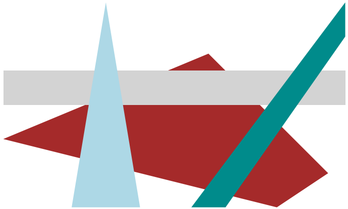
Plane Sweep Algorithm

Aritra Banik*
National Institute of Science Education and Research

Summer School on Graph Theory and Graph Algorithm at NIT Calicut

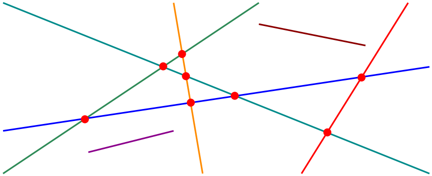
*Slide ideas borrowed from Marc van Kreveld and Subhash Suri

Intersection Detection



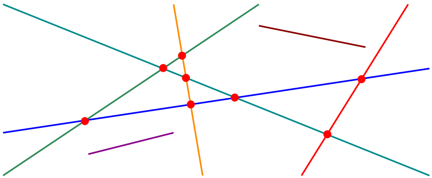
- Determine pairs of intersecting objects?
  - Collision detection in robotics and motion planning.
  - Visibility, occlusion, rendering in graphics.
  - Map overlay in GISs: e.g. road networks on county maps.

Line Segment Intersection



- Let's first look at the easiest version of the problem:
- Given a set of of n line segments in the plane, find all intersection points efficiently
- Naive algorithm?Check all pairs. $O(n^2)$.

Line Segment Intersection



- Let's first look at the easiest version of the problem:
- Given a set of of n line segments in the plane, find all intersection points efficiently
- Naive algorithm?Check all pairs. $O(n^2)$.

Algorithm 1 FindIntersections(S)

Input: A set S of line segments in the plane.
Output: The set of intersection points among the segments in S.

```
1: for each pair of line segments e_i, e_j ∈ S do
2:     if e_i and e_j intersect then
3:         report their intersection point
4:     end if
5: end for
```

- Question: Why can we say that this algorithm is optimal?

Algorithm 2 FindIntersections(S)

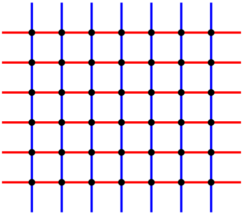
Input: A set S of line segments in the plane.

Output: The set of intersection points among the segments in S.

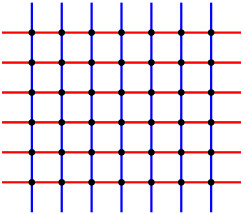
```
for each pair of line segments e_i, e_j ∈ S do
    if e_i and e_j intersect then
        report their intersection point
    end if
end for
```

- Question: Why can we say that this algorithm is optimal?

Line Segment Intersection

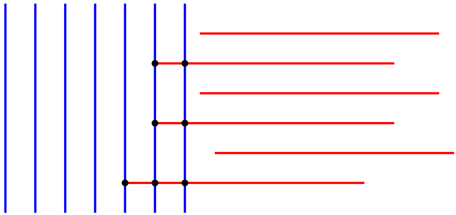


- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k) logn)$ solution. Also introduce a new technique : PLANE SWEEP.



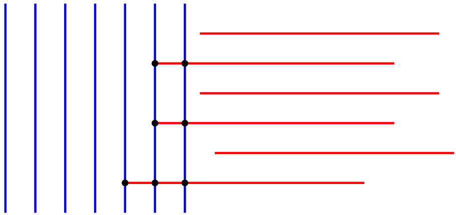
- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k)logn)$ solution. Also introduce a new technique : PLANE SWEEP.

Line Segment Intersection

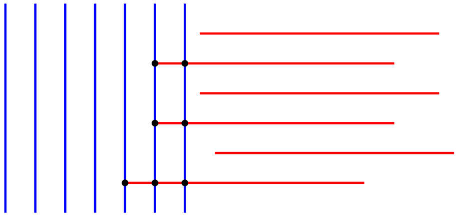


- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k) logn)$ solution. Also introduce a new technique : PLANE SWEEP.

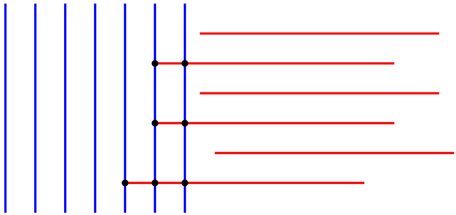
Line Segment Intersection



- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k)logn)$ solution. Also introduce a new technique : PLANE SWEEP.

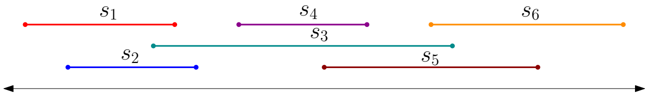


- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k) logn)$ solution. Also introduce a new technique : PLANE SWEEP.



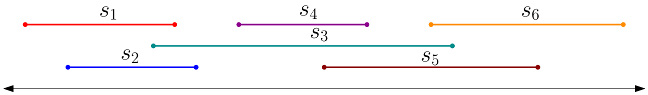
- The asymptotic running time of an algorithm is always input-sensitive (depends on n)
- We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm
- If there are k intersections, then ideal will be $O(n \log n + k)$ time.
- We will describe a $O((n+k)logn)$ solution. Also introduce a new technique : PLANE SWEEP.

An Easier Problem:



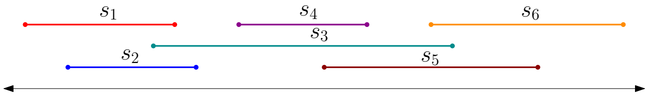
- Given a set of intervals on the real line, find all overlapping pairs.
- Imagine a horizontal line passing over the plane from left to right, solving the problem as it moves
- The sweep line stops and the algorithm computes at certain positions : EVENTS/ EVENT POINTS
- The algorithm stores the relevant situation at the current position of the sweep line : STATUS
- The algorithm knows everything it needs to know before the sweep line, and found all intersection pairs.

An Easier Problem:



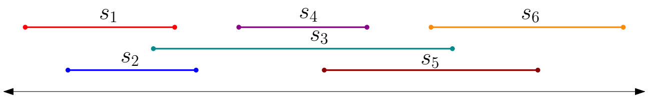
- Given a set of intervals on the real line, find all overlapping pairs.
- Imagine a horizontal line passing over the plane from left to right, solving the problem as it moves
- The sweep line stops and the algorithm computes at certain positions : EVENTS/ EVENT POINTS
- The algorithm stores the relevant situation at the current position of the sweep line : STATUS
- The algorithm knows everything it needs to know before the sweep line, and found all intersection pairs.

An Easier Problem:



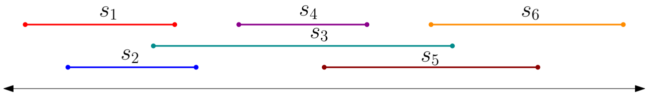
- Given a set of intervals on the real line, find all overlapping pairs.
- Imagine a horizontal line passing over the plane from left to right, solving the problem as it moves
- The sweep line stops and the algorithm computes at certain positions : EVENTS/ EVENT POINTS
- The algorithm stores the relevant situation at the current position of the sweep line : STATUS
- The algorithm knows everything it needs to know before the sweep line, and found all intersection pairs.

