VE527

Computer-Aided Design of Integrated Circuits

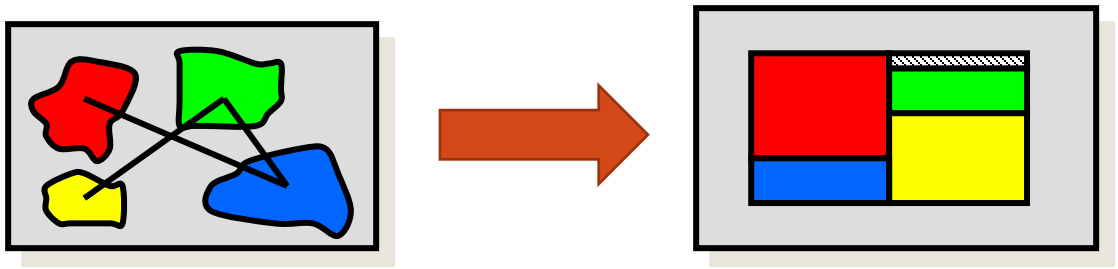
Floorplanning

Outline

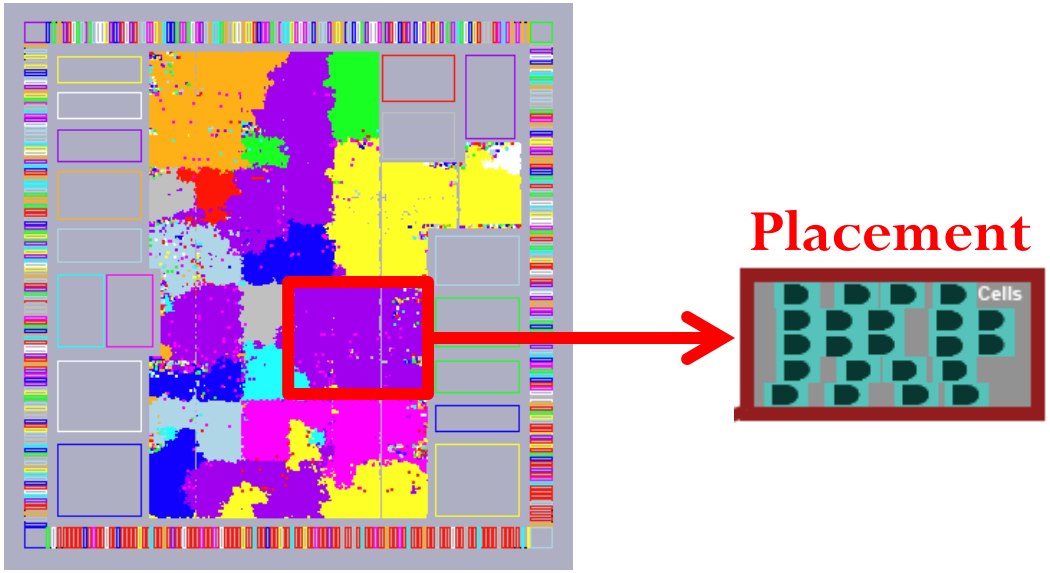
- Floorplanning
  - Basics
  - Placeholder Representation and Perturbation
  - Floorplan Sizing

Floorplanning: Problem

- Given **circuit modules** (may consist of a number of cells) and their connections, determine the approximate location of these modules



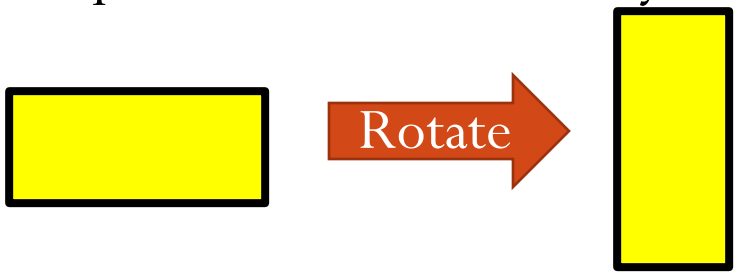
Real example of Floorplanning



Modules

- Results of partitioning
- Fixed area, generally rectangular

- Two types of modules
  - Hard modules: shape is fixed. Rotation may or may not be allowed



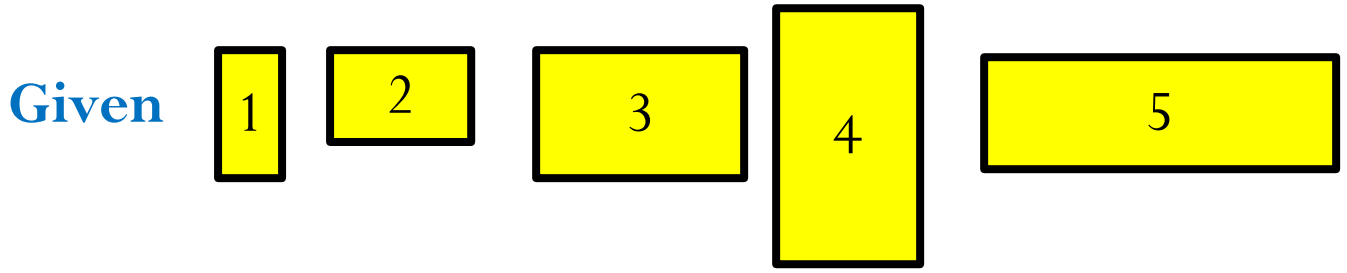
  - Soft modules: flexible shape
    - Discrete choices. E.g., {6x4, 8x3, 12x2}
    - Continuous choices. Any aspect radio (h_i/w_i) within [0.8, 1.2]

