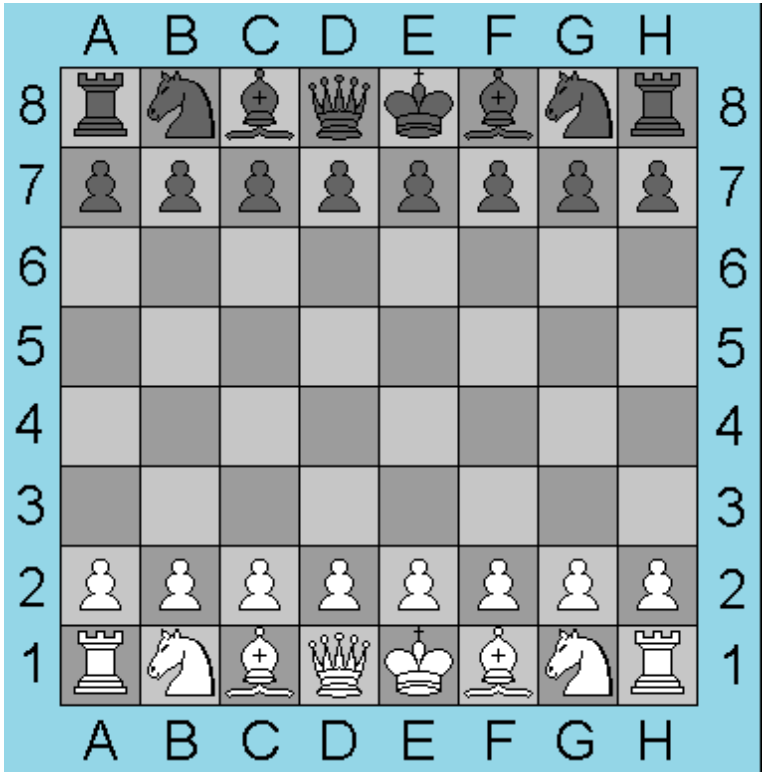
Week 2: Part 2

Recursion, Recurrences & Running time

More Applications

- Tiling
- Skylines

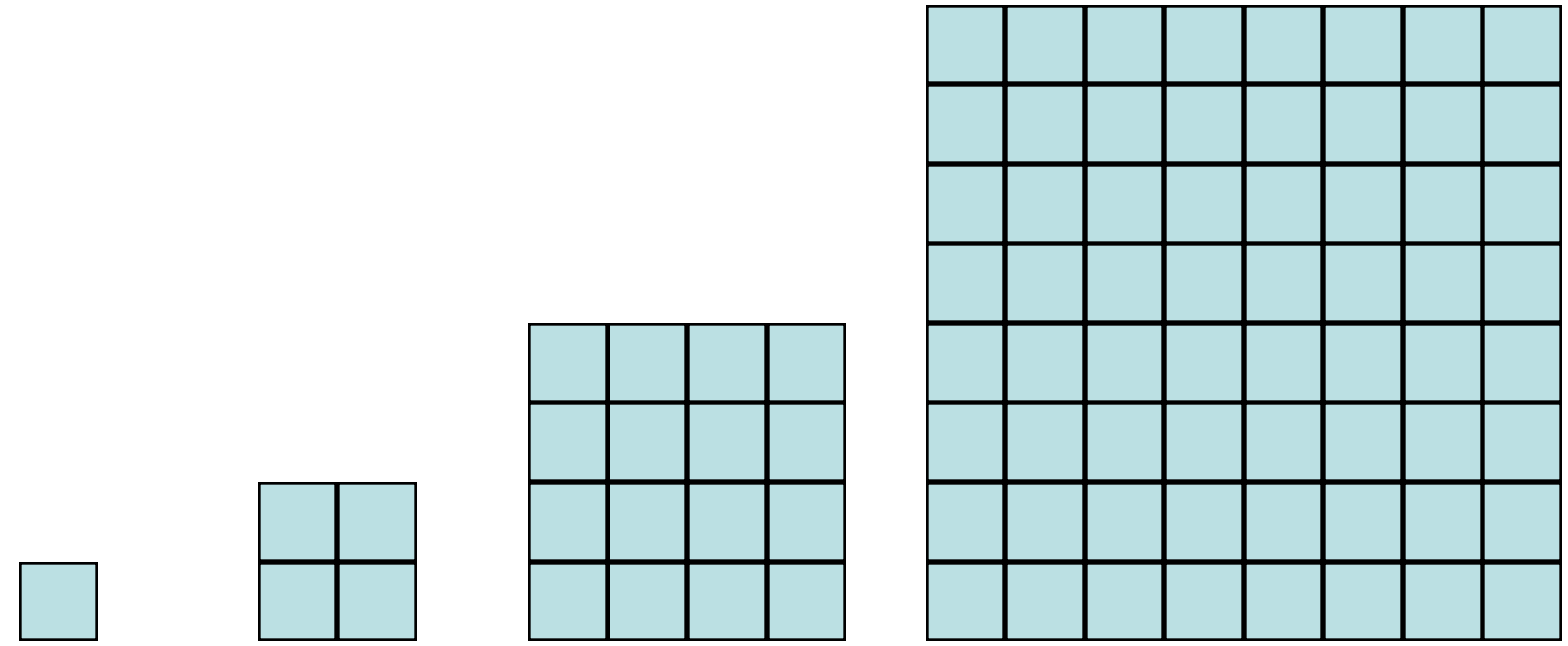
