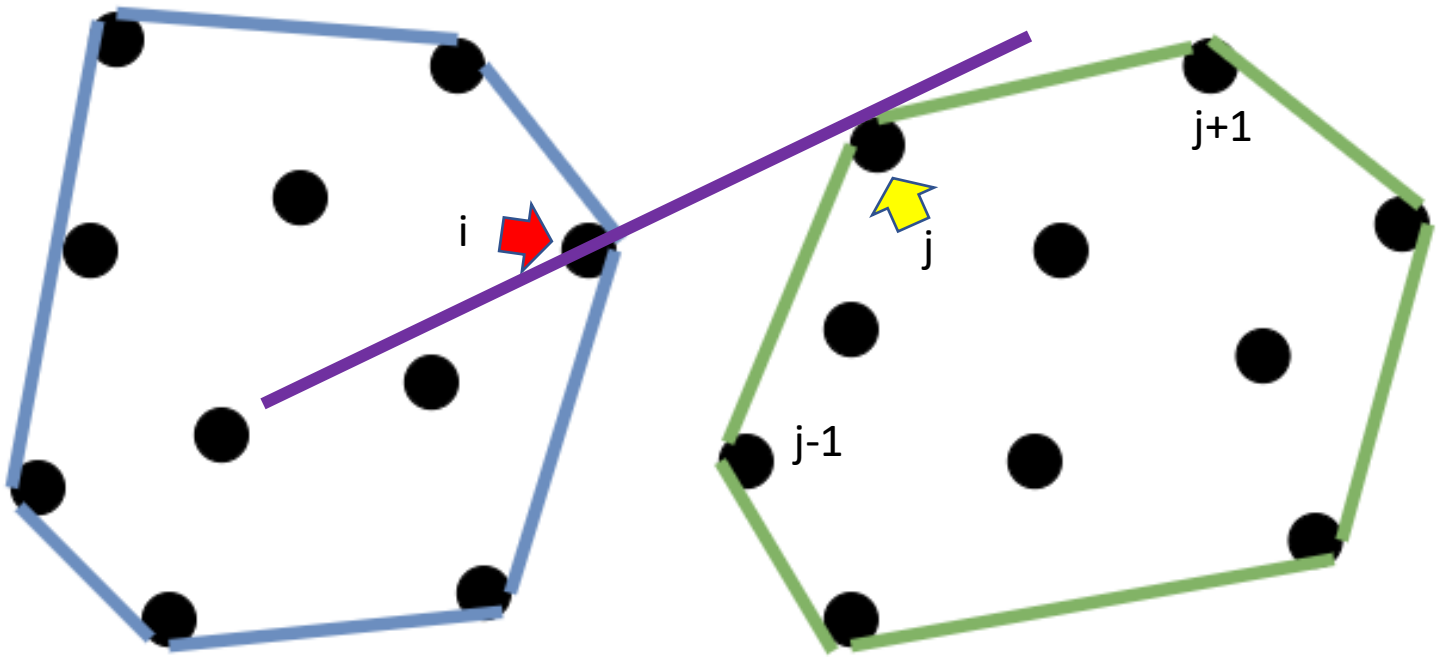
Now, merge the sub-convex-hulls;

- For the left sub-convex-hull, find the rightmost vertex
- For the right sub-convex-hull, find the leftmost vertex
- Alternately find the tangent lines until reached by changing the vertices

