CSE 554

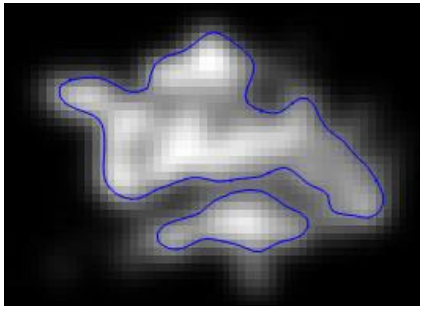
Lecture 5: Contouring (faster)

Fall 2018

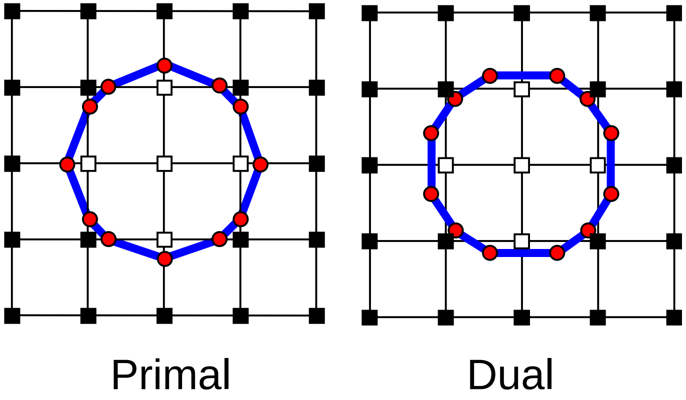
Review

Washington University in St.Louis
School of Engineering & Applied Science

- Iso-contours
  - Points where a function evaluates to be a given value (iso-value)
    - Smooth thresholded boundaries



- Contouring methods
  - Primal methods
    - Marching Squares (2D) and Cubes (3D)
    - Placing vertices on grid edges
  - Dual methods
    - Dual Contouring (2D,3D)
    - Placing vertices in grid cells



Efficiency

- Iso-contours are often used for visualizing (3D) medical images
  - The data can be large (e.g, 512^3)
  - The user often wants to change iso-values and see the result in real-time
- An efficient contouring algorithm is needed
  - Optimized for viewing one volume at multiple iso-values

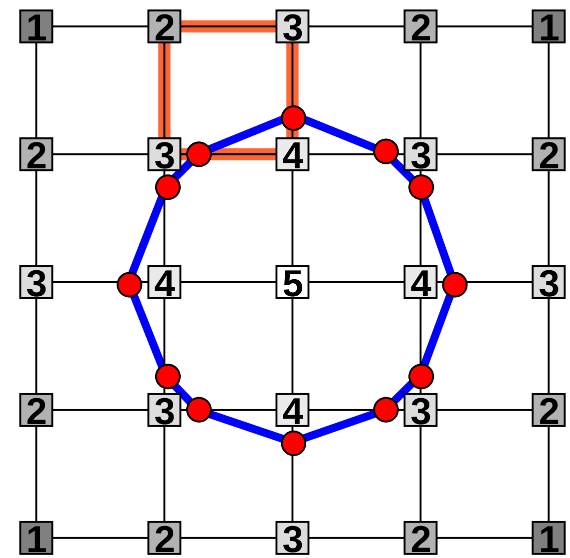
Marching Squares - Revisited

- Active cell/edge
  – A grid cell/edge where {min, max} of its corner/end values encloses the iso-value
- Algorithm

O(n) ⇨ Visit each cell.

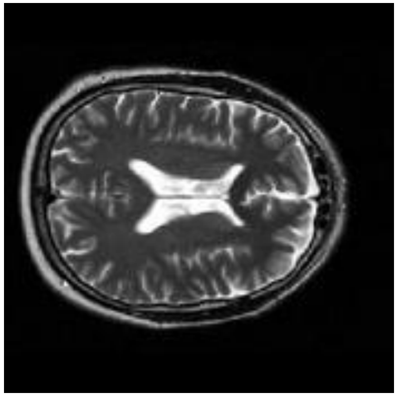
O(k) ⇨ If active, create vertices on active edges and connect them by lines

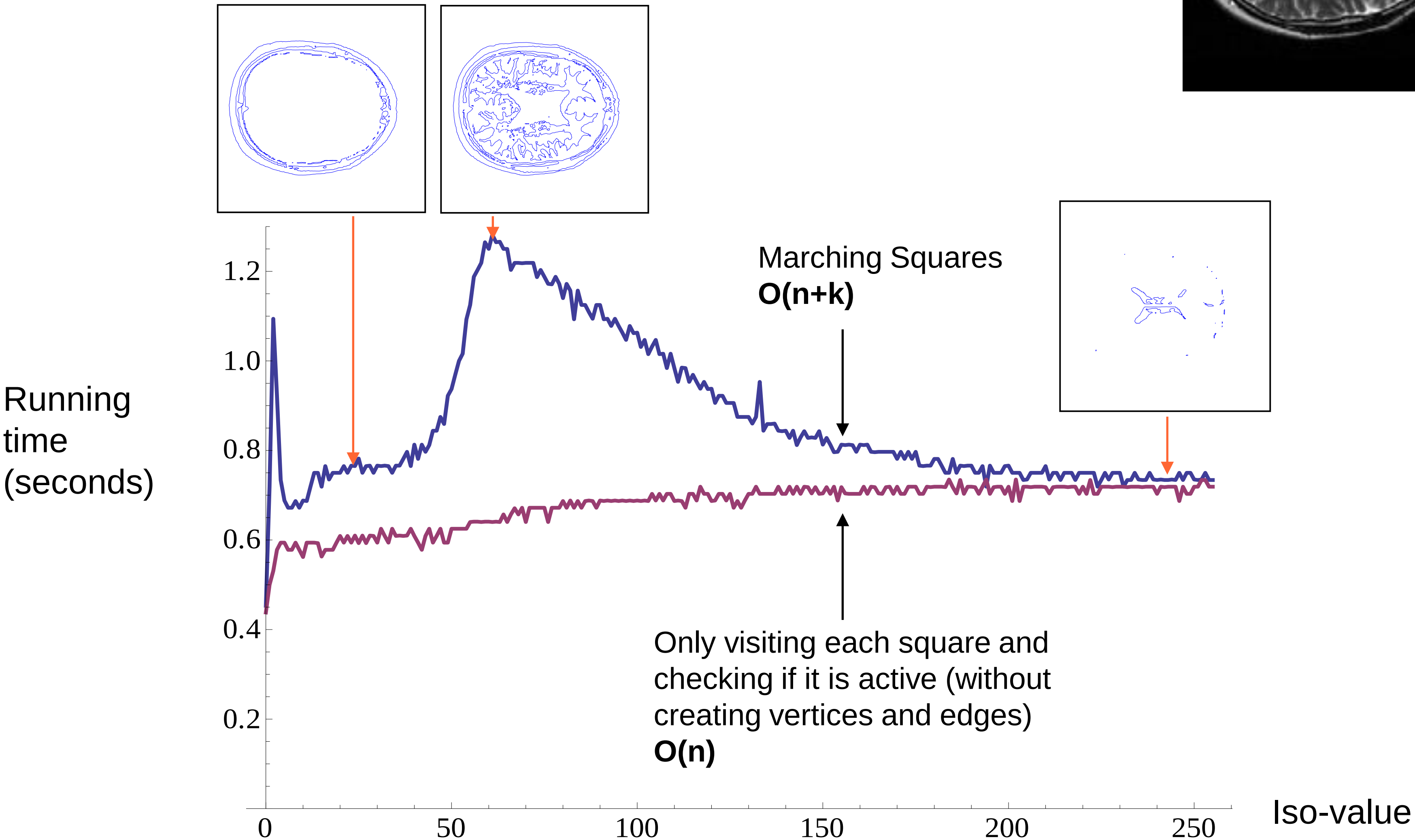
- Time complexity? O(n+k)
  - n: the number of all grid cells
  - k: the number of active cells (usually <<n)