Tiling A Defective chessboard



A real chessboard.

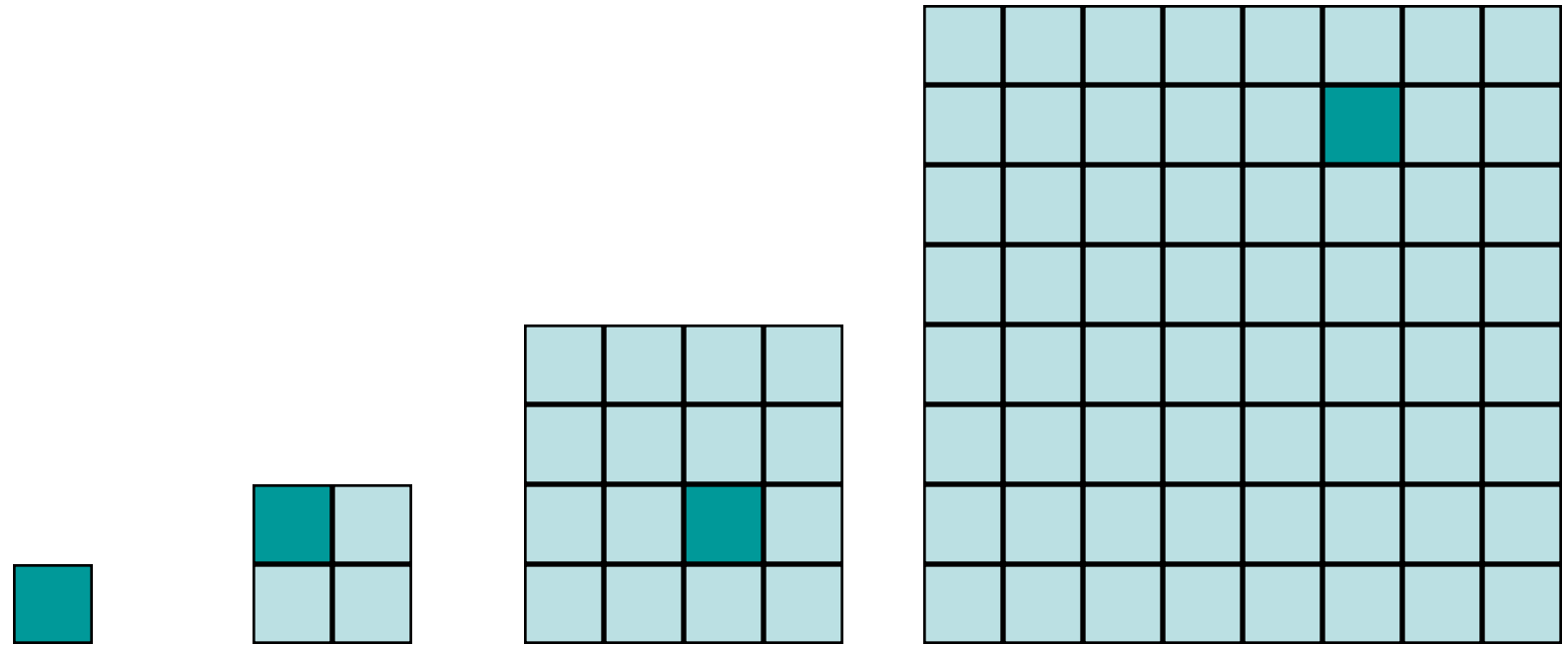
Our Definition Of A chessboard

A chessboard is an n x n grid, where n is a power of 2.