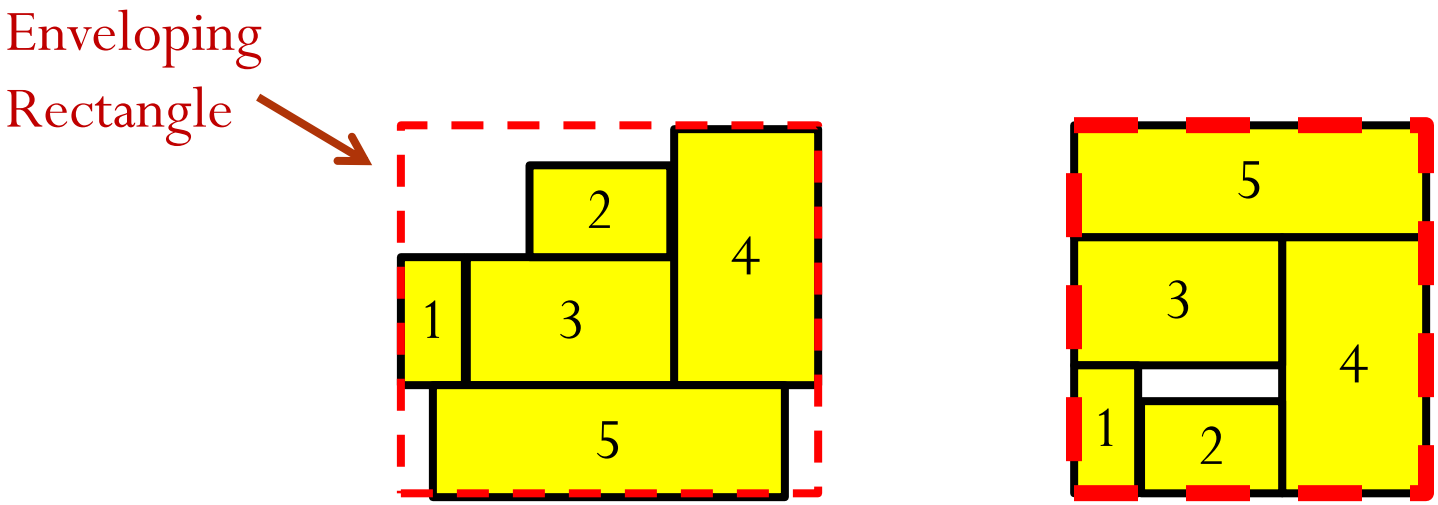
Floorplanning

- Objectives:
  - Minimize area of enveloping rectangle
  - Determine best shape of soft modules
  - Minimize total wirelength to make subsequent routing easy
  - Minimize power consumption
- Possible additional constraints:
  - Fixed location for some modules
  - Fixed dies. E.g., should put all the modules within a rectangle of 10um×12um
  - Range of die aspect ratio, e.g., $h/w \in [0.9,1.1]$
- NP-hard (what did you expect? ☺)

Floorplanning Example

- Minimizing area, with hard module and no rotation



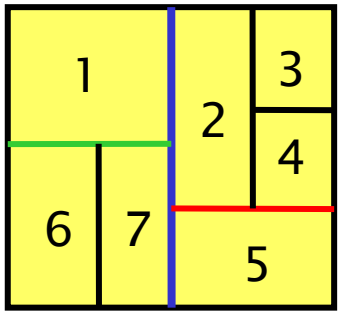


Floorplanning: Why Important?

- Early stage of physical design
  - Determines the location of large blocks
    ➔ detailed placement easier (divide and conquer!)
  - Estimates of area, delay, power
    ➔ important design decisions
  - Impact on subsequent design steps (e.g., routing, heat dissipation analysis and optimization)

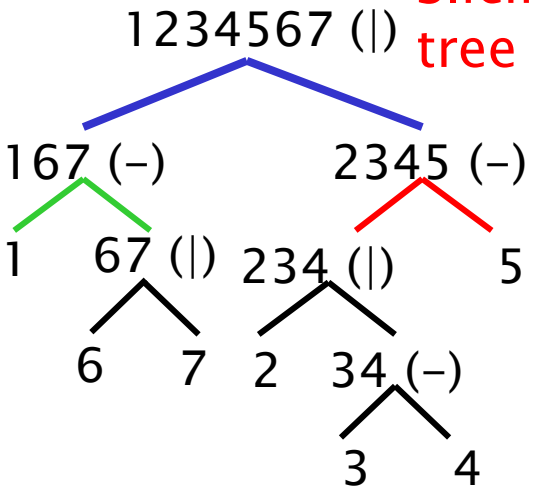
Floorplan Classes

- **Slicing**, recursively defined as:
  - A module, or
  - A floorplan that can be partitioned into two slicing floorplans with a horizontal or vertical cut line

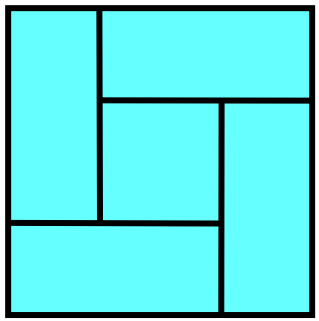


Slicing floorplan

Corresp. Slicing tree



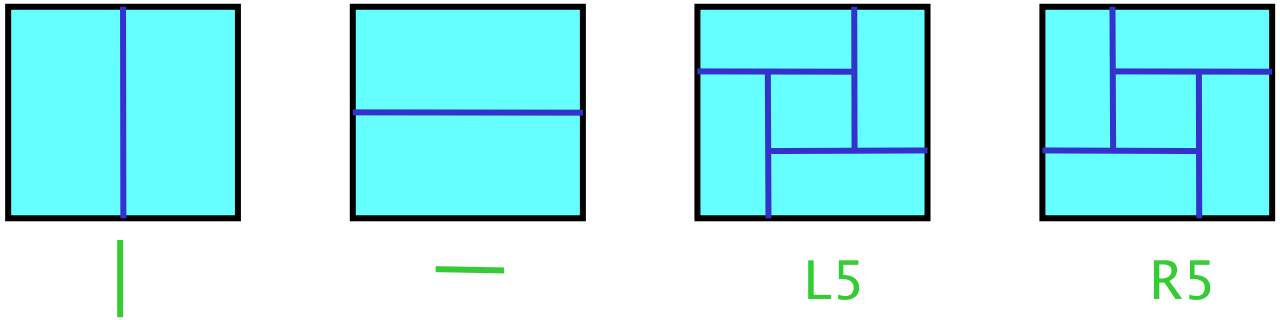
- **Non-slicing**
  - Superset of slicing floorplans
  - Contains the "wheel" shape too