Iso-value = 3.5

Marching Squares - Revisited





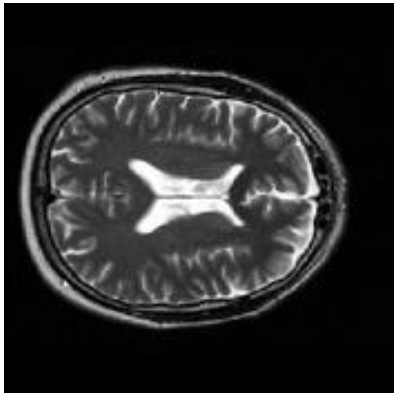
Speeding Up Contouring

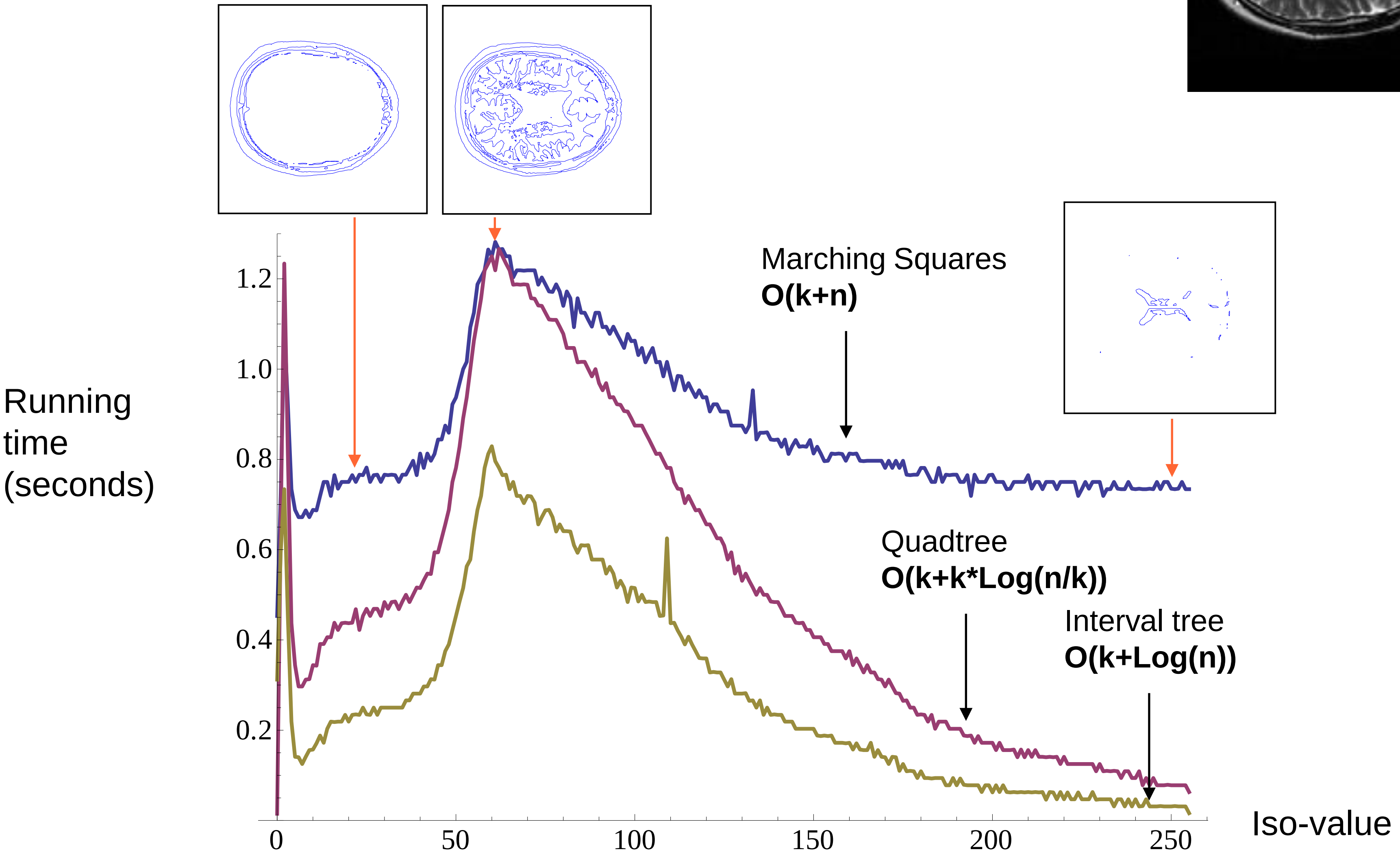
- Can we make the algorithm’s complexity *output sensitive*?
  - More dependent on **k**, rather then **n**
  - Need a faster way to locate active cells instead of a global scan

Speeding Up Contouring

- Precompute a data structure for fast active-cell queries
  - Quadtrees (2D), Octrees (3D)
    - Hierarchical spatial structures for quickly pruning large areas of inactive cells
    - Complexity: O(k+k*Log(n/k))
  - Interval trees
    - An optimal data structure for range finding
    - Complexity: O(k+Log(n))

Overview





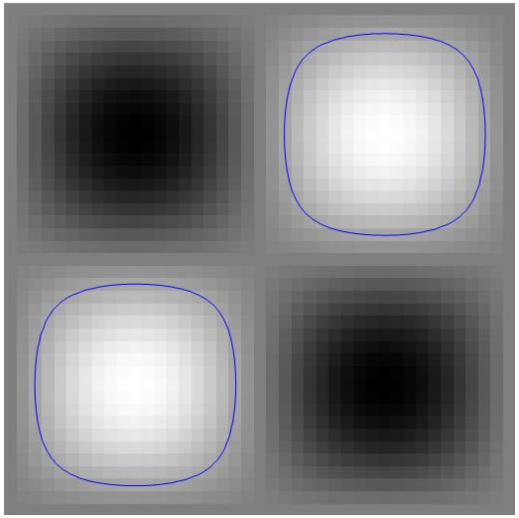
Running time (seconds)
1.2
1.0
0.8
0.6
0.4
0.2
0
50
100
150
200
250
Iso-value
Marching Squares
O(k+n)
Quadtree
O(k+k*Log(n/k))
Interval tree
O(k+Log(n))

Speeding Up Contouring

- Precompute a data structure for fast active-cell queries
  - Quadtrees (2D), Octrees (3D)
    - Hierarchical spatial structures for quickly pruning large areas of inactive cells
    - Complexity: O(k+k*Log(n/k))
  - Interval trees
    - An optimal data structure for range finding
    - Complexity: O(k+Log(n))

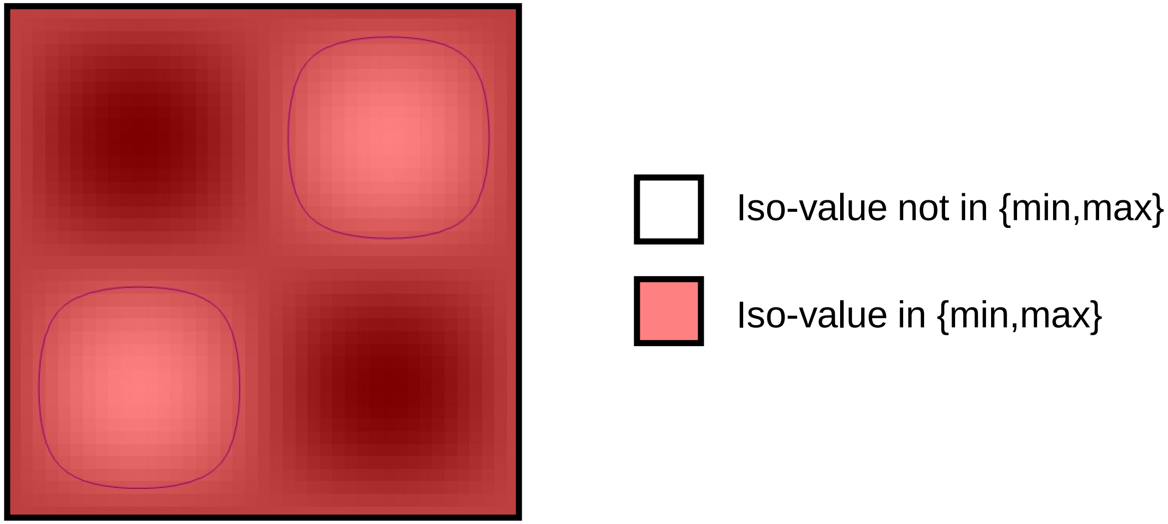
Quadtrees (2D)

- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



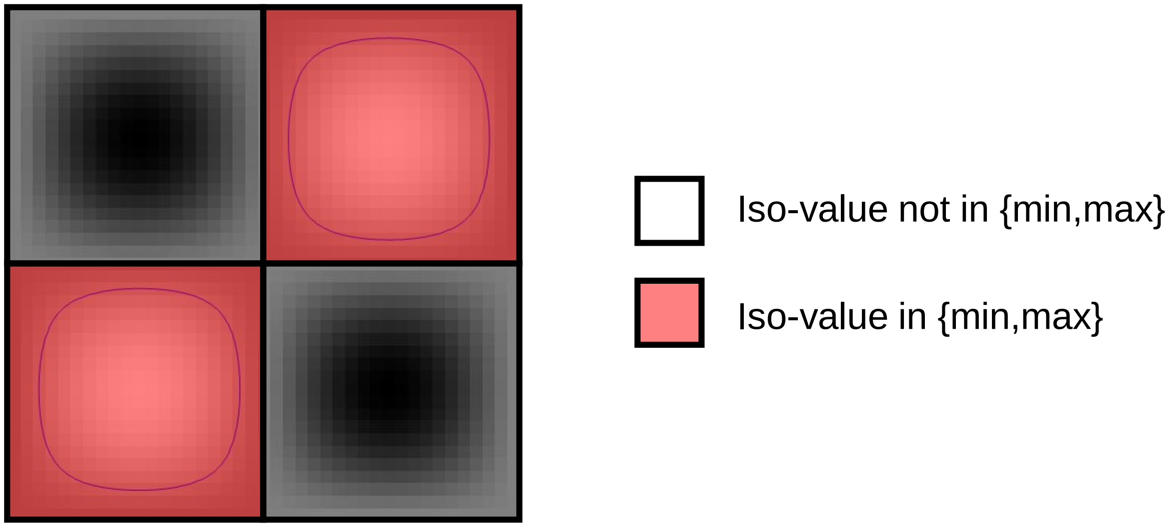
Quadtrees (2D)

- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



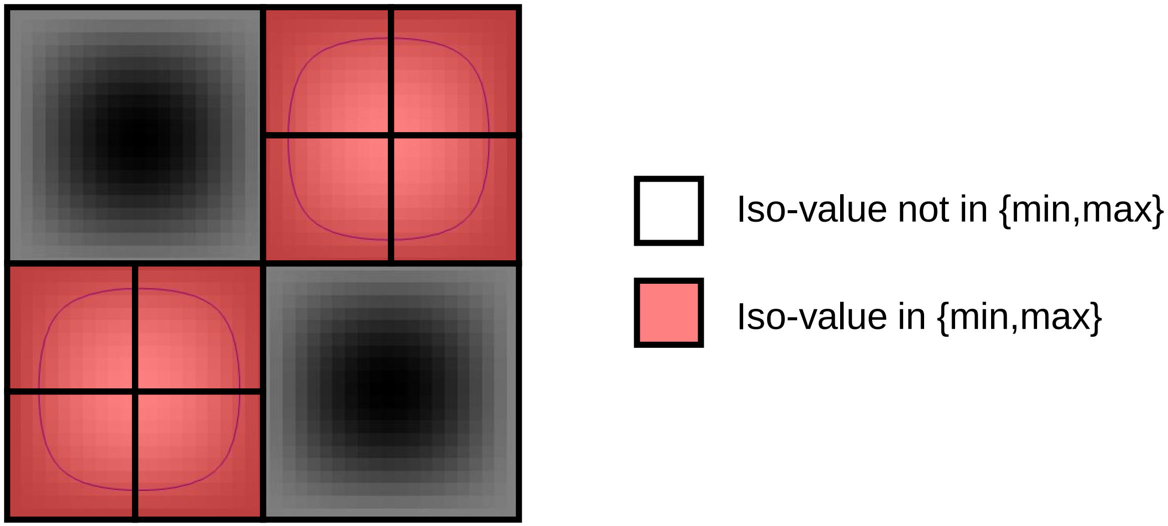
Quadtrees (2D)

- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



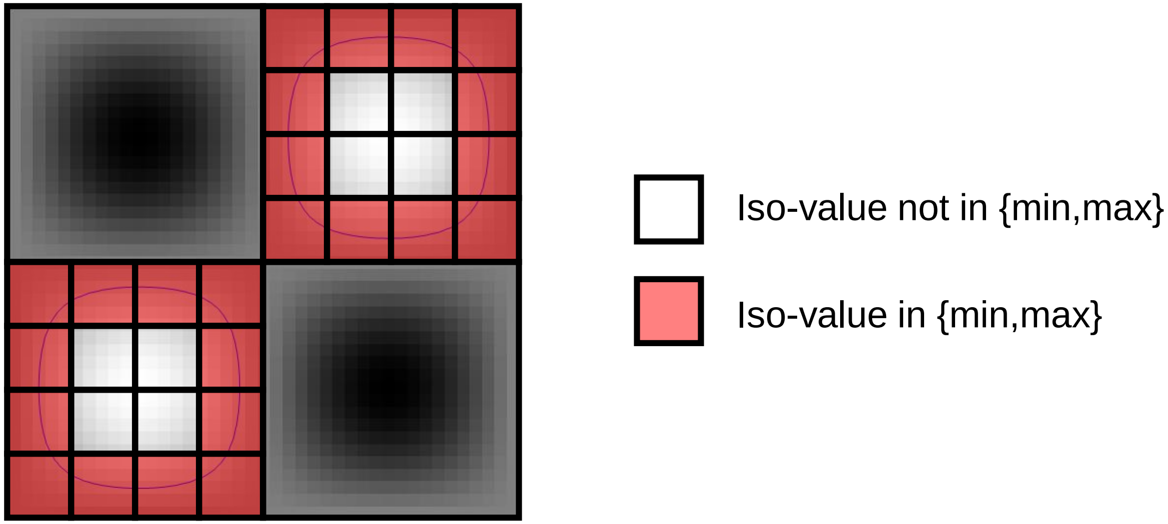
Quadtrees (2D)

- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



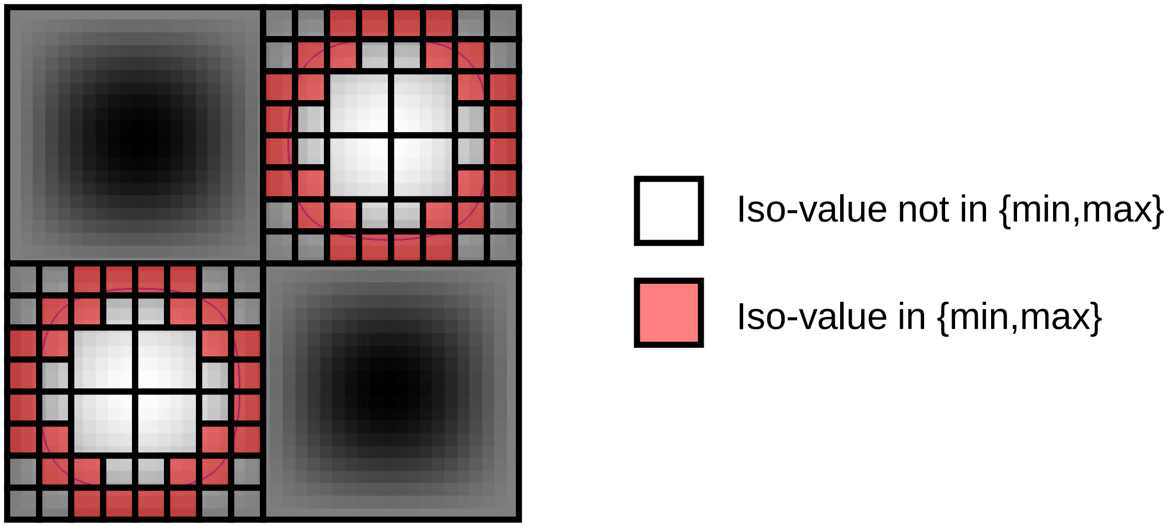
Quadtrees (2D)

- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



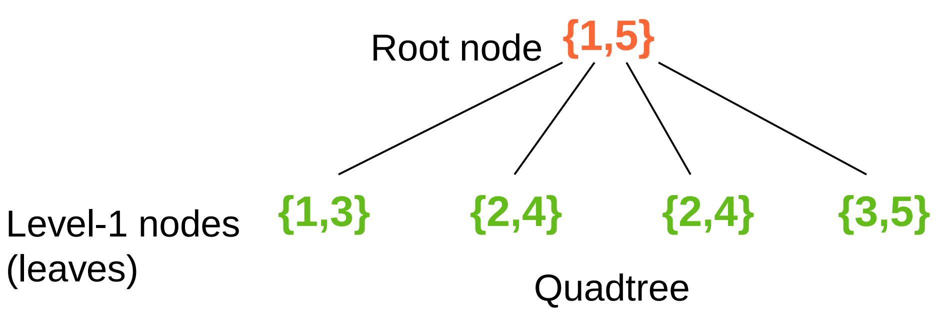
Quadtrees (2D)

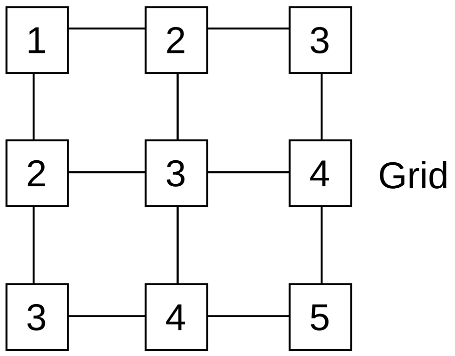
- Basic idea: coarse-to-fine search
  - Start with the entire grid:
    - If the {min, max} of all grid values does not enclose the iso-value, stop.
    - Otherwise, divide the grid into four sub-grids and repeat the check within each sub-grid. If the sub-grid is a unit cell, it's an active cell.



Quadtrees (2D)

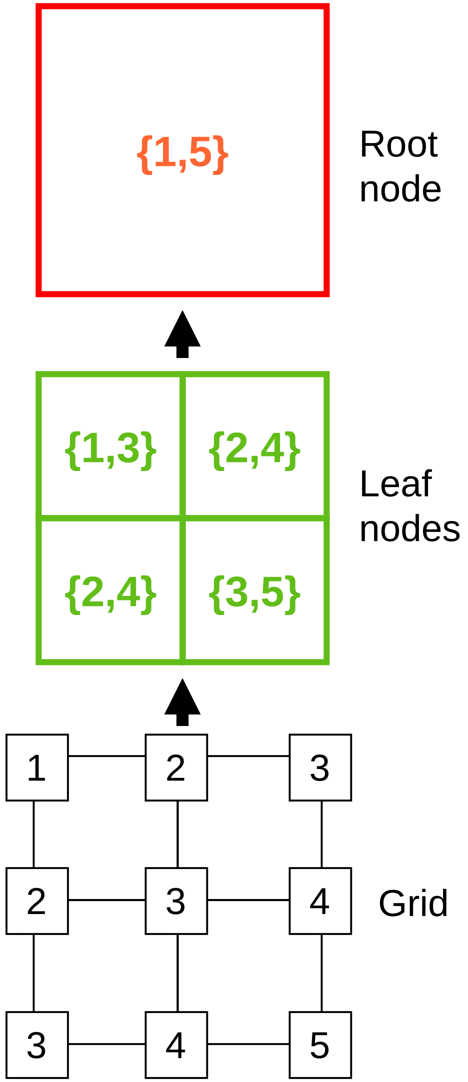
- A degree-4 tree
  - Root node represents the entire image
  - Each node represents a square part of the image, and stores:
    - {min, max} in the square
    - (in a non-leaf node) one child node for each quadrant of the square





Quadtrees (2D)

- Tree building: bottom-up
  - Each grid cell becomes a leaf node
  - Assign a parent node to every group of 4 nodes
    - Compute the {min, max} of the parent node from {min, max} of the children nodes
    - Padding dummy leaf nodes (e.g., {∞,∞}) if the dimension of grid is not power of 2



Quadtrees (2D)

- Tree building: a recursive algorithm

```
// A recursive function to build a single quadtree node
// for a sub-grid at corner (lowX, lowY) and with length len
buildSubTree (lowX, lowY, len)
1. If (lowX, lowY) is out of range: Return a leaf node with {∞,∞}
2. If len = 1: Return a leaf node with {min, max} of corner values
3. Else
     1. c1 = buildSubTree (lowX, lowY, len/2)
     2. c2 = buildSubTree (lowX + len/2, lowY, len/2)
     3. c3 = buildSubTree (lowX, lowY + len/2, len/2)
     4. c4 = buildSubTree (lowX + len/2, lowY + len/2, len/2)
     5. Return a node with children {c1,…,c4} and {min, max} of all
        {min, max} of these children
```

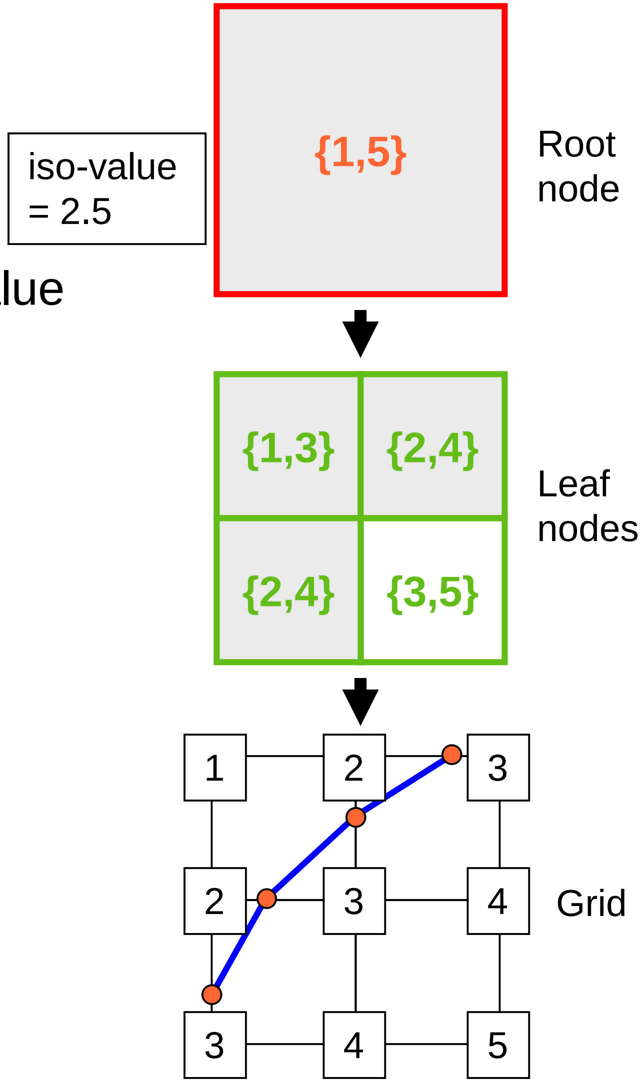
```
// Building a quadtree for a grid whose longest dimension is len
Return buildSubTree (1, 1, 2^Ceiling[Log[2,len]])
```