A Defective chessboard

A defective chessboard is a chessboard that has one unavailable (defective) position.



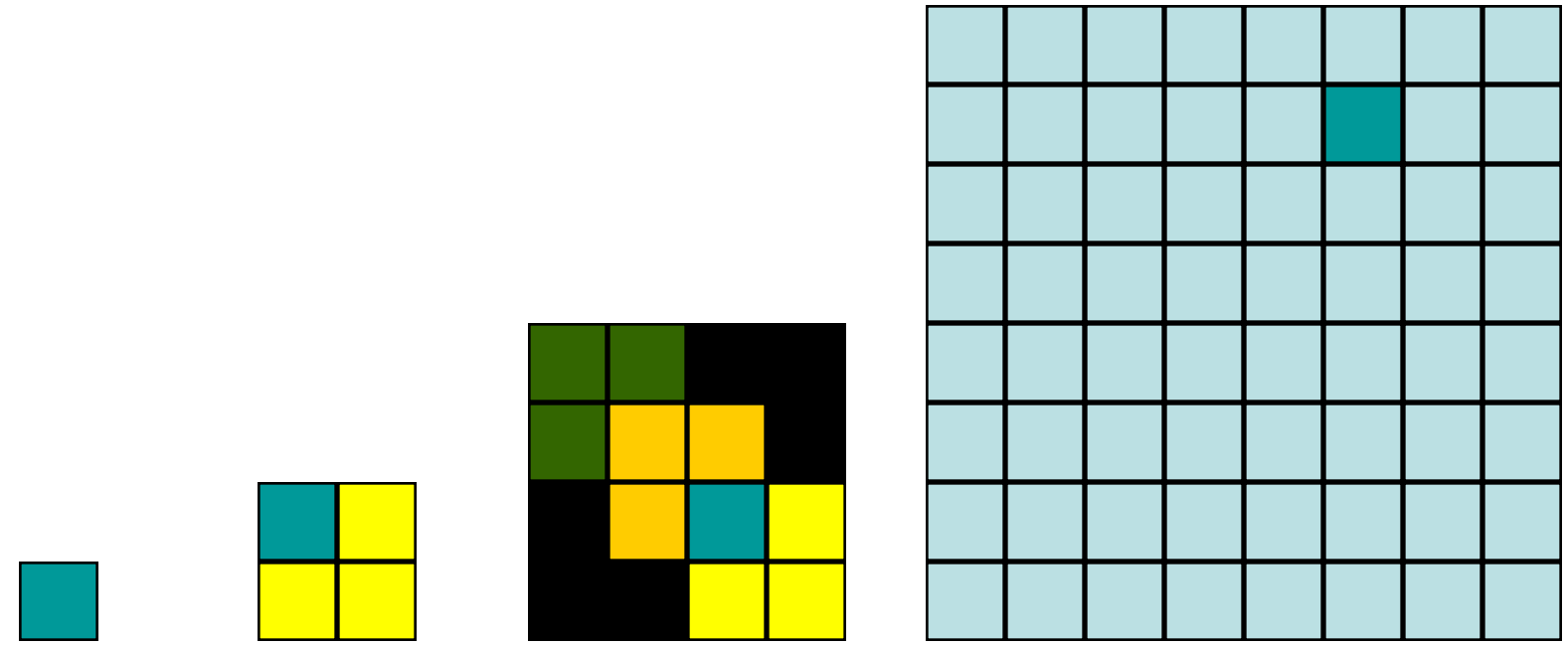
A Triomino

A triomino is an L shaped object that can cover three squares of a chessboard.