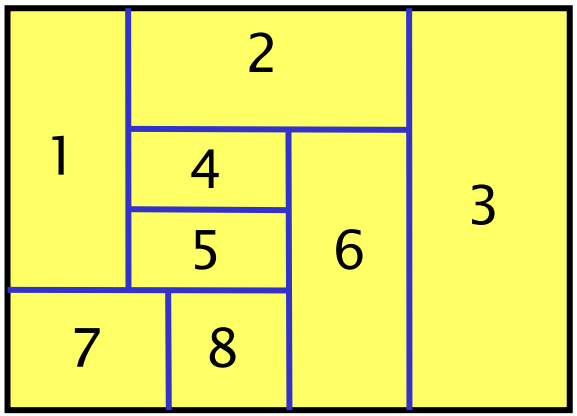
Non-Slicing floorplan

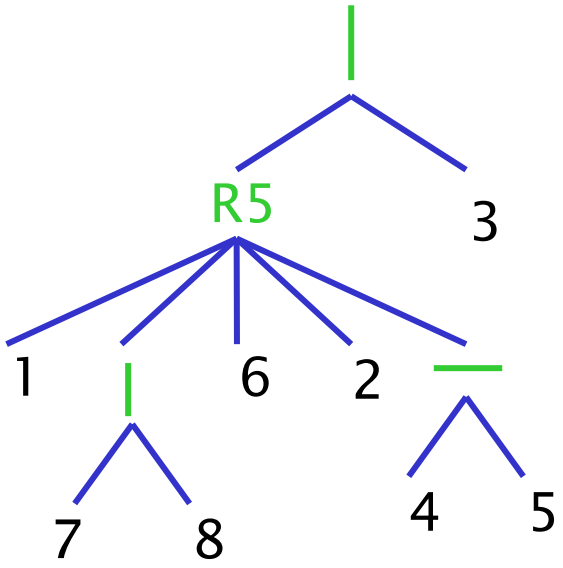
Non-slicing Floorplan Example

- Hierarchical floorplan of order 5
  - Templates



  - Floorplan and tree example





Review: Simulated Annealing Algorithm

```
curSoln = random initial solution;
T = temperature = hot;
while (T > Tfreeze) {
   for (s=1; s< #moves per temp;  s++) {
      nextSoln = perturb(curSoln);
      Compute Δcost = cost(nextSoln) - cost(curSoln);
      if (Δcost < 0 ) then curSoln = nextSoln;  //accept next solution
      else {
         if( uniform_random() < exp( -Δcost/T ) )
         then curSoln = nextSoln; //accept a new, worse solution
      }
   }
   T = coolDown(T);    // cool the temperature; do more gate swaps
}
return (curSoln as best solution);
```

Components of Simulated Annealing Algorithm

- Solution space (e.g., different slicing floorplans)
- Cost function (e.g., the area of a floorplan)
  - Determines how "good" a particular solution is
- Perturbation rules (e.g., transforming a floorplan to a new one)
- Simulated annealing engine
  - A temperature variable T
  - An initial temperature T_0 (e.g., $T_0 = 40{,}000$)
  - A freezing temperature T_{freeze} (e.g., $T_{freeze}=0.1$)
  - A cooling schedule (e.g., $T = 0.95 * T$)

Components of Floorplanning Algorithms

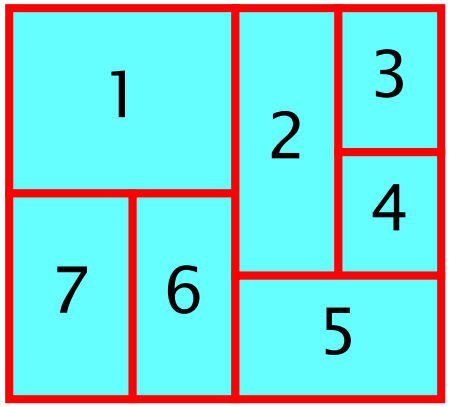
- Solution space representation: "Placeholder" representation
  - Defines the relative position of modules
  - Slicing class: slicing tree, reverse Polish notation
  - Non-slicing class: sequence pair, etc.
- Area cost calculation: floorplan sizing
  - Definition: given a floorplan tree, choose the best shape for each module to minimize area
  - Slicing: polynomial, bottom-up algorithm
  - Non-slicing: NP! Use mathematical programming (exact solution)
- Perturbation: going from one floorplan to another

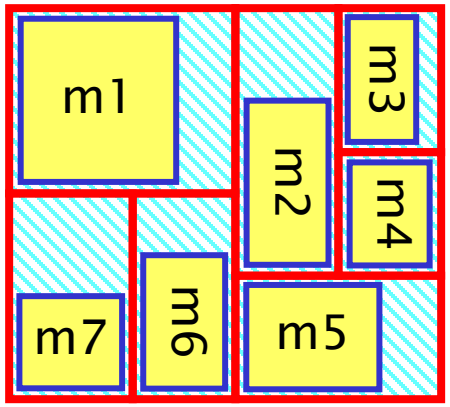
Outline

- Floorplanning
  - Basics
  - Placeholder Representation and Perturbation
  - Floorplan Sizing

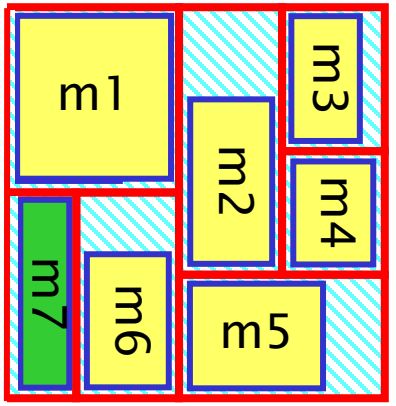
Placeholders and Soft Modules

- The hierarchy tree define "placeholders" for modules
- Area utilization
  - Depends on how nicely the hard modules' shapes are matched
  - Soft modules can take different shapes to "fill in" empty slots
    ➔ floorplan sizing





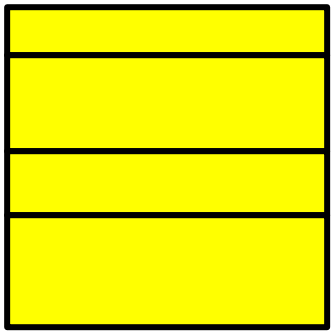
Area = 20x22 = 440



Area = 20x19 = 380

Bounds on Aspect Ratios

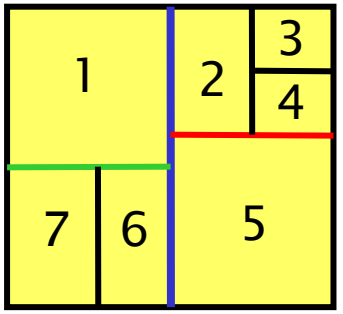
- If there is no bound on the aspect ratio (h_i/w_i) of each module, can we pack everything tightly?
  - Sure!

