# Interval search tree

DCTI – IT Companies Seminary

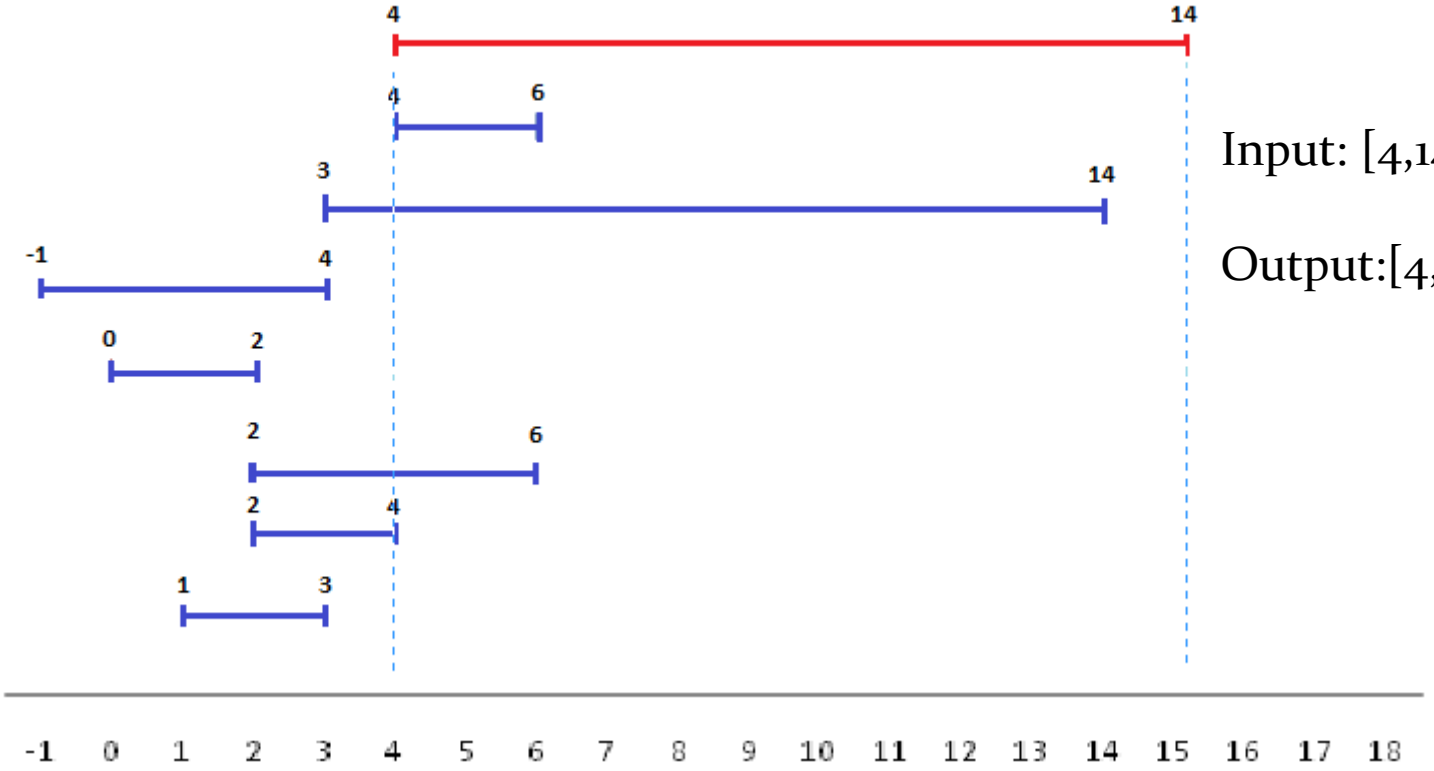
**Coordinator:**
Associate professor Cristian Mihăescu, PhD.

**Author:**
Lucian Iordache,
CE, 2-nd year

April, 2014

# Problem statement

Considering a set of intervals, find all intervals that overlap with any given interval or point.