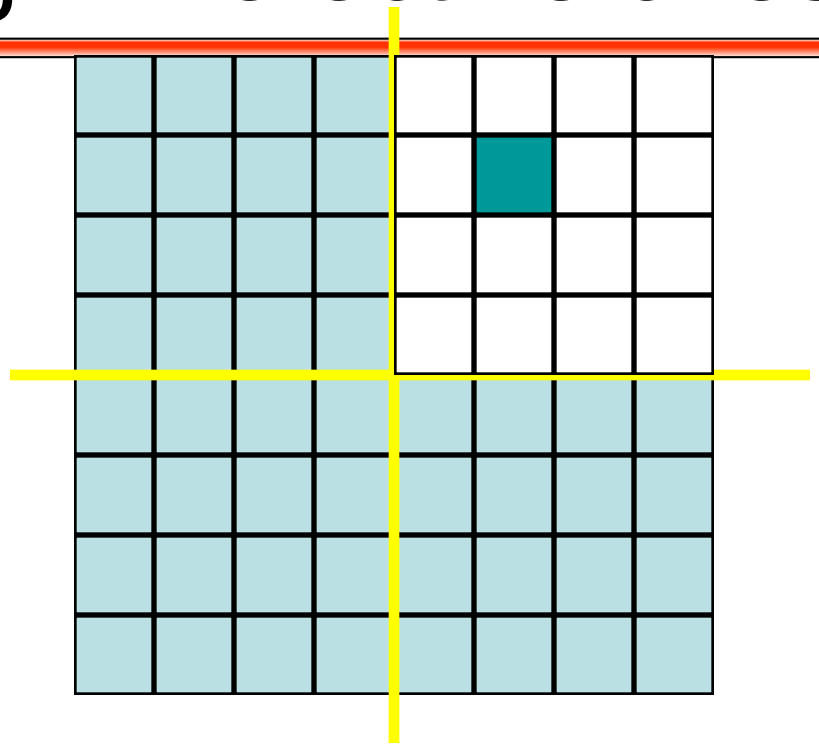
A triomino has four orientations.

Tiling A Defective chessboard

Place triominoes on an 2^n x 2^n defective chessboard so that all nondefective positions are covered.



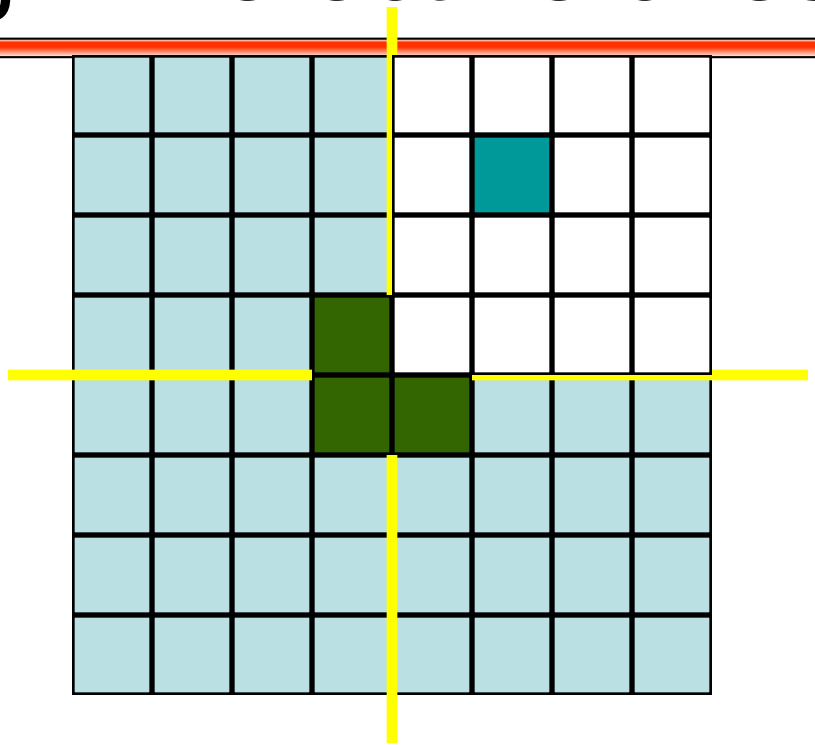
Tiling A Defective chessboard



Divide into four smaller chessboards. n/2 x n/2

One of these is a defective n/2 x n/2 chessboard.