Quadtrees (2D)

- Tree-building: time complexity?
  - O(n) – proportional to # nodes of tree
- Pre-computation
  - We only need to do this once (after that, the tree can be used to speed up contouring at any iso-value)

Quadtrees (2D)

- Contouring with a quadtree: top-down
  - Starting from the root node
  - If {min, max} of the node encloses the iso-value
    - If it is not a leaf, continue onto its children
    - If the node is a leaf, contour in that grid cell



Quadtrees (2D)

- Contouring with a quadtree: a recursive algorithm

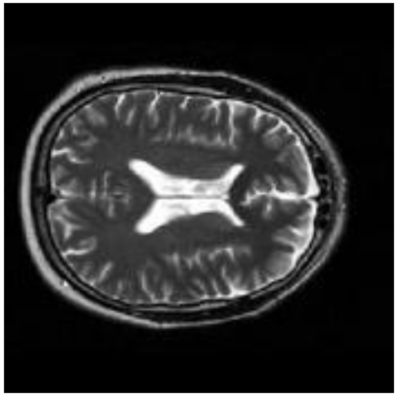
```
// A recursive function to contour in a quadtree node
// for a sub-grid at corner (lowX, lowY) and with length len
ctSubTree (node, iso, lowX, lowY, len)
1. If iso is within {min, max} of node
     1. If len = 1: do Marching Squares in this grid square
     2. Else (let the 4 children be c1,…,c4)
          1. ctSubTree (c1, iso, lowX, lowY, len/2)
          2. ctSubTree (c2, iso, lowX + len/2, lowY, len/2)
          3. ctSubTree (c3, iso, lowX, lowY + len/2, len/2)
          4. ctSubTree (c4, iso, lowX + len/2, lowY + len/2, len/2)
```

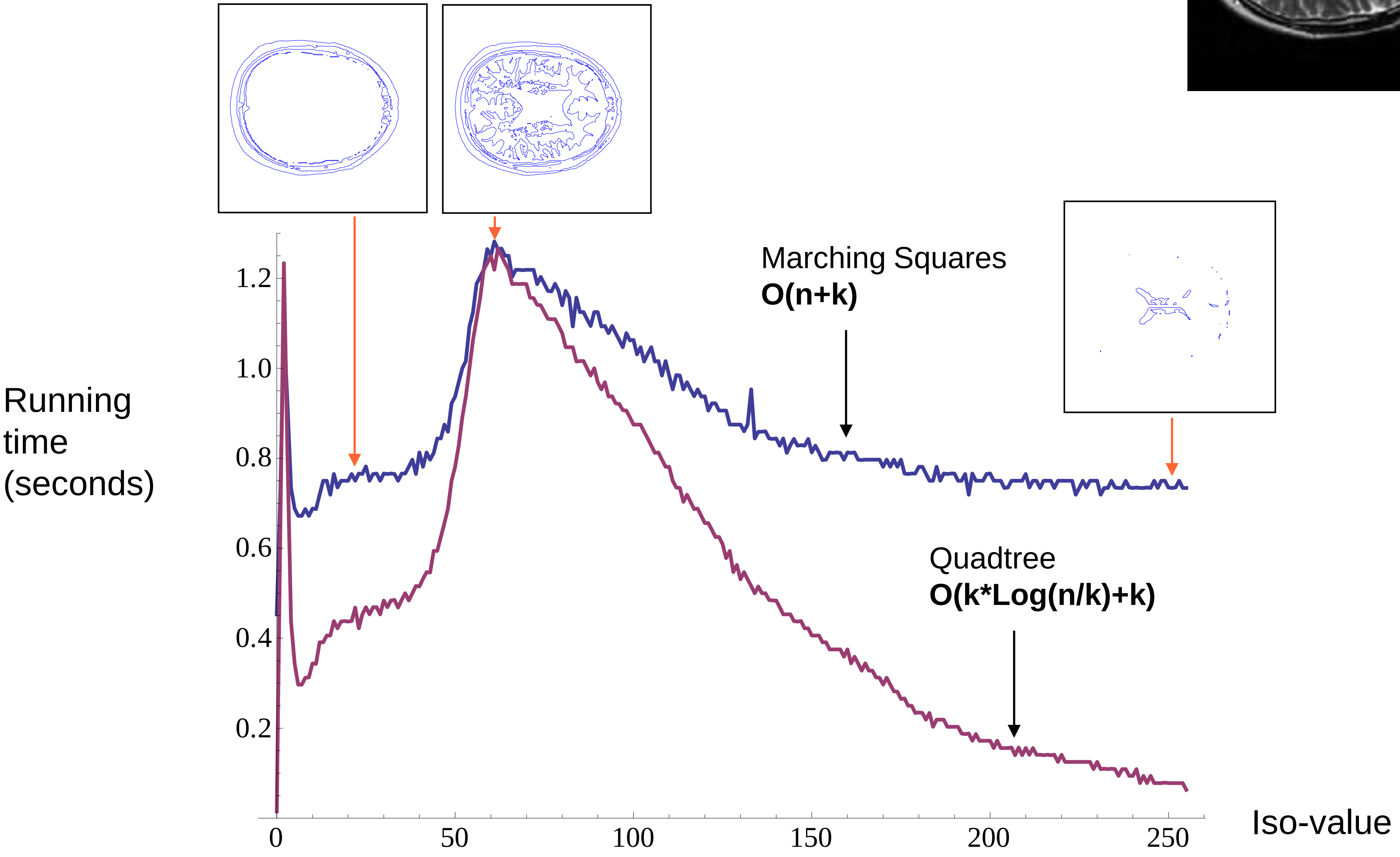
```
// Contouring using a quadtree whose root node is Root
// for a grid whose longest dimension is len
ctSubTree (Root, iso, 1, 1, 2^Ceiling[Log[2,len]])
```

Quadtrees (2D)

- Contouring with a quadtree: time complexity?
  - O(k+k*Log(n/k)) – # of nodes in the rooted subtree that contains only the active leaf nodes
  - Faster than O(k+n) (Marching Squares) if k<<n
  - But not efficient when k is large

Quadtrees (2D)





Quadtrees (2D): Summary

- Preprocessing: building the tree bottom-up
  - Independent of iso-values
- Contouring: traversing the tree top-down
  - For a specific iso-value
- Both are recursive algorithms

Octrees (3D)

Washington University in St. Louis
School of Engineering & Applied Science

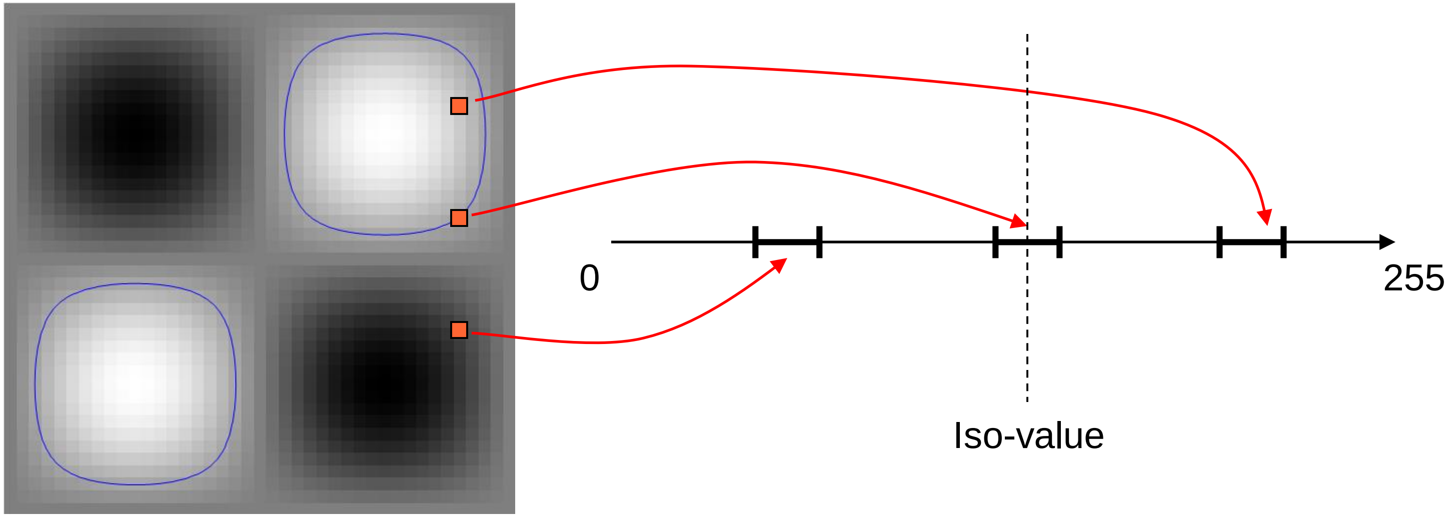
- Each tree node has 8 children nodes
- Similar algorithms and same complexity as quadtrees

Speeding Up Contouring

- Precompute a data structure for fast active-cell queries
  - Quadtrees (2D), Octrees (3D)
    - Hierarchical spatial structures for quickly pruning large areas of inactive cells
    - Complexity: O(k+k*Log(n/k))
  - Interval trees
    - An optimal data structure for range finding
    - Complexity: O(k+Log(n))

Interval Trees

- Basic idea
  - Each grid cell occupies an interval of values {min, max}
    - An active cell's interval intersects the iso-value
  - Stores all intervals in a search tree for efficient intersection queries