An Easier Problem:



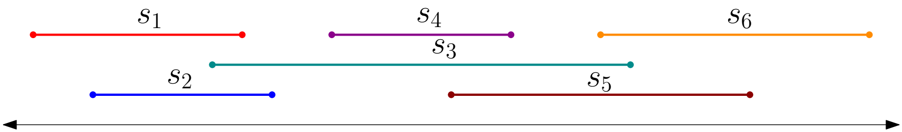
- Given a set of intervals on the real line, find all overlapping pairs.
- Imagine a horizontal line passing over the plane from left to right, solving the problem as it moves
- The sweep line stops and the algorithm computes at certain positions : EVENTS/ EVENT POINTS
- The algorithm stores the relevant situation at the current position of the sweep line : STATUS
- The algorithm knows everything it needs to know before the sweep line, and found all intersection pairs.

An Easier Problem:



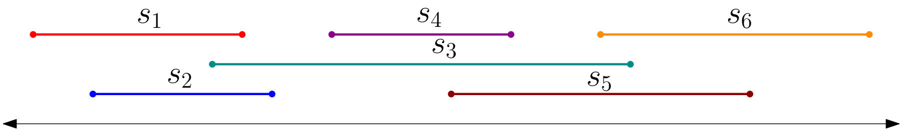
- Given a set of intervals on the real line, find all overlapping pairs.
- Imagine a horizontal line passing over the plane from left to right, solving the problem as it moves
- The sweep line stops and the algorithm computes at certain positions : EVENTS/ EVENT POINTS
- The algorithm stores the relevant situation at the current position of the sweep line : STATUS
- The algorithm knows everything it needs to know before the sweep line, and found all intersection pairs.

An Easier Problem:



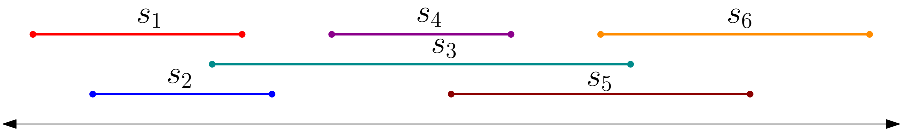
- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
  - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
  - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
  - Insert/Delete
  - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$

An Easier Problem:

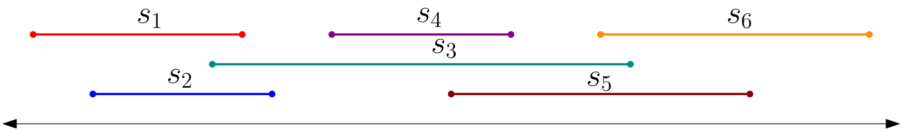


- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
  - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
  - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
  - Insert/Delete
  - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$

An Easier Problem:

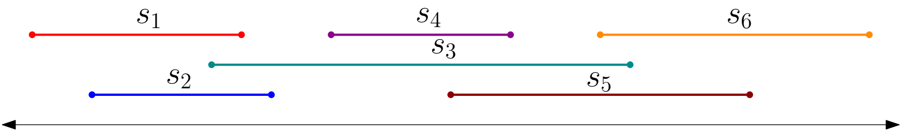


- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
    - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
    - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
    - Insert/Delete
    - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$



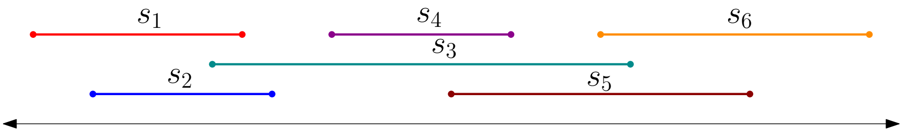
- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
  - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
  - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
  - Insert/Delete
  - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$

An Easier Problem:



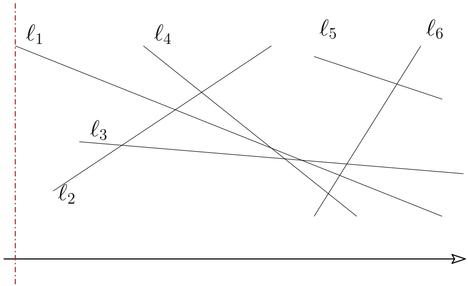
- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
  - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
  - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
  - Insert/Delete
  - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$

An Easier Problem:



- Sort the endpoints and handle them from left to right; maintain currently intersected intervals in a balanced search tree $\mathcal{T}$
  - Left endpoint of s_i : for each s_j in $\mathcal{T}$, report the pair (s_i, s_j). Then insert s_i in $\mathcal{T}$
  - Right endpoint of s_i : delete s_i from $\mathcal{T}$
- There will be $2n$ many event points.
- At each event point we do two operations
  - Insert/Delete
  - Report intersection
- Total time = Total insert delete time + Total time to report intersection
- $2n * \log n + k$

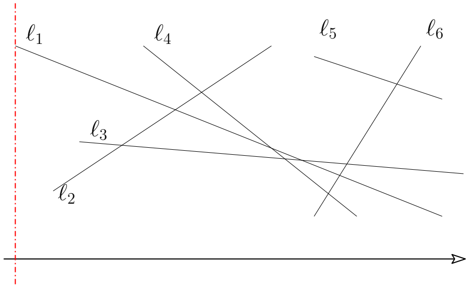
Back to 2D Problem



Imagine a horizontal line passing over the plane from top to bottom, solving the problem as it moves

- Question: What are the event points?
- Maintain vertical order of segments intersecting the sweep line;

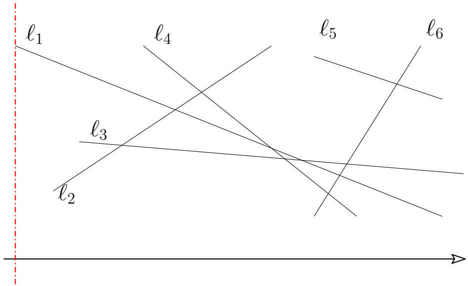
Back to 2D Problem



Imagine a horizontal line passing over the plane from top to bottom, solving the problem as it moves

- Question: What are the event points?
- Maintain vertical order of segments intersecting the sweep line;

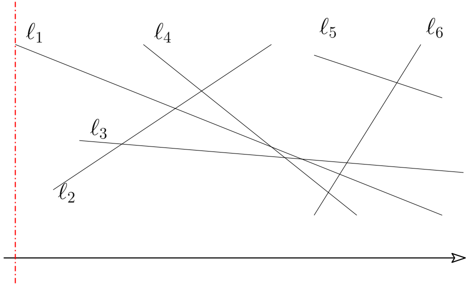
Back to 2D Problem



Imagine a horizontal line passing over the plane from top to bottom, solving the problem as it moves

- Question: What are the event points?
- Maintain vertical order of segments intersecting the sweep line;

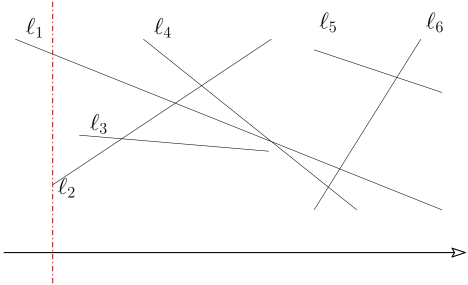
Back to 2D Problem



- Insert ℓ_1, add the end point of ℓ_1 to the event queue
- Insert ℓ_2, Current order ℓ_1, ℓ_2
- Insert ℓ_3, Current order ℓ_1, ℓ_3, ℓ_2,
  - Check whether ℓ_3 intersects with ℓ_1 or ℓ_2.
  - Insert intersection point of ℓ_2 and ℓ_3 into the event queue.
- Current order ℓ_1, ℓ_2, ℓ_3, insert intersection point of ℓ_3, ℓ_2 to the event queue.