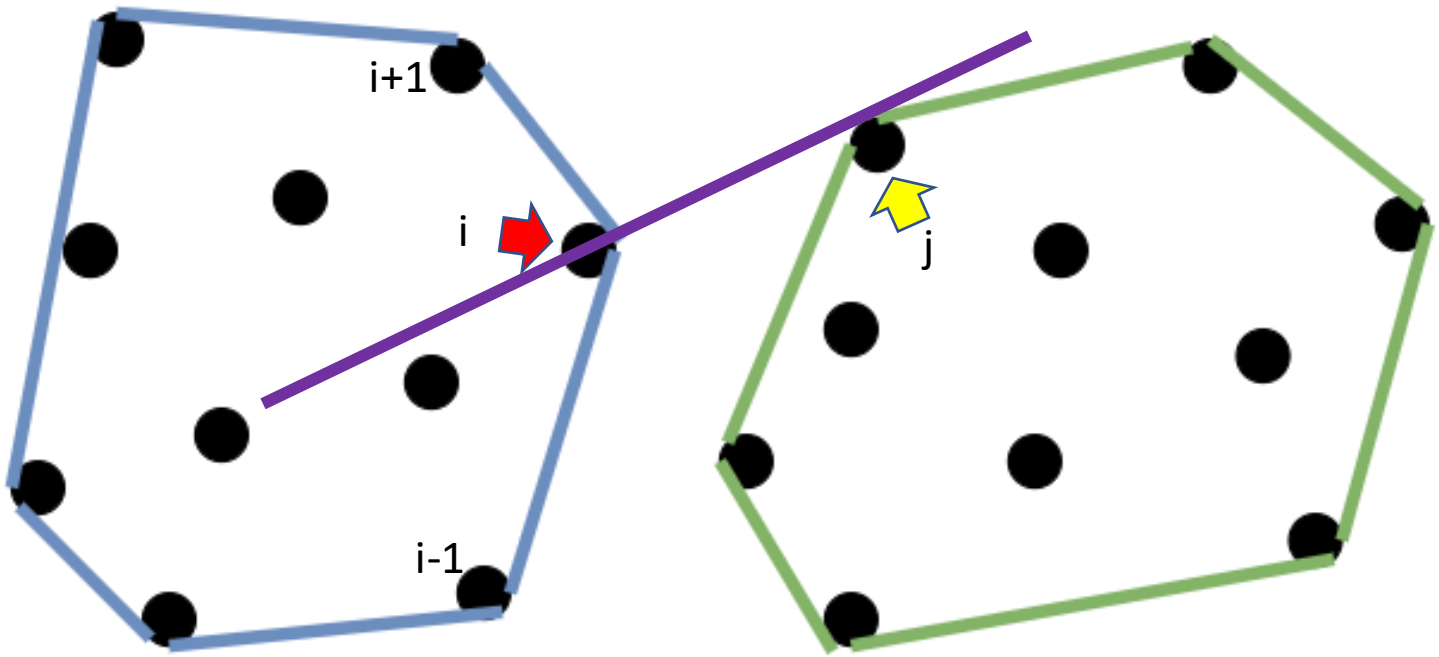
For now, just find the upper tangent



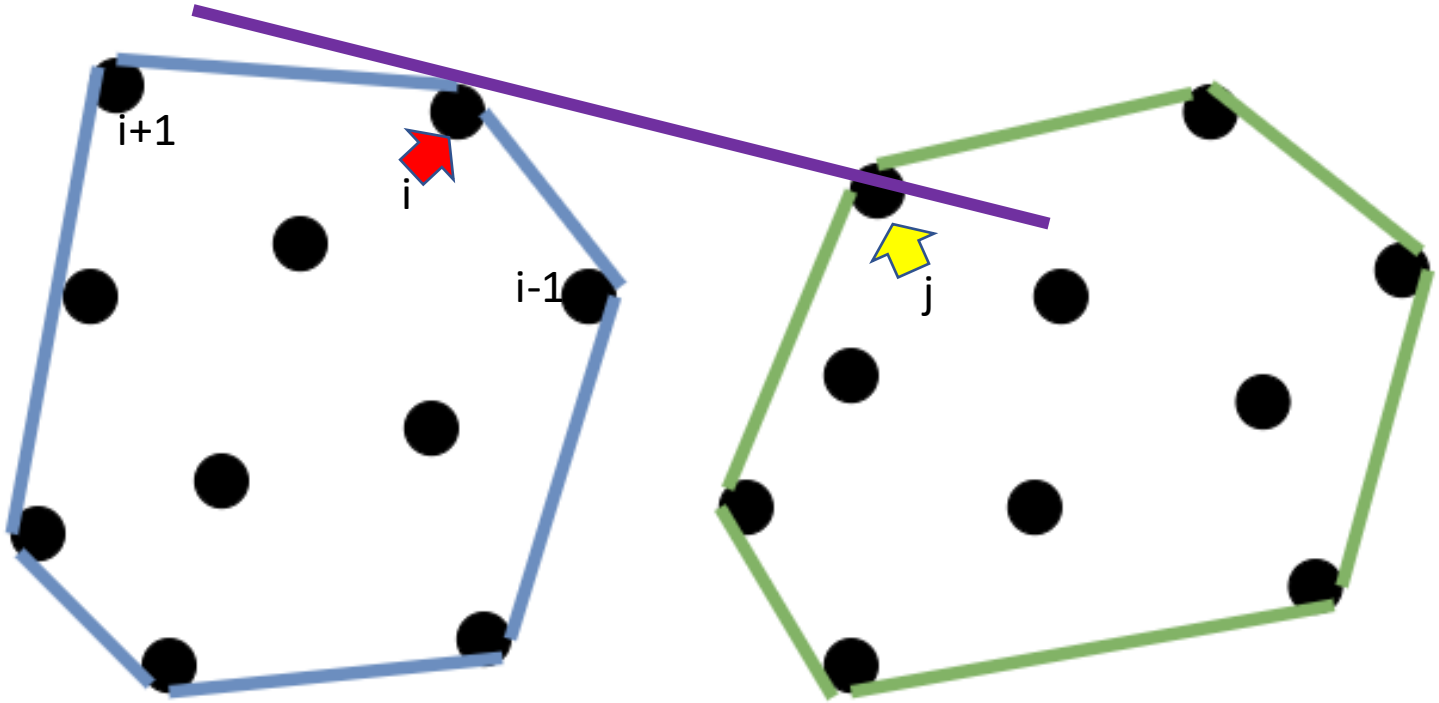
next is above, prev is below