Tiling A Defective chessboard



Make the other three n/2 x n/2 chessboards defective by placing a triomino at their common corner.

Recursively tile the four defective n/2 x n/2 chessboards.

Tiling: Algorithm

INPUT: *n* – the board size (nxn board), *L* – location of the hole.
OUTPUT: tiling of the board

```
Tile(n, L)
   if n = 2 then
      Trivial case
      Tile with one tromino
      return
   Divide the board into four equal-sized boards
   Place one tromino at the center to cut out 3 additional
    holes (orientation based on where existing hole, L, is)
   Let L1, L2, L3, L4 denote the positions of the 4 holes
   Tile(n/2, L1)
   Tile(n/2, L2)
   Tile(n/2, L3)
   Tile(n/2, L4)
```

Recurrence

Let T(n) be the time taken to tile a n x n defective chessboard.

$T(1) = 1,$

$T(n) = 4T(n/2) + c,$ when $n > 0.$

Use the master method

$a=4, b=2, f(n)=c, n^{lg4}$

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

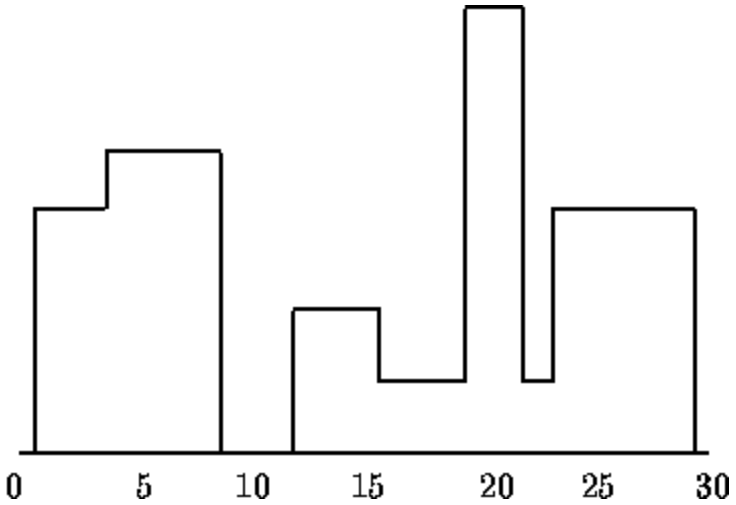
Skyline Problem

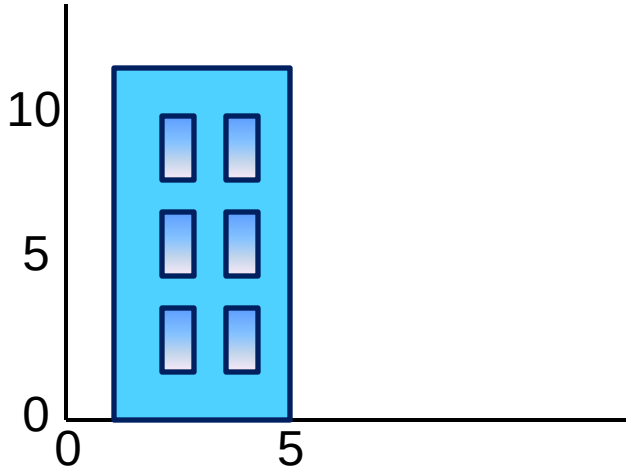
You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city.

- To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat).

A building is specified by an ordered triple (Li, Hi, Ri)

- where Li and Ri are left and right coordinates, respectively, of building *i* and Hi is the height of the building.
- Below the single building is specified by (1,11,5)





