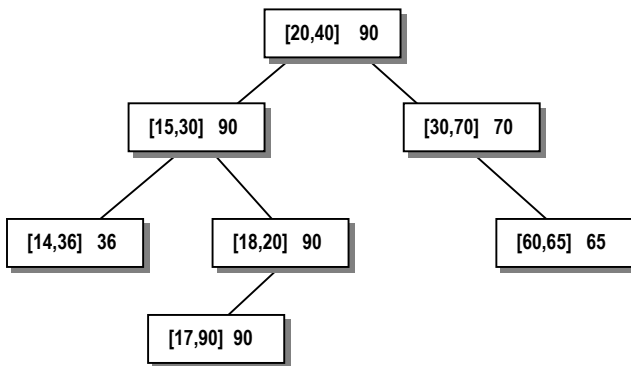


Program 1^x

Extra credit – supplementary to Assignment 1

An *IntervalSet* is a set of closed intervals of the form $I = [I_{lo}, I_{hi}] = \{ x \mid I_{lo} \leq x \leq I_{hi} \}$, where x , I_{lo} and I_{hi} are real numbers. Three operations are supported: *insert*, *remove*, and *findOverlap*. The first two are self-explanatory. In the third, calling *findOverlap*(I) on an *IntervalSet* S returns an interval in S that overlaps the interval I (if one exists).



An *IntervalSet* can be implemented as a simple variant of a binary search tree. Each node stores an interval $[I_{lo}, I_{hi}]$ along with a real number M equal to the least upper bound of all the intervals in the subtree at that node. The value I_{lo} is used as the key for ordering the tree. A call to *findOverlap*(I) works like binary search. You start at the root and work downward; if the node stores an interval that overlaps I you are done; otherwise if it has a left child with $M \geq I_{lo}$, you go down to the left child; otherwise the search proceeds to the right child. It fails when a leaf is reached without finding an overlap.¹

For this assignment you will code the *IntervalSet* ADT (using the implementation above) in C++ in three ways:

- A mean-and-speedy ANSI C procedural implementation. Use comments to make it clear, but feel free to sacrifice code beauty for performance.
- A concrete class *IntervalTree*. Inline as much as you can.
- An interface *IIntervalSet* implemented by a concrete class *CIntervalTree*, working with an abstract factory.

You will benchmark these on the same task: construct an interval set with (for example) $n=10,000$ random intervals, then execute a sequence of (for example) 10,000 random *findOverlap* operations. Try out at least two compilers, with optimizations turned on. Carefully control for what you can: e.g. make the under-the-hood data structures work identically in all three cases; seed the random number generator identically as well. You are encouraged to try out additional variations to pinpoint the precise locations of cost.

Submit three single-file self-contained programs (**p1A.c**, **p1B.cpp**, **p1C.cpp**) for the alternatives, and attach a brief (1 or 2 page) document summarizing your findings. The summary should contain (at least) a 3x2 table of benchmark times, with a few paragraphs of discussion. **Due:** Sunday December 15 by 6pm (by email). This is an individual assignment.

NOTE:

Since this is extra credit, you can choose to do only part of it. Part A is worth 15%, Part B 25%, and Part C 60%.

¹ Reference: Cormen, Leiserson, Rivest (2001), *Introduction to Algorithms*, 2nd Ed., pp. 311-314.