. . . and so on . . .

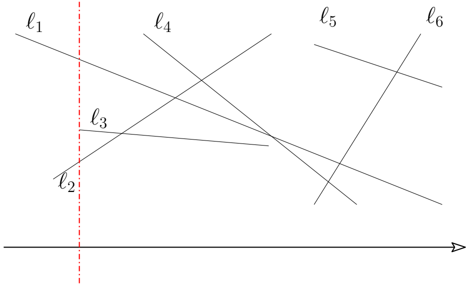
Back to 2D Problem



- Insert ℓ_1, add the end point of ℓ_1 to the event queue
- Insert ℓ_2, Current order ℓ_1, ℓ_2
- Insert ℓ_3, Current order ℓ_1, ℓ_3, ℓ_2,
  - Check whether ℓ_3 intersects with ℓ_1 or ℓ_2.
  - Insert intersection point of ℓ_2 and ℓ_3 into the event queue.
- Current order ℓ_1, ℓ_2, ℓ_3, insert intersection point of ℓ_3, ℓ_2 to the event queue.

. . . and so on . . .

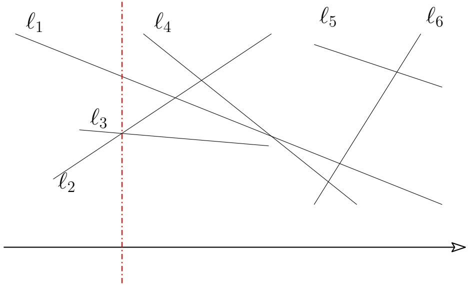
Back to 2D Problem



- Insert ℓ_1, add the end point of ℓ_1 to the event queue
- Insert ℓ_2, Current order ℓ_1, ℓ_2
- Insert ℓ_3, Current order ℓ_1, ℓ_3, ℓ_2,
  - Check whether ℓ_3 intersects with ℓ_1 or ℓ_2.
  - Insert intersection point of ℓ_2 and ℓ_3 into the event queue.
- Current order ℓ_1, ℓ_2, ℓ_3, insert intersection point of ℓ_3, ℓ_2 to the event queue.

. . . and so on . . .

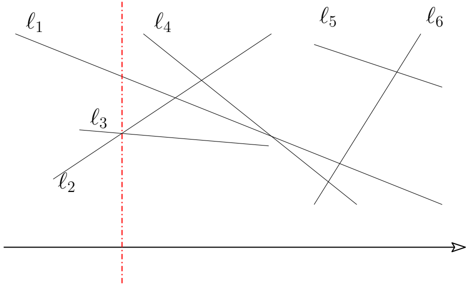
Back to 2D Problem



- Insert ℓ_1, add the end point of ℓ_1 to the event queue
- Insert ℓ_2, Current order ℓ_1, ℓ_2
- Insert ℓ_3, Current order ℓ_1, ℓ_3, ℓ_2,
  - Check whether ℓ_3 intersects with ℓ_1 or ℓ_2.
  - Insert intersection point of ℓ_2 and ℓ_3 into the event queue.
- Current order ℓ_1, ℓ_2, ℓ_3, insert intersection point of ℓ_3, ℓ_2 to the event queue.

. . . and so on . . .

Back to 2D Problem



- Insert ℓ_1, add the end point of ℓ_1 to the event queue
- Insert ℓ_2, Current order ℓ_1, ℓ_2
- Insert ℓ_3, Current order ℓ_1, ℓ_3, ℓ_2,
  - Check whether ℓ_3 intersects with ℓ_1 or ℓ_2.
  - Insert intersection point of ℓ_2 and ℓ_3 into the event queue.
- Current order ℓ_1, ℓ_2, ℓ_3, insert intersection point of ℓ_3, ℓ_2 to the event queue.

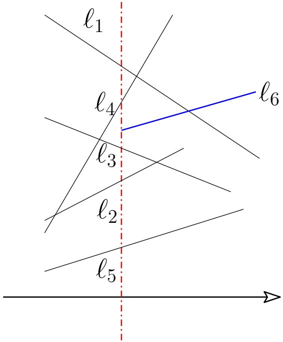
. . . and so on . . .

Events

When do the events happen? When the sweep line is at

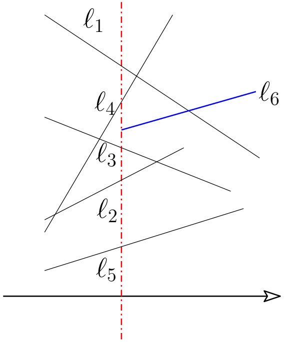
- a left endpoint of a line segment
- a right endpoint of a line segment
- an intersection point of a line segment

A left endpoint of a line segment



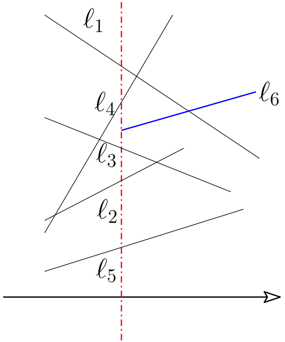
- We use a balanced binary search tree with the line segments in the leaves as the status structure.
- Search and insert.
- Should we find out intersection of ℓ_6
- At the time of insert ℓ_6 is adjacent to ℓ_4 and l_3.
- Check whether ℓ_6 intersects with ℓ_4 and l_3 or not, if intersects, insert the intersection points in the event queue.

A left endpoint of a line segment



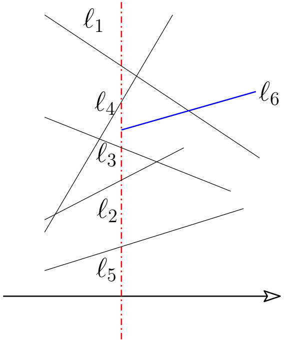
- We use a balanced binary search tree with the line segments in the leaves as the status structure.
- Search and insert.
- Should we find out intersection of ℓ_6
- At the time of insert ℓ_6 is adjacent to ℓ_4 and l_3.
- Check whether ℓ_6 intersects with ℓ_4 and l_3 or not, if intersects, insert the intersection points in the event queue.