-> select next as pivot



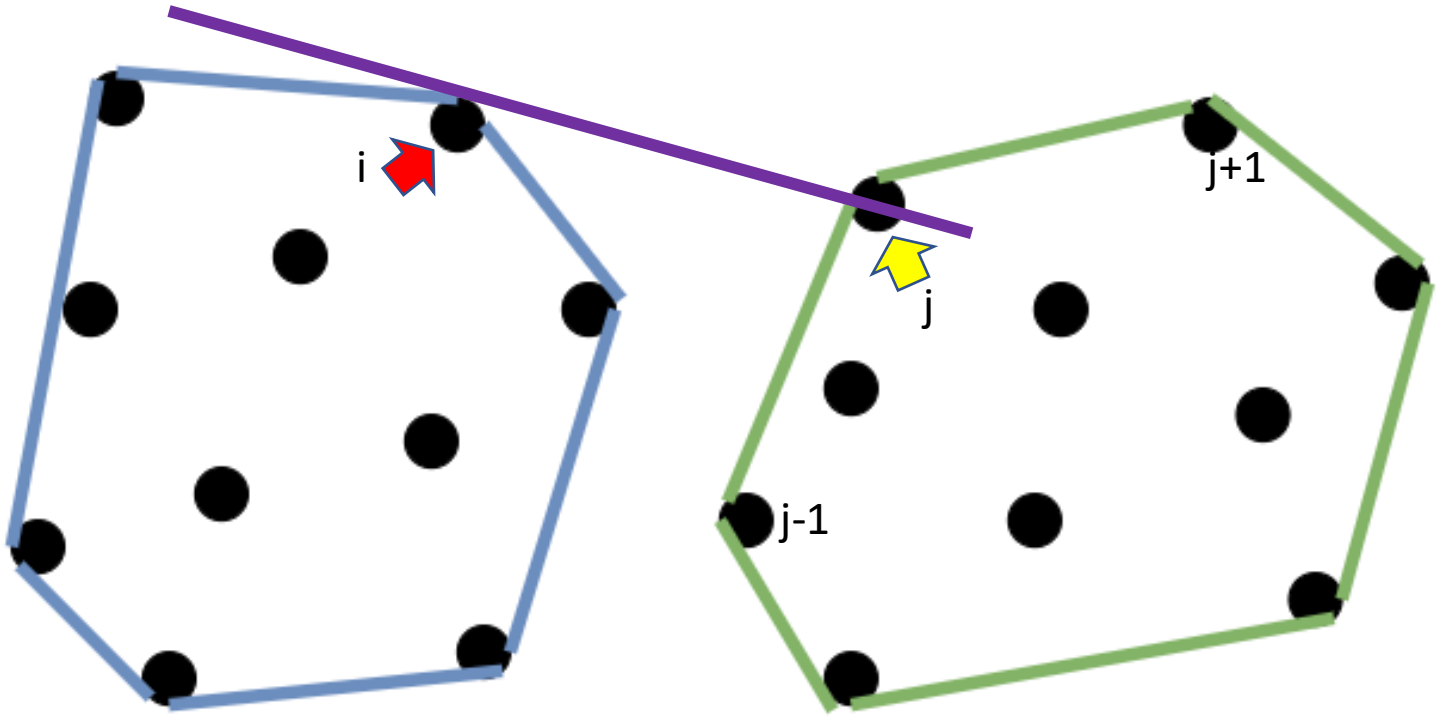
Both are below, we found an upper-tangent for right polygon