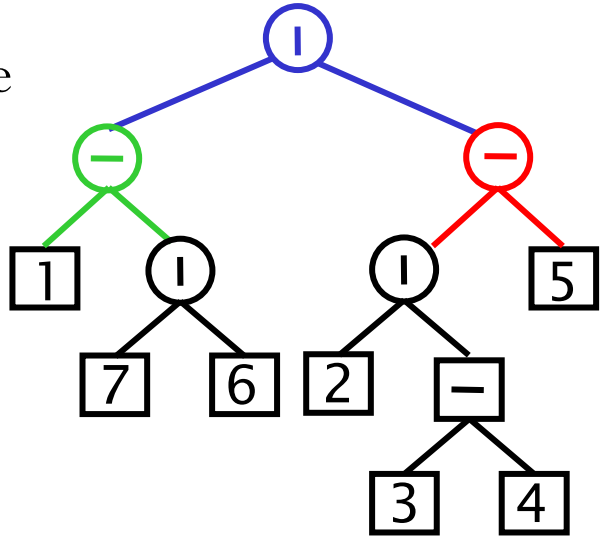


- But we don't want to layout blocks as long strips, so we have constraint $r_i \leq h_i/w_i \leq s_i$ for each soft module i

Reverse Polish Notation (RPN)

- Tree representation of the floorplan
  - Left (right) child of a V-cut in the tree represents the left (right) slice in the floorplan
  - Left (right) child of an H-cut in the tree represents the top (bottom) slice in the floorplan
- **Reverse Polish Notation (RPN)**
  - A string of symbols obtained by traversing a binary tree in post-order.

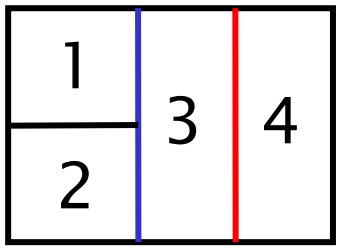


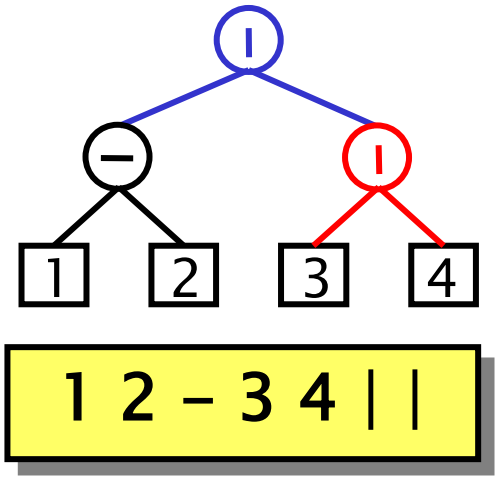


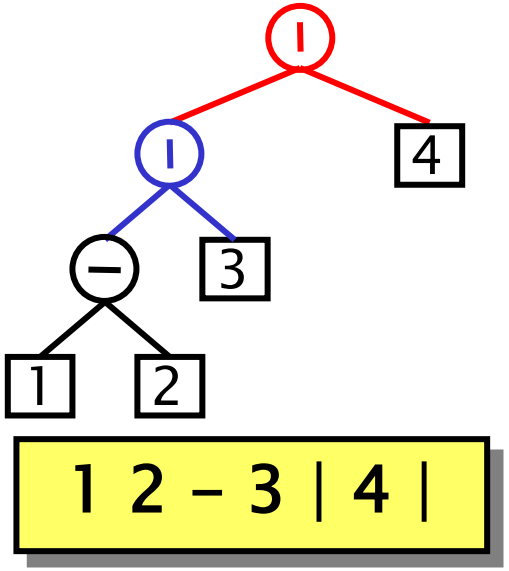
1 7 6 | – 2 3 4 – | 5 – |

Problem with Reverse Polish Notation?

- Multiple representations for some slicing trees
  - When more than one cut in one direction cut a floorplan
- Consequence #1: larger solution space on slicing trees
- Consequence #2: simulated annealing-based algorithm will be more biased towards floorplans with multiple representations

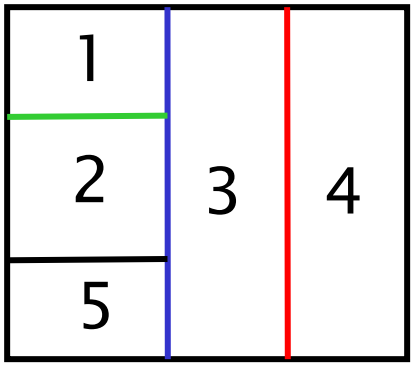


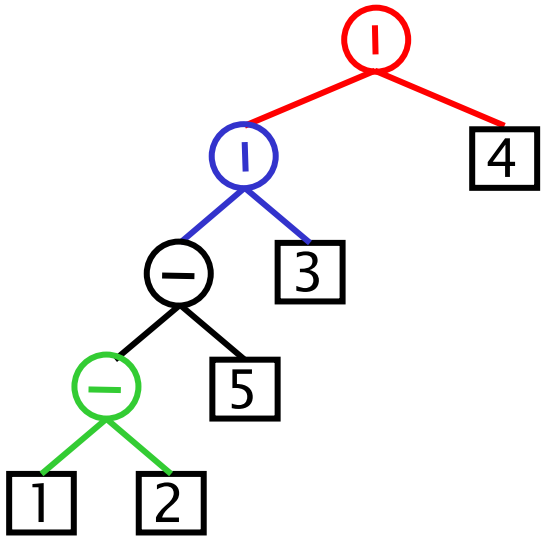




Solution: Normalized RPN

- Assign priorities to the cuts
- In a top-down tree construction, pick the **right-most** cut and the **lowest** cut first
- **Property**: no two same operators adjacent in the normalized RPN (i.e., no "| |" or "— —")
  - Not the case for general RPN