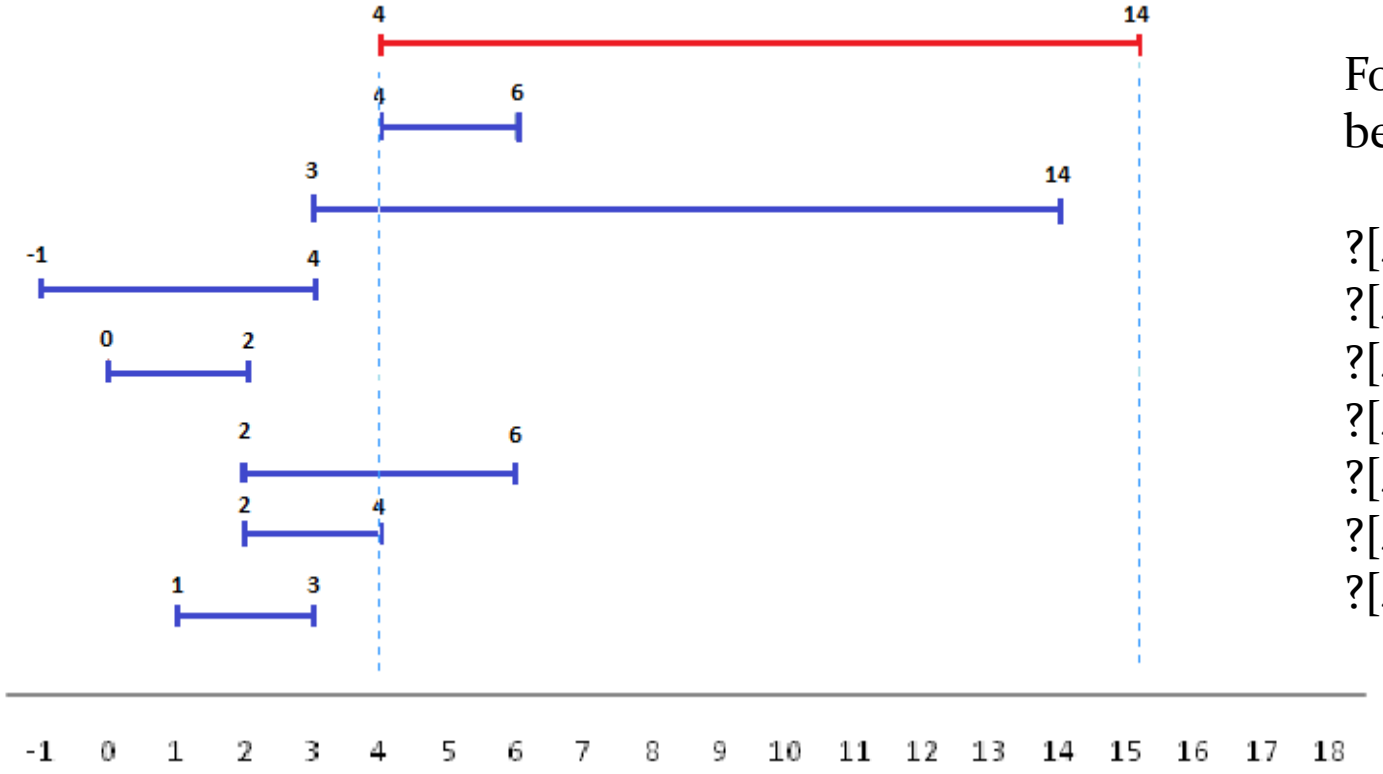



Input: [4,14]

Output:[4,6],[3,14],[2,6],[2,4].

# Trivial solution

The trivial solution is to visit each interval and test whether it intersects the given point or interval.