A left endpoint of a line segment



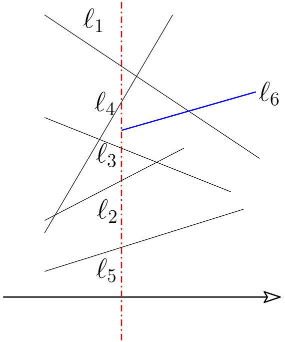
- We use a balanced binary search tree with the line segments in the leaves as the status structure.
- Search and insert.
- Should we find out intersection of ℓ_6
- At the time of insert ℓ_6 is adjacent to ℓ_4 and l_3.
- Check whether ℓ_6 intersects with ℓ_4 and l_3 or not, if intersects, insert the intersection points in the event queue.

A left endpoint of a line segment



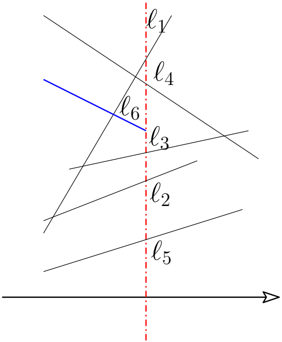
- We use a balanced binary search tree with the line segments in the leaves as the status structure.
- Search and insert.
- Should we find out intersection of ℓ_6
- At the time of insert ℓ_6 is adjacent to ℓ_4 and l_3.
- Check whether ℓ_6 intersects with ℓ_4 and l_3 or not, if intersects, insert the intersection points in the event queue.

A left endpoint of a line segment



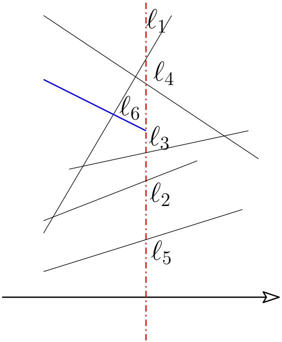
- We use a balanced binary search tree with the line segments in the leaves as the status structure.
- Search and insert.
- Should we find out intersection of ℓ_6
- At the time of insert ℓ_6 is adjacent to ℓ_4 and l_3.
- Check whether ℓ_6 intersects with ℓ_4 and l_3 or not, if intersects, insert the intersection points in the event queue.

A right endpoint of a line segment



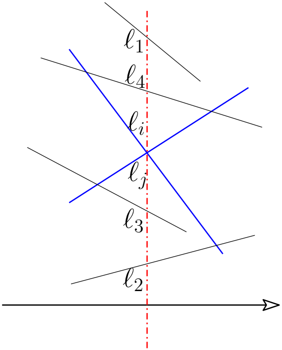
- Sweep line reaches right endpoint of a line segment: delete the line segment
- After deletion of ℓ_6, ℓ_3 and ℓ_4 becomes adjacent.
- If ℓ_3 and ℓ_4 intersects insert the intersection point into the event queue.

A right endpoint of a line segment



- Sweep line reaches right endpoint of a line segment: delete the line segment
- After deletion of ℓ_6, ℓ_3 and ℓ_4 becomes adjacent.
- If ℓ_3 and ℓ_4 intersects insert the intersection point into the event queue.

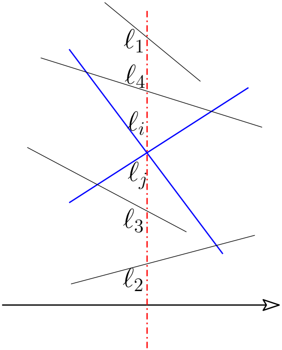
Sweep line reaches an intersection point



Sweep line reaches an intersection point of ℓ_i and ℓ_j

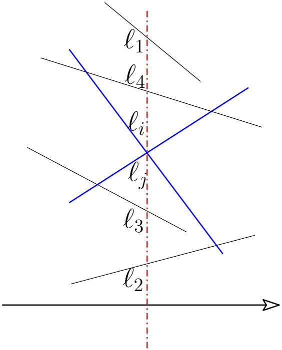
- Exchange ℓ_i and ℓ_j in the order list.
- If ℓ_i and its new left neighbor intersects, then insert this intersection point in the event queue
- If ℓ_j and its new left neighbor intersects, then insert this intersection point in the event queue.
- Report the intersection point.

Sweep line reaches an intersection point



Sweep line reaches an intersection point of ℓ_i and ℓ_j

- Exchange ℓ_i and ℓ_j in the order list.
- If ℓ_i and its new left neighbor intersects, then insert this intersection point in the event queue
- If ℓ_j and its new left neighbor intersects, then insert this intersection point in the event queue.
- Report the intersection point.



Sweep line reaches an intersection point of ℓ_i and ℓ_j

- Exchange ℓ_i and ℓ_j in the order list.
- If ℓ_i and its new left neighbor intersects, then insert this intersection point in the event queue
- If ℓ_j and its new left neighbor intersects, then insert this intersection point in the event queue.
- Report the intersection point.

Finding events

- Before the sweep algorithm starts, we know all upper endpoint events and all lower endpoint events
- But: How do we know intersection point events??? (those we were trying to find . . .)
- Observe: Two line segments can only intersect if they are horizontal neighbors

Runtime

- At each event constant many updates
- Since both the event queue and T are balanced binary search trees, handling an event takes only $O(\log n)$ time.
- Total no of events are $O(n + k)$.
- The algorithm takes $O(n \log n + k \log n)$ time If $k = O(n)$, then this is $O(n \log n)$
- Note that if k is really large, the brute force $O(n^2)$ time algorithm is more efficient

Runtime

- At each event constant many updates
- Since both the event queue and T are balanced binary search trees, handling an event takes only $O(\log n)$ time.
- Total no of events are $O(n + k)$.
- The algorithm takes $O(n \log n + k \log n)$ time If $k = O(n)$, then this is $O(n \log n)$
- Note that if k is really large, the brute force $O(n^2)$ time algorithm is more efficient

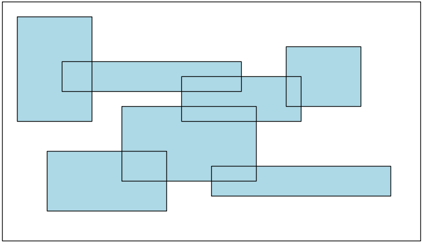
Runtime

- At each event constant many updates
- Since both the event queue and T are balanced binary search trees, handling an event takes only $O(\log n)$ time.
- Total no of events are $O(n+k)$.
- The algorithm takes $O(n \log n + k \log n)$ time If $k = O(n)$, then this is $O(n \log n)$
- Note that if k is really large, the brute force $O(n^2)$ time algorithm is more efficient

Runtime

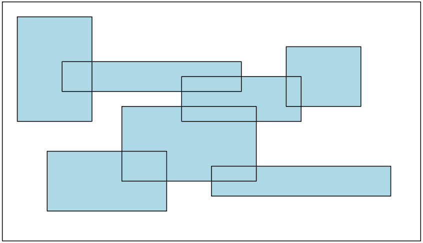
- At each event constant many updates
- Since both the event queue and T are balanced binary search trees, handling an event takes only $O(\log n)$ time.
- Total no of events are $O(n+k)$.
- The algorithm takes $O(n \log n + k \log n)$ time If $k = O(n)$, then this is $O(n \log n)$
- Note that if k is really large, the brute force $O(n^2)$ time algorithm is more efficient

Area of Union of rectangles

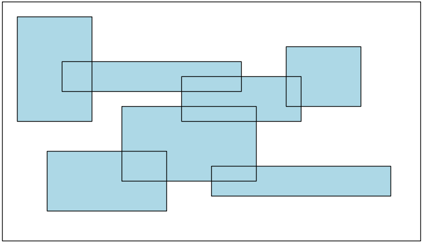


- Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to find the area covered by all the objects.
- PLANE SWEEP: Keep track of the area that being sweeped.
- EVENT POINTS: When is the intersection of the the rectangels with the sweep line changes?Left and right end point of the rectangeles.
- What is the area between any two event points?
- $\delta x \times y$ where y is the length of the intersection of the the rectangels with the sweep line.