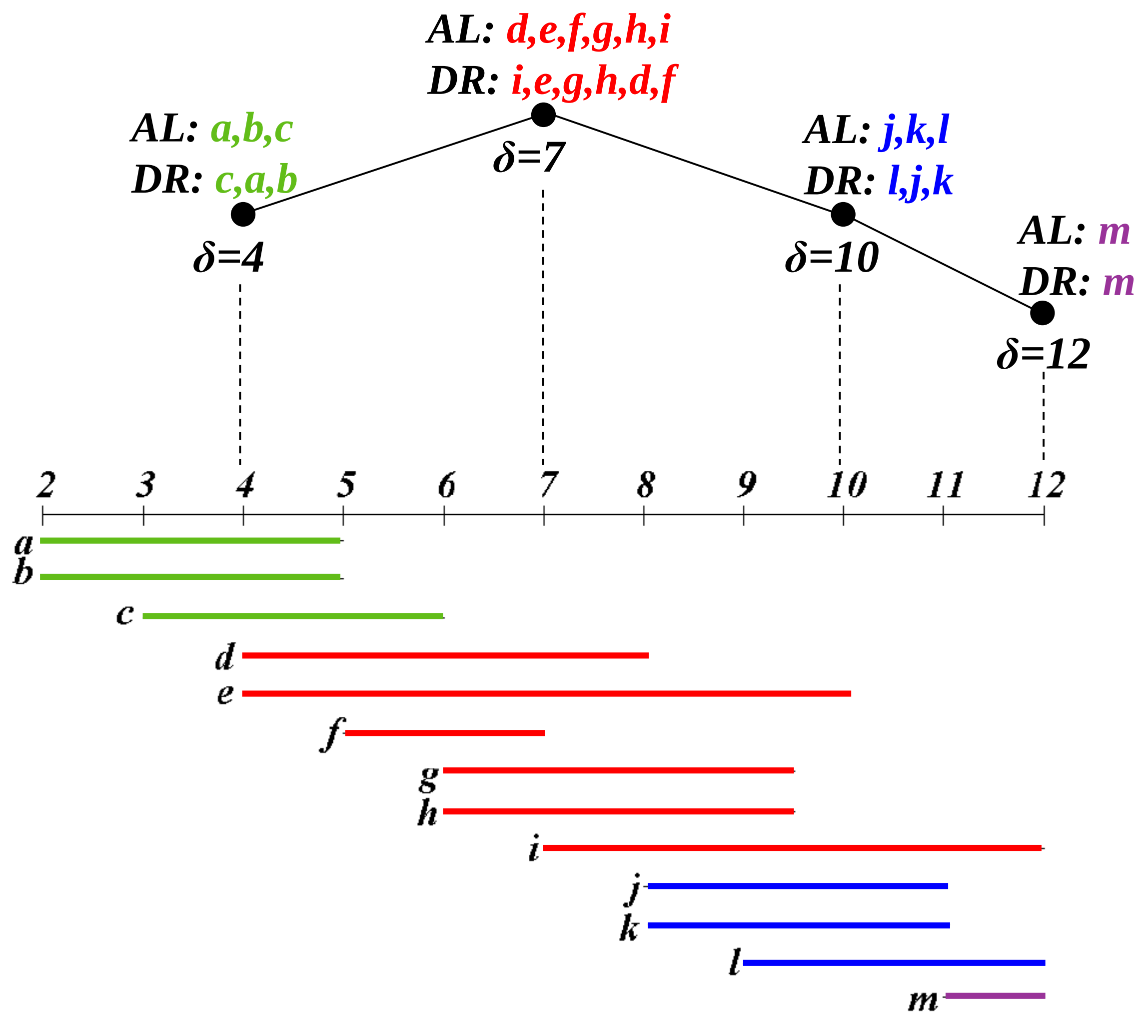
Interval Trees

- A binary tree
  - Root node: all intervals in the grid
  - Each node maintains:
    - δ: Median of end values of all intervals (used to split the children intervals)
    - (in a non-leaf node) Two child nodes
      - **Left child**: intervals < δ
      - **Right child**: intervals > δ
    - Two sorted lists of intervals intersecting δ
      - **Left list (AL)**: ascending order of min-end of each interval
      - **Right list (DR)**: descending order of max-end of each interval

Interval Trees

Interval tree:

Intervals:

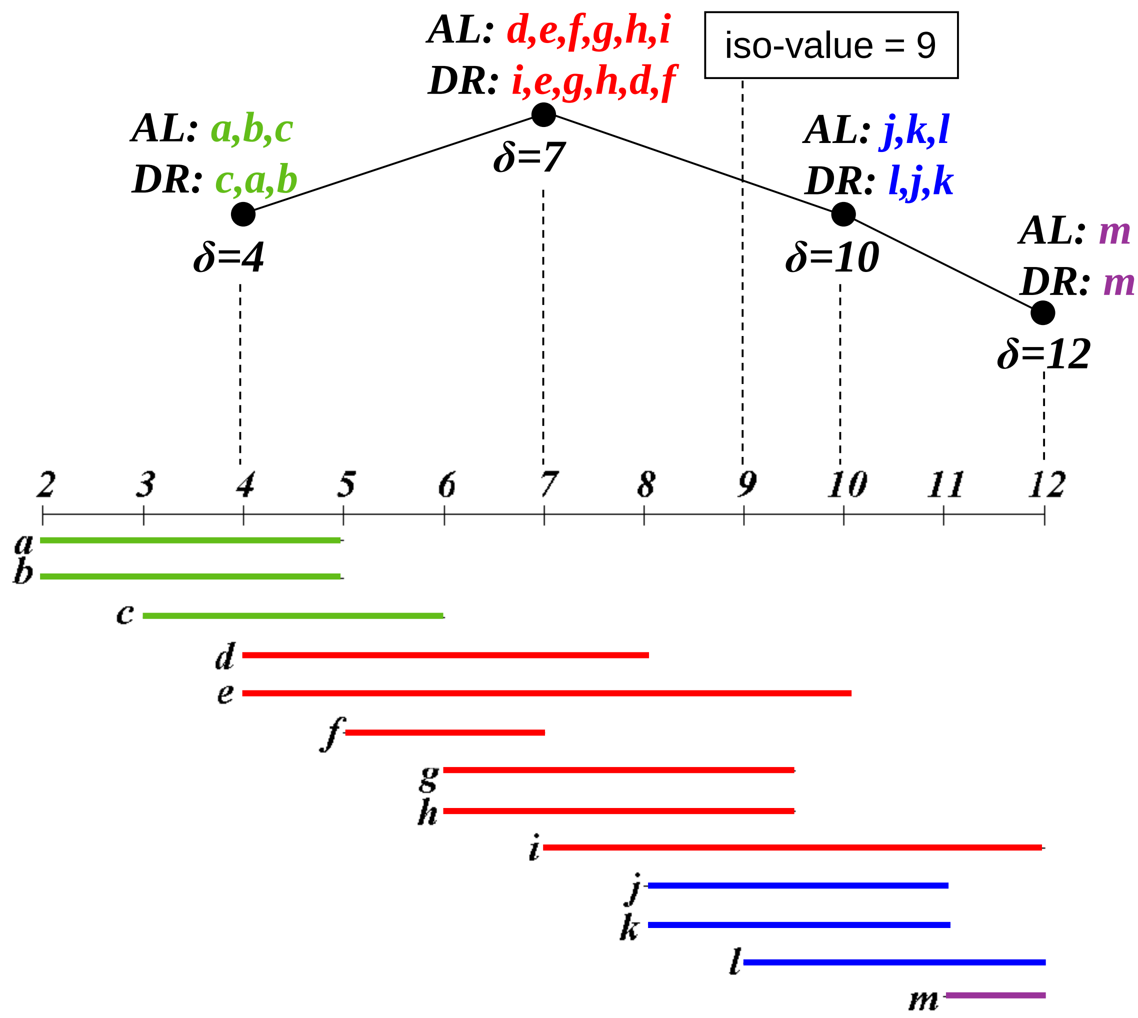


Interval Trees

- Intersection queries: top-down
  - Starting with the root node
    - If the iso-value is smaller than median **δ**
      - Scan through **AL**: output all intervals whose min-end <= iso-value.
      - Go to the **left child**.
    - If the iso-value is larger than **δ**
      - Scan through **DR**: output all intervals whose max-end >= iso-value.
      - Go to the **right child**.
    - If iso-value equals **δ**
      - Output **AL** (or **DR**).

Interval Trees

Interval tree:

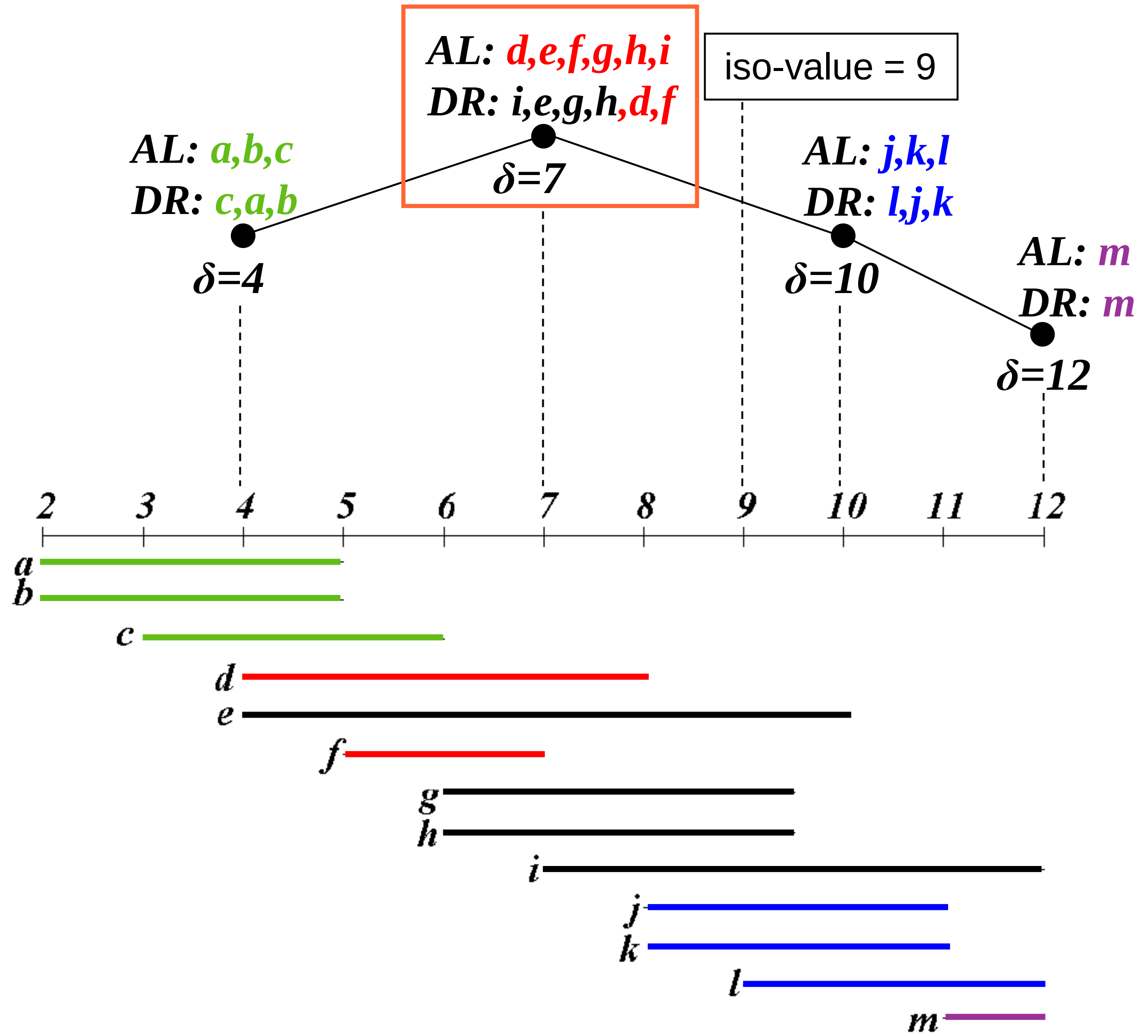


Intervals:

Interval Trees

Interval tree:

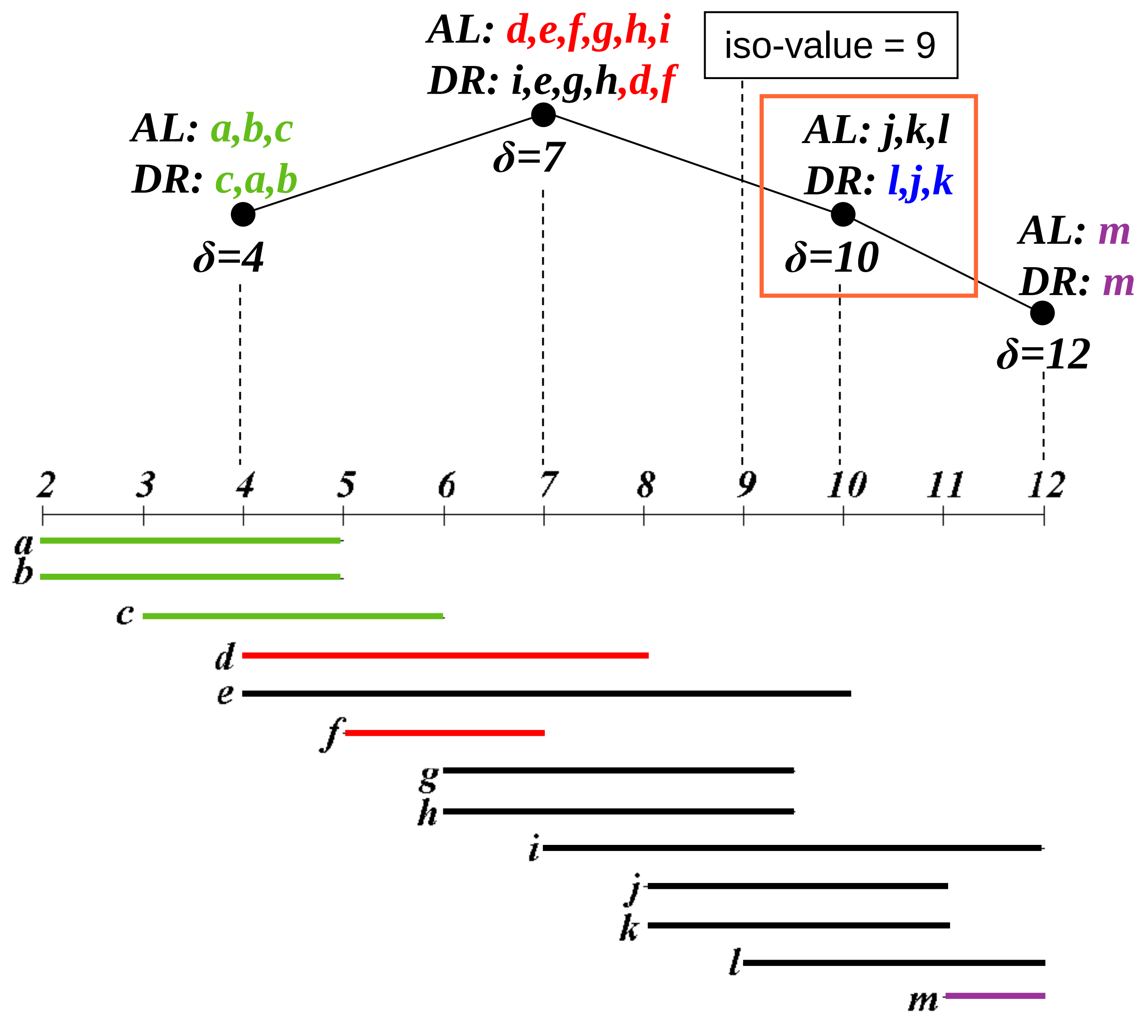
Intervals:



Interval Trees

Interval tree:

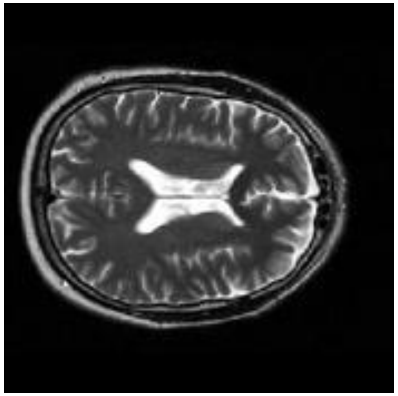
Intervals:

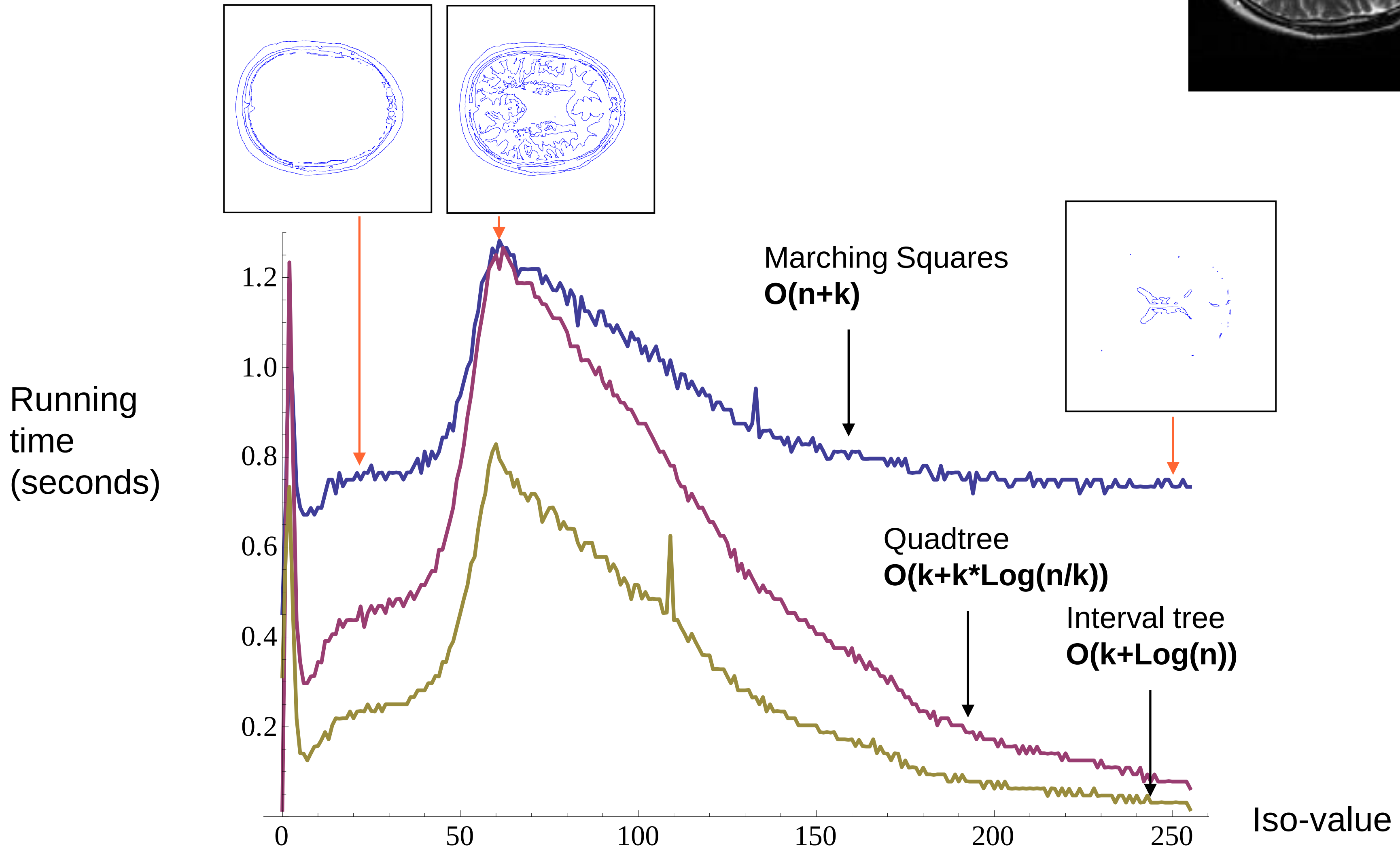


Interval Trees

- Intersection queries: time complexity?
  - O(k + Log(n)) - # active intervals + depth of tree
    - Walks down the tree along a single path! (unlike quadtree/octree)
  - Much faster than O(k+n) (Marching squares/cubes)