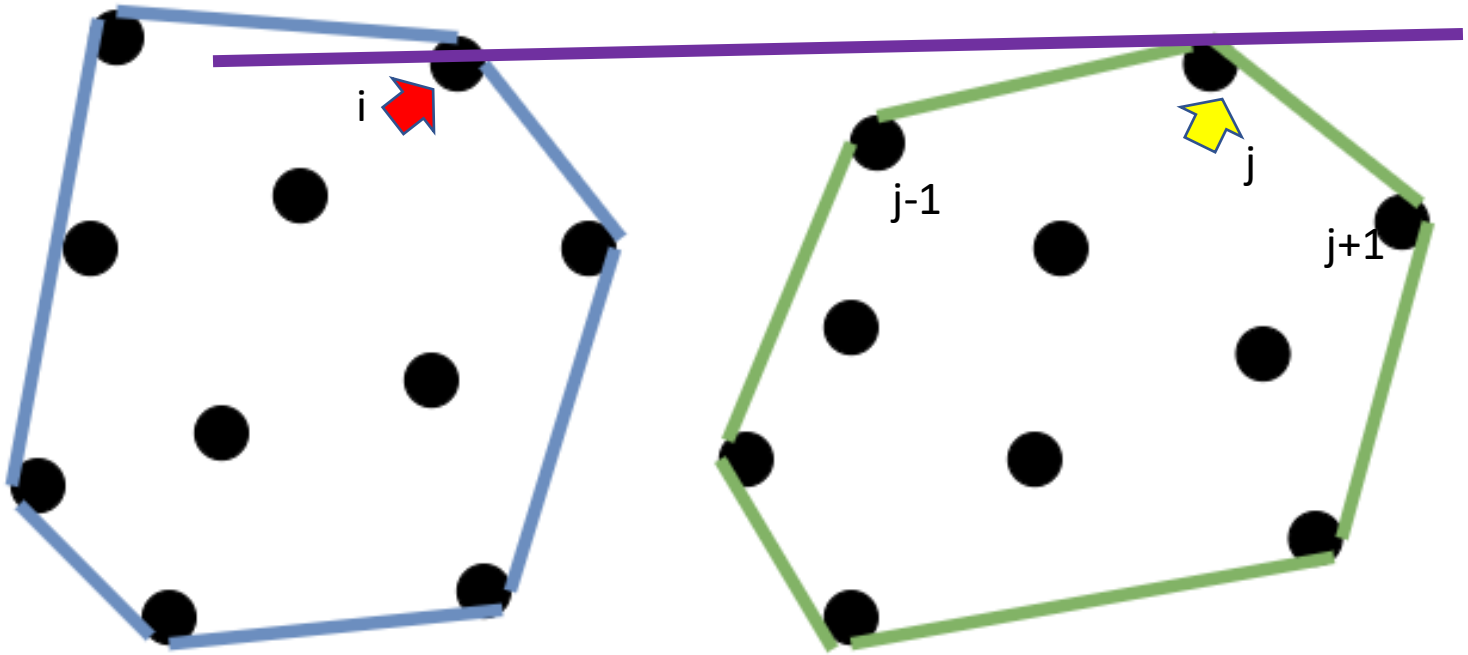
next is above, prev is below
-> select next as pivot