Area of Union of rectangles

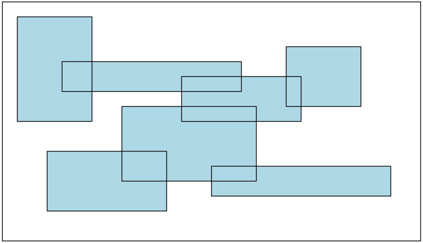


- Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to find the area covered by all the objects.
- PLANE SWEEP: Keep track of the area that being sweeped.
- EVENT POINTS: When is the intersection of the the rectangels with the sweep line changes?Left and right end point of the rectangeles.
- What is the area between any two event points?
- $\delta x \times y$ where y is the length of the intersection of the the rectangels with the sweep line.



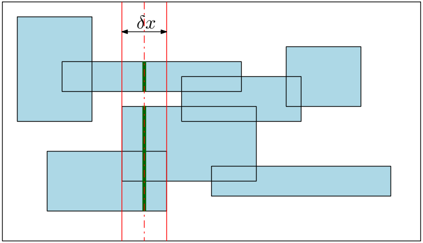
- Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to find the area covered by all the objects.
- PLANE SWEEP: Keep track of the area that being sweeped.
- EVENT POINTS: When is the intersection of the the rectangels with the sweep line changes? Left and right end point of the rectangeles.
- What is the area between any two event points?
- $\delta x \times y$ where y is the length of the intersection of the the rectangels with the sweep line.

Area of Union of rectangles

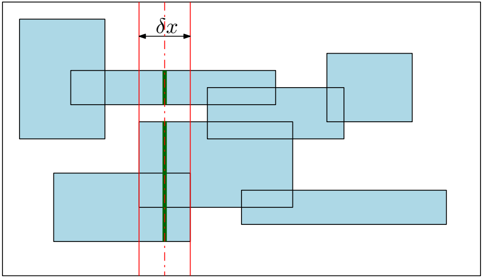


- Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to find the area covered by all the objects.
- PLANE SWEEP: Keep track of the area that being sweeped.
- EVENT POINTS: When is the intersection of the the rectangels with the sweep line changes?Left and right end point of the rectangeles.
- What is the area between any two event points?
- $\delta x \times y$ where y is the length of the intersection of the the rectangels with the sweep line.

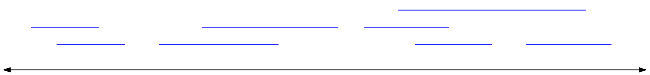
Area of Union of rectangles



- Given a layout in which objects are orthogonal polygons with sides parallel to the axises. The task is to find the area covered by all the objects.
- PLANE SWEEP: Keep track of the area that being sweeped.
- EVENT POINTS: When is the intersection of the the rectangels with the sweep line changes?Left and right end point of the rectangeles.
- What is the area between any two event points?
- $\delta x \times y$ where y is the length of the intersection of the the rectangels with the sweep line.

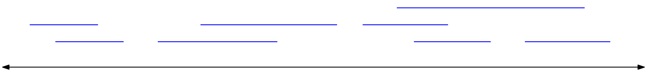


- Intersection of the the rectangels with the sweep line is a set of intervals.
- Thus the problem at hand becomes to maintain the intercepts. The y can change only at
  - The beginning of a rectangle.
  - The end of the rectangle.

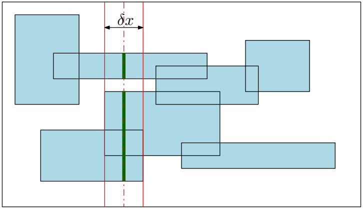


- Naïve Method:
- At each event point find out y by a sweepline method.
  - EVENT POINTS: Left and right end point of an interval.
  - STATUS: Balanced binary search tree to store intervals.
  - At each event point if tree is not empty sum+= distance between current and last event point.
- Complexity of sum of intervals $O(n \log n)$
- Complexity of area of union of rectangles $O(n^2 \log n)$
- Can we do better??How to maintain sum of the union of the intervals with respect to insertion and deletion

Sum of the union of the intervals

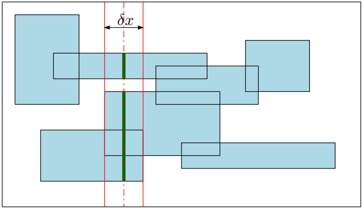


- Naïve Method:
- At each event point find out y by a sweepline method.
  - EVENT POINTS: Left and right end point of an interval.
  - STATUS: Balanced binary search tree to store intervals.
  - At each event point if tree is not empty sum+= distance between current and last event point.
- Complexity of sum of intervals $O(n \log n)$
- Complexity of area of union of rectangles $O(n^2 \log n)$
- Can we do better??How to maintain sum of the union of the intervals with respect to insertion and deletion



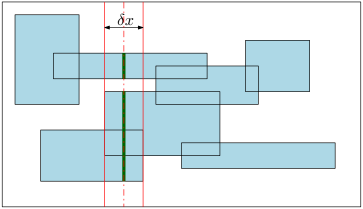
- Naïve Method:
- At each event point find out y by a sweepline method.
    - EVENT POINTS: Left and right end point of an interval.
    - STATUS: Balanced binary search tree to store intervals.
    - At each event point if tree is not empty sum+= distance between current and last event point.
- Complexity of sum of intervals $O(n \log n)$
- Complexity of area of union of rectangles $O(n^2 \log n)$
- Can we do better??How to maintain sum of the union of the intervals with respect to insertion and deletion

Sum of the union of the intervals

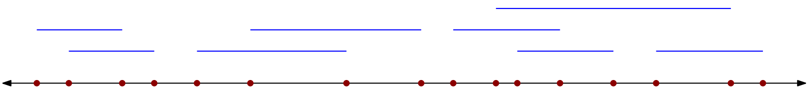


- Naïve Method:
- At each event point find out y by a sweepline method.
  - EVENT POINTS: Left and right end point of an interval.
  - STATUS: Balanced binary search tree to store intervals.
  - At each event point if tree is not empty sum+= distance between current and last event point.
- Complexity of sum of intervals $O(n \log n)$
- Complexity of area of union of rectangles $O(n^2 \log n)$
- Can we do better??How to maintain sum of the union of the intervals with respect to insertion and deletion

Sum of the union of the intervals

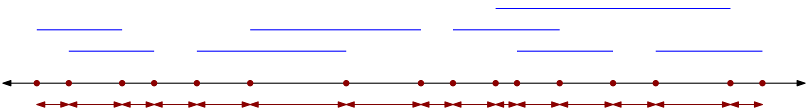


- Naïve Method:
- At each event point find out y by a sweepline method.
  - EVENT POINTS: Left and right end point of an interval.
  - STATUS: Balanced binary search tree to store intervals.
  - At each event point if tree is not empty sum+= distance between current and last event point.
- Complexity of sum of intervals $O(n \log n)$
- Complexity of area of union of rectangles $O(n^2 \log n)$
- Can we do better??How to maintain sum of the union of the intervals with respect to insertion and deletion

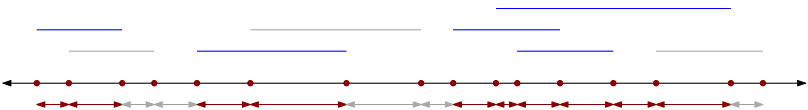


- Sort the end points of the intervals.
- This will create a set of elementary intervals.
- Depending on which intervals are ACTIVE , a set of elementary intervals will be ACTIVE .