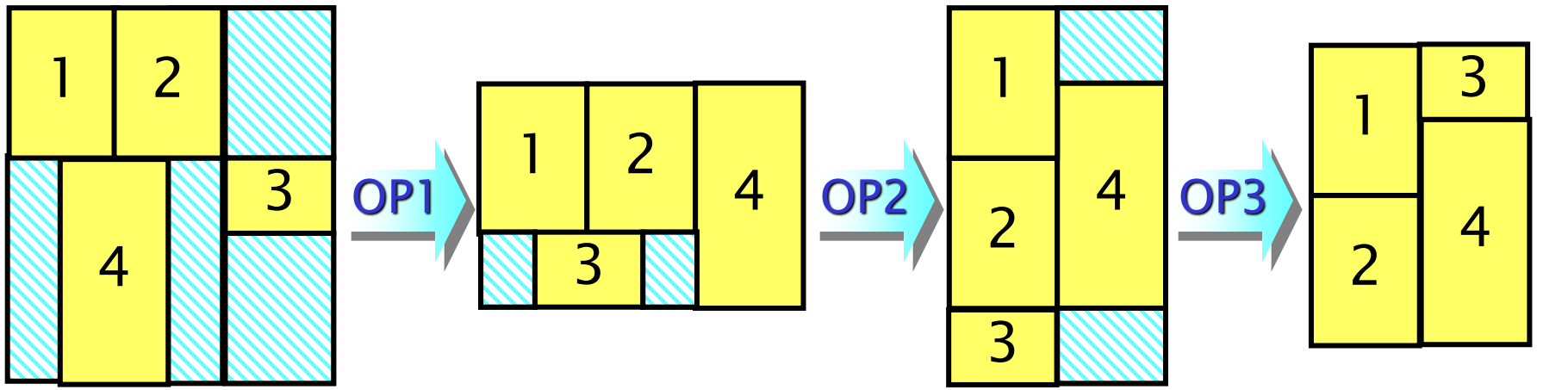




1 2 – 5 – 3 | 4 |

Perturbation

- OP1: Exchange two **operands** that have no other operands in between
- OP2: Complement a series of **operators** between two operands
- OP3: Exchange adjacent **operand and operator** if the resulting expression is still a normalized RPN

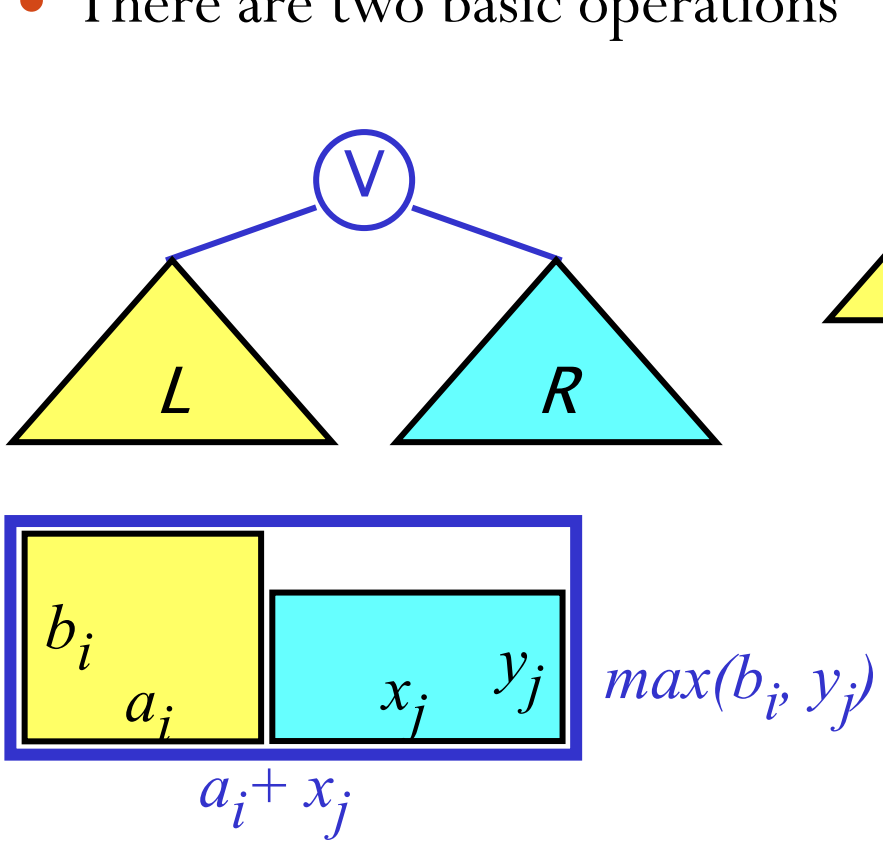


Outline

- Floorplanning
  - Basics
  - Placeholder Representation and Perturbation
  - Floorplan Sizing

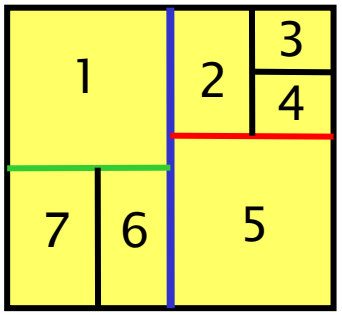
Floorplan Sizing for Slicing Floorplans

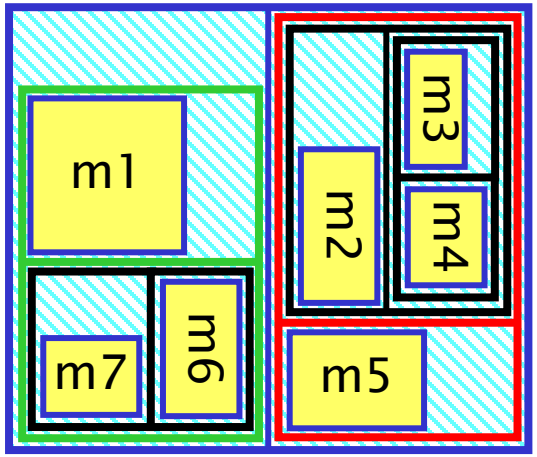
- Bottom-up process
- Has to be done per floorplan perturbation
- There are two basic operations

