# Conditions for Merging

$x_1$   $y_1$     $x_2$   $y_2$     $x_1$   $\max(y_1, y_2)$

$[\underline{1}, \underset{\sim}{3}], [2, \underset{\sim}{4}] \Rightarrow [\underline{1}, \underset{\sim}{4}]$    $(x_2 < y_1)$

$[\underline{1}, \underset{\sim}{3}], [3, \underset{\sim}{5}] \Rightarrow [\underline{1}, \underset{\sim}{5}]$    $(x_2 = y_1)$

$[\underline{1}, \underset{\sim}{6}], [2, \underset{\sim}{4}] \Rightarrow [\underline{1}, \underset{\sim}{6}]$    $(x_2 < y_1)$

$\underbrace{\qquad\qquad}$

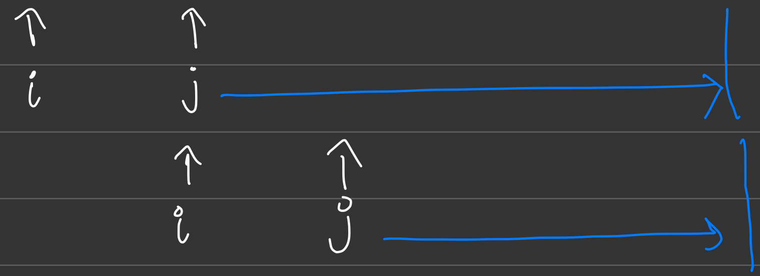
$(x_2 \le y_1)$

cond<sup></sup>ⁿ of merging

Note: The intervals should be sorted wrt $x$

Brute force: (Sort first)

$a = [[1,3], [2,6], [4,5], [8,10], [15,18]]$



Check for all pairs, if they are merging or not,

if merging, store the new pair in a vector

T.C → $O(n^2)$ | S → $O(n)$

# Optimised Solution
## (Single pass approach)

- Sort the intervals sort(a.begin(), a.end());
- Maintain a current_interval
- Linearly iterate over the intervals array
    - if ith interval merges with curr_interval
      (a[i][0] ≤ curr_interval[1])
        - Merge those intervals & update curr_interval
          curr_interval[0] is unchanged.
          curr_interval[1] = max(curr_interval[1], a[i][1]);
    - if not,
        - add the curr_interval to ans
          ans.push_back(curr_interval);
        - update the curr_interval to ith interval
          curr_interval[0] = a[i][0]
          curr_interval[1] = a[i][1]

- Finally, add the last curr_interval as loop breaked before doing that
  ans.push_back(curr_interval);

# Dry Run

a = [[1,3], [2, 6], [4,5], [8,10], [15,18]]

Curr_pair = [1, 3]

[1,3] , [2,6] → [1, 6]

curr_pair = [1, 6]

[1,6 ] , [4,5] → [1,6]

Curr_pair = [1, 6]

[1,6] , [8,10] → X

ans = [[1,6]]

Curr_pair = [8,10]

[8,10] , [15,18] → X

ans = [[1, 6] , [8,10]]

Curr_pair = [15,18]

ans = [[1,6], [8,10], [15,18]]

T.C → O(n logn); SC → O(n)