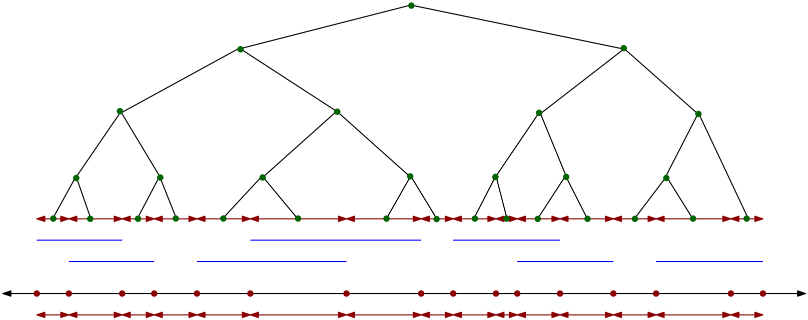
Sum of the union of the intervals



- Sort the end points of the intervals.
- This will create a set of elementary intervals.
- Depending on which intervals are ACTIVE , a set of elementary intervals will be ACTIVE .

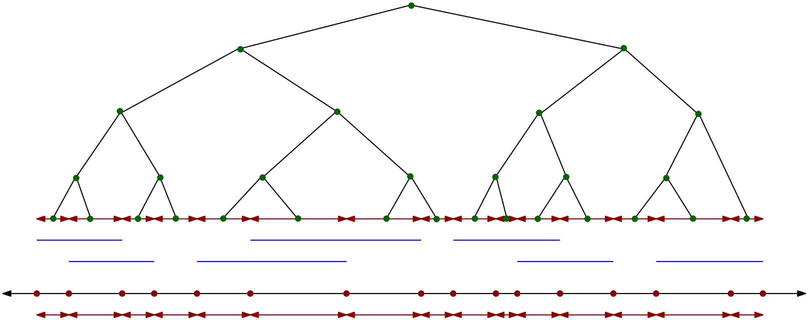


- Sort the end points of the intervals.
- This will create a set of elementary intervals.
- Depending on which intervals are ACTIVE , a set of elementary intervals will be ACTIVE .

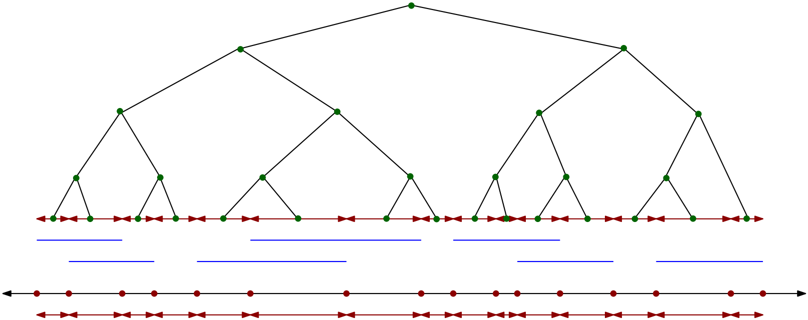


- We maintain a special data structure called the INTERVAL-TREE
- It is a balanced binary tree $\mathcal{T}$ of the ELEMENTORY INTERVALS.
- Each node represents an interval.

Interval-Tree

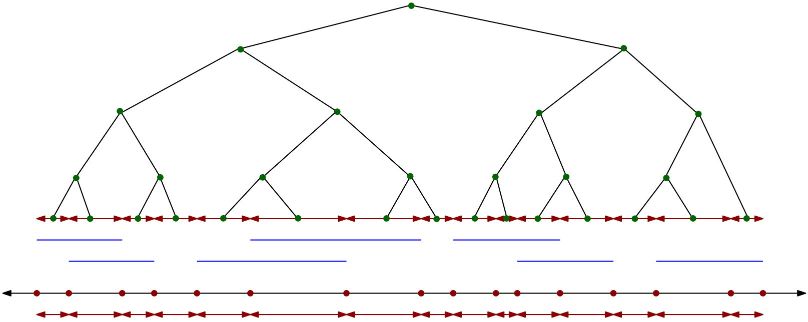


- We maintain a special data structure called the INTERVAL-TREE
- It is a balanced binary tree $\mathcal{T}$ of the ELEMENTORY INTERVALS.
- Each node represents an interval.



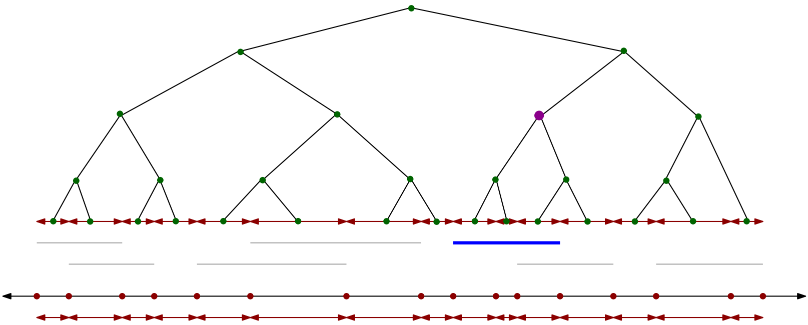
- We maintain a special data structure called the INTERVAL-TREE
- It is a balanced binary tree $\mathcal{T}$ of the ELEMENTORY INTERVALS.
- Each node represents an interval.

Interval-Tree

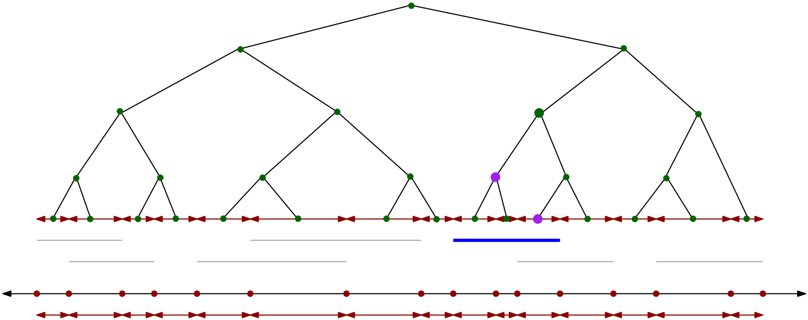


- How an interval I is stored in the tree?
- Start with the root and proceed.