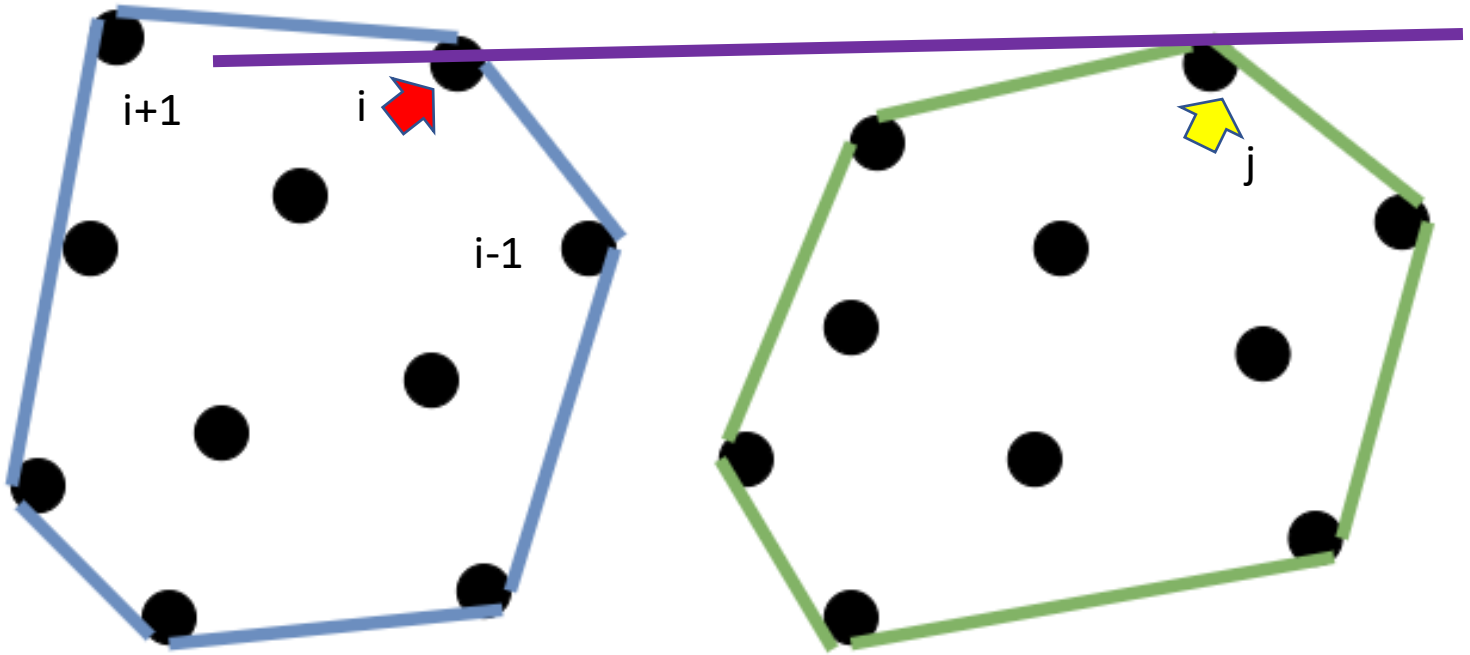
Both are below, we found
an upper-tangent for left polygon