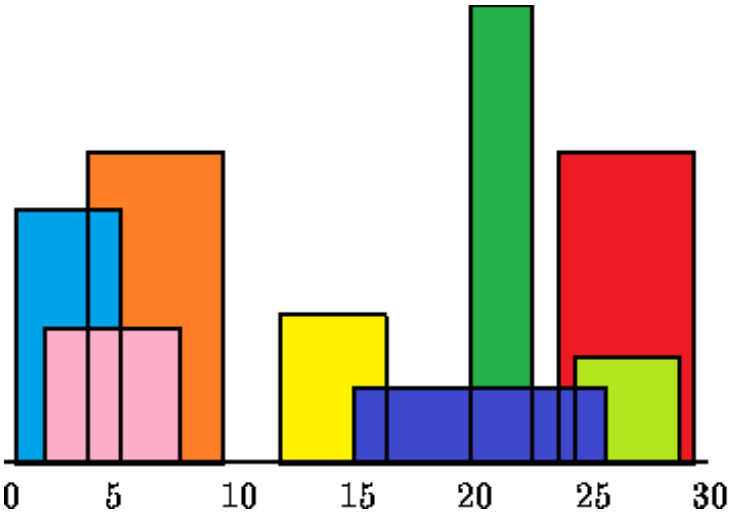
Skyline Problem

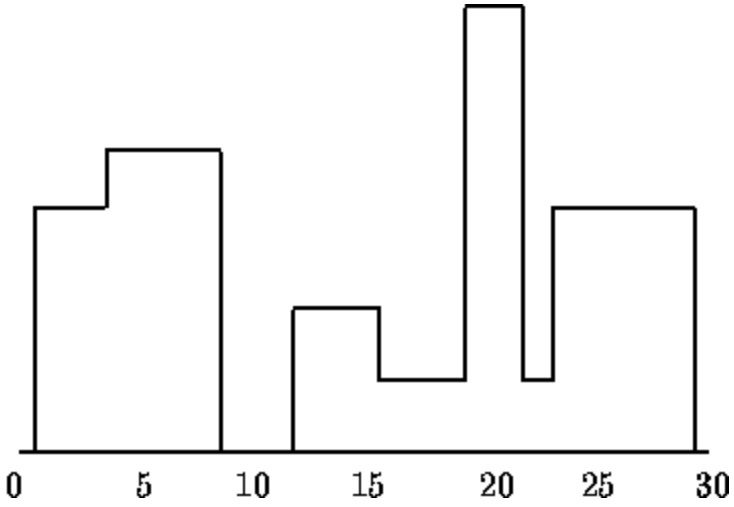
In the diagram below buildings are shown on the left with triples :

{(1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28)}

The skyline of those buildings is shown on the right, represented by the sequence:

{(1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13), (29, 0)}





Skyline Problem

- We can solve this problem by separating the buildings into two halves and solving those recursively and then Merging the 2 skylines.
  - Similar to merge sort.
  - Requires that we have a way to merge 2 skylines.

- Consider two skylines:
  - Skyline A: $\{(a_1, h_{11}),(a_2, h_{12}),(a_3, h_{13}),\ldots, (a_n, 0)\}$
  - Skyline B: $\{(b_1, h_{21}),(b_2, h_{22}), (b_3, h_{23}),\ldots, (b_m, 0) \}$

- Merge(list of a's, list of b's)
  - $\{(c_1, h_{11}),(c_2, h_{21}), \ldots, (c_{n+m}, 0) \}$

Skyline Problem

Clearly, we merge the list of a's and b's just like in the standard Merge algorithm.

- But, it addition to that, we have to properly decide on the correct height in between each set of these boundary values.
- We can keep two variables, one to store the current height in the first set of buildings and the other to keep the current height in the second set of buildings.
- Basically we simply pick the greater of the two to put in the gap.

Skyline Problem

- After we are done, (or while we are processing), we have to eliminate redundant "gaps", such as (8, 15), (9, 15), (12,5), where there is the same height between the x-coordinates 8 and 9 as there is between the x-coordinates 9 and 12.
  - (Similarly, we will eliminate or never form gaps such as 8, 15, 8, where the x-coordinate doesn't change.)

Skyline Problem - Runtime

- Since merging two skylines of size n/2 should take O(n), letting T(n) be the running time of the skyline problem for n buildings, we find that T(n) satisfies the following recurrence:
  - $T(n) = 2T(n/2) + O(n)$
- Thus, just like Merge Sort, for the Skyline problem **$\underline{T(n) = \Theta(n \lg n)}$**
- Code http://code.geeksforgeeks.org/6GxyYW