Interval-Tree

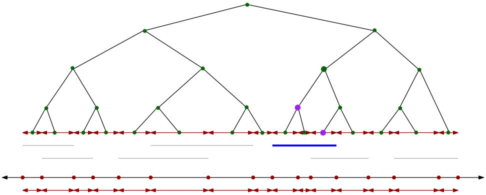


- How an interval I is stored in the tree?
- Start with the root and proceed.



- How an interval I is stored in the tree?
- Start with the root and proceed.

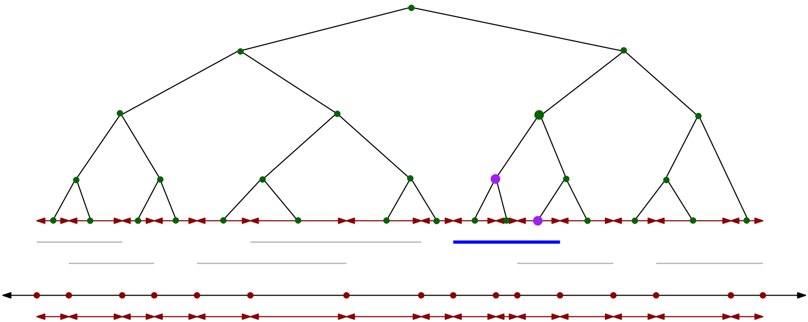
Interval-Tree



Algorithm 3 ReportInterval($\mathcal{T}, v, I$)

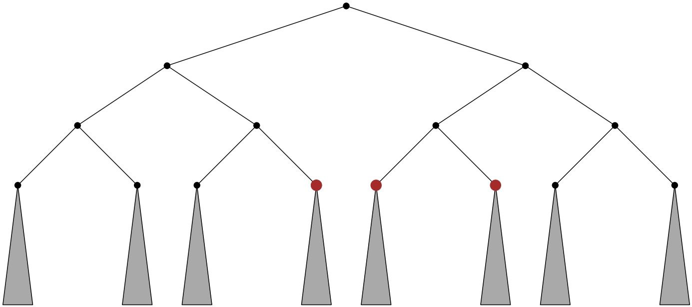
```
if v ⊆ I then
    Report v
    return
end if
if I ∩ lc(v) ≠ emptyset then
    ReportInterval(T, lc(v), I ∩ lc(v))
end if
if I ∩ rc(v) ≠ emptyset then
    ReportInterval(T, rc(v), I ∩ rc(v))
end if
```

Interval-Tree



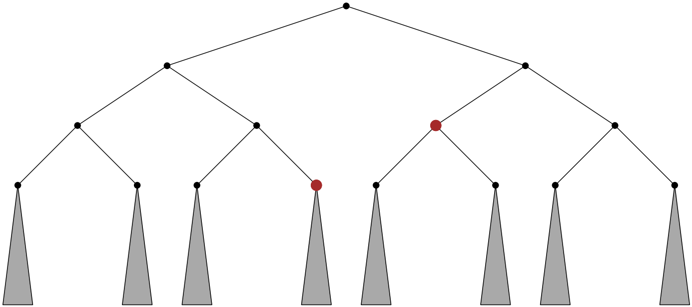
- Claim: Each interval is stored in $O(\log n)$ nodes.
- At each level there can be at most two nodes representing the interval.
- All of them have to be consecutive.

Interval-Tree



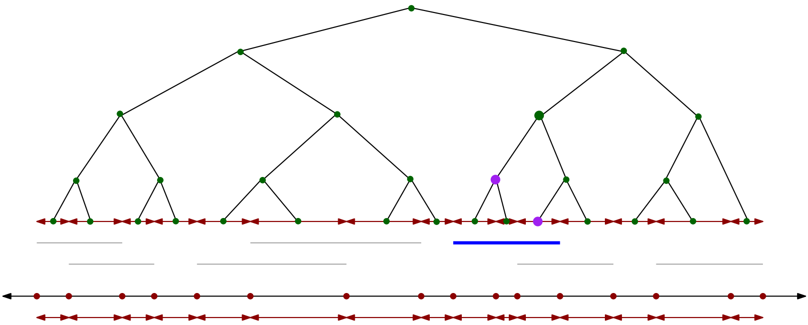
- Claim: Each interval is stored in $O(\log n)$ nodes.
- At each level there can be at most two nodes representing the interval.
- All of them have to be consecutive.

Interval-Tree

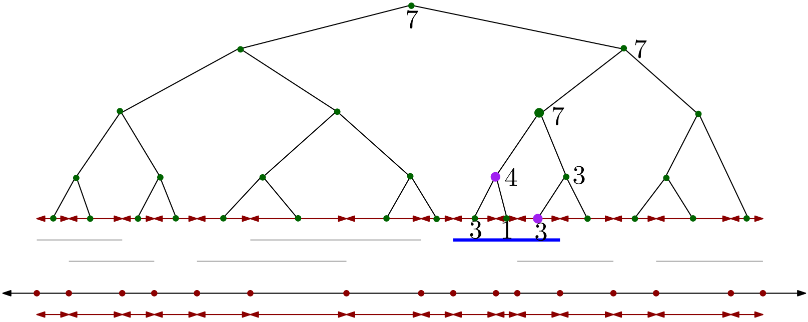


- Claim: Each interval is stored in $O(\log n)$ nodes.
- At each level there can be at most two nodes representing the interval.
- All of them have to be consecutive.

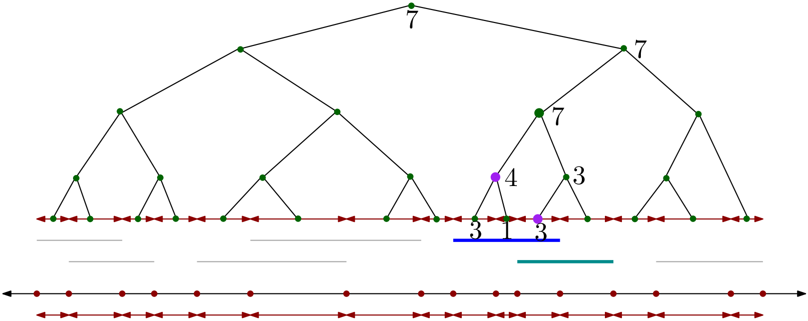
Interval-Tree



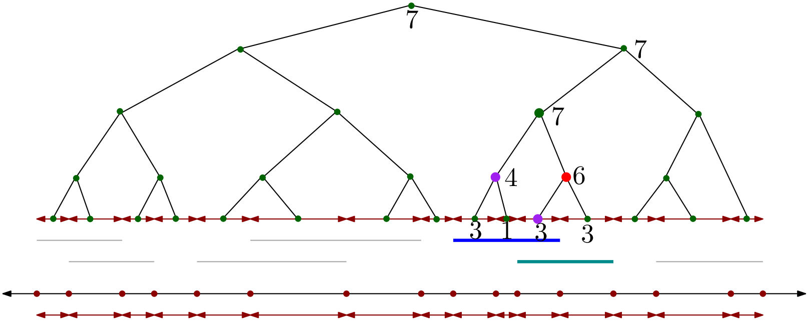
- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.