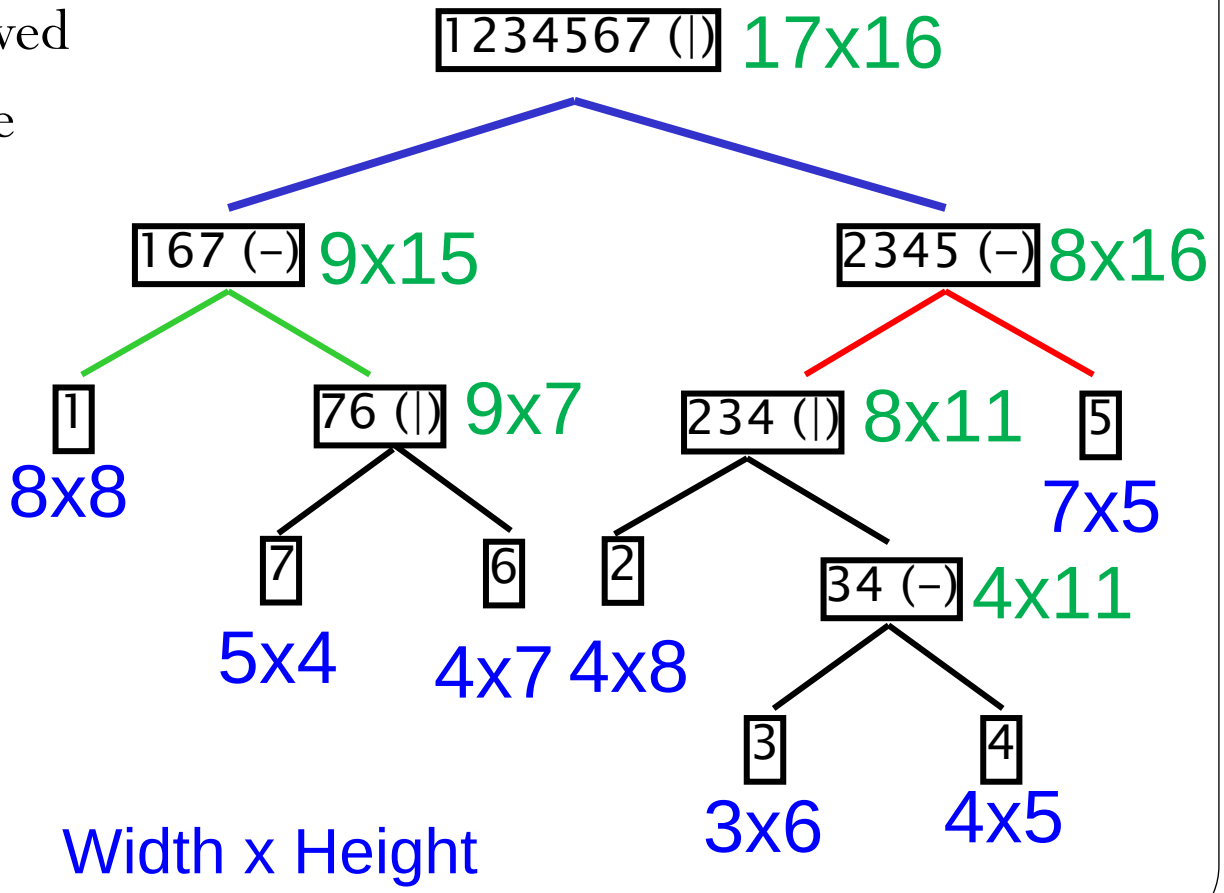


Sizing Slicing Floorplans

- Simple case:
  - All modules are hard macros
  - No rotation allowed
  - So, just one shape