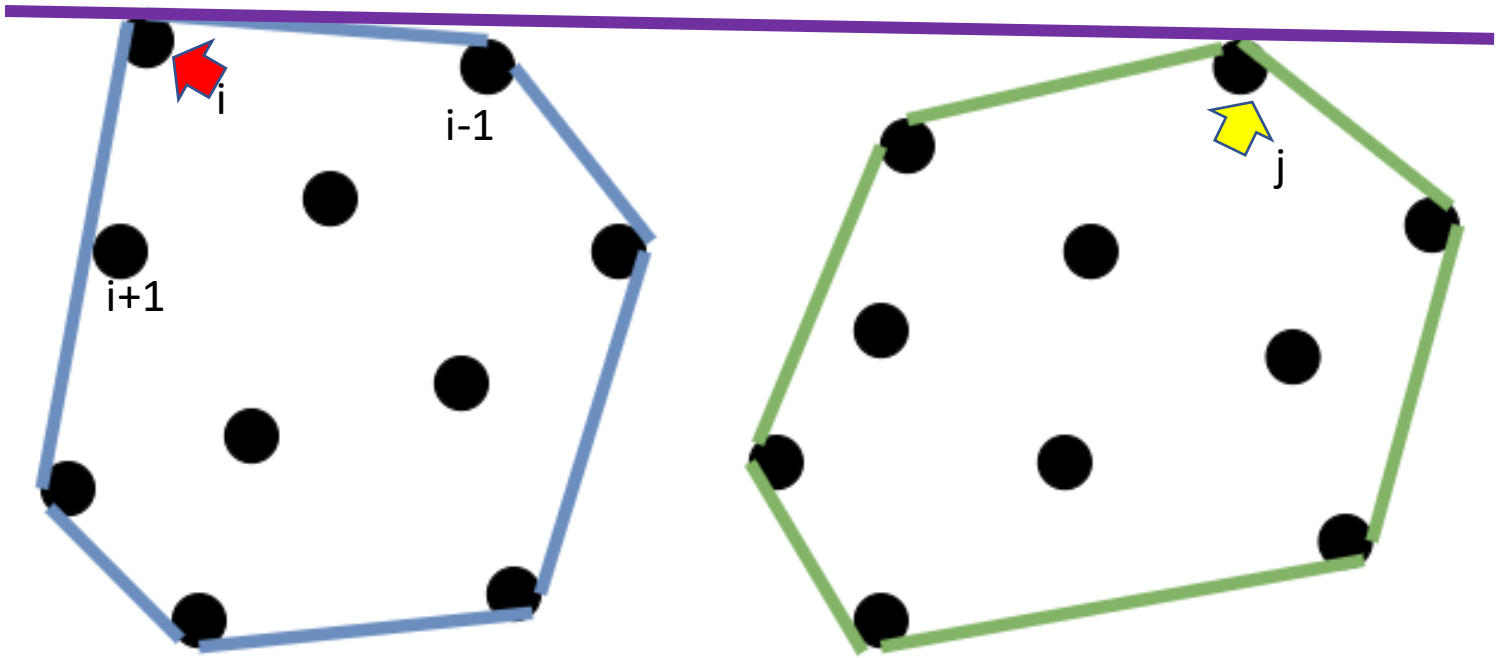
next is above, prev is below
-> select next as pivot