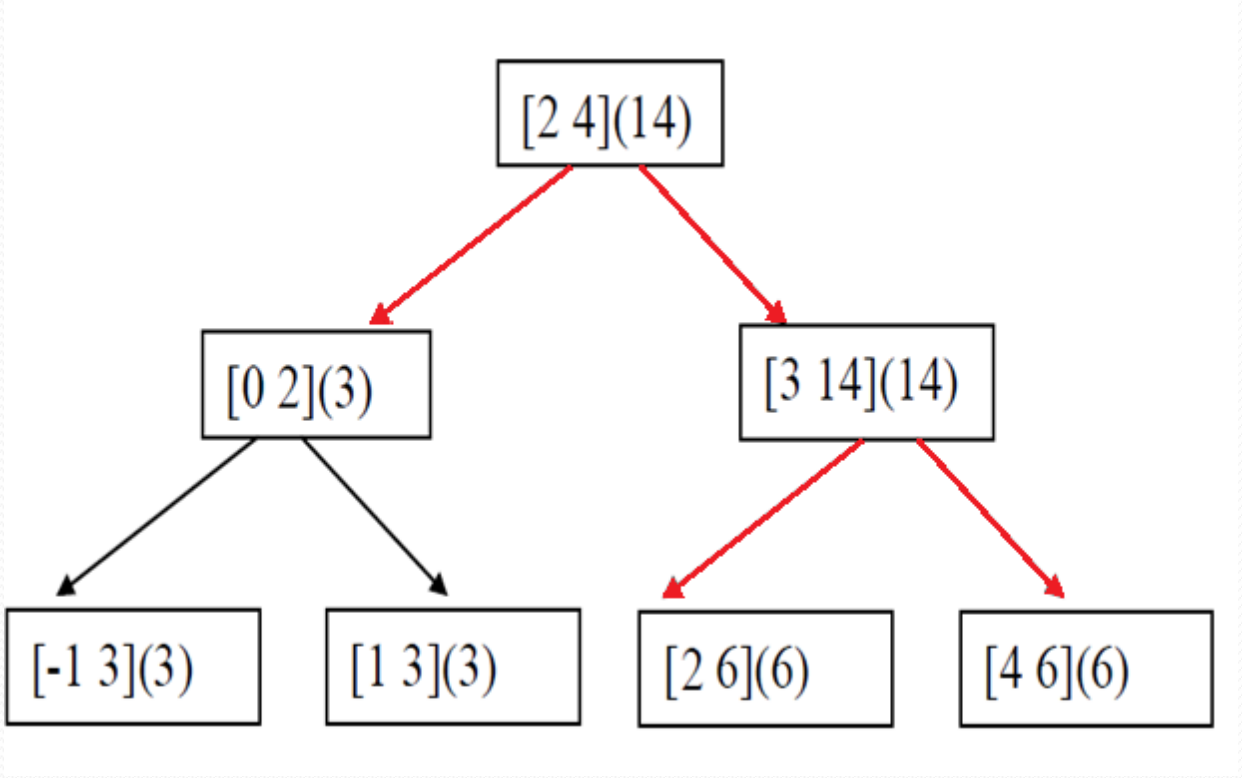



For this example will be 7 comparations:

?[4,14]^[4,6]
?[4,14]^[3,14]
?[4,14]^[-1,4]
?[4,14]^[0,2]
?[4,14]^[2,6]
?[4,14]^[2,4]
?[4,14]^[1,3]

# Interval search tree

Interval search tree implentation use a simple ordered tree,ordered by the 'low' values of the intervals, and an extra annotation is added to every node recording the maximum high value of both its subtrees,so we know that two intervals $A$ and $B$ overlap only when both $A.\text{low} \leq B.\text{high}$ and $A.\text{high} \geq B.\text{low}$.




For this example will be 5 comparations:

?[4,14]∩[2,4]
?[4,14]∩[0,2]
?[4,14]∩[3,14]
?[4,14]∩[2,6]
?[4,14]∩[4,6]

# Structure

```
struct BSTNode
{
BSTNode *left;
BSTNode *right;
Interval nodeInfo;
int maxRight;
};
```

Where:
BSTNode *left-is a pointer to the left node of the tree
BSTNode *right- is a pointer to the right node of the tree
Interval nodeInfo keep the left and the right end of the interval
int maxRight-keep the maximum end of the intervals added

Base function prototipes:

- struct BSTNode *insert(struct BSTNode *root,Interval *newInterval);
- void searchPoint(struct BSTNode *node,int point,FILE *fo);
- void segmentIntersect(struct BSTNode *node,Interval *interval,FILE *fo);

# Running time (computed)

- Trivial solution: Θ(n) time, where *n* is the number of intervals in the collection.
- Interval search tree: Θ(*nlog n*) avarage time, Θ(n) worst case(when all intervals intersects), where *n* is the number of intervals in the collection.