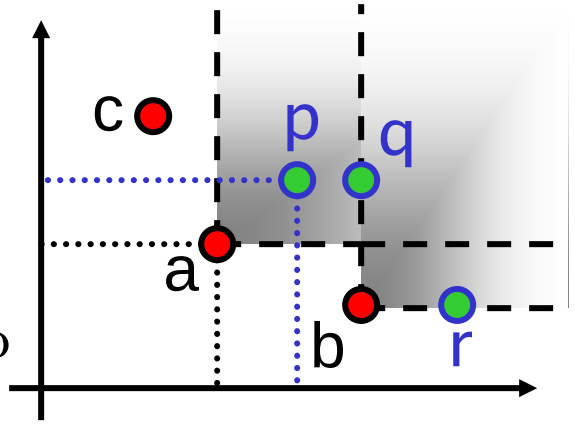


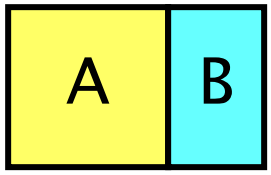




Sizing Slicing Floorplans with Multiple Module Shapes

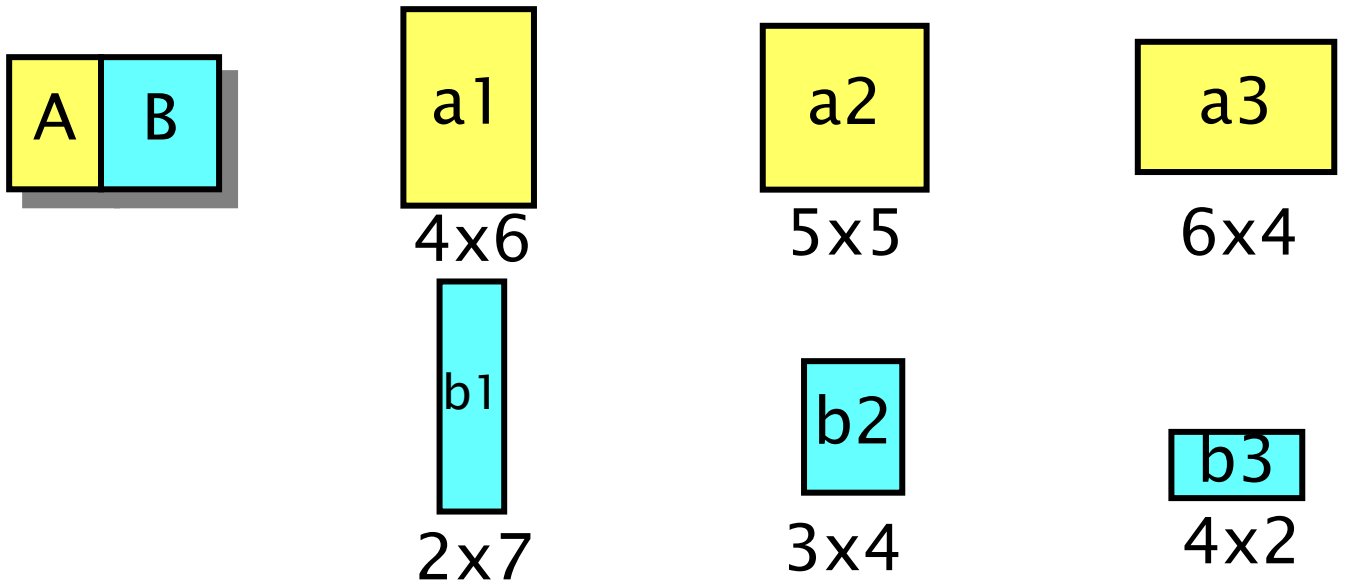
- What if modules have more than one shape?
- If area is the only concern:
  - Module A has shapes 4x6, 7x8, 5x6, 6x4, 7x4, which ones should we pick?
- Dominant points
  - Shape (x_1, y_1) dominates (x_2, y_2) if $x_1 \leq x_2$ and $y_1 \leq y_2$
- **Question**: can area along be a factor to eliminate a shape?
  - Consider module A has shapes 4x6, 5x5 and we have a B (3x5)



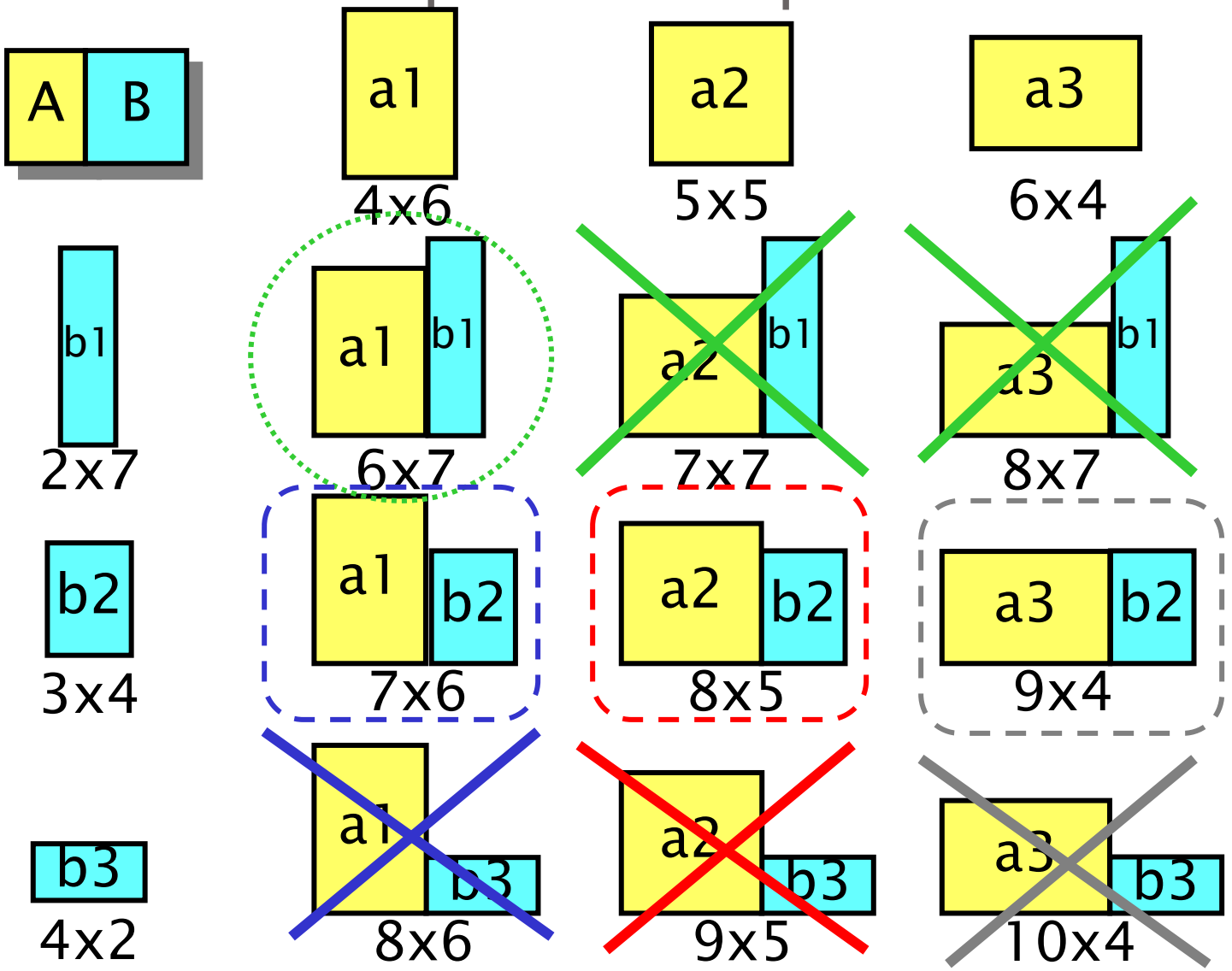


Sizing Slicing Floorplans with Multiple Module Shapes: Vertical Cut

- In general, module A and B have multiple shapes
- Sort these shapes in decreasing height
  - Claim: the width must be in increasing order. (Why?)