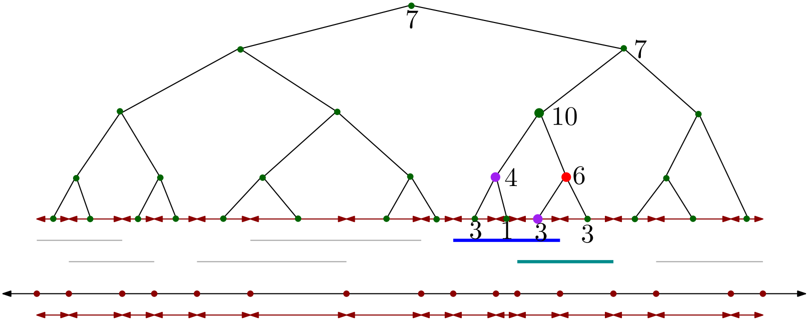
- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.



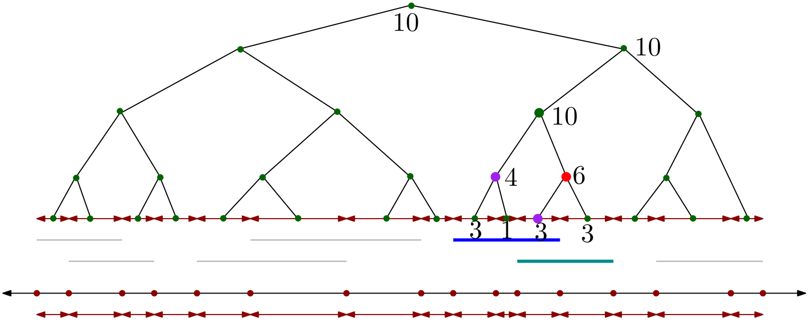
- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.



- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.

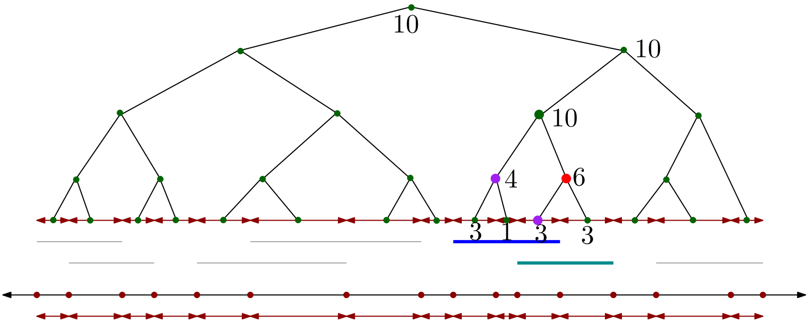


- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.

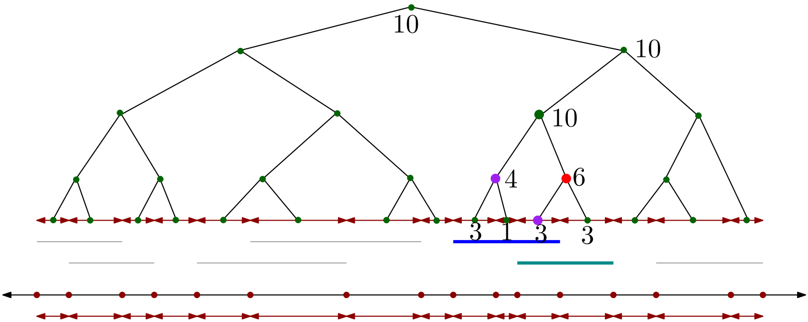


- Each interval can be inserted and deleted in $O(\log n)$ time.
- At each node we maintain the length of the active elementary intervals.

Interval-Tree

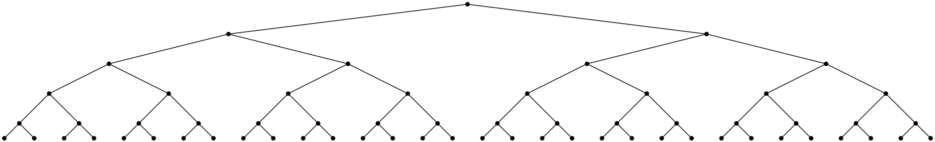


- $O(\log n)$ insert each takes $O(\log n)$ time.
- In time $O(\log^2 n)$ we can perform the updates.
- Can be done in time $O(\log n)$



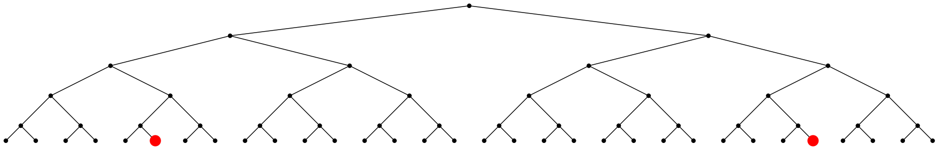
- $O(\log n)$ insert each takes $O(\log n)$ time.
- In time $O(\log^2 n)$ we can perform the updates.
- Can be done in time $O(\log n)$

Interval-Tree



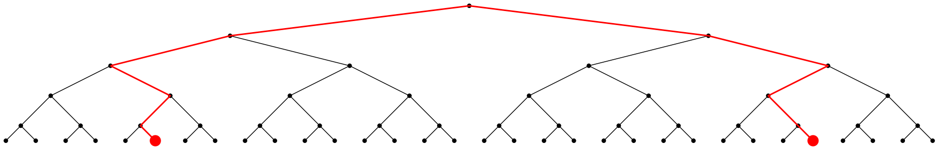
- Can be done in time $O(\log n)$.
- Consider the left most and right most elementary intervals.

Interval-Tree

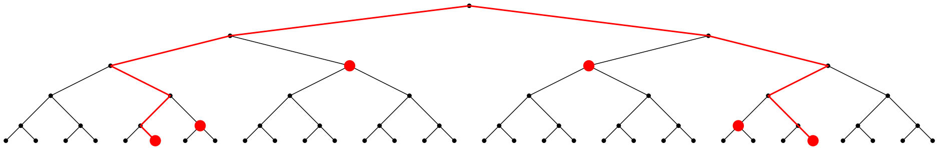


- Can be done in time $O(\log n)$.
- Consider the left most and right most elementary intervals.

Interval-Tree

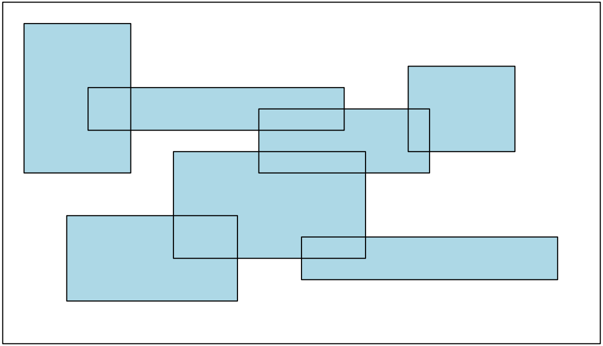


- Can be done in time $O(\log n)$.
- Consider the left most and right most elementary intervals.



- Can be done in time $O(\log n)$.
- Consider the left most and right most elementary intervals.

Sum of the union of the intervals



- In time $O(n \log n)$ we can find out the area of the union of n rectangles.