Both are below, we found
an upper-tangent for right polygon

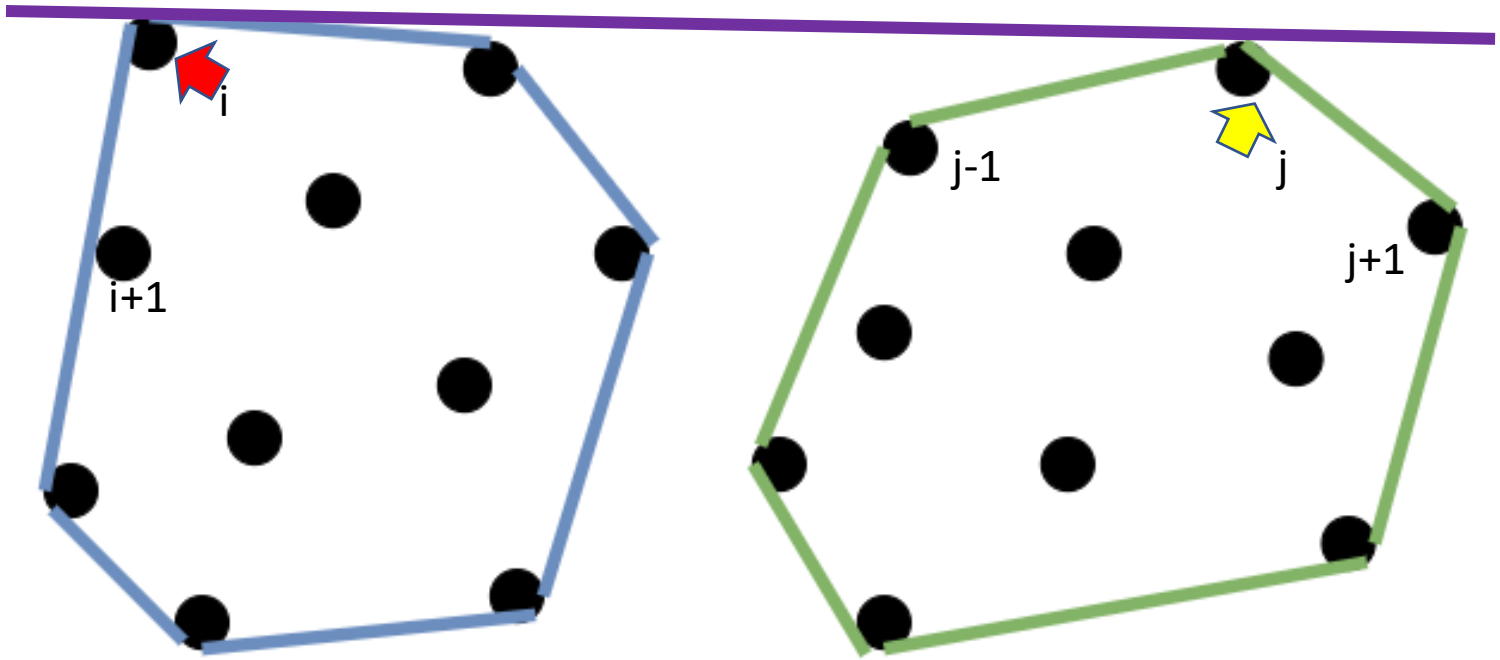


next is above, prev is below
-> select next as pivot



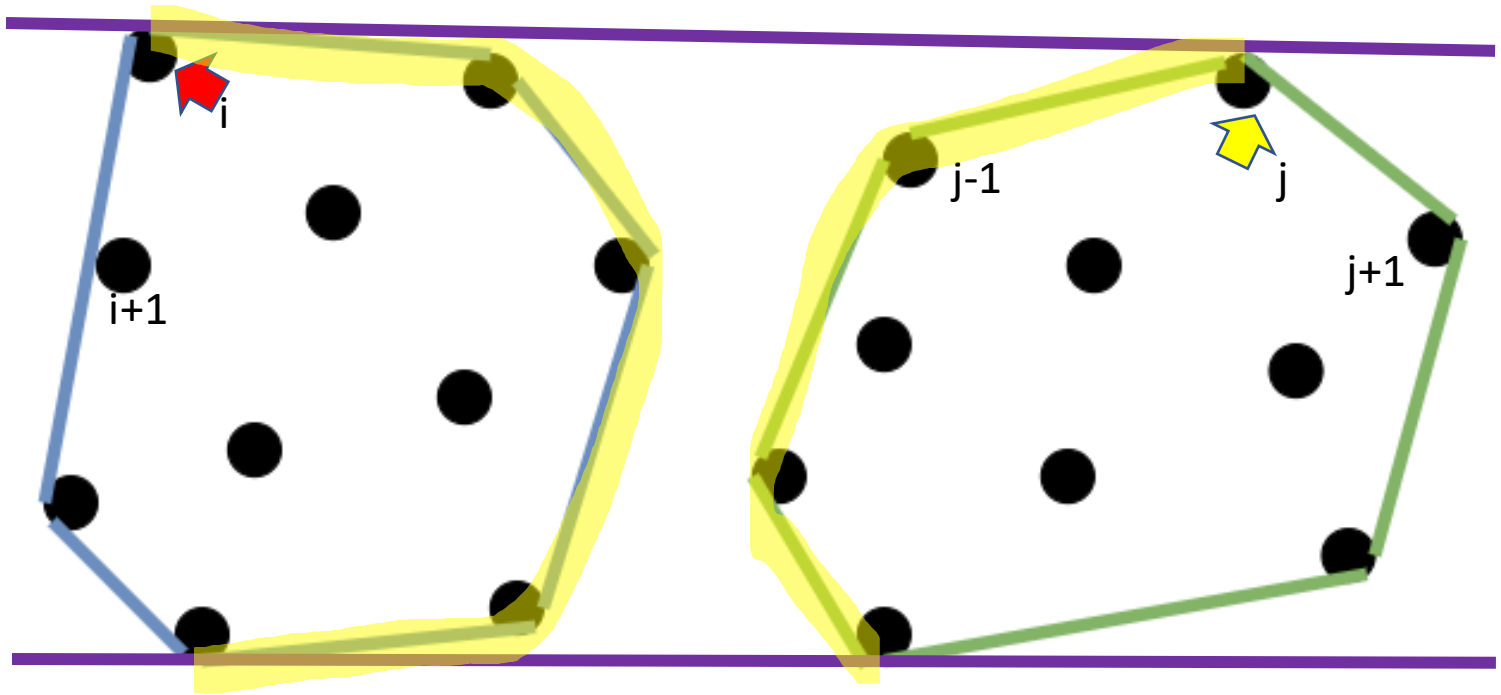
Both are below, we found an upper-tangent for left polygon