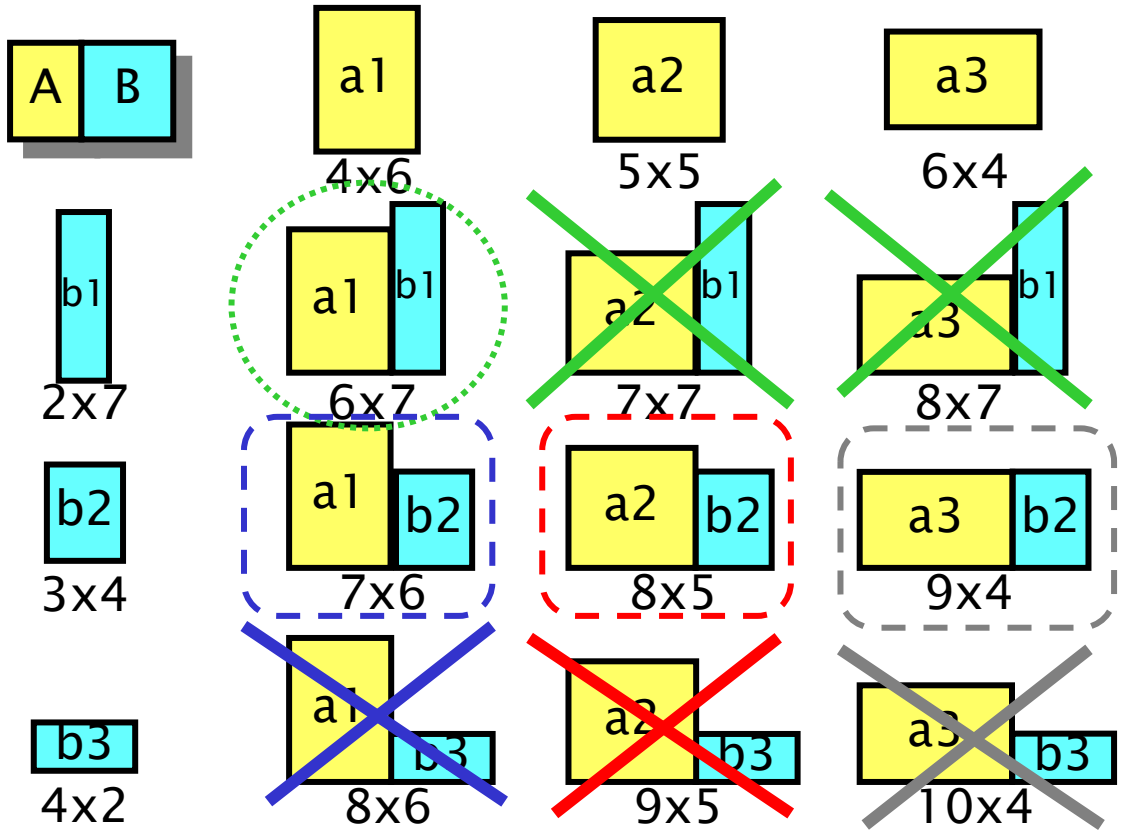


Sizing Slicing Floorplans with Multiple Module Shapes: Example



Analysis of the Example

Each dominating height will eliminate the elements to the right of an item in a row, or the elements to the bottom of an item in a column



Sizing Slicing Floorplans with Multiple Module Shapes: Algorithm

// Input: two sorted lists L={(a_1,b_1), ...,(a_s,b_s)}, R={(x_1, y_1), ..., // (x_t, y_t) }, where $a_i < a_{i+1}$, $b_i > b_{i+1}$, $x_i < x_{i+1}$, $y_i > y_{i+1}$ for all i
// Output: A sorted list H = {(c_1, d_1), ..., (c_u,d_u)},
// where $u \le s + t - 1$, $c_i < c_j$, $d_i > d_j$ for all $i < j$

```
Vertical_Node_Sizing{} {
    H = ϕ; i = 1; j = 1; k = 1;
    while ( (i ≤ s) and (j ≤ t) ) {
        (c_k, d_k) = (a_i + x_j, max(b_i, y_j))
        H = H ∪ { (c_k, d_k) }; k = k + 1;
        if  max(b_i, y_j) = b_i then  i = i + 1;
        if  max(b_i, y_j) = y_j then  j = j + 1;
    }
}
```

A_i higher than B_j

B_j higher than A_i

Note: the resulting shapes are also in increasing width/decreasing height order

Sizing Slicing Floorplans with Multiple Module Shapes: Algorithm

// Input: two sorted lists L={(a_1,b_1), ...,(a_s,b_s)}, R={(x_1, y_1), ..., // (x_t, y_t) }, where $a_i < a_{i+1}$, $b_i > b_{i+1}$, $x_i < x_{i+1}$, $y_i > y_{i+1}$ for all i
// Output: A sorted list H = {(c_1, d_1), ..., (c_u,d_u)},
// where $u \le s + t - 1$, $c_i < c_j$, $d_i > d_j$ for all $i < j$

```
Vertical_Node_Sizing{} {
    H = ϕ; i = 1; j = 1; k = 1;
    while ( (i ≤ s) and (j ≤ t) ) {
        (c_k, d_k) = (a_i + x_j, max(b_i, y_j))
        H = H ∪ { (c_k, d_k) }; k = k + 1;
        if  max(b_i, y_j) = b_i then  i = i + 1;
        if  max(b_i, y_j) = y_j then  j = j + 1;
    }
}
```

What happens if $b_i == y_j$? Would the algorithm still work correctly?

What's the time complexity?

How about the horizontal cut?

Sizing Slicing Floorplan: Algorithm

- Input: floorplan tree, modules shapes
- Start with sorted shape lists of modules
- In a bottom-up fashion, perform:
  - Vertical_Node_Sizing or Horizontal_Node_Sizing
- When get to the root node, we have a list of shapes. Select the one that is best in terms of area
- In a top-down fashion, traverse the floorplan tree and set module sizes
- Time complexity is O(n), where n is the sum of the number of shapes of all the modules