# Demonstration -Master Theorem-

The master theorem concerns recurrence relations of the form:

- $T(n) = aT(n/b) + f(n)$ , where:
  - $a$ is the number of subproblems in the recursion.
  - $n/b$ is the size of each subproblem
  - $n$ is the size of the problem.
  - $f(n)$ is the cost of the work done outside the recursive calls

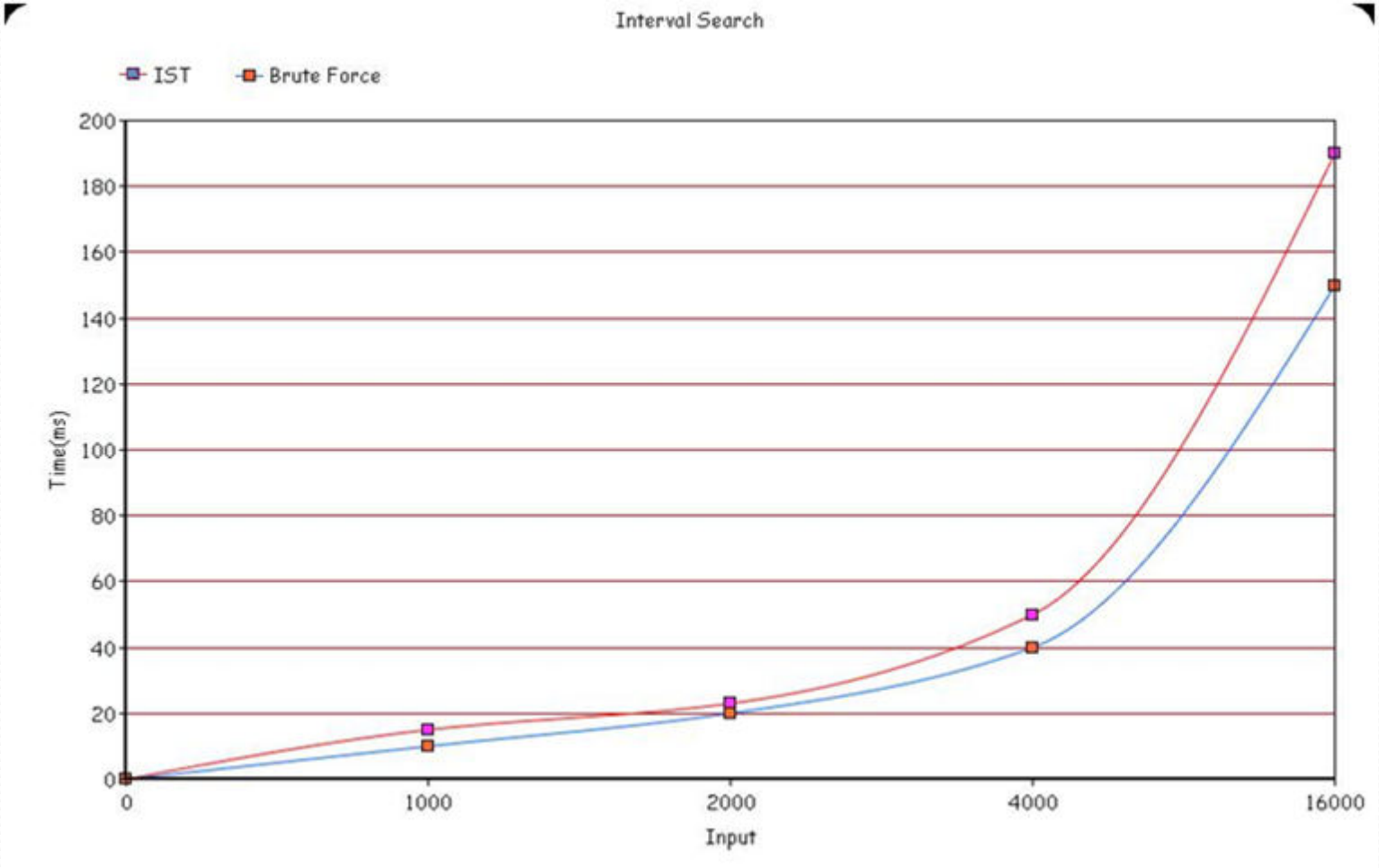
In our case : a = 2 , b = 2 , f(n) = Θ(1),so recurence relation is:

$$T(n) = 2T(\frac{n}{2}) + \theta(1)$$

$$c = \log_b^a = 1$$

$$=> T(n) = \theta(n^{\log_b^a} \log n) = \theta(n \log n)$$

# Running time (measured)



# Thank you!

The implementation of the algorithm can be found at:
https://github.com/mihaescu/ADS/tree/master/BST/Geometric%20Applications/Interval%20Search%20by%20Iordache%20Lucian

# Bibliography


- Wikipedia.org
- Google.ro
- Dumitru Dan Burdescu, Marian Cristian Mihăescu- Algorithms and Data Structures