As this line is upper tangent for both polygons, we can end our search

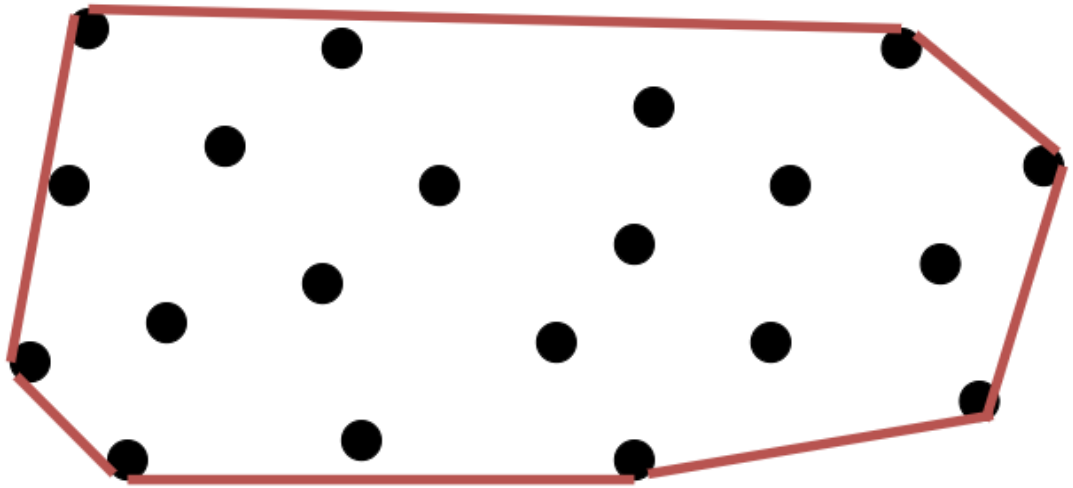


After finding the lower tangent in a similar manner, we can connect two sub-convex-hulls.

Do not forget discarding inner (old) edges



The final polygon is



$$T(n) = \underbrace{2T(n/2)}_{\text{Partition}} + \underbrace{O(n)}_{\text{Finding Median}\atop\text{Merging}} \quad \text{-->} \quad O(n\log n)$$

Additional Resources for Convex-Hull Problem

- https://www.youtube.com/watch?v=EzeYI7p9MjU
- https://www.youtube.com/playlist?list=PLtTatrCwXHzFp1Y_c6i82waJ2b7OWyiKM
- https://www.youtube.com/watch?v=NH6WbP3lDac
- https://www.youtube.com/watch?v=B6AOzBnenZU