Interval Trees





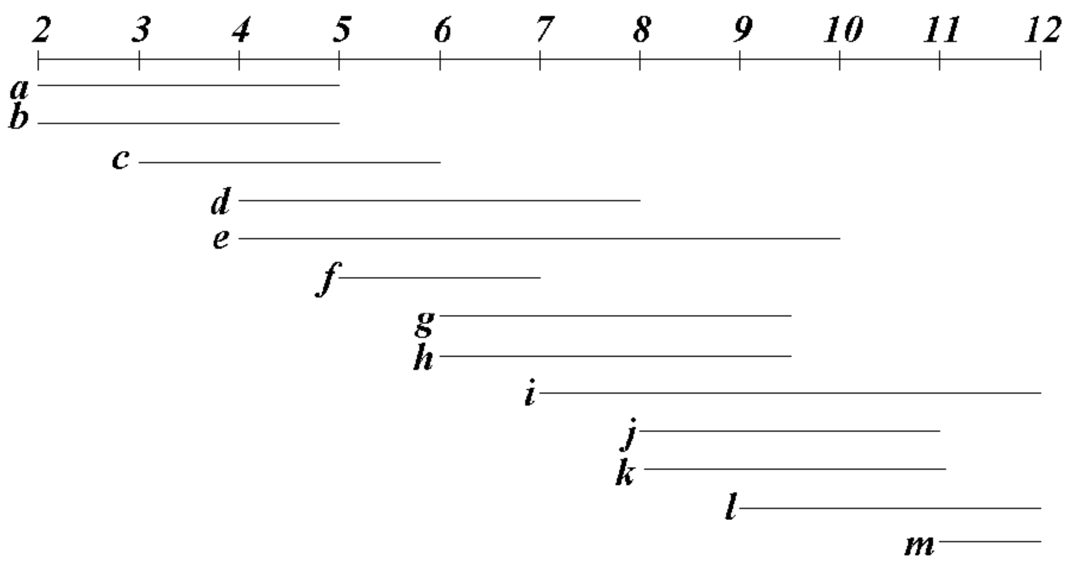
Interval Trees

- Tree building: top-down
  - Starting with the root node (which includes all intervals)
  - To create a node from a set of intervals,
    - Find δ (the median of all ends of the intervals).
    - Sort all intervals intersecting with δ into the **AL, DR**
    - Construct the **left** (**right**) **child** from intervals strictly below (above) δ
  - A recursive algorithm

Interval Trees

Interval tree:

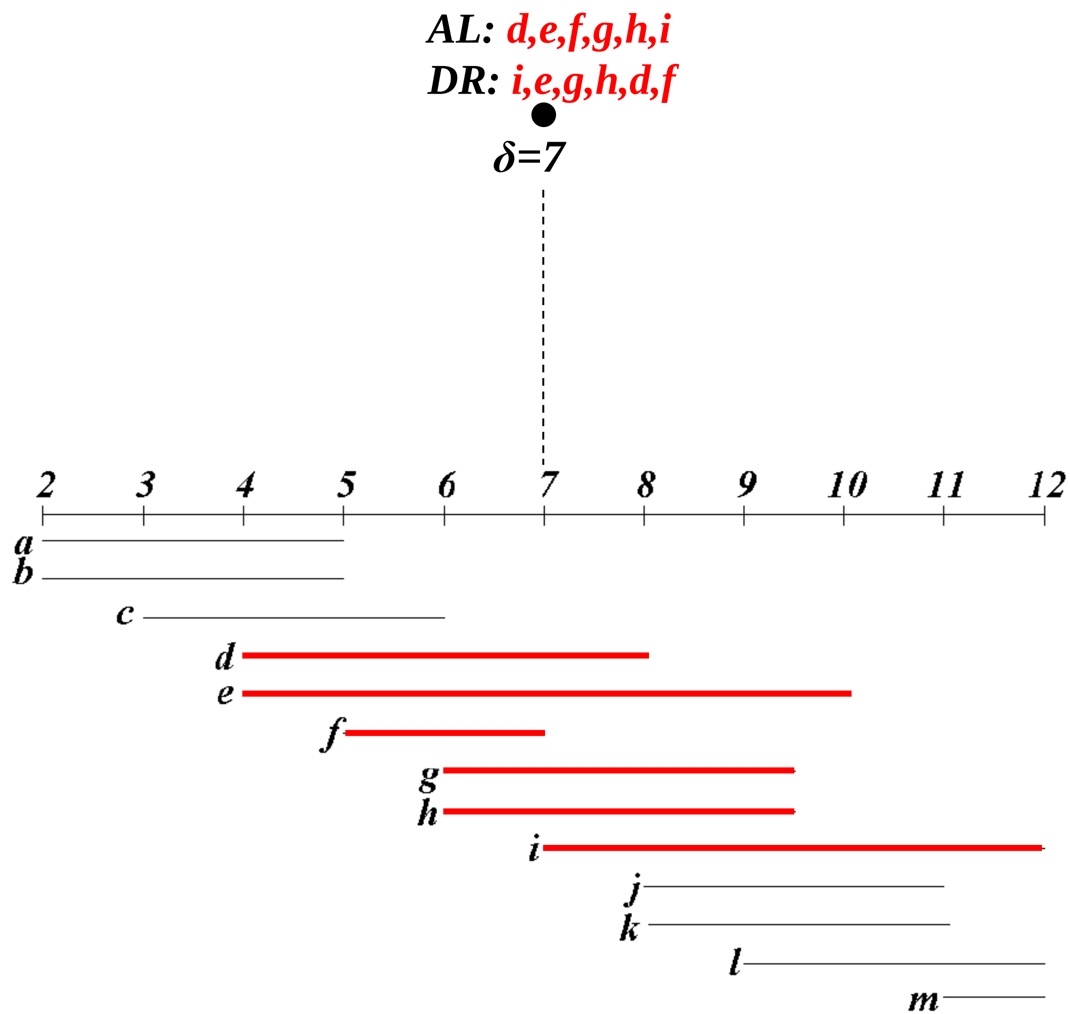
Intervals:



Interval Trees

Interval tree:

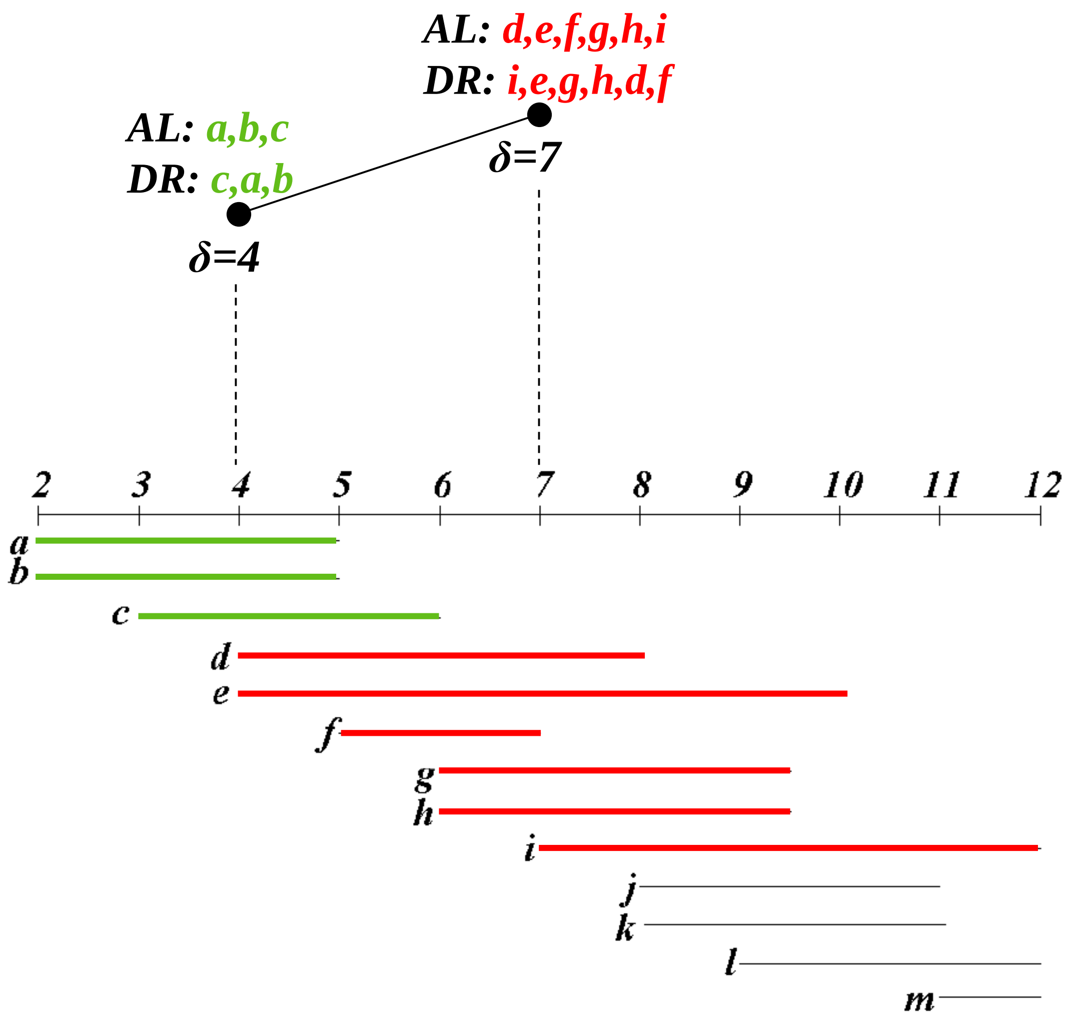
Intervals:



Interval Trees

Interval tree:

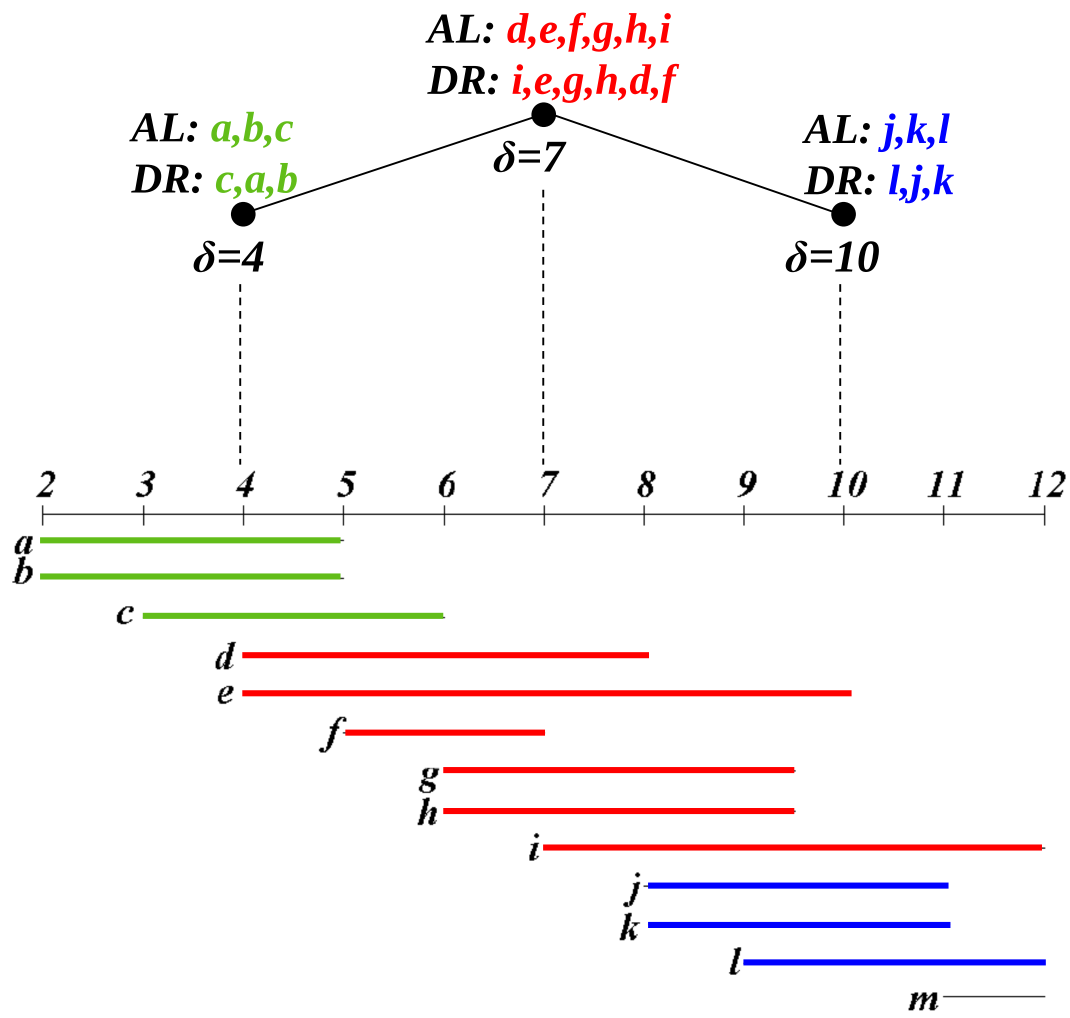
Intervals:



Interval Trees

Interval tree:

Intervals:

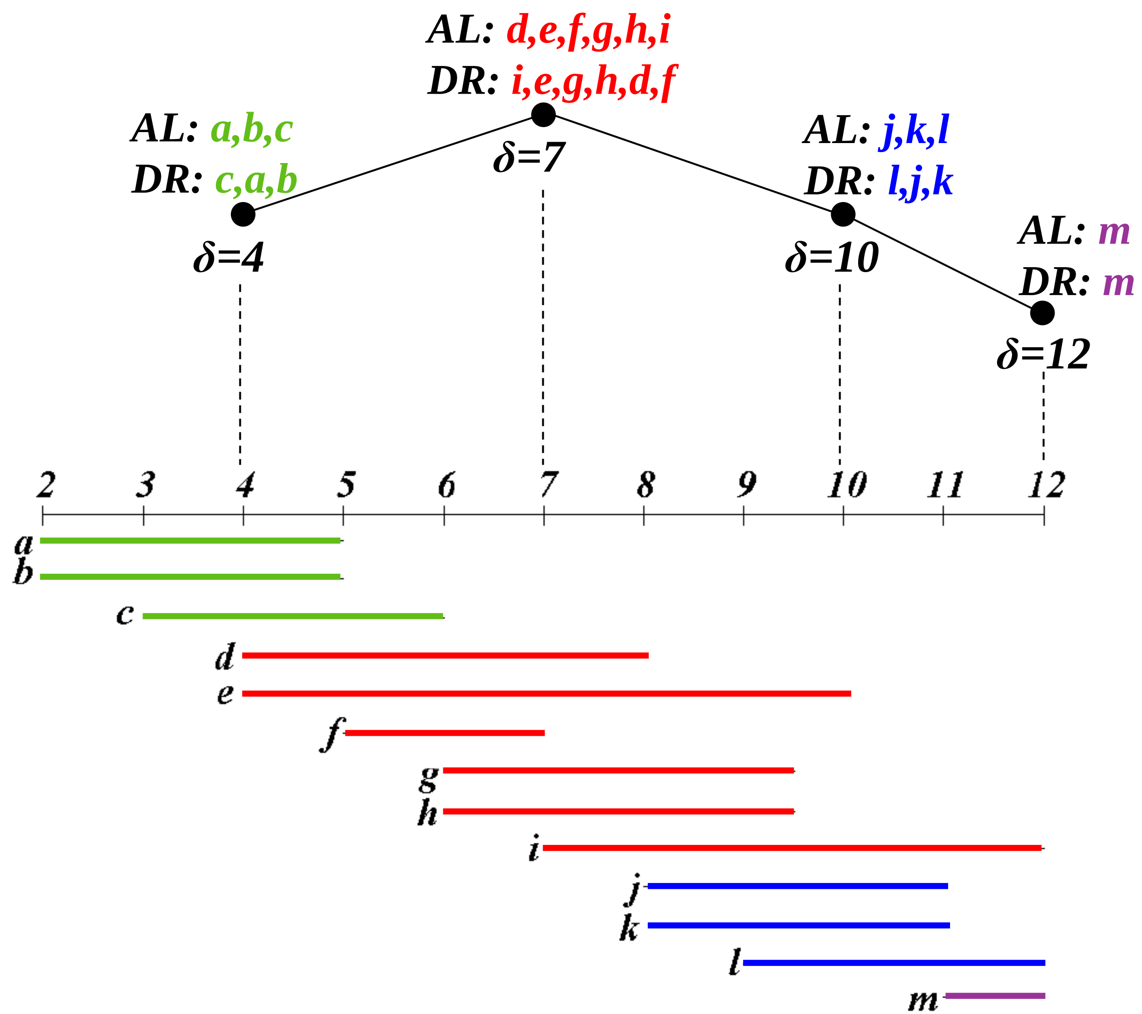


Interval Trees

Washington University in St.Louis
School of Engineering & Applied Science

Interval tree:

Intervals:



Interval Trees

- Tree building: time complexity?
  - O(n*Log(n))
    - O(n) at each level of the tree (after pre-sorting all intervals in O(n*Log(n)))
    - Depth of the tree is O(Log(n))

Interval Trees: Summary

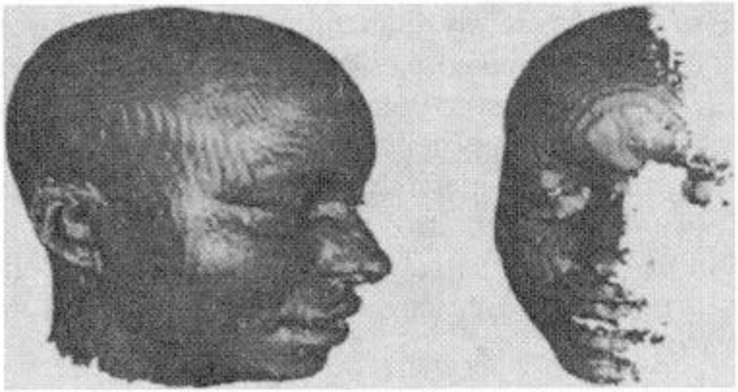
- Preprocessing: building the tree top-down
  - Independent of iso-values
- Contouring: traversing the tree top-down
  - For a specific iso-value
- Both are recursive algorithms

Further Readings

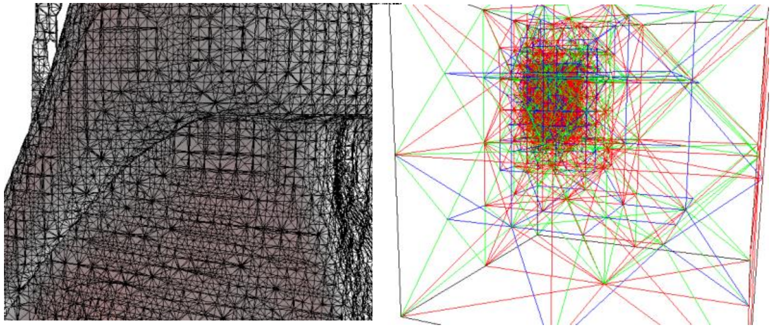
- Quadtree/Octrees:
  - "*Octrees for Faster Isosurface Generation*", Wilhelms and van Gelder (1992)
- Interval trees:
  - "*Dynamic Data Structures for Orthogonal Intersection Queries*", by Edelsbrunner (1980)
  - "*Speeding Up Isosurface Extraction Using Interval Trees*", by Cignoni et al. (1997)

Further Readings

- Other acceleration techniques:
  - “*Fast Iso-contouring for Improved Interactivity*”, by Bajaj et al. (1996)
    - Growing connected component of active cells from pre-computed seed locations
  - “*View Dependent Isosurface Extraction*”, by Livnat and Hansen (1998)
    - Culling invisible mesh parts
  - “*Interactive View-Dependent Rendering Of Large Isosurfaces*”, by Gregorski et al. (2002)
    - Level-of-detail: decreasing mesh resolution based on distance from the viewer



Livnat and Hansen



Gregorski et al.